



EXCALIBUR™

Nios Ethernet

Development Kit

**User Guide
July 2001
Version 1.0**



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>

A-UG-NIOETHERKIT-1.0
P25-06807-00

Copyright © 2001 Altera Corporation. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. ModelSim is a registered trademark of Mentor Graphics Corporation. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services. All rights reserved.





About This User Guide

User Guide Contents

The purpose of this manual is to provide you with the information necessary to get you started using the Nios™ Ethernet Development Kit (EDK). This manual provides:

- An overview of the Nios EDK, its contents, and its intended use
- A getting started section with a step-by-step guide to installing the development tools, installing hardware, and accessing the software application examples. A daughter card reference section providing a description of the daughter card including a functional overview, pinout information, and descriptions of the PC-board design files included with the kit
- A software overview introducing you to the C-language library, providing a description of the supported protocols and the general structure of the provided functions and data structures
- A plugs library reference describing the software routines

Table 1 below shows the *Nios Ethernet Development Kit User's Guide* revision history.

Table 1. Revision History		
Revision	Date	Description
Version 1.0	July 2001	Nios Ethernet Development Kit User Guide - printed

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide web site at <http://www.altera.com>.

For additional information about Altera products, consult the sources shown in Table 2.






Information Type	Access	USA & Canada	All Other Locations
Altera Literature Services	Electronic mail	lit_req@altera.com (1)	lit_req@altera.com (1)
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-7606	(408) 544-7606
Technical support	Telephone hotline	(800) 800-EPLD (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 (1) (7:00 a.m. to 5:00 p.m. Pacific Time)
	Fax	(408) 544-6401	(408) 544-6401 (1)
	Electronic mail	support@altera.com	support@altera.com
	FTP site	ftp.altera.com	ftp.altera.com
General product information	Telephone	(408) 544-7104	(408) 544-7104 (1)
	World-wide web site	http://www.altera.com	http://www.altera.com

Note:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The *Nios Ethernet Development Kit User Guide* uses the typographic conventions shown in Table 3.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , \maxplus2 directory, d: drive, chiptrip.gdf file.
<i>Bold italic type</i>	Book titles are shown in bold italic type with initial capital letters. Example: <i>1999 Device Data Book</i> .
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75 (High-Speed Board Design)</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>tPIA</i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (<>) and shown in italic type. Example: < <i>file name</i> >, < <i>project name</i> >.pdf file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of Quartus II and MAX+PLUS II Help topics are shown in quotation marks. Example: “Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix <i>_n</i> , e.g., reset_n. Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\max2work\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets are used in a list of items when the sequence of the items is not important.
	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

About This User Guide



Notes:



About This User Guide	iii
How to Contact Altera	iv
Typographic Conventions	v
Overview	1
Nios Ethernet Development Kit Description	1
Installed Components	2
MAC Addresses	3
Getting Started	5
Verifying Kit Contents	5
Setting Up the Daughter Card	5
Installing the Hardware and Software Files	8
Loading the Reference Design	9
Running Example Applications	12
The Hello Plugs Application Example	12
Configure Your Network Settings	14
The Networked-Based GERMS Monitor Application Example	16
The Simple Web Server Application Example	18
Daughter Card	21
Daughter Card Components	21
Functional Overview	22
Stacking Daughter Cards	22
SOPC Builder Library Component	23
Connector Pinouts	23
Nios System to Daughter Card Pin Map	25
Software Overview	29
Software Description	29
System Requirements	29
Protocols Supported	29
Library Features	29
Protocols Architecture	30
Standards	31
ARP (RFC 826)	31
IP (RFC 791)	31
ICMP (RFC 792)	31
UDP (RFC 768)	31
DNS (RFC 1034 & 1035)	32
TCP (RFC 793)	32
Build Options	32
PLUGS_DEBUG (Default Value = 1)	32

PLUGS_PLUG_COUNT (Default Value = 6).....	32
PLUGS_ADAPTER_COUNT (Default Value = 2)	32
PLUGS_DNS (Default Value = 1)	33
PLUGS_PING (Default Value = 1)	33
PLUGS_TCP (Default Value = 1).....	33
Byte Ordering.....	33
Data Structures.....	34
Payload Descriptions.....	37
Plugs Library Routines.....	39
nr_plugs_initialize	40
nr_plugs_terminate.....	41
nr_plugs_set_mac_led.....	42
nr_plugs_create	43
typedef int (*nr_plugs_receive_callback_proc)	45
nr_plugs_destroy	47
nr_plugs_connect.....	48
nr_plugs_send	50
nr_plugs_send_to.....	51
int nr_plugs_listen	52
typedef int (*nr_plugs_listen_callback_proc)	53
nr_plugs_ip_to_ethernet.....	54
nr_plugs_name_to_ip.....	55
nr_plugs_idle.....	56
void nr_plugs_print_ethernet_packet.....	57
nr_n2h16.....	58
nr_h2n16.....	58
nr_n2h32.....	58
nr_h2n32.....	58
Nios Terminology.....	59

Nios Ethernet Development Kit Description

The Nios Ethernet Development Kit (EDK) includes hardware and software components that provide network connectivity for your Nios-based embedded systems. The components included in this kit are:

- A network-interface daughter card that can plug directly into the Nios development board.
- An SOPC Builder library component that defines the logic and interface signals necessary to use the daughter card in a Nios system.
- A C-language library that provides a network-protocol stack. This library includes support for raw Ethernet, ARP, IP, ICMP, UDP, and TCP protocols and utility routines for controlling the daughter card hardware.

The kit includes APEX hardware reference designs and example software application programs. These reference designs and application examples are intended as starting points to be modified by you for your specific network-enabled application.

The Nios EDK library components and tools can be installed on Solaris, HP-UX or PC-Windows (98/NT/2000).

 The Excalibur™ Development Kit, featuring the Nios embedded processor must be installed before you can use the Nios Ethernet Development Kit.

The following items are included in the Nios EDK:

- Nios EDK daughter card based on the Cirrus Logic CS8900A PHY/MAC chip
- Male-to-male RJ-45 network cable
- Female-to-male crossover adapter, used for direct PC connection
- Nios EDK CD-ROM

The Nios EDK CD-ROM contains the following files:

- SOPC library components
- PC-board schematic and layout files for the Nios EDK daughter card
- Example hardware reference design configurations:

- Nios 32-bit CPU for a single daughter card
- Nios 16-bit CPU for a single daughter card
- Nios 32-bit for dual-stacked daughter cards

- Example software applications:

- Library general demonstration and configuration programs
- Example web server
- Nios 32-bit CPU network-based GERMS monitor application example

- Documentation:

- *CS8900A Product Data Sheet*
- *Nios Ethernet Development Kit User Guide*

Installed Components

The Nios EDK CD-ROM includes an InstallShield installation wizard for Windows workstations, and install scripts for Unix workstations. A step-by-step installation procedure is described in. See "Installing the Hardware and Software Files" on page 8.

When installed on your computer, the Nios EDK will add files to your SOPC Builder home `<SOPC-HOME>` directory. The default directory is `C:\Altera\excalibur\sopc_builder`. These files are:

- The SOPC Builder library component directory is:

- `<SOPC-HOME>\components\altera_avalon_cs8900\`

- The APEX hardware (PLD) reference designs are in:

- `<SOPC-HOME>\examples\ethernet_kit_reference_design\`
- `<SOPC-HOME>\examples\ethernet_kit_reference_design_16_bit\`
- `<SOPC-HOME>\examples\ethernet_stacked_reference_design\`

- The PDF documentation files are in:

- `<SOPC-HOME>\documents\`

- A complete set of PC-board manufacturing documents for the daughter card. This includes all design-files necessary to build and assemble the daughter card board and components. These documents are found in:

- `<SOPC-HOME>\documents\nedk_daughtercard_documents\`

The Nios EDK daughter card manufacturing and design documents give you all the necessary information to build copies of the daughter card yourself. You may also use these design files to cut-and-paste sections of the daughter card design into your own custom PC-board schematic, layout, or bill of materials (BOM).

MAC Addresses

All Ethernet devices require a unique 48-bit MAC address. All Nios EDK kits ship with the same default MAC address. This MAC address serves as a placeholder during development. To obtain your own block of unique Ethernet MAC addresses for your products, refer to the following website: <http://www.standards.ieee.org/regauth/oui/index.shtml>.

A single Nios EDK system can use the default MAC address on a LAN without a conflict. However, two Nios EDK systems with the same MAC address will cause conflicts. If you are using two or more Nios EDK systems on the same LAN you must assign a unique Ethernet MAC address to each system.

The Nios Development Kit and the Nios EDK form a prototyping platform for creating your custom embedded, networked system. These development boards, reference designs and applications allow you to rapidly prototype your application.

Overview



Notes:



This section explains how to set up the Nios EDK daughter card, install the Nios EDK hardware and software files, load the hardware reference design into the board and run the Ethernet application examples.

Verifying Kit Contents

Verify the following items are included in your Nios Ethernet Development Kit:

- Nios EDK daughter card based on the CS8900A PHY/MAC chip
- Male-to-male RJ-45 network cable
- Female-to-male crossover adapter, used for direct-PC connection
- Nios EDK CD-ROM
- O'Reilly *Internet Core Protocols Manual*
- *Nios Ethernet Development Kit User Guide*

Setting Up the Daughter Card

The hardware reference designs included with the Nios EDK assumes the daughter card is connected to the prototype connectors JP8, JP9, and JP10 on the Nios development board and that your board is already set up.



If you are setting up your Nios development board for the first time, refer to *Nios Embedded Processor Getting Started User Guide* that shipped with your Nios Development Kit.

