Home

OS-9® for TQM85xx Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com
Revision A • July 2006

Copyright and publication information

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

July 2006

Table of Contents

Chapter 1: Installing and Configuring OS-9®

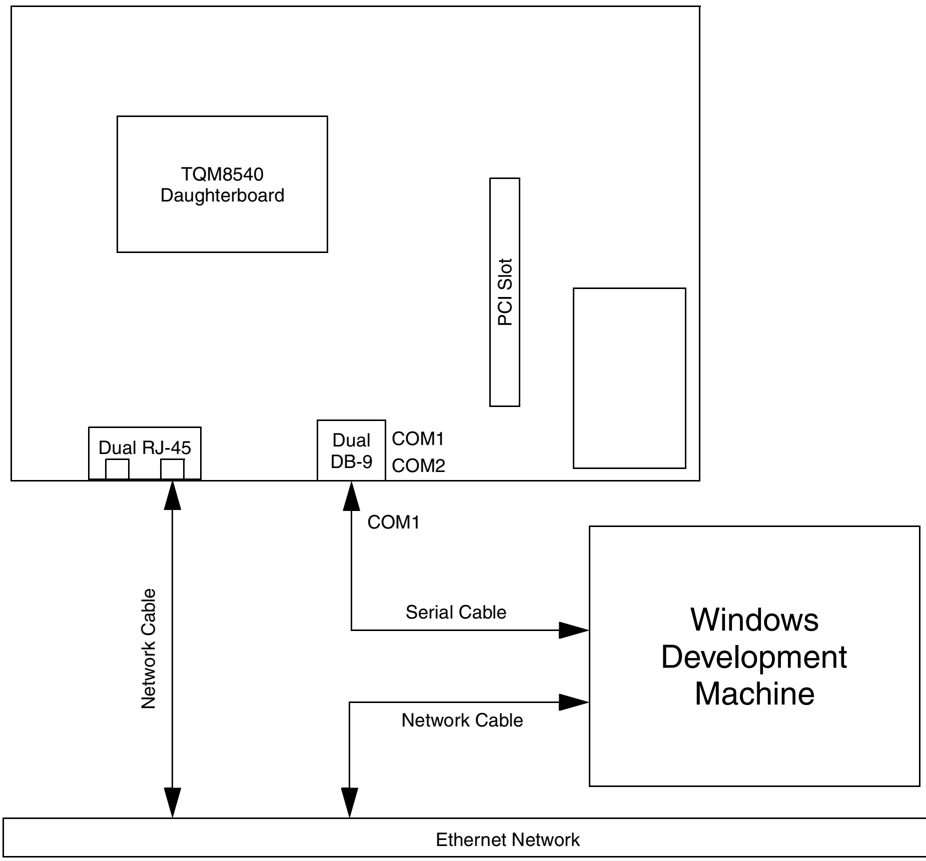
This chapter describes installing and configuring OS-9® on the TQ Components STK85xx target board and the TQM8540 daughterboard (hereafter referred to as the TQM85xx). The following sections are included:

- **Development Environment Overview**
- **Requirements and Compatibility**
- **Connecting the Target to the Host**
- **Building the OS-9 ROM Image**
- **Transferring the coreboot Image to the Target**
- **Optional Procedures**

Development Environment Overview

Figure 1-1 shows the development environment for the TQM85xx. The components shown include the minimum required to enable OS-9 to run on the board.

Figure 1-1 TQM85xx Development Environment



Requirements and Compatibility

Note

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

Host Hardware Requirements (PC Compatible)

Your host PC must meet the following minimum requirements:

- Windows 98 SE, ME, 2000, NT, or XP
- 300-400 MB of free disk space
- 64MB of RAM

Host Software Requirements (PC Compatible)

Your host PC must have the following applications installed:

- a terminal emulation program such as Hyperterminal, which comes with Microsoft Windows, or Tera Term Pro.
- a BOOTP/TFTP server for booting OS-9. Refer to the Optional Procedures section for information about burning OS-9 into the Flash memory, thus eliminating the need for network booting.

Target Hardware Requirements

Your target system requires the following hardware:

- TQ TK85xx base board
- TQM8540 CPU daughter board
- Standard ATX power supply

Special Application Considerations

After booting OS-9 initially from the sample boot explained in this chapter, you should reconfigure the system to more directly fit your application requirements.

For More Information

The ***OS-9 Device Descriptor and Configuration Module Reference*** manual included with your software distribution contains information to help you understand the purpose of each of the modules contained in this distribution and the variety of ways that the software can be configured to meet your needs.

Connecting the Target to the Host

This section describes connecting the target board to the host PC via serial and Ethernet connections.

Complete the following steps to connect the target to the host:

Step 1. Connect the target's RS-232 COM1 port (the upper DB-9 connector of the pair in the middle base board) to an unused RS-232 COM port on your Host PC using a serial cable.

Step 2. Connect the target board to an Ethernet network with a network cable plugged into the right RJ-45 of the pair nearest the left edge of the base board. Your Host PC must also be connected to a network.

Step 3. Start Hyperterminal on the Host PC by selecting `Start -> Programs -> Accessories -> Hyperterminal`.

Step 4. Enter a name for your Hyperterminal session.

Step 5. Select an icon for the new Hyperterminal session. A new icon will be created with the name of your session associated with it.

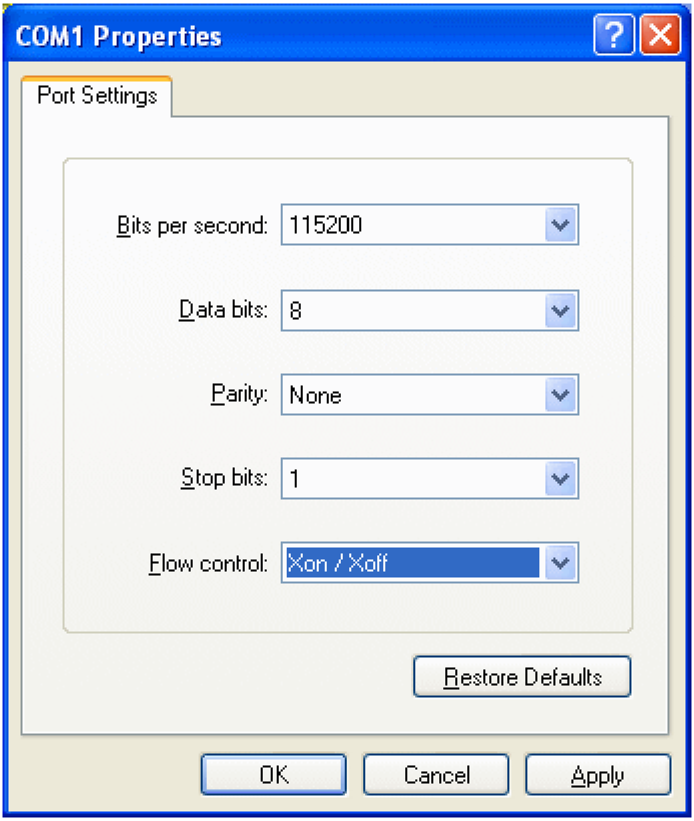
Step 6. Click `OK`.

Step 7. In the **Connect To** dialog box, go to the **Connect using** pull-down menu and enter the communications port to be used to connect to the target system.

Step 8. Click `OK`.

Step 9. Configure the **Port Settings** tab, as shown in **Figure 1-2**.

Figure 1-2 COM Port Settings



Step 10. Click `OK`.

