Home

OS-9® for PID7T Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com
Revision A • July 2006

Table of Contents

Chapter 1: Installing and Configuring OS-9®

This chapter describes installing and configuring OS-9® on the ARM LTD ARM7TDMI Microprocessor Reference Platform (PID7T). It includes the following sections:

Requirements and Compatibility

Host Hardware Requirements (PC Compatible)

Your host PC should have the following installed:

- Windows 95, 98, ME, 2000, or NT
- an Ethernet network card
- a PCMCIA card reader/writer
- the recommended hard drive storage for your particular operating system
- the recommended memory capacity for your particular operating system

Host Software Requirements (PC Compatible)

Your host PC should have a terminal emulation program (such as Hyperterminal that comes with Microsoft Windows 95, Windows 98, and Windows NT).

Target Hardware Requirements

Your reference board requires the following hardware:

- enclosure or chassis with power supply
- an RS-232 null modem serial cable

For More Information

Refer to your reference board’s hardware documentation for information on hardware preparation and installation, operating instructions, and functional descriptions prior to installing and configuring OS-9.

Target Hardware Setup

In order to operate properly, the PID7T must have the switch settings listed in Table 1-1.

For More Information

For more information, refer to the PID7T Development Card manual, included with your hardware.

Table 1-1 PID7T Switch Settings

Jumper	Position	Comment
LK4	open	Little-Endian
LK7	open	
LK8	closed	Set Edge interrupt for switch
LK9	open	
LK10	closed	Enable abort switch interrupt
LK18	closed	Boot from ROM (ROM at 0).
LK17	open	

Table 1-2 PID7T Switch Settings

Jumper Block	Position 1-x	Comment
LK6	closed, open, open, open	Boot from U12 EPROM, 512k, speed=120ns, width=8bit
LK16	open, closed, closed, closed	
LK11	(all open 1-8)	Parallel port defines
S1	off, off, on, on	20Mhz operation
S2	on,on,on,off,on,on, on,off	SRAM settings (change as desired)
S3	off,off,off,off	

SLOT A Must be populated with a 4meg or larger DRAM part.

Programming the Flash Devices

The on-board Flash part provided with the board must be reprogrammed with an OS-9 bootfile containing the necessary low-level modules to boot from Flash, ATA, or using bootp.

An example OS-9 bootfile, called `coreboot`, is located in the following directory; this example provides minimum functionality:

```
\MWOS\OS9000\ARMV4\PORTS\PID7T\BOOTS\SYSTEMS\PORTBOOT
```

Note

When reprogramming the on-board Flash, it is recommended that you obtain and use a separate Flash part for this procedure. The original Flash part should be removed and saved.

You can add functionality by creating your own coreboot file using the Configuration Wizard and an EPROM programmer. For example, if you want to use ROM Ethernet services such as System State Debugging, you must create a new coreboot image. The coreboot image that was shipped with the reference board does not allow you to perform System State Debugging because the IP address in Flash ROM is set to “0.0.0.0”.

Connecting the Target to the Host

Connect an RS-232 null modem cable from the reference board to the serial port of a Windows 95, Windows 98, or Windows NT system.

Step 1. Connect the serial cable to the PL4 connector on the reference board. The PL4 connector is serial port 1.

Step 2. Connect the other end of the serial cable to the Host PC.

Step 3. On the Windows desktop, click on the `Start` button and select `Programs -> Accessories -> Hyperterminal`.

Step 4. Open Hyperterminal and enter a name for your Hyperterminal session.

Step 5. Select an icon for the new Hyperterminal session. A new icon is created with the name of your session associated with it. The next time you want to establish the same session, follow the directions in Step 3 and look for the icon you created in Step 4.

Step 6. Click `OK`.

Step 7. In the **Phone Number** dialog, go to the **Connect Using** box and select the communications port to be used to connect to the reference board.

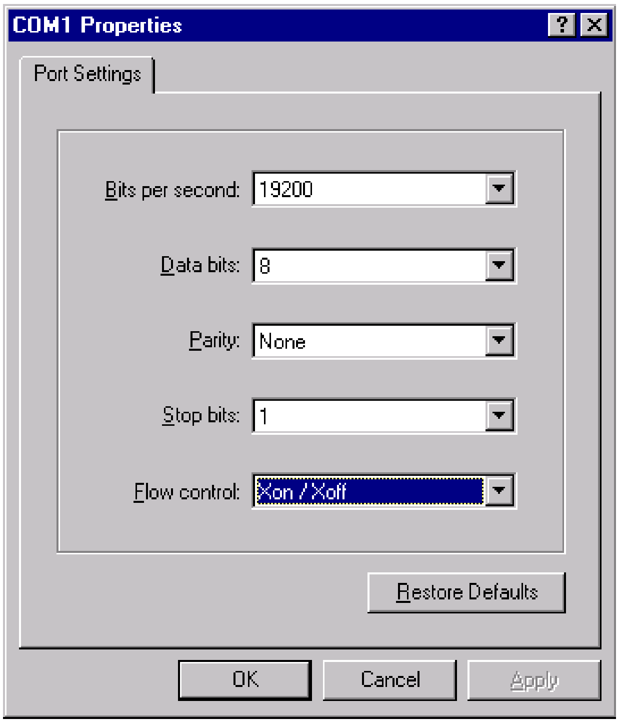
The port selected is the same port that you connected to the serial cable from the reference board.

Step 8. Click `OK`.

Step 9. In the **Port Settings** tab, enter the following settings:

```
Bits per second = 19200
Data Bits = 8
Parity = None
Stop bits = 1
Flow control = XOn/XOff
```

Figure 1-1 Port Settings



Bits per second: 19200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: Xon / Xoff

Step 10. Click OK. A connection should be established.

Note

If the word *connected* dows not appear in the lower left corner of the window, click `Call -> Connect` to establish the connection.

Step 11. Apply power to the board. The OS-9 bootstrap message is displayed.

