OS-9 for IQ80310 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

Copyright and publication information

This manual reflects version 3.2 of Enhanced OS-9 for Intel XScale.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

Table of Contents

Chapter 1: Installing and Configuring OS-9

This chapter describes installing and configuring OS-9 on the IQ80310 Evaluation Board. It includes the following sections:

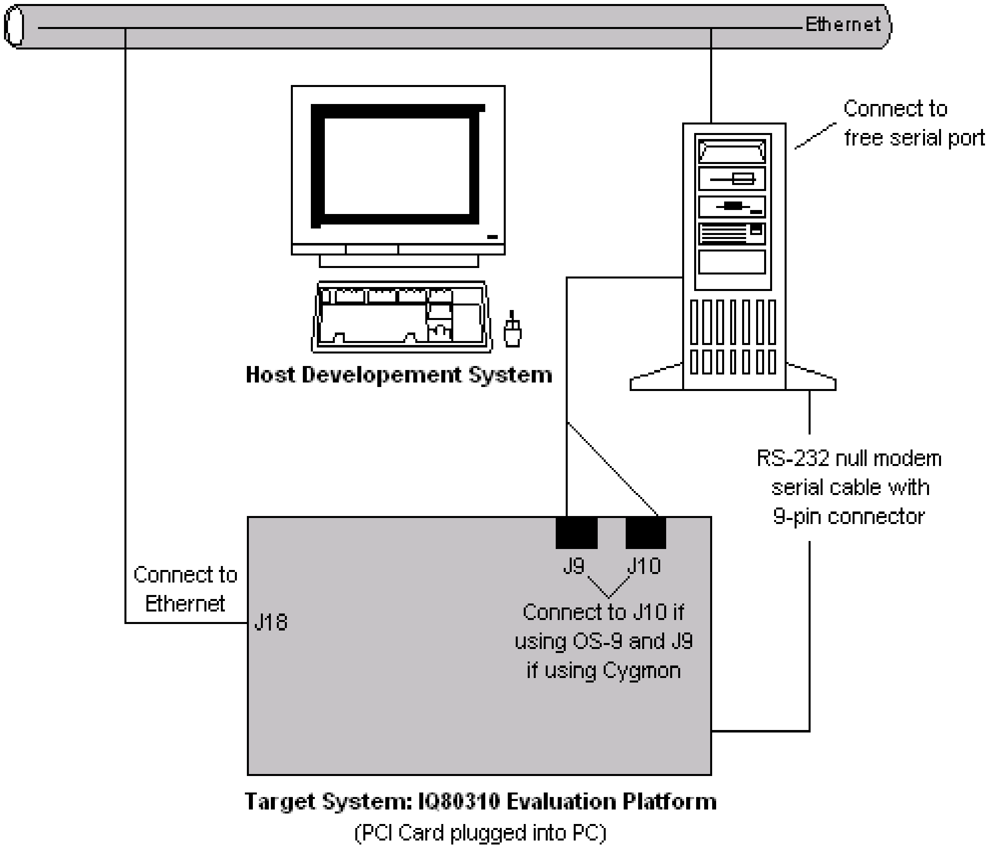
RadiSys.

MICROWARE SOFTWARE

Development Environment Overview

Figure 1-1 shows a typical development environment for the IQ80310 Evaluation System. The components shown include the minimum required to enable OS-9 to run on the IQ80310.

Figure 1-1 IQ80310 Development Environment



Requirements and Compatibility

Note

Before you begin, install the ***Enhanced OS-9 for Intel® XScale™*** 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 your 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:

- PC with DOS, Windows 95, or Windows 98
- an RS-232 null modem serial cable with 9-pin connectors
- access to an Ethernet network

OS-9 Architecture

The source and example code and makefiles for Enhanced OS-9 for Intel® XScale™ are located in the following directory. The directory structure is shown in Figure 1-2.

```
\mwos\OS9000\ARMV5\PORTS\IQ80310
```