Step 11. In the Hyperterminal window, select `File/Properties`. Click on the **Settings** tab and select the following:

```
Terminal Keys
Emulation = Auto Detect
Backscroll Buffer Lines = 500
```

Step 12. Click `OK`.

Step 13. Go to the Hyperterminal menu and select `Call/Connect` from the pull-down menu to establish your terminal session with the target. If you are connected, the bottom left corner of your Hyperterminal screen will display the word *Connected*.

Step 14. Leave the Hyperterminal window open on your desktop (or minimized); you will use the window again later in this procedure.

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 (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.

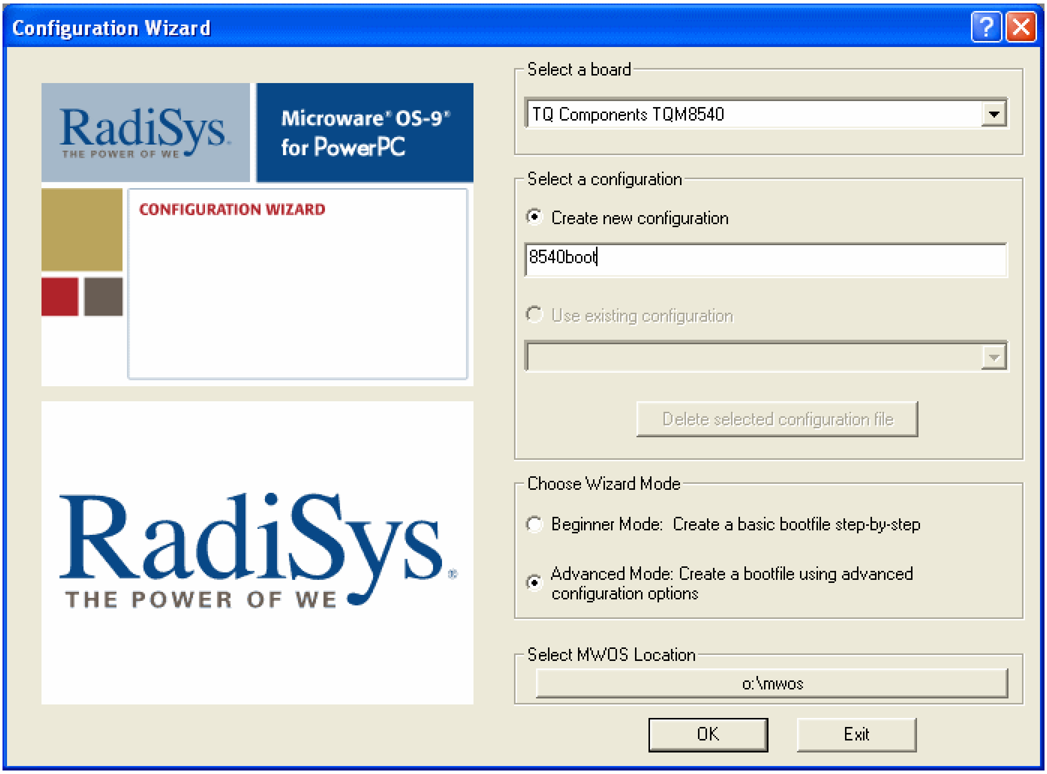
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the OS-9 installation process.

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:

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

Figure 1-3 Configuration Wizard Opening Screen

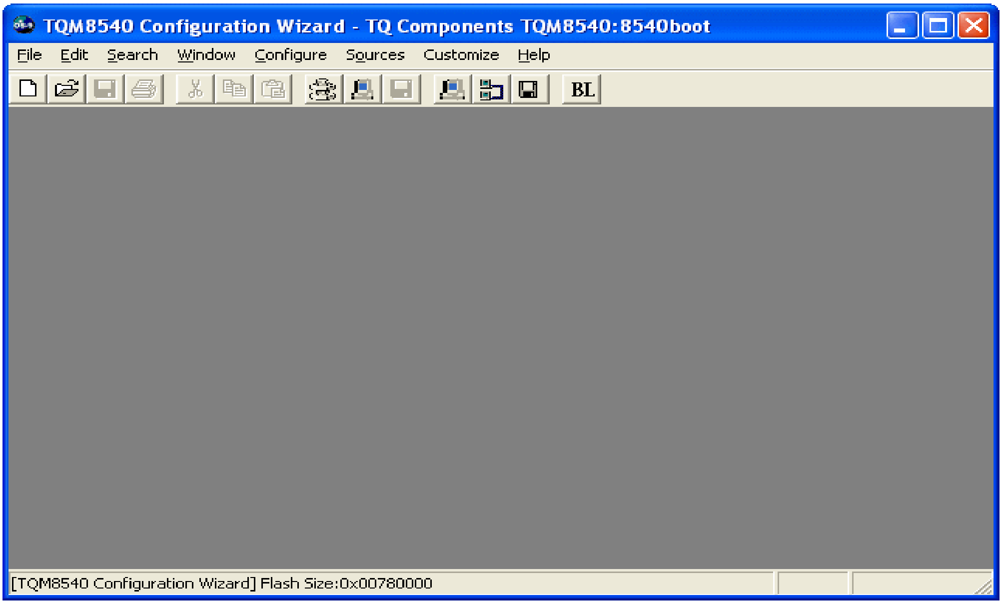


Step 2. Select your target board from the **Select a board** pull-down menu.

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 1-4**.

Figure 1-4 Configuration Wizard Main Window



Creating and Configuring the ROM Image

This section describes how to use the Configuration Wizard to create and configure your OS-9 ROM image.

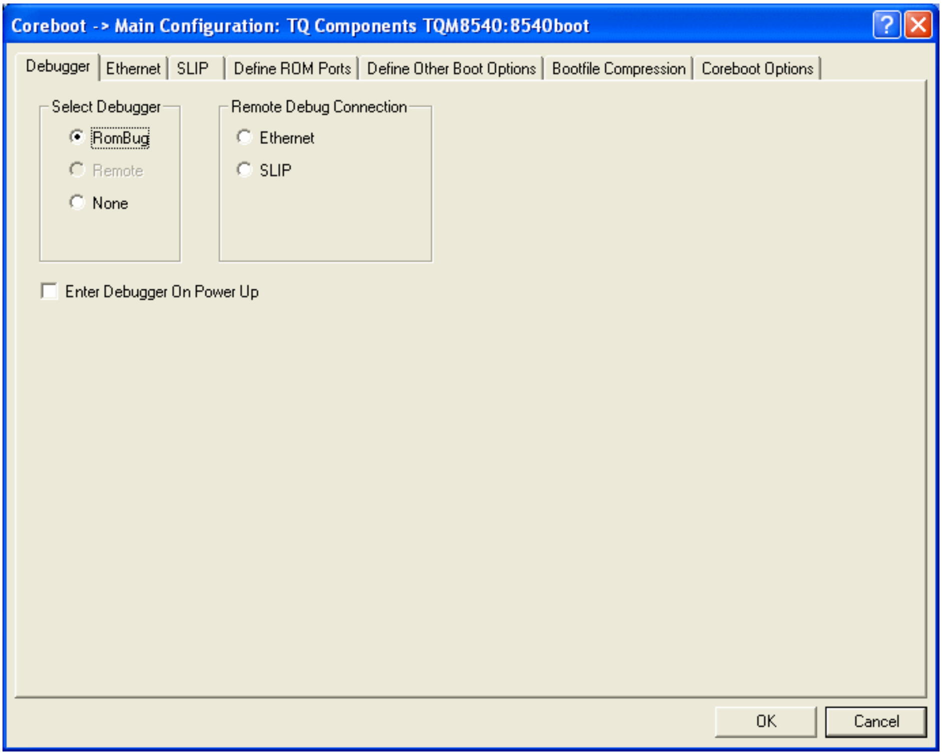
Note
This section provides an example of an OS-9 ROM image successfully built on a Host PC and transferred to your target board. You may have to modify your selections depending on your application.

