Home

OS-9® for Renesas MS7780SE Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

This manual reflects version 4.7 of Microware OS-9.
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.

Contents

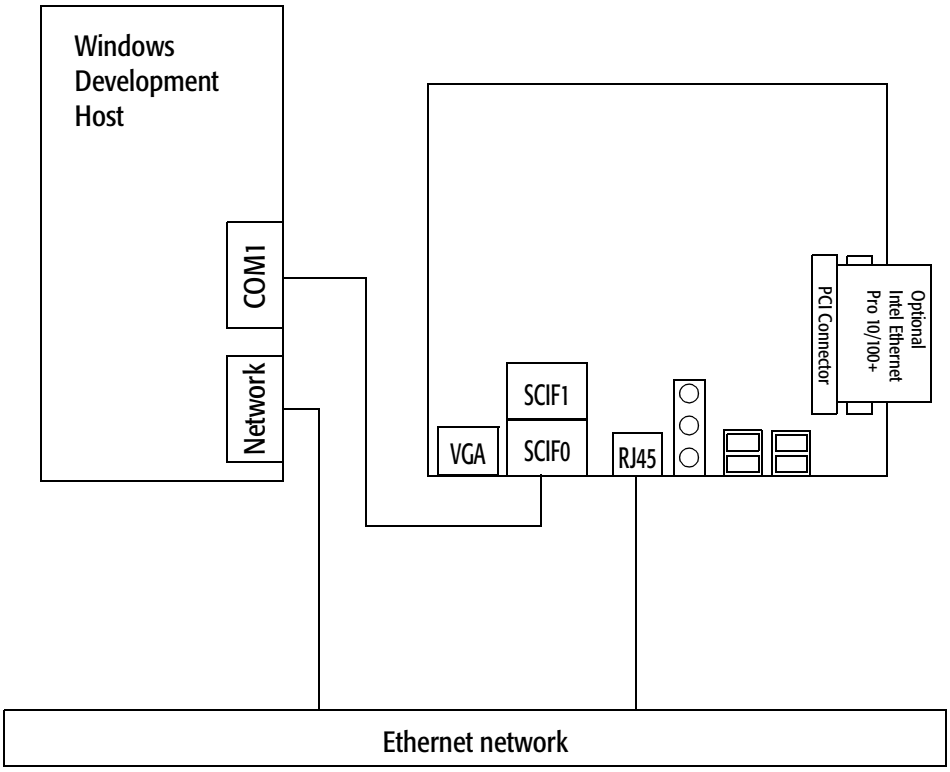
1 Installing and Configuring OS-9®

This chapter describes installing and configuring OS-9® on the Renesas MS7780SE board. It includes the following sections:

Development Environment Overview

Figure 1-1 shows a typical development environment for a Renesas MS7780SE board. The components shown those generally required to develop for OS-9 on the board.

Figure 1-1. Renesas MS7780SE Development Environment



Requirements and Compatibility

Before you begin, install the *Microware OS-9 for SH-4* CD-ROM on your host PC.

Host Hardware Requirements (PC Compatible)

The host PC must have the following minimum hardware characteristics:

- 500MB of free hard disk space
- the recommended amount of RAM for the host operating system
- a CD-ROM drive
- a free serial port
- an Ethernet network card
- access to an Ethernet network

Host Software Requirements (PC Compatible)

- Microware OS-9 for SH-4
- Windows 2000, Windows ME, or Windows XP.
- terminal emulation program

The examples in this document use Hyperterminal, a terminal emulation program, which is included with all Windows operating systems.

- [Optional] BOOTP and TFTP servers. BOOTP and TFTP servers are required if Ethernet booting OS-9.

Target Hardware Requirements

Your MS7780SE03 board requires the following hardware:

- an ATX power supply
- an RS-232 null modem serial cable (for OS-9 console)
- Ethernet cable

In addition to the above requirements, you may want to use a PCI Ethernet card. (Supported Ethernet card: Intel PRO 10/100.)

Target and Host Setup

Before installing and configuring OS-9 on your MS7780 evaluation board, refer to the hardware documentation for information on hardware setup.

Settings

The factory default setting for the DIP switches may not work with OS-9. The Renesas boot monitor will be used. Be sure the DIP switches agree with the following settings:

Switch Configuration	
SW1-1,2,3 set to OFF, OFF, OFF	IOS0, IOS1, and IOS2 grounded (SMSC 91C111 LAN controller)
SW1-4 set to OFF	EEPROM access enabled (SMSC 91C111 LAN controller)
SW6-1,2 set to ON, ON	115,200 Console Baud Rate
SW6-3 set to ON	EPROM at address 0x00000000
SW6-4,5 set to OFF	Reserved
SW6-6 set to ON	16-bit Support for Areas 5 and 6
SW7-1 set to ON	SH7780 MODE0
SW7-2 set to ON	SH7780 MODE1
SW7-3 set to OFF	SH7780 MODE2
SW7-4 set to OFF	SH7780 MODE7
SW7-5 set to OFF	SH7780 MODE3
SW7-6 set to OFF	SH7780 MODE4
SW7-7 set to ON	SH7780 MODE5

Connecting the Target to the Host

Connecting the MS7780SE board to your host PC involves attaching the power, serial, and Ethernet cables to the reference board. Once you have the board connected, you can use HyperTerminal to verify the serial connection.