Figure 1-2 OS-9 for IQ80310 Directories

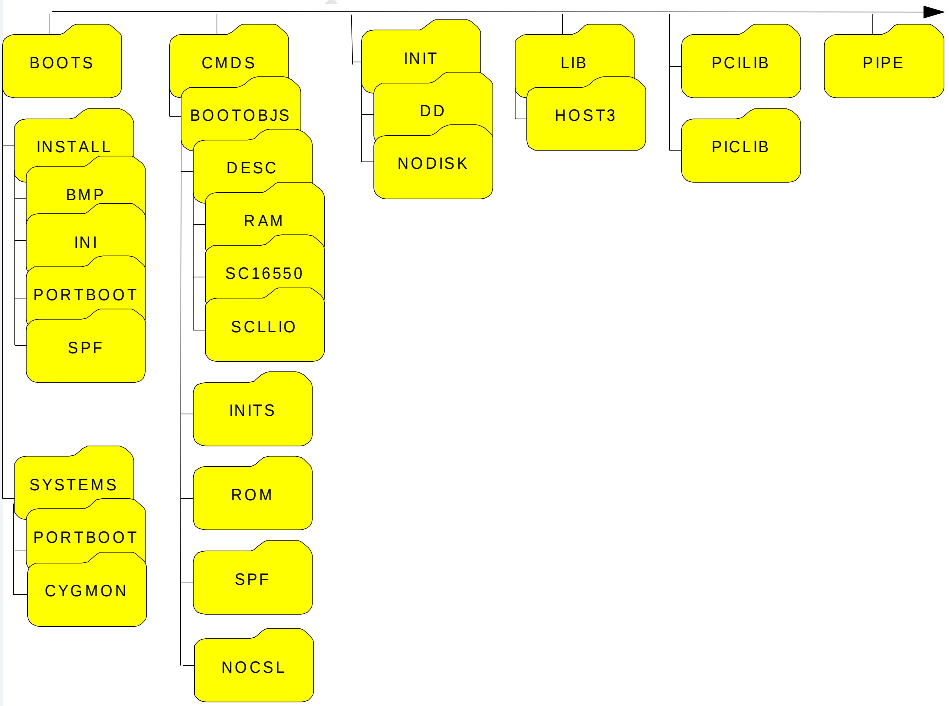
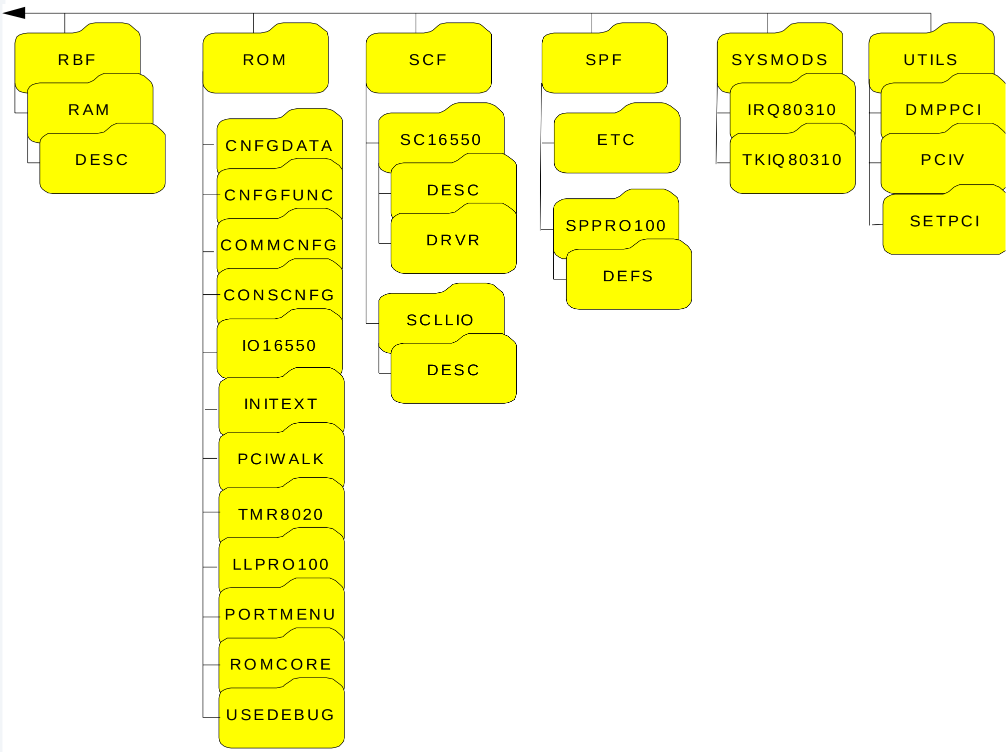