Configure Coreboot Options

Step 1. From the **Main Configuration** window, select `Configure` -> `Coreboot` -> `Main configuration`.

Step 2. Select the **Debugger** tab. The following window is displayed.

Figure 1-5 Coreboot Configuration—Debugger Tab



Step 3. Under **Select Debugger**, select `RomBug`. This sets Ethernet as the method for user state debugging. Select `None` if you do not want to debug your system.

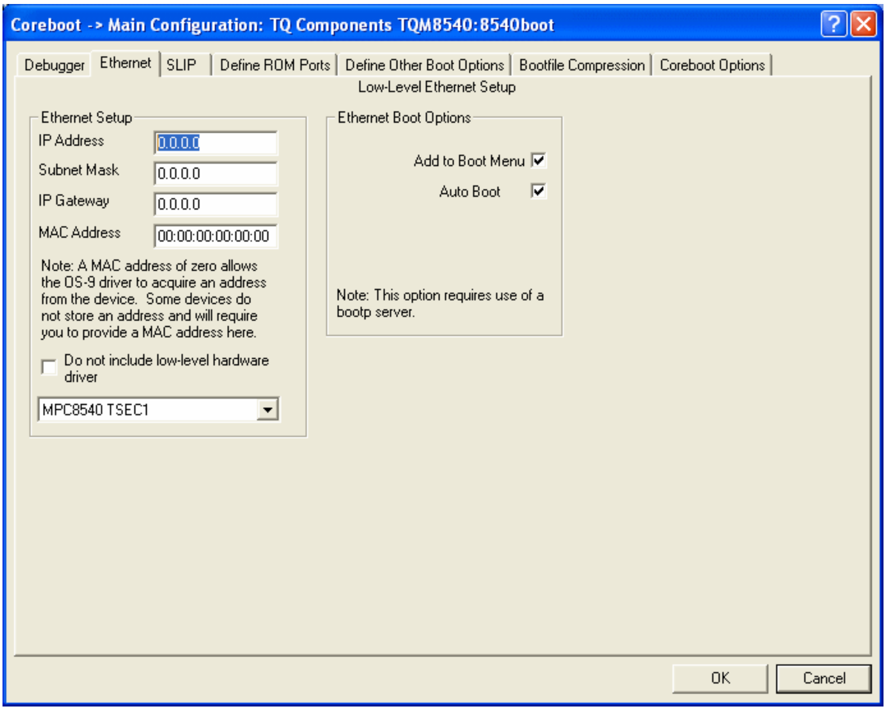
Note

To perform system state debugging, select `Ethernet` under **Remote Debug Connection**. If you set Ethernet as the method for system state debugging, you will not be able to perform user state debugging via Ethernet.

For system state debugging, you must also set the parameters in the **Ethernet** tab of the coreboot configuration.

Step 4. Select the **Ethernet** tab. The following window is displayed.

Figure 1-6 Coreboot Configuration—Ethernet Tab



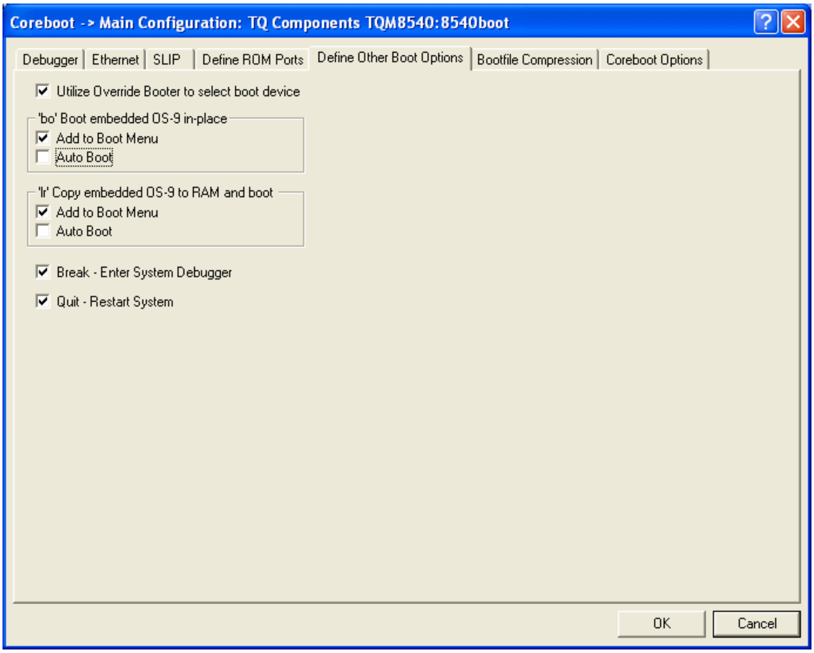
Step 5. Enter the appropriate Ethernet setup information. Leave the MAC Address as 00:00:00:00:00:00 to have the low-level Ethernet driver obtain the MAC address from the hardware. Check both the **Add to Boot Menu** and **Auto Boot** checkboxes.

Note

The addresses shown in **Figure 1-6** are for demonstration only. Contact your network administrator to obtain your Ethernet setup information.

Step 6. Select the **Define Other Boot Options** tab. The following window is displayed.

Figure 1-7 Coreboot Configuration—Define Other Boot Options Tab



Step 7. Check the boxes as shown above.

Step 8. Click the **Coreboot Options** tab and enable all the optional coreboot options.

Step 9. Click `OK` and return to the **Main Configuration** window.

Configure System Options

Configure system options by selecting `Configure -> Bootfile -> Configure System Options` from the **Main Configuration** window. You can bypass this option and use the default settings.

To use the target board across a network, you must configure the Ethernet network settings.

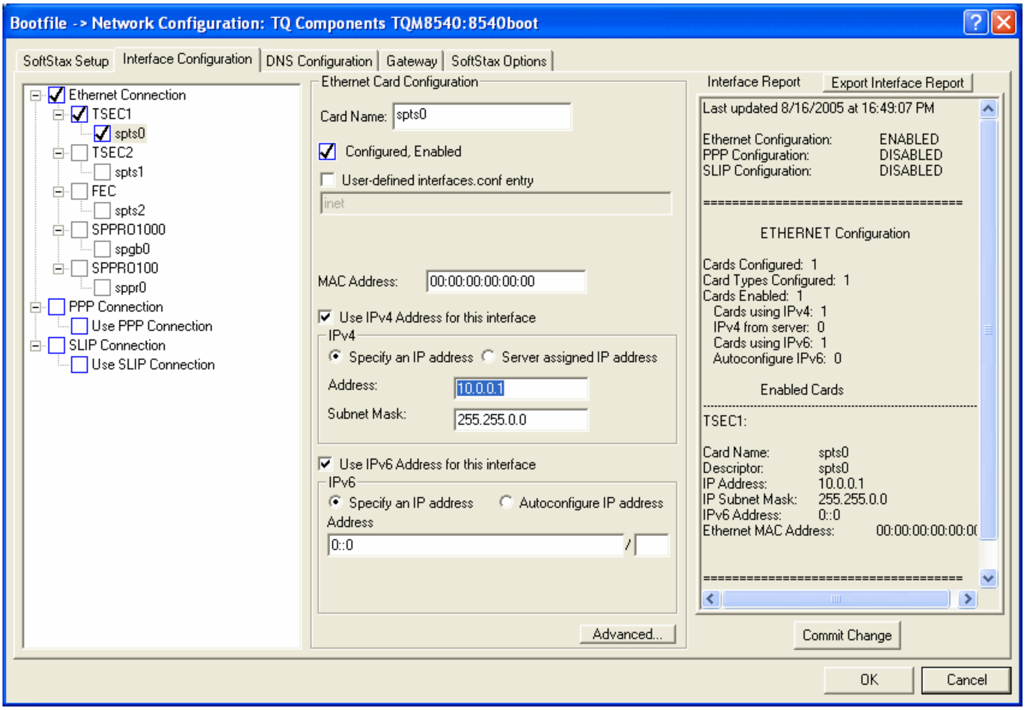
Network Configuration

To use the target board across a network, complete the following steps:

Step 10. 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 11. 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 1-8 shows an example of the **Interface Configuration** tab.

