



# Plugs Ethernet Library

---

## Reference Manual



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

Document Version: 1.2  
Document Date: July 2003

Copyright © 2003 Altera Corporation. All rights reserved. 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. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work 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.



I.S. EN ISO 9001

This manual provides a software overview introducing you to the C language library supporting SOPC Builder Ethernet components. It includes the following items:

- A description of the supported protocols and general structure of the provided functions and data structures.
- A description of the Plugs Ethernet Library, which is a collection of software subroutines for SOPC Builder Ethernet components.



For the most current version of this manual, see <http://www.altera.com/literature/lit-nio.html>

Table 1 shows the manual revision history.

<b>Table 1. Manual Revision History</b>	
<b>Date</b>	<b>Description</b>
July 2003	Minor protocol clarification
May 2003	Minor revisions
January 2003	First publication

## How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

## 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 technical support on this product, go to <http://www.altera.com/mysupport>. For additional information about Altera products, consult the sources shown in [Table 2](#).

**Table 2. How to Contact Altera**

Information Type	USA & Canada	All Other Locations
Technical support	<a href="http://www.altera.com/mysupport/">http://www.altera.com/mysupport/</a>	<a href="http://www.altera.com/mysupport/">http://www.altera.com/mysupport/</a>
	(800) 800-EPLD (3753) (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)
Product literature	<a href="http://www.altera.com">http://www.altera.com</a>	<a href="http://www.altera.com">http://www.altera.com</a>
Altera literature services	<a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)	<a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
FTP site	<a href="ftp.altera.com">ftp.altera.com</a>	<a href="ftp.altera.com">ftp.altera.com</a>

**Note:**



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

## Documentation Feedback

Altera values your feedback. If you would like to provide feedback on this document—e.g., clarification requests, inaccuracies, or inconsistencies—send e-mail to [nios\\_docs@altera.com](mailto:nios_docs@altera.com).

## Typographic Conventions

This document uses the typographic conventions shown in [Table 3](#).

<i>Table 3. Conventions</i>	
Visual Cue	Meaning
<b>Bold Type with Initial Capital Letters</b>	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <b>Save As</b> dialog box.
<b>bold type</b>	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: <b>f<sub>MAX</sub></b> , <b>lqdesigns</b> directory, <b>d:</b> drive, <b>chiptrip.gdf</b> file.
<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>t<sub>PIA</sub></i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pof 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 on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix n, e.g., resetn.  Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\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.

## Acronyms

ARP	address resolution protocol
ICMP	internet control message protocol
IETF	Internet Engineering Task Force
IP	internet protocol
PLD	programmable logic device
RFC	request for comment
SDK	software development kit
TCP	transmission control protocol
UDP	user datagram protocol



---

<b>About this Manual</b> .....	iii
How to Find Information .....	iii
How to Contact Altera .....	iv
Documentation Feedback .....	iv
Typographic Conventions .....	v
Acronyms .....	vi
<b>Software Overview</b> .....	9
Introduction .....	9
Software Description .....	9
System Requirements .....	9
Protocols Supported .....	10
Library Features .....	10
Protocol Architecture .....	10
Supported Protocols .....	11
ARP (RFC 826) .....	11
IP (RFC 791) .....	11
ICMP (RFC 792) .....	11
UDP (RFC 768) .....	11
DNS (RFC 1034 & 1035) .....	12
TCP (RFC 793) .....	12
Build Options .....	12
PLUGS_DEBUG (Default Value = 1) .....	12
PLUGS_PLUG_COUNT (Default Value = 6) .....	12
PLUGS_ADAPTER_COUNT (Default Value = 2) .....	13
PLUGS_DNS (Default Value = 1) .....	13
PLUGS_PING (Default Value = 1) .....	13
PLUGS_TCP (Default Value = 1) .....	13
Byte Order .....	13
Data Structures .....	15
Payload Descriptions .....	17
<b>Subroutines</b> .....	19
nr_plugs_initialize .....	20
nr_plugs_terminate .....	21
nr_plugs_set_mac_led .....	22
nr_plugs_create .....	23
typedef int (*nr_plugs_receive_callback_proc) .....	25

