digital

Debugging over Ethernet Using ARM SDT:

An Application Note

Order Number: EC–RB5PA–TE

February 1998

This document describes how to debug applications using the Angel remote debugger over an Ethernet connection. It also provides a description of how Angel interacts with the internet protocol (IP) stack and a description of the IP stack software sources.

Revision/Update Information: This is a new document.

Digital Equipment Corporation
Maynard, Massachusetts
http://www.digital.com/semiconductor

February 1998

While DIGITAL believes the information included in this publication is correct as of the date of publication, it is subject to change without notice.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Contents

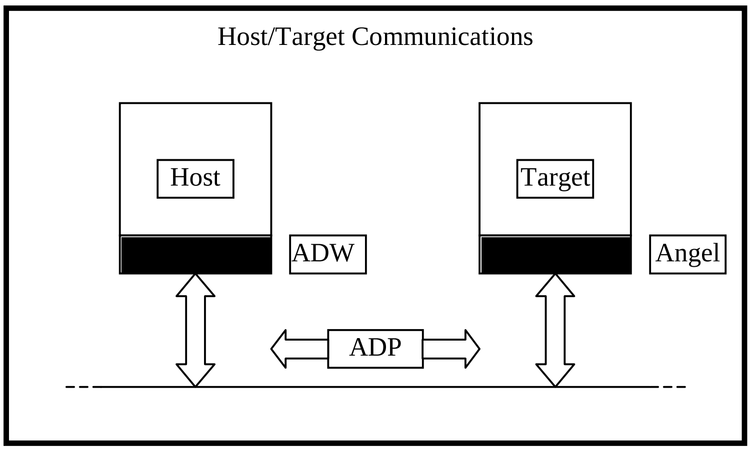
Figures

1. Introduction

All of the DIGITAL Semiconductor StrongARM evaluation boards support the Angel remote debugger. This allows a host-based debugger to load and debug software in the remote target via the Angel Debug Protocol (ADP). ADP is supported in both serial and Ethernet-based connections. The ADW is the ARM Debugger for Windows, currently Version 2.11.

To facilitate faster downloads and faster debugging of application software, an internet protocol (IP) stack has been added to the Angel debugger target for some StrongARM evaluation boards; for example, EBSA-285 and EBSA-110. See Figure 1, Debug Block Diagram.

Figure 1 Debug System Block Diagram



For it to resolve its own IP address, the system requires a BOOTP server to be running on the network to which the target is connected. The user may require assistance from a system administrator to do this. Beyond this, the differences between using the ARM debugger over serial and Ethernet should be largely transparent, other than the faster download speed.

2. Getting Started

To set up the target system for remote debugging via IP will require the following information:

- The BOOTP server needs to be set up to respond to the MAC address of the target in order to use the Ethernet connection. The MAC address of an Ethernet card should be provided on a label on the card itself. In the case of the EBSA-110, it is on the Angel banner, if the version of Angel installed is Ethernet aware. The banner may be read by connecting a terminal to COM1 of the EBSA-110 at 9600 baud, no parity, 8 data bits, 1 stop bit.
- To configure the host debugger, the user will need to know the IP address that will be assigned to the target system by the BOOTP server. This will be set up within the BOOTP server.

2.1 Target

Set up the target hardware (for example, in the case of an EBSA-285, insert the EBSA-285 into the system slot of a PCI backplane[1]) and the Ethernet card into one of the remaining slots. Connect the network cable and power up. The target will now attempt to execute BOOTP, then wait for a connect request from a host (on all supported communications media).

2.2 BOOTP

The BOOTP server will need to be set up on the same subnet as the target to be debugged. The user will need to add an entry in the BOOTP table that maps the IP address to be assigned to the target to the MAC address of the Ethernet device associated with the target.

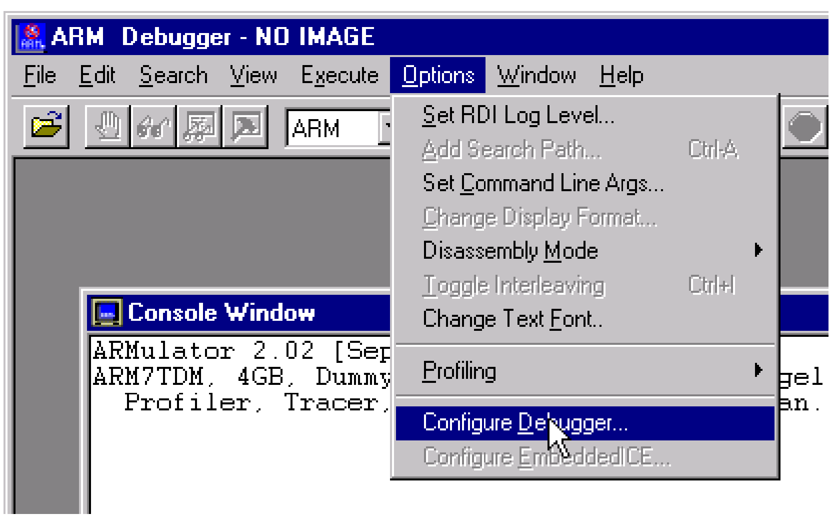
It may be useful to attach the target system and power it on, then verify that BOOTP is functioning correctly. This may be done by checking the details of BOOTP transactions in the log on the BOOTP server. There should be a record of the target system in use successfully executing BOOTP.

2.3 Host Side

Some options need to be changed to configure the host side debugger; namely, set the IP address and select Ethernet as a connection medium.

To do this, select Options | Configure Debugger from the menu bar (see Figure 2, ARM Debugger Options Menu).

Figure 2 ARM Debugger Options Menu

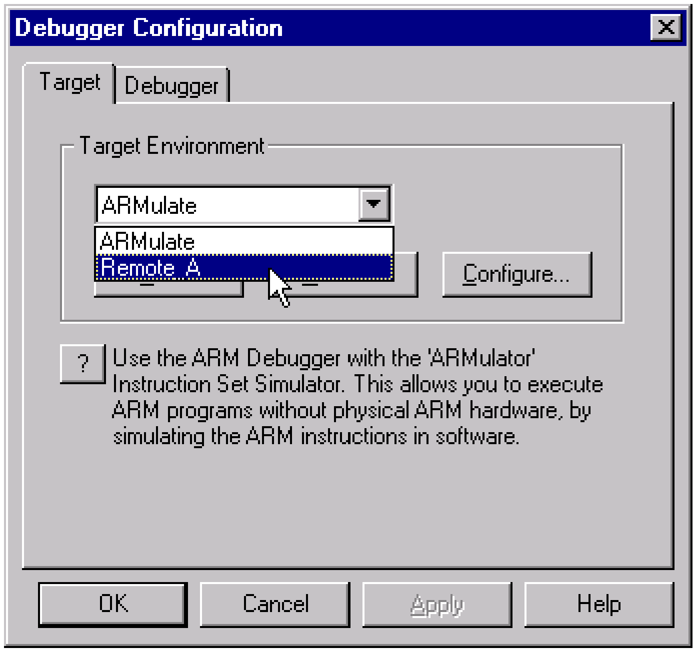


Next, select Remote_A, which is the remote Angel debugger plug-in, then press the Configure button (see Figure 3 Debugger Configuration).

---

[1] The EBSA-285 will need to be configured correctly. See the *DIGITAL Semiconductor EBSA-285 Evaluation Board Reference Manual*, EC-R6M5B-TE.

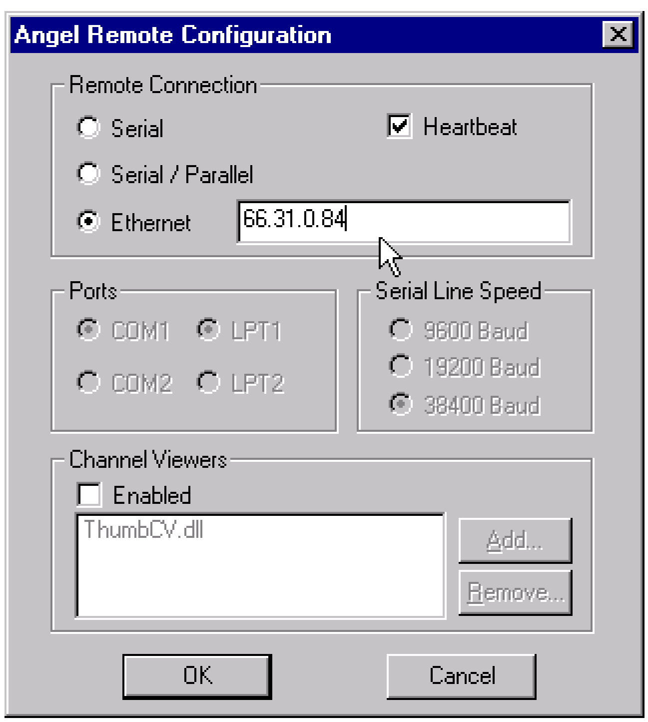
Figure 3 Debugger Configuration



The details of IP address and Ethernet may be entered in the window after pressing Configure (see Figure 4 Angel Remote Configuration).

Note: Heartbeat is switched on. Heartbeats are a part of the ADP and are used to detect lost packets.

Figure 4 Angel Remote Configuration



Press OK on all of the dialog boxes.

Pressing OK on the Debugger Configuration dialog () will cause the debugger to attempt a connection over IP. If the target has had time to complete its BOOTP (about 10-20 seconds), then the debugger should connect. If not, the user may need to retry.

Using the debugger from this point is the same as using over serial. (See the Limitations and Known Problems section.)

2.4 Troubleshooting

If the remote debugger cannot establish a connection, try the following:

- Check that the EBSA is powered on and configured.
- Check that there is an Ethernet cable connected to both the host and target system and the BOOTP server.
- Is there a BOOTP server set up on the network that will respond to the Ethernet card in your system (that is, is it set up to respond to the MAC address of the target Ethernet card in use)? Ask the system administrator for help, if necessary.
- Ensure that the host and target are on the same subnet of the network.

  The target must support the Ethernet card. Currently, the EBSA-285 supports 21040 and 21140 based Ethernet cards operating over 10BASE-T.
- Check that the EBSA-285 is in the system slot of the EBSA-BPL. For a description of the EBSA-BPL, see the *DIGITAL Semiconductor PCI Development Backplane User's Guide*, EC-R6M4B-TE.
- Ensure the link light is active on the Ethernet card. If it is not, the card is not being initialized. This may be because of jumper settings on the EBSA, a defective lead, or an inactive hub.

3. Description

3.1 BOOTP Sequence

BOOTP is a protocol that was designed to download programs to diskless network machines but may also be used to determine the IP address of a system. The target transmits broadcast BOOTP request packets and then listens for replies from a BOOTP server or times out in the attempt. (It will transmit 4 requests, approximately 2 seconds apart.) When a valid BOOTP packet is received and intended for the target, the IP address is copied from it and then is stored in the target system to use as its own address.

If a valid address is assigned, a connection may be made from the ADW using this target IP address; otherwise, the IP connection to Angel will not be usable.

During the BOOTP period, approximately 10-20 seconds, it will not be possible to connect to Angel by any medium.

3.2 Host to Target Connection

To initiate a connection, the host sends a UDP packet containing a 'magic' word to the target IP address. This is sent over a known port number, the control port, to the target from the host.

The target replies with a UDP packet containing the other application port and debug port numbers to the host. The channel is now considered to be open and is used like any other medium from the viewpoint of Angel and the user.

4. Limitations and Known Problems

4.1 General

The Ethernet stack will be disabled once a connection has been established using a medium other than Ethernet. **Solution:** The target will require a reset to re-enable Ethernet.[2]

If there is a large load on the host system, such as 'Microsoft findfast' running while using the debugger over Ethernet, then a packet may be missed by the host and cause the debugger to hang. **Solution:** Try to use the debugger on its own if possible.

Stepping too rapidly may cause the debugger to hang. The host end of the link appears to miss a packet being sent from the target. **Solution:** Wait for the debugger screen to update after each step.

Beware when debugging programs with large amounts of text output. On some machines, Windows may struggle to update the windows quickly, and may lose mouse clicks and key presses.

4.2 EBSA-110

Supports the onboard AM79C961 Ethernet controller only.

4.3 EBSA-285

The EBSA-285 supports only one Ethernet card per backplane. It will use the first Ethernet card it finds whether or not it is in use by another device. Each slot is checked for a particular device type, in descending slot number order, before checking for the next device type in each slot. The process will stop when a card is found or there are no more known device types.

The device search order is: 21040(*), 21041, 21140(*), 21142, and 21143.

Note: Only (*) device types are currently supported, although Angel will attempt to set up every device type in this list.

The EBSA-285 supports 21040 and 21140 based devices only.

[2] It is possible to connect via serial after Ethernet has been used without requiring a reset.

5. Software Description

The following is a description of how the various source modules interact and their primary functions.

The software was built using the ARM SDT Version 2.11 and the software supplied with the EBSA board.

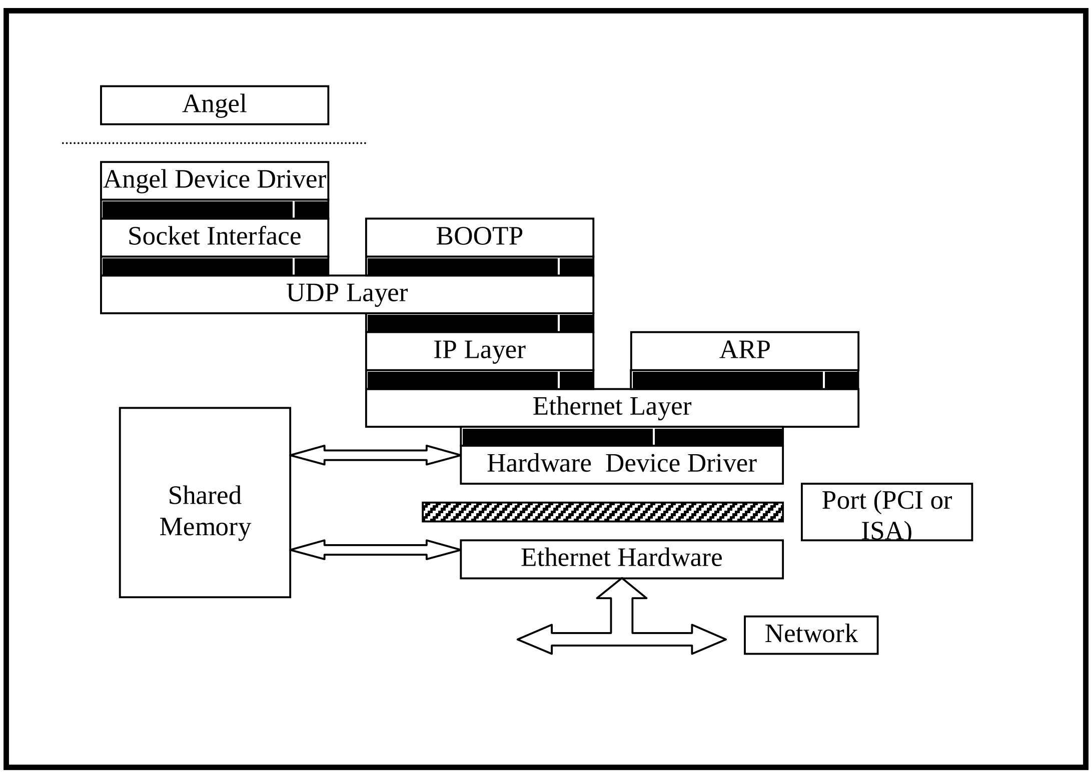
For toolkit information, refer to: http://www.arm.com/

For EBSA software, refer to:
http://www.digital.com/semiconductor/strongarm/strongar.htm

5.1 How the IP Stack Fits with Angel – Overview

Access to the top of the Ethernet stack is via a socket style interface (see Section 5.4). At startup, Angel will call the netboot module. It is within this code that the initialization functions for each module are called along with the BOOTP sequence. If successful, Angel will then attempt to open three sockets: one for control, one for application communication, and one for debug information. However, if netboot fails, then Angel is informed that no device is present[3].

Figure 5 IP Stack Layout



[3] Angel ignores this value; therefore, a workaround is used. The IP stack is accessed only if it is the active device.

The sockets may then be polled via `recv/recvfrom` functions. The data from these packets is passed up into Angel.

When a write is requested by Angel, the data is passed into the `sendto` function, packaged, and sent down the IP stack layers to the destination host.

The IP stack is not interrupt driven; it is polled. It was determined that there were no advantages in driving the IP stack by interrupts rather than polling.

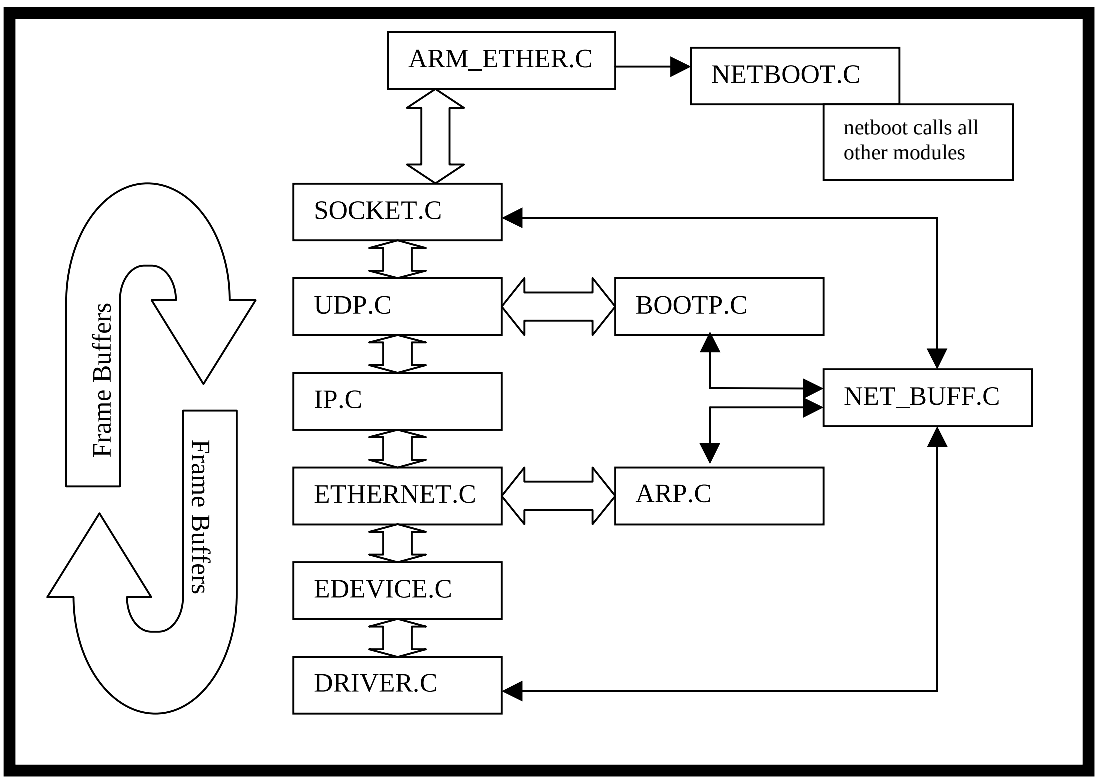
The Angel device driver layer contains all the functions Angel knows about concerning the IP stack.

5.2 Relationship Between the Source Files

The file interaction is shown in Figure 6, Source Calling Structure.

The frame buffers circulate through the stack from top to bottom for writes, and bottom to top on reads. They are 'borrowed' from a central pool in the shared memory area (see Figure 5) but do not leave the realms of the IP stack. Instead, their contents are copied to/from buffers passed into the socket layer. The frame buffers are allocated in an area of memory that is accessible by both the StrongARM and the Ethernet devices.

Figure 6 Source Calling Structure



5.3 Angel Device Driver

File: arm_ether.c

Usage: Interacts with socket layer, calls initialization routines for the IP stack, opens sockets, polls stack for data, and handles copying read/write buffers between Angel and the stack.

This contains the standard Angel device driver functions. Angel calls `Ethernet_Control` with a value of DC_INIT during its initialization phase. It is at this point that the IP stack is initialized, in file netboot.c, followed by an attempt to open and bind the three required sockets in `Ethernet_init`. The three sockets are called 'control', 'application', and 'debug'. **Note:** All IP stack operations occur in supervisor mode.

Polling the three sockets occurs in the `Angel_EthernetPoll` routine, but only if the stack is correctly initialized and Ethernet is the active medium. When a packet is found to be waiting and is destined for one of the open sockets, the buffer is copied from stack buffer to Angel buffer and an Angel callback is queued to handle the data.

5.4 The Socket Style Interface

File: socket.c

Usage: Partial socket layer implementation.

This is a fraction of the implementation of a full socket layer, implementing only the functions necessary for the Angel application. These functions and their limitations include:

- *SOCKET* – Socket ignores the protocol it is asked to use, assuming UDP only.
- *BIND* – Bind will attach only a socket to a UDP port. It does a check to see if the protocol/connection type combination exists. They must be SOCK_DGRAM/AF_INET for this to function correctly.
- *RECVFROM* – The stack is polled. Hence, when `recvfrom` is called, the lower layers are interrogated for packets before checking the outstanding packet queue in the socket layer. Any packets claimed are removed from the queue.
- *RECV* – This calls `recvfrom` but ignores the extra data in the address structure.
- *SENDTO* – The socket number is checked, then the packet is immediately dispatched down the stack to the UDP layer, making the assumption that this is the intended transmission protocol.

The socket layer maintains a list of open sockets, and the ports they are bound to, allowing it to direct packets to the correct destination ports and sockets. Packets coming in from the Ethernet are passed into the input function from the layer below (presently only UDP) and are queued to be read later by `recv/recvfrom`.

Note: The data packets do not move around, only the pointers to them.

Packets destined for the Ethernet are passed from the layer above into the `sendto` function.

5.5 The UDP Layer

File: udp.c

Usage: Provides unreliable datagram delivery layer. Directs upstream packets to the layer attached to a bound port number that is indicated within the packet. In the Angel case, the three ports that are open will correspond to the three opened sockets.

The UDP layer maintains a table of open port numbers and the input function associated with each port number. When a packet is passed into the UDP input function, it is checked for validity before being sent to the function associated with the port number in the packet, or discarded if the port is not open.

Before using the UDP layer, a port needs to be opened. There are two methods to acquire a port number. The first step is to open a 'well known' port – in this case, the layer above provides the port number to the `register_known_port` function. The second is to let the UDP layer assign a port number with the `create_port` function. This port number is used in the `udp_register_port` function and is bound to the function pointer specified in the call.

5.6 The IP Layer

File: ip.c

Usage: Handles IP traffic, sending packets from the network to a function bound with a registered protocol number (for example, UDP) contained within the packets. Outgoing packets have IP style addresses resolved to MAC type addresses in the ARP layer before being sent out.

Protocols are registered using the `ip_register_protocol` function, using the protocol number and the function associated with that protocol. Incoming packets are sent to this bound function.

5.7 The Ethernet Layer

File: ethernet.c

Usage: Passes incoming Ethernet packets to a function bound to the protocol type contained within the packet. Builds Ethernet type packets for transmission and then passes them to the device driver.

Packets need to be explicitly read using the `ethernet_process_one_packet` function, which checks the device driver for outstanding data frames, via an abstraction layer, passing any valid incoming data up to the function bound to the packet's protocol type. Protocols are registered via the `ethernet_register_protocol` function. Writes to the wire require a MAC style address and destination protocol number.

5.8 Device Driver

Files: <driver>.c (for example: dec21040.c , edevice.c)

Usage: Handles initialization, reading and writing with the physical hardware. The edevice.c contains an abstraction layer to handle multiple devices and sits above the device drivers.

An Ethernet device is hunted during module initialization. The first recognized device found is set up during the device initialization call. Its MAC address is retrieved before setting its CSRs to a condition where it filters MAC addresses as required and listens to the wire. This involves waking the device, acquiring shared memory areas for descriptors and frames to reside, setting medium selection registers, and then handing a setup frame containing the MAC address filter information.

Interrupts are not enabled so the device needs to be polled regularly to check for incoming frames.

6. Additional Information

The source files referred to within this document, along with prebuilt images, will have been included with the EBSA board package supplied by DIGITAL Semiconductor. Information about building Angel is also contained in the package. This can be used to migrate the port to a derivative design.

Work is ongoing in this area. Revised versions of source and Angel related files will be provided to the sales channels and posted on the WWW when available. Please check the release notes for any new features and supported devices.

Support, Products, and Documentation

If you need technical support, a *DIGITAL Semiconductor Product Catalog*, or help deciding which documentation best meets your needs, visit the DIGITAL Semiconductor World Wide Web Internet site:

http://www.digital.com/semiconductor

You can also contact the DIGITAL Semiconductor Information Line or the DIGITAL Semiconductor Customer Technology Center for support.

For documentation and general information:	
DIGITAL Semiconductor Information Line	
United States and Canada:	1–800–332–2717
Outside North America:	1–510–490–4753
Electronic mail address:	semiconductor@digital.com

For technical support:	
DIGITAL Semiconductor Customer Technology Center	
Phone (U.S. and international):	1–978–568–7474
Fax:	1–978–568–6698
Electronic mail address:	ctc@hlo.mts.dec.com

DIGITAL Semiconductor Products

To order the DIGITAL Semiconductor products, contact your local distributor. Evaluation board kits include a complete design kit, Windows NT installation kit, and an accessories kit with an evaluation board.

DIGITAL Semiconductor Documentation

The following table lists some of the available DIGITAL Semiconductor documentation.

Title	Order Number
DIGITAL Semiconductor EBSA-285 Evaluation Board Reference Manual	EC–R6M5B–TE
DIGITAL Semiconductor SA-110 Microprocessor Evaluation Board Reference Manual	EC–QU5KA–TE
DIGITAL Semiconductor PCI Development Backplane User’s Guide	EC–R6M4B–TE

Third-Party Documentation

You can order the following third-party documentation directly from the vendor.

Title	Vendor
Internetworking with TCP/IP: principles, Protocols and Architecure	Author: Douglas Comer. ISBN: 0-13-470188-7
ARM Software Development Toolkit User Guide	Author: Advanced RISC Machines Ltd. Doc Number: ARM DUI 0040C