1. Verify that your Nios board is set up correctly and the power is off.


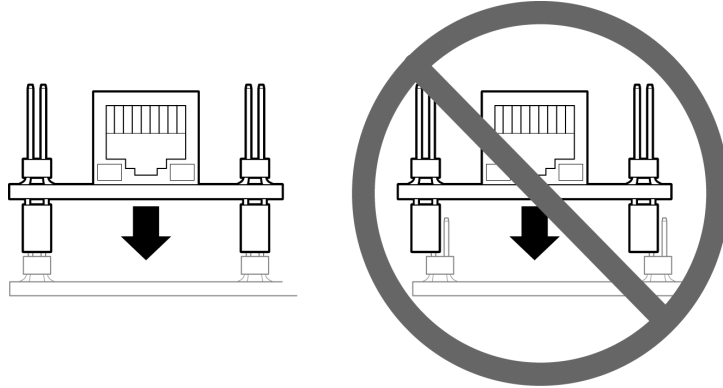
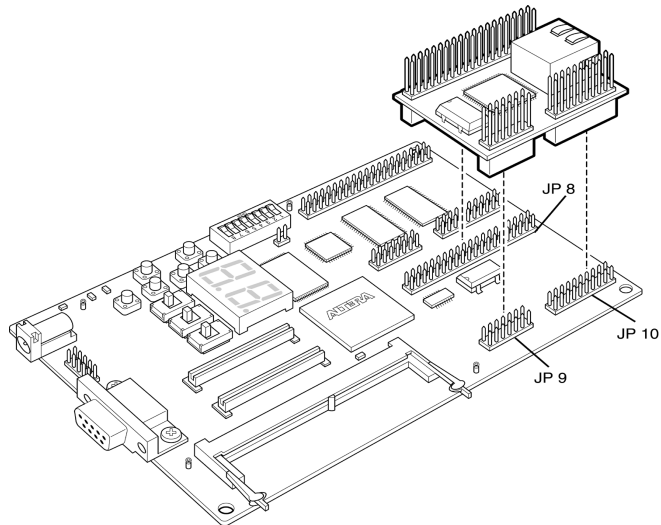
-  When connecting the daughter card, make sure you place the card on the prototype connectors correctly as shown in Figure 1. If you do not, the board may be permanently damaged.

Figure 1. Incorrect Daughter Card Connection



2. Place the daughter card on the JP8, JP9 and JP10 prototype connectors.

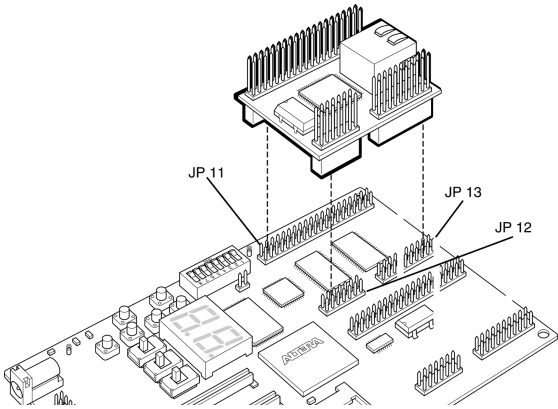
Figure 2. Daughter Card Placement for Use with the APEX Reference Design





The daughter card can also be placed on the JP11, JP12 and JP13 prototype connectors. However, the APEX hardware reference designs shipped with the kit will not work.

Figure 3. Alternative Placement of the Daughter Card

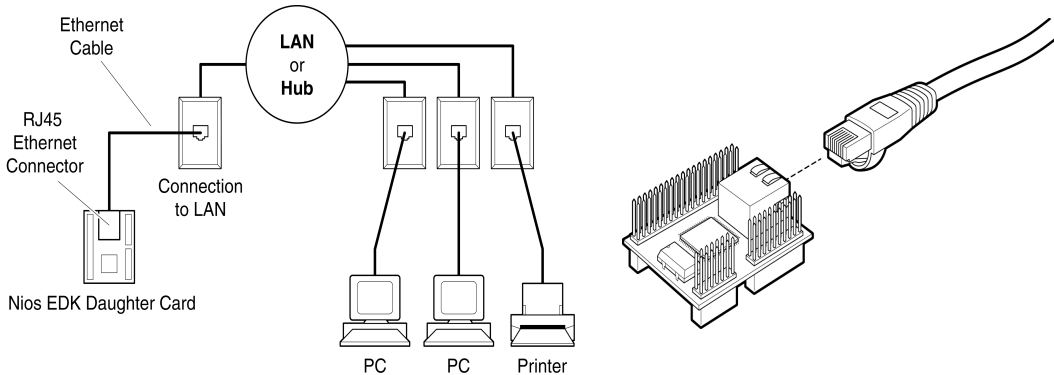


3. Connect the network cable:

If you are connected to a LAN or HUB:

Connect the male-to-male networking cable to the RJ-45 connector on the daughter card as shown in Figure 4.

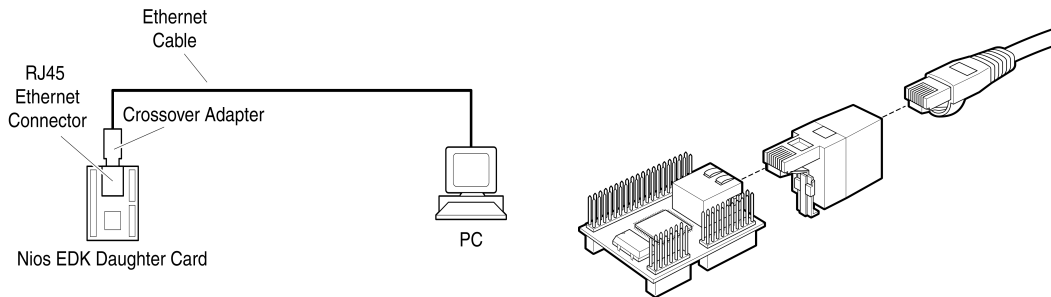
Figure 4. Using the Nios EDK with a LAN or HUB



Getting Started

If you are not using a LAN or HUB connection and are connected directly to a workstation or PC Ethernet jack, insert the female-to-male RJ-45 crossover adapter to the networking cable and then connect the adapter to the RJ-45 on the daughter card as shown in Figure 5.

Figure 5. Using the Nios EDK with a Workstation or PC Ethernet Jack



4. Connect the other end of the networking cable to your Ethernet LAN or workstation.

Installing the Hardware and Software Files

The instructions below are for a Windows PC. These instructions assume you have already installed or upgraded your Nios Development Kit to version 1.1.1 or higher.

1. Insert the Nios EDK CD-ROM into your CD-ROM drive. The InstallShield installer begins automatically.



Unix users: See the **Readme** text file for installation instructions.

2. Follow the install instructions. The screen in Figure 6 appears when you have completed the installation. Click **Finish**.

Figure 6. Install Completed



2

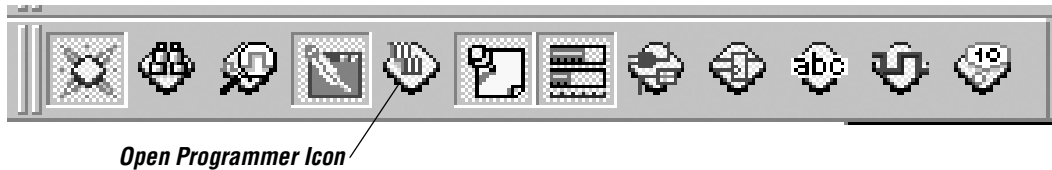
Getting
Started

Loading the Reference Design

This section explains how you will use Quartus® II version 1.0 software to load the Nios EDK reference design into the Nios development board. All instructions assume you are using Quartus II.

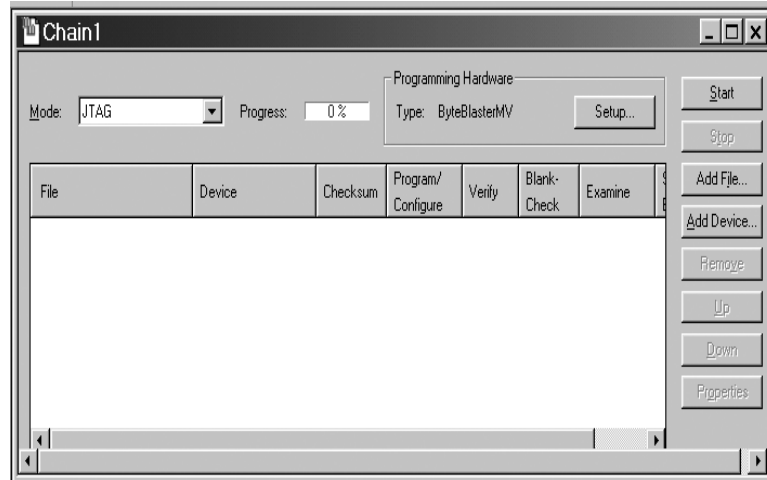
1. Click the **Start Menu > Programs > Altera > Excalibur > Nios Ethernet Development Kit Reference Design**.
2. In the Quartus II software, double-click the **Open Programmer** icon.

Figure 7. Quartus II Version 1.0 Icons



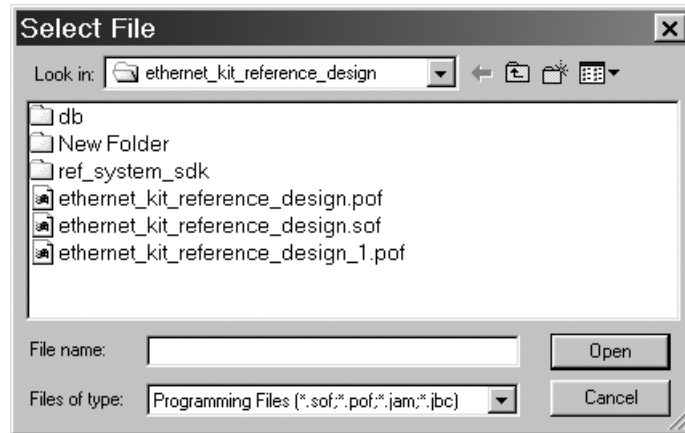
The **Chain 1** dialog box appears as shown in Figure 8.

