RadiSys.

OS-9 for ARM CL7212 Board Guide

Version 3.2

www.radisys.com
World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
December 2001

Copyright and publication information

Disclaimer

Reproduction notice

Table of Contents

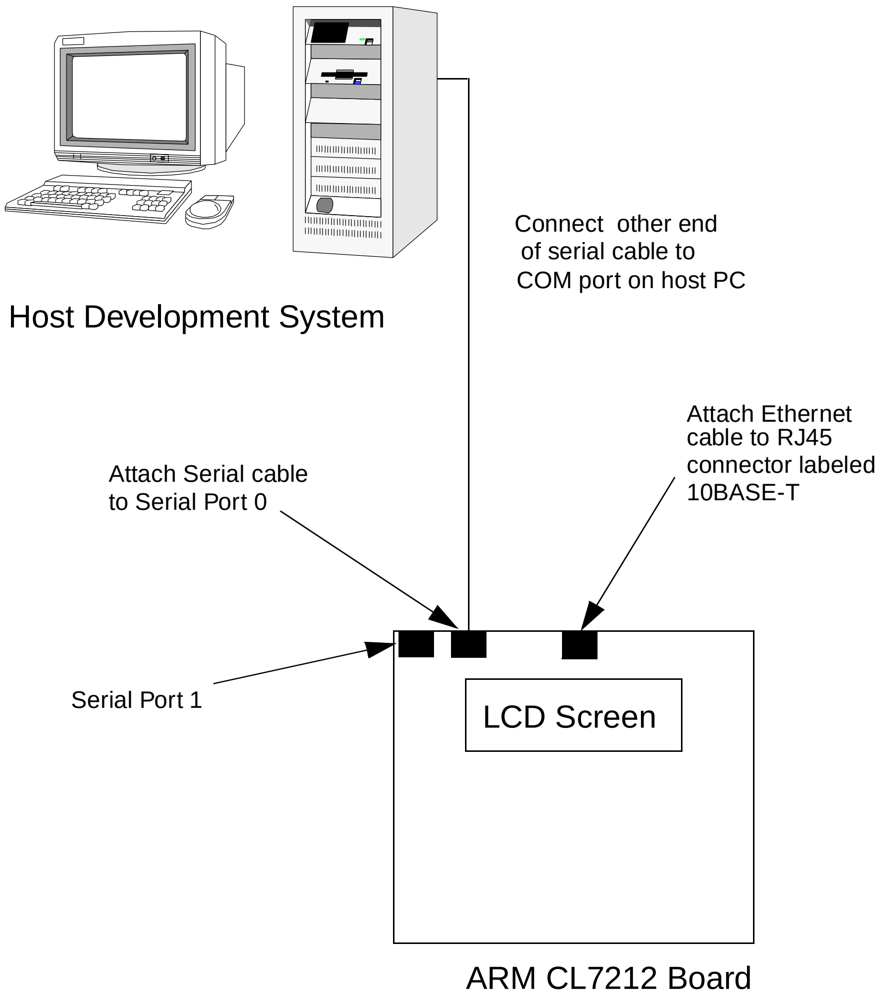
Chapter 1: Installing and Configuring OS-9

This chapter describes installing and configuring OS-9 on the ARM CL7212 board. It includes the following sections:

Development Environment Overview

Figure 1-1 shows a typical development environment for the ARM CL7212 board. The components shown include the minimum required to enable OS-9 to run on the ARM CL7212 board.

Figure 1-1 ARM CL7212 Development Environment


Host Development System
Connect other end of serial cable to COM port on host PC
Attach Serial cable to Serial Port 0
Attach Ethernet cable to RJ45 connector labeled 10BASE-T
Serial Port 1
LCD Screen
ARM CL7212 Board

Requirements and Compatibility

Host Hardware Requirements (PC Compatible)

Your host PC must have the following minimum hardware characteristics:

- the recommended amount of RAM for the host operating system
- an Ethernet network card

Host Software Requirements (PC Compatible)

Your host PC must have the following software installed:

- Windows 95, 98, ME, 2000, or NT 4.0
- Enhanced OS-9 for ARM

Target Hardware Requirements

Your ARM CL7212 evaluation board requires the following hardware:

- a power supply
- an RS-232 null modem serial cable (for serial console)
- an Ethernet cable or a second RS-232 null modem serial cable (for down-loading programs to the board)

Software Compatibility

OS-9 for ARM/StrongARM is compatible with the following software:

- Microware OS-9 for ARM
- Microware Hawk Version
- Microware SoftStax

Connecting the Target to the Host

Connecting the ARM CL7212 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 the serial console in Hawk to verify the serial connection.