Figure 1-2 OS-9 for IXP1200 Directories (continued)



Target Hardware Setup

Programming the Flash

Below are Intel®'s instructions for programming the Flash. If you are using the Configuration Wizard to build your image for programming Flash, you can find the default OS-9 files in the following location:

```
MWOS/OS9000/ARMV5/PORTS/IQ80310/BOOTS/INSTALL/PORTBOOT/ROM
```

For the test image provided by Microware, refer to the following file:

```
MWOS/OS9000/ARMV5/PORTS/IQ80310//BOOTS/SYSTEMS/PORTBOOT/ROM
```

The following steps are required to program the Flash ROM boot sectors:

Step 1. Set switch S3-1 and S3-2 to the ON position.

Step 2. Reset the board by cycling power on the workstation.

Step 3. Run the Intel® Flash Recovery Utility (FRU) to program the Flash ROM boot sectors.

For More Information

For more information on the FRU, refer to Appendix I of the ***Intel® IQ80310 Evaluation Platform Board Manual***.

Step 4. Set switch S3-1 and S3-2 to the OFF position.

Step 5. Reset the board by cycling power on the workstation.

Jumper Settings

All jumper settings can remain as shipped from the factory. The jumpers are used only in conjunction with programming the flash.

For More Information

See the ***Intel® IQ80310 Evaluation Platform Board Manual*** for more information about the hardware. The user's guide ships with your hardware. It 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.

Connecting the Target to the Host

For More Information

For a detailed view of the host-target setup, refer to **Figure 1-1**.

To connect the target to the host machine, complete the following steps:

Step 1. Connect the target system to an Ethernet network.

Step 2. Connect the target system to the host system using the RS-232 null modem serial cable provided with your board.

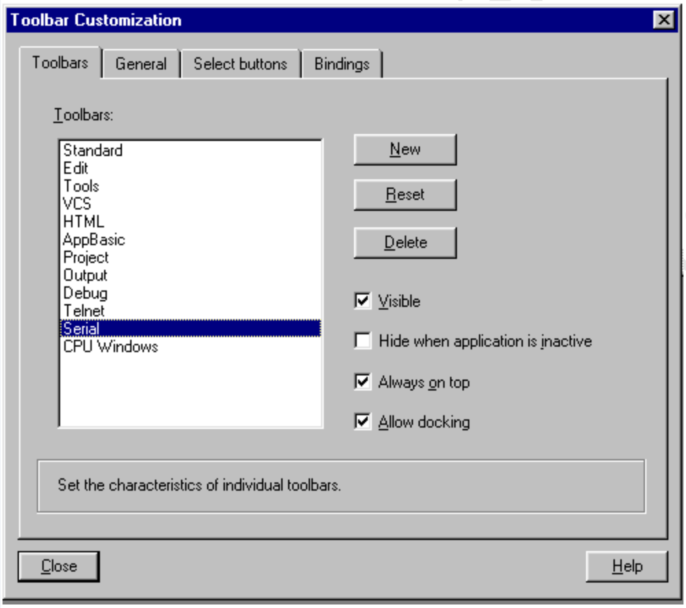
Verifying the Serial Connection

You may want to boot to the Cygmon/Redboot prompt to verify that your serial cable is connected properly. To do this, complete the following steps:

Step 1. From the desktop, click `Start` and select `Programs` -> `Microware` -> `Enhanced OS-9 for Intel XScale` -> `Hawk` to start the Microware Hawk IDE.

Step 2. If the **Serial** console window is not open, it can be opened from the **Toolbar Customization** dialog (shown in **Figure 1-3**). (Select `Tools` -> `Customize` -> `Toolbars` to open the **Toolbar Customization** dialog.)