Figure 8. Chain 1 Dialog Box



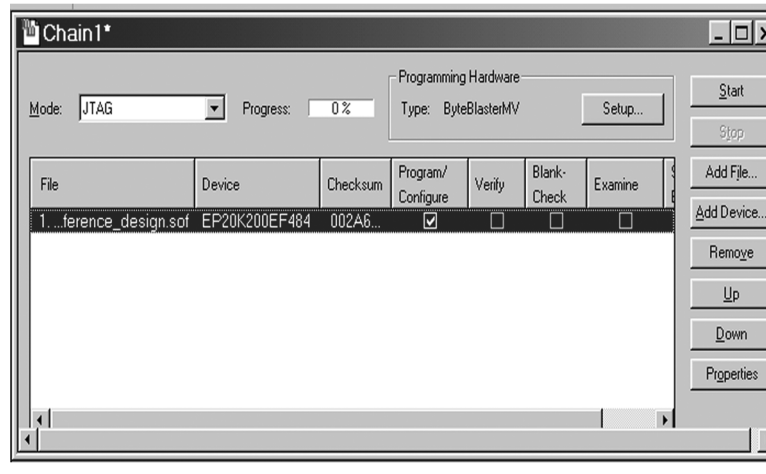
3. Click **Add File**. The **Select File** dialog box appears as shown in Figure 9.

Figure 9. The Select File Dialog Box



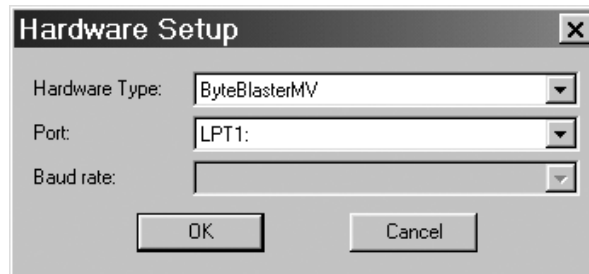
- Double-click the `ethernet_kit_reference_design.sof` file and the file appears in the **Chain 1** dialog box as shown in Figure 10.

Figure 10. Chain 1 Dialog Box



- Click the check box in the **Program/Configure** column as shown in Figure 10.
- Check the Programming Hardware section of the screen as shown in Figure 11. ByteBlasterMV should appear in the Type field selection. To change the type, click **Setup** and select **ByteBlaster** from the **Hardware Type** drop-down list box.

Figure 11. ByteBlasterMV Selection



If ByteBlasterMV is not an available selection, you will need to install the ByteBlaster driver. You *do not* need to install drivers if you are using Windows 98.



For information about installing the ByteBlasterMV driver, see page 23 in the *Quartus Installation & Licensing for PCs Manual* or go to <http://www.altera.com> for the PDF version of this manual.

7. Click **Start**. The two-digit 7-segment display on the Nios development board turns off. When the download is completed, the **Progress** bar reads 100% and the dual 7-segment LED display lights turns on.



If you encounter a JTAG error, see page 12 of the *Nios Embedded Processor Development Board Manual* for setting the switches correctly.

Running Example Applications

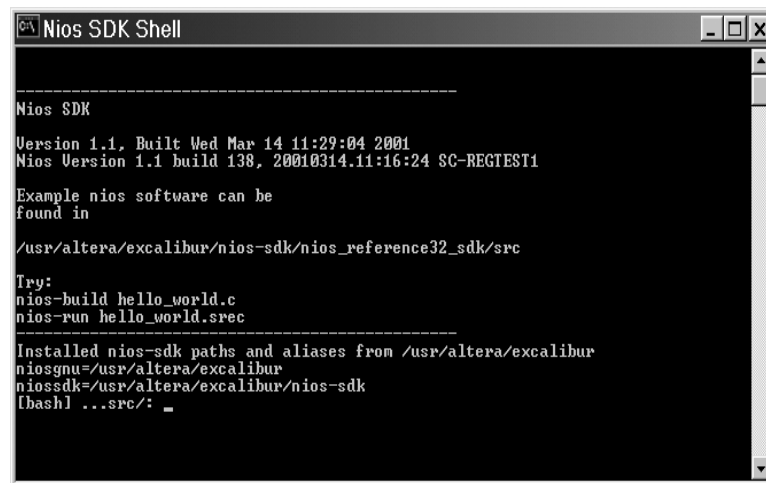
Using the bash shell you will now run the example applications. If you are unfamiliar with bash, refer to the *Nios Embedded Processor Software Development Reference Manual* for more information.

The Hello Plugs Application Example


To run Hello Plugs, follow these steps:

1. From the Windows **Start Menu**, select **Altera > Excalibur > Nios SDK Shell**. The bash window appears and displays the [bash]: prompt as shown in Figure 12.

Figure 12. The Nios SDK Shell



2. At the [bash]: prompt, type `cd //c/Altera/Excalibur/sopc_builder/examples/ethernet_kit_reference_design/ref_system_sdk/src` ←

 If you installed the Nios EDK program in another directory, make the appropriate change in step 2.

3. Type `nios-build hello_plugs.c` ←

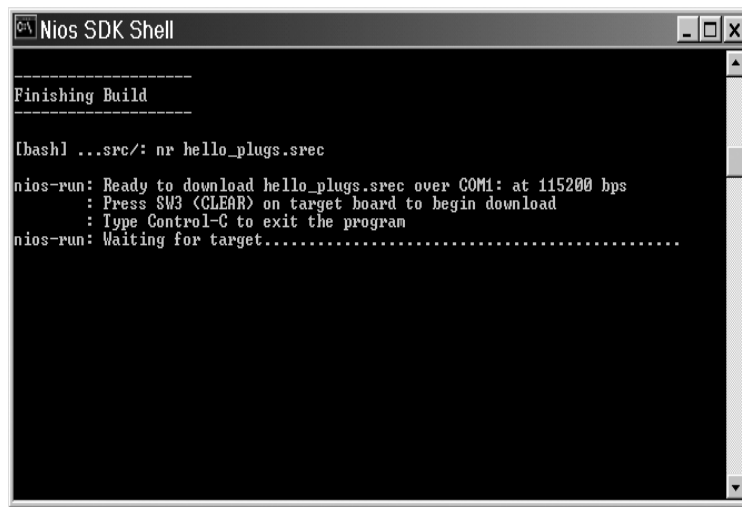


The default setting for the `nios-run` utility is `<COM1>`. For more information about specifying another serial port when executing `nios-run`, see *Appendix A* of the *Nios Embedded Processor Software Development Reference Manual* found at the Altera website (<http://www.altera.com>).

4. Type `nios-run hello_plugs.srec` ←

5. Press SW3 to clear the Nios development board. After pressing SW3, the `hello_plugs.srec` file begins downloading to the board as shown in Figure 13.


Figure 13. Downloading File to the Development Board



```

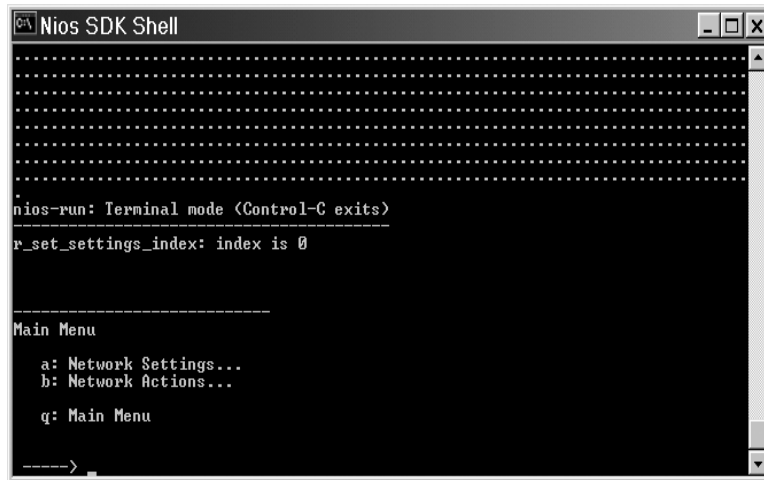
Nios SDK Shell
-----
Finishing Build
-----
[ubash] ..src/: nr hello_plugs.srec
nios-run: Ready to download hello_plugs.srec over COM1: at 115200 bps
: Press SW3 (CLEAR) on target board to begin download
: Type Control-C to exit the program
nios-run: Waiting for target.....

```

 If this is the first time you are using the Nios product, the Nios Peripheral Test Menu could appear. If this menu appears, press **Ctrl+C** and then repeat Step 4.

Once the download is completed, the Hello Plugs Main Menu appears as shown in Figure 14.


Figure 14. The Nios Ethernet Hello Plugs Main Menu



Before you can use Hello Plugs, the network-based GERMS monitor, or the simple web server application examples, you must first configure the network settings.

Configure Your Network Settings

The instructions for configuring your network settings will only work if your PC is connected directly to your Nios development board with a Nios EDK daughter card using an Ethernet cable and crossover adapter.

 You can also connect your Ethernet daughter card to your office LAN. Consult with your system administrator.

1. Type a **←** to select **Network Settings** to set the network settings of your Ethernet card. The Network Setting Menu appears as shown in Figure 15.

Figure 15. Configuring Your Network Settings

```

C:\ Nios SDK Shell
---->

-----
Network Settings

Network Settings 1 (of 4)
    ethernet address: 14:13:12:11:16:15
    ip address: 10.0.0.51
nameserver ip address: 165.227.1.1
    subnet mask: 255.255.255.0
    gateway ip address: 10.0.0.255




a: Select Setup
b: Save To Flash
c: Enter New Settings
d: Reset All Settings
q: Main Menu

---->

```

2

Getting
Started

2. Type **d**  to select **Reset All Settings**.
3. Type **c**  to select **Enter New Settings**. You will then be prompted to enter each of the five settings shown above.
4. Press  to keep the default settings for the **ethernet address**.
5. Type your PC's IP address for the **IP address**. Change the last number in the IP address up or down one digit (the allowable range is 2 to 254). For example, if your PC is using IP address 64.3.99.73, set your Nios EDK to 64.3.99.72 or 64.3.99.74.



If you are connecting your Nios EDK to your office LAN, ask your system administrator for an unused static IP address. If you are using more than one Nios board on your LAN, give each board a distinct Ethernet address (MAC address). Below are safe MAC addresses for you to use:

00:42:00:00:23:00

00:23:23:00:23:23

00:42:42:42:42:00

6. Type your PC's nameserver address for the **nameserver ip address**.



You will only use the **nameserver ip address** if your Nios EDK is connected to a LAN.