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

Attaching the Cables

Step 1. Attach an Ethernet cable to the RJ45 connector labeled “10BASE-T”.

Step 2. Connect a serial cable to the connector labeled “Serial Port 0”.

Step 3. Connect the other end of the serial cable to a COM port on the host PC.

Step 4. DO NOT attach the power cable unless the AC Adapter is not plugged into a power outlet. There is no on/off switch.

Note
If using Serial Port 1 on the target system, you must set your baud rate to 115,200 and select no hardware handshaking when configuring your serial port. You can use the default configuration for all other settings. You must also use a reversed serial cable.

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. Depending on the reference board's capabilities, the coreboot could be located on a FLASH part, a hard disk, or a floppy disk. 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 selected from the boot menu. The bootfile image normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

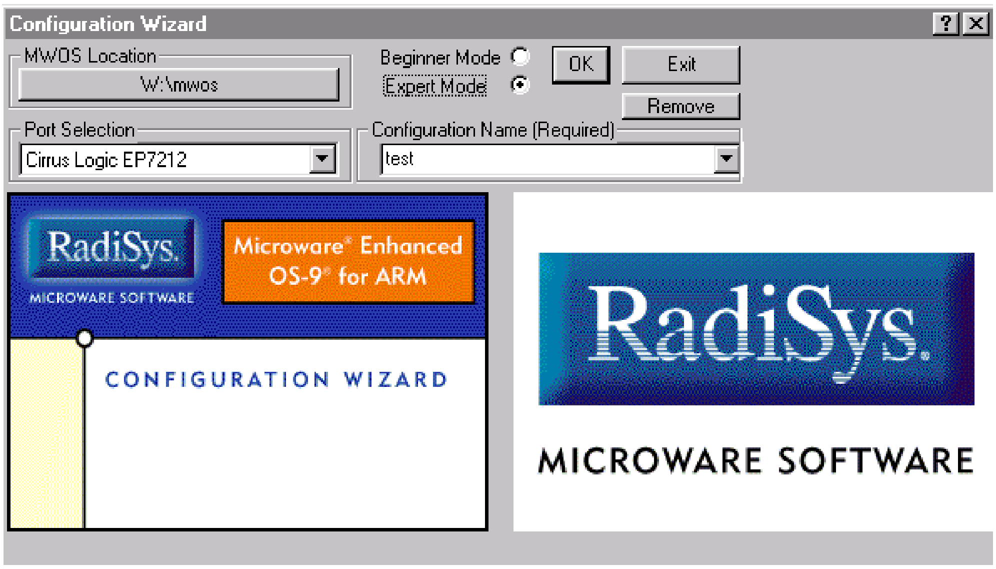
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on the host PC during the Enhanced OS-9 installation process.

Using the Configuration Wizard

This section describes using the Configuration Wizard to build the OS-9 ROM image. To open and use the Wizard, complete the following steps:

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

Figure 1-2 Configuration Wizard Opening Screen



MWOS Location: W:\mwos
Beginner Mode [✓ Expert Mode]
Port Selection: Cirrus Logic EP7212
Configuration Name (Required): test

Step 2. Select the path where the MWOS directory structure is located by clicking the **MWOS location** button.

Step 3. Select the EP7212 target board from the **Port Selection** pull-down menu.

