OS-9® for Assabet/Neponset Board Guide

Version 4.7
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.
# Table of Contents

## Chapter 1: Installing and Configuring OS-9®

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements and Compatibility</td>
<td>10</td>
</tr>
<tr>
<td>Host Hardware Requirements (PC Compatible)</td>
<td>10</td>
</tr>
<tr>
<td>Host Software Requirements (PC Compatible)</td>
<td>11</td>
</tr>
<tr>
<td>Target Hardware Requirements</td>
<td>11</td>
</tr>
<tr>
<td>Java Hardware Requirements</td>
<td>11</td>
</tr>
<tr>
<td>Target Hardware Setup</td>
<td>12</td>
</tr>
<tr>
<td>Connecting the Target to the Host</td>
<td>13</td>
</tr>
<tr>
<td>Building the OS-9 ROM Image</td>
<td>15</td>
</tr>
<tr>
<td>Overview</td>
<td>15</td>
</tr>
<tr>
<td>Coreboot</td>
<td>15</td>
</tr>
<tr>
<td>Bootfile</td>
<td>16</td>
</tr>
<tr>
<td>Starting the Configuration Wizard</td>
<td>16</td>
</tr>
<tr>
<td>Booting from the On-Board Flash</td>
<td>18</td>
</tr>
<tr>
<td>Booting from an ATA Flash Card or Using bootp</td>
<td>18</td>
</tr>
<tr>
<td>Pinging the Target Board</td>
<td>26</td>
</tr>
<tr>
<td>Creating a Startup File</td>
<td>27</td>
</tr>
<tr>
<td>Example Startup File</td>
<td>28</td>
</tr>
<tr>
<td>Optional Procedures</td>
<td>30</td>
</tr>
<tr>
<td>Creating a new OS-9 Coreboot Image or ROM Image in Flash Memory</td>
<td>30</td>
</tr>
<tr>
<td>Making a Coreboot Image with an EPROM programmer or Intel’s JFlash Utility</td>
<td>31</td>
</tr>
<tr>
<td>Compressing the Bootfile Image Image</td>
<td>33</td>
</tr>
</tbody>
</table>

## Chapter 2: Board-Specific Reference

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Options</td>
<td>36</td>
</tr>
</tbody>
</table>
36 Booting from Flash
37 Booting from PCMCIA ATA Card
37 Booting from PCMCIA Ethernet Card
37 Booting over Serial Communications Port via Kermit
38 Restart Booter
38 Break Booter
40 The Fastboot Enhancement
40 Overview
41 Implementation Overview
41 B_QUICKVAL
41 B_OKRAM
42 B_OKROM
42 B_1STINIT
42 B_NOIRQMASK
43 B_NOPARITY
43 Implementation Details
43 Compile-time Configuration
44 Runtime Configuration
45 OS-9 Vector Mappings
54 Assabet GPIO Usage
56 GPIO Interrupt Polarity
57 Board-Specific Register Access Functions
59 Example
60 Port Specific Utilities

Appendix A: Board-Specific Modules

68 Low-Level System Modules
72 High-Level System Modules
72 CPU Support Modules
73 System Configuration Module
73 Power Management Support Modules
73 Interrupt Controller Support
74 Real Time Clock

OS-9 for the Assabet/Neponset Board Guide
Ticker
Abort Handler
Generic IO Support modules (File Managers)
Pipe Descriptor
RAM Disk Support
Descriptors for Use with RAM
Serial and Console Devices
Descriptors for Use with sc1100
Descriptors for use with sc1101
Descriptors for use with sc16550
Descriptors for Use with sclio
PCMCIA Support for IDE Type Devices
Descriptors for Use with rb1003
PCMCIA Support for Socket-LPE Ethernet Card (NE2000 Compatible)
Descriptors for Use with spne2000
PCMCIA Support for 3COM Ethernet Card
spe509_pcm Descriptors
SMC91C94 Ethernet Support
Descriptor for Use with sp91c94
Network Configuration Modules
UCB1300 Support Modules
Descriptors for Use with spucb1200
Maui Graphical Support Modules
Descriptors for Use with gx_sa1100
Descriptors for Use with sd_ucb1200
MAUI configuration modules
MAUI protocol modules

Appendix B: MAUI Driver Descriptions

Assabet Objects
MAUI objects
GX_SA1100 LCD Graphic Driver Specification
Board Ports
Device Capabilities
Display Resolution
Coding Methods
Viewport Complexity
Memory
Location
Build the Driver
Build the Descriptor
SD_UCB1200 Sound Driver Specification
Device Capabilities
Gain Capabilities Array
Sample Rates
Number of Channels
Encoding and Decoding Formats
SPUCB1200 driver for the UCB1200 Codec
Capabilities
Descriptors
UCB
Audio
Touch Screen
GPIO
Telecom
Supporting Modules
MP_UCB1200 MAUI Touch screen Protocol Module
Overview
Data Format
Data Filter
Raw Mode
cdb.touch
Compile Time Options
Calibration Application
Assumptions/Dependencies
Command Line Options
105 Coordination with Protocol Module
105 Compiling
Chapter 1: Installing and Configuring OS-9®

This chapter describes installing and configuring OS-9® on the Intel SA-1110 Microprocessor Reference Platform (Assabet) and the Intel SA-1111 daughter card (Neponset). It includes the following sections:

- Requirements and Compatibility
- Target Hardware Setup
- Connecting the Target to the Host
- Building the OS-9 ROM Image
- Creating a Startup File
- Optional Procedures

Note
This description does not require that the Neponset daughter card be part of your development system. Neponset-specific modules and descriptors are noted.
Requirements and Compatibility

**Note**
Before you begin, install the *Microwave OS-9 for StrongARM* CD-ROM on your host PC.

**Host Hardware Requirements (PC Compatible)**

Your host PC should have the following:

- a minimum of 200 MB of free disk space (an additional 235 MB of free disk space is required to run PersonalJava™ Solution for OS-9)
- an Ethernet network card
- a PCMCIA card reader/writer
- at least 16 MB of RAM

**Note**
If you are a PersonalJava™ Solution for OS-9 licensee and you plan to use the Java JCC to pre-load your Java classes, you may need as much as 64 MB of RAM. Refer to the document *Using JavaCodeCompact* for a complete discussion of using the JCC.
Host Software Requirements (PC Compatible)

Your host PC should have the following:
- Windows 95, 98, ME, 2000, or NT 4.0
- A terminal emulation program (such as Hyperterminal)

Target Hardware Requirements

For More Information

Also refer to these manuals for information on hardware preparation and installation, operating instructions, and functional descriptions prior to installing and configuring OS-9.

Intel documentation is provided online in PDF format at www.intel.com.

Java Hardware Requirements

PersonalJava™ Solution for OS-9 requires the following features:
- 16MB of RAM
- 4MB of Flash (Boot)
- LCD Display
Target Hardware Setup

Microware OS-9 for StrongARM provides the necessary software to run the Assabet development board, with or without the Neponset daughter card. There are no OS-9-specific hardware considerations.

WARNING
If the Neponset daughter card is attached to the Assabet development board, do not insert a PCMCIA card into the Assabet's PCMCIA port. Use the Neponset's PCMCIA port instead.

Note
When the Neponset daughter card is used, the outer PCMCIA slot is slot #0 and the inner PCMCIA slot is slot #1.

For More Information

Intel documentation is provided online in PDF format at www.intel.com.
Connecting the Target to the Host

Connect an RS-232 null modem cable from the Target board to the serial port of a Windows host system.

Step 1. Connect the serial cable to the J10 connector (or the DB9 connector that connects to J10) on the Target board. The J10 connector is the SA1110 serial port 1 (SP1).

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

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

Step 4. Once Hyperterminal is open, enter a name for your Hyperterminal session.

Step 5. Select an icon for the new Hyperterminal session. A new icon is created with the name of your session. Click OK.

Step 6. In the Phone Number dialog, go to the Connect Using box and select the communications port to be used to connect to the reference board. The port selected is the same port that you connected to the serial cable from the reference board. Click OK.

Step 7. In the Port Settings tab, enter the following settings:

- Bits per second = 19200
- Data Bits = 8
- Parity = None
- Stop bits = 1
- Flow control = XOn/XOff
Step 8. Click **OK**. A connection should be established.

**Note**

If the word connected does not appear in the lower left corner of the window, click **Call -> Connect** to establish the connection.
Building the OS-9 ROM Image

Overview

The OS-9 ROM Image is a set of files and modules that collectively make up the OS-9 operating system. The specific ROM Image contents can vary from system to system depending on hardware capabilities and user requirements.

To simplify the process of loading and testing OS-9, the ROM Image is generally divided into two parts—the low-level image, called coreboot; and the high-level image, called bootfile.

Coreboot

The coreboot image is generally responsible for initializing hardware devices and locating the high-level (or bootfile) image as specified by its configuration. For example from a Flash part, a harddisk, or Ethernet. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, applications). The image is loaded into memory based on the device you select from the boot menu. The bootfile image normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the 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** -> **Programs** -> **RadiSys** -> **Microware OS-9 for <product>** -> **Configuration Wizard**. You should see the following opening screen:

Figure 1-2 Configuration Wizard Opening Screen

![Configuration Wizard Opening Screen](image)

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

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

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

**Figure 1-3**  Configuration Wizard Main Window

![Configuration Wizard Main Window](image-url)
Booting from the On-Board Flash

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

For More Information
Refer to the Creating a new OS-9 Coreboot Image or ROM Image in Flash Memory section for this procedure.

Note
When configuring your system to boot from the on-board Flash, it is recommended that you obtain and use a separate Flash part for this procedure. The original Flash part should be removed and saved.

Booting from an ATA Flash Card or Using bootp

This section describes how to use the Configuration Wizard to create a bootfile suitable for loading into memory from a PCMCIA ATA card or over the network using a bootp server (not provided).

Step 1. Start the Configuration Wizard, bringing it to the main configuration window as outlined previously.

Step 2. 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 3. 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
For More Information
To learn more about IPv4 and IPv6 functionalities, refer to the *Using LAN* manual, included with this product CD.

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

Step 4. Once you have made your settings in the **Network Configuration** dialog, click **OK**.

Step 5. Select **Configure -> Bootfile -> Disk Configuration**.

Step 6. On the **RAM Disk** tab, make sure that both check boxes are checked. Select an appropriate size from the pulldown for a RAM disk. (512K should be sufficient.)

Step 7. On the **IDE Configuration** tab, select the **Enable IDE disk** check box. Make sure that the Socket #0 radio button is selected.

Step 8. From the **Init Options** tab complete the following tasks:
1. Select MShell under **Initial Module Name**.
2. Select `/dd` under **Initial Device Name**. Then select **User**.
3. Edit the Parameter List field and delete the following portions of text:
   ```
   ipstart;
   inetd <>>>/nil &;
   > rtsol enet0
   ```
   This text should only be present if networking was enabled.

Step 9. Click **OK**.
Step 10. Select **Configure -> Build Image**.
Step 11. Under **Build Type/Options** select the **Bootfile Only Image** radio button.
Step 12. Make sure that the following radio buttons are checked:
- ROM Utility Set
- Disk Support
- Disk Utilities

Step 13. If networking is desired, make sure that the following radio buttons are checked:
- SoftStax® (SPF) Support
- User State Debugging Modules

Step 14. Click on the **Build** button.

When the build is complete the bootfile can be found in the following location:

```
%MWOS%\OS9000\ARMV4\PORTS\ASSABET\BOOTS\INSTALL\PORTBOOT\os9kboot
```

If you are booting using a bootp server skip to Step 18.