---

nr_plugs_destroy .....	27
nr_plugs_connect .....	28
nr_plugs_send .....	29
nr_plugs_send_to .....	30
int nr_plugs_listen .....	31
typedef int (*nr_plugs_listen_callback_proc) .....	32
nr_plugs_ip_to_ethernet .....	33
nr_plugs_name_to_ip .....	34
nr_plugs_idle .....	35
void nr_plugs_print_ethernet_packet .....	36
nr_n2h16 .....	37
nr_h2n16 .....	37
nr_n2h32 .....	37
nr_h2n32 .....	37
<b>Index</b> .....	<b>39</b>



## Introduction

The Plugs Ethernet Library is a library of software routines for managing network connections with the Nios processor. The plugs subroutines are similar to standard UNIX “socket” routines that implement protocols for opening network connections, and transmitting and receiving data packets. Using the Plugs Ethernet Library, a software developer can easily create network-enabled Nios processor systems. The Plugs Ethernet Library abstracts the physical network connection from the software developer, making plugs-based embedded software easily portable to other Nios processor systems.

When the SOPC Builder development tool generates a system that includes an Ethernet peripheral, SOPC Builder includes the plugs routines in the software development kit (SDK) for the system. SOPC Builder customizes the Plugs Ethernet Library with low-level drivers to match the network hardware used in the custom system. With appropriate knowledge of the underlying hardware, you can port the Plugs Ethernet Library to support any Ethernet MAC hardware, including on-chip MAC implementations.

## Software Description

The Plugs Ethernet Library allows your software to use network protocols for transmitting and receiving data. The information in this chapter is applicable to PHY/MAC devices.

### System Requirements

The Plugs Ethernet Library has the following system requirements.

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



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

## Protocols Supported

The Plugs Ethernet Library supports the following protocols.

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

## Library Features

The Plugs Ethernet Library has the following features.

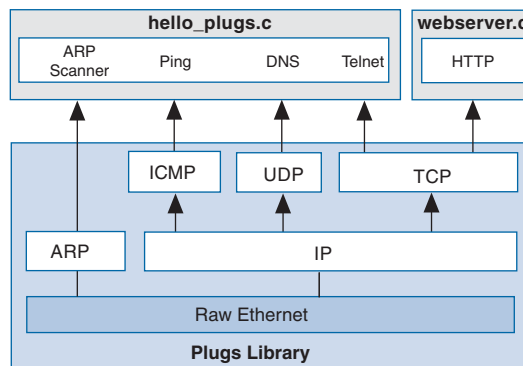
- Accesses low-level packets
- Accesses high-level packet payloads
- Conforms to the Internet Engineering Task Force (IETF) RFCs
- Allows you to open connections and send data with only a few lines of code
- Is similar to the UNIX standard socket subroutines
- Can set each plug to print debug information for either transmit or receive data

The customized SDK for Ethernet-adapter peripherals contains the Plugs Ethernet Library and example applications. This library can be used in either interrupt-driven or polled applications.

## Protocol Architecture

Figure 1 shows the relationship between the Plugs Ethernet Library-supported Ethernet protocols.

**Figure 1. Nios Ethernet Protocol Structure**



## Supported Protocols

The Plugs Ethernet Library supports protocols based upon standards recommended by the RFCs at <http://www.ietf.org>. The Plugs Ethernet Library is robust and versatile, although rigorous adherence to the recommended RFC standards is not guaranteed.

The Plugs Ethernet Library supports Ethernet and 802.3 packets. To send an 802.3 packet, the application has to construct all fields explicitly. Higher-level protocols do not support 802.3 and use Ethernet instead. The Plugs Ethernet Library does not support trailer encapsulation as documented in RFC 893.

The library subroutines send and receive Ethernet packets to and from arbitrary 48-bit MAC addresses. Higher-level protocols such as ICMP, UDP, and TCP use Ethernet transparently.