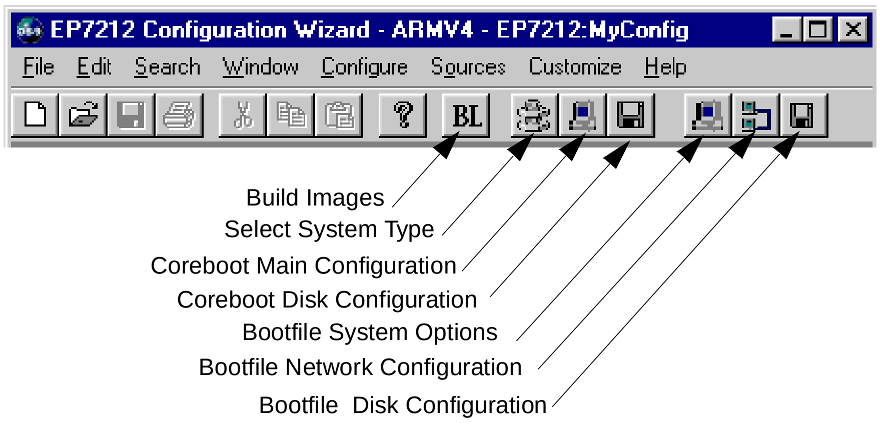
Note

The Configuration Wizard refers to the ARM CL7212 board as the EP7212 port.

Step 4. Select a name for your configuration in the **Configuration Name** field. Your settings are saved. This enables you to modify the ROM image incrementally, without having to reselect every option for each change.

Step 5. Select `Expert Mode` and click `OK`. The **Main Configuration** window is displayed. Advanced mode enables you to make more detailed and specific choices about what modules are included in your ROM image.

Figure 1-3 Main Configuration Window Toolbar


EP7212 Configuration Wizard - ARMV4 - EP7212:MyConfig
File
Edit
Search
Window
Configure
Sources
Customize
Help
BL
Build Images
Select System Type
Coreboot Main Configuration
Coreboot Disk Configuration
Bootfile System Options
Bootfile Network Configuration
Bootfile Disk Configuration

For More Information

The ***OS-9 Device Descriptor and Configuration Module Reference*** manual included on your CD describes each of the OS-9 modules.

Configuring Coreboot Options

To set up the coreboot image, you will set the options found by clicking on the Coreboot configuration buttons.

Step 1. Click the `Coreboot Main Configuration` button.

Step 2. Click on the **Debugger** tab. Make sure `Ethernet` is selected in the **Remote Debug Connection** area and `Remote` is selected in the **Select Debugger** area. Remote debugging is enabled so that system-state debugging can be performed in Hawk.

Step 3. Click on the `Ethernet` tab and enter the Ethernet address information in the address text boxes. For most situations you will need to fill out the following text boxes:

- **IP Address**
- **IP Broadcast**
- **Subnet Mask**
- **IP Gateway**
- **MAC Address**

If you are uncertain of the values for these text boxes, contact your system administrator.

Step 4. Click `OK` to close the window. The default settings for the other tabs do not require changing.

Configuring Bootfile Options

Most of the default options in the dialogs that control the configuration of the bootfile are correct. There are a few functions, such as Ethernet, that need additional information in order to be configured correctly. To configure your bootfile options, complete the following steps:

Step 1. To configure the Ethernet function, click on the `System Network Configuration` button.

Step 2. Click on the `Interface` tab.

Step 3. Click on `Ethernet Connection` in the **Select Interface** list.

Step 4. Make sure that **Specify an IP Address** is selected and the address information in the IP address text boxes is correct. They should have been copied from the coreboot Ethernet dialog box. If the information is not correct, correct it now.

Step 5. Select the `Ethernet` check box in the **Disable/Enable Interface** area.

Step 6. Make sure the name of the Ethernet controller chip is displayed in the combo box under the note about the MAC address. If it is not visible, the Ethernet modules will not be included in the build.

Step 7. Click on the `SoftStax Setup` tab, and select `Enable SoftStax`.

Step 8. Click `OK` to close the dialog box.

Step 9. Click on the **System Disk Configuration** button and verify that the default settings are acceptable to you.

Step 10. Leave the other default settings alone and click the `Build Images` button to display the **Master Builder** window.

Step 11. Select the following check boxes as they are appropriate to your setup:

- SoftStax (SPF) Support
- User State Debugging Modules
- If you are using a RAM disk, select `Disk Support`**.**
- If you are using a RAM disk, select `Disk Utilities`**.**

Step 12. Click `Coreboot + Bootfile` and click `Build`. This will build the ROM image that can be burned into flash memory. The name of the ROM image is `rom`.

Step 13. Click `Finish` and then select `File` -> `Save Settings` to save the configuration.

Step 14. Select `File` -> `Exit` to quit from the Configuration Wizard.