Step 15. If the image is to be placed on a PCMCIA ATA card complete the following tasks:

1. Insert the PCMCIA ATA card into the PCMCIA slot of your host computer.
2. Click Save As to save the file os9kboot to the root directory of the PCMCIA ATA card.
3. Turn off the power to the target board.
4. Remove the PCMCIA ATA card from the computer and insert it into the socket on the target board.
5. Apply power to the board.
6. When prompted to “Press the spacebar for a booter menu”, press the spacebar. If you miss it, simply power-cycle the target board.
7. Enter `ide0` and return. You should see the following screen:

```
Press the spacebar for a booter menu
BOOTING PROCEDURES AVAILABLE -------- <INPUT>
Boot embedded OS-9 in-place ---------- <bo>
Copy embedded OS-9 to RAM and boot ---- <lr>
Boot from PCMCIA-1 IDE -------------- <ide1>
Boot from PCMCIA-0 IDE ------------- <ide0>
```
Boot over Ethernet (NE2000) ----------- <eb>
Load bootfile via kermit Download ----- <ker>
Enter system debugger ----------------- <break>
Restart the System ----------------- <q>

Select a boot method from the above menu: ide0
Wait for IDE drive ready......ready.
IDE Model : Hitachi CV 6.1.2
Number Heads : 0x0004
Total Cylinders : 0x00f6
Sectors Per Track : 0x0020

Checking Partitions : 0
Fat Type : 0x12
File Name : OS9KBOOT
File Size : 0x002b5844
Start Cluster : 0x0000053d
Reading Bootfile....
Boot Address : 0xc002c860
Boot Size : 0x002b5844

OS-9 kernel was found.
A valid OS-9000 bootfile was found.

$  

Step 16. If the image is to be loaded from a bootp server (not provided), complete the following tasks:

1. Set up the bootp server such that it will serve the created image in response to a request from the given Ethernet card (which is identified by the Ethernet or MAC address -- this can be found by going through the following steps once before setting up the server).

2. Copy the os9kboot image from the following location to the location required by your server.
   \%MWOS\%OS9000\ARMV4\PORTS\ASSABET\BOOTS\INSTALL\PORTBOOT\os9kboot

3. Turn off the power to the target board.

4. Insert the PCMCIA networking card into the slot on the board; connect it to the Ethernet.

5. Apply power to the board.

6. When prompted to "Press the spacebar for a booter menu", press the spacebar. If you miss it, simply power-cycle the target board.

7. Enter eb and return.
   You can also specify the bootfile name using eb bootfile=os9kboot:
1 Installing and Configuring OS-9®

Note
The first line of output from the booter shows the Ethernet address of the given card.

Press the spacebar for a booter menu

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

Boot embedded OS-9 in-place ------- <bo>
Copy embedded OS-9 to RAM and boot ---- <lr>
Boot from PCMCIA-1 IDE --------------- <ide1>
Boot from PCMCIA-0 IDE --------------- <ide0>
Boot over Ethernet (NE2000) --------- <eb>
Load bootfile via kermit Download ---- <ker>
Enter system debugger ---------------- <break>
Restart the System ------------------ <q>

Select a boot method from the above menu: eb bootfile=os9kboot

bootp: 00:c0:1b:00:b5:d0 broadcasting for server...try 1/8
bootp: Got Response from server: 172.16.3.197
bootp: My IP Address will be: 172.16.3.197
bootp: My Bootfile is: /h0/TFTPBOOT/os9kboot
bootp: My bootfile size is: 0001629 (512-byte) blocks
bootp: My subnet mask is: 255.255.0.0
bootp: <<no timeoffset tag>>
tftp: Starting tftp transfer...
tftp: received file with 002c516c bytes

Bootfile received from server 172.16.1.6
Now searching for an OS-9000 Kernel...
A valid OS-9 bootfile was found.
$

If the bootfile was loaded using the bootp server, proceed directly to step 18. Otherwise, proceed to step 17.

Step 17. In order to use networking after booting off of the PCMCIA ATA Flash card, do the following:

- Deinitialize the PCMCIA slot:
  
  $ pcmcia -d

- Remove the PCMCIA Flash card from the slot. Slide the PCMCIA networking card into the slot.
• Initialize the PCMCIA slot:
  
  ```bash
  $ pcmcia -i
  PCMCIA Ethernet card found in socket #0
  ```

---

**Note**

It is not possible to swap the networking card out for the Flash card.

---

**WARNING**

Damage may occur to the PCMCIA card if it is inserted or removed improperly.

---

Step 18. Networking can be started by executing the following steps. These steps could have been executed automatically at startup for the bootp case.

  ```bash
  $ ipstart
  $ inetd >>>/nil &
  ```

Step 19. Test the Ethernet connections by pinging the target board. If the ping operation fails, evaluate the following scenarios:

  • Is the board connected to a live Ethernet port?
  • Is the Ethernet cable defective?
  • Are the network settings for the target board correct?
Pinging the Target Board

Windows 95, Windows 98, and Windows NT include a Ping command that can be used to test the Ethernet connection for the Target board.

Step 1. Go to the DOS prompt.

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

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

If the ping was successful, you will see the following response:
`Reply from <IP Address>: bytes=xx time=xms TTL=xx`

If the ping was unsuccessful, you will see the following response:
`Request timed out`.

---

**Note**

Windows 95, 98, and NT do not support IPv6.
Creating a Startup File

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

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

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

MWOS/OS9000/SRC/SYS

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

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

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

Step 4. Since the files are still in DOS format, you will be required to convert them into the OS-9 format with the cudo utility. The following command is an example:

cudo -cdo password

This will convert the password file from DOS to OS-9 format.
Installing and Configuring OS-9®

For More Information

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

Step 5.

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

Example Startup File

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

```
-tnxnp
tmode -w=1 nopause
*  
*OS-9 - Version 4.1
*Copyright 2002 by Microware Systems Corporation
*The commands in this file are highly system dependent and
*should be modified by the user.
*  
*setime </term              ;* start system clock
setime -s                   ;* start system clock
link mshell csl             ;* make "mshell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 term  ;* initialize devices
* load utils                ;* make some utilities stay in memory
* tsmon /term /t1 &         ;* start other terminals
list sys/motd
setenv TERM vt100
.tmode -w=1 pause
mshell<>>>/term -l &
```
For More Information
Refer to the Making a Startup File section in Chapter 9 of the *Using OS-9* manual for more information on startup files.
Optional Procedures

Creating a new OS-9 Coreboot Image or ROM Image in Flash Memory

If you want to boot and run OS-9 from the on-board Flash part, or if you want to use ROM Ethernet services such as system state debugging, you must create a new coreboot file and possibly a new bootfile.

The coreboot image that was shipped with the target board does not allow you to perform system state debugging because the IP address in the Flash ROM is set to 0.0.0.0. You can create a new image for your Flash part by using an EPROM programmer or by using Intel's Jflash utility.

Note
Recreating the Coreboot Image is required only when system state debugging is desired.
Making a Coreboot Image with an EPROM programmer or Intel’s JFlash Utility

This section describes creating a new Flash image. When you are done creating your image you can either use your EPROM programmer to load the image or use Intel's JFlash utility.

For More Information

Please see your EPROM programmer's instructions for a description of how to load the image via your programmer. See Intel's website for information about the JFlash utility.

http://developer.intel.com

Note

The shipping OS-9 systems have been tested with the following two Flash parts: FlashFile28F160S3's and StrataFlash E28F128J3.

Larger sizes can be chosen from the Configuration Wizard by selecting Select System Type and selecting the Setting based on pull down menu.

Note

When using Intel's JFlash utility, make sure you program the entire Flash part (select no when asked by the utility if you wish to save time by programming only part of the Flash).
You can also create and load a new ROM image into EPROM.

To make a coreboot image with an EPROM programmer, complete the following step:

Step 1. From the Windows desktop, select **Start -> Programs -> RadiSys -> Microware OS-9 for StrongARM <ver> -> Configuration Wizard.**

Step 2. Name the boot image in the **Configuration Name** field.

Step 3. Select **Expert Mode** and click **OK.** The Main Configuration window is displayed.

Step 4. Make any necessary changes to the coreboot settings and possibly to the bootfile settings. These procedures are described in the **Building the OS-9 ROM Image** section.

Step 5. Select **Configure -> Build Image** to display the Master Builder screen.

Step 6. Select the **Coreboot Only Image** setting and click **Build.**

Step 7. Click **Save As** to save the Coreboot image to a directory of your choice. If you do not have that directory on the drive, you can create it.

**Note**

If you are creating a new ROM Image instead of a Coreboot Image only, follow the steps described above and return to the Master Builder window. Select the **ROM Image** radio button and click **Build.**

This step creates a ROM image suitable for use with your binary ROM burner.
Step 8. Transfer the Coreboot Image (or ROM Image) to the EPROM with the EPROM programmer. You will need to follow the documentation for the EPROM programmer to complete this step.

Compressing the Bootfile Image

OS-9 bootfiles can be compressed to allow more modules to be loaded into a bootfile; this can be useful if you plan on storing your image on a small FLASH part or a floppy disk.

**Note**
The bootfile compression utility performs the compression at approximately a 2.5:1 ratio.

Complete the following steps to compress your image:

**Step 1.** Verify that your coreboot contains the uncompress module. This module can be found in the pre-built ROM and coreboot images that were shipped with your Microware OS-9 product.

**Note**
The uncompress module must be included in order for the compression to execute properly.

**Step 2.** Open the Configuration Wizard and select Configure -> Coreboot -> Main Configuration from the main menu.

**Step 3.** Select the Bootfile Compression tab. Verify that the Include bootfile uncompress module box is checked and select OK.
Step 4. When you are ready to build the image, open the **Master Builder** dialog. Verify that the **Compress Bootfile** box is checked and then press **Build** to begin the installing the image.
Chapter 2: Board-Specific Reference

This chapter contains information that is specific to the Intel StrongARM SA-1110 Microprocessor development board (Assabet) and the Intel StrongARM SA-1111 Microprocessor development module (Neponset). It includes the following sections:

- Boot Options
- The Fastboot Enhancement
- OS-9 Vector Mappings
- Assabet GPIO Usage
- Board-Specific Register Access Functions
- Port Specific Utilities

For More Information
For general information on porting OS-9, see the OS-9 Porting Guide.
Booting from Flash

When the romcnfg.h has a ROM search list defined the options bo and lr appear in the boot menu. If no search list is defined N/A appears in the boot menu. If an OS-9 bootfile is programmed into Flash in the address range defined in the default.des file, the system can boot and run from Flash.

bo

ROM boot—the system runs from the Flash bank.

lr

load to RAM—the system copies the Flash image into ram and runs from there.
Booting from PCMCIA ATA Card

The system can boot from a PC formatted PCMCIA hard card, which resides in the PCMCIA slot.

ide0

The file os9kboot is searched for in slot 0. If found it is copied to system RAM and runs from there.

Booting from PCMCIA Ethernet Card

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

eb

Ethernet boot—a PCMCIA card that supports Ethernet will use the bootp protocol to transfer a bootfile into RAM and the system runs from there.

Booting over Serial Communications Port via Kermit

The system can download a bootfile in binary form over its serial communication port at 115200 using the kermit protocol. The speed of this transfer depends of the size of the bootfile, but expect at least a 3 minute wait, dots will show the progress of the download. The communications port is located at header J14 and uses the SA1110’s SP3 UART.

ker

kermit boot—The os9kboot file is sent via the kermit protocol into system RAM and runs from there.
Restart Booter

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

q quit—quit and attempt to restart the booting process.

Break Booter

The break booter allows entry to the system level debugger (if one exists). If the debugger is not in the system the system will reset.

break break—break and enter the system level debugger rombug.

Example boot session and message.

OS-9 Bootstrap for the ARM

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

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

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

Select a boot method from the above menu: ide0

Wait for IDE drive ready.