Attaching the Cables

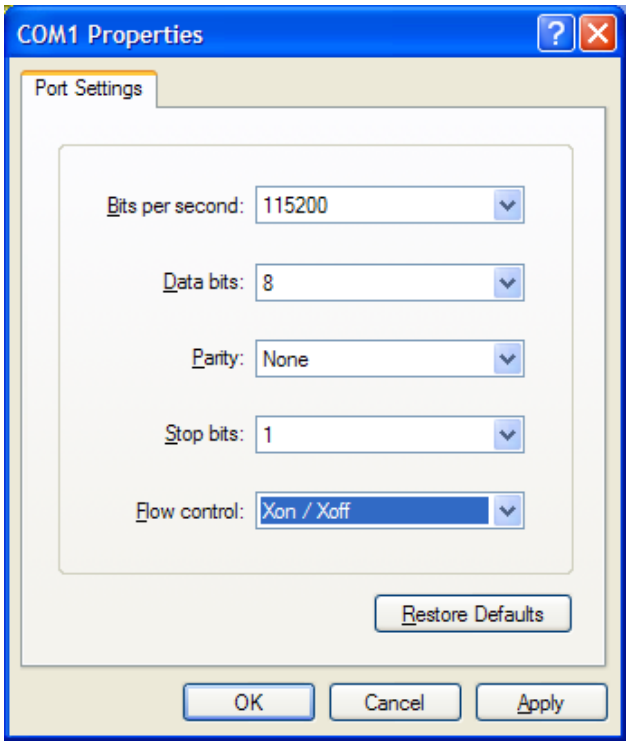
To attach the cables, complete the following steps:

1. Attach the ATX power supply to the connector labeled ATX POWER.
2. Plug the serial cable into the lower DB9 connector (SCIF0).
3. If you are using Ethernet, plug the cable into the on-board RJ-45 (CN13).

Booting to the Boot Menu

It will be necessary to boot to the Renesas prompt in order to verify that your serial cable is connected properly. To do this, complete the following steps:

1. From the desktop, click `Start` and select `All Programs -> Accessories -> Communications -> HyperTerminal` to start HyperTerminal.
2. Specify and name for your connection and choose an icon to represent it. Click OK to advance to the next dialog.
3. Change the Connect Using drop-down menu to the communications port that you have plugged your null modem cable into (e.g. COM1). Click OK to advance to the next dialog.
4. Change to setting to look like those below: 115,200 baud, 8 data bits, no parity, 1 stop bit, and XON/XOFF flow control.



   Click OK to enable the connection.

5. Apply power to the board and push the small button labeled POWER (SW10). The Renesas boot monitor boots the board. You should see the following text in your HyperTerminal window.

```
============================================================

  SH7780 Solution Engine(MS7780SE02) Self Debugger Ver0.1B

------------------------------------------------------------
```

```
(c)Copyright 2005. Hitachi-ULSI Systems Co.,Ltd.
                                          All right reserved
============================================================


                    H[elp] for help messages...
Ready>
```

Building the OS-9 ROM Image

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 image (bootfile) as specified by its configuration. For example from a flash device, 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 operating 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.

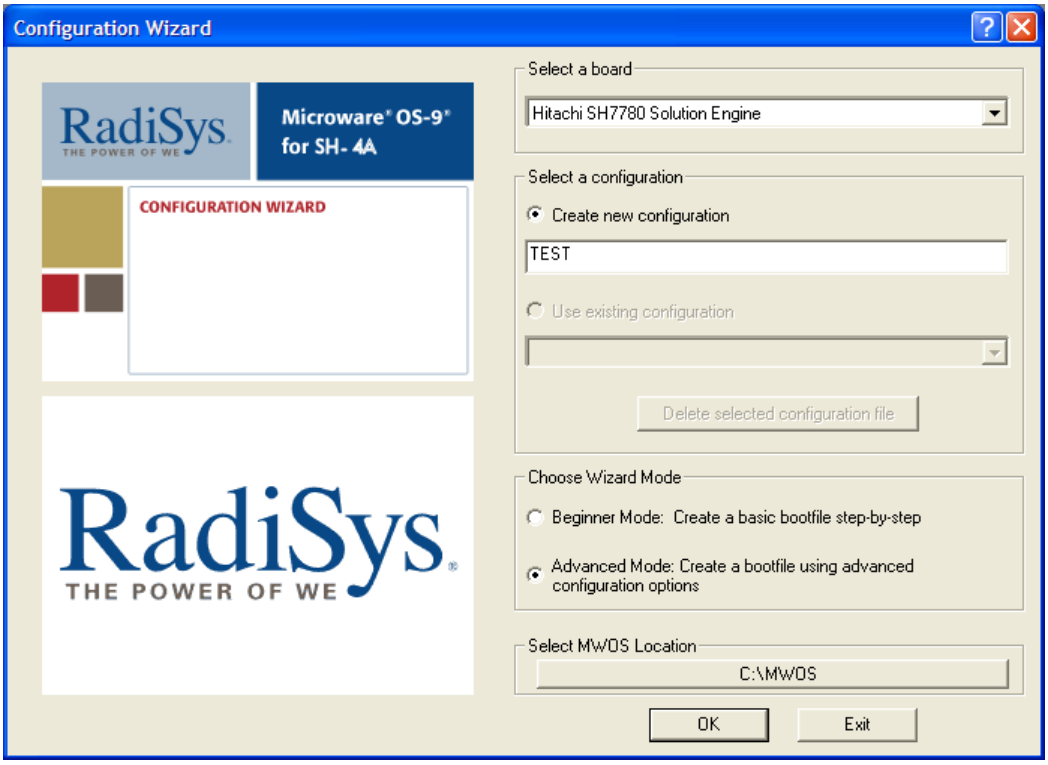
RadiSys provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the OS-9 installation process.