Figure 1-8 Bootfile -> Network Configuration -> Interface Configuration



For More Information

To learn more about IPv4 and IPv6 functionalities, refer to the ***Using LAN Communications*** 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 12. Once you have made your settings in the **Network Configuration** dialog, click `OK`.

Step 13. Select the `DNS Configuration` tab. The following window is displayed. More than one DNS server can be added in this dialog box. If your network does not use DNS, click `Disable DNS` and move to the **Gateway** tab. If you have DNS available, click `Enable DNS` and type your host name and domain.

Note

You add DNS IP addresses by clicking on the box directly under **DNS Server Search Order** and typing the IP address. Click the `Add` button when complete.

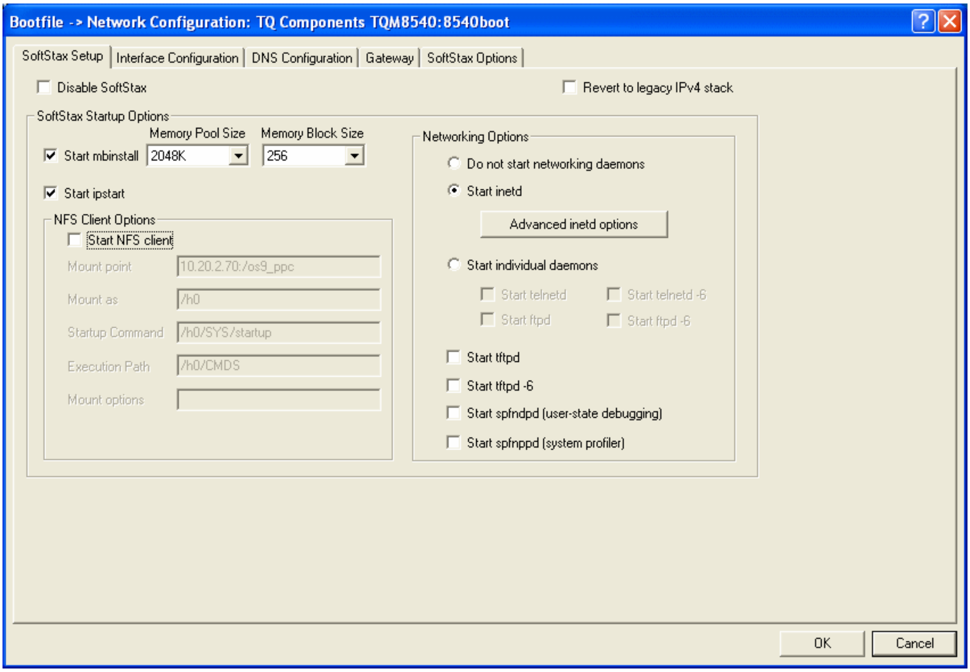
More than one DNS server can be added by repeating these steps.

Step 14. Select the **Gateway** tab. Add new gateway addresses by clicking on the box and typing in the gateway name. Click the `Add` button when complete.

Step 15. Select the `SoftStax® Setup` tab. The following window is displayed.

The options below represent daemons that can be automatically started if you want to FTP or telnet from a PC to the OS-9 target. NFS Client Options enable you to mount remote systems from your target.

Figure 1-9 Bootfile Configuration—SoftStax® Setup Tab



Step 16. Make sure it looks like the window above. The larger memory pool size and blocking factor are recommended to improve network performance.

Step 17. Select the **SoftStax Options** tab.

The **SoftStax Options** tab enables you to include networking utilities in the ROM image. By default, `ftp`, `hostname`, `ping`, and `netstat` are included. You can add other utilities as desired.

For More Information

The networking utilities are described in the ***Using LAN Communications*** manual.

Step 18. Click `OK` at the bottom of the **Network Configuration** dialog to complete network configuration and return to the **Main Configuration** window.

Disk Configuration

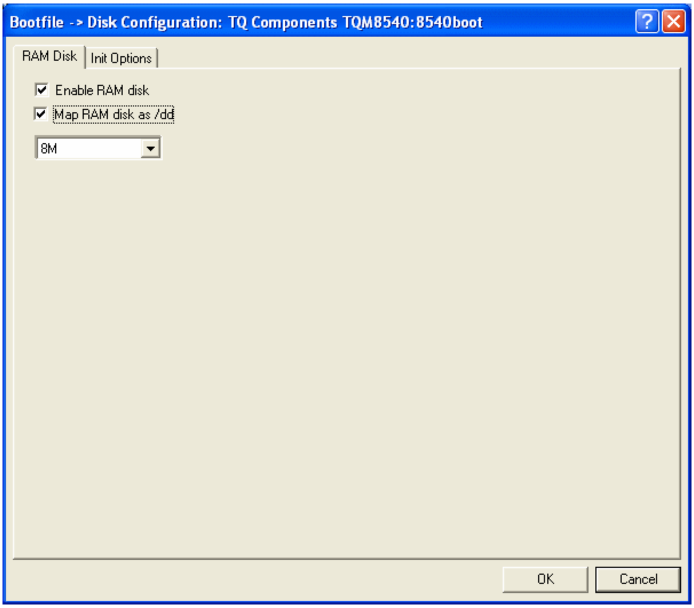
Step 1. From the main configuration window, select `Configure` -> `Bootfile` -> `Disk Configuration`.

The Disk Configuration options include the following two tabs:

- The **RAM Disk** tab enables you to specify RAM disk properties. The RAM disk is required for the FTP server.
- The **Init Options** tab allows configuration of the parameters that OS-9 uses to initialize itself on the target.