IDE Model : ATA_FLASH
Number Heads : 0x0002
Total Cylinders : 0x03d8
Sectors Per Track : 0x0020

Checking Partitions : 0
Fat Type : 0x16
File Name : OS9KBOOT
File Size : 0x000fdeb0
Start Cluster : 0x00003a57
Reading Bootfile....

Boot Address : 0xc002c850
Boot Size : 0x000fdeb0

OS-9 kernel was found.
A valid OS-9 bootfile was found.
$
The Fastboot Enhancement

The Fastboot enhancements to OS-9 provide faster system bootstrap performance to embedded systems. The normal bootstrap performance of OS-9 is attributable to its flexibility. OS-9 handles many different runtime configurations to which it dynamically adjusts during the bootstrap process.

The Fastboot concept consists of informing OS-9 that the defined configuration is static and valid. These assumptions eliminate the dynamic searching OS-9 normally performs during the bootstrap process and enables the system to perform a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

Overview

The Fastboot enhancement consists of a set of flags that control the bootstrap process. Each flag informs some portion of the bootstrap code that a particular assumption can be made and that the associated bootstrap functionality should be omitted.

The Fastboot enhancement enables control flags to be statically defined when the embedded system is initially configured as well as dynamically altered during the bootstrap process itself. For example, the bootstrap code could be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources which would indicate different bootstrap requirements.

In addition, the Fastboot enhancement’s versatility allows for special considerations under certain circumstances. This versatility is useful in a system where all resources are known, static, and functional, but additional validation is required during bootstrap for a particular instance, such as a resource failure. The low-level bootstrap code may respond to some form of user input that would inform it that additional checking and system verification is desired.
Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. An entire 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within the set of data structures shared by the ModRom sub-components and the kernel. Hence, the field is available for modification and inspection by the entire set of system modules (high-level and low-level). Currently, there are six bit flags defined with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed below:

**B_QUICKVAL**

The **B_QUICKVAL** bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. This causes the CRC check on modules to be omitted. This option is a potential time saver, due to the complexity and expense of CRC generation. If a system has many modules in ROM, where access time is typically longer than RAM, omitting the CRC check on the modules will drastically decrease the bootstrap time. It is rare that corruption of data will ever occur in ROM. Therefore, omitting CRC checking is usually a safe option.

**B_OKRAM**

The **B_OKRAM** bit informs both the low-level and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on the defined RAM parameters. This allows system designers to specify a possible RAM range, which the system validates upon startup. Thus, the system can accommodate varying amounts of RAM. In an embedded system where the RAM limits are usually statically defined and presumed to be functional, however, there is no need to validate the defined RAM list. Bootstrap time is saved by assuming that the RAM definition is accurate.
**B_OKROM**

The B_OKROM bit causes acceptance of the ROM definition without probing for ROM. This configuration option behaves like the B_OKRAM option, except that it applies to the acceptance of the ROM definition.

**B_1STINIT**

The B_1STINIT bit causes acceptance of the first init module found during cold-start. By default, the kernel searches the entire ROM list passed up by the ModRom for init modules before it accepts and uses the init module with the highest revision number. In a statically defined system, time is saved by using this option to omit the extended init module search.

**B_NOIRQMASK**

The B_NOIRQMASK bit informs the entire bootstrap system that it should not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, some systems that have a well defined interrupt system (i.e. completely calmed by the sysinit hardware initialization code) and also have a requirement to respond to an installed interrupt handler during system startup can enable this option to prevent the ModRom and the kernel cold-start from disabling interrupts. This is particularly useful in power-sensitive systems that need to respond to “power-failure” oriented interrupts.

---

**Note**

Some portions of the system may still mask interrupts for short periods during the execution of critical sections.
B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization at all or systems that only require it for “power-on” reset conditions. Systems that only require parity initialization for initial “power-on” reset conditions can dynamically use this option to prevent parity initialization for subsequent “non-power-on” reset conditions.

Implementation Details

This section describes the compile-time and runtime methods by which the bootstrap speed of the system can be controlled.

Compile-time Configuration

The compile-time configuration of the bootstrap is provided by a pre-defined macro (BOOT_CONFIG), which is used to set the initial bit-field values of the bootstrap flags. You can redefine the macro for recompilation to create a new bootstrap configuration. The new over-riding value of the macro should be established by redefining the macro in the rom_config.h header file or as a macro definition parameter in the compilation command.

The rom_config.h header file is one of the main files used to configure the ModRom system. It contains many of the specific configuration details of the low-level system. Below is an example of how you can redefine the bootstrap configuration of the system using the BOOT_CONFIG macro in the rom_config.h header file:

```
#define BOOT_CONFIG (B_OKRAM + B_OKROM + B_QUICKVAL)
```

Below is an alternate example showing the default definition as a compile switch in the compilation command in the makefile:

```
SPEC_COPTS = -dNEWINFO -dNOPARITYINIT -dBOOT_CONFIG=0x7
```
This redefinition of the `BOOT_CONFIG` macro results in a bootstrap method that accepts the RAM and ROM definitions without verification, and also validates modules solely on the correctness of their module headers.

### Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine what user-defined bootstrap procedure should be used. An example P2 module is shown below.

**Note**

If the override is performed in the `sysinit2()` function, the effect is not realized until after the low-level system memory searches have been performed. This means that any runtime override of the default settings pertaining to the memory search must be done from the code in the P2 module code.

```c
#define NEWINFO
#include <rom.h>
#include <types.h>
#include <const.h>
#include <errno.h>
#include <romerrno.h>
#include <p2lib.h>

error_code p2start(Rominfo rinf, u_char *glbls)
{
    /* if switch or jumper setting is set... */
    if (switch_or_jumper == SET) {
        /* force checking of ROM and RAM lists */
        rinf->os->boot_config &= ~(B_OKROM+B_OKRAM);
    }
    return SUCCESS;
}
```
OS-9 Vector Mappings

This section contains the OS-9 vector mappings for the Intel SA1110 Assabet and the Intel SA1111 Neponset platforms.

The ARM standard defines exceptions 0x0-0x8. The OS-9 system maps these 1-1. External interrupts from vector 0x6 are expanded to the virtual vector rage shown below by the irq1110 and irq11x1 modules.

**Note**

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

**For More Information**


Intel documentation is provided online in PDF format at [www.intel.com](http://www.intel.com).
Table 2-1 shows the OS-9 IRQ assignment for the Assabet SA1110/SA1111 board.

**Table 2-1  OS-9 IRQ Assignment for the SA1110/SA1111 Assabet/Neponset Boards**

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>ARM Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Processor Reset</td>
</tr>
<tr>
<td>0x1</td>
<td>Undefined Instruction</td>
</tr>
<tr>
<td>0x2</td>
<td>Software Interrupt</td>
</tr>
<tr>
<td>0x3</td>
<td>Abort on Instruction Prefetch</td>
</tr>
<tr>
<td>0x4</td>
<td>Abort on Data Access</td>
</tr>
<tr>
<td>0x5</td>
<td>Unassigned/Reserved</td>
</tr>
<tr>
<td>0x6</td>
<td>External Interrupt</td>
</tr>
<tr>
<td>0x7</td>
<td>Fast Interrupt</td>
</tr>
<tr>
<td>0x8</td>
<td>Alignment error</td>
</tr>
</tbody>
</table>

Table 2-2 shows the OS-9 SA1110 Specific Functions.

**Table 2-2  SA1110 Specific Functions**

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1110 Specific Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>GPIO[0] Edge Detect</td>
</tr>
<tr>
<td>0x41</td>
<td>GPIO[1] Edge Detect</td>
</tr>
<tr>
<td>0x42</td>
<td>GPIO[2] Edge Detect</td>
</tr>
</tbody>
</table>
### Table 2-2 SA1110 Specific Functions (continued)

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1110 Specific Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x43</td>
<td>GPIO[3] Edge Detect</td>
</tr>
<tr>
<td>0x44</td>
<td>GPIO[4] Edge Detect</td>
</tr>
<tr>
<td>0x45</td>
<td>GPIO[5] Edge Detect</td>
</tr>
<tr>
<td>0x46</td>
<td>GPIO[6] Edge Detect</td>
</tr>
<tr>
<td>0x47</td>
<td>GPIO[7] Edge Detect</td>
</tr>
<tr>
<td>0x48</td>
<td>GPIO[8] Edge Detect</td>
</tr>
<tr>
<td>0x49</td>
<td>GPIO[9] Edge Detect</td>
</tr>
<tr>
<td>0x4a</td>
<td>GPIO[10] Edge Detect</td>
</tr>
<tr>
<td>0x4b</td>
<td>OR of GPIO edge detects 27 - 11</td>
</tr>
<tr>
<td>0x4c</td>
<td>LCD controller service request</td>
</tr>
<tr>
<td>0x4d</td>
<td>UDC service request (0)</td>
</tr>
<tr>
<td>0x4e</td>
<td>SDLC service request (1a)</td>
</tr>
<tr>
<td>0x4f</td>
<td>UART service request (1b)</td>
</tr>
<tr>
<td>0x50</td>
<td>UART/HSSP service request (2)</td>
</tr>
<tr>
<td>0x51</td>
<td>UART service request (3)</td>
</tr>
<tr>
<td>0x52</td>
<td>MCP service request (4a)</td>
</tr>
<tr>
<td>0x53</td>
<td>SSP service request (4b)</td>
</tr>
</tbody>
</table>
Table 2-2  SA1110 Specific Functions  (continued)

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1110 Specific Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x54</td>
<td>DMA controller channel 0</td>
</tr>
<tr>
<td>0x55</td>
<td>DMA controller channel 1</td>
</tr>
<tr>
<td>0x56</td>
<td>DMA controller channel 2</td>
</tr>
<tr>
<td>0x57</td>
<td>DMA controller channel 3</td>
</tr>
<tr>
<td>0x58</td>
<td>DMA controller channel 4</td>
</tr>
<tr>
<td>0x59</td>
<td>DMA controller channel 5</td>
</tr>
<tr>
<td>0x5a</td>
<td>OS timer 0</td>
</tr>
<tr>
<td>0x5b</td>
<td>OS timer 1</td>
</tr>
<tr>
<td>0x5c</td>
<td>OS timer 2</td>
</tr>
<tr>
<td>0x5d</td>
<td>OS timer 3</td>
</tr>
<tr>
<td>0x5e</td>
<td>One Hz clock tick</td>
</tr>
<tr>
<td>0x5f</td>
<td>RTC als alarm register</td>
</tr>
<tr>
<td>0x60</td>
<td>GPIO[11] Edge Detect (The 0x4b OR is broken out here)</td>
</tr>
<tr>
<td>0x61</td>
<td>GPIO[12] Edge Detect (to make each one distinct)</td>
</tr>
<tr>
<td>0x62</td>
<td>GPIO[13] Edge Detect</td>
</tr>
<tr>
<td>0x63</td>
<td>GPIO[14] Edge Detect</td>
</tr>
<tr>
<td>0x64</td>
<td>GPIO[15] Edge Detect</td>
</tr>
</tbody>
</table>
Table 2-3  SA1110 Specific Functions  (continued)

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1110 Specific Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x65</td>
<td>GPIO[16] Edge Detect</td>
</tr>
<tr>
<td>0x66</td>
<td>GPIO[17] Edge Detect</td>
</tr>
<tr>
<td>0x67</td>
<td>GPIO[18] Edge Detect</td>
</tr>
<tr>
<td>0x68</td>
<td>GPIO[19] Edge Detect</td>
</tr>
<tr>
<td>0x69</td>
<td>GPIO[20] Edge Detect</td>
</tr>
<tr>
<td>0x6a</td>
<td>GPIO[21] Edge Detect (Assabet CF irq)</td>
</tr>
<tr>
<td>0x6b</td>
<td>GPIO[22] Edge Detect</td>
</tr>
<tr>
<td>0x6c</td>
<td>GPIO[23] Edge Detect (Assabet UCB1300 irq)</td>
</tr>
<tr>
<td>0x6d</td>
<td>GPIO[24] Edge Detect (Assabet GFX irqs)</td>
</tr>
<tr>
<td>0x6e</td>
<td>GPIO[25] Edge Detect (Assabet 11x1 irqs)</td>
</tr>
<tr>
<td>0x6f</td>
<td>GPIO[26] Edge Detect</td>
</tr>
<tr>
<td>0x70</td>
<td>GPIO[27] Edge Detect</td>
</tr>
</tbody>
</table>

Table 2-3 shows the OS-9 SA1111 Specific functions.