### ARP (RFC 826)

The Plugs Ethernet Library provides subroutines 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)

The Nios processor encapsulates IP on Ethernet (RFC 894). Nios Ethernet provides library subroutines 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 Ethernet does not support IP packet fragmentation.
- Nios Ethernet supports IPv4.

### ICMP (RFC 792)

The Plugs Ethernet Library can respond to an ICMP echo request (ping). The library includes subroutines to send and receive ICMP error messages.

### UDP (RFC 768)

UDP is a low-level packet format built on top of IP. The Nios processor provides library subroutines 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)

The Nios processor provides library subroutines to transmit a DNS query for a host name to a specified name server. If the query finds the host name, the name server returns the associated IP address requested. The Nios processor 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. The library provides subroutines 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.



The TCP implementation in the Plugs Ethernet Library is not guaranteed to be robust in all circumstances, unless your application implements retry- and window-management.

## Build Options

The Plugs Ethernet Library provides the following build options for modulating its features and footprint.

### PLUGS\_DEBUG (Default Value = 1)

You must set this build option to 0 to disable all debug printing features, or set it to 1 or 2 to enable debug-printing for plugs that are created with the `ne_plugs_flag_debug_rx` or `ne_plugs_flag_debug_tx` flags set. When set to 0, 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 library can handle a maximum of 32 plugs. The library itself uses two or three ports per adapter for managing ARP, pings, and DNS. Changing this option affects the amount of static storage used by the library.



When using a stacked reference design (two adapters), it is necessary to increase the default value number. The recommended value is 10.

## PLUGS\_ADAPTER\_COUNT (Default Value = 2)

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

## PLUGS\_DNS (Default Value = 1)

The Plugs Ethernet 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 Ethernet Library contacts a domain name server to translate it into an IP address. If your application does not need to establish outgoing connections (e.g., the application is a server only) or only uses IP addresses, then you can set this build option to 0 to omit the code that implements name lookups.

## PLUGS\_PING (Default Value = 1)

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 0.

## 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 Order

Network byte order is big endian. The Nios CPU byte order is little endian. Therefore, packet header numbers reside in memory in reverse order. This order 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 whether a particular integer in memory or a register is in host order or network order when using the Plugs Ethernet Library.

Some parameters to subroutines in the Plugs Ethernet 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

The following sections describe the library data structures.

### ns\_plugs\_network\_setting (Part 1 of 2)

**Structure:**

```
typedef struct
{
    net_48 ethernet_address;
    short flags;
    net_32 ip_address;
    net_32 nameserver_ip_address;
    net_32 subnet_mask;
    net_32 gateway_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 Ethernet Library subroutine `nr_plugs_initialize()` for each adapter.

**Structure member:**

<code>ethernet_address</code>	This member 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 <a href="http://www.ieee.org">www.ieee.org</a> ; search for <i>OUI, Organizationally Unique Identifier</i> )
<code>flags</code>	This field can be zero or can contain the single flag <code>ne_plugs_flag_dhcp</code> , which causes the Plugs Ethernet Library to attempt to use a DHCP server to provide its network settings. If DHCP is used, only the Ethernet address must be included during initialization.
<code>ip_address</code>	This member 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 member is a 32-bit IP address in network-byte order. If the 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 Ethernet Library <code>nr_plugs_connect()</code> or <code>nr_plugs_name_to_ip()</code> subroutines), then provide the name server's IP address for the Plugs Ethernet Library to use.

## ns\_plugs\_network\_setting (Part 2 of 2)

<code>subnet_mask</code>	This member is a 32-bit value in network-byte order. This mask value determines if a particular remote network device is on the same LAN as the Nios-based device. If any of the Nios-based device's IP address bits differ from any of the remote network device's IP address bits, and the corresponding subnet mask bit is set, then the remote device is not on the LAN. The Plugs Ethernet Library sends packets for remote devices that are not on the LAN to the local gateway.
<code>gateway_ip_address</code>	This member is a 32-bit value in network-byte order. If the Nios-based device communicates with devices that are not on the LAN, it must send packets to the gateway. The gateway is then responsible for routing packets appropriately.
<code>ethernet_address</code>	This member 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 <a href="http://www.ieee.org">www.ieee.org</a> ; search for <i>OUI, Organizationally Unique Identifier</i> )



## 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 Ethernet 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 Ethernet Library subroutine `nr_plugs_initialize()` to use the nonvolatile network settings selected by the `settings_index` member by passing zero for the `settings` parameter.

**Structure member:**

<code>setting_index</code>	This member is an integer that ranges from 0 to 3. This index determines which of the four stored network settings to use.
<code>setting</code>	This member is an array of four elements of type <code>ns_plugs_network_settings</code> . You can store up to four complete network settings in the flash memory; the <code>settings_index</code> member determines which one is used.

## 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 4](#) below describes which part of the packet is treated as the payload for each of the supported protocols.

**Table 4. Plugs Ethernet Library Protocol Payload Descriptions**

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>	1,500
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>	1,024
ICMP	Header portion of the ICMP packet	<code>ns_icmp_packet *</code>	1,024
UDP	Payload portion of the UDP packet	<code>unsigned char *</code>	1,024
TCP	Sequential bytes from the stream	<code>unsigned char *</code>	512



Table 5 lists and describes the Nios Plugs Ethernet Library subroutines. These subroutines are described in the following sections.

<b>Table 5. Nios Plugs Ethernet Library Subroutines</b>	
<b>Subroutine</b>	<b>Description</b>
nr_plugs_initialize	Initializes the Plugs Ethernet Library.
nr_plugs_terminate	Terminates the Plugs Ethernet 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 subroutine 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 subroutine 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 subroutines.
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:**