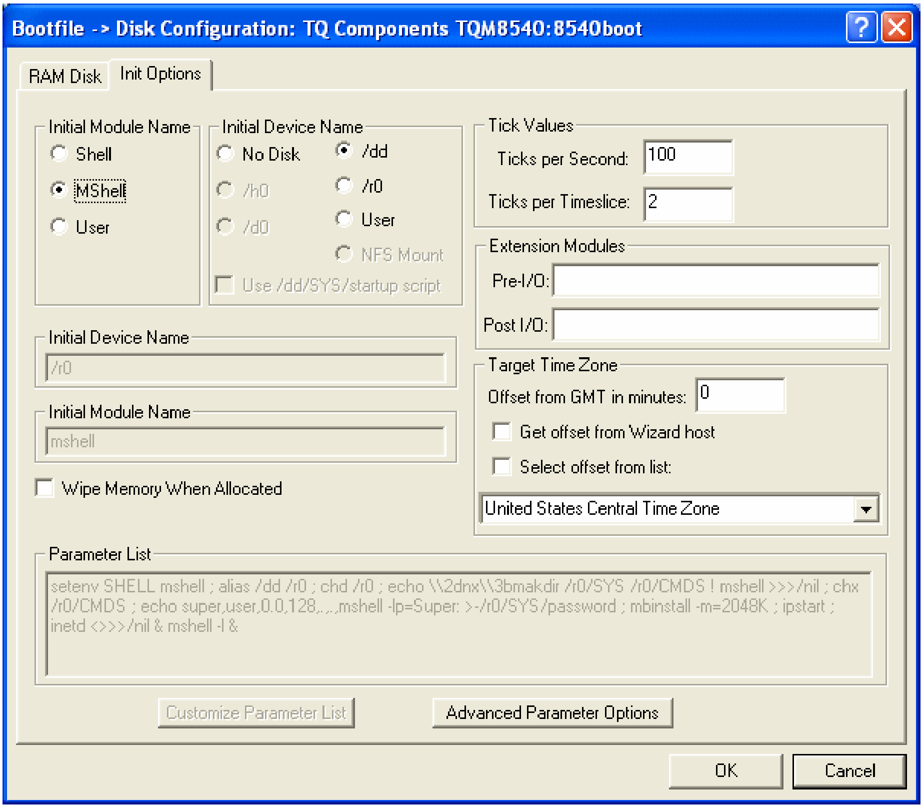
Step 2. Configure the RAM disk as shown below:

Figure 1-10 Bootfile Configuration—RAM Disk Options Tab



Step 3. Select the **Init Options** tab. The following window is displayed.

Figure 1-11 Bootfile Configuration—Init Options Tab



Step 4. Select the `Mshell` option for the initial module name. This causes OS-9 to start a console shell usable from your terminal window. Initial **Device Name** should be selected as `/dd`.

The tick rate is `100` and ticks per timeslice is set to `2`. If you look at the **Parameter List** box, you see the commands that OS-9 executes upon system start-up. These commands initialize networking and enable the FTP server to accept connections.

Step 5. Click `OK` to return to the **Main Configuration** window.

Build Image

Complete the following steps to build the target board image.

Step 1. From the **Main Configuration** window, select `Configure -> Build Image`. The **Master Builder** window appears.

Step 2. Select the **Coreboot Only Image** option.

Step 3. Click `Build`. It should display progress information and show the statistics of the image just created.

Step 4. The coreboot and coreboot.S (Motorola s-record) files are created in the following directory:

```
MWOS/OS9000/E500/PORTS/TQM85XX/BOOTS/INSTALL/PORTBOOT
```

Step 5. Select the **Bootfile Only Image** option. Uncheck the **Compress Bootfile** option. Also, select the **User State Debugging** checkbox.

Step 6. Click `Build`. It should display progress information and show the statistics of the image just created.

Step 7. Click Save As and save the newly created bootfile in a location that your BOOTP server can access.

At this point you can either close the Configuration Wizard or leave it open for use in modifying your ROM Image. If you choose to close, you can save your configuration settings for later use.

Transferring the coreboot Image to the Target

Complete the following steps to transfer your coreboot image to the reference board.

For More Information

This process uses the TQ MON85xx software. For more information about this software refer the manual supplied by TQ Components.

Step 1. Utilize your existing Hyperterminal connection to the target.

Step 2. Reset the target hardware by pushing the reset button located near the serial connector. Press Enter several times in Hyperterminal after pressing reset. You will get a monitor prompt:

```
MON85xx.100 on TQM8540 - (C) TQ-Systems 1998-2004
Clock speeds ( CPU /  CCB /  Bus [MHz] )
Maximum        833 /  333 /   42
Current        834 /  333 /   42

POST 1 skipped

MON:>
```

Step 3. At the MON: prompt, issue the command to erase the first 512K of the Flash memory:

```
MON:>erase 80000000 8007ffff
* Erasing FLASH from 80000000h to 8007FFFFh
* Please wait

MON:>
```

Step 4. In preparation of downloading the ASCII S-record file, turn off the echoing of characters by the target:

```
MON:>echo off
```

```
MON:>
```

Step 5. Issue the command to load the Flash starting at offset 0:

```
MON:>load 0 f (not visible)* Ready for s-record download to FLASH
...
```

Step 6. Initiate the ASCII file transfer in Hyperterminal by choosing the menu option `Transfer -> Send Text File...` Change the file type on the file chooser dialog to `All files` and choose the file \MWOS\OS9000\E500\PORTS\TQM85XX\BOOTS\INSTALL\PORTBOOT\coreboot.S.

Hyperterminal will transfer the s-records to the target and the monitor will program the binary data into the Flash.

```
* Application loaded
* S-Record Start address    00000000
* Application Start address 00000000

MON:>
```

Step 7. After the transfer finishes, reset the board and press `Enter` several times in the Hyperterminal window to restore echo:

```
*** press reset button ***
<Enter>
<Enter>


MON85xx.100 on TQM8540 - (C) TQ-Systems 1998-2004
Clock speeds ( CPU /  CCB /  Bus [MHz] )
Maximum        833 /  333 /   42
Current        833 /  333 /   42

POST 1 skipped

MON:>
```

Step 8. Transfer control to the OS-9 coreboot by entering the go command below. The board will fail to boot because you have not yet configured your BOOTP server with the IP and MAC address of your target.

```
MON: >go 80000000
* Starting application at 80000000
```

```
OS-9 Bootstrap for the PowerPC(tm) (Edition 68)

LLTSEC: PHY(0) is Marvell 88E1101 ($1410cc1)
LLTSEC: Full Duplex
LLTSEC: Speed 100BT
LLTSEC: Link is up
Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Boot over Ethernet TSEC1.
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 1/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 2/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 3/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 4/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 5/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 6/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 7/8
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 8/8
bootp: Exiting after 8 tries
Boot over Ethernet TSEC1 error #000:007:000:025.


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

Boot over Ethernet TSEC1 -------------- <eb>
Boot embedded OS-9000 in-place -------- <bo>
Copy embedded OS-9000 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:
```

Step 9. Use the MAC address shown by your board and the IP address that you previously configured to setup your BOOTP server to serve the bootfile previously created to your target. Once configured correctly, a correct boot should look something like this:

```
MON:>go 80000000
* Starting application at 80000000


OS-9 Bootstrap for the PowerPC(tm) (Edition 68)
```

```
LLTSEC: PHY(0) is Marvell 88E1101 ($1410cc1)
LLTSEC: Full Duplex
LLTSEC: Speed 100BT
LLTSEC: Link is up
Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Boot over Ethernet TSEC1.
bootp: 00:00:5b:01:b0:8e broadcasting for server...try 1/8
bootp: Server host name:
bootp: My IP Address will be: 10.20.3.205
bootp: My Bootfile is: .\os9kboot
bootp: My bootfile size is: 00000f02 (512-byte) blocks
bootp: My subnet mask is: 255.255.254.0
bootp: <<no timeoffset tag>>
bootp: <<no client host name specified>>
bootp: <<no gateway tag>>
bootp: Next Server address: 10.20.2.65
bootp: Using Server assigned bootfile size of 001e0400 bytes
tftp: Starting tftp transfer...
tftp: Block size: 000005b0
tftp: Boot file size: 001e0368
tftp: received file with 001e0368 bytes

Bootfile received from server 10.20.2.65
Now searching for an OS-9 kernel...
A valid OS-9 bootfile was found.
+3
+5
$
```

Optional Procedures

The following section provides optional procedures you can perform after installing and configuring OS-9 on your board.