Table 2-3  SA1111 Specific Functions

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1111 Specific Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x71</td>
<td>GPIOA[0] (GPIOA)</td>
</tr>
<tr>
<td>0x72</td>
<td>GPIOA[1] (GPIOA)</td>
</tr>
<tr>
<td>OS-9 IRQ #</td>
<td>SA1111 Specific Function</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>0x73</td>
<td>GPIOA[2] (GPIOA)</td>
</tr>
<tr>
<td>0x74</td>
<td>GPIOA[3] (GPIOA)</td>
</tr>
<tr>
<td>0x75</td>
<td>GPIOB[0] (GPIOB)</td>
</tr>
<tr>
<td>0x76</td>
<td>GPIOB[1] (GPIOB)</td>
</tr>
<tr>
<td>0x77</td>
<td>GPIOB[2] (GPIOB)</td>
</tr>
<tr>
<td>0x78</td>
<td>GPIOB[3] (GPIOB)</td>
</tr>
<tr>
<td>0x79</td>
<td>GPIOB[4] (GPIOB)</td>
</tr>
<tr>
<td>0x7a</td>
<td>GPIOB[5] (GPIOB)</td>
</tr>
<tr>
<td>0x7b</td>
<td>GPOIC[0] (GPOIC)</td>
</tr>
<tr>
<td>0x7c</td>
<td>GPOIC[1] (GPOIC)</td>
</tr>
<tr>
<td>0x7d</td>
<td>GPOIC[2] (GPOIC)</td>
</tr>
<tr>
<td>0x7e</td>
<td>GPOIC[3] (GPOIC)</td>
</tr>
<tr>
<td>0x7f</td>
<td>GPOIC[4] (GPOIC)</td>
</tr>
<tr>
<td>0x80</td>
<td>GPOIC[5] (GPOIC)</td>
</tr>
<tr>
<td>0x81</td>
<td>GPOIC[6] (GPOIC)</td>
</tr>
<tr>
<td>0x82</td>
<td>GPOIC[7] (GPOIC)</td>
</tr>
<tr>
<td>0x83</td>
<td>MsTxint (PS2 Mouse)</td>
</tr>
<tr>
<td>OS-9 IRQ #</td>
<td>SA1111 Specific Function</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>0x84</td>
<td>MsRxint (PS2 Mouse)</td>
</tr>
<tr>
<td>0x85</td>
<td>MsStopErrint (PS2 Mouse)</td>
</tr>
<tr>
<td>0x86</td>
<td>TpxInt (PS2 Trackpad)</td>
</tr>
<tr>
<td>0x87</td>
<td>TpRxInt (PS2 Trackpad)</td>
</tr>
<tr>
<td>0x88</td>
<td>TpStopErrint (PS2 Trackpad)</td>
</tr>
<tr>
<td>0x89</td>
<td>SspXmitint (SSP)</td>
</tr>
<tr>
<td>0x8a</td>
<td>SspRcvint (SSP)</td>
</tr>
<tr>
<td>0x8b</td>
<td>SspROR (SSP)</td>
</tr>
<tr>
<td>0x8c</td>
<td>reserved</td>
</tr>
<tr>
<td>0x8d</td>
<td>reserved</td>
</tr>
<tr>
<td>0x8e</td>
<td>reserved</td>
</tr>
<tr>
<td>0x8f</td>
<td>reserved</td>
</tr>
<tr>
<td>0x90</td>
<td>reserved</td>
</tr>
<tr>
<td>0x91</td>
<td>AudXmtDmaDoneA (AUDIO)</td>
</tr>
<tr>
<td>0x92</td>
<td>AudRcvDmaDoneA (AUDIO)</td>
</tr>
<tr>
<td>0x93</td>
<td>AudXmtDmaDoneB (AUDIO)</td>
</tr>
<tr>
<td>0x94</td>
<td>AudRcvDmaDoneB (AUDIO)</td>
</tr>
<tr>
<td>OS-9 IRQ #</td>
<td>SA1111 Specific Function</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>0x95</td>
<td>AudTFSR (AUDIO)</td>
</tr>
<tr>
<td>0x96</td>
<td>AudRFSR (AUDIO)</td>
</tr>
<tr>
<td>0x97</td>
<td>AudTUR (AUDIO)</td>
</tr>
<tr>
<td>0x98</td>
<td>AudROR (AUDIO)</td>
</tr>
<tr>
<td>0x99</td>
<td>AudDTS (AUDIO)</td>
</tr>
<tr>
<td>0x9a</td>
<td>AudRDD (AUDIO)</td>
</tr>
<tr>
<td>0x9b</td>
<td>AudSTO (AUDIO)</td>
</tr>
<tr>
<td>0x9c</td>
<td>USBPwr (AUDIO)</td>
</tr>
<tr>
<td>0x9d</td>
<td>nIrqHciM (USB)</td>
</tr>
<tr>
<td>0x9e</td>
<td>IrqHciBuffAcc (USB)</td>
</tr>
<tr>
<td>0x9f</td>
<td>IrqHciRmtWkp (USB)</td>
</tr>
<tr>
<td>0xa0</td>
<td>nHciMFCIr (USB)</td>
</tr>
<tr>
<td>0xa1</td>
<td>USB port resume (USB)</td>
</tr>
<tr>
<td>0xa2</td>
<td>S0Readynint (PCMCIA)</td>
</tr>
<tr>
<td>0xa3</td>
<td>S1Readynint (PCMCIA)</td>
</tr>
<tr>
<td>0xa4</td>
<td>S0CDValid (PCMCIA)</td>
</tr>
<tr>
<td>0xa5</td>
<td>S1CDValid (PCMCIA)</td>
</tr>
</tbody>
</table>
Table 2-3  SA1111 Specific Functions (continued)

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>SA1111 Specific Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xa6</td>
<td>S0_Bvd1Stschg (PCMCIA)</td>
</tr>
<tr>
<td>0xa7</td>
<td>S1_Bvd1Stschg (PCMCIA)</td>
</tr>
<tr>
<td>0xa8</td>
<td>reserved</td>
</tr>
<tr>
<td>0xa9</td>
<td>reserved</td>
</tr>
<tr>
<td>0xaa</td>
<td>reserved</td>
</tr>
<tr>
<td>0xab</td>
<td>reserved</td>
</tr>
<tr>
<td>0xac</td>
<td>reserved</td>
</tr>
<tr>
<td>0xad</td>
<td>reserved</td>
</tr>
<tr>
<td>0xae</td>
<td>reserved</td>
</tr>
<tr>
<td>0xaf</td>
<td>reserved</td>
</tr>
<tr>
<td>0xb0</td>
<td>reserved</td>
</tr>
</tbody>
</table>

Table 2-4 shows the Neponset Specific Functions.

Table 2-4 Neponset Specific Functions

<table>
<thead>
<tr>
<th>OS-9 IRQ #</th>
<th>Neponset Specific Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb1</td>
<td>Ethernet Iq</td>
</tr>
<tr>
<td>0xb2</td>
<td>USAR Iq</td>
</tr>
</tbody>
</table>
Assabet GPIO Usage

For More Information
See the Intel *Assabet User’s Manual* for alternate pin functions.

Table 2-5 shows GPIO usage of the Assabet board as used by OS-9.

**Table 2-5  GPIO Usage of the Assabet Board**

<table>
<thead>
<tr>
<th>GPIO</th>
<th>Signal Name</th>
<th>Direct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO0</td>
<td>ON_OFF_SW1</td>
<td>Input</td>
<td>Switch 1 (Abort Switch for OS-9)</td>
</tr>
<tr>
<td>GPIO1</td>
<td>ON_OFF_SW2_FIQ</td>
<td>Input</td>
<td>Switch 2</td>
</tr>
<tr>
<td>GPIO2</td>
<td>LCD_D8_CNFG0</td>
<td>Output</td>
<td>LCD Green bit 3 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO3</td>
<td>LCD_D9_CNFG1</td>
<td>Output</td>
<td>LCD Green bit 4 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO4</td>
<td>LCD_D10_CNFG2</td>
<td>Output</td>
<td>LCD Green bit 5 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO5</td>
<td>LCD_D11_CNFG3</td>
<td>Output</td>
<td>LCD Red bit 0 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO6</td>
<td>LCD_D12_CNFG4</td>
<td>Output</td>
<td>LCD Red bit 1 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO7</td>
<td>LCD_D13_CNFG5</td>
<td>Output</td>
<td>LCD Red bit 2 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO8</td>
<td>LCD_D14_CNFG6</td>
<td>Output</td>
<td>LCD Red bit 3 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO9</td>
<td>LCD_D15_CNFG7</td>
<td>Output</td>
<td>LCD Red bit 4 in 16 bit color mode</td>
</tr>
<tr>
<td>GPIO10</td>
<td>SSP_UDA134_TXD</td>
<td>Output</td>
<td>SSP Port transmit</td>
</tr>
<tr>
<td>GPIO</td>
<td>Signal Name</td>
<td>Direct</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------</td>
<td>--------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>GPIO11</td>
<td>SSP_UDA134_RXD</td>
<td>Input</td>
<td>SSP Port Receive</td>
</tr>
<tr>
<td>GPIO12</td>
<td>SSP_UDA134_SCLK</td>
<td>Output</td>
<td>SSP Port Clock</td>
</tr>
<tr>
<td>GPIO13</td>
<td>SSP_UDA134_SFRM</td>
<td>Output</td>
<td>SSP Port Frame</td>
</tr>
<tr>
<td>GPIO14</td>
<td>RADIO_IRQ</td>
<td>Input</td>
<td>Radio IRQ input</td>
</tr>
<tr>
<td>GPIO15</td>
<td>L1_I2C_SDA</td>
<td>Output</td>
<td>Data line for shared I2C/L3 serial bus</td>
</tr>
<tr>
<td>GPIO16</td>
<td>PS_MODE_SYNC</td>
<td>Input</td>
<td>Power Mode switch input</td>
</tr>
<tr>
<td>GPIO17</td>
<td>L3_MODE</td>
<td>Output</td>
<td>L3 control bus signal</td>
</tr>
<tr>
<td>GPIO18</td>
<td>L3_I2C_SCL</td>
<td>Output</td>
<td>Clock line for shared I2C/L3 serial bus</td>
</tr>
<tr>
<td>GPIO19</td>
<td>STERO_64FS_CLK</td>
<td>Input</td>
<td>External Input clock for SSP</td>
</tr>
<tr>
<td>GPIO20</td>
<td>UART3_CLK</td>
<td>Input</td>
<td>External Input clock for SP3</td>
</tr>
<tr>
<td>GPIO21</td>
<td>nMBGNT_CF_IRQ</td>
<td>Input</td>
<td>ARM Bus grant/Assabet Compact Flash IRQ</td>
</tr>
<tr>
<td>GPIO22</td>
<td>nMBREQ_CF_CARDDDET</td>
<td>In/Out</td>
<td>ARM Bus request/Assabet CF Card Detect</td>
</tr>
<tr>
<td>GPIO23</td>
<td>UCB1300_IRQ</td>
<td>Input</td>
<td>UCB 1300 Irq</td>
</tr>
<tr>
<td>GPIO24</td>
<td>GFX_IRQ_CFBVD2_Irq</td>
<td>Input</td>
<td>Graphics development board IRQ (MQ200)</td>
</tr>
<tr>
<td>GPIO25</td>
<td>SA111_IRQ_CF_BVD1</td>
<td>Input</td>
<td>1111 Development board Irq (Neponset)</td>
</tr>
</tbody>
</table>
When GPIOs are used as interrupt sources, the `PIC_ENABLE()` function will set default polarity based on settings in `systype.h` along with enabling the interrupt at the SA1110 PIC. If the opposite edge is required, software must assert/negate the appropriate bits in the GFER/GRER (if no edge defines were in `systype.h`, the pic code will default to rising edge).