Building the OS-9 ROM Image

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 can vary from system to system depending on hardware capabilities and user requirements.

To simplify the process of loading and testing OS-9, the ROM image 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 Microware OS-9 installation process.

PID7T Valid Configurations

There are several valid configurations for generating a ROM image for the PID7T target board. Each of these configurations uses a combination of the OS-9 low-level and high-level systems. The low-level system is typically stored on a small boot part (~128Kb) of Flash. The high-level system is usually stored on a larger Flash part or on other sources such as an ATA-PCMCIA card, or on another machine (which requires booting via bootp). For the PID7T, you can create boots of the following configurations:

- a ROM image on the 512k FLASH part
- a low-level system on the 512k/128k FLASH part

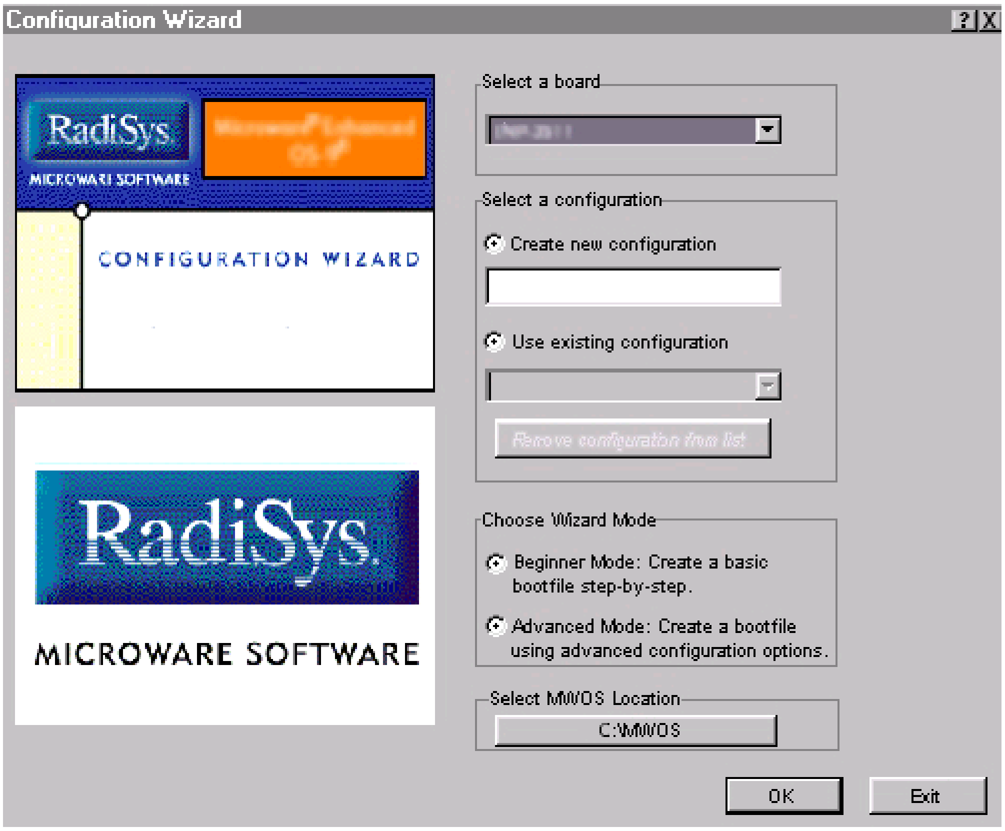
  (This is a high-level system that either exists on the PCMCIA ATA card or is downloaded from an Ethernet BootP server or serial port)

Starting the Configuration Wizard

The Configuration Wizard is the application used to build the coreboot, bootfile, or ROM image. To start the Configuration Wizard, perform the following steps:

Step 1. From the Windows desktop, select `Start` -> `RadiSys` -> `Microware OS-9 for <product>` -> `Configuration Wizard`. You should see the following opening screen:

Figure 1-2 Configuration Wizard Opening Screen

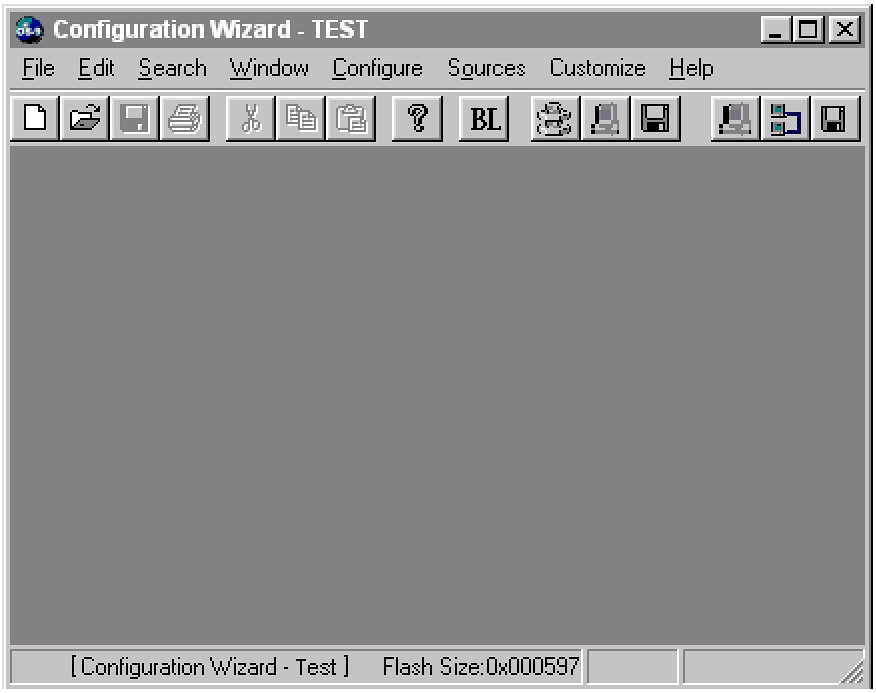


Step 2. Select your target board from the **Select a board** pull-down menu.

