5. Overview of the Hardware Abstraction Layer

NII52003-11.0.0

This chapter introduces the hardware abstraction layer (HAL) for the Nios® II processor. This chapter contains the following sections:

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

The HAL serves as a device driver package for Nios II processor systems, providing a consistent interface to the peripherals in your system. The Nios II software development tools extract system information from your SOPC Information File (**.sopcinfo**). The Nios II Software Build Tools (SBT) generate a custom HAL board support package (BSP) specific to your hardware configuration. Changes in the hardware configuration automatically propagate to the HAL device driver configuration. As a result, changes in the underlying hardware are prevented from creating bugs.

HAL device driver abstraction provides a clear distinction between application and device driver software. This driver abstraction promotes reusable application code that is resistant to changes in the underlying hardware. In addition, the HAL standard makes it straightforward to write drivers for new hardware peripherals that are consistent with existing peripheral drivers.

Getting Started with the Hardware Abstraction Layer

The easiest way to get started using the HAL is to create a software project. In the process of creating a new project, you also create a HAL BSP. You need not create or copy HAL files, and you need not edit any of the HAL source code. The Nios II SBT generates the HAL BSP for you.

For an exercise in creating a simple Nios II HAL software project, refer to "Getting Started with Eclipse" in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

In the Nios II SBT command line, you can create an example BSP based on the HAL using one of the **create-this-bsp** scripts supplied with the Nios II Embedded Design Suite.

You must base the HAL on a specific hardware system. A Nios II system consists of a Nios II processor core integrated with peripherals and memory. Nios II systems are generated by Qsys or SOPC Builder.

If you do not have a custom Nios II system, you can base your project on an Altera-provided example hardware system. In fact, you can first start developing projects targeting an Altera® development board, and later re-target the project to a custom board. You can easily change the target hardware system later.

For information about creating a new project with the Nios II SBT, refer to the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*, or to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*.

HAL Architecture for Embedded Software Systems

This section describes the fundamental elements of the HAL architecture.

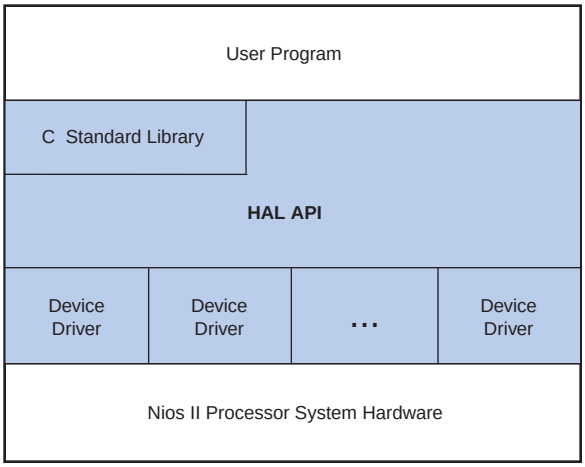
Services

The HAL provides the following services:

- Integration with the newlib ANSI C standard library—Provides the familiar C standard library functions
- Device drivers—Provides access to each device in the system
- The HAL API—Provides a consistent, standard interface to HAL services, such as device access, interrupt handling, and alarm facilities
- System initialization—Performs initialization tasks for the processor and the runtime environment before `main()`
- Device initialization—Instantiates and initializes each device in the system before `main()` runs

Figure 5–1 shows the layers of a HAL-based system, from the hardware level up to a user program.

Figure 5–1. The Layers of a HAL-Based System



Applications versus Drivers

Application developers are responsible for writing the system's `main()` routine, among other routines. Applications interact with system resources either through the C standard library, or through the HAL API. Device driver developers are responsible for making device resources available to application developers. Device drivers communicate directly with hardware through low-level hardware access macros.

For further details about the HAL, refer to the following chapters:

- The *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook* describes how to take advantage of the HAL to write programs without considering the underlying hardware.
- The *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook* describes how to communicate directly with hardware and how to make hardware resources available with the HAL API.

Generic Device Models

The HAL provides generic device models for classes of peripherals found in embedded systems, such as timers, Ethernet MAC/PHY chips, and I/O peripherals that transmit character data. The generic device models are at the core of the HAL's power. The generic device models allow you to write programs using a consistent API, regardless of the underlying hardware.

Device Model Classes

The HAL provides models for the following classes of devices:

- Character-mode devices—Hardware peripherals that send and/or receive characters serially, such as a UART.
- Timer devices—Hardware peripherals that count clock ticks and can generate periodic interrupt requests.
- File subsystems—A mechanism for accessing files stored in physical device(s). Depending on the internal implementation, the file subsystem driver might access the underlying device(s) directly or use a separate device driver. For example, you can write a flash file subsystem driver that accesses flash using the HAL API for flash memory devices.
- Ethernet devices—Devices that provide access to an Ethernet connection for a networking stack such as the Altera-provided NicheStack® TCP/IP Stack - Nios II Edition. You need a networking stack to use an ethernet device.
- Direct memory access (DMA) devices—Peripherals that perform bulk data transactions from a data source to a destination. Sources and destinations can be memory or another device, such as an Ethernet connection.
- Flash memory devices—Nonvolatile memory devices that use a special programming protocol to store data.