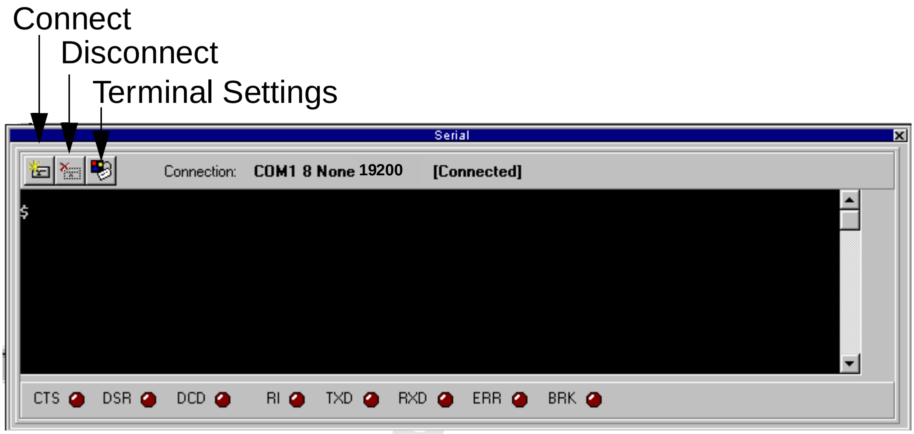
Figure 1-3 Toolbar Customization dialog box



Step 3. Once the **Toolbar Customization** dialog box is open, select `Serial` in the **Toolbars** list box.

Step 4. Click the `Visible` check box, then click the `Close` button. The **Serial** console window opens. (The **Serial** window can be seen in **Figure 1-4**.)

Figure 1-4 Hawk Serial Console Window



Step 5. Once you have the serial console window open, click on the `Connect` button in the upper left corner of the serial console window. The **Com Port Options** dialog box appears.

Step 6. Click on the `OK` button because the default settings are correct. The message *[Not Connected]* should change to *[Connected]*.

Step 7. Apply power to the board. The Cygmon/Redboot monitor boots the board. The display looks similar to **Figure 1-5**.

Figure 1-5 Cygmon Initial Screen

```
Cygmon, the Cygnus ROM monitor.
Copyright (C) 1997, 1998, 1999, 2000 Cygnus Solutions

Version: release 2.0D
This image was built on Thu Nov 2 12:51:01 EST 2000

CPU: Intel® 80310
Board: IQ80310
ARM is a Registered Trademark of Advanced RISC Machines
Limited. Other Brands and Trademarks are the property of
their respective owners.
DRAM Size: 0x2000000
Num PCI Devices / Functions: 3 / 3
cygmon>
```

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

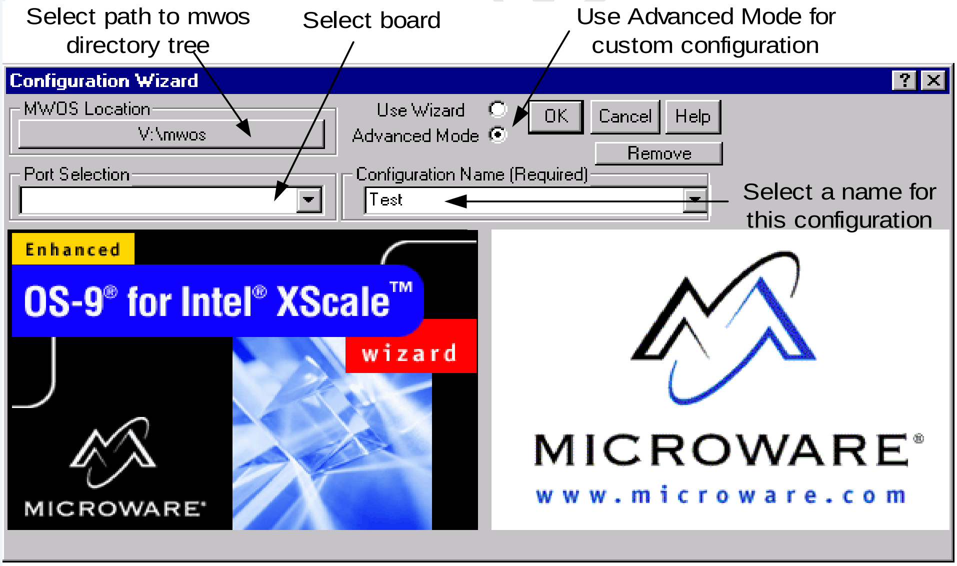
The Configuration Wizard is used to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The Configuration Wizard can also be used to modify an existing image. The Wizard is automatically installed on your host PC during the Enhanced OS-9 installation process.