7. Use the default setting for the **subnet mask**. If you are connected to a LAN, use the same subnet mask that your PC uses.
8. Type the same setting as your PC for the **gateway ip address**.



You will only use the **gateway ip address** if your Nios EDK is connected to a LAN.

9. Type **b** (Save To Flash) after entering all the settings. This writes the the network settings to the Nios development board's Flash memory.



See the description of the plugs library routine `nr_plugs_initialize()` for an explanation about using these stored settings in your own software applications.

The Networked-Based GERMS Monitor Application Example

To use this application example, you should be familiar with the Nios GERMS monitor and you must first run the Hello Plugs application to setup your network parameters. See "The Hello Plugs Application Example" on page 12 for more information.



For more information about using the GERMS monitor refer to the *Nios Embedded Processor Software Development Reference Manual* that shipped with your Excalibur™ Development Kit, featuring the Nios embedded processor.

To run the networked-based GERMS monitor application example, follow these steps:

1. Run the Hello Plugs application to setup your network parameters if you have not done so already. This only needs to be done once. When completed, press **Ctrl+C**.

2. Build the germs server and client by typing the following command:

```
make -f Makefile_nedk all ←
```

3. Press **SW3** to clear the development board.

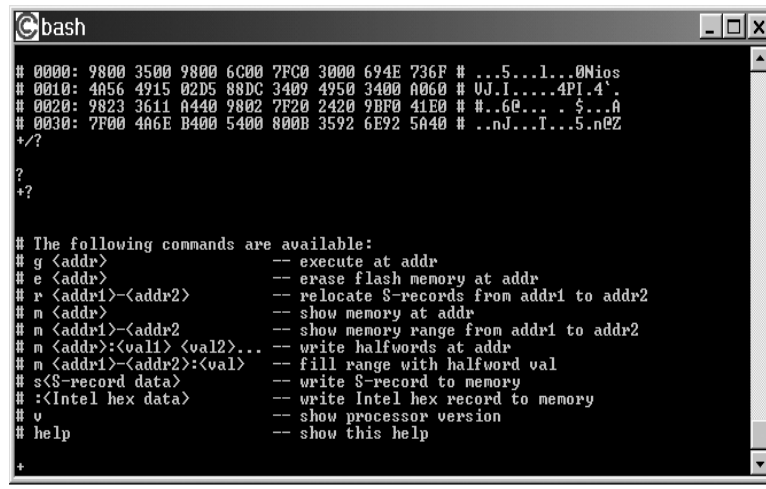
4. Download the germs server application SREC (serial transfer) by typing the following command:

```
nios-run -x germs_server.srec ←
```



For more information about the `nios-run` command line options, see *Appendix A* of the *Nios Embedded Processor Software Development Reference Manual* found at the Altera website (<http://www.altera.com>).

Figure 17. The Nios GERMS Menu



```

bash
# 0000: 9800 3500 9800 6C00 7FC0 3000 694E 736F # ...5...1...0Nios
# 0010: 4A56 4915 02D5 88DC 3409 4950 3400 A060 # UJ.I....4PI.4'
# 0020: 9823 3611 A440 9802 7F20 2420 9BF0 41E0 # ..6@... . $.A
# 0030: 7F00 4A6E B400 5400 800B 3592 6E92 5A40 # ..nJ...I...5.n@Z
+/?
?
+?
# The following commands are available:
# g <addr>          -- execute at addr
# e <addr>          -- erase flash memory at addr
# r <addr1>-<addr2> -- relocate S-records from addr1 to addr2
# m <addr>          -- show memory at addr
# n <addr1>-<addr2> -- show memory range from addr1 to addr2
# m <addr>:<val1> <val2>... -- write halfwords at addr
# m <addr1>-<addr2>:<val> -- fill range with halfword val
# s<S-record data> -- write S-record to memory
# :<Intel hex data> -- write Intel hex record to memory
# v                -- show processor version
# help            -- show this help
+

```

- To change the dual 7-segment display, type `m420:3636` ← This command confirms your networked GERMS monitor-based application example is working.



For more information, see the `Readme_nedk_germs.txt` file included in your kit.

The Simple Web Server Application Example

To use this application example, you must first run the Hello Plugs application to setup your network parameters. See “The Hello Plugs Application Example” on page 12 for more information. To run the simple web server application example, follow these steps:

- Begin by building a flash image of the web pages. To do this, type the following command:

```

wosfs_maker.pl exc-nios.gif index.html
template_page.html.template 404_page.html
static_page.html > pages.flash ←

```

- Store the web pages in Flash memory by typing the following command:

```

nios-run -x pages.flash ←

```



For more information about the `nios-run` command line options, see *Appendix A of the Nios Embedded Processor Software Development Reference Manual* found at the Altera website (<http://www.altera.com>).

3. Build the example web server by typing the following command:

```
nios-build wosfs.c nedk_example_web_server.c ↵
```

4. Run the web server by typing the following command:

```
nios-run nedk_example_web_server.srec ↵
```

5. Open your web browser to view the web page you built.
6. In the **Address** field, enter the IP address you used as a network settings for your Ethernet card to display your web page.

Getting Started



Notes:

Daughter Card Components

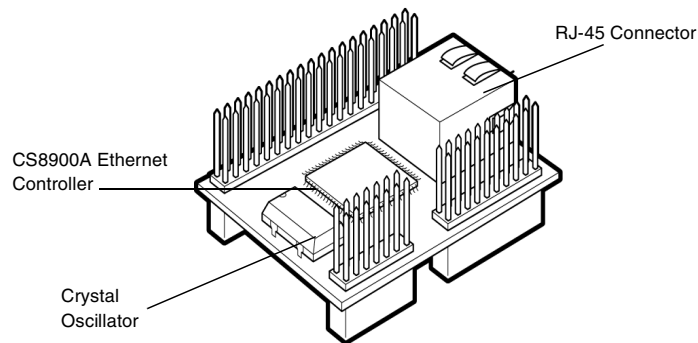
This section describes the network interface daughter card included in the Nios EDK.

The Nios EDK daughter card is a circuit-board with the following components: See Figure 18.

- A Cirrus Logic CS8900A integrated Ethernet 10Mbit PHY/MAC chip
- A RJ-45 network connector with integrated-transformer magnetics and Link/LAN LEDs
- Three female connectors for mounting on the Nios development board
- Three male headers for stacking two daughter cards
- A 20 MHz crystal oscillator that is used by the CS8900A chip
- All necessary resistors and capacitors



A complete manufacturing bill of materials for the daughter card is provided in the installed `nedk_daughter_card_documents` directory.

Figure 18. The Nios EDK Daughter Card



Functional Overview

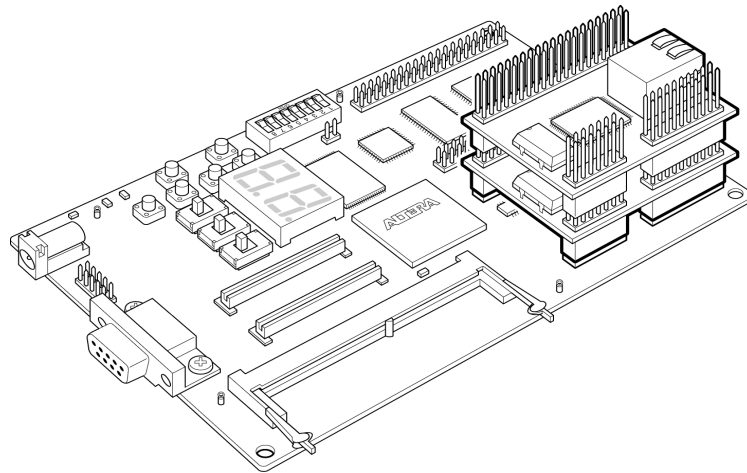
The main functional component on the Nios EDK daughter card is a CS8900A integrated PHY/MAC chip. A Portable Document Format (PDF) data sheet for this chip is included in the installed Nios EDK documentation. The CS8900A chip presents an ISA-bus interface to the Nios CPU. The necessary electrical-interface signals are provided on the set of female connectors. These connectors are compatible with the Expansion Prototype Connector groups on the Nios Development Board. The Nios EDK daughter card is compatible with either the 5-V (JP11, JP12, JP13) or the 3.3-V (JP8, JP9, JP10) Expansion Prototype Connector groups. The daughter card does not use any 5-V signals.

-  All of the included reference designs use a Nios EDK daughter card connected to the 3.3-V Expansion Prototype Connector group (JP8, JP9, and JP10).
-  To use a Nios EDK daughter card connected to the 5-V Expansion Prototype Connector group (JP11, JP12 and JP13), you will need to create a new APEX configuration with the appropriate pin-assignments.

Stacking Daughter Cards

The Nios EDK daughter card connectors are arranged such that two daughter cards can be stacked vertically as shown in Figure 19. Two stacked daughter cards can be accessed via the same (shared) tri-state data bus. The Nios EDK includes only one daughter card. The electrical interface does not support stacks more than two cards deep.

Figure 19. Stacked Daughter Cards



SOPC Builder Library Component

The Nios EDK includes an SOPC Builder library component that provides all logic and I/O signals necessary for using the daughter card. A library component is an add-on to the Nios SOPC Builder that makes a new peripheral available. After the Nios EDK is installed, you will see a new library component in the SOPC Builder's menu of available peripherals. The new component is named Ethernet Interface (CS8900). You may add one or more of these components to your Nios system using the Nios System Builder MegaWizard®.

Each Ethernet Interface (CS8900) component in your system will have an associated group of I/O pins on your system module. A detailed description of how to connect these system-module I/O pins to the Nios EDK daughter card can be found "Nios System to Daughter Card Pin Map" on page 25. An example of the necessary connections (pin-assignments) can also be found in the included reference designs.

To access two stacked daughter cards, their associated Ethernet Interface (CS8900) peripherals must be assigned to the same, shared tri-state data bus.

The CS8900A chip can be used in either memory mode or I/O-mode. For more information, see the *CS8900A Product Data Sheet*. The included Ethernet Interface (CS8900) library component and all associated software libraries use the CS8900A chip in I/O-mode. The electrical interface on the daughter card supports memory-mode operation, but none of the included Nios EDK interface logic, reference designs, or software libraries make use of this feature. All of the examples, software, and documentation in the Nios EDK show the CS8900A being used in I/O-mode.