Step 3. Select the `Create new configuration` radio button from the **Select a configuration** menu and type in the name you want to give your ROM image in the supplied text box. This names your new configuration, which can later be accessed by selecting the **Use existing configuration** pull down menu.

Step 4. Select the `Advanced Mode` radio button from the **Choose Wizard Mode** field and click `OK`. The Wizard's main window is displayed. This is the dialog from which you will proceed to build your image. An example is shown in **Figure 1-3**.

Figure 1-3 Configuration Wizard Main Window



[Configuration Wizard - Test]   Flash Size:0x000597

Creating a Coreboot Image Using the Flash Device

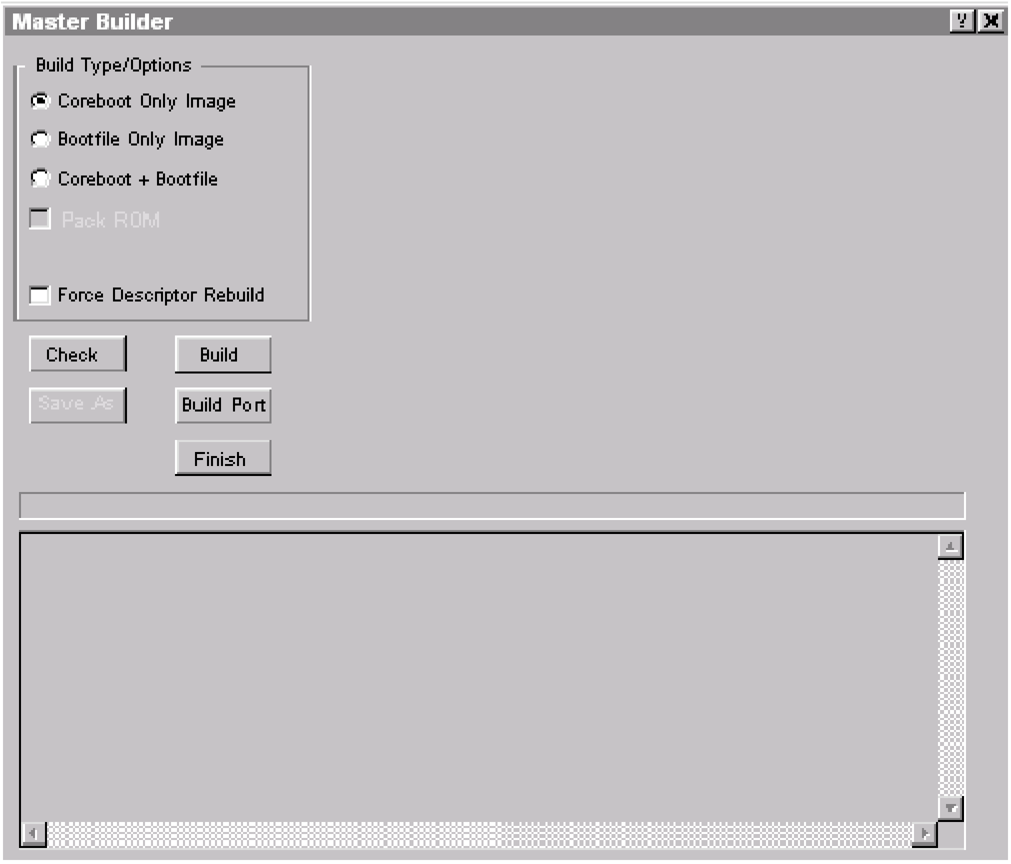
The following procedure describes creating a new coreboot image. When you are done creating the coreboot image, please refer to your EPROM programmer's instructions to learn how to load the coreboot image into the EPROMS.

Step 1. Open the Configuration Wizard to the main window.

Step 2. From the Wizard's main menu, select `Configure -> Build Image`. This displays the **Master Builder** window.

Step 3. From the Master Builder window, select the Coreboot Only Image radio button. The following window is displayed:

Figure 1-4 Master Builder Window



Step 4. Select the Build button. The coreboot image is built according to your configuration settings.

Step 5. Once the image is finished building, click Save As to save it to a directory of your choosing. If you do not have that directory on the drive, you can create it.

Step 6. Transfer the coreboot image to the EPROMs with the EPROM programmer. You will need to follow the documentation for the EPROM programmer to complete this step.

Note

Make sure the Secure Option is selected when burning Flash parts.

Building the Bootfile Image

To use the Configuration Wizard to build a bootfile image, perform the following steps:

Step 1. Apply power to your board.