To use the configuration wizard, perform the following steps:

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

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

Figure 1-6 Intel® XScale™ Configuration Wizard



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 `Advanced Mode` and click `OK`. The Main Configuration window is displayed. Advanced 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 IQ80310 Target System

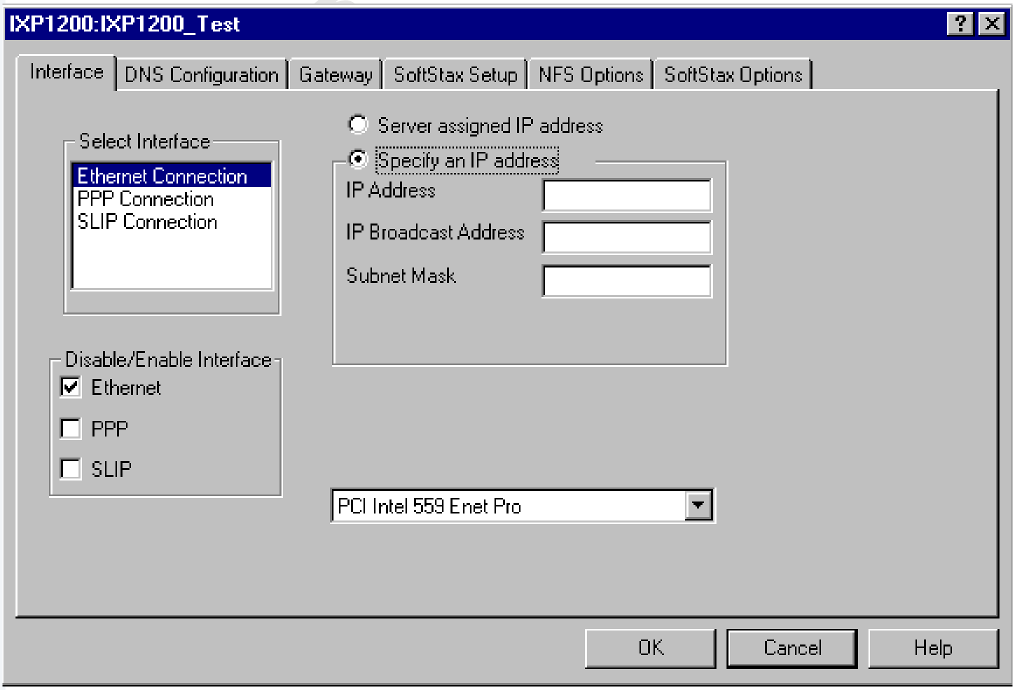
The OS-9 ROM image shipped with Enhanced OS-9 for Intel® XScale™ 3.0 include the minimum software required to boot the target to an OS-9 prompt. The following procedure describes using the configuration wizard to modify the bootfile and add functionality to your system.

Two of the most common modifications are described below. These include adding networking functionality and enabling a host/target connection via Hawk, the Microware integrated development environment.

To configure your system for networking and add Hawk functionality, complete the following steps:

Step 1. From the Wizard's main configuration window, select `Configure -> Bootfile -> NetWork Configuration`. The following dialog window should appear:

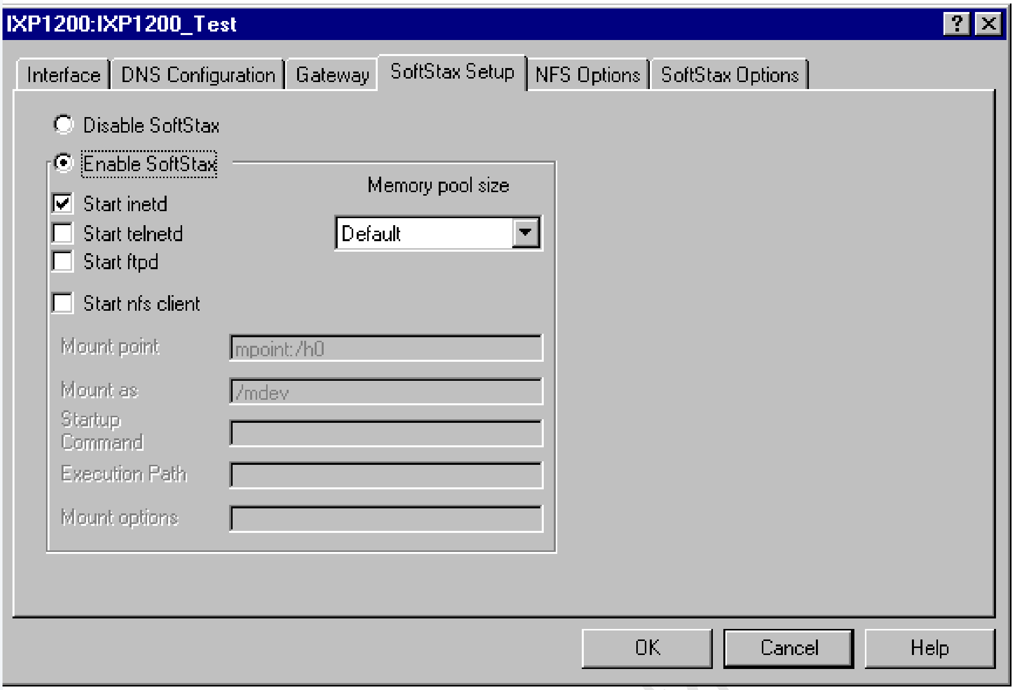
Figure 1-7 Network Configuration—Interface Tab



Step 2. Configure your system as shown in **Figure 1-7**. The **IP Address**, **IP Broadcast Address**, and **Subnet Mask** addresses must be obtained from your network administrator.

Step 3. Select the **SoftStax Setup** tab. The following dialog window appears:

Figure 1-8 Network Configuration—SoftStax Setup Tab



Step 4. Configure your system as shown in **Figure 1-8**.

Step 5. Leave the other **Network Configuration** options at the default settings. Click `OK`.

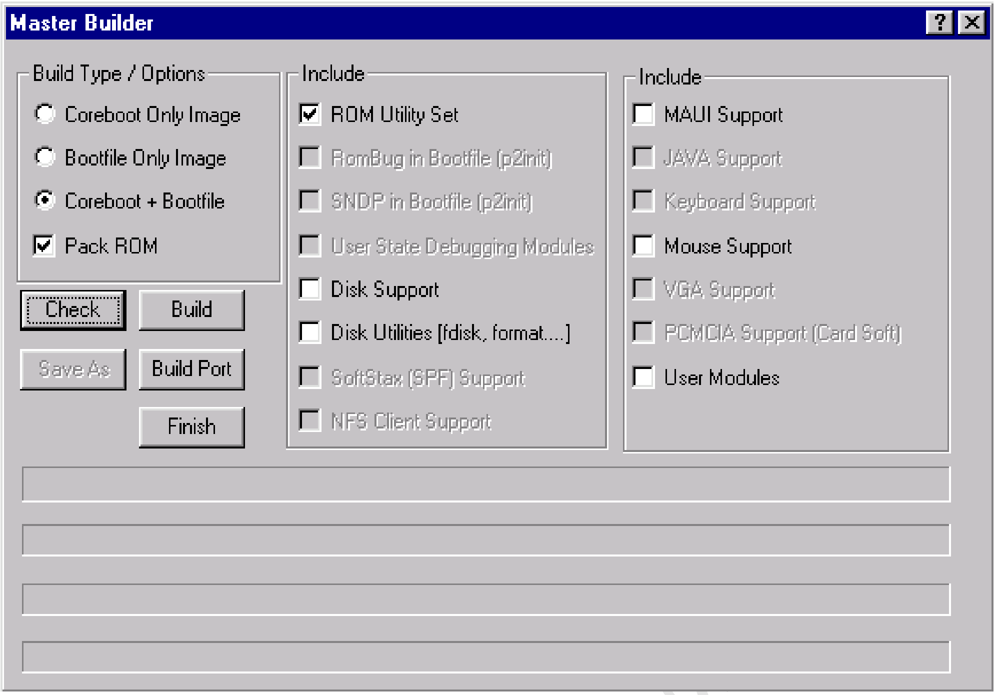
Note

Other **Network Configuration** options can be changed in this dialog according to your specific requirements and your network.

Step 6. Select `Configure -> Bootfile -> Disk Configuration`. From the **Init Options** tab, select the `/dd` radio button in the **Initial Device Name** field. This automatically enables networking at system start up.