### Table 2-5 GPIO Usage of the Assabet Board (continued)

<table>
<thead>
<tr>
<th>GPIO</th>
<th>Signal Name</th>
<th>Direct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO26</td>
<td>VBAD_LOW/RCLK</td>
<td>Input</td>
<td>Battery Low/Ref clk out.</td>
</tr>
<tr>
<td>GPIO27</td>
<td>3.68M_32K</td>
<td>Output</td>
<td>3.68Mhz output clock.</td>
</tr>
</tbody>
</table>
Board-Specific Register Access Functions

A board-specific global storage area has been defined for the Assabet due to the board's use of the write-only register (BCR) in a variety of Assabet board functions.

For More Information

In order to guarantee the coherency of the BCR value, several functions have been defined to give access to the global storage area. These functions provide atomic updates for bit or register operations and can be used by system state code by linking to the library hwlib.1, which is found at in the following location:


The provided OS-9 drivers all link against this library to access BCR values. Bit defines for the BCR can be found in the ports systype.h file. A brief description of the access functions is provided below.

The global area is allocated at boot time by the oemglobal module and its access pointer is contained within the OS-9 rominfo structure (RINF).

The global area consists of an array of 8 four byte values.

- hardware_shadow_globals[0] - BCR value (read/write address).
- hardware_shadow_globals[1] - SCR value (probed value)
- hardware_shadow_globals[3-7] - Reserved.
The following functions are used to access or update any of the globals and their associated hardware shadows. The use of the bit operations is preferred.

- `clear_oem_reg_bit()` - Clears a bit to 0.
- `set_oem_reg_bit()` - Sets a bit to a 1.
- `get_oem_reg_bit()` - Returns the value of 1 or 0 for a bit.
- `set_oem_reg()` - Sets the 32bit value of an array entry.
- `get_oem_reg()` - Returns the 32bit value of an array entry.

All the global access functions take the `rinf` as a parameter. In boot modules prior to the kernel (system call ability) the `rinf` is passed as a parameter to each ROM p2module. After the kernel is up the `rinf` can be obtained with the `os_getrinf()` function (see example below).

```c
void clear_oem_reg_bit(u_int32 regoffset, u_int32 bitnum, Rominfo rinf)
void set_oem_reg_bit(u_int32 regoffset, u_int32 bitnum, Rominfo rinf)
u_int32 get_oem_reg_bit(u_int32 regoffset, u_int32 bitnum, Rominfo rinf)

regoffset is the offset to the array entry to access.
bitnum is the bit number to access.
rinf is a pointer to the rom information structure.
```

```c
get_oem_reg_bit() returns the value of the referenced bit (0 or 1)
u_int32 get_oem_reg(u_int32 regoffset, Rominfo rinf)
```

```c
regoffset is the offset to the array entry to access.
rinf is a pointer to the rom information structure.
```

```c
get_oem_reg returns the 32 bit value of the selected array entry on success, or -1 on error if a bad rinf is passed.

error_code set_oem_reg(u_int32 regoffset, u_int32 new_value, u_int32
 *old_value, Rominfo rinf);

regoffset is the offset to the array entry to access.
new_value is the value to be set the array entry.
old_value is a pointer to the previous value of the selected OEM reg
(obtained by get_oem_reg()).
```
set_oem_reg returns SUCCESS if the operation succeeded. If the operation failed because the old_value is different than current value, a -1 is returned. If the operation failed due to a bad rinf, EOS_ILLPRM is returned.

---

**Note**

get_oem_reg() must be called before a call to set_oem_reg(), this insures that the associated array entry has not asynchronously changed from another context. The set_oem_reg() call should be in a loop with get_oem_reg() until SUCCESS is reached (this will almost always be on the first attempt).

---

**Example**

```c
#include <systype.h>  /* get BCR/SCR def's /mwos/.../assabet */
#include <rom.h>      /* get rinf defs, /mwos/src/defs/rom */
#include <p2lib.h>    /* get os_getrinf protos /mwos/src/defs/rom */

/* Power up the local Assabet LCD */
static void _lcdOn()
{
    Rominfo rinf;  /* allocate local pointer for rinf */
    os_getrinf(&rinf);  /* get global rom structure pointer */

    /* Power up local LCD if NO external GFX board preset*/
    if(get_oem_reg_bit(A_SCR_OFFSET, GFXDB, rinf))
    {
        set_oem_reg_bit(A_BCR_OFFSET,LCDON, rinf);  /* Enable power to LCD panel */
        clear_oem_reg_bit(A_BCR_OFFSET,LCD12_16, rinf);  /* Set to RGB444 */
        set_oem_reg_bit(A_BCR_OFFSET,LIGHT, rinf);  /* BackLight on if there */
    }
    return;
}
```
Port Specific Utilities

The following port specific utilities are included:

- pcmcia
- touch_cal
- ucbtouch
**pcmcia**

**Syntax**

`pcmcia [<opts>]`

**options**

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

**Description**

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

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

### Syntax

```
touch_cal <options>
```

### Options

- `-f[=]<name>`: Output filename
- `-c`: Only run calibration if output filename does not exist
- `-m[=]<font_module>`: Use given UCM font module to display text

### Description

The `touch_cal` utility will present a text message on the LCD screen as well as points for the user to press. After the points are pressed, the protocol module `mp_ucb1200` will be updated with the new calibration information.

### Example

```
$ touch_cal
Found touch screen device '/ucb_touch/mp_ucb1200'
```
ucbtouch

Syntax

ucbtouch <>

Description

The ucbtouch utility prints the raw x,y and pressure values at a set sample rate.

Press the touch screen and observe the output on your console. The utility is helpful in determining whether your touch screen is connected properly.

Example

$ ucbtouch
Touch[00000]: Touch=0x30c3 X1=00328 Y1=00321 P= 28 X=329 Y=322
Touch[00001]: Touch=0x30c3 X1=00329 Y1=00325 P= 28 X=330 Y=326
Touch[00002]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00003]: Touch=0x30c3 X1=00329 Y1=00321 P= 29 X=330 Y=322
Touch[00004]: Touch=0x30c3 X1=00329 Y1=00319 P= 29 X=330 Y=320
Touch[00005]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00006]: Touch=0x30c3 X1=00329 Y1=00327 P= 28 X=330 Y=328
Touch[00007]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00008]: Touch=0x30c3 X1=00329 Y1=00321 P= 29 X=330 Y=322
Touch[00009]: Touch=0x30c3 X1=00329 Y1=00322 P= 28 X=330 Y=323
Touch[00010]: Touch=0x30c3 X1=00329 Y1=00319 P= 28 X=0 Y=0
Touch[00011]: Touch=0x30c3 X1=00328 Y1=00321 P= 28 X=-1 Y=2
Touch[00012]: Touch=0x30c3 X1=00329 Y1=00315 P= 28 X=0 Y=-4
Touch[00013]: Touch=0x30c3 X1=00329 Y1=00322 P= 29 X=0 Y=3
Appendix A: Board-Specific Modules

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

- Low-Level System Modules
- High-Level System Modules
For More Information

For a complete list of OS-9 modules common to all boards, see the *OS-9 Device Descriptor and Configuration Module Reference* manual.

The following low-level system modules are tailored specifically for the Intel SA1110 Assabet platform. The functionality of these modules can be altered through changes to the configuration data module (cnfgdata). Table A-1 provides a list and brief description of the modules.

These modules can be found in the following directory:

MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBJ/ROM

**Table A-1  Assabet-Specific Low-Level System Modules**

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnfgdata</td>
<td>Contains the low-level configuration data.</td>
</tr>
<tr>
<td>cnfgfunc</td>
<td>Provides access services to cnfgdata data.</td>
</tr>
<tr>
<td>commcnfg</td>
<td>Inits communication port defined in cnfgdata.</td>
</tr>
<tr>
<td>conscnfg</td>
<td>Inits console port defined in cnfgdata.</td>
</tr>
<tr>
<td>ide</td>
<td>IDE boot support module. PCMCIA compatible.</td>
</tr>
<tr>
<td>io1100</td>
<td>Provides polled serial driver support for the low-level system.</td>
</tr>
<tr>
<td>Module Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ll91c94</td>
<td>Provides low-level Ethernet services.</td>
</tr>
<tr>
<td>llcis</td>
<td>Inits the PCMCIA interface including cards.</td>
</tr>
<tr>
<td>llne2000</td>
<td>Provides low-level Ethernet services via SOCKET-LPE PCMCIA card.</td>
</tr>
<tr>
<td>lle509</td>
<td>Provides low-level Ethernet services via 3COM PCMCIA card.</td>
</tr>
<tr>
<td>lle509_pcm</td>
<td>Provides low-level Ethernet services.</td>
</tr>
<tr>
<td>portmenu</td>
<td>Inits booters defined in the cnfgdata.</td>
</tr>
<tr>
<td>romcore</td>
<td>Board specific initialization code.</td>
</tr>
<tr>
<td>splash</td>
<td>Provides way to init LCD screen with a compressed image.</td>
</tr>
<tr>
<td>oemglob</td>
<td>Creates a shared variable area for high/low level system interactions.</td>
</tr>
<tr>
<td>dbinit</td>
<td>Initalizes any daughter boards present.</td>
</tr>
<tr>
<td>tmr1_1100</td>
<td>Provides low-level timer services via time base register.</td>
</tr>
<tr>
<td>usedebug</td>
<td>Inits low-level debug interface to RomBug, SNDP, or none.</td>
</tr>
</tbody>
</table>
The following low-level system modules provide generic services for OS9000 Modular ROM. Table A-2 provides a list and brief description of the modules.