Transferring the ROM Image to the Target

The next step is to transfer the ROM image to the on-board flash memory on target board. Microware supplies a utility called `cl_download` that performs this transfer. It takes a binary file and transfers it across any COM port at any desired baud rate. By default, `cl_download` uses COM1 and a baud rate of 115200. The defaults can be changed via command line options. Here are some usage examples:

The following command uses COM1 AND 115200 baud:

```
$ cl_download rom
```

The following command uses COM3 AND 115200 baud:

```
$ cl_download rom 3
```

The following command uses COM3 AND 19200 baud:

```
$ cl_download rom 3 19200
```

On the ARM CL7212 reference board, Serial Port 0 will receive the download. The target board must be put into flash burn/download mode. The ROM image must be programmed into flash, and the board must be rebooted in its normal running mode. The following steps describe how to do this.

Download Steps

Step 1. Open a DOS shell and change to the following directory:
`<drive>:\mwos\OS9000\ARMV4\PORTS\EP7212\BOOTS\INSTALL\PORTBOOT`

Step 2. Remove power from the board

Step 3. Close the `JP2` jumper. The jumper JP2 is located near the center of the board by the processor.

Step 4. Apply power to the board.

Step 5. Run `cl_download` with the following command:

```
cl_download rom 119200
```

Step 6. Press the URESET button, then press the WAKUP button. The board should start receiving data.

If the board doesn't start receiving data, press the URESET button again for at least 1/2 second, then press the WAKUP button for at least 1/2 second. Once the file has been downloaded and burned into flash memory, the message "`Successfully programmed 'rom'`" appears on the LCD. Repeat steps 2-6 if the message does not appear.

Step 7. Remove power from the board and open jumper JP2.

Step 8. Start up a Terminal program such as HyperTerminal or the serial console in Hawk. Use a baud rate of 19200, 8 bits, 1 stop bit, and no flow control.

Step 9. Apply power to the board, press URESET followed by WAKEUP. The OS-9 bootstrap message appears in the terminal program's window.

When you see the shell prompt "$", the board is booted and running OS-9.

Chapter 2: Board Specific Reference

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

- **The Fastboot Enhancement**

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.)

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 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.

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;
}
```

Appendix A: Board Specific Modules

This chapter describes the modules specifically written for the ARM CL7212 board. It includes the following sections:

Low-Level System Modules

The following low-level system modules are tailored specifically for the ARM CL7212 board. They are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/EP7212/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
`io7212`	low-level based serial IO driver
`ll8900`	Low-level Ethernet ROM driver
`portmenu`	inits booters defined in the `cnfgdata`
`romcore`	bootstrap code
`tmr7212`	ROM timer services
`usedebug`	debugger configuration module

High-Level System Modules

The following OS-9 system modules are tailored specifically for the ARM CL7212 board.

Modules located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/EP7212/CMDS/BOOTOBJS`

sc7212	Serial driver that supports baud rates up to 115200. Default Baud Rate is 19,200. Descriptors `/term` and `/t1` are assigned to Port 0. Descriptors `/term2` and `/t2` are assigned to Port 1.
tkarm	System clock module

Modules located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/EP7212/CMDS/BOOTOBJS/SPF`

sp8900	Driver module for the Ethernet controller
spcs0	Descriptor for the Ethernet driver sp8900

Modules located in the following directory:
`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

vect110	Vector module for ARM
fpu	ARM FPU emulator

Common System Modules List

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

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

`bootsys`	provides booter registration services
`console`	provides console services
`dbgentry`	inits debugger entry point for system use
`dbgserv`	provides debugger services
`excption`	provides low-level exception services
`fdc765`	provides PC style floppy support
`flboot`	is a SCSI floptical drive disk booter
`flshcach`	provides low-level cache management services
`hlproto`	provides user level code access to protoman
`ide`	provides target-specific standard IDE support, including PCMCIA ATA PC cards
`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
`pcman`	provides a booter that reads MS-DOS file system
`protoman`	provides a protocol management module
`restart`	provides a booter that causes a soft reboot of 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
`scsiman`	is a target-independent booter support module that provides general SCSI command protocol services
`sndp`	provides low-level system debug protocol
`srecord`	provides a booter that accepts S-Records
`swtimer`	provides timer services via software loops
`tsboot`	is a SCSI TEAC tape drive booter
`type41`	is a primary partition type