Connector Pinouts

This section provides complete pinouts for connectors F8, F9, and F10 on the Nios EDK daughter card. The Nios CPU accesses the daughter card through these connectors. Most of the interface pins connect directly to device pins on the CS8900A chip. Where appropriate, the connector diagrams indicate the name of the CS8900A pin that corresponds to each connector pin. Detailed schematics showing all components and connections on the daughter card are found in the *CS8900A Product Data Sheet* found in your kit. See Figure 20 through Figure 22.

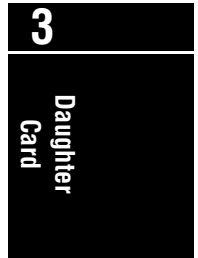


Figure 20. F8 Connector Pinouts

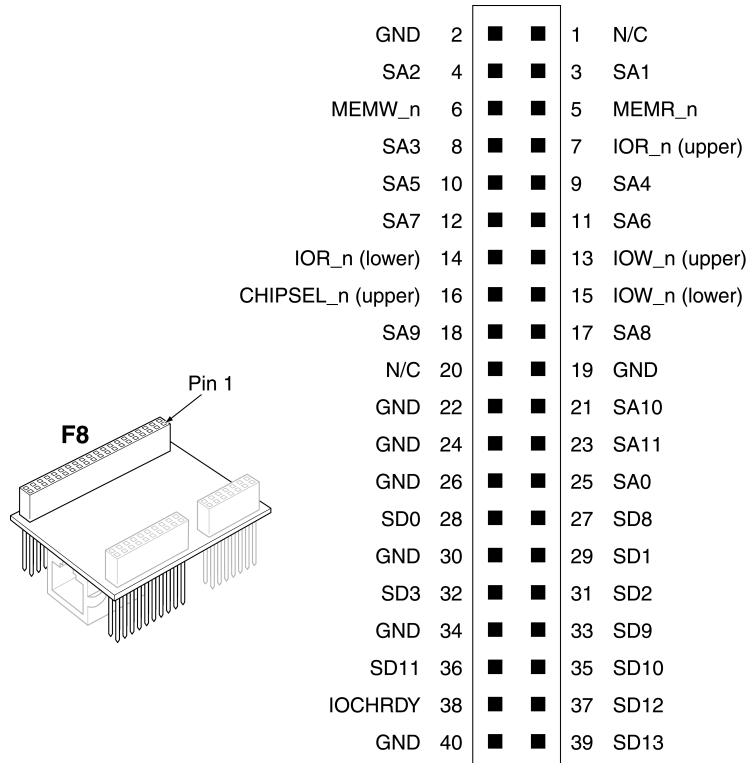


Figure 21. F9 Connector Pinouts

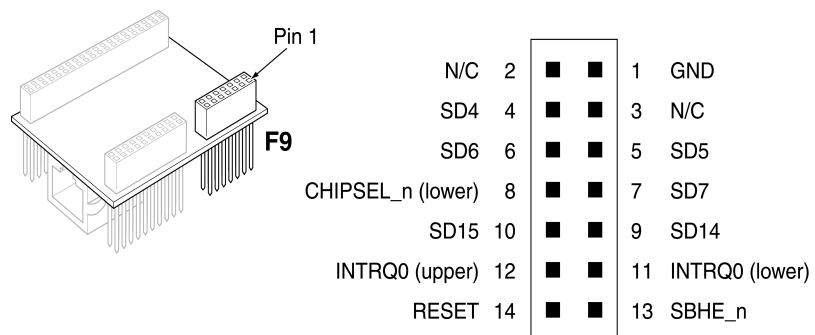
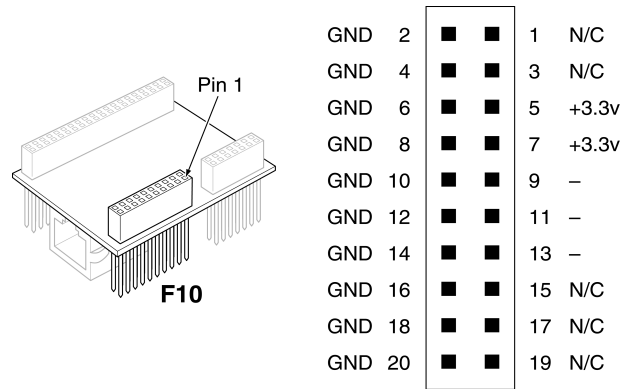


Figure 22. F10 Connector Pinouts



Nios System to Daughter Card Pin Map

Each Ethernet Interface (CS8900) peripheral in your Nios system will have an associated set of I/O pins on your system module. This section describes how to connect these system-module I/O pins to the daughter card. In general, you will establish these connections by making pin-assignments in your PLD design. The reference designs included with the kit include correct pin-assignment information that you can modify for your own design.

The names given to the system-module I/O ports will depend on the name you provide for the Ethernet Interface (CS8900) peripheral. In Tables 3 and 4, *<your_name>* indicates the name you assigned to this component. The name for some system-module I/O ports will also depend on the tri-state bus you have selected for this peripheral. In Tables 3 and 4, *<your_bus_name>* indicates the name of the bus to which you assigned this Ethernet Interface (CS8900) peripheral.

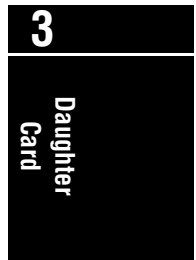
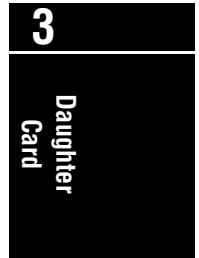


Table 3. Nios 32-bit CPU System Module I/O Port Name and Daughter Card Pin Name	
32-bit CPU System Module I/O Port Name	Daughter Card Pin Name (Lower of Two Stacked Cards)
<your_bus_name>_data	SD[15..0]
<your_bus_name>_address [4]	SA[3]
<your_bus_name>_address [3]	SA[2]
<your_bus_name>_address [2]	SA[1]
<your_bus_name>_byteenablen[1]	SHBE_n
ior_n_to_the_<your_name>	IOR_n (lower)
iow_n_to_the_<your_name>	IOW_n (lower)
irq_to_the_<your_name>	INTRQ0 (lower)
~(system module reset_n)	RESET
constant Logic-1	MEMW_n
constant Logic-1	MEMR_n
constant Logic-1	SA[9..8]
constant Logic-0	SA[11..10]
constant Logic-0	SA[7..4]
constant Logic-0	SA[0]
constant Logic-0	CHIPSEL_n (lower)
constant Logic-0	CHIPSEL_n (upper)

If you are connecting an Ethernet Interface (CS8900) peripheral component to the upper of the two stacked daughter cards, then substitute (*upper*) for (*lower*) in the right column of the above table.

Table 4. Nios 16-bit CPU System Module I/O Port Name and Daughter Card Pin Name

16-bit CPU System Module I/O Port Name	Daughter Card Pin Name (Lower of Two Stacked Cards)
<your_bus_name>_data	SD[15..0]
<your_bus_name>_address [3]	SA[3]
<your_bus_name>_address [2]	SA[2]
<your_bus_name>_address [1]	SA[1]
<your_bus_name>_byteenable[1]	SHBE_n
ior_n_to_the_<your_name>	IOR_n (lower)
iow_n_to_the_<your_name>	IOW_n (lower)
irq_to_the_<your_name>	INTRQ0 (lower)
~(system module reset_n)	RESET
constant Logic-1	MEMW_n
constant Logic-1	MEMR_n
constant Logic-1	SA[9..8]
constant Logic-0	SA[11..10]
constant Logic-0	SA[7..4]
constant Logic-0	SA[0]
constant Logic-0	CHIPSEL_n (lower)
constant Logic-0	CHIPSEL_n (upper)



If you are connecting an Ethernet Interface (CS8900) peripheral component to the upper of the two stacked daughter cards, then substitute (*upper*) for (*lower*) in the right column of the above table.

Daughter Card



Notes:

Software Description

The software library included in the Nios EDK is called the Plugs Library. The Plugs Library included in the Nios EDK allows your software to use network protocols for transmitting and receiving data.

System Requirements

- Nios CPU
- 20K code footprint
- 8K data footprint
- Nios Timer peripheral named timer 1

Protocols Supported

- Raw Ethernet
- Address resolution protocol (ARP)
- Internet protocol (IP)
- Internet control message protocol (ICMP)
- User datagram protocol (UDP)
- Transmission control protocol (TCP)

Library Features

- Access to low-level packets
- Access to high level-packet payloads
- Conforms to RFCs
- Allows you to open connections and send data with only a few lines of code.
- Is similar to the Unix-standard *sockets* routines.
- Each plug can be set to print debug information for either transmit or receive data.



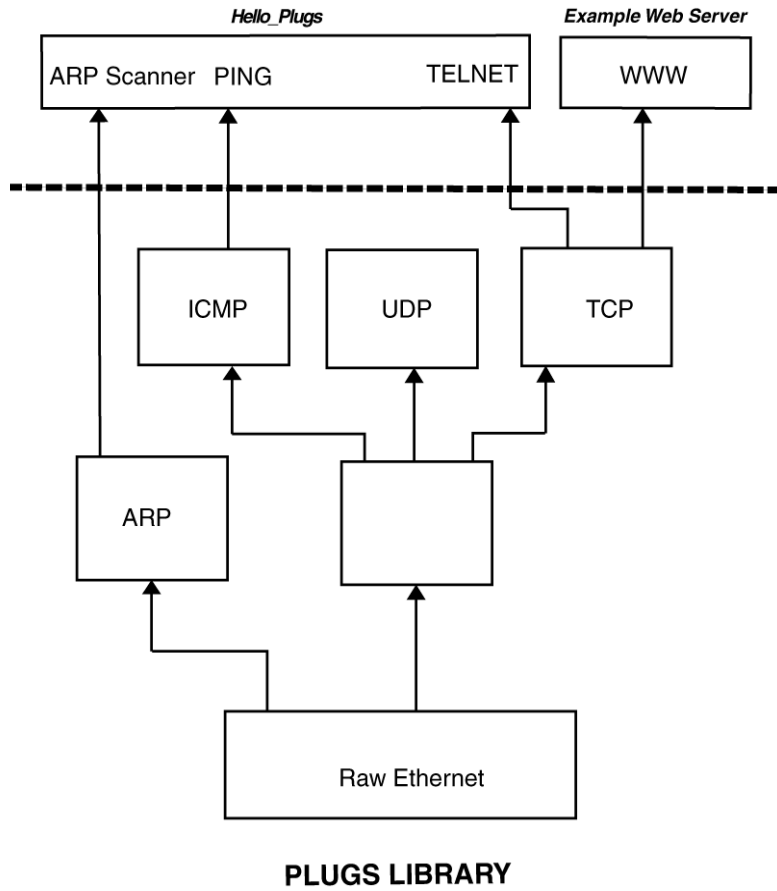
The Plugs Library requires your system to have a Timer peripheral named *timer 1*.