These modules can be found in the following directory:
MWOS/OS9000/ARMV4/CMDS/BOOTOBJS/ROM

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootsys</td>
<td>Booter registration service module.</td>
</tr>
<tr>
<td>console</td>
<td>Provides console services.</td>
</tr>
<tr>
<td>dbgentry</td>
<td>Inits debugger entry point for system use.</td>
</tr>
<tr>
<td>dbgerv</td>
<td>Provides debugger services.</td>
</tr>
<tr>
<td>exception</td>
<td>Provides low-level exception services.</td>
</tr>
<tr>
<td>flshcach</td>
<td>Provides low-level cache management services.</td>
</tr>
<tr>
<td>hlproto</td>
<td>Provides user level code access to protoman.</td>
</tr>
<tr>
<td>llbootp</td>
<td>Booter which provides bootp services.</td>
</tr>
<tr>
<td>llip</td>
<td>Provides low-level IP services.</td>
</tr>
<tr>
<td>llslip</td>
<td>Provides low-level SLIP services.</td>
</tr>
<tr>
<td>lltcp</td>
<td>Provides low-level TCP services.</td>
</tr>
<tr>
<td>lludp</td>
<td>Provides low-level UDP services.</td>
</tr>
<tr>
<td>llkermit</td>
<td>Booter which uses kermit protocol.</td>
</tr>
<tr>
<td>Module Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>notify</td>
<td>Provides state change information for use with LL and HL drivers.</td>
</tr>
<tr>
<td>override</td>
<td>Booter which allows choice between menu and auto booters.</td>
</tr>
<tr>
<td>parser</td>
<td>Provides argument parsing services.</td>
</tr>
<tr>
<td>pcman</td>
<td>Booter which reads MS-DOS file system.</td>
</tr>
<tr>
<td>protoman</td>
<td>Protocol management module.</td>
</tr>
<tr>
<td>restart</td>
<td>Booter which cause a soft reboot of system.</td>
</tr>
<tr>
<td>romboot</td>
<td>Booter which allows booting from ROM.</td>
</tr>
<tr>
<td>rombreak</td>
<td>Booter which calls the installed debugger.</td>
</tr>
<tr>
<td>rombug</td>
<td>Low-level system debugger.</td>
</tr>
<tr>
<td>sndp</td>
<td>Provides low-level system debug protocol.</td>
</tr>
<tr>
<td>srecord</td>
<td>Booter which accepts S-Records.</td>
</tr>
<tr>
<td>swtimer</td>
<td>Provides timer services via software loops.</td>
</tr>
</tbody>
</table>
High-Level System Modules

The following OS-9 system modules are tailored specifically for your Intel SA1110 Assabet board and peripherals. Unless otherwise specified, each module is located in a file of the same name in the following directory:

MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBJ

CPU Support Modules

These files are located in the following directory:

MWOS/OS9000/ARMV4/CMDS/BOOTOBJ

kernel The kernel provides all basic services for the OS-9 system.
cache Provides cache control for the CPU cache hardware. The cache module is in the file cach1100.
fpu Provides software emulation for floating point instructions.
ssm The System Security Module provides support for the Memory Management Unit (MMU) on the CPU.
_vectors Provides interrupt service entry and exit code. The vectors module is found in the file vect110.
Board-Specific Modules

System Configuration Module

These files are located in the following directory:

MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBS/INITS

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

Power Management Support Modules

These modules provide an interface to control the power states of the Assabet device.

The supported SA1110 CPU power states are RUN, IDLE and SLEEP.

- pwrman: P2module which provides generic power management functions.
- pwrplcy: P2module which provides power state control functions.
- sysif: P2module which provides SA1100 CPU power state control.

Interrupt Controller Support

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

For More Information

The mappings are described in Chapter 2.
### Board-Specific Modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq1100</td>
<td>P2module that provides interrupt acknowledge and dispatching support for the SA1100 pic.</td>
</tr>
<tr>
<td>irq1111</td>
<td>P2module which provides interrupt acknowledge and dispatching support for the SA1111 pic (vector range 0x71-0xB2).</td>
</tr>
</tbody>
</table>

### Real Time Clock

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc1100</td>
<td>Driver that provides OS-9 access to the SA1110 on-board real time clock.</td>
</tr>
</tbody>
</table>

### Ticker

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk1100</td>
<td>Driver that provides the system ticker based on the SA1110 Operating System Timer.</td>
</tr>
</tbody>
</table>

### Abort Handler

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>P2module which provides a way to enter the system-state debugger via vector 0xb40 interrupt triggered by Assabet button at S1.</td>
</tr>
</tbody>
</table>

### Generic IO Support modules (File Managers)

These files are located in the following directory:

```
MWOS/OS9000/ARMV3/CMDS/BOOTOBS
```

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioman</td>
<td>Provides generic IO support for all IO device types.</td>
</tr>
<tr>
<td>scf</td>
<td>Provides generic character device management functions.</td>
</tr>
</tbody>
</table>
Board-Specific Modules

- **rbf**: Provides generic block device management functions for OS-9 specific format.
- **pcf**: Provides generic block device management functions for MS-DOS FAT format.
- **spf**: Provides generic protocol device management function support.
- **mfm**: Provides generic graphics device support for MAUI®.
- **pipeman**: Provides a memory FIFO buffer for communication.

Pipe Descriptor

This file is located in the following directory:

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

- **pipe**: Pipeman descriptor that provides a RAM based FIFO that can be used for process communication.

RAM Disk Support

- **ram**: RBF driver that provides a RAM based virtual block device.

Descriptors for Use with RAM

These files are located in the following directory:

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

- **r0**: RBF descriptor that provides access to a ram disk.
- **r0.dd**: Same as r0 except with module name dd (for use as the default device).
ABoard-Specific Modules

Serial and Console Devices

**sc1100**
SCF driver that provides serial support the SA1110's SP1 and SP3 ports when configured as UARTS.

**Descriptors for Use with sc1100**

**term1/t1**
Descriptor modules for use with sc1100 and SP1 (default term).
- Assabet Board header: J10
- Default Baud Rate: 19200
- Default Parity: None
- Default Data Bits: 8
- Default Handshake: Software

**term3/t3**
Descriptor modules for use with sc1100 and SP3.
- Assabet Board header: J14
- Default Baud Rate: 115200
- Default Parity: None
- Default Data Bits: 8
- Default Handshake: Software

**sc1101**
SCF driver that provides serial support for the SA1111.

**Descriptors for use with sc1101**

**m1**
Descriptor module used with the sc1101 and SP1.

**m2**
Descriptor module used with the sc1101 and SP2.
Board-Specific Modules

**sc16550**  
SCF driver that provides serial support for PCMCIA modem cards.

**Descriptors for use with sc16550**

<table>
<thead>
<tr>
<th>Descriptors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0m</td>
<td>Descriptor modules for use with PCMCIA modem cards.</td>
</tr>
<tr>
<td>Assabet Board header:</td>
<td>J6</td>
</tr>
<tr>
<td>Default Baud Rate:</td>
<td>9600</td>
</tr>
<tr>
<td>Default Parity:</td>
<td>None</td>
</tr>
<tr>
<td>Default Data Bits:</td>
<td>8</td>
</tr>
<tr>
<td>Default Handshake:</td>
<td>Software</td>
</tr>
</tbody>
</table>

**scllio**  
SCF driver that provides serial support via the polled low-level serial driver.

**Descriptors for Use with scllio**

<table>
<thead>
<tr>
<th>Descriptors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcons/term</td>
<td>Descriptor modules for use with scllio in conjunction with a low-level serial driver. Port configuration and set up follows what is configured in cnfgdata for the console port. It is possible for scllio to communicate with a true low-level serial device driver like io1100, or with an emulated serial interface provided by iovcons.</td>
</tr>
</tbody>
</table>
**PCMCIA Support for IDE Type Devices**

**rb1003**

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

**Descriptors for Use with rb1003**

**hc1, hc1fmt, hc1.dd**

RBF Descriptor modules for use with PCMCIA slot #0

- Assabet Board header: J6
- hc1fmt: format enabled
- hc1.dd: module name of dd

**mhcl, mhclfmt, mhclfmt.dd mhcl.dd**

PCF Descriptor modules for use with PCMCIA slot #0 on the Assabet

- Assabet Board header: J6
- mhclfmt: format enabled
- mhclfmt.dd: format enabled
- mhcl.dd: module name of dd

**mhcl_d, mhcl_dfmt, mhcl_dfmt.dd mhcl_d.dd**

PCF Descriptor modules for use with PCMCIA slot #0 on the Neponset

- Neponset Board header: J23
- mhcl_dfmt: format enabled
- mhcl_dfmt.dd: format enabled
- mhcl_d.dd: module name of dd
PCF Descriptor modules for use with PCMCIA slot #1 on the Neponset Descriver modules

Board header: J23
mhe1fmt: format enabled
mhe1fmt.dd format enabled
mhe1.dd: module name of dd

PCMCIA Support for Socket-LPE Ethernet Card (NE2000 Compatible)

These files are located in the following directory:
MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBS/SPF
spne2000 SPF driver to support ethernet for a Socket-LPE CF card.

Descriptors for Use with spne2000

spne0 SPF descriptor module for use with PCMCIA slot #0 (bottom J6)
spne1 SPF descriptor module for use with PCMCIA slot #1 (J23, Neponset)

PCMCIA Support for 3COM Ethernet Card

These files are found in the following directory:
MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBS/SPF
spe509_pcm SPF driver to support ethernet for a 3COM EtherLink III PCMCIA card.
Board-Specific Modules

**spe509_pcm Descriptors**

- **spe30**: SPF descriptor module for use with PCMCIA slot #0 (J6).
- **spe31**: SPF descriptor module for use with PCMCIA slot #1 (J23, Neponset).

**SMC91C94 Ethernet Support**

These files are located in the following directory:

```
MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBJ/SPF
```

- **sp91c94**: SPF driver to support ethernet for the SMC91C94 chip (on Neponset).

**Descriptor for Use with sp91c94**

- **spsm0**: SPF descriptor module for use with SMC91C94 at U2 (on Neponset).

**Network Configuration Modules**

- **inetdb/inetdb2/rpcdb**
UCB1300 Support Modules

**Note**
These drivers and descriptors work with UCB1100/1200/1300.

These files are located in the following directory:
MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBJ/SPF

| spucb1200       | SPF driver that supports the on-board Phillips UCB1200 chip. This device communicates to the SA1100 over SP4 using MCP. |

Descriptors for Use with spucb1200

| ucb             | SPF descriptor module that provides access to UCB1200. |
| ucb_touch       | SPF descriptor module used with the touch screen. |

Maui Graphical Support Modules

These files are located in the following directory:
MWOS/OS9000/ARMV4/PORTS/ASSABET/CMDS/BOOTOBJ/MAUI

| gx_sa1100       | MFM MAUI driver module with support for the Assabet LCD panel. |

Descriptors for Use with gx_sa1100

| gfx             | MFM MAUI descriptor module for Assabet LCD. |
Board-Specific Modules

sd_ucb1200  MFM MAUI driver module that provides PCM/mu-law sound support via the ucb1300. The UDA1341 must be configured to pass through UCB1300 signals.

Descriptors for Use with sd_ucb1200

snd  MFM MAUI descriptor module for UCB1200 sound functions.

MAUI configuration modules

cdb  MAUI configuration data base module.
cdb_ptr  Serial mouse configuration data base module.
cdb_touch  Touch screen configuration data base module.

MAUI protocol modules

mp_kybrd  Keyboard protocol module.
mp_msptr  Serial mouse protocol module.
mp_ucb1200  ucb1200 protocol module.

For More Information
The MAUI drivers are described in more detail in Appendix B: MAUI Driver Descriptions.
Appendix B: MAUI Driver Descriptions

This chapter provides MAUI driver descriptions. It includes the following sections:

- Assabet Objects
- GX_SA1100 LCD Graphic Driver Specification
- SD_UCB1200 Sound Driver Specification
- SPUCB1200 driver for the UCB1200 Codec
- MP_UCB1200 MAUI Touch screen Protocol Module
**Assabet Objects**

This package provides object-level support for the Intel Assabet reference board. The port directory is at the following location:

MWOS/OS9000/ARMV4/PORTS/ASSABET

**MAUI objects**

- **cdb**  Lists the devices on the system.
- **mp_msptr**  Serial mouse protocol module.
- **mp_ucc1200**  Touch screen protocol module for the UCB1200.
- **gfx and gx_sa1100**  LCD graphics descriptor and driver.
This section describes the hardware specification of the StrongARM SA1110 LCD driver (named gx_sa1100) and descriptor (named gfx). The hardware sub-type defines the board configuration. This specification should be used with the MAUI Graphics Device API.

Board Ports

This driver is used in the following example board StrongArm ports. The GraphicsClient board uses a Sharp LQ64D341 18 bpp color (16 used), TFT, with a resolution of 640x480 single panel. This panel is connected to the GraphicsClient with one of several possible cables:

- 8 bpp - most common to date
- RGB 565 - next most common
- RGB 655
- RGB 556

The SideArm board can support an LCD panel, but does not typically ship with one. For this reason the SideArm port does not build this driver. If the user did connect a LCD panel to this board, simply copy the makefiles from one of the other ports into the SideArm port.