```
int nr_plugs_initialize
(
    long flags,
    ns_plugs_network_settings *network_settings,
    void *adapter_base_address,
    int adapter_irq,
    ns_plugs_adapter_description *adapter_description
);
```

**Description:** This subroutine can either initialize the Plugs Ethernet Library or add an additional adapter to the Plugs Ethernet Library. Each adapter is completely distinct from each other. If you are using more than one adapter, each adapter should be added using this subroutine before calling any other subroutine. Each adapter has its own network settings (IP address, netmask, etc.). Only the first adapter added can perform DNS lookups.

**Parameters:**

`flags` This parameter can be 0 or any combination of `ne_plugs_flag_add_adapter` and `ne_plugs_flag_dhcp`. If you set this parameter to `ne_plugs_flag_add_adapter`, then the device only initializes this adapter and adds it to the Plugs Ethernet 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 subroutines use this index number to specify a particular adapter. You can force `nr_plugs_initialize` to use DHCP to discover all of its network settings dynamically (except the Ethernet address) by setting the flag `ne_plugs_flag_dhcp`.

`network_setting` This parameter is a pointer to a structure of type `ns_plugs_network_settings` to configure this adapter. If you set the `network_setting` to null, the device retrieves the network settings from the flash memory.

`adapter_base_address` The hardware address of the adapter peripheral device, if applicable.

`adapter_irq` The interrupt number of the ethernet hardware device. If this parameter is set to zero, interrupts are not enabled and the adapter is not instructed to enter interrupt mode.

`adapter_description` A pointer to a structure of type `ns_plugs_adapter_description`, that determines the low-level driver subroutines for this adapter.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** `plugs.h`

## nr\_plugs\_terminate

**Syntax:**

