OS-9 for ENP-2505 Board Guide

Version 3.2

www.radisys.com
World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044
International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620
RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000
Revision A
December 2001

Table of Contents

Chapter 1: Installing and Configuring OS-9

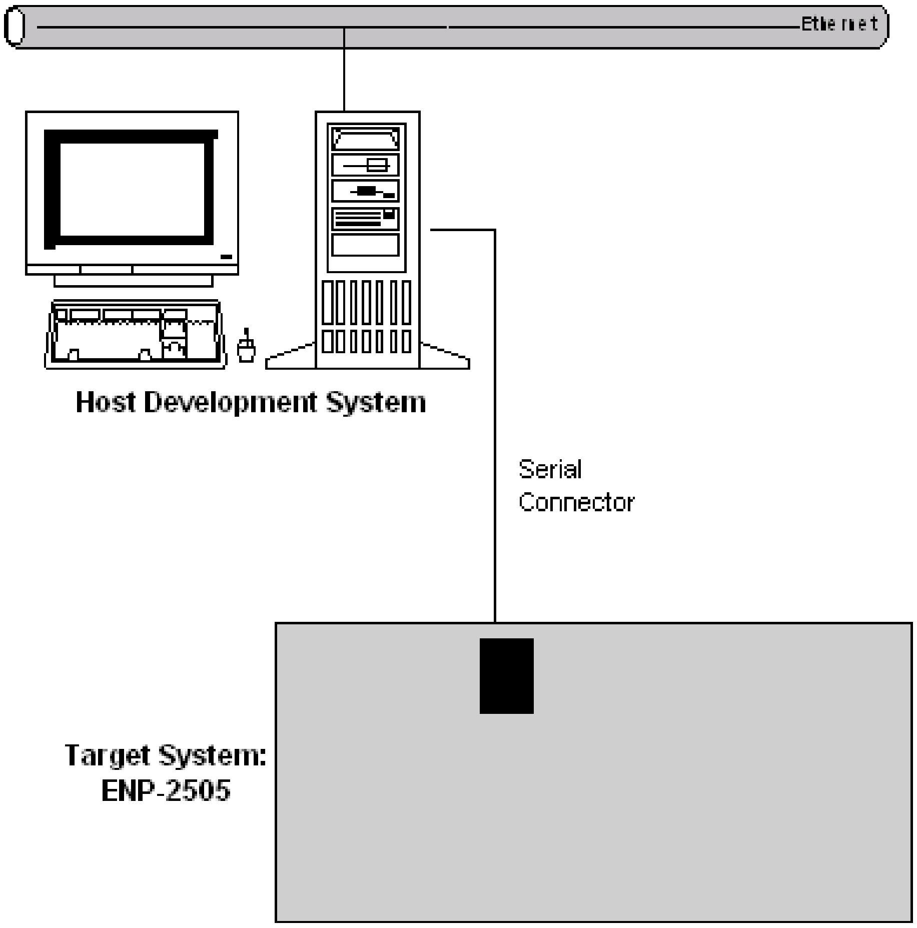
This chapter describes installing and configuring OS-9 on the RadiSys ENP-2505 board. It includes the following sections:

- **Development Environment Overview**
- **Requirements and Compatibility**
- **Connecting the Target to the Host**
- **OS-9 ROM Image Overview**
- **Using the Configuration Wizard**
- **Optional Procedures**

Development Environment Overview

Figure 1-1 shows a typical development environment for the RadiSys ENP-2505 board. The components shown include the minimum required to enable OS-9 to run on the ENP-2505.

Figure 1-1 ENP-2505 Development Environment



Requirements and Compatibility

Note

Before you begin, install the ***Enhanced OS-9 for IXP1200*** CD-ROM on your host PC.

Host Hardware Requirements (PC Compatible)

The host PC must have the following minimum hardware characteristics:

- 250MB of free hard disk space
- the recommended amount of RAM for the operating system
- a CD ROM drive
- a free serial port
- an Ethernet network card
- access to an Ethernet network

Host Software Requirements (PC Compatible)

The host PC must have the following software installed:

- Enhanced OS-9
- Windows 95, 98, ME, 2000**,** or NT 4.0
- terminal emulation program

Target Hardware Requirements

Your reference board requires the following hardware:

- enclosure or chassis with power supply and backplane
- access to an Ethernet network

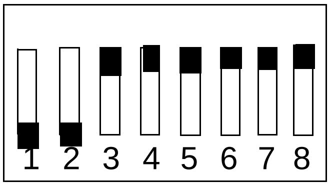
Connecting the Target to the Host

Please refer to the ***ENP-2505 Hardware Reference*** manual for appropriate installation and configuration procedures.

DPI Switch Settings

Configure your settings according to the following diagram. This will allow the Configuration Register to boot the boot manager.

Figure 1-2 SW1301



OS-9 ROM Image Overview

Overview

The OS-9 ROM Image is a set of files and modules that collectively make up the OS-9 operating system. The specific ROM Image contents and functionality can vary from system to system depending on hardware capabilities and user requirements.

To simplify the process of creating, loading, and testing OS-9, the ROM Image, called rom, is generally divided into two parts—the low-level image, called coreboot; and the high-level image, called bootfile.

Coreboot

The coreboot image is generally responsible for initializing hardware devices and locating the high-level (or bootfile) image as specified by its configuration. For example from a Flash part, a harddisk, or Ethernet. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, applications). The image is loaded into memory based on the device you select from the boot menu. The bootfile image normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

Microware provides a configuration wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The configuration wizard is automatically installed on your host PC during the Enhanced OS-9 installation process.

Using the Configuration Wizard

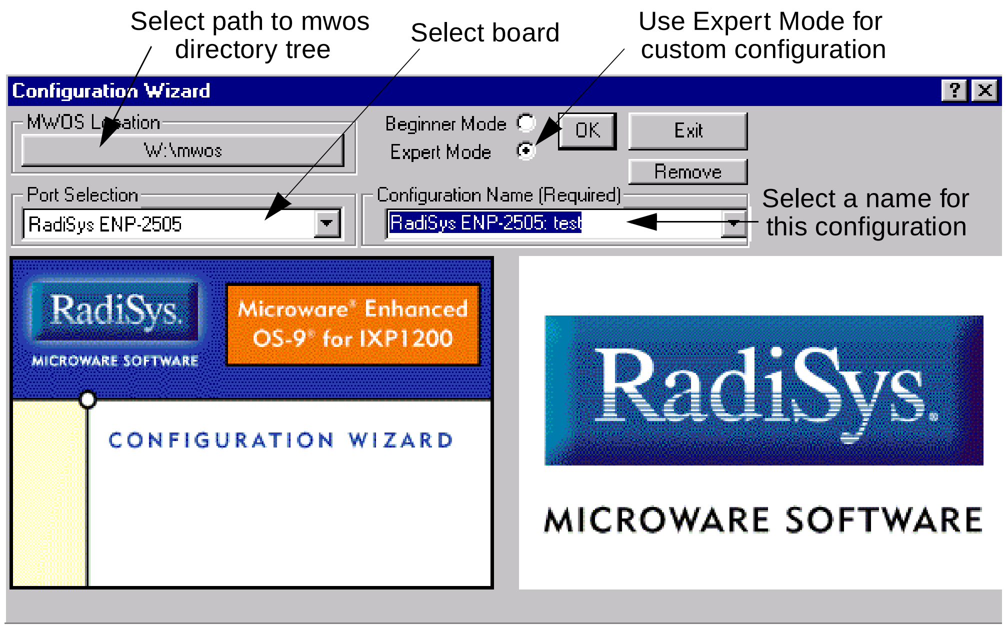
The Configuration Wizard is used to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The Wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the Enhanced OS-9 installation process.

To start the Configuration Wizard, perform the following steps:

Step 1. Click the `Start` button on the Windows desktop.

Step 2. Select `Programs` -> `RadiSys` -> `Enhanced OS-9 for IXP1200` -> `RadiSys Configuration Wizard`. You should see the following opening screen:

Figure 1-3 StrongARM Configuration Wizard



Select path to mwos directory tree
Select board
Use Expert Mode for custom configuration
Select a name for this configuration
Configuration Wizard
OK
Exit
Remove
RadiSys.
MICROWARE SOFTWARE
Microware® Enhanced OS-9® for IXP1200
CONFIGURATION WIZARD
RadiSys.
MICROWARE SOFTWARE

MWOS Location: W:\mwos

Beginner Mode ○
Expert Mode ●

Port Selection: RadiSys ENP-2505

Configuration Name (Required): RadiSys ENP-2505: test

Step 3. Select the path where the MWOS directory structure can be located by clicking the MWOS location button.

Step 4. Select the target board from the Port Selection pull-down menu.

Step 5. Select a name for your configuration in the Configuration Name field. Your settings will be saved for future use. This enables you to modify the ROM image incrementally, without having to reselect every option for each change.

Step 6. Select `Expert Mode` and click `OK`. The Main Configuration window is displayed. **Expert** Mode enables you to make more detailed and specific choices about what modules are included in your ROM image.

Creating a ROM Image for the ENP-2505 Target System

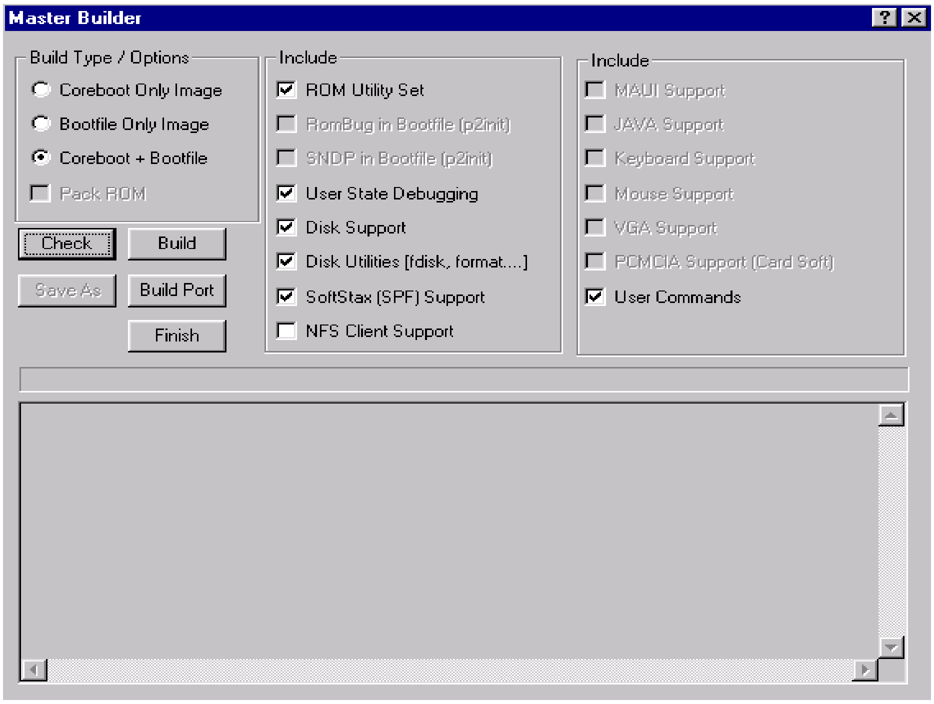
Once you are in **Expert** mode of the Configuration Wizard, you can build a ROM image for the target system. To do this, complete the following steps:

Note

Advanced configuration procedures are described in the **Optional Procedures** section.

Step 1. Select `Configure -> Build Image`. The **Master Builder** dialog window appears.

Figure 1-4 Master Builder Dialog Window



Master Builder

Build Type / Options
- ○ Coreboot Only Image
- ○ Bootfile Only Image
- ● Coreboot + Bootfile
- [ ] Pack ROM

Include
- [✓] ROM Utility Set
- [ ] RomBug in Bootfile (p2init)
- [ ] SNDP in Bootfile (p2init)
- [✓] User State Debugging
- [✓] Disk Support
- [✓] Disk Utilities [fdisk, format....]
- [✓] SoftStax (SPF) Support
- [ ] NFS Client Support

Include
- [ ] MAUI Support
- [ ] JAVA Support
- [ ] Keyboard Support
- [ ] Mouse Support
- [ ] VGA Support
- [ ] PCMCIA Support (Card Soft)
- [✓] User Commands

Step 2. Configure your Master Builder options as shown in **Figure 1-4**.

Step 3. Click `Build`. A ROM image is created to be placed on the target. The image, `os9kboot`, is stored in the following default location:

```
\mwos\OS9000\ARMV4\PORTS\ENP2505\BOOTS\INSTALL\PORTBOOT\
```

Clicking `Save As` after the build operation is optional; it enables you to save the ROM image to a location of your choice.

Note

The specific contents of your bootfile can vary from system to system depending on hardware capabilities and user requirements.

Programming the Flash

There are two ways to program the Flash: programming the ROM image and programming the Flash on OS-9. These two methods are discussed in the following sections.

Programming the ROM Image

The ROM image you created in the previous section will be placed at the address 0xc0000, in the Flash part (OS section of Flash on the ENP-2505). Once this has been done, you can program the image onto the ENP-2505 using the `futil.exe` program. Below is a sample session showing the use of the `futil.exe` program.

```
% FUtil.exe
Flash Utility, Version 1.1
Numeric parameters (other than the bank and comm port) should be
entered as hexadecimal numbers without the leading '0x'.
Enter a blank line to accept the default value.
Enter a period ('.') to abort the program.
Use Serial Port (Yes):yes
Comm port (1):1
FlashUtil, Version 1.0.0
CPU Revision: 6901C123
Flash Width: 32
Flash Bank Size: 4096 KBytes
Flash Type Bank[0]: 28F320C3-B
Flash Type Bank[1]: 28F320C3-B
Flash Type Bank[2]: 28F320C3-B
Flash Type Bank[3]: 28F320C3-B
Starting at bank 0
Filename (): rom
Starting offset in flash (000000): 200000
Ending offset in flash (1C0000): 800000
```

```
Starting offset in file (000000)
Sending data to remote system...
 Percent complete: 100
Updating flash...
 Percent Complete: 100
done
```

For More Information

Refer to Appendix C of the ***ENP Software Development Kit Programmer’s Guide*** for more information on using the `futil.exe` utility.

Programming the Flash on OS-9

If you choose, instead of programming the ROM image, you may program the Flash onto OS-9. Once the ENP-2505 boots to OS-9, you can use the `pflash` utility to reprogram the Flash. There are two ways to do this. The first example shows how to program a ROM image:

```
$ pflash –f=rom -ri
```

The second shows how to program a bootfile image:

```
$ pflash –f=os9kboot
```

In either case, you are responsible for transferring the file to the ENP-2505 (via ftp or kermit).

Optional Procedures

The following section assumes you have successfully built a ROM image using the Configuration Wizard and have successfully programmed the Flash.

Establishing an Ethernet Connection

There are no resident Ethernet connections on the ENP-2505 board, aside from the 4 10/100 Microcode connections. However, an Ethernet connection can be established using a special OS-9 driver that communicates across the PCI bus to the PCI Master. Once the connection has been established, the PCI Master can route the packets to your host machine. The sections below describe how to configure the PCI Master and your host machine in order to establish an Ethernet connection to the ENP-2505.

For More Information

Refer to the “Debugging with the 10/100 Mbit Ethernet Port” section of Appendix C: Debuging with the Ethernet Router Application for more information on debugging with the Ethernet Router Application.

X86 PCI Master Setup

If you intend to use Ethernet-based debugging, such as Hawk or the Intel Workbench, you must configure the PCI Master. The hardware that usually acts as the PCI Master is an X86 machine running OS-9. This hardware should contain a 133MHz Pentium with at least 32MB RAM.

To set up the X86 PCI Master, complete the following steps:

Step 1. Configure the OS-9 X86 target with networking enabled.

For More Information

Refer to the ***80386 Board Guide*** for help on enabling Networking capabilities.

Note

The Ethernet stack must be configured to route packets. To do this, complete the following steps:

1. Edit the `/MWOS/SRC/DPIO/SPF/DRVR/SPIP/DEFS/spf_desc.h` file.
2. Change `#define GATEWAY 0` to `#define GATEWAY 1`.
3. Navigate to the `/MWOS/SRC/DPIO/SPF/DRVR/SPIP` directory and type the following command:

   ```
   os9make –f spfdesc.mak
   ```

   This will recompile the `ip0` networking descriptor to allow routing of packets between interfaces.

Step 2. Boot the X86 board and copy the PCI Ethernet driver onto the target using FTP. (`/MWOS/OS9000/80386/PORTS/PCAT/CMDS/BOOTOBJS/SPF/spixhpch,sppch`)

Step 3. Create a directory on the target in which to place the PCI driver by typing the following command:

```
$ makdir /h0/HPCH
```

On Windows, use FTP to transfer `spixphcph` and `sppch` to the `/h0/HPCH` directory on the target.

Step 4. Load and configure the PCI Ethernet driver. On the X86 target, type the following commands to load the PCI Ethernet driver and to initialize communication with the ENP2505:

```
$ load –ld /h0/HPCH/*
$ ifconfig slip0 192.168.10.5 192.168.10.4 binding
/sppch
```

Note

The lines above can be added to your `/h0/sys/startup` file. This will cause these commands to run automatically when your X86 boots.

At this point, you should be able to ping, FTP, or telnet between the ENP-2505 and the X86.

Step 5. Log onto FTP by typing the following commands at the prompt:

```
user: super
password: user
```

Step 6. Type the following command to telnet into the ENP-2505:

```
$ telnet 192.168.10.4
```

The following message should appear:

```
Trying 192.168.10.4…Connected to 192.168.10.4
```

Configuring your Host Machine

To connect directly to the ENP-2505 board from your host machine, a route must be added. The route will direct all packets to 192.168.10.4 to the X86 target. From a DOS prompt, type the following command:

```
% route add 192.168.10.4 <x86 IP Address>
% route print
```

The following message should appear:

```
Active Routes:

Network        Netmask          Gateway    Interface           Metric
Destination

192.168.10.4   255.255.255.255  <x86 IP    <Windows IP Addr.>  1
                                Addr.>
```

You should now be able to ping, telnet, or FTP directly to the ENP-2505 board.

Note

A default route exists from the ENP-2505 to the X86 target. This can be verified by using the `route` command on the ENP-2505. If a default route does not exist, type the following command:

```
$ route add default 192.168.10.5
```

Using FTP and pflash

Once you have configured the X86 PCI Master, you can transfer the bootfile image (os9kboot) from the host system to the target system. To do this, complete the following steps:

Step 1. Start a DOS shell on the host system.

Step 2. Navigate to the directory in which the bootfile image, `os9kboot`, resides. The default location for this file is in the following directory:

```
\mwos\OS9000\ARMV4\PORTS\ENP2505\BOOTS\INSTALL\PORTBOOT\
```

Step 3. At the prompt type the following command:

```
ftp <IP address of target>
```

Step 4. Log onto FTP by typing the following commands at the prompt:

```
user: super
password: user
```

Step 5. At the `ftp>` prompt type the following command:

```
bin
```

This designates binary format.

Step 6. At the `ftp>` prompt type the following command:

```
cd /r1
```

Step 7. At the `ftp>` prompt type the following command:

```
put os9kboot
```

The `os9kboot` file is transferred to the target's RAM disk (/r1).

Step 8. Open the IXP1200 Hyperterminal window on the Windows host desktop.

Step 9. From the OS-9 prompt ($) type the following command:

```
$ pflash -f=/r1/os9kboot
```

The `os9kboot` file is transferred to the target system's Flash memory. `pflash` is configured to copy the os9kboot file to the correct address.

Step 10. Reset the target system.

For More Information

The pflash utility is documented in the **Port Specific Utilities** section of **Chapter 2**.

This procedure moves `os9kboot` to the target system's Flash memory. The next time the target system is booted, it will boot to OS-9 from Flash with the functionality you added.

The Boot Manager

The Boot Manager (BootMgr) allows you to select which of the operating systems residing in Flash will be booted by the board. The BootMgr is accesible via a terminal connected to the board's serial port. Your current settings are saved into the Flash; thus, if no interaction is detected by the BootMgr, the current default operating system is automatically booted.

Note

The Boot Manager region in Flash is write protected and the reflashing tools described here will not update that region of the flash memory.

When the BootMgr starts, it prints out a banner on the serial port (at 38400 Baud):

```
Press space bar to stop auto-boot...
     10
```

This value (which is configurable) then counts down to zero. If it reaches zero before you press the space bar, the current default operating system will be booted. After auto-booting is stopped, the BootMgr prompt is displayed. You can then enter a command. To display all possible commands, type h, as shown below:

```
[BootMgr]: h
BootMgr commands:
p : Print boot parameters
c : Change boot parameters
b : Boot with current parameters
b <os> : Boot given os without changing parameters
h : Print this help message
```

Boot Manager Commands

The following section describes some of the available Boot Manager commands:

The print command p will display the current parameters:

```
[BootMgr]: p
BootMgr Version 1.0.2
CPU Revision 6901C123
OS list:
     0 Flash Utility
     1 Monitor
     2 Reserved
     3 VxWorks
     4 OS-9
Default OS: 4
Countdown value: 10
Disable initial display: 0

[BootMgr]:
```

The four parameters that may be varied include:

default operating system	This is the OS that will be booted if the autoboot countdown gets to zero. It's value is limited to the displayed list.

countdown value:	This is the number of seconds that the BootMgr will wait before auto-booting the default OS. The tradeoff is that with smaller values, there will be less delay in booting the desired OS, but the user will also have to be quicker in changing the boot parameters. This parameter is limited to values between 1 and 60 inclusive; i.e. a value of 0 is not allowed.
disable initial display:	If this value is set to 1, then the countdown display will not be printed. The BootMgr will still count down before auto booting, but no messages will be displayed unless the autoboot is stopped.

These parameters can be modified with the change command c. This command will print the current parameters and then prompt the user for new values. A blank line will leave the current value unchanged. This dialog looks similar to:

```
BootMgr Version 1.0.2
CPU Revision 6901C123
OS list:
     0 Flash Utility
     1 Monitor
     2 Reserved
     3 VxWorks
     4 OS-9
Disable initial display: 0
Enter blank line to leave value unchanged
Default OS:
Countdown value: 3
Disable initial display:

[BootMgr]:
```

In the example above, the countdown value was changed from 10 to 3, and the other parameters reamained unchanged.

The boot command b will boot the default operating system. Alternately, it can be given a numeric argument (such as `b 0`), which will boot the indicated operating system without changing the default. This could be used, for example, to boot the flash utility without modifying the default operating system.

Chapter 1: Board Specific Reference

This chapter contains porting information that is specific to the Intel IXP1200 Network Processor Reference Board. It includes the following sections:

- **Boot Options**
- **The Fastboot Enhancement**
- **OS-9 Vector Mappings**
- **Port Specific Utilities**
- **Intel Work Bench Daemons**

For More Information

For general information on porting OS-9, see the ***OS-9 Porting Guide.***

Boot Options

The default boot options for the IXP1200 Network Processor Evaluation Board are listed below. The boot options can be selected by hitting the space bar during the IXP1200 bootup when the following message appears on the serial console:

```
Now trying to Override autobooters
```

The configuration of these booters can be changed by altering the `default.des` file, which is located in the following directory:

```
mwos\OS9000\ARMV4\PORTS\ENP2505\ROM
```

Booters can be configured to be either menu or auto booters. The auto booters automatically attempt to boot in order from each entry in the auto booter array. Menu booters from the defined menu booter array are chosen interactively from the console command line after getting the boot menu.

Booting from Flash

When the `rom_cnfg.h` file has a ROM search list defined, the options `ro` and `lr` appear in the boot menu. If no search list is defined `N/A` appears in the boot menu. If an OS-9 bootfile is programmed into Flash in the address range defined in the port's `default.des` file, the system can boot and run from Flash.

`rom_cnfg.h` is located in the following directory:

```
MWOS\os9000\ARMV4\PORTS\ENP2505\ROM\ROMCORE
```

`ro`	ROM boot—the system runs from the Flash bank.
`lr`	load to RAM—the system copies the Flash image into RAM and runs from there.

Booting over a Serial Port via kermit

The system can download a bootfile in binary form over its serial port at speeds up to 115200 using the kermit protocol. The speed of this transfer depends of the size of the bootfile, but it usually takes at least three minutes to complete. Dots on the console will show the progress of the boot.

`ker`	kermit boot—the `os9kboot` file is sent via the kermit protocol into system RAM and runs from there.

Restart Booter

The restart booter enables a way to restart the bootstrap sequence.

q	quit—quit and attempt to restart the booting process.

Break Booter

The break booter allows entry to the system level debugger (if one exists). If the debugger is not in the system the system will reset.

`break`	break—break and enter the system level debugger rombug.

Sample Boot Session and Messages

Following is an example boot of the IXP1200 Network Processor Evaluation Board using the `lr` boot option.

```
OS-9 Bootstrap for the ARM (Edition 64)

Now trying to Override autobooters.

Press the spacebar for a booter menu


BOOTING PROCEDURES AVAILABLE ---------- <INPUT>

Boot embedded OS-9000 in-place -------- <bo>
Copy embedded OS-9000 to RAM and boot - <lr>
Load bootfile via kermit Download ----- <ker>
Enter system debugger ----------------- <break>
Restart the System -------------------- <q>

Select a boot method from the above menu: lr

Now searching memory ($00040000 - $001fffff) for an OS-9
Kernel...

An OS-9 kernel was found at $00040000
A valid OS-9 bootfile was found.
$
$ pciv

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:00:00  8086 1200 0017 2210 0b4001 03 00 6e 01 IXP1200
Processor [S]
000:01:00  8086 b555 0007 02b0 000000 02 00 6e 01 Prior to rev.
2.0 spec [S]
```

The Fastboot Enhancement

The Fastboot enhancements to OS-9 provide faster system bootstrap performance to embedded systems. The normal bootstrap performance of OS-9 is attributable to its flexibility. OS-9 handles many different runtime configurations to which it dynamically adjusts during the bootstrap process.

The Fastboot concept consists of informing OS-9 that the defined configuration is static and valid. These assumptions eliminate the dynamic searching OS-9 normally performs during the bootstrap process and enables the system to perform a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

Overview

The Fastboot enhancement consists of a set of flags that control the bootstrap process. Each flag informs some portion of the bootstrap code that a particular assumption can be made and that the associated bootstrap functionality should be omitted.

The Fastboot enhancement enables control flags to be statically defined when the embedded system is initially configured as well as dynamically altered during the bootstrap process itself. For example, the bootstrap code could be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources which would indicate different bootstrap requirements.

In addition, the Fastboot enhancement’s versatility allows for special considerations under certain circumstances. This versatility is useful in a system where all resources are known, static, and functional, but additional validation is required during bootstrap for a particular instance, such as a resource failure. The low-level bootstrap code may respond to some form of user input that would inform it that additional checking and system verification is desired.

Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. An entire 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within the set of data structures shared by the ModRom sub-components and the kernel. Hence, the field is available for modification and inspection by the entire set of system modules (high-level and low-level). Currently, there are six bit flags defined with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed below:

B_QUICKVAL

The `B_QUICKVAL` bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. This causes the CRC check on modules to be omitted. This option is a potential time saver, due to the complexity and expense of CRC generation. If a system has many modules in ROM, where access time is typically longer than RAM, omitting the CRC check on the modules will drastically decrease the bootstrap time. It is rare that corruption of data will ever occur in ROM. Therefore, omitting CRC checking is usually a safe option.

B_OKRAM

The `B_OKRAM` bit informs both the low-level and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on the defined RAM parameters. This allows system designers to specify a possible RAM range, which the system validates upon startup. Thus, the system can accommodate varying amounts of RAM. In an embedded system where the RAM limits are usually statically defined and presumed to be functional, however, there is no need to validate the defined RAM list. Bootstrap time is saved by assuming that the RAM definition is accurate.

B_OKROM

The B_OKROM bit causes acceptance of the ROM definition without probing for ROM. This configuration option behaves like the B_OKRAM option, except that it applies to the acceptance of the ROM definition.

B_1STINIT

The B_1STINIT bit causes acceptance of the first init module found during cold-start. By default, the kernel searches the entire ROM list passed up by the ModRom for init modules before it accepts and uses the init module with the highest revision number. In a statically defined system, time is saved by using this option to omit the extended init module search.

B_NOIRQMASK

The B_NOIRQMASK bit informs the entire bootstrap system that it should not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, some systems that have a well defined interrupt system (i.e. completely calmed by the sysinit hardware initialization code) and also have a requirement to respond to an installed interrupt handler during system startup can enable this option to prevent the ModRom and the kernel cold-start from disabling interrupts. This is particularly useful in power-sensitive systems that need to respond to “power-failure” oriented interrupts.

Note

Some portions of the system may still mask interrupts for short periods during the execution of critical sections.

B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization at all or systems that only require it for “power-on” reset conditions. Systems that only require parity initialization for initial “power-on” reset conditions can dynamically use this option to prevent parity initialization for subsequent “non-power-on” reset conditions.

Implementation Details

This section describes the compile-time and runtime methods by which the bootstrap speed of the system can be controlled.

Compile-time Configuration

The compile-time configuration of the bootstrap is provided by a pre-defined macro (BOOT_CONFIG), which is used to set the initial bit-field values of the bootstrap flags. You can redefine the macro for recompilation to create a new bootstrap configuration. The new over-riding value of the macro should be established by redefining the macro in the rom_config.h header file or as a macro definition parameter in the compilation command.

The rom_config.h header file is one of the main files used to configure the ModRom system. It contains many of the specific configuration details of the low-level system. Below is an example of how you can redefine the bootstrap configuration of the system using the BOOT_CONFIG macro in the rom_config.h header file:

```
#define BOOT_CONFIG (B_OKRAM + B_OKROM + B_QUICKVAL)
```

Below is an alternate example showing the default definition as a compile switch in the compilation command in the makefile:

```
SPEC_COPTS = -dNEWINFO –dNOPARITYINIT –dBOOT_CONFIG=0x7
```

This redefinition of the `BOOT_CONFIG` macro results in a bootstrap method that accepts the RAM and ROM definitions without verification, and also validates modules solely on the correctness of their module headers.

Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine what user-defined bootstrap procedure should be used. An example P2 module is shown below.

Note

If the override is performed in the `sysinit2()` function, the effect is not realized until after the low-level system memory searches have been performed. This means that any runtime override of the default settings pertaining to the memory search must be done from the code in the P2 module code.

```
#define NEWINFO
#include <rom.h>
#include <types.h>
#include <const.h>
#include <errno.h>
#include <romerrno.h>
#include <p2lib.h>

error_code p2start(Rominfo rinf, u_char *glbls)
{
    /* if switch or jumper setting is set… */
    if (switch_or_jumper == SET) {
        /* force checking of ROM and RAM lists */
        rinf->os->boot_config &= ~(B_OKROM+B_OKRAM);
    }
    return SUCCESS;
}
```

OS-9 Vector Mappings

This section contains the OS-9 vector mappings for the Intel IXP1200 Ethernet Evaluation platform.

The ARM standard defines exceptions 0x0-0x7. The OS-9 system maps these one-to-one. External interrupts from vector 0x6 are expanded to the virtual vector rage shown below by the `irqixp1200` module.

For More Information

See the ***IXP1200 Network Processor Programer's Reference*** for further information on individual sources.

Table 1-1 OS-9 IRQ Assignment for the IXP1200 Ethernet Evaluation Board

OS-9 IRQ #	ARM Function
0x0	Processor Reset
0x1	Undefined Instruction
0x2	Software Interrupt
0x3	Abort on Instruction Prefetch
0x4	Abort on Data Access
0x5	Unassigned/Reserved
0x6	External Interrupt

Table 1-1 OS-9 IRQ Assignment for the IXP1200 Ethernet Evaluation Board (continued)

OS-9 IRQ #	ARM Function
0x7	Fast Interrupt
0x8	Alignment error Form of Data abort

Table 1-2 IXP1200 Specific Functions

OS-9 IRQ #	IXP1200 Specific Function
0x40	MicroEngine 0, thread 0 (FBI block sources 0-28)
0x41	MicroEngine 0, thread 1
0x42	MicroEngine 0, thread 2
0x43	MicroEngine 0, thread 3
0x44	MicroEngine 1, thread 0
0x45	MicroEngine 1, thread 1
0x46	MicroEngine 1, thread 2
0x47	MicroEngine 1, thread 3
0x48	MicroEngine 2, thread 0
0x49	MicroEngine 2, thread 1
0x4a	MicroEngine 2, thread 2

Table 1-2 IXP1200 Specific Functions (continued)

OS-9 IRQ #	IXP1200 Specific Function
0x4b	MicroEngine 2, thread 3
0x4c	MicroEngine 3, thread 0
0x4d	MicroEngine 3, thread 1
0x4e	MicroEngine 3, thread 2
0x4f	MicroEngine 3, thread 3
0x50	MicroEngine 4, thread 0
0x51	MicroEngine 4, thread 1
0x52	MicroEngine 4, thread 2
0x53	MicroEngine 4, thread 3
0x54	MicroEngine 5, thread 0
0x55	MicroEngine 5, thread 1
0x56	MicroEngine 5, thread 2
0x57	MicroEngine 5, thread 3
0x58	Debug interrupt 0
0x59	Debug interrupt 1
0x5a	Debug interrupt 2
0x5b	CINT pin

Table 1-2 IXP1200 Specific Functions (continued)

OS-9 IRQ #	IXP1200 Specific Function
0x5c	Reserved (PCI block sources 0-32)
0x5d	Soft Interrupt
0x5e	Reserved
0x5f	Reserved
0x60	Timer 1
0x61	Timer 2
0x62	Timer 3
0x63	Timer 4
0x64	Reserved
0x65	Reserved
0x66	Reserved
0x67	Reserved
0x68	Reserved
0x69	Reserved
0x6a	Reserved
0x6b	Door Bell from host
0x6c	DMA channel 1

Table 1-2 IXP1200 Specific Functions (continued)

OS-9 IRQ #	IXP1200 Specific Function
0x6d	DMA channel 2
0x6e	PCI_IRQ_1 (External PCI interrupts)
0x6f	Reserved
0x70	DMA1 not busy
0x71	DMA2 not busy
0x72	Start BIST
0x73	Received SERR
0x74	Reserved
0x75	I20 inbound post_list
0x76	Power management
0x77	Discard timer expired
0x78	Data parity error detected
0x79	Received master abort
0x7a	Received target abort
0x7b	Detected PCI Parity error
0x7c	SRAM interrupt (SRAM block source)
0x7d	RTC (RTC block source)

Table 1-2 IXP1200 Specific Functions (continued)

OS-9 IRQ #	IXP1200 Specific Function
0x7e	SDRAM (SDRAM block source)
0x7f	UART (UART block source)

Fast Interrupt Vector (0x7)

The ARM4 defined fast interrupt (FIQ) mapped to vector 0x7 is handled differently by the OS-9 interrupt code and can not be used as freely as the external interrupt mapped to vector 0x6. To make fast interrupts as quick as possible for extremely time critical code, no context information is saved on exception (except auto hardware banking) and FIQs are never masked. This requires any exception handler to save and restore its necessary context if the FIQ mechanism is to be used. This requirement means that a FIQ handler’s entry and exit points must be in assembly, as the C compiler will make assumptions about context. In addition, no system calls are possible unless a full C ABI context save has been performed first. The OS-9 IRQ code for the SA1110 has assigned all interrupts as normal external interrupts. It is up to the user to re-define a source as an FIQ to make use of this feature.

Port Specific Utilities

Utilities for the IXP1200 Network Processor Evaluation Board are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS`

The following port specific utilities are included:

`dmppci`	peeks PCI device information
`pciv`	displays board PCI bus information
`pflash`	programs onboard Flash
`setpci`	pokes PCI device settings

dmppci

Show PCI Information

SYNTAX

```
dmppci <bus_number> <device_number>
       <function_number> {<size>}
```

OPTIONS

`-?`	Display help.

DESCRIPTION

`dmppci` displays PCI configuration information that is not normally available by other means, except programming, using the PCI library.

EXAMPLE

```
$ dmppci 0 5 0 0x40

    PCI DUMP Bus:0 Dev:5 Func:0 Size:64
    -----------------------------------

VID  DID  CMD  STAT CLASS  RV CS IL IP LT HT BI MG ML SVID SDID
---  ---- ---- ---- -----  -- -- -- -- -- -- -- -- -- ---- ----
11ad 0002 0007 0280 020000 20 00 6e 01 00 00 00 00 00 1385 f004

BASE[0]  BASE[1]  BASE[2]  BASE[3]  BASE[4]  BASE[5]  CIS_P    EXROM
-------- -------- -------- -------- -------- -------- -------- --------
54000001 7fffff00 00000000 00000000 00000000 00000000 00000000 00000000

Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
------------------------------------------------------
0000   ad 11 02 00 07 00 80 02 20 00 00 02 00 00 00 00
0010   01 00 00 54 00 ff ff 7f 00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00 00 00 00 00 85 13 04 f0
0030   00 00 00 00 00 00 00 00 00 00 00 00 6e 01 00 00
```

pciv PCI Configuration Space View

SYNTAX

```
pciv [<opts>]
```

OPTIONS

`-?`	Display help.
`-a`	Display base address information and size.
`-r`	Display PCI routing information.

DESCRIPTION

The `pciv` utility allows visual indication of the status of the PCIbus.

EXAMPLES

When using the `pciv` command with an IXP1200 board, the following information is displayed:

```
$ pciv
BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:00:00  8086 1200 0017 2210 0b4001 03 00 6e 01 IXP1200 Processor [S]
000:01:00  8086 b555 0007 02b0 000000 02 00 6e 01 Prior to rev. 2.0 spec [S]
```

The `pciv` command in the previous example reports configuration information related to specific hardware attached to the system.

```
DETAIL OF BASIC VIEW:
   BUS       : Bus Number
   DEV       : Device Number
   VID       : Vendor ID
   DID       : Device ID
   CLASS     : Class Code
   RV        : Revision ID
   IL        : Interrupt Line
   IP        : Interrupt Pin
   [S]       : Single function device
   [M]       : Multiple function device
```

When the `-a` option is used, address information is displayed along with the size of the device blocks in use.

The fields in the previous example are, from left to right, as follows:

- Prefetchable
- Memory Type
- Address Fields
- Actual Value Stored
- Type of Access
- Translated Access Address Used (shown on second line)
- Size of Block (shown on second line)

When the `-r` option is used, PCI-specific information related to PCI interrupt routing is displayed. If an ISA BRIDGE controller is found in the system, the routing information is used. The use of ISA devices and PCI devices in the same system requires interrupts to be routed either to ISA or PCI devices. Since ISA devices employ edge-triggered interrupts and PCI devices use level interrupts, the `EDGE/LEVEL` control information is also displayed. If an interrupt is shown as `LEVEL` with a PCI route associated with it, no ISA card can use that interrupt. This command also shows the system interrupt mask from the interrupt controller.

Note

ISA and PCI interrupts cannot be shared.

pflash — Program Strata Flash

Syntax

```
pflash [options]
```

Options

`-f[=]filename`	input filename
`-eu`	erase used space only (default)
`-ew`	erase entire Flash
`-ne`	do not erase Flash
`-b[=]addr`	specify base address of Flash (hex) for part identification (replaces `-r`,`-p0`,`-p1`)
`-s[=]addr`	specify write/erase address of Flash (hex) (defaults to base address)
`-i`	print out information on Flash
`-nv`	do not verify erase or write
`-q`	no progress indicator
`-ri`	ROM image, allows overwriting raw boot area of flash

Description

The pflash utility allows the programming of Intel Strata Flash parts. The primary use will be in the burning of the OS-9 ROM image into the on-board flash parts. This allows for booting using the lr/bo booters.

setpci

Set PCI Value

SYNTAX

```
setpci <bus> <dev> <func> <offset> <size{bwd}>
<value>
```

OPTIONS

`-?`	Display help.

DESCRIPTION

The `setpci` utility sets PCI configuration information that is not normally available by other means, other than programming with the PCI library. The `setpci` utility may also be used to read a single location in PCI space. The following parameters are included:

`<bus>`	= PCI Bus Number 0..255
`<dev>`	= PCI Device Number 0..32
`<func>`	= PCI Function Number 0..7
`<offset>`	= Offset value (i.e. command register offset = 4)
`<size>`	= Size b=byte w=word d=dword
`<value>`	= The value to write in write mode. If no value is included, the utility is in read mode.

EXAMPLES

```
$ setpci 0 7 0 0x10 d

PCI READ MODE
-------------

PCI Value.....0x7feff000 (dword) READ

PCI Bus.........0x00
PCI Device......0x07
PCI Function....0x00
PCI Offset....0x0010
$ setpci 0 7 0 0x10 d 0x1234500

PCI WRITE MODE
--------------

PCI Value.....0x01234500 (dword) WRITE

PCI Bus.........0x00
PCI Device......0x07
PCI Function....0x00
PCI Offset....0x0010
$ setpci 0 7 0 0x10 d

PCI READ MODE
-------------

PCI Value.....0x01234000 (dword) READ

PCI Bus.........0x00
PCI Device......0x07
PCI Function....0x00
PCI Offset....0x0010
```

Intel Work Bench Daemons

Intel Work Bench Daemons for the IXP1200 Network Processor Evaluation Board are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/ENP2505/CMDS/BOOTOBJS`

The following daemons are available as part of Intel Work Bench, the IXP1200 software development suite:

`ixp_engine`	System state deamon used to setup an OS-9 to `ixp_serv` interrupt interface.
`ixp_serv`	User state thread-based deamon that communicates with the Intel Work Bench to provide microcode load and debug support.

Note

Non-CSL versions of the daemons are located in the following directory:

`MWOS\os9000\ARMV4\PORTS\ENP2505\CMDS\NOCSL`

Dependencies

Note

There are incompatibilities between version 1.0 and 1.1 of the Intel Work bench. To deal with these incompatibilities a 1.0 version compatible daemon, `ixp_serv1_0`, is included. It is used in the same manner as `ixp_serv` but works with version 1.0 of the Intel Workbench.

The `ixp_serv` and `ixp_engine` daemons provide communication between the Windows host and the target system via RPC over an Ethernet interface. To use the daemons you must perform minimal configuration on both the StrongARM target system and on the Windows host system.

On the target system, your ROM image (`os9kboot`) must include a properly configured Ethernet interface.

For More Information

See the **Creating a ROM Image for the ENP-2505 Target System** section in **Chapter 1**.

At boot time, ensure that the network interface is running, along with the RPC portmapper service. The example in this section shows one variation of daemon startup.

On the Windows host system, you must start the Intel WorkBench daemon. After loading a project, set it to the hardware option. Choose Ethernet as your means of communication and set the IP address of your StrongARM target.

For More Information

Refer to **Appendix B: Running OS-9 and the Intel Workbench** for more information on using the Intel Workbench.

uclo — Load Microcode Object File

SYNTAX

```
uclo [<options>] [<microcode object files>]
```

OPTIONS

`-?`	Display help.

DESCRIPTION

The `uclo` utility loads a microcode object file into the IXP1200 microengines. The microcode object file must be in UOF format. (See the ***IXP1200 Network Processor Development Tool User's Guide*** for more information about generating and using UOF files.) The `uclo` utility causes the microengines to be initialized, but it does not start the IXP1200 microengines; this must be done by an application, driver, or other StrongARM code.

Example Daemon Start Up

```
OS-9 Bootstrap for the ARM (Edition 64)

Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Copy embedded OS-9000 to RAM and boot.
Now searching memory ($00040000 - $001fffff) for an OS-9
Kernel...

An OS-9 kernel was found at $00040000
A valid OS-9 bootfile was found.
$ mbinstall
$ ipstart
$ portmap &
+3
$ ixp_engine &
+5
$ ixp_serv &
+6
$
```

Appendix A: Board Specific Modules

This chapter describes the modules specifically written for the target board. It includes the following sections:

- **Low-Level System Modules**
- **High-Level System Modules**
- **Common System Modules List**

Low-Level System Modules

The following low-level system modules are tailored specifically for the Intel IXP1200 Evaluation platform. The functionality of many of these modules can be altered through changes to the configuration data module (`cnfgdata`).These modules are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/ROM`

`cnfgdata`	Contains the low-level configuration data.
`cnfgfunc`	Provides access services to the `cnfgdata` data.
`commcnfg`	Inits communication port defined in `cnfgdata`.
`conscnfg`	Inits console port defined in `cnfgdata`.
`ioixp1200`	Provides low-level serial services via the IXP1200 serial unit.
`portmenu`	Inits booters defined in the `cnfgdata`.
`romcore`	Provides board-specific initialization code.
`armtimr`	Provides low-level timer services via time base register.
`usedebug`	Initializes low-level debug interface to RomBug, SNDP, or none.
`ioixp1200`	Provides low level serial access to the 1200's UART.
`initext`	Initializes PCI bus devices.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the Intel IXP1200 Network Processor Evaluation Board and peripherals. Unless otherwise specified, each module is located in a file of the same name in the following directory:

`MWOS/OS9000/ARMV4/PORTS/ENP2505/CMDS/BOOTOBJS`

CPU Support Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`kernel`	Provides all basic services for the OS-9 system.
`cache`	Provides cache control for the CPU cache hardware. The `cache` module is in the file `cach1100`.
`fpu`	Provides software emulation for floating point instructions.
`ssm`	System Security Module—provides support for the Memory Management Unit (MMU) on the CPU.
`vectors`	Provides interrupt service entry and exit code. The `vectors` module is found in the file `vect110`.

System Configuration Module

The system configuration modules are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/INITS`

`dd`	Descriptor module with high level system initialization information.
`nodisk`	Descriptor module with high level system initialization information, but used in a diskless system.
`configurer`	Descriptor module with high level system (generated by the Wizard)

Interrupt Controller Support

The interrupt controller support module provides an extension to the vectors module by mapping the single interrupt generated by an interrupt controller into a range of pseudo vectors. The pseudo vectors are recognized by OS-9 as extensions to the base CPU exception vectors. See the **GPIO Usage** section for more information.

`irqixp1200`	P2module that provides interrupt acknowledge and dispatching support for the SA1200's pic (vector range 0x40-0x7d).

Real Time Clock

`rtcixp1200`	Driver that provides OS-9 access to the SA1200's on-board real time clock.

Ticker

`tkarm`	Driver that provides the system ticker based on the IXP1200's PCI timer.
`hcsub`	Subroutine module that provides a high speed timer interface used by the HawkEye Profiler

Generic I/O Support Modules (File Managers)

The generic I/O support modules are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`ioman`	Provides generic I/O support for all I/O device types.
`scf`	Provides generic character device management functions.
`rbf`	Provides generic block device management functions for the OS-9 format.
`pcf`	Provides generic block device management functions for MS-DOS FAT format.
`spf`	Provides generic protocol device management function support.
`pipeman`	Provides a memory FIFO buffer for communication.

Pipe Descriptor

The pipe descriptor is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/DESC`

`pipe`	Pipeman descriptor that provides a RAM-based FIFO, which can be used for process communication.

RAM Disk Support

The RAM disk driver is located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`ram`	RBF driver that provides a RAM-based virtual block device

RAM Descriptors

The RAM descriptors are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/DESC/RAM`

`r0`	RBF descriptor that provides access to a RAM disk.
`r0.dd`	RBF descriptor that provides access to a ram disk—with module name dd (for use as the default device).
`r1`	RBF descriptor that provides access to a 4Meg RAM disk for Flash burning.

Serial and Console Devices

`scixp1200`	SCF driver that provides serial support the SA1200's internal UART.

Descriptors for use with scllio

The `scllio` descriptors are located in the following directory:

```
mwos\os9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\DESC\
SCLLIO
```

`vcons/term`	Descriptor modules for use with `scllio` in conjunction with a low-level serial driver. Port configuration and set up follows that which is configured in `cnfgdata` for the console port. It is possible for `scllio` to communicate with a true low-level serial device driver like `io1100`, or with an emulated serial interface provided by `iovcons`.

For More Information

See the ***OS-9 Porting Guide*** and the ***OS-9 Device Descriptor and Configuration Module Reference*** for more information.

SPF Device Support

Network Configuration Modules

`inetdb`

`inetdb2`

`rpcdb`

Support for Communication Across the PCI Backplane

The PCI Backplane support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

The following boards are supported as targets:

- ENP-3511
- ENP-2505
- Intel Ethernet Eval

`spixhpc`	is the SPF driver to support backplane communications as a target. The descriptor files for `spixhpc` are located in the following directory: `MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF` `sppc0` is the SPF descriptor module for use as the target. `sppch` is the SPF descriptor module for use as the host.
`spixhpch`	is the SPF driver to support backplane communications as a host.

Common System Modules List

The following low-level system modules provide generic services for OS9000 Modular ROM. They are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS/ROM`

`bootsys`	Provides booter registration services.
`console`	Provides console services.
`dbgentry`	Inits debugger entry point for system use.
`dbgserv`	Provides debugger services.
`excption`	Provides low-level exception services.
`flshcach`	Provides low-level cache management services.
`hlproto`	Provides user level code access to `protoman`.
`llbootp`	Provides bootp services.
`llip`	Provides low-level IP services.
`llslip`	Provides low-level SLIP services.
`lltcp`	Provides low-level TCP services.
`lludp`	Provides low-level UDP services.
`llkermit`	Provides a booter that uses kermit protocol.
`notify`	Provides state change information for use with LL and HL drivers.
`override`	Provides a booter which allows choice between menu and auto booters.
`parser`	Provides argument parsing services.
`pcman`	Provides a booter that reads MS-DOS file system.

`protoman`	Provides a protocol management module.
`restart`	Provides a booter that causes a soft reboot of system.
`romboot`	Provides a booter that allows booting from ROM.
`rombreak`	Provides a booter that calls the installed debugger.
`rombug`	Provides a low-level system debugger.
`sndp`	Provides low-level system debug protocol.
`srecord`	Provides a booter that accepts S-Records.
`swtimer`	Provides timer services via software loops.

For More Information

For a complete list of OS-9 modules common to all boards, see the ***OS-9 Device Descriptor and Configuration Module Reference*** manual.

Appendix A: Running OS-9 and the Intel Workbench

This appendix provides a brief overview of using the Intel Workbench with an OS-9 system. It includes the following sections:

Overview

The OS-9 ROM image provides an interface with the Intel Core Software Library. This enables loading and debugging of microcode to the IXP1200 Developers Work Bench in a simulated environment or directly on the hardware. The simulated environment has more features than the hardware version and is more appropriate for detailed debugging.

The standard OS-9 image also contains the SPF daemons for Hawk debugging on the StrongARM core, the daemons for using the profiler, and most standard OS-9 utilities.

Intel Workbench

All IXP1200 development boards ship with the Intel Developers Workbench provided by Intel.

For More Information

The Intel IXP1200 Developers Workbench, which was shipped with your hardware, is also available on CD from the Intel Literature Center at the following URL:

`http://developer.intel.com`

You can also order by phone at 800-548-4725, 7am to 7pm CST. Outside U.S. please allow 2-3 weeks for delivery.

The Enhanced OS-9 for IXP1200 board-level solution provides the necessary daemons to enable integration of the Intel and OS-9 environments. The following procedures describe how to configure the environment and run a sample microcode project.

Note

The following procedures assume that you have completed the steps described in **Chapter 1**.

ENP-2505 System Configuration

Complete the following steps to configure the ENP-2505 board for use with the Intel Workbench. These steps assume you have established an Ethernet connection, as detailed in the **Optional Procedures** section of **Chapter 1**.

Step 1. Install the Intel Workbench. Installation requires a key, available from the Intel website at the following URL:

`http://developer.intel.com/design/network/products/software/sdk.htm`

Step 2. Run the sample project.

In the Hyperterminal window, the following messages display as the IXP1200 is booted up:

```
    OS-9 Bootstrap for the ARM (Edition 64)

    Now trying to Override autobooters.

    Press the spacebar for a booter menu

    Now trying to Copy embedded OS-9000 to RAM and boot.
    Now searching memory ($00040000 - $001fffff) for an OS-9 Kernel...

    An OS-9 kernel was found at $00040000

    A valid OS-9 bootfile was found.
```

Step 3. Start RPC by typing the following command at the OS-9 prompt: `portmap &`.

Step 4. Start the Intel Workbench demon1 by typing the following command at the OS-9 prompt: `ixp_engine &`.

Step 5. Start the Intel Workbench demon2 by typing the following command at the OS-9 prompt: `ixp_serv &`.

Step 6. Configure the Ethernet settings and check the network settings by typing the following two commands at the OS-9 prompt:

```
$ netstat -in
Name  Mtu   Network     Address                Ipkts Ierrs    Opkts Oerrs  Coll
lo0   1536  <Link>                                 4     0        4     0     0
lo0   1536  127         127.0.0.1                  4     0        4     0     0
enet0 1500  <Link>      0.0.0.0.0.0      267       0        0     0     0
enet0 1500  192.168      192.168.10.4             267     0        0     0     0
```

Note

The IP address used above is an example only. You must get your specific network information from your network administrator.

Running the Sample Project

The following example requires the Hawk and Profiler daemons to be running on the IXP1200 target system.

Step 1. Start the Hawk daemon by typing the following command at the OS-9 prompt (in the Hyperterminal window):

```
spfndpd &
```

Step 2. Start Profiler daemon by typing the following command at the OS-9 prompt:

```
spfnppd &
```

Step 3. From the Windows host system, select `Start --> Programs --> Intel IXP1200 --> Developers Work Bench`. The Intel Workbench opens in the foreground.

Step 4. From the Intel Workbench select `File -> Open Project`. The standard Windows file search box is displayed.

Step 5. Browse to and open the following file:

`IXP1200\Microcode\examples\bit_swiz.dwp`

The project is opened and the file tree is visible on the right (source macro script).

Step 6. Select `Debug->Hardware`. A check box goes to the hardware menu.

Step 7. Select `Hardware-->Options`. The **Hardware Options** screen is displayed.

Step 8. Click on **Connect via Ethernet** and enter your target IP address.

Step 9. Click `OK`.

Step 10. Click on the **Bug** picture on the right side of the screen. The Workbench screen changes, and the MicroEngine window opens at the bottom. You may have to click on **spool**.

Step 11. Expand the MicroEngine 0 by clicking `+`. Thread 0-0 through Thread 3-0 will be exposed.

Step 12. Click the green traffic sign to load and start executing the microcode. The arrow in the MicroEngine window will advance as code runs on different engines.

Workbench Interface Notes

- The button that resembles steps (labeled **HOP)** will single step after stopping.
- Double clicking on code can bring it to the foreground, and arrows will move through the code when it is being executed.
- Right clicking and holding brings up a breakpoint window. This can also be done from the debugger pull-down menu.
- This procedure can be run in the SIMULATOR by not choosing the two hardware options WB4/WB5. This gives you a practice run.

IXP1200 StrongARM Core Software Libraries

Functions for the Intel IXP1200 StrongARM core software libraries are located in the following directory:

```
MWOS/OS9000/ARMV4/PORTS/ENP2505/LIB/netapp.l
```

For More Information

For more information regarding the Intel IXP1200 libraries, refer to the ***IXP1200 Software Reference Manual*** and library source (`IXP1200\SA1_CoreLibs`) distributed with the IXP1200 Developer's Workbench.

Appendix A: Debuging with the Ethernet Router Application

Your ENP-2505 board may have been configured to include Ethernet router demonstration software. This is a factory demonstration that will stop running after eight hours. The following section explains how to re-create the bundle demonstration without the time limitation.

Using the Ethernet Router Bundle

Complete the following steps to re-create the Ethernet Router Bundle demonstration without a time limitation. Before you begin, however, make sure you have the following add-on products installed on your development system: EMANATE/Lite 15.3 SNMP for OS-9, IP Infusion OSPFv2 v1.1 for OS-9, IP Infusion RIPv1v2 v1.1 for OS-9, IP Infusion ZebOS Daemon v1.1 for OS-9, IXP1200 Forwarding Table Manager for OS-9, and Enhanced OS-9 for IXP1200.

Step 1. From the opening Configuration Wizard screen, under **Port Selection**, select `Radisys ENP-2505`. Under **Configuration Name**, type `ethroute_2505`. Click `OK`.

Step 2. This step is showing you which check boxes were pre-selected for you to include the proper routing software. After the Wizard has started, select `Configure -> Bootfile -> Network Configuration` from the menu.

Step 3. From the **Network Configuration** dialog, select the **SoftStax Options** tab. You should see the following check boxes selected: snmp, Zebos OSPF, Zebos RIP, FTM, and Routing Demo. Click `Cancel`.

Step 4. Select `Configure -> Build Image`. From the **Master Builder** dialog, click `Build`. After the bootfile has finished building, you can transfer the image to the ENP-2505 and program it into Flash.

Debugging with the 10/100 Mbit Ethernet Port

The ENP-2505 board contains one serial port and no Ethernet port for the StrongARM core processor. This makes it a challenge to use an IDE (such as Hawk) and a console application (such as mshell) at the same time when developing software.

However, to ease this situation, the ENP-2505 board contains four 10/100 Mbit Ethernet ports. Thus, if you are running a microcode subsystem (such as Ethernet MSL) on microengines that allow the sending of IP traffic to the core processor, it is possible to use one of these Ethernet ports as a debugging port. Below is the procedure for using the fourth microcode port on the ENP-2505 board as a general-purpose Ethernet port.

Note

This port can only be used for Ethernet debugging if you have installed the Ethernet Router bundle.

Step 1. From the command line for the target machine, type in the following command:

```
$ route delete default
```

The following message is displayed:

```
delete net default: gateway
```

Step 2. Type the following commands:

```
$ ifconfig slip0 unbind
$ ifconfig enet3 <ip address> binding /spixp3/enet
```

Step 3. Type the following command to set the default gateway address: (This step is optional.)

```
$ route add default 208.252.116.254
```

Step 4. If the Forwarding Table Manager is not already running, type the following command to start it:

```
$ ftm&
```

You should now have a functioning network connection on the bottom microcode port.

Step 5. You now need to ftp three debuger modules to the target board and load them. To do this, run the pushit script (which contains the files `spfndpd`, `spfndpdc`, and `ndpio`), located in the following directory:

```
\MWOS\OS9000\ARMV4\CMDS
```

Step 6. Load the modules and start the debugger daemon by typing the following commands:

```
$ load -ld spf* ndpio
$ spfndpd <>>>/nil &
```

You are now ready to initiate a hawk debug session.