Step 7. Select `Configure -> Build Image`. The **Master Builder** dialog window appears.

Figure 1-9 Master Builder Dialog Window



Step 8. Configure your Master Builder options as shown in **Figure 1-9**.

Step 9. Click `Build`. A bootfile image is created to be placed on the target. The bootfile image, `bootfile`, is stored in the following default location:

```
\mwos\OS9000\ARMV5\PORTS\IQ80310\BOOTS\INSTALL\PORTBOOT\
```

Note

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

Chapter 2: Board Specific Reference

This chapter contains porting information that is specific to the Intel® IQ080310 reference board. It includes the following sections:

- **Boot Options**
- **The Fastboot Enhancement**
- **OS-9 Vector Mappings**
- **GPIO Usage**
- **Port Specific Utilities**

For More Information

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

RadiSys.
MICROWARE SOFTWARE

Boot Options

The default boot options for the IQ80310 Evaluation board are listed below. The boot options can be selected by hitting the space bar during the IQ80310 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\ARMV5\PORTS\IQ80310\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 `bo` 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\ARMV5\PORTS\IQ80310\ROM\ROMCORE
```

`bo`	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 from Ethernet

The system can boot using the bootp protocol over Ethernet with the eb option.

eb	Ethernet boot. Ethernet will use the bootp protocol to transfer a bootfile into RAM and the systems 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 `bootfile` 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

Below is an example boot of the `IQ80310` Evaluation board using the `lr` boot option. (J10 is the default console for OS-9.)

```
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>
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.
$ mdir

Current Module Directory



alias        attr     break      cache     copy
csl          date     deiniz     del       deldir
devs         dir      dmppci     dump      echo
events       fpu      free       ident     init
iniz         ioman    irq80310   irqs      kermit
kernel       link     list       load      makdir
mdir         mfree    mshell     nil       null
p2init       pciv     pd         pipe      pipeman
printenv     procs    qsort      r0        r1
ram          rbf      save       sc16550   scf
setime       setpci   sleep      t2        term
tkiq80310    unlink   vectors    xmode
```

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® `IQ80310` Evaluation platform.

The ARM standard defines exceptions 0x0-0x7; the OS-9 system maps these exceptions one-to-one. External interrupts from vector 0x6 are expanded to the virtual vector rage shown below by the `irq80310` module.

Table 2-1 OS-9 IRQ Assignment for the IQ80310 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
0x7	Fast Interrupt
0x8	Alignment error Form of Data abort

Table 2-2 IQ80310 Specific Functions

OS-9 IRQ #	IQ80310 Specific Functions
0x06	External interrupt
0x07	Fast interrupt
0x70	Performance monitoring unit interrupt
0x71	Bus control unit interrupt
0x72	DMA channel 0 interrupt
0x73	DMA channel 1 interrupt
0x74	DMA channel 2 interrupt
0x75	Performance monitor interrupt
0x76	Application accelerator interrupt
0x77	I2C interrupt
0x78	Messaging unit interrupt
0x79	Memory controller error
0x7a	Primary ATU error
0x7b	Secondary ATU error
0x7c	Primary bridge error
0x7d	Secondary bridge error
0x7e	DMA channel 0 error

Table 2-2 IQ80310 Specific Functions (continued)

OS-9 IRQ #	IQ80310 Specific Functions
0x7f	DMA channel 1 error
0x80	DMA channel 2 error
0x81	Messaging unit interrupt or error
0x82	Application accelerator unit error
0x83	Core interface unit error
0x84	PLD timer interrupt
0x85	Ethernet interrupt
0x86	com1 interrupt
0x87	com2 interrupt
0x88	PCI INTD interrupt

Fast Interrupt Vector (0x7)

The ARM-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.

GPIO Usage

There are eight GPIO pins on the IQ80310 board.

For More Information

For more information on GPIO pins, refer to Chapter 13 of the ***Intel® 80312 I/O Companion Chip Developer's Manual***.

Port Specific Utilities

The following port specific utilities, located in `MWOS/OS9000/ARMV5/PORTS/IQ80310/CMDS`, are supported with the IQ80310 board:

`dmppci`	peeks PCI device information
`pciv`	displays board PCI bus information
`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
```

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 -a

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:00:00  8086 1200 0017 2200 0b4001 00 00 6e 0e
(NC) [32-bit] base_addr[0] = 0x40000008  PCI/MEM 0x60000008 Size = 0x00100000
(C)  [32-bit] base_addr[1] = 0x50000001  PCI/IO  0x54000000 Size = 0x00000080
(NC) [32-bit] base_addr[2] = 0xc0000008  PCI/MEM 0xe0000008 Size = 0x02000000
IXP1200 Processor [S]

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:05:00  11ad 0002 0007 0280 020000 20 00 6e 01
(C)  [32-bit] base_addr[0] = 0x54000001  PCI/IO  0x54000000 Size = 0x00000100
(C)  [32-bit] base_addr[1] = 0x7fffff00  PCI/MEM 0x7fffff00 Size = 0x00000100
Network Controller [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.

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
```

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® IQ80310 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/ARMV5/PORTS/IQ80310/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`.
`io16550`	provides low-level serial services via the 16550 serial unit.
`llpro100`	provides low-level Ethernet services via on-board Intel® 82559.
`portmenu`	inits booters defined in the `cnfgdata`.
`romcore`	provides board-specific initialization code.
`tmr80200`	provides low-level timer services via time base register.
`usedebug`	initializes low-level debug interface to RomBug, SNDP, or none.
`pciwalk`	initializes PCI bus devices.
`initext`	contains user modifiable romcore extension.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the Intel® IQ80310 Evaluation board and peripherals. Unless otherwise specified, each module is located in a file of the same name in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IQ80310/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 `cach80200`.
`fpu`	provides software emulation for floating point instructions.
`ssm`	is a System Security Module, which 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/ARMV5/PORTS/IQ80310/CMDS/BOOTOBJS/INITS`

`dd`	is a descriptor module with high level system initialization information.
`nodisk`	is a descriptor module with high level system initialization information, but used in a diskless system.
`configurer`	is a 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.

`irq80310`	is a P2module that provides interrupt acknowledge and dispatching support for the 80312's pic (vector range 0x70-0x88).

Ticker

`tkiq80310`	is a driver that provides the system ticker based on the 80310's PLD timer.
`hcsub`	is a 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/ARMV5/PORTS/IQ80310/CMDS/BOOTOBJS/DESC`

`pipe`	is a 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`	is an RBF driver that provides a RAM-based virtual block device

RAM Descriptors

The RAM descriptors are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IQ80310/CMDS/BOOTOBJS/DESC/RAM`

`r0`	is an RBF descriptor that provides access to a RAM disk.
`r0.dd`	is an RBF descriptor that provides access to a ram disk—with module name dd (for use as the default device).

Serial and Console Devices

`sc16550`	is an SCF driver that provides serial support the 16550 UART.

Descriptors for use with scixp1200

`term1/t1`	is a descriptor modules for use with `sc16550`. IXP1200 Board header: J10 Default Baud Rate: 19200 Default Parity: None Default Data Bits: 8 Default Handshake: XON/XOFF
term2/t2	is a descriptor modules for use with `sc16550`. IXP1200 Board header: J9 Default Baud Rate: 19200 Default Parity: None Default Data Bits: 8 Default Handshake: XON/XOFF
`scllio`	is an SCF driver that provides serial support via the polled low-level serial driver.

Descriptors for use with scllio

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

`mwos\os9000\ARMV5\PORTS\IQ80310\CMDS\BOOTOBJS\DESC\SCLLIO`

`vcons/term`	are 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 `io16550`, 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

PCI Support for on-board Intel® Ethernet Pro

The Intel® Ethernet Pro support module is located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IQ80310/CMDS/BOOTOBJS/SPF`

`sppro100`	SPF driver to support ethernet for a Intel® Ethernet Pro PCI.

sppro100 Descriptors

`sppr0`	SPF descriptor module for use with on-board 82559.

Network Configuration Modules

`inetdb`

`inetdb2`

`rpcdb`

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.
`dbgserve`	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.

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:____

Company:_______________________________________

Phone:____

Fax:_____________________Email:__________________

Product Name:

Description of Problem:

Host Platform_____________________________________

Target Platform___________________________________

RadiSys.

MICROWARE SOFTWARE