Burning Complete ROM Image in Flash

Once you have established an OS-9 prompt on your target system using the coreboot and Ethernet booting, you can perform the following procedure to program the Flash with an entire ROM image (coreboot and bootfile):

Step 1. Start the configuration Wizard if it is not already running. Reuse your existing configuration from the previous exercise.

Step 2. Ensure the RAMdisk is at least 4MB. Choose `Configure -> Bootfile -> Disk Configuration` and select a size of at least 4 MB. Click OK to save the setting. Reboot with this bootfile.

Step 3. From the **Main Configuration** window, select `Configure -> Build Image`. The **Master Builder** window appears.

Step 4. Select the **Coreboot + Bootfile** option.

Step 5. Check the **User State Debugging** option.

Make sure the Compress Bootfile option is selected. Using a compressed boot ensures that modules will not be found in Flash when you boot via BOOTP.

A compressed bootfile, of course, can not be executed directly from Flash.

Step 6. Click `Build`. It should display progress information and show the statistics of the image just created.

Step 7. The `rom` file is created in the following directory:

```
MWOS/OS9000/E500/PORTS/TQM85XX/BOOTS/INSTALL/PORTBOOT
```

Step 8. FTP this `rom` file to your target's `/r0` device (RAMdisk).

Step 9. On your target (Hyperterminal window) program the `rom` file into the Flash with the `pflash` utility.

```
$ pflash /r0/rom
Unlocking Flash from 0x80000000 to 0x801a0000
Programming /r0/rom at 0x80000000
Locking entire Flash part
$
```

Step 10. Press the reset button and reboot.

```
*** press reset button ***
<Enter>
<Enter>

MON85xx.100 on TQM8540 - (C) TQ-Systems 1998-2004
Clock speeds ( CPU /  CCB /  Bus [MHz] )
Maximum        833 /  333 /   42
Current        835 /  334 /   42

POST 1 skipped

MON:>go 80000000
* Starting application at 80000000


OS-9 Bootstrap for the PowerPC(tm) (Edition 68)

Now trying to Override autobooters.

Press the spacebar for a booter menu
<spacebar>


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

Boot over Ethernet TSEC1 -------------- <eb>
Boot embedded OS-9000 in-place -------- <bo>
Copy embedded OS-9000 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
```

```
Compressed bootfile found at $80080000
A valid OS-9 bootfile was found.
+3
+5
$
```

Chapter 2: Board Specific Reference

This chapter contains information that is specific to the TQ Components TQM85xx board. It contains the following sections:

- **Boot Options**
- **OS-9 Vector Mappings**
- **SPE Floating-point Support**

For More Information

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

Boot Options

Select your boot device menu options using the Configuration Wizard. For each boot device option, you can select whether you want it to be displayed on a boot menu, set up to autoboot, or both. The autoboot option enables the device selected to automatically boot up the high-level bootfile, bypassing the boot device menu.

Note

When using the Configuration Wizard, you should select only one device for autoboot on your system.

Following is an example of the Boot menu displayed in the terminal emulation window (using Hyperterminal):