The Assabet board uses a Sharp LQ039Q2DS01, TFT, with a resolution of 320x240 single panel. This panel is connected to the Assabet with one of several possible cables:

- 8 bpp - most common to date
- RGB 565 - next most common
Device Capabilities

Information about the hardware capabilities is determined by calling `gfx_get_dev_cap()`. The hardware sub-type defines the board configuration. This function returns a data structure formatted as shown in Table B-1. See `GFX_DEV_CAP` for more information about this data structure.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw_type</td>
<td>Hardware type (embedded in driver)</td>
<td>SA1100 LCD Controller</td>
</tr>
<tr>
<td>hw_subtype</td>
<td>Hardware subtype (embedded in descriptor)</td>
<td>Assabet 8 bit color LCD, or Assabet 16 bit color LCD</td>
</tr>
<tr>
<td>sup_vpmix</td>
<td>Supports viewport mixing</td>
<td>FALSE</td>
</tr>
<tr>
<td>sup_extvid</td>
<td>Supports external video as a backup</td>
<td>FALSE</td>
</tr>
<tr>
<td>sup_bkcol</td>
<td>Supports background color</td>
<td>FALSE</td>
</tr>
<tr>
<td>sup_vptrans</td>
<td>Supports viewport transparency</td>
<td>FALSE</td>
</tr>
<tr>
<td>sup_vpinten</td>
<td>Supports viewport intensity</td>
<td>FALSE</td>
</tr>
<tr>
<td>sup_sync</td>
<td>Supports retrace synchronization</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
### Table B-1  gfx_get_dev_cap() Data Structure  (continued)

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_res</td>
<td>Number of display resolutions</td>
<td>1</td>
</tr>
<tr>
<td>res_info</td>
<td>Array of display resolution information</td>
<td>See Display Resolution table</td>
</tr>
<tr>
<td>dac_depth</td>
<td>Depth of the DAC in bits</td>
<td>12</td>
</tr>
<tr>
<td>num_cm</td>
<td>Number of coding methods</td>
<td>1</td>
</tr>
<tr>
<td>cm_info</td>
<td>Array of coding method information</td>
<td>See Coding Methods table</td>
</tr>
<tr>
<td>sup_viddecode</td>
<td>Supports video decoding into a drawmap</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

## Display Resolution

The display resolution is configured by the descriptor and can be changed to support LCD panels of different sizes. The driver is only designed to support one resolution at a time. That resolution is
specified by the descriptor. Modify the `DEFAULT_RES` macro in `mfm_desc.h` to change the resolution. If you change the resolution, you must also change all of the LCD timing fields as well.

*Refresh rate is determined by timing specified in descriptor. The devcap is not automatically update to reflect this.*

### Coding Methods

The coding method is also configured by the descriptor and can be changed to support b/w and color LCD panels. The coding method can be selected in the descriptor by simply specifying the coding method in the `DEFAULT_CM` macro in `mfm_desc.h`.

<table>
<thead>
<tr>
<th>Board</th>
<th>Width</th>
<th>Height</th>
<th>Refresh Rate</th>
<th>Interlace Mode</th>
<th>Aspect Ratio X:Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics-Client</td>
<td>640</td>
<td>480</td>
<td>0*</td>
<td>GFX_INTL_OFF</td>
<td>1:1</td>
</tr>
</tbody>
</table>
This driver was verified on the Assabet board with both a 8-bit and 565 cables. The maximal coding method supported by SA1100 LCD Controller is 16 bpp.

### Table B-3  Coding Method Description

<table>
<thead>
<tr>
<th>Board</th>
<th>Coding Method</th>
<th>CLUT Based</th>
<th>X,Y Multipliers</th>
<th>Palette Color Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics-Client w/8 bit cable</td>
<td>GFX_CM_8BIT</td>
<td>TRUE</td>
<td>1,1</td>
<td>GFX_COLOR_RGB</td>
</tr>
<tr>
<td>Graphics-Client w/16 bit cable</td>
<td>GFX_CM_565, GFX_CM_655, or GFX_CM_556</td>
<td>FALSE</td>
<td>1,1</td>
<td>NA</td>
</tr>
<tr>
<td>No current hardware implementation available</td>
<td>GFX_CM_4BIT</td>
<td>TRUE</td>
<td>1,1</td>
<td>GFX_COLOR_RGB</td>
</tr>
</tbody>
</table>

### Viewport Complexity

The driver supports one active viewport at a time. The application can create multiple viewports and stack them. The viewport must be aligned with, and the same size as the display. Display drawmaps must be the same size as the viewport.

### Memory

Applications are expected to request graphics memory from the driver. The driver allocates memory from the system as needed. It requests this memory from color 0x80. This memory (specified in the init module) is located at the bottom of 16/32 MB DRAM address space and is marked as non cached.
Location

This driver's source is located in:

SRC/DPIO/MFM/DRVR/GX_SA1100

This driver's makefiles are located in:

OS9000/ARMV4/PORTS/ASSABET/MAUI/GX_SA1100

This directory contains the makefiles and descriptor header file to build the descriptor(s) and driver(s) (not all packages include driver source) for the StrongARM reference platform. This directory contains:

- `makefile` Calls each of the other makefiles in this directory
- `drvr.mak` Builds the driver
- `desc.mak` Builds the descriptor(s)
- `mfm_desc.h` Defines values for all modifiable fields of the descriptor(s)

Build the Driver

The driver source is located in SRC/DPIO/MFM/DRVR/GX_SA1100. To build the driver, use the following commands:

```bash
cd OS9000/ARMV4/PORTS/ASSABET/MAUI/GX_SA1100
os9make -f drvr.mak
```

Build the Descriptor

To build a new descriptor, modify `mfm_desc.h`, and use the following commands to compile:

```bash
cd OS9000/ARMV4/PORTS/ASSABET/MAUI/GX_SA1100
os9make -f desc.mak
```

To build both the driver and the descriptor you can specify `os9make` with no parameters.
SD_UCB1200 Sound Driver Specification

This section describes the hardware specifications for the Philips UCB1200 driver sd_ucb1200. The hardware sub-type defines the board configuration. This specification should be used in conjunction with the MAUI Sound Driver Interface.

This driver works in conjunction with the spucb1200 driver.

Device Capabilities

Information about the hardware capabilities is determined by calling _os_gs_snd_devcap(). This function returns a data structure formatted as in the following table. See SND_DEV_CAP for more information about this data structure.

Table B-4  Data Returned in SND_DEV_CAP

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw_type</td>
<td>&quot;CS4231&quot;</td>
<td>&quot;Hardware type&quot;</td>
</tr>
<tr>
<td>hw_subtype</td>
<td>&quot;CS4231A&quot;</td>
<td>&quot;Hardware sub-type&quot;</td>
</tr>
<tr>
<td>sup_triggers</td>
<td>SND_TRIG_ANY</td>
<td>Supported triggers</td>
</tr>
<tr>
<td>play_lines</td>
<td>SND_LINE_SPEAKER</td>
<td>Play gain/mix lines</td>
</tr>
<tr>
<td>record_lines</td>
<td>SND_LINE_MIC</td>
<td>Record gain/mix lines</td>
</tr>
<tr>
<td>sup_gain_cmds</td>
<td>SND_GAIN_CMD_MONO</td>
<td>Mask of supported gain commands</td>
</tr>
<tr>
<td>num_gain_caps</td>
<td>2</td>
<td>Number of SND_GAIN_CAPS</td>
</tr>
<tr>
<td>Member Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>gain_caps</td>
<td>See Gain Capabilities</td>
<td>Pointer to SND_GAIN_CAP array</td>
</tr>
<tr>
<td>num_rates</td>
<td>30</td>
<td>Number of sample rates</td>
</tr>
<tr>
<td>sample_rates</td>
<td>See Sample Rates</td>
<td>Pointer to sample rate array</td>
</tr>
<tr>
<td>num_chan_info</td>
<td>1</td>
<td>Number of channel info entries</td>
</tr>
<tr>
<td>channel_info</td>
<td>See Number of Channels</td>
<td>Pointer to channel info array</td>
</tr>
<tr>
<td>num_cm</td>
<td>3</td>
<td>Number of coding methods</td>
</tr>
<tr>
<td>cm_info</td>
<td>See Encoding and Decoding Formats</td>
<td>Pointer to coding method array</td>
</tr>
</tbody>
</table>
Gain Capabilities Array

The following tables show the various gain capabilities for the Philips UCB1200. This information is pointed to by the gain_cap member of the SND_DEV_CAP data structure. See SND_GAIN_CAP for more information about this data structure. This driver allows control of following individual physical gain controls:

**Table B-5  Individual Gain Controls**

<table>
<thead>
<tr>
<th>SND LINE SPEAKER</th>
<th>Output Attenuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SND LINE MIC</td>
<td>Microphone Gain</td>
</tr>
</tbody>
</table>

The following tables detail the various individual gain capabilities:

**Table B-6  Speaker Gain Enable**

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Value</th>
<th>Step</th>
<th>HW</th>
<th>Level</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines</td>
<td>SND_LINE_SPEAKER</td>
<td>0-3</td>
<td>31</td>
<td>-69 dB</td>
<td>default_level</td>
</tr>
<tr>
<td>sup_mute</td>
<td>TRUE</td>
<td>4-7</td>
<td>30</td>
<td>-66.8 dB</td>
<td></td>
</tr>
<tr>
<td>default_type</td>
<td>SND_GAIN_CMD_MONO</td>
<td>8-11</td>
<td>29</td>
<td>-64.7 dB</td>
<td></td>
</tr>
<tr>
<td>default_level</td>
<td>SND_LEVEL_MAX</td>
<td>12-15</td>
<td>28</td>
<td>-62.5 dB</td>
<td></td>
</tr>
<tr>
<td>zero_level</td>
<td>SND_LEVEL_MIN</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>num_steps</td>
<td>32</td>
<td>112-115</td>
<td>3</td>
<td>-6.5 dB</td>
<td></td>
</tr>
<tr>
<td>step_size</td>
<td>216</td>
<td>116-119</td>
<td>2</td>
<td>-4.3 dB</td>
<td></td>
</tr>
<tr>
<td>mindb</td>
<td>-6900</td>
<td>120-123</td>
<td>1</td>
<td>-2.2 dB</td>
<td></td>
</tr>
<tr>
<td>maxdb</td>
<td>0</td>
<td>124-127</td>
<td>0</td>
<td>0.0 dB</td>
<td>zero_level</td>
</tr>
</tbody>
</table>
## Table B-7 Mic Gain Enable

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Value</th>
<th>Step</th>
<th>HW</th>
<th>Level</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines</td>
<td>SND_LINE_MIC</td>
<td>0-3</td>
<td>0</td>
<td>0 dB</td>
<td>zero_level</td>
</tr>
<tr>
<td>sup_mute</td>
<td>FALSE</td>
<td>4-7</td>
<td>1</td>
<td>0.7 dB</td>
<td></td>
</tr>
<tr>
<td>default_type</td>
<td>SND_GAIN_CMD_MONO</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>default_level</td>
<td>SND_LEVEL_MAX</td>
<td>64-67</td>
<td>16</td>
<td>11.3 dB</td>
<td>default_level</td>
</tr>
<tr>
<td>zero_level</td>
<td>SND_LEVEL_MIN</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>num_steps</td>
<td>32</td>
<td>112-115</td>
<td>20.4 dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>step_size</td>
<td>70</td>
<td>116-119</td>
<td>21.1 dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mindb</td>
<td>0</td>
<td>120-123</td>
<td>21.8 dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxdb</td>
<td>2250</td>
<td>124-127</td>
<td>22.5 dB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sample Rates

Following is an abbreviated list of the supported sample rates for the UCB1200. Below is a formula to derive valid sample rates:

\[ \text{sample rate} = \frac{11981000}{32 \times i}, \text{ where } 8 < i < 128 \]

This information is pointed to by the `sample_rates` member of the `SND_DEV_CAP` data structure.