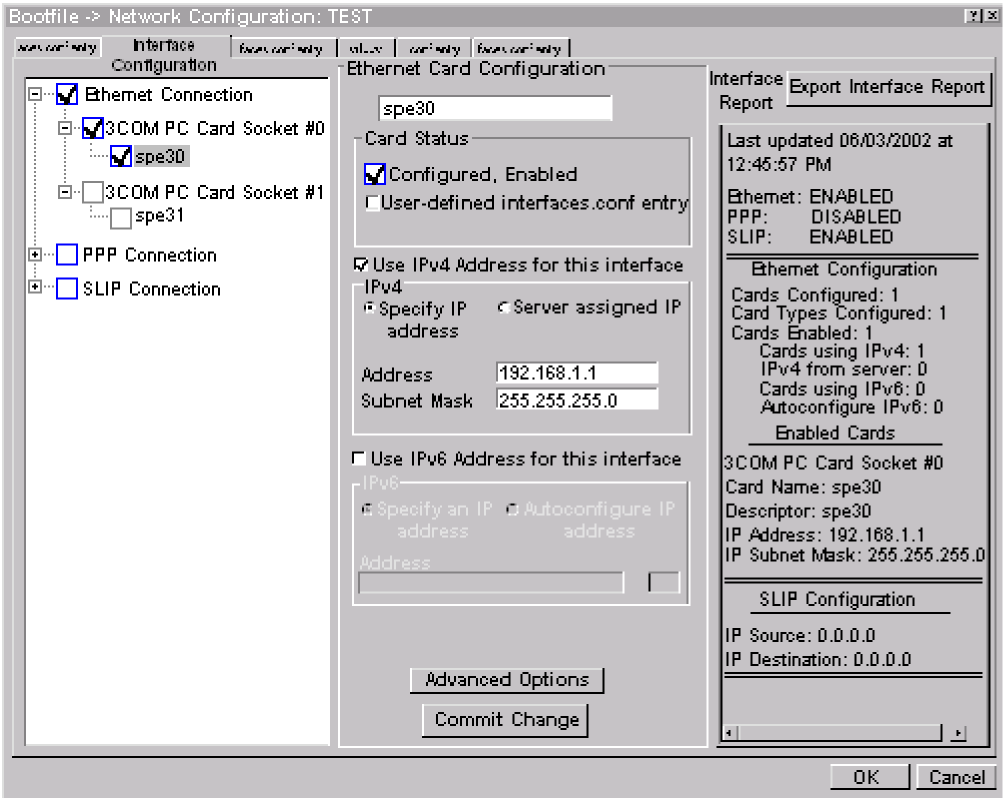
Step 2. Open the Configuration Wizard to the main window.

If you intend on using the target board across a network, proceed to step three. If not, go directly to step six.

Step 3. If you want to use the target board across a network, you will need to configure the Ethernet settings within the Configuration Wizard. To do this, select `Configure -> Bootfile -> Network Configuration` from the Wizard's main menu.

Step 4. From the **Network Configuration** dialog, select the `Interface Configuration` tab. From here you can select and enable the interface. For example, you can select the appropriate Ethernet card from the list of options on the left and specify whether you would like to enable IPv4 or IPv6 addressing. Figure 1-5 shows an example of the **Interface Configuration** tab.

Figure 1-5 Bootfile -> Network Configuration -> Interface Configuration


Bootfile -> Network Configuration: TEST

Interface Configuration

- [✓] Ethernet Connection
  - [✓] 3COM PC Card Socket #0
    - [✓] spe30
  - [ ] 3COM PC Card Socket #1
    - [ ] spe31
- [ ] PPP Connection
- [ ] SLIP Connection

Ethernet Card Configuration

spe30

Card Status

- [✓] Configured, Enabled
- [ ] User-defined interfaces.conf entry

[✓] Use IPv4 Address for this interface

IPv4

[✓ Specify IP address] Server assigned IP

Address 192.168.1.1

Subnet Mask 255.255.255.0

[ ] Use IPv6 Address for this interface

IPv6

Specify an IP address / Autoconfigure IP address

Address

Interface Report

```
Last updated 06/03/2002 at
12:45:57 PM

Ethernet: ENABLED
PPP:      DISABLED
SLIP:     ENABLED

Ethernet Configuration
Cards Configured: 1
Card Types Configured: 1
Cards Enabled: 1
   Cards using IPv4: 1
   IPv4 from server: 0
   Cards using IPv6: 0
   Autoconfigure IPv6: 0
     Enabled Cards
3COM PC Card Socket #0
Card Name: spe30
Descriptor: spe30
IP Address: 192.168.1.1
IP Subnet Mask: 255.255.255.0

SLIP Configuration
IP Source: 0.0.0.0
IP Destination: 0.0.0.0
```

For More Information

To learn more about IPv4 and IPv6 functionalities, refer to the ***Using LAN*** manual, included with this product CD.

For More Information

Contact your system administrator if you do not know the network values for your board.

Step 5. Once you have made your settings in the **Network Configuration** dialog, click `OK`.

Step 6. From the main menu, select `Configure -> Build Image` to display the **Master Builder** window. If you want networking included on this build, make sure the **SoftStax® (SPF) Support** box is checked.

Step 7. If you are not building a coreboot image with bootfile “uncompression” functionality, be sure to *deselect* the **Compressed Bootfile** check box.

Step 8. Select the **Bootfile Only Image** radio button, then click `Build`. This will build an image that can be placed on the PCMCIA card.

Step 9. Turn off the board and insert the PCMCIA IDE card into the PCMCIA slot of your computer.

WARNING

Inserting and removing a PCMCIA card with the power on is not supported in this release. Damage may occur to the PCMCIA card if it is inserted or removed while power is applied to the board.

Step 10. Click `Save As` to save the file `os9kboot` to the root directory of the PCMCIA IDE card.

Step 11. Remove the PCMCIA IDE card from the computer.

Step 12. Position the PCMCIA card so that the end with the connector holes is facing the PCMCIA socket and the label is facing up.

Step 13. Slide the card into the upper socket (socket 0) of the reference board until the card snaps onto the connector pins and the eject button pops out.

Step 14. Apply power to the board. The reference board will boot from the IDE PCMCIA card and you should see the “`$`” prompt.

Creating a Startup File

When the Configuration Wizard is set to use a hard drive or another fixed drive such as a PC Flash card as the default device, it automatically sets up the init module to call the `startup` file in the `SYS` directory in the target (For example: `/h0/SYS/startup`, `/mhc1/SYS/startup`). However, this directory and file will not exist until you create it. To create the startup file, complete the following steps:

Step 1. Create a `SYS` directory on the target machine where the `startup` file will reside (for example: `makdir /h0/SYS`, `makdir /dd/SYS`).

Step 2. On the host machine, navigate to the following directory:

```
MWOS/OS9000/SRC/SYS
```

In this directory, you will see several files. The files related to this section are listed below:

- `motd`: Message of the day file
- `password`: User/password file
- `termcap`: Terminal description file
- `startup`: Startup file

Step 3. Transfer all files to the newly created `SYS` directory on the target machine. (You can use Kermit, or FTP in ASCII mode to transfer these files.)