The customized software development kit for the CS8900A Ethernet adapter peripherals contains the Plugs Library and example applications. This library contains single-threaded routines that rely on polling.

Protocols Architecture

Figure 23 shows the relationships between the library-supported Nios EDK protocols.

Figure 23. Nios EDK Protocol Structure



Standards

The protocols supported by the Plugs Library adhere to the standards recommended by the RFCs at <http://www.ietf.org>.

The Nios EDK supports Ethernet and 802.3 packets. To send a 802.3 packet, the application will have to construct all fields explicitly. Higher level protocols will NOT support 802.3, and will use Ethernet instead. Nios EDK does NOT support trailer encapsulation as documented in RFC 893.

The library routines send and receive Ethernet packets to and from arbitrary 48-bit Ethernet media access control (MAC) addresses. Higher level protocols such as ICMP, UDP, and TCP use Ethernet transparently.

ARP (RFC 826)

Library routines are provided to query the LAN for the Ethernet address of a particular remote IP address, and to respond to queries for the local IP address. Other protocols like IP use ARP transparently.

IP (RFC 791)

Nios EDK encapsulates IP on Ethernet (RFC 894). Library routines are provided for sending and receiving IP packets to and from a user-defined 32-bit remote IP address. Higher level protocols like ICMP, UDP, and TCP use IP transparently.

- Nios EDK does not support IP packet fragmentation.
- Nios EDK supports IPv4.

ICMP (RFC 792)

The Plugs Library can respond to an ICMP echo request (ping). Library routines are provided to send and receive ICMP error messages.

UDP (RFC 768)

UDP is a low-level packet format built on top of IP. Library routines are provided to send and receive UDP packets to and from an arbitrary 32-bit remote IP address and 16-bit port number. Higher-level protocols like DNS use UDP transparently.

DNS (RFC 1034 & 1035)

Library routines are provided to transmit a DNS query for a host name to a specified name server. If the host name is found, the name server returns the associated IP address requested. The Nios EDK supports UDP encapsulation of DNS and does not support TCP encapsulation of DNS.

TCP (RFC 793)

TCP is a connection-oriented protocol built on top of IP. Library routines are provided to open a TCP connection to an arbitrary 32-bit IP address and 16-bit port. This protocol receives requests for incoming connections, accepts or denies requests for incoming connections, transmits and receives bytes on an established connection, and closes an established connection.

Build Options

The following build options are provided for modulating the features and the footprint of the Plugs Library.

PLUGS_DEBUG (Default Value = 1)

This build option may be set to 0, to disable all debug-printing features, or 1 or 2 which enables debug-printing for those plugs that are created with the `ne_plugs_flag_debug_rx` or `ne_plugs_flag_debug_tx` flags set. When set to zero, no printing code is linked to the plug.

PLUGS_PLUG_COUNT (Default Value = 6)

This build option sets the maximum number of plugs that you can create. The maximum number of plugs the library can handle is 32 plugs. The library itself uses 2 or 3 ports per adapter for managing ARP, pings, and DNS. Changing this option affects the amount of static storage used by the library.

PLUGS_ADAPTER_COUNT (Default Value = 2)

The Plugs Library can support multiple network adapters. This build option sets the maximum number of adapters that can be used. It affects the amount of static storage used by the library.

PLUGS_DNS (Default Value = 1)

The Plugs Library lets you establish connections to a remote network device using either its name or its IP address. If you use its name, the Plugs Library contacts a domain name server to translate it into an IP address. If your application does not need to establish outgoing connections (the application is a server only), or uses IP addresses only, then this build option can be set to zero to omit the code that implements name lookups.

PLUGS_PING (Default Value = 1)

In general, every network device should respond to an ICMP echo request message (ping). You can disable a ping response to save a small amount of code space by setting this build option to zero.

PLUGS_TCP (Default Value = 1)

If your application does not use TCP for any of its plugs, you can disable it and save a small amount of code space by setting this build option to zero.

Byte Ordering

Network-byte order is big endian. The Nios CPU byte order is little endian. Because of this, packet header numbers reside in memory in reverse order. This is often desirable for comparing the packet header numbers to other packet header numbers being sent over the network. The normal ordering for a particular CPU is called *host ordering*.

It is important to know if a particular integer in memory or a register is in host order or network order when using the Nios EDK Plugs Library.

Some parameters to routines in the Plugs Library are given in network order, and others are given in host order. To distinguish between network order and host order, the following data types are declared:

```
typedef unsigned char host_8
typedef unsigned short host_16
typedef unsigned long host_32

typedef unsigned char net_8
typedef unsigned short net_16
typedef unsigned long net_32
```

Data Structures

ns_plugs_network_setting

Structure:

```
typedef struct
{
    net_48 ethernet_address;
    short pad;
    net_32 ip_address;
    net_32 nameserver_ip_address;
} ns_plugs_network_settings;
```

Description: This structure is used to configure an adapter with all the necessary network information. It is passed to the Plugs Library routine `nr_plugs_initialize()` for each adapter.

Structure member:

<code>ethernet_address</code>	This is a 48-bit value in network-byte order. Every Ethernet card must have a unique 48-bit MAC address. (These addresses are managed by the IEEE. Information on obtaining a legal Ethernet MAC address can be found at www.ieee.org ; search for <i>OUI, Organizationally Unique Identifier</i>)
<code>pad</code>	This member is unused.
<code>ip_address</code>	This is a 32-bit IP address in network-byte order. It should be an unused IP address within the range of the LAN connection to the Nios-based device.
<code>nameserver_ip_address</code>	This is a 32-bit IP address in network-byte order. If your Nios-based device needs to establish connections with remote network devices using their DNS names (using the <code>remote_name</code> parameter of the Plugs Library <code>nr_plugs_connect()</code> or <code>nr_plugs_name_to_ip()</code> routines), then you must provide the name server's IP address for the Plugs Library to use.

ns_plugs_network_setting

`subnet_mask` This is a 32-bit value in network-byte order. This mask value is used to determine if a particular remote network device is on the same LAN as the Nios-based device. If any bits of the Nios-based device's IP address differ from any bits of the remote network devices's IP address, and the corresponding subnet mask bit is set, then the remote device is not on the LAN. The Plugs Library sends packets for remote devices that are not on the LAN to the local gateway.

`gateway_ip_address` This is a 32-bit value in network-byte order. If the Nios-based device is communicating with devices that are not on the LAN, it must send packets to the gateway. The gateway is then responsible for routing packets appropriately.

ns_plugs_persistent_network_settings

Structure:

```
typedef struct
{
    long settings_index; // 0..3
    ns_plugs_network_settings settings[4];
} ns_plugs_persistent_network_settings;
```

Description:

The example programs that use the Plugs Library make use of nonvolatile network settings stored in the Flash memory. The program `hello_plugs.c` lets you enter up to four sets of network settings, and use these setting interchangeably. The default location in the Flash memory on the Nios development board is 0x00106000. You can direct the Plugs Library routine `nr_plugs_initialize()` to use the nonvolatile network settings selected by the `settings_index` member by passing zero for the settings parameter.

Structure member:

`setting_index` An integer that ranges from 0 to 3. This index determines which of the 4 stored network settings to use.

`setting` An array of four elements of type `ns_plugs_network_settings`. Up to four complete network settings can be stored in the Flash memory; the one that is used is determined by the `settings_index` member.

Payload Descriptions

Each protocol treats a different part of the raw Ethernet packet as the payload. The payload is the part of the packet passed to the receive callback procedure. The callback procedure can access the payload and all encapsulating header information. Table 5 below describes which part of the packet is treated as the payload for each of the supported protocols.

Protocol	Payload Description	Payload Protocol Type	Maximum Payload Size (bytes)
Ethernet	Header portion of Ethernet packet followed by any other contents	<code>ns_ethernet_packet *</code>	1500
ARP	Header portion of ARP packet, which is the payload portion of the Ethernet packet	<code>ns_arp_packet *</code>	28
IP	Payload portion of the IP packet	<code>unsigned char *</code>	1024
ICMP	Header portion of the ICMP packet	<code>ns_icmp_packet *</code>	1024
UDP	Payload portion of the UDP packet	<code>unsigned char *</code>	1024
TCP	Sequential bytes from the stream	<code>unsigned char *</code>	512

Software Overview



Notes:



Plugs Library Routines

Table 6 lists and describes the Nios plugs library routines.