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:

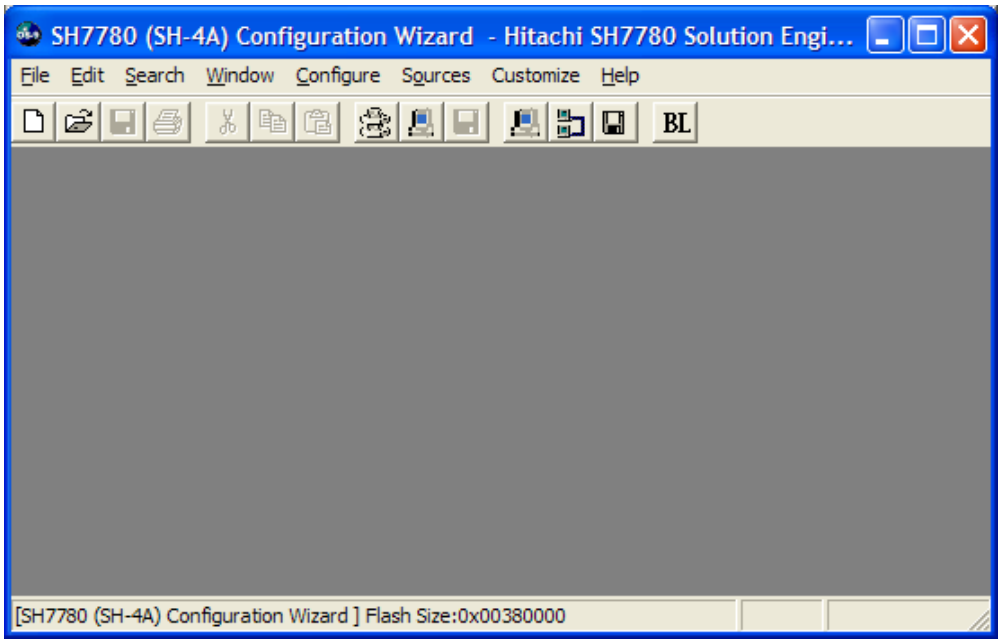
1. From the Windows desktop, select `Start` -> `All Programs` -> `RadiSys` -> `Microware OS-9 for SH-4` -> `Microware Configuration Wizard`. You should see the following opening screen:

Figure 1-2. Configuration Wizard Opening Screen



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



Configuring Coreboot and Bootfile Options

The only steps necessary in configuring the coreboot and bootfile options involve filling in the Ethernet information. All other default settings in the Configuration Wizard are correct for the MS7780SE board.

If you choose to no use the on-board Ethernet, you may skip the sections on Ethernet setup and proceed directly to the Building the Coreboot + Bootfile Image section.

Ethernet Configuration (Coreboot)

To configure the Ethernet settings from the Configuration Wizard, complete the following steps:

1. From the Main Configuration window, select `Configure -> Coreboot -> Main configuration`.
2. Select the `Ethernet` tab.
3. Enter the Ethernet address information in the address text boxes. This includes the following information:
   - IP Address
   - Subnet Mask
   - IP Gateway

   If you are uncertain of the values for these text boxes, contact your system administrator.
4. Select the appropriate Ethernet card in the drop down menu box at the bottom of the screen. (This should be either the on-board 91C111 or PCI Intel PRO 100.)

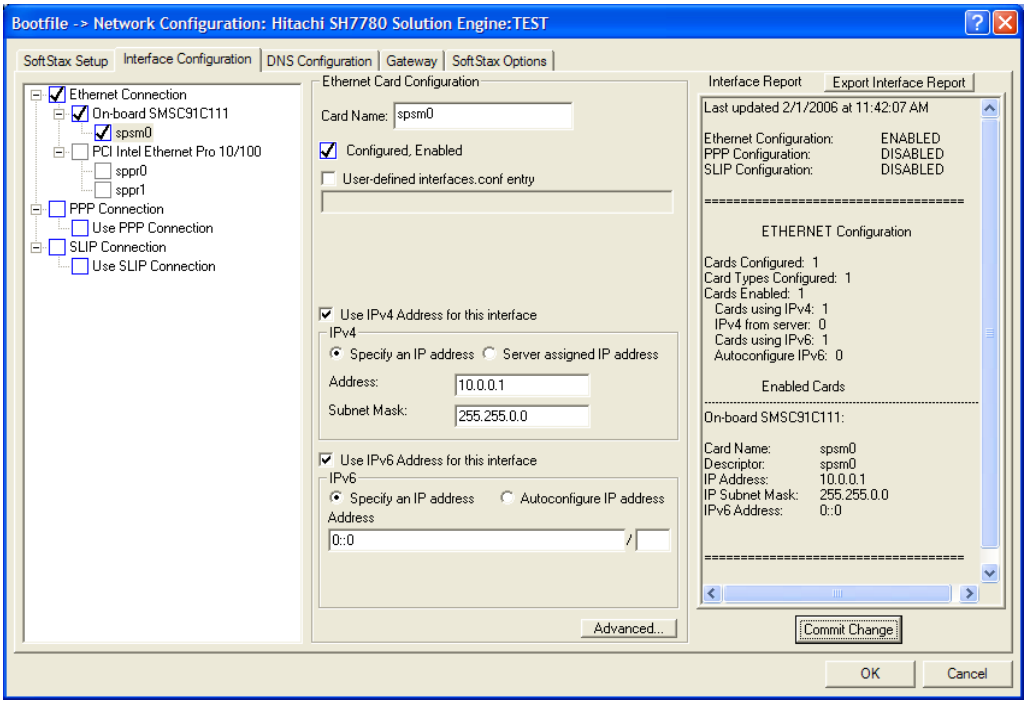
5. Check the Add to Boot Menu check box under the Ethernet Boot Options.
6. Click OK to close the window.

Ethernet Configuration (Bootfile)

To configure the Ethernet settings from the Configuration Wizard, complete the following steps:

1. 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.
2. 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 1-4 shows an example of the Interface Configuration tab.

Figure 1-4. Bootfile -> Network Configuration -> Interface Configuration



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

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

3. Select the DNS Configuration and Gateway tabs to specify setting specific to your site.
4. Select the SoftStax Options tab and select choose any networking options.
5. Click OK to close the dialog box.

Building the Coreboot + Bootfile Image

Once your coreboot and bootfile settings have been properly configured, complete the following steps to build the ROM image:

1. Select Configure -> Build Image. The Master Builder window is displayed.
2. From the Master Builder window, select the check boxes appropriate for your setup:
   - If you are using a RAM disk, ensure the Disk Support and Disk Utilities check boxes are checked.
   - If you have Ethernet setup for your system, select the SoftStax (SPF) Support and User State Debugging Modules check boxes.
3. Select the Coreboot + Bootfile radio button.
4. Click Build. This builds the ROM image that can later be burned into flash memory. The name of this ROM image is rom.s. The file containing the image is in Motorola S-record format.
5. Click Finish.
6. Select File -> Save Settings to save the configuration.
7. Select File -> Exit to quit the Configuration Wizard.

Transferring the ROM Image to the Target

In the previous section, you built a ROM image. To load this ROM image onto the target board, complete the following steps, using a capable terminal emulation program such as Hyperterminal:

1. The serial cable should be connected to SCIF0 (the lower DB9 connector) with a setting of 115,200 baud, 8-bit words, no parity, 1 stop bit, and XON/XOFF flow control. Once you apply power, the Hitachi H-UDI boot monitor prompt should appear.
2. At the Ready> prompt, type fl a1000000. The screen should display the following information:

   ```
   Flash Memory data copy to RAM
   Please Send A S-format Record
   ```

3. From the Hyperterminal window, transfer the rom.s file from the following directory to the target:

   ```
   MWOS/OS9000/SH4A/PORTS/SH7780SE/BOOTS/INSTALL/
   PORTBOOT
   ```

This transfer will take some time to complete. When it is finished the following messages are displayed:

```
Start Addrs = 01000000
End   Addrs = 013FFFFF

Transfer complete
Flash chip erase: complete
Program :complete
Flash write complete
Ready>
```

4. At the `Ready>` prompt type `g a1000000` to start OS-9. OS-9 will boot and display a shell prompt. If networking was configured, your board will be able to communicate on the network.

Optional Procedures

The following sections detail procedures you may perform once you have installed and configured OS-9.

Programming a ROM Image with the pflash Utility

`pflash` is an OS-9 utility that transfers an image into flash. The following steps detail how to create a new `rom` (`coreboot` + `bootfile`) image and burn the image into flash using `pflash`.

1. If you are no longer running the Configuration Wizard (from the previous steps) on your Windows desktop, select **Start** -> **All Programs** -> **RadiSys** -> **Microware OS-9 for SH-4** -> **Microware Configuration Wizard**. The Configuration Wizard opening screen displays. Click on the **Use existing configuration** radio button in the **Select a configuration** group. Ensure that your previous configuration appears in the drop-down menu and that the **Advanced Mode** radio button is selected in the Choose Wizard Mode group. Click OK.
2. Select Configure -> Build Image.. to display the **Master Builder** screen.
3. Ensure **Coreboot + Bootfile** is selected and click Build.
4. Once the build is complete, click Save As to save the ROM image to a directory of your choosing.
5. The default location for the `rom` file is in the following directory:

   `\MWOS\OS9000\SH4A\PORTS\SH7780SE\BOOTS\INSTALL\PORTBOOT`
6. Start a DOS shell on the host system.
7. Navigate to the directory in which the OS-9 ROM image, `rom`, is located.
8. On the target, initialize the large RAM disk (`/r1`) so it can hold the ROM image by entering the following command in the Hyperterminal window:

   ```
   $ iniz /r1
   ```

9. On your Windows host, use FTP to transfer the new image to the target system. At the prompt enter the following command:

   ```
   ftp <IP address or host name of target>
   ```

10. Log in with the username `super` and the password `user`.
11. At the `ftp>` prompt, enter the following command:

    ```
    ftp> cd /r1
    ```

    This specifies that the `/r1` device 's root is the current directory.
12. At the `ftp>` prompt, enter the following command:

    ```
    ftp> bin
    ```

    This designates binary format.
13. At the `ftp>` prompt, enter the following command:

    ```
    ftp> put rom
    ```

    The OS-9 ROM image file is transfered to the target's RAM disk (`/r1`).
14. On the target, enter the following command:

    ```
    $ pflash /r1/rom
    ```

    The file is programmed into the target system's flash memory for the next reboot of the system.

Flashed `rom` Image Issues

There are a number of issues to keep in mind when programming the flash with binary images:

If OS-9 fails to boot after using `pflash` to write the `coreboot` or `rom` image, you must revert back to using the Hitachi boot monitor to program the flash.

If a `rom` image (`coreboot` + `bootfile`) with an uncompressed bootfile is programmed into the flash, but an Ethernet or kermit boot is used, any modules not found in the downloaded boot will be found and used from the flash. The "`nokrs=1`" option can be used with the Ethernet or kermit booters (e.g. `eb nokrs=1` at the ROM boot menu) to prevent modules in flash from being found.

To erase the `bootfile` portion of a `rom` image programmed into flash, erase the entire part and then reprogram with only `coreboot`.