### Table B-8  Sample Rate (Hz)

<table>
<thead>
<tr>
<th>2948</th>
<th>3941</th>
<th>4926</th>
<th>5942</th>
<th>6933</th>
</tr>
</thead>
<tbody>
<tr>
<td>7966</td>
<td>8914</td>
<td>9852</td>
<td>10697</td>
<td>11700</td>
</tr>
<tr>
<td>12910</td>
<td>13866</td>
<td>14976</td>
<td>15600</td>
<td>17828</td>
</tr>
<tr>
<td>18720</td>
<td>19705</td>
<td>20800</td>
<td>22023</td>
<td>23400</td>
</tr>
<tr>
<td>24960</td>
<td>26743</td>
<td>28800</td>
<td>31200</td>
<td>34036</td>
</tr>
<tr>
<td>37440</td>
<td>41600</td>
<td>46801</td>
<td>53486</td>
<td>62401</td>
</tr>
</tbody>
</table>

Number of Channels

The following table shows the different supported number of channels for the Philips UCB1200. The first entry in the table is the default number of channels. This information is pointed to by the `channel_info` member of the `SND_DEV_CAP` data structure.

### Table B-9  Number of Channels

<table>
<thead>
<tr>
<th>Channels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mono</td>
</tr>
</tbody>
</table>
Encoding and Decoding Formats

The following table shows the supported encoding and decoding formats for the Philips UCB1200. The first entry in the table is the default format. This information is pointed to by the cm_info member of the SND_DEV_CAP data structure.

Table B-10 Encoding and Decoding Formats

<table>
<thead>
<tr>
<th>Coding Method</th>
<th>Sample Size</th>
<th>Boundary Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SND_CM_PCM_ULAW</td>
<td>8</td>
<td>2</td>
<td>8 bit u-Law commanded</td>
</tr>
<tr>
<td>SND_CM_PCM_SLINEAR</td>
<td>16</td>
<td>4</td>
<td>16 bit Linear (two's complement)</td>
</tr>
<tr>
<td>SND_CM_LSBYTE1ST</td>
<td>16</td>
<td>4</td>
<td>little-endian</td>
</tr>
<tr>
<td>SND_CM_PCM_SLINEAR</td>
<td>16</td>
<td>4</td>
<td>16 bit Linear signed (two's complement) big-endian</td>
</tr>
</tbody>
</table>
SPUCB1200 driver for the UCB1200 Codec

This document describes the hardware specifications for the Philips UCB1200 driver. This is an SPF driver.

Capabilities

The UCB1200 is capable of controlling a microphone/speaker, input/output telecommunications lines, resistive style touch screen, and 16 General Purpose Input/Output lines. This driver currently can only control the touch screen, and general purpose input/output lines. The microphone/speaker can be controlled with a MAUI Sound driver called sd_ucb1200. No driver has been written for the telecommunications part of the UCB1200.

Descriptors

Table B-11 lists the UCB1200 descriptors.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucb</td>
<td>UCB1200 Chip Initialization</td>
</tr>
<tr>
<td>ucb_audio</td>
<td>Not Implemented</td>
</tr>
<tr>
<td>ucb_touch</td>
<td>Touch Screen</td>
</tr>
<tr>
<td>ucb_gpio</td>
<td>Control GPIO Lines</td>
</tr>
<tr>
<td>ucb_telecom</td>
<td>Not Implemented</td>
</tr>
</tbody>
</table>
MAUI Driver Descriptions

UCB

Opening the /ucb device will perform basic chip initialization. Normally this is not necessary, unless another driver is written to control part of the UCB1200 functions. This is the case for audio. The MAUI Sound driver *sd_ucb1200* will open /ucb to perform chip initialization. In this way, the MAUI Sound driver play audio and this driver can control the touch screen at the same time.

Audio

This portion of the driver is not implemented since the MAUI Sound driver *sd_ucb1200* already exists. *sd_ucb1200* and this driver can co-exist.

Touch Screen

This portion of the driver controls the touch screen operation. When pressure is applied to the touch screen, a hardware interrupt is raised, and this driver's interrupt service routine will execute. A system state alarm, then, will fire at regular intervals to sample data from the touch screen. When pressure is removed, the alarm stops. This mechanism leaves the UCB1200 in a low power state until the user presses the touch screen. The alarm rate can be controlled in the *ucb_touch* descriptor.
Each sample contains an x, y coordinate as well as pressure information. The data is formatted into a six byte packet as defined in the table below. Each packet contains 10 bits of x, 10 bits of y, and 8 bits of pressure information.

Table B-12  Touch Screen Descriptor Data

<table>
<thead>
<tr>
<th>Byte number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sync code - 0x80</td>
</tr>
<tr>
<td>1</td>
<td>header:</td>
</tr>
<tr>
<td></td>
<td>bit 1: pendown</td>
</tr>
<tr>
<td></td>
<td>bit 2: penup</td>
</tr>
<tr>
<td></td>
<td>bit 3: penmove (may occur with pendown or penup)</td>
</tr>
<tr>
<td>2</td>
<td>bits 0..2: high 3 bits of x</td>
</tr>
<tr>
<td></td>
<td>bits 3..6: high 4 bits of pressure</td>
</tr>
<tr>
<td></td>
<td>bit 7: 0</td>
</tr>
<tr>
<td>3</td>
<td>bits 0..6: low 7 bits of x</td>
</tr>
<tr>
<td></td>
<td>bit 7: 0</td>
</tr>
<tr>
<td>4</td>
<td>bits 0..2: high 3 bits of y</td>
</tr>
<tr>
<td></td>
<td>bits 3..6: low 4 bits of pressure</td>
</tr>
<tr>
<td>5</td>
<td>bits 0..6: low bits of y</td>
</tr>
<tr>
<td></td>
<td>bit 7: 0</td>
</tr>
</tbody>
</table>

GPIO

This section of the driver has basic GPIO line control, where lines 0..9 are connected to a 7 segment display or LED. Each line can be controlled with an _os_write() call. (Refer to the UCBHEX program in the TEST directory.)
**Telecom**

This portion of the driver is not implemented.

**Supporting Modules**

Before this driver can be used, the following modules must be in memory: `spf, sysmbuf, mbinstall`. `mbinstall` must also be run before use.
MP_UCB1200 MAUI Touch screen Protocol Module

This document describes the function of the mp ucbl200 protocol module, as well as a high level discussion of the touch screen driver and calibration application.

Overview

The protocol module converts the driver raw data into a MAUI MSG structure. In this way, applications can remain somewhat ignorant of the details of the hardware since it deals with the MAUI Input layer. In this protocol module, the raw hardware data is converted into screen coordinates. In addition, some data filtering occurs to reduce the amount of erroneous data that the touch screen hardware can produce.

Data Format

The touch screen driver sends a 6 byte packet that contains x, y, and pressure information. The exact format of this packet is described in the spucb1200 driver.

Data Filter

This protocol module filters the data coming from the hardware in an attempt to reduce erroneous data. Two methods are implemented: data point averaging and low pressure point removal. The first method will average the last two points received from the driver. The data point will lag slightly behind the current position, then, but the average will reduce erroneous data points produced by the hardware. The second method throw out data points where the pressure below a certain threshold. It seems that extremely light touches will cause the data to become erratic, although the exact pressure threshold is hardware dependent.
Raw Mode

An application can put this protocol module in a "raw" mode where data points are not filtered, averaged, or converted to screen coordinates. That is, the data from the hardware is passed directly up to the application.

The application can put this protocol module in a "raw" mode by calling: `inp_set_sim_meth(inpdev, RAW_MODE)`. After calibration, the program will need to put the protocol module back in NATIVE mode by calling: `inp_set_sim_meth(inpdev, DEFAULT_SIM_METH)`. There is a sample touch screen Calibration Application in the TOUCH_CAL directory.

When the protocol module is taken out of "raw" mode, it will try to read new calibration data points from the ucb1200.dat data module. After the data is read from the module, it is no longer needed.

cdb.touch

The touch screen can be registered with MAUI by loading the cdb.touch module in memory before any programs using input are started. This will specify the spucb1200 as the driver, cdb.touch as the descriptor, and mp_ucb1200 as the protocol module.
Compile Time Options

Table B-13 shows compile time options used to control the default calibration settings and also the screen size. These options can be specified with a value in the mp_ucb1200 makefile to modify the defaults.

Table B-13  Compile Time Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCREEN_WIDTH</td>
<td>Screen Width in Pixels</td>
</tr>
<tr>
<td>SCREEN_HEIGHT</td>
<td>Screen Weight in Pixels</td>
</tr>
<tr>
<td>DEFAULT_CALIBRATION_X</td>
<td>Left Calibration Hardware Point</td>
</tr>
<tr>
<td>DEFAULT_CALIBRATION_Y</td>
<td>Top Calibration Hardware Point</td>
</tr>
<tr>
<td>DEFAULT_CALIBRATION_WIDTH</td>
<td>Width of Screen In Hardware Points</td>
</tr>
<tr>
<td>DEFAULT_CALIBRATION_HEIGHT</td>
<td>Height of Screen In Hardware Points</td>
</tr>
<tr>
<td>JITTER_THRESHOLD</td>
<td>Minimum Pixel Change Required Before Points are Reported to the Application.</td>
</tr>
<tr>
<td>NUM_PTS</td>
<td>This allows you to choose how many successive data points to average in order to produce less erroneous screen coordinate data to the application. The default is 2, and valid choices are 1, 2, 4, 8, 16.</td>
</tr>
<tr>
<td>MIN_PRESSURE</td>
<td>Any pressure point less than this value will be ignored. This is another way to reduce erroneous data. This represents the 8 bit pressure value we get from the driver. The default is 40.</td>
</tr>
</tbody>
</table>
Calibration Application

There is a sample calibration application located in the 
$(MWOS)/SRC/MAUI/MP/MP_UCB1200/TOUCH_CAL directory. This
application, called touch_cal, will present a text message on the
screen as well as points for the user to press. After the points are
pressed, the protocol module mp_ucb1200 will be updated with the
new calibration information.

Assumptions/Dependencies

1. A Window Manager must be running before this application will
   operate.
2. A font module must be present to run the demo. default.fnt is
   the default module, or you can specify one on the command line.
3. touch_cal will open the first CDB_TYPE_REMOTE device in the
cdb.

Command Line Options

-f=[<outfile>] Specifies the file name of the calibration
information module. This program will write the
calibration information to this file name if it is
specified. The file contains the calibration
information as a data module, thus allowing the
information to be stored on disk, nv RAM, flash,
etc. for use the next time the hardware is
rebooted.

-c This option only works if -f is specified. This
will cause the calibration program to run only if
the filename specified with -f is not present.

-m=<font module> Specifies the font module to use for displaying
the text message on the screen.
**Coordination with Protocol Module**

The protocol module `mp_ucl200` and the touch screen application `touch_cal` work together to provide the calibration functionality. `touch_cal` must first open the touch screen device, and then must set it into Raw Mode. After the user selects each calibration point, `touch_cal` computes the average of them. These averaged hardware points (as well as the screen resolution) are then stored in a data module called `ucb1200.dat`. When the input device is taken out of Raw Mode, the protocol module will link to `ucb1200.dat` and update itself with the new calibration information.

**Compiling**

The makefile for `touch_cal` exists in the `$ (PORT)/MAUI/MP_UCB1200/TOUCH_CAL` directory.