```
nr_plug_terminate(void)
(
    void
);
```

**Description:** Call this subroutine when you are done using the Plugs Ethernet Library. If you need to reinitialize the Plugs Ethernet Library with different network settings, call this subroutine first before reinitializing.

**Parameters:** None

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** `plugs.h`

## nr\_plugs\_set\_mac\_led

**Syntax:**

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

**Description:** This subroutine controls the LED present on most Ethernet jacks. If a particular adapter does not have a LED on the Ethernet jack, this subroutine does nothing. By default, the LED is on if it is connected to a network and 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. 0 turns the LED off, 1 turns the LED on and -1 returns the LED to its default behavior as specified for the particular adapter.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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 subroutine creates a plug (a logical endpoint for network communications). A plug is in some ways similar to a traditional UNIX socket. When you create a plug, you must 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:

<code>plugs_handle_out</code>	This parameter is a pointer to an integer that contains a reference to the new plug. The new plug reference specifies this particular plug to other Plugs Ethernet Library subroutines.
<code>protocol</code>	This parameter specifies which network protocol the plug can receive and transmit. The possible values for this parameter are: <pre>ne_plugs_ethernet ne_plugs_arp ne_plugs_ip ne_plugs_icmp ne_plugs_udp ne_plugs_tcp</pre>
<code>port</code>	If the plug's protocol is UDP or TCP, then the plug must be associated with a particular port number. If this parameter is 0, the software will select an unused port number.
<code>callback</code>	When data arrives for this plug, your callback routine is called with the data. The parameters of the callback subroutine are documented under <code>nr_plugs_receive_callback_proc</code> .
<code>callback_context</code>	The device passes this parameter unmodified to your callback subroutine. It can carry state information to your callback subroutine.

flags

Multiple flags should be grouped together using the `OR` instruction with the vertical-bar operator. If you are using more than one adapter, you can add an integer between 0 and 15 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 0. Flags can be any combination of the following:

`ne_plugs_flag_ethernet_broadcast`: If the plug is an Ethernet protocol, this flag transmits outgoing packets as broadcast messages.

`ne_plugs_flag_ethernet_all`: If the plug is an Ethernet protocol, this plug receives all packets, regardless of whether their Ethernet address matches this adapter's address.

`ne_plugs_flag_debug_rx`: This flag prints debugging information for each packet received by this plug. The debugging information is printed using `printf()`, and appears on the same serial port as other `printf()` outputs.

`ne_plugs_flag_debug_tx`: This flag prints debugging information for each packet transmitted by this plug. The debugging information is printed using `printf()`, and appears on the same serial port as other `printf()` outputs.

**Return Value:**

If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:**

**plugs.h**



## 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 subroutine you provide when you create a plug. The Plugs Ethernet Library calls this subroutine 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	This is a reference to the plug that is receiving a packet.
context	The value passed for the parameter named <code>callback_context</code> in <code>nr_plugs_create()</code> .

`p` This parameter is 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 0, 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, if you create a plug using the TCP protocol, when your callback subroutine 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` are all 0 because these protocols are not a part of a TCP packet.

The header field is a pointer to the first byte of the header for that protocol layer. If the header pointer is 0, 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, if you create a plug using the TCP protocol, when your callback subroutine is called, you could examine the enclosing Ethernet packet header by reading at `p[ne_plugs_ethernet].header`. You could also examine the enclosing IP packet header by reading at `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` are all 0 because these protocols are not a part of a TCP packet.

`payload` This is 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:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** `plugs.h`

## nr\_plugs\_destroy

**Syntax:**

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