Step 4. Since the command lines in the startup file are system-dependent, it may be necessary to modify this file to fit your system configuration. It is recommended that you modify the file before transferring it to the target machine.

Step 5. Since the files are still in DOS format, you will be required to convert them into the OS-9 format with the `cudo` utility. The following command is an example:

```
cudo -cdo password
```

This will convert the `password` file from DOS to OS-9 format.

For More Information

For a complete description of all the `cudo` command options, refer to the ***Utilities Reference Manual*** located on the Microware OS-9 CD.

Example Startup File

Below is the example startup file as it appears in the `MWOS/OS9000/SRC/SYS` directory:

```
-tnxnp
tmode -w=1 nopause
*
*OS-9 - Version 3.0
*Copyright 2001 by Microware Systems Corporation
*The commands in this file are highly system dependent and
*should be modified by the user.
*
*setime </term              ;* start system clock
setime -s                   ;* start system clock
link mshell csl             ;* make "mshell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 term  ;* initialize devices
* load utils                ;* make some utilities stay in memory
* tsmon /term /t1 &         ;* start other terminals
list sys/motd
setenv TERM vt100
tmode -w=1 pause
mshell<>>>/term -l&
```

For More Information

Refer to the **Making a Startup File** section in Chapter 9 of the ***Using OS-9*** manual for more information on startup files.

Optional Procedures

Configuring Your ATA Card

The ATA card can be used to validate that the reference board is operational without a connection to the host machine:

To configure the ATA card, complete the following steps:

Step 1. From a DOS prompt on the host machine, navigate to the following directory:

```
MWOS\OS9000\ARMV4\PORTS\PID7T\BOOTS\SYSTEMS\PORTBOOT
```

Step 2. Run `os9make`.

Step 3. On the host machine, copy the following file into the root directory to the ATA card.

```
MWOS\OS9000\ARMV4\PORTS\PID7T\BOOTS\SYSTEMS\PORTBOOT\os9kboot
```

Step 4. Install the card in socket 1 (top) on the reference board.

Building a ROM Image for Networking

Although the PID7T boards come with two PCMCIA slots, only one slot is functional. If networking functionality is desired, it will be necessary to create and configure a ROM image that uses one of the following techniques:

- bootp to download a bootfile that includes high-level networking
- a network boot placed on an ATA PCMCIA card

  This type of boot is necessary because the 512K ROM part on the board does not have enough space to include a full function ROM image, including networking.

One of the following three board configurations will enable you to build a full featured ROM image that includes networking:

- Your PID7T board has a working Slot 1.

  This enables you to use a PCMCIA ATA card in Slot 0 and a PCMCIA ethernet card in slot 1.

  To determine if your board has this configuration, power the board off and place an ATA card in slot 0 and an Ethernet card in slot 1. Apply power to the board and watch your terminal as the board boots. If it finds both cards, you can use the Configuration Wizard to build a full bootfile for the ATA card, which includes networking for slot 1.

Note

The above method only works for high-level networking. For low-level networking, you must boot the board with only one slot workng, either from a network or from Flash memory.

- Slot 1 of your PID7T board is non-functional.

  In this configuration you can use the Configuration Wizard to build a ROM image that includes both ATA PCMCIA support and Ethernet support for slot 0. To accomplish this task, complete the following steps:

Step 1. Change the Init module's parameter list so that networking related commands (such as mbinstall, ipstart, ndbmod, dhcp, and inet) are not executed. Record the commands removed elsewhere so they can be issued by hand after the system boots. After configuring your networking options in the Configuration Wizard, select `Configuration -> Bootfile -> Disk Configuration -> Init Options`. Select the **/dd**, **User**, **Change Parameter List**, and **Edit Current Parameter List** radio buttons.

Step 2. Remove the series of networking related commands from the parameter list and click OK until all the dialog boxes are closed.

Step 3. Select the **Master Builder** window and rebuild the bootfile image.

Step 4. After making your `os9kboot` file, copy it to a PC-CARD.

Step 5. Place the ATA card into slot 0 (top) and apply power to the board. Be sure to boot from ide0.

Step 6. At the shell prompt type the following command:

```
pcmcia -d -s=0 -v
```

This powers down the ATA card.

Step 7. Remove the ATA card from slot 0.

Step 8. Insert your networking card into slot 0 and type the following command:

```
pcmcia -i -s=0 -v
```

The system should report finding your card.

Step 9. Type all the networking related commands that were removed from the parameter list. Networking should now be avaialable for use.

Following is an example boot of this configuration:

```
OS-9 Bootstrap for the ARM (Edition 65)

MICROWARE PCMCIA SOCKET SERVICES
i82365sl step B PCMCIA type controller
socket #00 occupied [0x04]
ATA IDE disk found in socket 00
IDE Base 0x0c0002e0 : Vector 0x00000047
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>
Boot from PCMCIA-1 IDE ---------------- <ide1>
Boot from PCMCIA-0 IDE ---------------- <ide0>
Enter system debugger ----------------- <break>
```

```
Restart the System -------------------- <q>

Select a boot method from the above menu: ide0

Wait for IDE drive ready......ready.
IDE Model                  : ATA_FLASH
Number Heads               : 0x0002
Total Cylinders            : 0x03d8
Sectors Per Track          : 0x0020
Checking Partitions        : 0
Fat Type                   : 0x16
File Name                  : OS9KBOOT
File Size                  : 0x00167648
Start Cluster              : 0x000057f4
Reading Bootfile....

Boot Address               : 0x0302c760
Boot Size                  : 0x00167648

OS-9 kernel was found.
A valid OS-9 bootfile was found.
[1]$ pcmcia -d -v -s=0
socket0:  occupied
It is now safe to remove the card is socket #00

[2]$ pcmcia -i -v -s=0
MICROWARE PCMCIA SOCKET SERVICES
i82365sl step B PCMCIA type controller
socket #0 occupied [0x06]
Ethernet card found in socket0
Base = 0x0c000600 Vector = 71

[3]$ mbinstall
[4]$ ipstart
[5]$ ping hobbes
PING hobbes.microware.com (172.16.1.6): 56 data bytes
64 bytes from 172.16.1.6: ttl=255 time=20 ms
[6]$
```

- Slot 1 of your PID7T board is non-functional.