Benefits to Application Developers

The HAL defines a set of functions that you use to initialize and access each class of device. The API is consistent, regardless of the underlying implementation of the device hardware. For example, to access character-mode devices and file subsystems, you can use the C standard library functions, such as `printf()` and `fopen()`. For application developers, you need not write low-level routines just to establish basic communication with the hardware for these classes of peripherals.

Benefits to Device Driver Developers

Each device model defines a set of driver functions necessary to manipulate the particular class of device. If you are writing drivers for a new peripheral, you need only provide this set of driver functions. As a result, your driver development task is predefined and well documented. In addition, you can use existing HAL functions and applications to access the device, which saves software development effort. The HAL calls driver functions to access hardware. Application programmers call the ANSI C or HAL API to access hardware, rather than calling your driver routines directly. Therefore, the usage of your driver is already documented as part of the HAL API.

C Standard Library—newlib

The HAL integrates the ANSI C standard library in its runtime environment. The HAL uses newlib, an open-source implementation of the C standard library. newlib is a C library for use on embedded systems, making it a perfect match for the HAL and the Nios II processor. newlib licensing does not require you to release your source code or pay royalties for projects based on newlib.

The ANSI C standard library is well documented. Perhaps the most well-known reference is *The C Programming Language* by B. Kernighan and D. Ritchie, published by Prentice Hall and available in over 20 languages. Redhat also provides online documentation for newlib at **http://sources.redhat.com/newlib**.

Embedded Hardware Supported by the HAL

This section summarizes Nios II HAL support for Nios II hardware.

Nios II Processor Core Support

The Nios II HAL supports all available Nios II processor core implementations.

Supported Peripherals

Altera provides many peripherals for use in Nios II processor systems. Most Altera peripherals provide HAL device drivers that allow you to access the hardware with the HAL API. The following Altera peripherals provide full HAL support:

- Character mode devices
  - UART core
  - JTAG UART core
  - LCD 16207 display controller

- Flash memory devices
  - Common flash interface compliant flash chips
  - Altera's erasable programmable configurable serial (EPCS) serial configuration device controller
- File subsystems
  - Altera host based file system
  - Altera read-only zip file system
- Timer devices
  - Timer core
- DMA devices
  - DMA controller core
  - Scatter-gather DMA controller core
- Ethernet devices
  - Triple Speed Ethernet MegaCore® function
  - LAN91C111 Ethernet MAC/PHY Controller

The LAN91C111 and Triple Speed Ethernet components require the MicroC/OS-II runtime environment.

For more information, refer to the *Ethernet and the NicheStack TCP/IP Stack - Nios II Edition* chapter of the *Nios II Software Developer's Handbook.* Third-party vendors offer additional peripherals not listed here. For a list of other peripherals available for the Nios II processor, visit the Embedded Software page of the Altera website.

All peripherals (both from Altera and third party vendors) must provide a header file that defines the peripheral's low-level interface to hardware. Therefore, all peripherals support the HAL to some extent. However, some peripherals might not provide device drivers. If drivers are not available, use only the definitions provided in the header files to access the hardware. Do not use unnamed constants, such as hard-coded addresses, to access a peripheral.

Inevitably, certain peripherals have hardware-specific features with usage requirements that do not map well to a general-purpose API. The HAL handles hardware-specific requirements by providing the UNIX-style `ioctl()` function. Because the hardware features depend on the peripheral, the `ioctl()` options are documented in the description for each peripheral.

Some peripherals provide dedicated accessor functions that are not based on the HAL generic device models. For example, Altera provides a general-purpose parallel I/O (PIO) core for use with the Nios II processor system. The PIO peripheral does not fit in any class of generic device models provided by the HAL, and so it provides a header file and a few dedicated accessor functions only.

For complete details regarding software support for a peripheral, refer to the peripheral's description. For further details about Altera-provided peripherals, refer to the *Embedded Peripherals IP User Guide*.

MPU Support

The HAL does not include explicit support for the optional memory protection unit (MPU) hardware. However, it does support an advanced exception handling system that can handle Nios II MPU exceptions.

For details about handling MPU and other advanced exceptions, refer to the *Exception Handling* chapter of the *Nios II Software Developer's Handbook*. For details about the MPU hardware implementation, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

MMU Support

The HAL does not support the optional memory management unit (MMU) hardware. To use the MMU, you need to implement a full-featured operating system.

For details about the Nios II MMU, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Document Revision History

Table 5–1 shows the revision history for this document.

Table 5–1. Document Revision History

Date	Version	Changes
May 2011	11.0.0	Introduction of Qsys system integration tool
February 2011	10.1.0	Removed "Referenced Documents" section.
July 2010	10.0.0	Maintenance release.
November 2009	9.1.0	Maintenance release.
March 2009	9.0.0	■ Reorganized and updated information and terminology to clarify role of Nios II Software Build Tools. ■ Corrected minor typographical errors.
May 2008	8.0.0	Maintenance release.
October 2007	7.2.0	Maintenance release.
May 2007	7.1.0	■ Scatter-gather DMA core. ■ Triple-speed Ethernet MAC. ■ Refer to HAL generation with Nios II Software Build Tools. ■ Added table of contents to "Introduction" section. ■ Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	NicheStack TCP/IP Stack - Nios II Edition.
May 2006	6.0.0	Maintenance release.
October 2005	5.1.0	Maintenance release.
May 2005	5.0.0	Maintenance release.
May 2004	1.0	Initial release