**Description:** De-allocates a plug. When you no longer need a plug, call this subroutine to de-allocate any resources associated with the discarded plug.

**Parameters:**

`plug_handle` This parameter is a reference to the plug you are eliminating.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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 subroutine associates a plug with a particular remote IP address and port on the network. If the plug uses TCP, then this subroutine 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.

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

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

### Parameters:

<code>plug_handle</code>	A reference to the plug you are connecting
<code>remote_name</code>	A pointer to a string containing the name of a remote-network device (for example, <a href="http://www.altera.com">http://www.altera.com</a> ). The subroutine attempts to resolve the name to an IP address using the DNS server associated with the first adapter installed. If this parameter is 0, the <code>remote_ip_address</code> parameter is used instead.
<code>remote_ip_address</code>	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 <code>remote_name</code> parameter.
<code>remote_port</code>	If the port uses UDP or TCP, this parameter specifies the port number of the connection on the remote-network device.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** `plugs.h`



The `nr_plugs_connect` subroutine does not support transmission to another plug on the same Nios system or loopback.

## nr\_plugs\_send

**Syntax:**

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

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

**Parameters:**

<code>plug_handle</code>	A reference to a plug.
<code>data</code>	The payload to send.
<code>data_length</code>	The number of bytes in the payload.
<code>flags</code>	This parameter augments the flags specified by <code>nr_plugs_create()</code> . This parameter adds <code>ne_flag_debug_tx</code> to one particular transmission.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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 subroutine 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 subroutine. Do not use this subroutine on a plug using TCP.

**Parameters:**

<code>plug_handle</code>	A reference to a plug.
<code>data</code>	The payload to send.
<code>data_length</code>	The number of bytes in the payload.
<code>flags</code>	This parameter augments the flags specified by <code>nr_plugs_create()</code> . Typically this subroutine is used to add <code>ne_plugs_flag_debug_tx</code> to a particular transmission.
<code>ip_address</code>	The IP address of a remote-network device. The packet is transmitted to this remote-network device. This 32-bit parameter is in network byte order. To convert a 32-bit integer to network byte order, use the macro <code>nm_h2n32(x)</code> .
<code>port</code>	If the plug uses UDP, the packet transmits to this port on the remote-network device. This 16-bit parameter is in network byte order. To convert a 16-bit integer to network byte order, use the macro <code>nm_h2n16(x)</code> .

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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:** You should only call this subroutine if the plug uses TCP. This subroutine 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 subroutine is called. When a connection request is received, the callback subroutine 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 subroutines will be called and the first plug to accept the connection will be connected.

### Parameters:

<code>plug_handle</code>	A reference to the plug.
<code>callback</code>	A subroutine you provide to accept or decline an incoming TCP connection request. You may pass 0 for this parameter and any incoming TCP connection request will be accepted.
<code>callback_context</code>	This parameter is passed unmodified to your callback subroutine. It can carry state information to your subroutine.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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 subroutine you provide when you allow a TCP plug to accept connections using the `nr_plugs_listen()` subroutine. This subroutine can accept or decline the connection by returning a 0 (meaning no error occurred — accept the connection) or a negative value (meaning an error did occur — do not accept the connection).

**Parameters:**

<code>plug_handle</code>	A reference to a plug.
<code>context</code>	The value passed for the parameter named <code>callback_context</code> in <code>nr_plugs_listen()</code> .
<code>remote_ip_address</code>	The IP address of the remote-network device attempting to connect to this plug.
<code>remote_port</code>	The port on the remote-network device attempting to connect to this plug.

**Return Value:** Your subroutine should return 0 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 subroutine is given an IP address, it determines 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:**

<code>adapter_index</code>	This is the index number for the adapter being used.