  In this configuration you can use the Configuration Wizard to build a ROM image that includes the Ethernet boot option along with configuring both high and low-level networking. You can build a full-featured bootfile image and place it on the host system (or any other machine), which can be configured to send your `os9kboot` file to the target during start up.

Connecting the Target to an Ethernet Network

Microware OS-9 for ARM supports using a 3COM Etherlink III - LAN PC Card for SoftStax® TCP/IP connections. Also, Microware OS-9 for ARM provides system level support for telnet, FTP, and NFS.

To use Ethernet networking, you must create a bootfile that has the Ethernet options enabled and insert an Ethernet PCMCIA card into the reference board. To do this, complete the following steps:

Step 1. Click the `Start -> RadiSys -> Microware OS-9 for <product> -> Configuration Wizard` and click `OK`. The Configuration Wizard is displayed.

Step 2. Select `Configure -> Bootfile -> Network Configuration`. The **Network Configuration** dialog box appears.

Step 3. Change the network settings as needed. Set the configuration to specify so that the networking card will be in socket #1 and the PCMCIA IDE disk will be in socket #0.

Step 4. Create a new bootfile by following the directions in the **Building the OS-9 ROM Image** section.

Step 5. Turn off the power to the reference board.

WARNING
Inserting and removing a PCMCIA card with the power on is not supported in this release. Damage may occur to the PCMCIA card if it is inserted or removed while power is applied to the board.

Step 6. Position the PCMCIA IDE card so that the end with the PCMCIA female connector is facing PCMCIA socket 0 (the upper socket) and the label is facing up.

Step 7. Slide the PCMCIA IDE card into socket 0 (the upper socket) until the card snaps onto the pins and the eject button pops out.

Step 8. Position the Ethernet PCMCIA card so that the end with the PCMCIA female connector is facing PCMCIA socket 1 (the lower socket) and the label is facing up.

Step 9. Slide the PCMCIA Ethernet card into socket 1 (the lower socket) until the card snaps onto the pins and the eject button pops out.

Step 10. Apply power to the board.

Step 11. Test the Ethernet connection by pinging the reference board.

If the ping operation fails, the following scenarios should be evaluated:

- Is the board connected to a live Ethernet port?
- Is the Ethernet cable defective?
- Are the network settings for the reference board correct?

WARNING
Both of the PCMCIA sockets must work before the steps above will work.

Pinging the Target

Windows 95, 98, and NT include a ping command that can be used to test the Ethernet connection for the reference board. To do this, complete the following steps:

Step 1. Go to the DOS prompt.

Step 2. Type `ping <IP Address>`.

The IP Address is the address you assigned to the evaluation board in either the Coreboot module or the bootfile module. The address is typed without the <> brackets.

If the ping was successful, you will see the following response:

```
Reply from <IP Address>: bytes=xx time =xms TTL= xx
```

If the ping was unsuccessful, you will see the following response:

```
Request timed out.
```

Note

Windows 85, 88, and NT do not support IPv6.

Chapter 2: Board-Specific Reference

This chapter contains information that is specific to the PID7T reference board from ARM Ltd. It contains the following sections:

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

Note

This document describes using the PID7T with the ARM Ltd. ARM7TDMI processor.

For More Information

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

RadiSys.
MICROWARE SOFTWARE

Boot Options

Following are the default boot options for the reference board. Select these options by hitting the space bar during boot up when the following message appears on the console port:

```
Press the spacebar for a booter menu
```

Change the configuration of these booters by altering the `default.des` file located in the following directory:

```
MWOS/os9000/ARMV4/PORTS/PID7T/ROM/
```

Booters can be configured to be either menu or auto booters. The auto booters automatically try and 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 `romcnfg.h` 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 ROM image is programmed into Flash in the address range defined in ports `default.des` file, the system can boot and run from Flash.

`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 from PCMCIA ATA Card

The system can boot from either from a PC formatted PCMCIA hard card residing in slot 0 or slot 1.

Note

The system will hang during boot if there is not PCMCIA card, and it is configured to boot from one.

`ide1`	The file `os9kboot` is searched for in slot 1, if found, it is copied to system RAM and runs from there.
`ide0`	The file `os9kboot` is searched for in slot 0, if found, it is copied to system RAM and runs from there.

Booting from PCMCIA Ethernet Card

The system can boot using the BootP protocol using an Ethernet card and `eb` option.

`eb`	Ethernet boot, a PCMCIA card that supports ethernet will use the `bootp` protocol to transfer a bootfile into RAM. The system runs from there.

Restart Booter

The restart booter allows a method to restart the bootstrap sequence.

`q`	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 and enter the system level debugger.

Example Boot Session and Message

```
OS-9 Bootstrap for the ARM (Edition 65)

ATA IDE disk found in socket 00
Now trying to Override autobooters.

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

Boot embedded OS-9 in-place -------------- <N/A>
Copy embedded OS-9 to RAM and boot ------- <N/A>
Boot from PCMCIA-1 IDE ------------------- <ide1>
Boot from PCMCIA-0 IDE ------------------- <ide0>
Restart the System ----------------------- <q>
Enter system debugger -------------------- <break>

Select a boot method from the above menu: ide0

Wait for IDE drive ready.
IDE Model                 :       ATA_FLASH
Number Heads              : 0x0002
Total Cylinders           : 0x03d8
Sectors Per Track         : 0x0020

Checking Partitions       : 0
Fat Type                  : 0x16
File Name                 : OS9KBOOT
File Size                 : 0x000fdeb0
Start Cluster             : 0x00003a57
Reading Bootfile....
Boot Address              : 0xc002c850
Boot Size                 : 0x000fdeb0
OS-9 kernel was found.
A valid OS-9 bootfile was found.
$
```

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 vector mappings for the OS-9 PID7T implementation on the ARM7TDMI.