Table 6. Nios Plugs Library	
Routine	Description
nr_plugs_initialize	Initializes the plug library
nr_plugs_terminate	Terminates the plugs library
nr_plugs_set_mac_led	Controls the LED on the RJ-45 jack.
nr_plugs_create	Allocates a plug
typedef int (*nr_plugs_receive_callback_proc)	Application-provided callback routine to receive data
nr_plugs_destroy	Deallocates a plug
nr_plugs_connect	Associates a plug with a remote IP address and port on the network
nr_plugs_send	Sends a packet to the connected remote-network device
nr_plugs_send_to	Sends a packet to a specified IP address and port
nr_plugs_listen	Tells a plug to wait for an incoming TCP connection request
typedef int (*nr_plugs_listen_callback_proc)	Application-provided callback routine to accept or reject a TCP connection request
nr_plugs_ip_to_ethernet	Converts an IP address to an Ethernet address
nr_plugs_name_to_ip	Uses name server to convert a remote-network device name to an IP address
nr_plugs_idle	Polls all network adapters for incoming packets and dispatches the packets to the receive callback routines
nr_plugs_print_ethernet_packet	Prints an Ethernet packet report
nr_n2h16	Translates a network-short integer to a short integer
nr_h2n16	Translates a short integer to a network-short integer
nr_n2h32	Translates a network-long integer to a long integer
nr_h2n32	Translates a long integer to a network-long integer

nr_plugs_initialize

Syntax:	<pre>int nr_plugs_initialize (long flags, ns_plugs_network_settings *network_settings, void *adapter_address, ns_plugs_adapter_description *adapter_description);</pre>								
Description:	This routine can either initialize the plugs library, or add an additional adapter to the plugs library. Each adapter is completely distinct from each other. If you are using more than one adapter, each adapter should be added using this routine before calling any other routine. Each adapter has its own network settings (IP address, netmask, etc.) Only the first adapter added can perform DNS lookups.								
Parameters:	<table><tr><td><code>flags</code></td><td>This can be 0 or <code>ne_plugs_flag_add_adapter</code>. If it is <code>ne_plugs_flag_add_adapter</code>, then only the adapter is initialized and added to the plugs library list of available adapters. The first adapter has an index number of zero, the second adapter has an index number of one, and so forth. Some other routines use this index number to specify a particular adapter.</td></tr><tr><td><code>network_setting</code></td><td>A pointer to a structure of type <code>ns_plugs_network_settings</code> to configure this adapter. If the <code>network_setting</code> is NULL, the network settings will be retrieved from the Flash memory.</td></tr><tr><td><code>adapter_address</code></td><td>The hardware address of the adapter peripheral device, if applicable.</td></tr><tr><td><code>adapter_description</code></td><td>A pointer to a structure of type <code>ns_plugs_adapter_description</code>, that determines the low-level driver routines for this adapter.</td></tr></table>	<code>flags</code>	This can be 0 or <code>ne_plugs_flag_add_adapter</code> . If it is <code>ne_plugs_flag_add_adapter</code> , then only the adapter is initialized and added to the plugs library list of available adapters. The first adapter has an index number of zero, the second adapter has an index number of one, and so forth. Some other routines use this index number to specify a particular adapter.	<code>network_setting</code>	A pointer to a structure of type <code>ns_plugs_network_settings</code> to configure this adapter. If the <code>network_setting</code> is NULL, the network settings will be retrieved from the Flash memory.	<code>adapter_address</code>	The hardware address of the adapter peripheral device, if applicable.	<code>adapter_description</code>	A pointer to a structure of type <code>ns_plugs_adapter_description</code> , that determines the low-level driver routines for this adapter.
<code>flags</code>	This can be 0 or <code>ne_plugs_flag_add_adapter</code> . If it is <code>ne_plugs_flag_add_adapter</code> , then only the adapter is initialized and added to the plugs library list of available adapters. The first adapter has an index number of zero, the second adapter has an index number of one, and so forth. Some other routines use this index number to specify a particular adapter.								
<code>network_setting</code>	A pointer to a structure of type <code>ns_plugs_network_settings</code> to configure this adapter. If the <code>network_setting</code> is NULL, the network settings will be retrieved from the Flash memory.								
<code>adapter_address</code>	The hardware address of the adapter peripheral device, if applicable.								
<code>adapter_description</code>	A pointer to a structure of type <code>ns_plugs_adapter_description</code> , that determines the low-level driver routines for this adapter.								
Return Value:	The return value will be zero for success or a negative value for failure.								
Include:	<code>plugs.h</code>								

nr_plugs_terminate

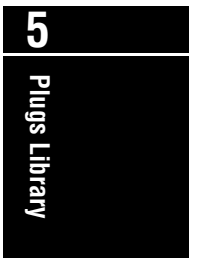
Syntax: `nr_plug_terminate(void)`
(
void
);

Description: Call this routine when you are done using the plugs library. If you need to reinitialize the plugs library with different network settings, call this routine first before reinitializing.

Parameters: None

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`



nr_plugs_set_mac_led

Syntax:

```
int nr_plug_set_mac_led
(
    int adapter_index,
    int led_onoff
);
```

Description: This routine controls the LED present on most Ethernet jacks. If a particular adapter does not have a LED on the Ethernet jack, this routine does nothing. The CS8900A LED's default behavior is to be on if it is connected to a network, or off if it is not connected to a network.

Parameters:

`adapter_index` The index number of the adapter to control.

`on_off` This parameter can have one of three values. Zero turns the LED off, one turns the LED on and negative one returns the LED to its default behavior as specified for the particular adapter.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_create

Syntax:

```
int nr_plugs_create
(
    int *plugs_handle_out,
    int protocol,
    host_16 port,
    nr_plugs_receive_callback_proc callback,
    void *callback_context,
    long flags
);
```

Description: This routine creates a plug. A plug is a logical endpoint for network communications. A plug is in some ways similar to a traditional UNIX socket. When you create a plug, you specify its protocol, and if applicable to the particular protocol, its port number. You must also specify a callback procedure. The callback procedure is called whenever data arrives over the network for this plug. A plug is associated with exactly one adapter.

Parameters:

`plugs_handle_out` This parameter is a pointer to an integer that contains a reference to the new plug. The new plug reference is used to specify this particular plug to other plugs library routines.

`protocol` This parameter specifies which network protocol the plug can receive and transmit. The possible values for this parameter are as follows:

```
ne_plugs_ethernet
ne_plugs_arp
ne_plugs_ip
ne_plugs_icmp
ne_plugs_udp
ne_plugs_tcp
```

`port` If the plug's protocol is UDP or TCP, then the plug must be associated with a particular port number. If this parameter is zero, an unused port number will be chosen for you.

Plugs Library Routines

<code>callback</code>	When data arrives for this plug, your callback routine is called with the data. The parameters of the callback routine are documented under <code>nr_plugs_receive_callback_proc</code> .
<code>callback _context</code>	This parameter is passed unmodified to your callback routine. It can be used to carry state information to your callback routine.
<code>flags</code>	<p>Multiple flags should be grouped together using the OR instruction with the vertical-bar operator. If you are using more than one adapter, an integer between 0 and 15 can be added to the value for the flags parameter. This indicates the index number of the adapter associated with the plug. If you are using only one adapter, then its index is always zero. Flags can be any combination of the following:</p> <p><code>ne_plugs_flag_ethernet_broadcast</code></p> <p>If the plug is Ethernet protocol, this flag transmits outgoing packets as broadcast messages.</p> <p><code>ne_plugs_flag_ethernet_all</code></p> <p>If the plug is Ethernet protocol, this plug receives all packets, regardless of whether their Ethernet address matches this adapter's address.</p> <p><code>ne_plugs_flag_debug_rx</code></p> <p>This flag prints debugging information for each packet received by this plug. The debugging information is printed using <code>printf()</code>, and appears on the same serial port as other <code>printf()</code> output.</p> <p><code>ne_plugs_flag_debug_tx</code></p> <p>This flag prints debugging information for each packet transmitted by this plug. The debugging information is printed using <code>printf()</code>, and appears on the same serial port as other <code>printf()</code> output.</p>
Return Value:	The return value will be zero for success or a negative value for failure.
Include:	<code>plugs.h</code>

typedef int (*nr_plugs_receive_callback_proc)

Syntax:

```
typedef int (*nr_plug_receive_callback_proc)
(
    int plug_handle
    void *context,
    ns_plugs_packet *p,
    void *payload,
    int payload_length
);
```

Description: This is a routine you provide when you create a plug. The plugs library will call this routine whenever a packet arrives for the plug. The plug receives the packet's payload and length and also a pointer to a list containing the packet header for each network protocol layer used by the incoming packet.

Parameters:

`plug_handle` A reference to the plug that is receiving a packet.

`context` The value passed for the parameter named `callback_context` in `nr_plugs_create()`.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

Plugs Library Routines

- p A pointer to an array of entries. These entries can be indexed by the various network protocol enumeration constants (the same constants used to specify the network protocol in `nr_plugs_create()`). Each entry consists of two fields, as follows:

```
typedef struct
{
    void *header;
    int length;
} ns_plugs_packet;
```

The header field is a pointer to the first byte of the header for that protocol layer. If the header pointer is zero, then the packet does not conform to the indexed protocol. The length is the combined length of the header and payload for that protocol layer.

For example, suppose you created a plug that was using the TCP protocol. When your callback routine is called, you could examine the enclosing Ethernet packet header by reading at location

`p[ne_plugs_ethernet].header`. You could also examine the enclosing IP packet header by reading at location `p[ne_plugs_ip].header`. However, the values for `p[ne_plugs_arp].header`, `p[ne_plugs_icmp].header`, and `p[ne_plugs_udp].header` will all be zero, because these protocols are not a part of a TCP packet.

`payload` A pointer to the meaningful payload portion of the packet to be received by this plug. In the case of TCP and UDP, the payload contains the bytes transmitted.

`payload_length` The length of the payload. In the case of TCP and UDP protocol, this is the number of bytes transmitted.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_destroy

Syntax:

```
int nr_plugs_destroy
(
    int plug_handle,
);
```

Description: Deallocates a plug. When you no longer need a plug, call this routine to deallocate any resources associated with the discarded plug.

Parameters:

`plug_handle` A reference to the plug you are eliminating.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_connect

Syntax:

```
int nr_plug_connect
(
    int plug_handle,
    char *remote_name,
    host_32 remote_ip_address,
    host_16 remote_port
);
```

Description:

This routine associates a plug with a particular remote IP address and port on the network. If the plug is using TCP, then this routine will perform the necessary network transaction to establish a connection with the remote host. If the connection cannot be established, an error is returned. If the plug is not using TCP, then the remote address and port are stored in the plug's state as the default destination for packets.

This routine can be used to allow packets to be received from any remote-network device (only if the plug does not use TCP), by connecting to IP address -1, port -1. This routine can be useful when providing a UDP service.