Building with Makefiles

Building boots with makefiles allows you greater control over which modules are included in the boot. For the MS7780SE reference board, the directory in which boots can be made is listed below:

```
MWOS/OS9000/SH4A/PORTS/SH7780/BOOTS/SYSTEMS/PORTBOOT
```

Makefile Network Option

Networking is not included in a bootfile by default. To include the networking modules in the bootfile, set the NETWORK macro definition to TRUE in the file named makefile. The style, IPv4/IPv6 or IPv4-only, of networking stack can also be chosen by using one of the two lines that appear below the NETWORK macro. In addition, be certain that the IP address for the board is set correctly; this helps to avoid network problems.

Using Makefiles

When using a makefile to build boots, three bootlist files are used to include the modules for booting. These bootlist files can be edited to include or exclude modules required for the system.

These bootfile lists are located in /MWOS/OS9000/SH4A/PORTS/SH7780SE/BOOTS/SYSTEMS/PORTBOOT, and are defined as follows:

coreboot.ml	used to make the low-level boot (called coreboot) When using this file, the romcore file must be input first, followed by the initext file. These two files are not OS-9 modules. romcore is the raw code needed to bring the hardware to a known stable state, while initext is a way for users to extend the low level sysinit code without changing sysinit.c or remaking romcore. The rest of the files included with coreboot.ml are actual OS-9 modules. Low-level booters and debuggers can be added or removed. In addition, the low-level Ethernet, IP stack, and SCSI system can be uncommented in order to provide bootp booting and/or SCSI booting. Low-level Ethernet or low-level SLIP can also provide system state debugging through Hawk.
bootfile.ml	used to create the high-level boot (called bootfile) This file contains all of the modules needed to produce an OS-9 system. This includes the kernel, system protection, cache control, file managers, and drivers and descriptors. Also included are various utilities and application programs. Not included with this file are networking modules. Additional modules can be included or excluded where appropriate.
lan.ml	contains the SoftStax modules and network utilities for the IPv4/IPv6 networking stack. These modules are simply merged into the end of the bootfile created from the bootfile.ml bootlist.

`spf.ml`	contains the SoftStax modules and network utilities for the IPv4-only networking stack. These modules are simply merged into the end of the bootfile created from the `bootfile.ml` bootlist.

Making Network Configuration Changes

To configure the network parameters for SoftStax and Ethernet, one file needs to be altered and one makefile needs to be run. To do this, complete the following steps:

1. Navigate to the `MWOS\OS9000\SH4A\PORTS\SH7780\SPF\ETC` directory and open the `interfaces.conf` file.
2. From the `interfaces.conf` file, fill in the correct IP address, broadcast address, and netmask values. You can also supply the host name in this area as well.
3. Save the file.

Once you have saved the file, run the makefile in the directory listed in step one. This makes the appropriate `inetdb` and `inetdb2` modules.

Because the Configuration Wizard configures the network in its own manner, if you are using it to configure network parameters, the above changes are not needed. However, if you choose to make the above changes, the Wizard will remain unaffected.

Low-Level Network Configuration Changes

To configure the low-level Ethernet parameters, one file needs to be altered and one makefile needs to be run. To do this, complete the following steps:

1. Navigate to the `MWOS\OS9000\SH4A\PORTS\SH7780SE\ROM\CNFGDATA` directory and open the `config.des` file.
2. From the `config.des` file, you will need to correctly define the macros for the IP address, broadcast, subnet, and mac.
3. Run the makefile in the directory listed in step one and a new `cnfgdata` module will be created. A coreboot can now be created with this configuration.

Because the Configuration Wizard configures the network in its own manner, if you are using it to configure Ethernet parameters, the above changes are not needed. However, if you choose to make the above changes, the Wizard will remain unaffected.

2 Board Specific Reference

This chapter contains porting information specific to the Toshiba RBHM4x00 board. It includes the following sections:

Boot Options

Default boot options for the MS7780SE are listed below. The boot options can be selected by pressing the space bar during system boot when the following message appears on the serial console:

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

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

```
MWOS\OS9000\SH4A\PORTS\SH7780SE\ROM\CNFGDATA
```

Booters can be configured to be either of these:

- Auto booters, which automatically attempt to boot in the same order as listed in the auto booter array.
- Menu booters, from the defined menu booter array, which are chosen interactively from the console command line after the boot menu displays.

Booting from Flash

When the `rom_cnfg.h` file has a defined ROM search list, the options `bo` and `lr` appear in the boot menu. If no ROM search list is defined, `N/A` appears in the boot menu. If an OS-9 bootfile is programmed into flash memory 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\SH4A\PORTS\SH7780SE\ROM\ROMCORE
```

`bo`	ROM boot: the system runs "in-place", from the flash. This makes it impossible to set a breakpoint with the debugger since the code in flash cannot be modified.
`lr`	load to RAM boot: the system copies the ROM bootfile image into RAM and runs from there. Thus, breakpoints and tracing will work as expected.

Booting over a Serial Port via kermit

The system can download a bootfile in binary form over its serial port at speeds up to 115,200 baud using the kermit protocol. Dots on the console indicate download progress.

`ker`	kermit boot: The boot file is sent via kermit protocol into system RAM and it runs from there.

Restart Booter

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

`q`	quit: Quit and reboot the board 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 resets.

`break` break: Break and enter the system level debugger Rombug.

Sample Boot Session and Messages

Below is a MS7780SE example boot using the `bo` boot option.

```
OS-9 Bootstrap for the SuperH (Edition 68)

PCI device initialization - Completed

ll91c111: auto-negotiating....

Now trying to Override autobooters.

Press the spacebar for a booter menu
<spacebar>


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

Boot over Ethernet (SMSC91C111) ---- <eb>
Boot embedded OS-9 in-place -------- <bo>
Copy embedded OS-9 to RAM and boot - <lr>
Kermit download -------------------- <ker>
PCI View Utility ------------------- <pciv>
Enter system debugger -------------- <break>
Restart the System ----------------- <q>

Select a boot method from the above menu: bo

Now searching memory ($81080000 - $813fffff) for an OS-9 Kernel...

An OS-9 kernel was found at $81080000
A valid OS-9 bootfile was found.
+3
+6
$ mfree
Current total free RAM: 120528.00 K-bytes
$
```

The Fastboot Enhancement

The Fastboot enhancements to OS-9 were added to address the needs of embedded systems that require faster system bootstrap performance. The Fastboot concept exists to inform OS-9 that the defined configuration is static and valid. This eliminate the dynamic search OS-9 usually performs during the bootstrap process. It also allows the system to perform for 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 of a particular assumption, and that the associated bootstrap functionality should be omitted.

One important feature of the Fastboot enhancement is the ability of the flags to become dynamically altered during the bootstrap process. For example, the bootstrap code might be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources that indicate different bootstrap requirements.

Another important feature of the Fastboot enhancement is its versatility. The enhancement's versatility allows for special considerations under a variety of circumstances. This can be useful in a system in which most resources are known, static, and functional, but whose additional validation is required during bootstrap for a particular instance (such as a resource failure).

Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. One 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within a set of data structures shared by the kernel and the ModRom sub-components. Hence, the field is available for modification and inspection by the entire set of system modules (both 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 in the following sections.

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. Limiting validation in this manner will omit the CRC check on modules, which may save you a considerable amount of time. For example, if a system has many modules in ROM, in which access time is typically longer than it is in RAM, omitting the CRC check will drastically decrease the bootstrap time. Furthermore, since it is rare that data corruption will occur in ROM, omitting the CRC check is a safe option.

In addition, the `B_OKRAM` bit instructs the low-level and high-level systems to accept their respective RAM definitions without verification. Normally, the system probes

memory during bootstrap based on the defined RAM parameters. This method allows system designers to specify a possible range of RAM the system will validate upon startup; thus, the system can accommodate varying amounts of RAM. However, in an embedded system (where the RAM limits are usually statically defined and presumed to be functional) 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 similarly to the `B_OKRAM` option with the exception 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 takes the `init` module with the highest revision number. Using the `B_1STINIT` in a statically defined system omits the extended `init` module search, which can save a considerable amount of time.

B_NOIRQMASK

The `B_NOIRQMASK` bit instructs the entire bootstrap system to 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, in systems with a well-defined interrupt system (systems that are calmed by the `sysinit` hardware initialization code) and a requirement to respond to an installed interrupt handler during startup, this option can be used. Its implementation will prevent the ModRom and kernel cold-start from disabling interrupts. (This is useful in power-sensitive systems that need to respond to "power-failure" oriented interrupts.)

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 or 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 you can control the bootstrap speed of your system.

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 as a redefinition of the macro in the `rom_config.h` header file or 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 your 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, which accepts the RAM and ROM definitions without verification. It 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 which user-defined bootstrap procedure should be used. An example P2 module is shown below.

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 MS7780SE board.

Table 2-1. OS-9 IRQ Assignment

OS-9 IRQ #	SH-4A Vector Description
0x00	Power-on reset
0x01	Manual reset
0x02	TLB miss/invalid, load
0x03	TLB miss/invalid, store
0x04	Initial page write
0x05	TLB protection violation, load
0x06	TLB protection violation, store
0x07	Address error, load
0x08	Address error, store
0x09	FPU classed exception
0x0b	TRAPA instruction
0x0c	Reserved instruction
0x0d	Illegal slot instruction
0x0e	Nonmaskable IRQ
0x0f	User break point
0x10	External IRQ 0
0x11	External IRQ 1
0x12	External IRQ 2
0x13	External IRQ 3
0x14	External IRQ 4
0x15	External IRQ 5
0x16	External IRQ 6
0x17	External IRQ 7
0x18	External IRQ 8
0x19	External IRQ 9
0x1a	External IRQ 10
0x1b	External IRQ 11
0x1c	External IRQ 12
0x1d	External IRQ 13
0x1e	External IRQ 14
0x24	RTC ATI IRQ
0x25	RTC PRI IRQ
0x26	RTC CUI IRQ
0x2b	WDT ITI IRQ
0x2c	TMU0 IRQ
0x2d	TMU1 IRQ
0x2e	TMU2 IRQ

Table 2-1. OS-9 IRQ Assignment

OS-9 IRQ #	SH-4A Vector Description
0x2f	TMU2 IC IRQ
0x30	Hitachi-UDI
0x32	DMAC INT0 IRQ
0x33	DMAC INT1 IRQ
0x34	DMAC INT2 IRQ
0x35	DMAC INT3 IRQ
0x36	DMAC DMAE IRQ
0x38	SCIF0 ERI IRQ
0x39	SCIF0 RXI IRQ
0x3a	SCIF0 BRI IRQ
0x3b	SCIF0 TXI IRQ
0x3c	DMAC INT4 IRQ
0x3d	DMAC INT5 IRQ
0x3e	DMAC INT6 IRQ
0x3f	DMAC INT7 IRQ
0x40	General FPU disable exception
0x41	Slot FPU disable exception
0x48	CMT IRQ
0x4c	HAC IRQ
0x50	PCI SERR IRQ
0x51	PCI INTA IRQ
0x52	PCI INTB IRQ
0x53	PCI INTC IRQ
0x54	PCI INTD IRQ
0x55	PCI ERR IRQ
0x56	PCI PWD3 IRQ
0x57	PCI PWD2 IRQ
0x58	PCI PWD1 IRQ
0x59	PCI PWD0 IRQ
0x5c	SCIF1 ERI IRQ
0x5d	SCIF1 RXI IRQ
0x5e	SCIF1 BRI IRQ
0x5f	SCIF1 TXI IRQ
0x60	SIOF IRQ
0x64	HSPI IRQ
0x68	MMCIF FSTAT IRQ
0x69	MMCIF TRAN IRQ
0x6a	MMCIF ERR IRQ
0x6b	MMCIF FRDY IRQ
0x6c	DMAC INT8 IRQ

Table 2-1. OS-9 IRQ Assignment

OS-9 IRQ #	SH-4A Vector Description
0x6d	DMAC INT9 IRQ
0x6e	DMAC INT10 IRQ
0x6f	DMAC DMINT11 IRQ
0x70	TMU3 IRQ
0x71	TMU4 IRQ
0x72	TMU5 IRQ
0x74	SSII IRQ
0x78	FLCTL FLSTE IRQ
0x79	FLCTL FLTEND IRQ
0x7a	FLCTL FLTRQ0 IRQ
0x7b	FLCTL FLTRQ1 IRQ
0x7c	GPIOI0 IRQ
0x7d	GPIOI1 IRQ
0x7e	GPIOI2 IRQ
0x7f	GPIOI3 IRQ

Port Specific Utilities

Utilities for the MS7780SE boards are located in the following directory:

`MWOS/OS9000/SH4A/PORTS/SH7780/CMDS`

The following port specific utilities are included:

dmppci	Shows PCI device information.
pciv	Displays board PCI bus information.
pflash	Programs On-board flash.
setpci	Pokes PCI device settings.

dmppci
Show PCI Information

Syntax

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

Description

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

The following is an example display of an Intel Ethernet Pro 10/100 PCI board:

```
$ dmppci 0 12 0

    PCI DUMP Bus:0 Dev:12 Func:0 Size:64
    ------------------------------------

VID  DID  CMD  STAT CLASS  RV CS IL IP LT HT BI MG ML SVID SDID
---  ---- ---- ---- -----  -- -- -- -- -- -- -- -- -- ---- ----
8086 1229 0007 0290 020000 08 08 53 01 20 00 00 08 38 8086 000c

BASE[0]  BASE[1]  BASE[2]  BASE[3]  BASE[4]  BASE[5]  CIS_P    EXROM
-------- -------- -------- -------- -------- -------- -------- --------
08100000 0ff00001 08000000 00000000 00000000 00000000 00000000 00000000

Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
------------------------------------------------------
0000   86 80 29 12 07 00 90 02 08 00 00 02 08 20 00 00
0010   00 00 10 08 01 00 f0 0f 00 00 00 08 00 00 00 00
0020   00 00 00 00 00 00 00 00 00 00 00 00 86 80 0c 00
0030   00 00 00 00 dc 00 00 00 00 00 00 00 53 01 08 38
```

pciv
PCI Configuration Space View

Syntax

```
pciv {[options]}
```

Options

`-a`	Display base address information and size.
`-r`	Display PCI routing information.
`-i`	Show class information.

Description

The `pciv` utility allows visual indication of the status of the PCI bus.

The following is an example display:

```
$ pciv -a

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:12:00  8086 1229 0007 0290 020000 08 08 53 01
(C)  [32-bit] base_addr[0] = 0x08100000  PCI/MEM 0xa8100000 Size = 0x00001000
(C)  [32-bit] base_addr[1] = 0x0ff00001  PCI/IO  0xaff00000 Size = 0x00000040
(C)  [32-bit] base_addr[2] = 0x08000000  PCI/MEM 0xa8000000 Size = 0x00100000
Network Controller [S]
```

pflash

Program Toshiba Flash

Syntax

```
pflash {[options]} <filename>
```

Options

`-b[=]addr`	Specify base address of flash (hex). The default is 0xa1000000.
`-eu`	Erase used space only. This is the default mode of operation.
`-ew`	Erase all of flash. This will erase all TrueFFS and/or OS-9 boot code stored in the flash.
`-i`	Print information about flash.
`-ne`	Do not erase flash prior to programming.
`-nv`	Do not verify erase or write operations.
`-q`	Do not display progress messages and spinning indicator.
`-s[=]addr`	Specify write address for `<filename>` in hexadecimal. The default start address is 0xa100000.
`-u`	Leave flash part unlocked upon completion.
`-z[[=]<file>]`	Read additional command line arguments from <file>. The default <file> is standard input.

Description

The `pflash` utility allows programming of the Fujitsu flash parts. The primary use is in burning the OS-9 ROM or coreboot image into the on-board flash parts. This allows for booting using the lr/bo booters.

`pflash` requires `<filename>`, `-i`, and/or `-ew`.

`<filename>`'s size need not be a multiple on the flash's erase block size. `pflash` will only reprogram the part of the flash that `<filename>` will occupy. The remainder of the flash remains undisturbed. This feature allows for small updates within flash erase blocks.

For this board, `pflash` does no locking nor unlocking of the flash parts. Thus, `-u` has no effect.

setpci
Set PCI Value

Syntax

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

Description

The `setpci` utility sets PCI configuration information not normally available by other means, other than programming with the PCI library. The `setpci` utility can also 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 (e.g. command register offset = 4).
`<size>`	Size b = byte, w = 16-bit word, or d = 32-bit double word.
`<value>`	An optional value to write. If no `<value>` is specified, `setpci` will read and display the value at the specified offset.

A Board Specific Modules

This chapter describes the modules specifically written for the Toshiba RBHM4x00 board. It includes the following sections:

Port-Specific Low-Level System Modules

The following low-level system modules are tailored specifically for the Renesas MS7780SE board. They are located in the following directory:

`MWOS/OS9000/SH4A/PORTS/SH7780SE/CMDS/BOOTOBJS/ROM`

`cnfgdata`	contains low-level configuration data
`cnfgfunc`	provides access services to the `cnfgdata`
`commcnfg`	inits communication port defined in `cnfgdata`
`conscnfg`	inits console port defined in `cnfgdata`
`initext`	user-customizable system initialization module
`ioscifsh7780`	low-level serial driver for SCIF serial ports
`ll91c111`	low-level Ethernet driver module for the on-board SMSC 91c111 chip
`llpro100`	low-level Ethernet driver module for the Intel Ethernet Pro 10/100 PCI board
`pciwalk`	inits devices found on the PCI bus
`portmenu`	inits booters defined in the `cnfgdata`
`romcore`	bootstrap code for the MS7780SE
`rpciv`	low-level booter used to display PCI bus information
`sh4atimer`	low-level timer module for SH-4A processors
`usedebug`	debugger configuration module

Common Low-Level System Modules

The following low-level system modules provide generic services for OS-9 Modular ROM. They are located in the following directory:

`MWOS/OS9000/SH4A/CMDS/BOOTOBJS/ROM`

`bootsys`	provides booter registration services
`console`	provides console services
`dbgentry`	inits debugger entry point for system use
`dbgserv`	provides debugger services
`excption`	provides low-level exception services
`flshcach`	provides low-level cache management services
`hlproto`	provides user level code access to protoman
`llbootp`	provides `bootp` services
`llip`	provides low-level IP services
`llkermit`	provides a booter that uses kermit protocol

`llslip`	provides low-level SLIP services
`lltcp`	provides low-level TCP services
`lludp`	provides low-level UDP services
`notify`	provides state change information for use with LL and HL drivers
`override`	provides a booter that allows a choice between menu and auto booters
`parser`	provides argument parsing services
`protoman`	provides a protocol management module
`restart`	provides a booter that causes a soft reboot of the 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

Port-Specific High-Level System Modules

The following OS-9 system modules are tailored specifically for the Renesas MS7780SE board. Unless otherwise specified, each module is located in the following directory:

`MWOS/OS9000/SH4A/PORTS/SH7780SE/CMDS/BOOTOBJS`

`INITS/nodisk`	system configuration module for a diskless OS-9 boot
`abort`	handler for the abort button
`picsub`	programmable interrupt controller handling subroutine module
`pwrext`	extension module for power management
`pwrplcy`	power management policy implementation module
`rbftl`	driver for flash file system device
`tk3927`	system ticker module
`rtc7780`	battery backed real-time clock module for SH-4A
`sc7780`	serial port driver for SCIF ports. The descriptors for this driver are found in the `DESC/SC7780` sub-directory.

SCIF0 (lower DB9 connector) Descriptors

`term1` - /term descriptor with XON/XOFF flow control

`term1_hw` - /term descriptor with software implemented hardware flow control

`term1_auto` - /term descriptor with hardware implemented

hardware flow control

`t1` - /t1 descriptor with XON/XOFF flow control

`t1_hw` - /t1 descriptor with software implemented hardware flow control

`t1_auto` - /t1 descriptor with hardware implemented hardware flow control

`ps_t1` - /ps descriptor for a serial printer

`ps_t1_hw` - /ps descriptor for a serial printer, with software implemented hardware flow control

`ps_t1_auto` - /ps descriptor for a serial printer, with hardware implemented hardware flow control

`m0_t1` - /m0 descriptor for a serial mouse

SCIF1 (upper DB9 connector) Descriptors

`term2` - /term descriptor

`t2` - /t2 descriptor

`ps_t2` - /ps descriptor for a serial printer

`m0_t2` - /m0 descriptor for a serial mouse

Baud Rates

The following baud rates are supported by the `sc7780` driver: 50, 75, 110, 135, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200, 31250, 38400, 56000, 57600, 64000, and 115200.

`sysif`	system interface module for power management
`tk7780`	system clock tick handler module, implementing the ticker using one of the TMU tickers (default = TMU0).
`tkrtc`	system clock tick handler module, implementing the ticker using the periodic RTC interrupt for the ticker (hard-coded at 256 ticks per second).
`vectsh7780`	exception and interrupt handling code for SH-4A 7780 series processors.

Port-Specific High-Level Utilities

The following MS7780SE-specific programs are provided. For more information about their functions and syntax, enter the `-?` command-line option. They are located in the following directory:

`MWOS/OS9000/SH4A/PORTS/SH7780SE/CMDS`

`dmppci`	Allows a specific PCI device's configuration area to be displayed.
`pciv`	Displays configuration information about all available PCI

	devices.
`pflash`	Programs the on-board Fujitsu flash device.
`setpci`	Allows changes to the PCI configuration of devices.

Common High-Level System Modules

These files are located in the following directory:

`MWOS/OS9000/SH4A/CMDS/BOOTOBJS`

`kernel`	Provides all basic services for the OS-9 system.
`cache`	Provides cache control for the CPU cache hardware.
`fpuexcpt`	Provides handling of the floating-point hardware.
`ioman`	Provides common I/O support for the operating system.
`mq`	Device descriptor for inter-process message queues.
`msgman`	File manager that provides support for inter-process message queues.
`pcf`	Random block device management functions for MS-DOS FAT format.
`pipeman`	Memory FIFO buffer management for inter-process communication.
`rbf`	Generic random block device management functions for the OS-9 disk format.
`scf`	Sequential character device management functions.
`spf`	Generic protocol device management function support.
`ssm`	System Security Module—provides support for the CPU's MMU (Memory Management Unit).
`trans`	User/System translation module. This module translates user-state (P0) addresses to system-state (P1) addresses and vice versa.