The ARM standard defines exceptions 0x0-0x8. The OS-9 system maps these 1-1. External interrupts from vector 0x6 are expanded to the virtual vector rage shown below by the armirq module.

Note

Vectors can be virtually remapped from a ROM at physical address 0, into DRAM at virtual address 0. This speeds up interrupt response time and is enabled by defining the first cache list entry as a sub 1 Meg size.

For More Information

See the ***Arm Development Card Manual*** for further information on individual sources.

Table 2-1 IRQ Assignments for the PID7T ARM7TDMI Board

OS-9 IRQ #	ARM Function
`0x0`	Processor Reset
`0x1`	Undefined Instruction
`0x2`	Software Interrupt

Table 2-1 IRQ Assignments for the PID7T ARM7TDMI Board (continued)

OS-9 IRQ #	ARM Function
`0x3`	Abort on Instruction Prefetch
`0x4`	Abort on Data Access
`0x5`	Unassigned/Reserved
`0x6`	External Interrupt (expanded to virtual vectors 0x40-0x4f)
`0x7`	Fast Interrupt
`0x8`	Alignment error

Table 2-2 IRQ Assignments for the PID7T ARM7TDMI Board

OS-9 IRQ #	PID7T APB FPGA (pic)
`0x40`	Reserved
`0x41`	Soft Interrupt
`0x42`	COMMSRX from processor
`0x43`	COMMSTX from processor
`0x44`	TIMER1 (internal)
`0x45`	TIMER2 (internal)
`0x46`	PC card slot A

Table 2-2 IRQ Assignments for the PID7T ARM7TDMI Board (continued)

OS-9 IRQ #	PID7T APB FPGA (pic)
`0x47`	PC card slot B
`0x48`	SERIAL A (16552)
`0x49`	SERIAL B (16552)
`0x4a`	PARALLEL
`0x4b`	ASB expansion 0
`0x4c`	ASB expansion 1
`0x4d`	APB expansion 0
`0x4e`	APB expansion 1
`0x4f`	APB expansion 2

Note

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 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 done first. The OS-9 IRQ code for the ARM7TDMI has assigned all interrupts as normal external interrupts and the user must re-define a source as an FIQ to make use of this feature.

Port Specific Utilities

The following port specific utilities are included:

- `pcmcia`

pcmcia

Syntax

```
pcmcia [<opts>]
```

options

`-s=`	<socket> socket number [default: all sockets]
`-d`	de-iniz socket(s)
`-i`	iniz socket(s)
`-v`	verbose mode
`-x`	dump CIS/Config information
`-?`	Print this help message

Description

`pcmcia` provides the ability to initilize or deinitilize a PCMCIA card after the system has booted. It also displays a PCMCIA cards CIS structure.

Example

```
$ pcmcia -x -s=0
ATA IDE disk found in socket0
Dump CIS Window for Socket #0
  Addr     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0 2 4 6 8 A C E
--------  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
28000000  01 03 d9 01 ff 1c 04 03 d9 01 ff 18 02 df 01 20  ...............
28000020  04 01 4e 00 01 15 2b 04 01 56 49 4b 49 4e 47 20  ..N...+..VIKING
28000040  43 4f 4d 50 4f 4e 45 4e 54 53 20 20 20 20 20 20  COMPONENTS
28000060  20 20 00 43 46 20 41 54 41 20 00 56 2e 31 30 32  .CF ATA .V.102
28000080  00 ff 21 02 04 01 22 02 01 01 22 03 02 04 5f 1a  ..!..."..."..._.
280000a0  05 01 03 00 02 0f 1b 09 c0 40 a1 21 55 55 08 00  .........@.!UU..
280000c0  22 1b 06 00 01 21 b5 1e 35 1b 0b c1 41 99 21 55  "....!..5...A.!U
280000e0  55 64 f0 ff ff 22 1b 06 01 01 21 b5 1e 35 1b 0d  Ud..."....!..5..
28000100  82 41 98 ea 61 f0 01 07 f6 03 01 ee 22 1b 06 02  .A..a......."...
28000120  01 21 b5 1e 35 1b 0d 83 41 98 ea 61 70 01 07 76  .!..5...A..ap..v
28000140  03 01 ee 22 1b 06 03 01 21 b5 1e 35 14 00 ff ff  ..."....!..5....
28000160  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
28000180  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
280001a0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
280001c0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
280001e0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
Dump Config Window for Socket #0
  Addr     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0 2 4 6 8 A C E
--------  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
28000200  43 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00  C...............
28000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
28000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280002a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280002c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280002e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

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**

Low-Level System Modules

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.

The following low-level system modules are tailored specifically for the ARM Ltd. ARM7TDMI PID7T platform. The functionality of these modules can be altered through changes to the configuration data module (cnfgdata). **Table A-1** provides a list and brief description of the modules. These modules can be found in the following directory:

```
MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS/ROM
```

Table A-1 PID7T-Specific Low-Level System Module

Module Name	Description
`cnfgdata`	Contains the low-level configuration data.
`cnfgfunc`	Provides access services to cnfgdata's data.
`commcnfg`	Inits communication port defined in cnfgdata.
`conscnfg`	Inits console port defined in cnfgdata.
`ide`	IDE boot support module. PCMCIA compatible.
`io16550`	Provides polled serial driver support for the low-level system.
`llcis`	Inits the PCMCIA interface including cards.

Table A-1 PID7T-Specific Low-Level System Module (continued)

Module Name	Description
`lle509`	Provides low-level ethernet services via 3COM PCMCIA card.
`portmenu`	Inits booters defined in the cnfgdata.
`romcore`	Board specific initialization code.
`Armtimr`	Provides low-level timer services via time base register.
`usedebug`	Inits low-level debug interface to RomBug, SNDP, or none.

The following low-level system modules provide generic services for OS-9 Modular ROM. Table A-2 provides a list and brief description of the modules. These modules can be found in the following directory:

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

Table A-2 Generic Services Low-Level System Modules

Module Name	Description
`bootsys`	Booter registration service module.
`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.

Table A-2 Generic Services Low-Level System Modules (continued)

Module Name	Description
`hlproto`	Provides user level code access to protoman.
`llbootp`	Booter which 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`	Booter which uses kermit protocol.
`notify`	Provides state change information for use with LL and HL drivers.
`override`	Booter which allows choice between menu and auto booters.
`parser`	Provides argument parsing services.
`pcman`	Booter which reads MS-DOS file system.
`protoman`	Protocol management module.
`restart`	Booter which cause a soft reboot of system.
`romboot`	Booter which allows booting from ROM.
`rombreak`	Booter which calls the installed debugger.
`rombug`	Low-level system debugger.

Table A-2 Generic Services Low-Level System Modules (continued)

Module Name	Description
`sndp`	Provides low-level system debug protocol.
`srecord`	Booter which accepts S-Records.
`swtimer`	Provides timer services via software loops.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the ARM Ltd. ARM7TDMI PID7T board peripherals. Unless otherwise specified, each module can be found in a file of the same name in the following directory:

`MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS`

CPU Support Modules

These files are all found in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`kernel`	The kernel provides all basic services for the OS-9 system.
`fpu`	Provides software emulation for floating point instructions.
`vectors`	Provides interrupt service entry and exit code. The vectors module is found in the file vectarm.

System Configuration Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS/INITS`

`init`	Descriptor module with high level system initialization information.
`nodisk`	Same as init, but used in a disk-less system.

Interrupt Controller Support

This module provides an extension to the vectors module by mapping the single interrupt generated by an interrupt controller into a range of pseudo vectors which are recognized by OS-9 as extensions to the base CPU exception vectors.

For More Information

The mappings are described in **Chapter 2**.

Vector Module

`armirq`	P2module which provides interrupt acknowledge and dispatching support for the ARM7TDMI pic.

Ticker

`tkarm`	Driver which provides the system ticker based on the ARM7TDMI system timer.

Abort Handler

`abort`	P2module which provides a way to enter the system-state debugger via the interrupt triggered by PID7T switch SW1.

Generic IO Support Modules (File Managers)

These modules are found in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`ioman`	Provides generic io support for all IO device types.
`scf`	Provides generic character device management functions.
`rbf`	Provides generic block device management functions for OS-9 format.
`pcf`	Provides generic block device management functions for MS-DOS FAT format.
`spf`	Provides generic protocol device management function support.
`mfm`	Provides generic graphics device support for MAUI®.
`pipeman`	Provides a memory FIFO buffer for communication.

Pipe Descriptor

The pipe descriptor is found in the following directory:

`MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS/DESC`

`pipe`	Pipeman descriptor which provides a RAM based FIFO for process communication.

RAM Disk Support

`ram`	RBF driver which provides a RAM based virtual block device.

RAM Disk Descriptors

The RAM disk descriptors are found in the following directory:

`MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS/DESC/RAM`

`r0`	RBF descriptor which provides access to a ram disk.
`r0.dd`	Same as r0 except with module name dd (for use as the default device).

Serial and Console Devices

`sc16550`	SCF driver which provides serial support the PID7T's 16552.

sc16550 Descriptors

`term_t1/t1`	Descriptor modules for use with SC16552 UART A.
	PID7T Board header: PL4
	Default Baud Rate: 19200
	Default Parity: None
	Default Data Bits: 8
	Default Handshake: Software
`term_t3/t3`	Descriptor modules for use with SC16552 UART B.
	PID7T Board header: PL3
	Default Baud Rate: 115200
	Default Parity: None
	Default Data Bits: 8
	Default Handshake: Software

`scllio`	SCF driver which provides serial support via the polled low-level serial driver.

scllio Descriptors

`vcons/term`	Descriptor modules for use with scllio in conjunction with a low-level serial driver. Port configuration and set up follows what 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*** for more information.

PCMCIA Support for IDE type Devices

`rb1003`	RBF/PCF driver which provides driver support for IDE/EIDE devices. This driver is used to provide disk support for PCMCIA ATA FLASH.

rb1003 Descriptors

`hc1/hc1fmt`, `hc1.dd`	RBF Descriptor modules for use with PCMCIA slot #0 (bottom). PID7T Board header: SK4 hc1fmt: format enabled hc1.dd: module name of dd

`mhc1, mhc1.dd`	PCF Descriptor modules for use with PCMCIA slot #0 (bottom). PID7T Board header: SK4 mhc1.dd: module name of dd
`he1, he1fmt, he1.dd`	RBF Descriptor modules for use with PCMCIA slot #1 (top). PID7T Board header: SK4 he1fmt: format enabled he1.dd: module name of dd
`mhe1, mhe1.dd`	PCF Descriptor modules for use with PCMCIA slot #1 (top). PID7T Board header: SK4 mhc1.dd: module name of dd

PCMCIA Support for 3COM Ethernet card

These files are found in the following directory:

`MWOS/OS9000/ARMV4/PORTS/PID7T/CMDS/BOOTOBJS/SPF`

`spe509_pcm`	SPF driver to support ethernet for a 3COM EtherLink III PCMCIA card.

spe509_pcm Descriptors

`spe30`	SPF descriptor module for use with PCMCIA slot #0 (bottom, SK4).
`spe31`	SPF descriptor module for use with PCMCIA slot #1 (top, SK4).

Network Configuration Modules

`inetdb/inetdb2/rpcdb`