<code>ip_address</code>	This is an IP address of a remote-network device.
<code>ethernet_address_out</code>	A pointer to a 48-bit Ethernet address. This subroutine will fill out this structure with the discovered Ethernet address.
<code>flags</code>	This flag can be 0 or <code>ne_plugs_flag_debug_tx</code> . If the flag is <code>ne_plugs_flag_debug_tx</code> and the operation fails, then a message is printed.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**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 subroutine is given the name of a remote-network device, it queries the name server to find out the IP address. This subroutine uses adapter number 0.

**Parameters:**

host\_name This is a pointer to a string containing the name of a network device.

host\_ip\_address\_out This is a pointer to a 32-bit IP address. This subroutine will fill out this value with the discovered IP address.

**Return Value:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** **plugs.h**

## nr\_plugs\_idle

**Syntax:**

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

**Description:** If the device does not enable interrupts, then this subroutine must be called frequently in your program's inner loop, or from a timer interrupt subroutine. It polls the hardware device for incoming packets, and dispatches them via each plug's callback subroutine.

**Parameters:** None

**Return Value:** If any errors occur, the return value will be negative.

**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 subroutine 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:** If this subroutine is successful, the return value is 0. If this subroutine fails, the return value is negative.

**Include:** `plugs.h`

The following subroutines and macros are for your general use and are required when translating between host-byte ordering and network-byte ordering for plugs Ethernet library-network programming.



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

## nr\_n2h16

<b>Syntax:</b>	<code>nr_n2h16(net_16 value)</code>
<b>Parameters:</b>	A network-short integer
<b>Description:</b>	Translates a network-short integer to a short integer.
<b>Equivalent Macro:</b>	<code>nm_n2h16(host_16)</code>

## nr\_h2n16

<b>Syntax:</b>	<code>nr_h2n16(host_16 value)</code>
<b>Parameters:</b>	A short integer
<b>Description:</b>	Translates a short integer to a network-short integer.
<b>Equivalent Macro:</b>	<code>nm_h2n16(host_16)</code>

## nr\_n2h32

<b>Syntax:</b>	<code>nr_n2h32(net_32 value)</code>
<b>Parameters:</b>	A network-long integer
<b>Description:</b>	Translates a network-long integer to a long integer.
<b>Equivalent Macro:</b>	<code>nm_n2h32(host_32)</code>

## nr\_h2n32

<b>Syntax:</b>	<code>nr_h2n32(host_32 value)</code>
<b>Parameters:</b>	A long integer
<b>Description:</b>	Translates a long integer to a network-long integer.
<b>Equivalent Macro:</b>	<code>nm_h2n32(host_32)</code>



*Notes:*

## A

ARP 17  
    Plugs Ethernet Library support 11

## B

build options  
    Plugs Ethernet Library 12  
byte order  
    Plugs Ethernet Library 13

## D

data structures  
    Plugs Ethernet Library 15  
DNS  
    Plugs Ethernet Library 12  
documentation feedback iv

## E

Ethernet 17

## F

features  
    Plugs Ethernet Library 10  
feedback, documentation iv

## I

ICMP 17  
    Plugs Ethernet Library 11  
int nr\_plugs\_listen 31  
IP 17  
    Plugs Ethernet Library support 11

## N

ne\_plugs\_flag\_debug\_rx 12  
ne\_plugs\_flag\_debug\_tx 12  
nr\_h2n16 19, 37  
nr\_h2n32 19, 37  
nr\_n2h16 19, 37  
nr\_n2h32 19, 37  
nr\_plugs\_connect 19, 28  
nr\_plugs\_create 19, 23  
nr\_plugs\_destroy 19, 27  
nr\_plugs\_idle 19, 35  
nr\_plugs\_initialize 19, 20  
nr\_plugs\_ip\_to\_ethernet 19, 33  
nr\_plugs\_listen 19  
nr\_plugs\_listen\_callback\_proc 19, 32  
nr\_plugs\_name\_to\_ip 19, 34  
nr\_plugs\_print\_ethernet\_packet 19, 36  
nr\_plugs\_receive\_callback\_proc 19, 25  
nr\_plugs\_send 19, 29  
nr\_plugs\_send\_to 19, 30  
nr\_plugs\_set\_mac\_led 19, 22  
nr\_plugs\_terminate 19, 21

## O

overview  
    plugs 9

## P

plugs  
    overview 9  
    SOPC Builder 9  
Plugs Ethernet Library 9  
    ARP (RFC 826) 11  
    build options 12  
    byte order 13  
    data structures 15

- DNS (RFC 1034 & 1035) 12
  - features 10
  - host ordering 13
- ICMP (RFC 792) 11
- IP (RFC 791) 11
  - payload descriptions 17
  - protocol architecture 10
  - protocols 11
  - protocols supported 10
  - system requirements 9
- TCP (RFC 793) 12
- UDP (RFC 768) 11
- PLUGS\_ADAPTER\_COUNT 13
- PLUGS\_DEBUG 12
- PLUGS\_DNS 13
- PLUGS\_PING 13
- PLUGS\_PLUG\_COUNT 12
- PLUGS\_TCP 13
- protocol
  - Plugs Ethernet Library architecture 10
- protocols
  - Plugs Ethernet Library 10, 11

## S

- SOPC Builder
  - plugs 9
  - Plugs Ethernet Library 9
- system requirements
  - Plugs Ethernet Library 9

## T

- TCP 17
  - Plugs Ethernet Library 12

## U

- UDP 17
  - Plugs Ethernet Library 11