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

Boot over Ethernet TSEC1 -------------- <eb>
Boot embedded OS-9000 in-place -------- <bo>
Copy embedded OS-9000 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:
```

Your boot option selections in the Configuration Wizard determine which modules are included in the coreboot image. Table 2-1 lists some of the supported boot devices for OS-9.

Table 2-1 Supported Boot Methods

Type of Boot	Description
Boot over Ethernet TSEC1	Use BOOTP and TFTP protocols to download a bootfile (`eb`).
Boot embedded OS-9 in-place	Boot OS-9 from FLASH (`bo`).
Copy embedded OS-9 to RAM and Boot	Copy OS-9 from FLASH (if stored there) to RAM and boot (`lr`).
Kermit download	Initiate a kermit download of the bootfile (`ker`).
PCI View Utility	Execute a scan of the local PCI bus and report on the devices found. The boot menu will return after the scan (`pciv`).
Enter system debugger	Drop into RomBug. From RomBug you can enter g, for go, and booting will proceed as normal. Refer to the RomBug manual for more information (`break`).
Restart the System	Execute a system reboot (`q`).

OS-9 Vector Mappings

This section contains the vector mappings for the MPC85xx.

The system module `openpicirq` maps interrupts coming from the PIC into the OS-9 vector table according to the following mappings.

PIC vectors are mapped starting at vector 0x40 in the order shown in the following table.

Table 2-2 PIC Vectors

Vector	Source
0x40	L2 Cache
0x41	ECM
0x42	DDR
0x43	LBIU
0x44	DMA0
0x45	DMA1
0x46	DMA2
0x47	DMA3
0x48	PCI1
0x49	PCI2 / Rapid IO Error
0x4a	Rapid IO Bell
0x4b	Rapid IO Transmit

Table 2-2 PIC Vectors (continued)

Vector	Source
0x4c	Rapid IO Receive
0x4d	TSEC1 Transmit
0x4e	TSEC1 Receive
0x52	TSEC1 Error
0x53	TSEC2 Transmit
0x54	TSEC2 Receive
0x58	TSEC2 Error
0x59	FEC
0x5a	Both On-chip UARTs
0x5b	IIC1
0x5c	Performance Monitor
0x5d	SEC2
0x5e	CPM
0x62	PCI INTA
0x63	PCI INTB
0x65	PCI INTD
0x66	PCI INTC

External IRQ vectors are mapped starting at vector 0x60 in the order shown in the following table.

Table 2-3 External IRQ Vectors

Vector	Source
0x60	IRQ 0
0x61	IRQ 1
0x62	IRQ 2 / PCI INTA
0x63	IRQ 3 / PCI INTB
0x64	IRQ 4
0x65	IRQ 5 / PCI INTD
0x66	IRQ6 / PCI INTC
0x67	IRQ 7
0x68	IRQ 8
0x69	IRQ 9
0x6a	IRQ 10
0x6b	IRQ 11

SPE Floating-point Support

This section contains information about the support for the scalar, single-precision SPE floating-point instructions.

Support for SPE instructions is contained in a header file and two example programs making use of this header file. The header file makes it possible to use the various “efs” instructions of the e500 core processor.

Each instruction is encapsulated in a function, the compiler will inline the function at each call site, resulting in minimal instructions in the final executable. If debugging is enabled, this inlining will not occur so the resulting binary will contain multiple copies of these functions, resulting in a larger binary file.

Files

The following are the files in your distribution related to SPE support (all files are relative to the root of your MWOS directory (e.g. `C:\MWOS`):

`OS9000\E500\DEFS\spe.h` This is the header to include to enable support for the SPE functions described below. Use the `-v` option to `xcc.exe` to add the `$(MWOS)\OS9000\E500\DEFS` directory to your #include search path and then include this header with a line like: `#include <spe.h>`

`OS9000\E500\SRC\SPE\spe_speed.c`
This is an example program that measures the performance of SPE code versus non-SPE code. It can be compiled two ways: with `DO_SPE` defined or without `DO_SPE` defined. If `DO_SPE` is defined it will take advantage of the SPE to perform the single-precision math. If `DO_SPE` is not defined, normal single-precision floating-point instructions will be used.

`OS9000\E500\SRC\SPE\spe_test.c`
: This is an example of how to use the various functions in `spe.h`.

The remainder of this section describes the various definitions in spe.h.

Data Types

The following data types are defined by spe.h:

`efp`	Embedded floating-point. This type is used to hold a 32-bit floating-point value in an integer general-purpose register.
`boolean`	Boolean result. The various compare and test instructions return the `boolean` data type.
`spefscr`	Signal Processing Engine Floating-point Status and Control Register. This data type is used for variables that are to hold a value of the SPEFSCR. The SPEFSCR is swapped at task switch time like any other general-purpose register.

Macros

`SPEFSCR_SOVH`	mask for the summary integer overflow bit.
`SPEFSCR_OVH`	mask for the integer overflow high bit.
`SPEFSCR_FGH`	mask for the embedded floating-point guard high bit.
`SPEFSCR_FXH`	mask for the embedded floating-point sticky high bit.
`SPEFSCR_FINVH`	mask for the embedded floating-point invalid operation error high bit.

SPEFSCR_FDBZH	mask for the embedded floating-point divide by zero high bit.
SPEFSCR_FUNFH	mask for the embedded floating-point underflow high bit.
SPEFSCR_FOVFH	mask for the embedded floating-point overflow high bit.
SPEFSCR_FINXS	mask for the embedded floating-point inexact sticky bit.
SPEFSCR_FINVS	mask for the embedded floating-point invalid operation sticky bit.
SPEFSCR_FDBZS	mask for the embedded floating-point divide by zero sticky bit.
SPEFSCR_FUNFS	mask for the embedded floating-point underflow sticky bit.
SPEFSCR_FOVFS	mask for the embedded floating-point overflow sticky bit.
SPEFSCR_MODE	mask for the embedded floating-point mode bit.
SPEFSCR_SOV	mask for the integer summary overflow bit.
SPEFSCR_OV	mask for the integer overflow bit.
SPEFSCR_FG	mask for the embedded floating-point guard bit.
SPEFSCR_FX	mask for the embedded floating-point sticky bit.
SPEFSCR_FINV	mask for the embedded floating-point invalid operation bit.
SPEFSCR_FDBZ	mask for the embedded floating-point divide by zero error bit.
SPEFSCR_FUNF	mask for the embedded floating-point underflow bit.
SPEFSCR_FOVF	mask for the embedded floating-point overflow bit.

`SPEFSCR_FINXE`	mask for the embedded floating-point inexact enable bit.
`SPEFSCR_FINVE`	mask for the embedded floating-point invalid operation/input error exception enable bit.
`SPEFSCR_FDBZE`	mask for the embedded floating-point divide by zero exception enable bit.
`SPEFSCR_FUNFE`	mask for the embedded floating-point underflow exception enable bit.
`SPEFSCR_FOVFE`	mask for the embedded floating-point overflow exception enable bit.
`SPEFSCR_FRMC1`	mask for the embedded floating-point rounding mode control bit 1.
`SPEFSCR_FRMC0`	mask for the embedded floating-point rounding mode control bit 0.
`SPEFSCR_E_MASK`	mask for all the exception enable and rounding mode bits.
`efs_is_finv()`	returns TRUE if `SPEFSCR_FINV` is currently set in SPEFSCR
`efs_is_fofv()`	returns TRUE if `SPEFSCR_FOFV` is currently set in SPEFSCR
`efs_is_funv()`	returns TRUE if `SPEFSCR_FUNV` is currently set in SPEFSCR
`efs_is_fdbz()`	returns TRUE if `SPEFSCR_FDBZ` is currently set in SPEFSCR
`efs_is_fdbzs()`	returns TRUE if `SPEFSCR_FDBZS` is currently set in SPEFSCR
`efs_is_finxs()`	returns TRUE if `SPEFSCR_FINXS` is currently set in SPEFSCR
`efs_is_fg()`	returns TRUE if `SPEFSCR_FG` is currently set in SPEFSCR
`efs_is_fx()`	returns TRUE if `SPEFSCR_FX` is currently set in SPEFSCR

`float_to_efp(float_x)` converts the ANSI `float` value to the `efp` data type. The macro's result is a value of type `efp`.

`efp_from_const(efp_x, value)`
initializes a variable of type `efp`, `efp_x`, with the ANSI `float` bits of a constant, `value`. This macro instantiates as a statement, not an expression.

`efp_from_float(efp_x, float_x)`
initializes a variable of type `efp`, `efp_x`, with the ANSI `float` bits of a 32-bit floating-point variable, `float_x`. The value of this macro is the value assigned to `efp_x`.

`float_from_efp(float_x, efp_x)`
initializes a variable of type ANSI `float` with the `efp` bits of either an `efp` constant or an `efp` variable. The value of this macro is value of `efp_x`.

Functions

`efp efs_abs(efp rA)` returns the floating-point absolute value of `rA`

`efp efs_add(efp rA, efp rB)`
returns the sum of `rA` and `rB`

`efp efs_cfsf(int32 rB)` returns the result of converting `rB` from the signed fraction format to the 32-bit single-precision format.

`efp efs_cfsi(int32 rB)` returns the result of converting `rB` from signed 32-bit integer format to the 32-bit single-precision format.

`efp efs_cfuf(u_int32 rB)`
: returns the result of converting `rB` from unsigned fractional format to the 32-bit single-precision format.

`efp efs_cfui(u_int32 rB)`
: returns the result of converting `rB` from unsigned 32-bit integer format to the 32-bit single-precision format.

`boolean efs_cmpeq(efp rA, efp rB)`
: returns TRUE if `rA` and `rB` are equal, FALSE otherwise.

`boolean efs_cmpgt(efp rA, efp rB)`
: return TRUE if `rA` is greater than `rB`, FALSE otherwise.

`boolean efs_cmplt(efp rA, efp rB)`
: returns TRUE if `rA` is less than `rB`, FALSE otherwise.

`int32 efs_ctsf(int32 rB)`
: returns the result of converting `rB` from 32-bit single-precision format to signed fraction format.

`int32 efs_ctsi(efp rB)`
: returns the result of converting `rB` from 32-bit single-precision format to signed integer format.

`int32 efs_ctsiz(efp rB)`
: returns the result of converting `rB` from 32-bit single-precision format to signed fraction format with rounding toward zero (truncation).

`u_int32 efs_ctuf(efp rB)`
: returns the result of converting `rB` from 32-bit single-precision format to unsigned fraction format.

`u_int32 efs_ctui(efp rB)`
: returns the result of converting `rB` from 32-bit single-precision format to unsigned integer format.

`u_int32 efs_ctuiz(efp rB)`
: returns the result of converting `rB` from 32-bit single-precision format to unsigned integer format with rounding toward zero (truncation).

`efp efs_div(efp rA, efp rB)`
: returns the value of `rA` divided by `rB`.

`efp efs_mul(efp rA, efp rB)`
: returns the product of `rA` and `rB`.

`efp efs_nabs(efp rA)`
: returns the opposite of the floating-point absolute value of `rA`

`efp efs_neg(efp rA)`
: returns the opposite of `rA`

`efp efs_sub(efp rA, efp rB)`
: return the difference between `rA` and `rB` (`rA - rB`)

`boolean efs_tsteq(efp rA, efp rB)`
: tests for `rA` being equal to `rB`, without regard for illegal value

`boolean efs_tstgt(efp rA, efp rB)`
: tests for `rA` being greater than `rB`, without regard for illegal value

`boolean efs_tstlt(efp rA, efp rB)`
: tests for `rA` being less than `rB`, without regard for illegal value

`spefscr efs_get_spefscr()`
: returns the current value of SPEFSCR

`void efs_set_spefscr(spefscr rA)`
: sets the current value of SPEFSCR to `rA`

`void efs_clear_spefscr()`

clears all exception enable bits in SPEFSCR and returns the rounding mode to “round to nearest” (b00).

Appendix A: Board Specific Modules

This appendix contains lists of high and low-level modules. The following sections are included:

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

Low-Level System Modules

The following low-level system modules are tailored specifically for the PowerPC TQM85xx platform. The functionality of many of these modules can be altered through changes to the configuration data module (`cnfgdata`). These modules are located in the following directory:

`MWOS/OS9000/E500/PORTS/TQM85XX/CMDS/BOOTOBJS/ROM`

`cnfgdata`	provides low-level configuration data including configuration of a serial console.
`confgfunc`	retrieves configuration parameters from the `cnfgdata` module.
`conscnfg`	retrieves the name of the low-level console driver from the `cnfgdata` module.
`initext`	configures the on-chip PCI bus controller.
`io16550`	provides console services for the 16550 DUART on the MPC85xx.
`lltsec`	provides network driver services for the TSEC Ethernet ports.
`portmenu`	retrieves a list of configured booter names from the ROM `cnfgdata` module.
`romcore`	initializes the board after system reset.
`romstart`	is an initial vector table.
`rpciv`	is a PCI bus device viewer “booter” utility.
`tbtimer`	provides polling timer services using the `tblo` and `tbhi` registers in the MPC85xx processors.
`usedebug`	is a debugger configuration module.

High-Level System Modules

The following OS-9 system modules are tailored specifically for your TQM85xx platform. Unless otherwise specified, each module can be found in a file of the same name in the following directory. Some modules are found in sub-directories:

`MWOS/OS9000/E500/PORTS/TQM85XX/CMDS/BOOTOBJS`

`ds1337u_mpc85xx`	provides Real-Time Clock (RTC) OS-9 driver support for the on-board RTC chip.
`idle_doze`	provides support for putting the processor into DOZE low-power mode when the operating system is idle.
`idle_nap`	provides support for putting the processor into NAP low-power mode when the operating system is idle. NAP mode will disable Ethernet networking.
`openpicirq`	maps the PIC IRQ vectors to OS-9 IRQ vectors.
`rbftl`	provides Flash device driver support for the RBF file manager. This module will only be present if the TrueFFS add-on is installed.
`picsub`	provides interrupt enable and disable routines to handle platform specific interrupt controller issues for device drivers. This module is called by all drivers and should be included in your bootfile.
`sc16550`	provides support for the on-chip 16550 serial ports. This driver is used to drive the console over the COM1 and COM2 ports.
`tkdec`	provides the system ticker based on the PowerPC decrementer.
`DESC/RAM/r0`	provides device configuration information for the RAM disk.

`DESC/SC16550/t1`	provides device configuration information for the COM1 port.
`DESC/SC16550/term_t1`	provides device configuration information for the COM1 port. The module name is `term`.
`DESC/SC16550/t2`	provides device configuration information for the COM2 port.
`DESC/SC16550/term_t2`	provides device configuration information for the COM2 port. The module name is `term`.
`DESC/SCLLIO/term`	provides device configuration information for using the low-level console for high-level serial I/O. The module name is `term`.
`DESC/SCLLIO/vcons`	provides device configuration information for using the low-level console for high-level serial I/O. The module name is `vcons`, virtual console.
`DESC/pipe`	provides device configuration information for the FIFO file manager (`pipeman`).
`INITS/nodisk`	provides system initialation information that indicates no default RBF device is to be used. The module name is `init`.
`SPF/inetdb`	is a data module containing Internet configurations.
`SPF/inetdb2`	is a data module containing additional Internet configurations.
`SPF/rpcdb`	is an NFS/RPC database module.
`SPF/spgb0`	is an Ethernet descriptor for the Intel Ethernet Pro 1000 PCI card.
`SPF/sppr0`	is an Ethernet descriptor for the Intel Ethernet Pro 10/100 PCI card.
`SPF/sppr1`	is an Ethernet descriptor for a second Intel Ethernet Pro 10/100 PCI card.

`SPF/sppro100`	is an Ethernet driver for the Intel Ethernet Pro 10/100 PCI card.
`SPF/sppro1000`	is an Ethernet driver for the Intel Ethernet Pro 1000 PCI card.
`SPF/spts0`	is an Ethernet descriptor for the TSEC1 10/100/1000 Ethernet port.
`SPF/spts1`	is an Ethernet descriptor for the TSEC2 10/100/1000 Ethernet port.
`SPF/spts2`	is an Ethernet descriptor for the FEC 10/100 Ethernet port.
`SPF/sptsec`	is an Ethernet driver for the TSEC and FEC Ethernet ports.

Port-specific Utilities

The following OS-9 programs are tailored specifically for your TQM85xx platform. Each module can be found in a file of the same name in the following directory:

`MWOS/OS9000/E500/PORTS/TQM85XX/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 AMD flash device.
`setpci`	Allows changes to the PCI configuration of devices.
`testpci`	Runs several tests for the PCI library functions.

Common System Modules List

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

`MWOS/OS9000/PPC/CMDS/BOOTOBJS/ROM`

Table 2-4 Common System Modules List

Module	Description
`bootsys`	provides booter services.
`console`	provides high-level I/O hooks into the low-level console serial driver.
`dbgentry`	provides hooks to the low-level debugger server.
`dbgserv`	is a debugger server module.
`excption`	is a interrupt and exception service module.
`fdc765`	provides PC style floppy support.
`fdman`	is a target-independent booter support module providing general booting services for RBF file systems.
`flboot`	is a SCSI floptical drive disk booter.
`flshcach`	provides the cache flushing routine.
`fsboot`	is a SCSI TEAC floppy disk drive booter.
`hlproto`	allows user-state debugging over the low-level system-state networking layer.

Table 2-4 Common System Modules List (continued)

Module	Description
`hsboot`	is a SCSI hard disk driver booter.
`ide`	provides target-specific standard IDE support, including PCMCIA ATA PC cards.
`iovcons`	is a hardware independent virtual console driver that provides a telnetd-like interface to the low-level system console.
`llbootp`	is a target-independent BOOTP protocol booter module.
`llip`	is a target-independent internet protocol module.
`llkermit`	is a kermit booter (serial down loader).
`llslip`	is a target-independent serial line internet protocol module. This modules uses the auxiliary communications port driver to perform serial I/O
`lltcp`	is a target-independent transmission control protocol module.
`lludp`	is a target-independent user datagram protocol modules.
`notify`	coordinates use of low-level I/O drivers in system and user-state debugging.
`override`	enables overriding of the autobooter. If the space bar is pressed within three seconds after booting the target, a boot menu is displayed. Otherwise, booting proceeds with the first autobooter.

Table 2-4 Common System Modules List (continued)

Module	Description
`parser`	parses key fields from the `cnfgdata` module and the user parameter fields.
`pcman`	is a target-independent booter support module providing general booting services for PCF file systems (PC FAT file systems).
`protoman`	is a target-independent protocol module manager. This module provides the initial communication entry points into the protocol module stack.
`restart`	restarts boot process.
`romboot`	locates the OS-9 bootfile in ROM, FLASH, NVRAM.
`rombreak`	enables break option from the boot menu.
`rombug`	is a debugger client module.
`scsiman`	is a target-independent booter support module that provides general SCSI command protocol services
`sndp`	is a target-independent system-state network debugging protocol module. This module acts as a debugging client on the target, invoking the services of dbgserv to perform debug tasks.
`srecord`	booter that receives a Motorola S-record format file from the communications port and loads it into memory.
`swtimer`	is a software timer.

Table 2-4 Common System Modules List (continued)

Module	Description
`tsboot`	is a SCSI TEAC tape drive booter.
`type41`	is a primary partition type.
`uncompress`	is the module that handles the decompressing of bootfiles.
`vcons`	is the virtual console driver.
`vsboot`	is a SCSI archive viper tape drive booter.