If the plug uses TCP, this routine closes an existing TCP connection. To close a connection on a TCP plug, call this routine with a remote IP address of 0 and a remote port of 0.

Parameters:

`plug_handle` A reference to the plug you are eliminating.

`remote_name` A pointer to a string containing the name of a remote-network device (for example, <http://www.altera.com>). The routine will attempt to resolve the name to an IP address by using the DNS server associated with the first adapter installed. This parameter may be zero, in which case, the `remote_ip_address` parameter is used instead.

`remote_ip_address` A 32-bit value that is an IP address of a remote-network device. This parameter is ignored if a remote name is provided for the `remote_name` parameter.

`remote_port` If the port uses UDP or TCP, this parameter specifies the port number of the connection on the remote-network device.

Return Value: The return value will be zero for success or a negative value for failure.

Include:

plugs.h



Transmission to another plug on the same Nios system will not succeed and loopback is not supported.

nr_plugs_send

Syntax:

```
int nr_plugs_send
(
    int plug_handle,
    void *data,
    int data_length,
    long flags
);
```

Description: This routine transmits a packet of data using a particular plug. Before you call this routine, you must call `nr_plugs_connect()` to associate the plug with a particular remote-network device.

Parameters:

`plug_handle` A reference to a plug.

`data` The payload to send.

`data_length` The number of bytes in the payload.

`flags` This parameter augments the flags specified by `nr_plugs_create()`. Typically this is used to add `nr_flag_debug_tx` to one particular transmission.

Return Value: The return result will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_send_to

Syntax:

```
int nr_plugs_send_to
(
    int plug_handle,
    void *data,
    int data_length,
    long flags,
    net_32 ip_address, //|net order
    net_16 port        //|net order
);
```

Description: This routine is identical to `nr_plugs_send()`, with the addition of a destination IP address and port. When a plug uses UDP, you can easily send a packet to any destination using this routine. Do not use this routine on a plug using TCP.

Parameters:

`plug_handle` A reference to a plug.

`data` The payload to send.

`data_length` The number of bytes in the payload.

`flags` This parameter augments the flags specified by `nr_plugs_create()`. Typically this routine is used to add `nr_plugs_flag_debug_tx` to a particular transmission.

`ip_address` The IP address of a remote-network device. The packet is transmitted to this remote-network device.

`port` If the plug uses UDP, the packet transmits to this port on the remote-network device.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

int nr_plugs_listen

Syntax:

```
int nr_plugs_listen
(
    int plug_handle,
    nr_plugs_listen_callback_proc callback,
    void *callback_context
);
```

Description:

Only call this routine if the plug uses TCP. This routine tells the plug to wait for an incoming TCP connection request. If the plug is already connected to a remote-network device, the connection is closed immediately when this routine is called. When a connection request is received, the callback routine you provide is called and can accept or reject the connection request. If there is an existing TCP connection established on this plug, the connection is closed and the plug begins to wait for an incoming TCP connection request. You may create multiple TCP plugs for the same port. When a connection request is received, each of the plugs' callback routines will be called and the first plug to accept the connection will be connected.

Parameters:

`plug_handle` A reference to the plug.

`callback` A routine you provide to accept or decline an incoming TCP connection request. You may pass zero for this parameter and any incoming TCP connection request will be accepted.

`callback_context` This parameter is passed unmodified to your callback routine. It can be used to carry state information to your routine.

Return Value:

The return value will be zero for success or a negative value for failure.

Include:

plugs.h

typedef int (*nr_plugs_listen_callback_proc)

Syntax:

```
typedef int (*nr_plugs_listen_callback_proc)
(
    int plug_handle,
    void *context,
    host_32 remote_ip_address,
    host_16 remote_port
);
```

Description: This is a routine you provide when you allow a TCP plug to accept connections using the `nr_plugs_listen()` routine. This routine can accept or decline the connection by returning a zero (meaning no error occurred — accept the connection) or a negative value (meaning an error did occur — do not accept the connection).

Parameters:

`plug_handle` A reference to a plug.

`context` The value passed for the parameter named `callback_context` in `nr_plugs_listen()`.

`remote_ip_address` The IP address of the remote-network device attempting to connect to this plug.

`remote_port` The port on the remote-network device attempting to connect to this plug.

Return Value: Your routine should return zero to accept the incoming connection request, or a negative value to reject the connection request.

Include: `plugs.h`

nr_plugs_ip_to_ethernet

Syntax:

```
int nr_plugs_ip_to_ethernet
(
    int adapter_index,
    net_32 ip_address,
    net_48 *ethernet_address_out,
    long flags
);
```

Description: When this routine is given an IP address, it discovers which is the correct Ethernet address for the packets being sent. When the IP address is on the LAN, the Ethernet address is the address for the network device; otherwise, the Ethernet address is the address for the local gateway.

Parameters:

`adapter_index` The index number for the adapter being used.

`ip_address` An IP address of a remote-network device.

`ethernet_address_out` A pointer to a 48-bit Ethernet address. This routine will fill out this structure with the discovered Ethernet address.

`flags` This flag can be 0 or `nr_plugs_flag_debug_tx`. If the flag is `nr_plugs_flag_debug_tx` and the operation fails, then a message is printed.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_name_to_ip

Syntax:

```
int nr_plugs_name_to_ip
(
    char *host_name,
    net_32 *host_ip_address_out
);
```

Description: When this routine is given the name of a remote-network device it queries the name server to find out the IP address. This routine uses adapter number 0.

Parameters:

<code>host_name</code>	A pointer to a string containing the name of a network device.
<code>host_ip_address_out</code>	A pointer to a 32-bit IP address. This routine will fill out this value with the discovered IP address.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`

nr_plugs_idle

Syntax:

```
int nr_plugs_idle(void)
{
    void
};
```

Description: This routine must be called frequently in your program's inner loop, or from a timer interrupt routine. It polls the hardware device for incoming packets, and dispatches them via each plug's callback routine.

Parameters: None

Return Value: The return value will be negative if any errors occur.

Include: plugs.h

void nr_plugs_print_ethernet_packet

Syntax:

```
void nr_plugs_print_ethernet_packet(void)
(
  ns_plugs_ethernet_packet *p,
  int length,
  char *title
);
```

Description: This routine prints an Ethernet packet in a friendly, human-readable format.

Parameters:

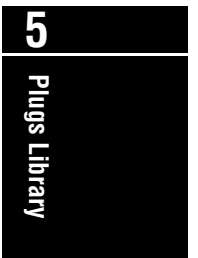
`p` A pointer to an Ethernet packet.

`length` The length of the Ethernet packet.

`title` A short string printed at the beginning of each line.

Return Value: The return value will be zero for success or a negative value for failure.

Include: `plugs.h`



The following routines and macros are for your general use, and are required when you are translating between host-byte ordering and network-byte ordering for Nios EDK-network programming.



Network-byte ordering is always big-endian and Nios host-byte ordering is little-endian.

nr_n2h16

Syntax: `nr_n2h16(net_16 value)`
Parameters: A network-short integer
Description: Translates a network-short integer to a short integer.
Equivalent Macro: `nm_n2h16`

nr_h2n16

Syntax: `nr_h2n16(host_16 value)`
Parameters: a short integer
Description: Translates a short integer to a network-short integer.
Equivalent Macro: `nm_h2n16(host_16)`

nr_n2h32

Syntax: `nr_n2h32(net_32 value)`
Parameters: A network-long integer
Description: Translates a network-long integer to a long integer.
Equivalent Macro: `nm_n2h32(host_32)`

nr_h2n32

Syntax: `nr_h2n32(host_32 value)`
Parameters: A long integer
Description: Translates a long integer to a network-long integer.
Equivalent Macro: `nm_h2n32(host_32)`



The Excalibur Development Kit featuring the Nios embedded processor and the Nios Ethernet Development Kit uses the following terms:

Table 7. Nios Acronym List

Acronym	Meaning
API	Application program interface
APP	Application programming platform
ARP	Address resolution protocol for ethernet
ASSP	Application specific standard products
CPLD	Complex programmable logic device
BOM	Bill of Materials
CTS	Clear to send
CWP	Current window pointer
DIP	Dual in-line package
DLL	Delay-locked loop
DNS	Domain Name System
DSP	Digital signal processing
E	Exception condition
EDN	Emergency data network
EDK	Ethernet Development Kit
EVB	Evaluation board
FE	Framing error
FTP	File transfer protocol
GDB	GNU debugger
GPP	General purpose processor
HDK	Hardware development kit
HTTP	Hyper text transmission protocol
iBRK	Interrupt-enable break detect
ICMP	Internet control message protocol
IDC	Insulation displacement connector
IDE	Integrated drive electronics
iE	Interrupt-enable exception condition
iFE	Interrupt-enable framing error
IGMP	Internet group management protocol

Table 7. Nios Acronym List	
Acronym	Meaning
IP	Internet protocol
IPR	Interrupt priority
iROE	Interrupt-enable receiver-override error
iRRDY	Interrupt-enable read ready
ISA	Industry-standard application
iTMT	Interrupt-enable transmitter shift register empty
iTO	Interrupt-enable time-out
iTOE	Interrupt-enable transmitter override error
iTRDY	Interrupt-enable transmission ready
LAN	Local area network
LDAP	Lightweight directory access protocol
LSB	Least significant bit
MAC	Media access controller
MIPS	Millions of instructions per second
MISO	Master in slave out
MOSI	Master out slave in
MPLS	Multi protocol label switching
MSB	Most significant bit
PC	Program counter
PE	Parity error
PCI	Peripheral component interconnect
PHY/MAC	Physical interface/ media access control
PIO	Parallel input/output module
PMC	PCI (peripheral component interconnect) mezzanine card
PPP	Point-to-point protocol
PTF	Peripheral template file
RFC	Request for comment
ROE	Receiver-overflow error
RRDY	Reading the read only
RTS	Request to send
RXD	Receive
SDK	Software development kit
SOPC	system-on-a-programmable-chip
TCP	Transmission control protocol
UART	Universal asynchronous receiver transmission
UDP	User datagram protocol



Notes:

Nios Terminology



Notes: