Abstract

This document is the user’s guide for the PSP Bootloader product and include build instructions, shell commands description, environment variables and FFS documentation, etc.

Table of Contents

1 INTRODUCTION ........................................................................................................4
  1.1 Purpose..................................................................................................................4
  1.2 Reference Documents ...........................................................................................4
  1.3 Tools Used ............................................................................................................4
  1.4 Abbreviations........................................................................................................4
2 STARTING UP .............................................................................................................6
  2.1 PSP Bootloader features .......................................................................................6
  2.2 POST in PSPBoot .................................................................................................6
  2.3 PSPBoot Configuration .........................................................................................7
  2.4 Makefile targets ....................................................................................................7
  2.5 Build options.........................................................................................................7
  2.6 Burning Bootloader on Flash ..............................................................................10
    2.6.1 Burning PSPBoot with VisionICE ..............................................................10
    2.6.2 Burning PSPBoot with BDI 2000 ...............................................................11
  2.7 PSPBoot File and Directory Structure ................................................................17
    2.7.1 Directory Structure......................................................................................17
    2.7.2 Files List......................................................................................................17
  2.8 Supported Flash Devices.....................................................................................19
3 BOOTLOADER USAGE AND SHELL COMMANDS ...........................................20
  3.1 Bootloader built-in commands............................................................................21
    3.1.1 tftp...............................................................................................................21
    3.1.2 oclk..............................................................................................................22
    3.1.3 boot .............................................................................................................23
    3.1.4 df .................................................................................................................24
    3.1.5 fmt...............................................................................................................24
    3.1.6 setenv ..........................................................................................................25
    3.1.7 unsetenv ......................................................................................................26
    3.1.8 ls..................................................................................................................26
    3.1.9 cp.................................................................................................................26
    3.1.10 rm ................................................................................................................26
    3.1.11 reboot ..........................................................................................................27
    3.1.12 dm ...............................................................................................................27
    3.1.13 printenv .......................................................................................................27
    3.1.14 cat................................................................................................................28
    3.1.15 help..............................................................................................................28
    3.1.16 defragenv.....................................................................................................28
    3.1.17 ftp................................................................................................................28
    3.1.18 fa .................................................................................................................29
    3.1.19 info ..............................................................................................................30
    3.1.20 version.........................................................................................................30
4 SYSTEM ENVIORNMENT VARIABLES .............................................................31
1 INTRODUCTION

1.1 Purpose

This document discusses the PSP Bootloader Package (referred to as “PSPBoot”). It describes the Makefiles, configurations, directory structure and files, shell commands and usage, system environment variables, Flash File System (FFS), customization of PSPBoot and writing applications for PSPBoot.

1.2 Reference Documents

The following documents are among the reference material used to develop the PSP Bootloader:

- Architecture specific documents for Avalanche I, Avalanche D, Puma S (TNETC4401), Titan (TNETV1050) and Sangam (TNETD73XX) and Apex (TNETV1020) the User Guides for their reference boards.

- Books
  - “See MIPS Run”, Dominic Sweetman, 1999
  - “MIPS R4000 Users Manual”, Joe Heinrich, 1993

1.3 Tools Used

The following tools are used in developing the PSP Bootloader:

- VxWorks for MIPS, Tornado™ 2.1.2 (Wind River Systems)

- MontaVista™ Tool chain for MIPS.

- Vision ICE II with VisionClick™ (Wind River Systems).

1.4 Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasePSP</td>
<td>Base Platform Support Package</td>
</tr>
<tr>
<td>BU</td>
<td>TI Business Unit</td>
</tr>
<tr>
<td>CPMAC</td>
<td>TI proprietary Ethernet Interface</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>ELF</td>
<td>Executable and Linking Format</td>
</tr>
<tr>
<td>FFS</td>
<td>Flash File System</td>
</tr>
<tr>
<td>FMM</td>
<td>Flash Memory Manager</td>
</tr>
<tr>
<td>GZIP</td>
<td>GNU Zip utility</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware Abstraction Layer</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>ICE</td>
<td>In-Circuit Emulator</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>POST</td>
<td>Power On Self Test</td>
</tr>
<tr>
<td>PSBL</td>
<td>Platform Support Package Bootloader</td>
</tr>
<tr>
<td>SoC</td>
<td>System On a Chip</td>
</tr>
<tr>
<td>SRR</td>
<td>System Resident Routines</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>VoB</td>
<td>Versioned Object Base</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
</tbody>
</table>
2 STARTING UP

To use the PSP Bootloader, it should be built for the respective board and then burned on to the Flash. To build from ClearCase™, only one VoB – `psp_boot` needs to be mounted. Following is the high level features as well as the description for configuring and building it. No particular setup of environment for build is required, except for making sure that the tool-chain intended for build is present in the path. No utility is provided for putting the tool-chain in path. Currently two tool-chains are supported: Tornado™ tool-chain (for build on a windows host) and MontaVista™ tool-chain (for build on a Linux host).

2.1 PSP Bootloader features

- Flash File System and Environment variables support
- Enhanced Shell engine with command support for OS load, boot, set/get/unset environment variables, Flash usage, etc.
- ELF as the standard executable file format, with extended support for BIB format
- GZIP decompression utility for ELF files
- Auto-detection of file formats for execution
- Serial/TFTP/FTP download of images
- Distributed application execution from Flash File System or Host machine
- DHCP support for dynamic target configuration
- Application Framework for developing applications on PSPBoot along with sample applications
- PC Application support for control of bootloader from remote host

2.2 POST in PSPBoot

PSPBoot performs minimal POST (Power On Self Test) after it boots up.

As part of POST, the following tests are carried out in order:

- **External SDRAM accessibility Test:** This tests out the accessibility of the external SDRAM.
- **External SDRAM Address Bus Test:** This tests out the address lines [2:18] of the address bus to the External SDRAM.

- **External SDRAM Data Bus Test:** This tests out the data lines [0:31] of the data bus to the External SDRAM.
- **Internal SDRAM accessibility Test:** This tests out the accessibility of the Internal SDRAM.
§ **Internal SDRAM Address Bus Test:** This tests out the address lines [2:18] of the address bus to the Internal SDRAM.
§ **Internal SDRAM Data Bus Test:** This tests out the data lines [0:31] of the data bus to the Internal SDRAM.

### 2.3 PSPBoot Configuration

The PSP Bootloader supports configuration options for building it. These options are defined in `sysconf.h`. Some of the configuration options are board and SoC specific and will not be documented here. Following are the common parameters:

- `CONF_CPU_FREQ`: Boot CPU frequency
- `CONF_SYS_FREQ`: Boot System frequency
- `CONF_SDRAM_SZ`: SDRAM size
- `CONF_FLASH_SZ`: FLASH size
- `CONF_CACHE`: CACHE configuration (Write Back, Write through or None)

### 2.4 Makefile targets

- **boot**: This builds the bootloader images, `psbl.rec` and `psbl.elf`, to be used for burning to the flash at location 0xb000'0000. The image is placed in `/bin` directory. `boot` option is the default rule for make.

- **all**: Same as `boot`

- **clean**: clean the Build. Removes all of the .o's and .a's.

The Bootloader Makefile is placed in the top-level directory.

### 2.5 Build options

The Build option can be provided through `bootcfg.mak` file placed in the top level directory. Following are the build options that should be supplied while invoking the makefile. The values of any of these options provided in `bootcfg.mak` can be overridden by providing another option at the command line.

- **BOARD=<BOARD NAME>**

  This option defines the reference board type. For the latest list of supported boards, please refer to the `bootcfg.mak` file of the latest release. The Platforms supported under WLAN Board type are TNETWA113VG, TNETWA622, PCI-EVM and TNETWA123VAG

- **ENDIAN=<LE | BE>**

  This option defines Endian type. LE for Little Endian and BE for Big Endian.
This option defines protocol to be used for file transfer over the network. Choose FTP for File Transfer Protocol, TFTP for Trivial File Transfer Protocol and NONE for no networking support. Please note that FTP can be used only in the absence of Flash File System (see option FFS).

-DHCP=<YES | NO>
This option defines if DHCP Support be included in or not.

-PC_APP=<YES | NO>
This option defined if PC Application support is included or not. PC application can only be compiled in the Little Endian Mode. PC application cannot be used combined with FTP or TFTP support.

-FFS=<YES | NO>
This option defines whether to use TI proprietary Flash File System or not.

-PLATFORM=<WINDOWS | LINUX>
This option defines the tool-chain using which the build is taking place. Use WINDOWS for building using the Tornado™ tool-chain installed on a Windows™ host and LINUX for MontaVista™ tool-chain.

-CONF_OMOD_GZIP=<YES | NO>
This option defines if GZIP decompression support is to be built in or not.

-CONF_OMOD_ELF=<YES | NO>
This option defines if ELF file format support is to be built in or not.

-CONF_OMOD_TIBINARY=<YES | NO>
This option defines if TI Proprietary Binary File Format support is to be built in or not.

-CONF_OCMD_REBOOT=<YES | NO>
This option defines if the 'reboot' shell command is to be provided or not.

-CONF_OCMD_VERSION=<YES | NO>
This option defines if the 'version' shell command is to be provided or not.

-CONF_OCMD_INFO=<YES | NO>
This option defines if the 'info' shell command is to be provided or not.

-CONF_OCMD_FA=<YES | NO>
This option defines if the 'fa' shell command is to be provided or not.

-CONF_OCMD_PRINTENV=<YES | NO>
This option defines if the 'printenv' shell command is to be provided or not.

-CONF_OCMD_SETENV=<YES | NO>
This option defines if the 'setenv' shell command is to be provided or not.

-CONF_OCMD_UNSETENV=<YES | NO>
This option defines if the `unsetenv` shell command is to be provided or not.

- `CONF_OCMD_DEFRAGENV=<YES | NO>`
  This option defines if the `defragenv` shell command is to be provided or not.

- `CONF_OCMD_FMT=<YES | NO>`
  This option defines if the `fmt` shell command is to be provided or not.

- `CONF_OCMD_BOOT=<YES | NO>`
  This option defines if the `boot` shell command is to be provided or not.

- `CONF_OCMD_DM=<YES | NO>`
  This option defines if the `dm` shell command is to be provided or not.

- `CONF_OCMD_OCLK=<YES | NO>`
  This option defines if the `oclk` shell command is to be provided or not.

- `CONF_OCMD_HELP=<YES | NO>`
  This option defines if the `help` shell command is to be provided or not.

- `CONF_OCMD_LS=<YES | NO>`
  This option defines if the `ls` shell command is to be provided or not. Without FFS support this option is ignored.

- `CONF_OCMD_DF=<YES | NO>`
  This option defines if the `df` shell command is to be provided or not. Without FFS support this option is ignored.

- `CONF_OCMD_CP=<YES | NO>`
  This option defines if the `cp` shell command is to be provided or not. Without FFS support this option is ignored.

- `CONF_OCMD_CAT=<YES | NO>`
  This option defines if the `cat` shell command is to be provided or not. Without FFS support this option is ignored.

- `CONF_OCMD_RM=<YES | NO>`
  This option defines if the `rm` shell command is to be provided or not. Without FFS support this option is ignored.

- `CONF_OCMD_TFTP=<YES | NO>`
  This option defines if the `tftp` shell command is to be provided or not. Without TFTP support this option is ignored.

- `CONF_OCMD_FTP=<YES | NO>`
  This option defines if the `ftp` shell command is to be provided or not. Without FTP support this option is ignored.
After building the bootloader image, it is burned onto the Flash at a base address of 0xb000'0000 using VisionCLICK. Refer to the next section ‘Burning BootLoader on Flash’ for details.

2.6 Burning Bootloader on Flash

2.6.1 Burning PSPBoot with VisionICE

The following procedure should be used for burning the bootloader on flash using VisionICE.

Step 1 – Convert the bootloader S-RECORD file to the visionCLICK binary file format:

In order to load a file into flash memory, visionCLICK™ requires that the file be in visionCLICK™ BIN format. Typically, we do not include this file format in the bootloader software distribution, but we do provide S-Record format versions of the bootloader kernel. Therefore, the first step is to convert the bootloader kernel from S-Record format to the visionCLICK™ BIN format. Note that this step is not required if you already have the visionCLICK™ BIN format version of the bootloader kernel for your platform.

1. From the visionCLICK™ menu, click on File, and then Open Project Files / Load Options Dialog. The PROJECTS / LOAD dialog box is displayed.
2. Click the Load Options tab, then click the Convert… button
3. In the CONVERT BINARY AND SYMBOL OBJs dialog box:
   - Enter the filename of the bootloader kernel file you want to convert in the Select Input Object Module To Convert field. Bootloader S-Record files have a filename extension of .rec
   - Mark the check box to Create Flat BIN File For Flash Programming and input the range of 90000000 to 9003FFFF. Clear all other check boxes. Enter the destination filename where you wish the resulting visionCLICK™ BIN format file to be placed.
   - Enter –s in the Miscellaneous Parameters field. This specifies that the input file be in S-Record format.
   - Click the Convert button to perform the conversion. An MS-DOS window will display the results of the conversion process. Press any key to close the MS-DOS window.
   - Click the OK button to close the CONVERT BINARY AND SYMBOL OBJs dialog box.

Step 2 – Write the converted BIN file to your platform’s flash memory

1. After performing Step 1, the Load Options tab of the PROJECTS / LOAD dialog box should be displayed. If not, then from the visionCLICK menu, click on File, and then Open Project Files / Load Options Dialog. Then click the Load Options tab.
2. Click the Flash Setup… button
3. In the TF FLASH PROGRAMMING dialog box…

Enter the filename of the visionCLICK™ BIN format file created in Step 1 above in the Flash Card or PC Host File Name and Path (0 Bias) field.
Select the appropriate flash device used on your platform in the Programming Algorithm field. See the table below.
Enter B0000000 in the Base Address field, and click the Erase All radio button. The whole flash should be erased for the first time only.
Enter A0000000 in the Start Address field.
Click the Erase and Program button. The specified BIN file will be loaded into flash.
Reset your platform to execute the newly loaded flash code.

The following table gives the VisionICE flash settings for currently supported Reference Boards.

<table>
<thead>
<tr>
<th>Reference Board</th>
<th>Flash Device Choice for VisionClick™</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR7VDB</td>
<td>INTEL 28F128Jx (8192 x 16) 1 Device</td>
</tr>
<tr>
<td>AR7DB / AR7RD / AR7WRD / AR7VWi / AR7L0 / AR7Wi</td>
<td>AMD 29DL32xxB(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETV1050VDB</td>
<td>AMD 29DL32xxB(2048 x 16) 2 Devices</td>
</tr>
<tr>
<td>TNETV1050SDB</td>
<td>INTEL 28F320C3T(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC401B</td>
<td>INTEL 28F320Jx (2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC620</td>
<td>INTEL 28F320C3B(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC621</td>
<td>INTEL 28F320C3B(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>WLAN</td>
<td>INTEL 28F320C3B(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC520</td>
<td>INTEL 28F320C3B(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC420</td>
<td>INTEL 28F160C3B(2048 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETC405T</td>
<td>INTEL 28F160C3T(1024 x 16) 1 Device</td>
</tr>
<tr>
<td>TNETV1020VDB</td>
<td>INTEL 28F128Jx (8192 x 16) 1 Device</td>
</tr>
</tbody>
</table>

2.6.2 Burning PSPBoot with BDI 2000

2.6.2.1 Installing BDI2000 Software.

Unzip the Host software provided with BDI2000 onto the local disk. This installs the BDI2000 software on the Host PC.

2.6.2.2 Configuring BDI2000 for MIPS

BDI2000 has to be configured for running with MIPS processor. To configure BDI2000 for MIPS, please follow the instructions given in Section 2.5 - Installation of the Configuration Software from the BDI2000 user manual.

2.6.2.3 Establishing EJTAG connection to BDI2000
1. Connect the BDI2000 Emulator to DUT using the EJTAG port.  
   **Warning:** Pay special attention to the EJTAG connector pin orientation. Failing to connect with proper orientation may lead to damage of the DUT or BDI2000 or both.

2. Use a Null Modem cable to establish a serial connection between Host PC and the BDI2000.

3. Connect the BDI2000 to the Local network using Ethernet Cable.  
   **Warning:** Power up sequence between BDI2000 and DUT is important. On power-up of the system, first power-up the BDI only then the DUT. On system power-down, first power-down the DUT and only then the BDI2000.

4. Run the executable “B20R4KGD.EXE” provided with the BDI Software.  
   From the GUI menu, select **Setup → BDI2000...**  
   By default, BDI2000 comes up with a baud rate of 57600.

   ![BDI2000 Update/Setup](image)

   Fill up the fields under “**Connect BDI2000 Loader**” and “**Configuration**”. The configuration file to be provided is a board specific configuration file. The details of the configuration file will be discussed later. At this stage, only name of the configuration file is taken by BDI2000.
Now press the connect button. On success, you should see the SN field updated. The Transmit button is also enabled. Press the Transmit Button now. All the Data entered is saved with BDI2000. Once the data is updated to BDI2000, there is no further need of the serial connection. Power Cycle the BDI2000 now.

5. After Reboot BDI uses TFTP the protocol to download configuration files from the Host PC. The configuration files accessed are reg4kc.def (provided with the installation software) and the board specific configuration file configured in the previous step. After successfully downloading these files, BDI2000 initializes the target.

6. Connect to BDI2000 by using telnet. BDI prompt should be available. Typing help at this prompt gives a list of supported commands along with a brief description.

2.6.2.4 Erasing and programming the flash using BDI2000

Use the command erase at the BDI telnet prompt to erase the flash. The command erase <address> erases sector starting with that address. erase called without arguments takes the sectors to be erased from the board configuration file.

Use the command prog to program psbl.bin into flash. The name of the file to be programmed and the address to start the program, can be configured using the configuration file.

2.6.2.5 Using the board specific configuration file

A brief description of various sections of the configuration file is given here. For more detailed explanation, please refer to the BDI2000 user manual.

[INIT]

Optional section. Use this to perform any specific initializations. Examples: EMIF initialization, TLB initialization, invalidating data and instruction cache or unlocking of flash blocks for Intel flash chips. See example below.

[TARGET]

1. JTAGCLOCK
   Configures the JTAG clock rate that BDI2000 uses when communicating with the target CPU.

2. CPUTYPE
   Set to M4KC.

3. ENDIAN
   Defines whether the target CPU is in Big or Little Endian mode.

4. STARTUP
Set to RESET.

[HOST]

1. **IP**
The IP address of the Host PC.

2. **PROMPT**
The Telnet prompt string.

[FLASH]

1. **WORKSPACE**
The flash programming workspace. This should be pointed to internal RAM (0xA0000000)

2. **CHIPTYPE**
This parameter defines the type of flash used. It is used to select the correct programming algorithm.
Supported formats: AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16

3. **CHIPSIZE**
The size of one flash chip in bytes

4. **BUSWIDTH**
The width of the memory bus that leads to the flash chips.

3. **FILE**
The name of the file to be used for flash programming (psbl.bin)

4. **FORMAT**
The format of the image file and an optional load address offset. For psbl.bin the format is BIN. This entry also defines the address where the file is programmed into the flash.

5. **ERASE**
The flash memory may be individually erased via the Telnet interface. In order to make erasing of multiple flash sectors easier, you can enter an erase list. All entries in the erase list will be processed if you enter erase at the Telnet prompt without any parameter.

Example:

```
erase 0xb0000000 ; erase sector 0 of flash
erase 0xb0010000 ; erase sector 1 of flash
```

**NOTE:** Some Intel flash chips (e.g. 28F800C3, 28F160C3, 28F320C3) power-up with all blocks in locked state. In order to erase/program those flash chips, use the init list to unlock the appropriate blocks.

```
WM16 0xB0000000 0x0060 ; unlock block 0
WM16 0xB0060000 0x00D0
WM16 0xB0100000 0x0060 ; unlock block 1
WM16 0xB0160000 0x00D0
WM16 0xB0000000 0xFFFF ; select read mode
```
### 2.6.2.6 Flash settings for currently supported boards

Following table defines the various settings for currently supported flash types.

<table>
<thead>
<tr>
<th>Reference Board</th>
<th>Chip Type</th>
<th>Chip Size</th>
<th>Bus Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR7VDB</td>
<td>AM29BX16</td>
<td>0x400000</td>
<td>16</td>
</tr>
<tr>
<td>AR7DB / AR7RD / AR7Wi / AR7VWi / AR7WRD / AR7L0</td>
<td>AM29BX16</td>
<td>0x400000</td>
<td>16</td>
</tr>
<tr>
<td>TNETV1050VDB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETV1050SDB</td>
<td>I28BX16</td>
<td>0x1000000</td>
<td>32</td>
</tr>
<tr>
<td>TNETC401B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETC620</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETC621</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLAN</td>
<td>I28BX16</td>
<td>0x200000</td>
<td>16</td>
</tr>
<tr>
<td>TNETC520</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETC420</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETC405T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNETV1020VDB</td>
<td>AM29BX16</td>
<td>0x1000000</td>
<td>16</td>
</tr>
</tbody>
</table>
2.7 PSPBoot File and Directory Structure

2.7.1 Directory Structure

<table>
<thead>
<tr>
<th>Directory Path/Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>psp_boot</td>
<td>PSPBoot directory</td>
</tr>
<tr>
<td>psp_boot\apps</td>
<td>PSPBoot applications directory</td>
</tr>
<tr>
<td>psp_boot\bin</td>
<td>PSPBoot binary images</td>
</tr>
<tr>
<td>psp_boot\lib</td>
<td>PSPBoot libraries</td>
</tr>
<tr>
<td>psp_boot\inc</td>
<td>PSPBoot headers directory</td>
</tr>
<tr>
<td>psp_boot\inc\psbl</td>
<td>PSPBoot headers</td>
</tr>
<tr>
<td>psp_boot\inc\apps</td>
<td>PSPBoot application headers</td>
</tr>
<tr>
<td>psp_boot\psbl</td>
<td>PSPBoot sources</td>
</tr>
<tr>
<td>psp_boot\psbl\gzip</td>
<td>PSPBoot compression sources</td>
</tr>
<tr>
<td>psp_boot\psbl\kernel</td>
<td>PSPBoot kernel sources</td>
</tr>
<tr>
<td>Psp_boot\psbl\kernel\cmd</td>
<td>PSPBoot command sources</td>
</tr>
<tr>
<td>psp_boot\psbl\net</td>
<td>PSPBoot network sources</td>
</tr>
<tr>
<td>psp_boot\tools</td>
<td>PSPBoot tools/utilities</td>
</tr>
<tr>
<td>Psp_boot\docs</td>
<td>PSPBoot Documentation</td>
</tr>
<tr>
<td>Psp_boot\export</td>
<td>PSPBoot files exported for environment and FFS support.</td>
</tr>
</tbody>
</table>

2.7.2 Files List

<table>
<thead>
<tr>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>PSP Bootloader Makefile</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>Bootloader howto</td>
</tr>
<tr>
<td>rules.mak</td>
<td>Makefile rules for inclusion in sub-directory makefiles</td>
</tr>
<tr>
<td>psbl.xn</td>
<td>Bootloader linker Script</td>
</tr>
<tr>
<td>debug.h</td>
<td>Debug Support Header File</td>
</tr>
<tr>
<td>elf.h</td>
<td>ELF header file</td>
</tr>
<tr>
<td>env.h</td>
<td>Environment Variables Header File</td>
</tr>
<tr>
<td>fmm.h</td>
<td>Flash Memory Manager Header File</td>
</tr>
<tr>
<td>errno.h</td>
<td>Error codes</td>
</tr>
<tr>
<td>ffs.h</td>
<td>Flash File System Header File</td>
</tr>
<tr>
<td>flashop.h</td>
<td>Flash HAL Header File</td>
</tr>
<tr>
<td>heapmgr.h</td>
<td>Heap Manager Header File</td>
</tr>
<tr>
<td>hw.h</td>
<td>Hardware definitions</td>
</tr>
<tr>
<td>mips.h</td>
<td>MIPS processor Definitions</td>
</tr>
<tr>
<td>revision.h</td>
<td>Revision Logs</td>
</tr>
<tr>
<td>sysconf.h</td>
<td>Buildable Configuration Parameters</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>Stdarg header file</td>
</tr>
<tr>
<td>stdio.h</td>
<td>Standard I/O Include File</td>
</tr>
<tr>
<td>stddef.h</td>
<td>Standard Definitions and Typedefs</td>
</tr>
<tr>
<td>system.h</td>
<td>System Header File</td>
</tr>
<tr>
<td>crc32.c</td>
<td>Gzip CRC file</td>
</tr>
<tr>
<td>/psbl/gzip/*.[ch]</td>
<td>Gzip Header files</td>
</tr>
<tr>
<td>mips4k_init.S</td>
<td>Processor initialization routines</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>platform_init.S</td>
<td>SoC initialization routines</td>
</tr>
<tr>
<td>srt0.S</td>
<td>System startup routines</td>
</tr>
<tr>
<td>avreset.h</td>
<td>SoC specific header file</td>
</tr>
<tr>
<td>fcb.h</td>
<td>FCB header file</td>
</tr>
<tr>
<td>ffs_extra.h</td>
<td>FFS extra routines header file</td>
</tr>
<tr>
<td>ffs_util.h</td>
<td>FFS Utilities Header File</td>
</tr>
<tr>
<td>files.h</td>
<td>Files abstraction header file</td>
</tr>
<tr>
<td>gdb_stub.h</td>
<td>GDBL debug header file</td>
</tr>
<tr>
<td>load_bin.h</td>
<td>Binary file load header file</td>
</tr>
<tr>
<td>load_elf.h</td>
<td>Elf file load header file</td>
</tr>
<tr>
<td>loadfile.h</td>
<td>File loading header file</td>
</tr>
<tr>
<td>main.h</td>
<td>Main.c specific header file</td>
</tr>
<tr>
<td>mms.h</td>
<td>Dynamic Memory Management Routines Header File</td>
</tr>
<tr>
<td>shell.h</td>
<td>Shell header file</td>
</tr>
<tr>
<td>sio.h</td>
<td>Serial interface header file</td>
</tr>
<tr>
<td>support.h</td>
<td>Ctypes header file</td>
</tr>
<tr>
<td>sysdefs.h</td>
<td>System Definitions Header File</td>
</tr>
<tr>
<td>env.c</td>
<td>Environment Variable Routines Source File</td>
</tr>
<tr>
<td>fmm.c</td>
<td>Flash Memory Manager Source File</td>
</tr>
<tr>
<td>exception.c</td>
<td>Exceptions support Source File</td>
</tr>
<tr>
<td>fcb.c</td>
<td>FCB source file</td>
</tr>
<tr>
<td>ffs.c</td>
<td>FFS Routines Source File</td>
</tr>
<tr>
<td>ffs_extra.c</td>
<td>Extra FFS routines Source File</td>
</tr>
<tr>
<td>ffs_util.c</td>
<td>FFS Utilities Source File</td>
</tr>
<tr>
<td>files.c</td>
<td>Files abstraction Source File</td>
</tr>
<tr>
<td>flashop_istrata_16.c</td>
<td>Flash HAL (Intel Strata 16-bit Flash) Source file</td>
</tr>
<tr>
<td>flashop_istrata_32.c</td>
<td>Flash HAL (Intel Strata 32-bit Flash) Source file</td>
</tr>
<tr>
<td>flashop_amd_16.c</td>
<td>Flash HAL (AMD 16-bit Flash) Source file</td>
</tr>
<tr>
<td>heapmgr.c</td>
<td>Heap Manager Source File</td>
</tr>
<tr>
<td>load_bin.c</td>
<td>Binary load Source File</td>
</tr>
<tr>
<td>load_elf.c</td>
<td>Elf load Source File</td>
</tr>
<tr>
<td>loadfile.c</td>
<td>File loading Source File</td>
</tr>
<tr>
<td>main.c</td>
<td>C startup routines Source File</td>
</tr>
<tr>
<td>mms.c</td>
<td>Dynamic Memory Management Routines Source File</td>
</tr>
<tr>
<td>printf.c</td>
<td>Printf Source File</td>
</tr>
<tr>
<td>shell.c</td>
<td>Shell support Source File</td>
</tr>
<tr>
<td>sio.c</td>
<td>Serial Interface Source File</td>
</tr>
<tr>
<td>support.c</td>
<td>Ctypes Source File</td>
</tr>
<tr>
<td>i2c_defs.h</td>
<td>I2C defines</td>
</tr>
<tr>
<td>i2c_hal.c</td>
<td>I2C HAL sources</td>
</tr>
<tr>
<td>i2c_hal.h</td>
<td>I2C HAL header file</td>
</tr>
<tr>
<td>cpmac.c</td>
<td>CPMAC HAL source</td>
</tr>
<tr>
<td>cpmac.h</td>
<td>CPMAC HAL header</td>
</tr>
<tr>
<td>arp.c</td>
<td>ARP source</td>
</tr>
<tr>
<td>arp.h</td>
<td>ARP header</td>
</tr>
<tr>
<td>ether.c</td>
<td>Ethernet layer packet handling source</td>
</tr>
<tr>
<td>ether.h</td>
<td>Ethernet layer header</td>
</tr>
<tr>
<td>ip.c</td>
<td>IP layer routine source</td>
</tr>
<tr>
<td>ip.h</td>
<td>Ip layer header</td>
</tr>
</tbody>
</table>
2.8 Supported Flash Devices

These are the Flash devices currently supported on PSPBoot. (Found on various reference boards).

<table>
<thead>
<tr>
<th>Flash Type</th>
<th>Part Number</th>
<th>Flash HAL file</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Flash</td>
<td>AMD AM29LV320D</td>
<td>flashop_amd_16.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>E28F320-J3A1110</td>
<td>flashop_istrata_16.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>E28F128-J3C150</td>
<td>flashop_istrata_16.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>DA28F640J5-150</td>
<td>flashop_istrata_32.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>E28F160-C3TA110</td>
<td>flash_is1050_16.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>E28F320-C3TA110</td>
<td>flash_is1050_16.c</td>
</tr>
<tr>
<td>Intel Strata Flash</td>
<td>E28F640-C3TC80</td>
<td>flash_is1050_16.c</td>
</tr>
<tr>
<td>Atmel Flash</td>
<td>AT49LV320-90TI</td>
<td>flashop_atmel_32.c</td>
</tr>
<tr>
<td>Intel Advanced+ C3</td>
<td>TE28F320C3TC90</td>
<td>flashop_ic3.c</td>
</tr>
</tbody>
</table>
3  BOOTLOADER USAGE AND SHELL COMMANDS

The PSP Bootloader is to be used with the sead1.exe application running on host as a serial terminal for the emulation board. This application supports file download/upload from/to the host to the target board via the serial interface. The sead1.exe terminal defaults to 38400 baud with hardware flow control and port 1. To specify a baud rate, for e.g. 9600, with no hardware flow control, and connect to serial port 2, run from the DOS as:

```
Z:\psp_boot\bin> sead1.exe -b 9600 -f -p 2
```

After reset, if the BOOTCFG environment variable is set, the bootloader comes up and tries to load the default OS boot image. It waits for 3 seconds for the user to break to the shell prompt, if desired. For doing so, user should hit the ESC key when prompted. While booting the OS image by default, if the image is not found, it falls to the shell. If the BOOTCFG environment variable is not set, the bootloader automatically falls to shell. For detailed description on the usage of BOOTCFG environment variable, please refer to the description of ‘boot’ command.

Following is the screen shot for jumping to the shell prompt:

```
PSPBoot1.1 rev: 1.0.2
(c) Copyright 2002-2003 Texas Instruments, Inc. All Rights Reserved.

Press ESC for monitor... 1   # Press ESC key for shell
(psbl)                        # Bootloader prompt
```

The shell prompt is available for commands execution. The shell buffer size is 150 bytes.

Following in this section is a list of commands that are built-in. Applications can also be built for running with the shell. These application execution is distributed and can reside either on the host machine in the directory where from sead1.exe is running, or can be copied to the FFS in the /bin directory. The order of execution for PSPBoot is:

```
Builtin  - bootloader built-in commands
/bin     - applications in FFS, dir /bin
/ttyS    - applications on Host, using serial interface
```

The path for execution of files on FFS can be changed using the PATH environment variable. The built-in commands are always executed first. They cannot be overridden by executables placed in the FFS. The FFS is installed in the flash by use of the ‘fmt’ command.

Applications for execution from the shell should be in either of ELF format, Gzipped ELF format or TI binary format.
3.1 Bootloader built-in commands

Following is the list of Bootloader built-in shell commands with their descriptions and usage. At the end is provided a sample usage for booting an OS image.

3.1.1 tftp

Usage:
tftp  [-i  <server-ip>]  <srcfile>  <dstfile>
tftp  [-r]  <srcfile>  [optional arguments]

Description:
This command downloads the `srcfile` from a given host to `dstfile` on FFS, over the network using TFTP protocol. The host should be running TFTP server to support the file download. This command can also be used to download an executable file directly to RAM and start its execution. If `dstfile` is missing TFTP to RAM is assumed. If arguments need be passed to the executable, `-r` option should be used. Note that files non-executable for the platform cannot be downloaded to RAM.

This tftp module depend on environment variables for relevant network related information. They are:

- HWA_[0|1] : System MAC address.
- IPA : System IP address.
- IPA_SVR: Target (or TFTP server) IP address, running tftp server.
- IPA_GATEWAY: The IP Address of the Gateway for the subnet in which the target is present. This parameter is not required if IPA and IPA_SVR belong to the same subnet.
- SUBNET_MASK: The Subnet Mask for the subnet in which the target is present.
- MAC_PORT: The active MAC port number. (Valid values 0 or 1).
- TFTPCFG: (Optional) Configuration of Re-transmissions count and time-out time in seconds.

The server IP address, configured in the environment variable can be overridden if the server-ip is given in the command line.

The TFTP server should be configured to transfer data packets of 512 byte-blocks.

NOTE:
- This command doesn’t support file upload to the server.

[-i <server-ip>]
Host IP, running tftp server. This option overrides the IPA_SVR environment variable.

[-r]
Download `srcfile` directly to RAM. Any following arguments ([Optional arguments]) are considered as arguments to `srcfile`.

<srcfile>
File to retrieve from the tftp server.
If FFS is built in, this should be a file name with path, to copy *srcfile* to.
In case FFS is not built in, this should be an environment variable of the format base_address,end_address. (both addresses in Flash) (eg 0x90200000,0x90300000) In this case, *srcfile* will be copied to the raw flash, starting at the base address specified.

See Also:
*cp*

### 3.1.2 oclk

Usage: 

```
oclk –d
oclk <[-s sys_freq] [cpu_freq]>
```

Note: *sys_freq* and *cpu_freq* are specified in MHz units.

Description:
This command supports run-time configuration of system and CPU frequencies.

For the SYNC mode, either of the CPU or System frequency can be provided. In this mode, only the System frequency is configured.

`[-d]`

Dump the current configuration of both CPU and System frequencies.

`[-s <sys_freq>]`

System frequency for configuration. If provided value is out of valid range, it is configured to the closest valid frequency. A value of zero leads to the System frequency value remaining unchanged.

`[cpu_freq]`

CPU frequency for configuration. If provided value is out of valid range, it is configured to the closest frequency. A value of zero leads to the CPU frequency value remaining unchanged.

Important Note:
1. The oclk command sets the CPU and System Frequency only in steps of 12.5 MHz starting from the minimum value and going upto the maximum permissible value.
2. oclk command does not accept floating point values as input for system and CPU frequency.
3. For a given input of System or CPU frequency, the nearest multiple of 12.5 MHz *lower* to the asked frequency is set. For example, To set the CPU frequency to 32.5 Mhz, use:

   (psbl) oclk 33

<table>
<thead>
<tr>
<th>SoC Name</th>
<th>Synchronous Mode*: Min Freq (MHz)</th>
<th>Synchronous Mode*: Max Freq (MHz)</th>
<th>Asynchronous Mode*: Min Freq (MHz)</th>
<th>Asynchronous Mode*: Max Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
* In TNETV1020, Synchronous mode denotes MIPS-2-TO-1 mode.

See Also:
sysconf.h for default boot frequency.

3.1.3 boot

Usage: boot

Description:
Boot up an OS image from the FFS/Network. The boot image and the order of boot is configurable with the environment variable, BOOTCFG. BOOTCFG has the following format

<method of getting configuration>:<order of boot>:<boot-image>

The valid values are:

<m|d>:<f|[n]>:<a|“bootfile”>

‘m’ stands for manual configuration. In this case DHCP will not be invoked. All the configuration must be made manually.

‘d’ stands for DHCP configuration. All valid information that DHCP server provides will be taken.

‘f’ stands for execute image stored in Flash

‘n’ stands for boot from network using TFTP

‘a’ stands for auto boot-file configuration ie. Let the DHCP server provide the filename to boot. This option is invalid if ‘m’ is selected. The boot-file provided by DHCP server can be over-ridden by providing an alternate filename in double-quotes. In case of manual configuration, provision of bootimage name is must.

Boot Order

f : boot from flash only

n : Attempt to download the boot-image by TFTP and boot.

fn : Attempt to boot from the flash first, if it fails, attempt to download the
bootimage from network to flash and boot from flash.

nf : Attempt to download the boot-image from network to RAM and boot, if it fails, boot from flash. No update of flash is done.

Boot File

a : Use the boot-image name provided by DHCP server. If FFS is used, the boot-image name is searched in /bin directory. If FFS is not used, boot-image name will be treated as an environment variable of the type start_address,end_address. Attempt will be made to boot by reading the image from flash starting at start_address. If manual configuration is used, this option is invalid and will result in boot failure.

“filename”: If FFS is used, the filename is searched in /bin directory. If FFS is not used, filename will be treated as an environment variable of the type start_address,end_address. Attempt will be made to boot by reading the image from flash starting at start_address. If DHCP is used, this option can be used to override the filename provided by DHCP server. If manual configuration is used, this option is must.

See Also:
setenv

3.1.4 df

Usage: df

Description:
Display raw flash device usage statistics by FFS. This command is not supported if FFS is not present.

See Also:
ls

3.1.5 fmt

Usage: fmt [-a start_address,end_address] [Environment Variable]

Description:
If FFS is present, this command with no arguments, formats the FFS. Following a format, it creates the directories, /bin, /etc and /ttyS. All of the previously existing files will be erased.

Usage of this command with no arguments in absence of FFS is invalid.
fmt can be called with –a option, in which it takes a start address and end address and erases the flash area between those addresses. (End address excluded in case of block aligned addresses) Eg

fmt -a 0x90200000,0x90300000

The same effect can be achieved by the use of an environment variable of the format start_address,end_address. (both addresses in Flash) (eg 0x90200000,0x90300000). In this case fmt will erase flash area starting at the start_address till the end_address. Eg.

(psbl) setenv MTD0 0x90200000,0x90300000
(psbl) fmt MTD0

See Also:
ls, setenv

3.1.6 setenv
Usage: setenv <ENV NAME> <env value>

Description:
Set/Update a system environment variable. The environment variables can be predefined and listed in env.c. (placed in psp_boot\psbl\kernel). Environment variables can also be created dynamically. Please note that usage of dynamic environment variables will lead to wastage of environment space as both the variable and the value string will be stored on the flash. It is suggested that dynamic variables be used only for debugging/development purposes. Once the system is ready for deployment, pre-defined variables should preferably be used.

Please note that both the environment variable and value are case-sensitive.

The command setenv will automatically de-fragment the environment space if the environment space is exhausted and if there is garbage left to be collected. This de-fragmentation is however, not power-fail safe. Setting the macro AUTO_DEFrag_ENVIRONMENT in env.h to FALSE will turn off the auto de-fragmentation

<ENV NAME>
  Defines the name of the new Environment Variable.

<env value>
  Defines the value of the new environment variable.

See Also:
unsetenv, printenv, defragenv
3.1.7 unsetenv

Usage: unsetenv <ENV NAME>

Description:
Delete an environment variable.

<ENV NAME>
Environment Variable name.

See Also:
setenv, printenv

3.1.8 ls

Usage: ls

Description:
List information about the files/directories in the FFS. In case FFS is not used, this command is not supported.

3.1.9 cp

Usage: cp <srcfile> <dstfile>

Description:
This command copies a given file, `srcfile', to `dstfile'. Both of the file arguments should be provided with appropriate path. `cp' can copy files from FFS to the Serial File System and vice-versa. This command can also be used to copy files within the Flash File System.

<srcfile>
Source filename, to copy from.

dstfile>
Destination filename, to copy source file to.

In case FFS is not used, this command is not supported.

Example:
(psbl) cp /ttyS/app.bin /bin/app

See Also:
tftp

3.1.10 rm

Usage: rm <file>
Description:
This command removes a file from the Flash File System. Only the file node is freed from FFS. The actual space occupied by the file being removed is not restored to FFS. For restoring the space to FFS, de-fragmentation of FFS should be run. In case FFS is not used, this command is not supported.

=file=
File with the whole path, to be removed.

Example:
(psbl) rm /bin/app

See Also:
<file>

3.1.11 reboot
Usage: reboot

Description:
This command reboots the whole system and brings it up as during a normal power-up reset.

3.1.12 dm
Usage: dm [<hex address> [num words]]

Description:
Display memory contents from the defined location. If the parameter num words is not given, it dumps 128 bytes form the given address. The user can break the memory dump and return to prompt at any time while the dump is taking place by pressing the ESC key.

Issuing the command without any parameters will display the memory starting at the last displayed address. If there is no last displayed address, then this command will dump the memory starting at 0xb4000000.

hex address
Starting Address, in hex, to display the memory contents from.

num words
Defines the number of words of memory to be dumped.

3.1.13 printenv
Usage: printenv
printenv envlist
3.1.14 cat

Usage: cat <file>

Description:
Print file to the shell, in ASCII. In case FFS is not used, this command is not supported.

Example:
(psbl) cat /etc/config.txt

3.1.15 help

Usage: help [cmd]

Description:
Display the list of bootloader built-in commands (when called with no arguments) or provide help on a specified command.

[cmd]
command to fetch help on.

3.1.16 defragenv

Usage: defragenv

Description:
This command de-fragments the environment space. De-fragmentation involves removal of obsolete (deleted or updated) environment entries. This command is not completely power fail-safe and should be used with caution.

3.1.17 ftp

Usage: ftp [-b] net_file env_var
        ftp -p

Description:
This command transfers a file from the host machine to the raw flash using the File Transfer Protocol. net_file is the name of the file to be fetched from the remote machine. env_var is an environment variable which has a value of the form start_address,end_address. (both addresses in Flash)
0x90200000-0x90300000. ftp will fetch the remote file and burn it in the flash starting at the address specified in the environment variable. (start_address)

CAUTION: This command should not be used with FFS as this will corrupt the installed FFS.

The ftp command depends on environment variables for relevant network/authentication related information. They are:

$ HWA_[0|1] : System MAC address.
$ IPA     : System IP address.
$ IPA_SVR : Target (or FTP server) IP address, running FTP server.
$ IPA_GATEWAY: The IP Address of the Gateway for the subnet in which the target is present.
$ SUBNET_MASK: The Subnet Mask for the subnet in which the target is present.
$ REMOTE_USER: The user name on the remote machine.
$ REMOTE_PASS: The password on the remote machine. Stored in an encrypted format.
$ REMOTE_DIR: The directory on the remote machine in which the remote file is present
$ MAC_PORT : The active MAC port number.

[-b]
This option checks if the start_address passed to the ftp command (through env_var) is block aligned or not. If not block aligned, ftp when used with –b option returns error.

-p
When FTP is called with this option, it prompts for the remote password and stores it in an encrypted format.

3.1.18 fa

Usage: fa

Description:
This command prints the flash memory allocation information. While printing the allocation for the bootloader, it prints the exact size of the bootloader instead of printing the size rounded to the nearest block. This provision helps determine the PSPBoot image size exactly.

The allocations for PSPBoot image, the environment space and FFS (if compiled in) are shown by default. Also information about amount of unallocated (free) space in flash is displayed.
3.1.19 info

Usage: info

Description:
This command prints the SoC name and revision number. It also displays various SoC related information like Cache size etc.

3.1.20 version

Usage: version

Description:
This command will print the PSPBoot version, build information and the optional modules included.
4 SYSTEM ENVIRONMENT VARIABLES

The environment variables supported in the system provide means for configuration data retention across power-cycles as well as parameter sharing between bootloader and its applications (OS inclusive). These variables are stored as a sort of file system on the Flash, in a contiguous erasable block.

This document covers the APIs required to configure and initialize the environment variables, describes how to add a new pre-defined variable for the system.

4.1 Customizing Environment Support

The location of the Flash Erase Block(s) where environments will be stored is determined by configuration using the Flash Memory Manager.

The Flash Memory manager will allocate a minimum of one erase block for storing the environment variables. Configuration can however be made to store environment variables only in a part of the block allocated. For making such a configuration, the macro ‘ENV_SPACE_SIZE’ in ‘inc/psbl/env.h’ should be set to the size of the environment space required. PSPBoot will not touch the remaining space in the erase block allocated. It can be used to store any application specific information. For example, the NSP configuration is stored in such a manner.

Configuration can also be made to de-fragment the environment space automatically after the environment space fills up. This de-fragmentation is carried out only when an attempt is made to set an environment variable (See setenv). After the de-fragmentation, the set command is called again with the same set of arguments. This de-fragmentation is carried out only if there is garbage left to be collected. By default, auto de-fragmentation of environment space is turned on. To turn it off set the macro AUTO_DEFRAg_ENVIRONMENT in ‘inc/psbl/env.h’ to FALSE.

4.2 Environment API List

4.2.1 sys_initenv

Name: sys_initenv  - initialize the environment variables module

Synopsis:
#include <env.h>

int sys_initenv(void);

Description:
This function validates the environment existing on the system, and if successful, initializes the internal data structures for access to the environment variables.

This routine should be called before using the environment variables.
Return Value:
Upon successful execution, SBL_SUCCESS is returned.

Errors:
SBL_EINVALID:
The environment file system is not of PSP Bootloader type. Under such conditions, environment variables cannot be used. The bootloader should be run again to install the appropriate environment type.

4.2.2 sys_setenv
Name: sys_setenv - update an environment variable

Synopsis:
#include <env.h>

int sys_setenv(char *env_var, char *val);

   env_var: Environment variable to be created.
   val:     Environment variable value, as string.

Description:
The sys_setenv() will update an environment variable of the system. The variable, identified with env_var will be set to value to which val points. If env_var was not created earlier, this call will create a new instance of it. Else, if the new val is different from the existing value, it will delete the earlier value and configure the variable to point to the new value, val.

The string supplied by val will be copied by this routine.

sys_setenv() is not re-entrant. Care should be taken by the caller to support thread-safe execution.

Return Value:
Upon successful execution, SBL_SUCCESS is returned.

Errors:
SBL_ERERESCRUNCH:
Insufficient memory in environment variables file system. Run the bootloader and defragment the environment space.

SBL_EINVALID:
Either sys_initenv() is not executed or the environment variable type is undefined. Refer to error SBL_EINVALID description in sys_initenv().

SBL_EFAILURE:
This identifies a Checksum error or data abuse or data corruption of the environment file system. On such error, sys_setenv() will not successfully execute. Run the PSP bootloader to format and install its environment file system.
4.2.3 sys_getenv

Name: sys_getenv - get value of an environment variable

Synopsis:
#include <env.h>

char *sys_getenv(char *env_var);

   env_var:  Environment variable.

Description:
The sys_getenv() function will search the environment variables file system for the given variable, env_var, if it exists and return a pointer to the value of the same. If the specified variable cannot be found, a null pointer will be returned.

The value, returned as a string, should not be modified by the caller. On doing so, the behavior is undefined. Instead, the caller should copy the string to another location for any processing.

sys_getenv() is not re-entrant. Care should be taken by the caller to support thread-safe execution.

Return Value:
Upon successful completion, sys_getenv() will return a pointer to the string containing the value of the specified env_var. If the specified env_var cannot be found on the environment file system or not defined in the list of supported environment variables for this system, it will return a NULL pointer.

4.2.4 sys_unsetenv

Name: sys_unsetenv - delete an environment variable

Synopsis:
#include <env.h>

char *sys_unsetenv(char* env_var);

   env_var:  Environment variable identifier.

Description:
The sys_unsetenv() function will remove an environment variable, env_var, from the environment file system. If the environment variable is not listed for system, then the environment file system will remain unchanged and the function is considered to have completed successfully.

sys_unsetenv() is not re-entrant. Care should be taken by the caller to support thread-safe execution.

Return Value:
Upon successful execution, SBL_SUCCESS is returned.
Errors:
SBL_EFAILURE:
This identifies a Checksum error or data abuse or data corruption of the environment file system. On such error, \texttt{sys\_setenv()} will not successfully execute. Run the PSP bootloader to format and install it's environment file system.

SBL_EINVALID:
Either \texttt{sys\_initenv()} is not executed or the environment variable type is undefined. Refer to error SBL_EINVALID description in \texttt{sys\_initenv()}.

4.2.5 \texttt{sh\_printenv} (shell command)

Name: \texttt{sh\_printenv} - dump the environment variables on the shell

Synopsis:
\verbatim
#include <env.h>

int sh_printenv(void);
\endverbatim

Description:
The \texttt{sh\_printenv} routine prints all of the environment variables for the system on the shell. This is targeted for the debug environment.

Return Value:
Upon successful execution, SBL_SUCCESS is returned.

Errors:
SBL_EINVALID:
Either \texttt{sys\_initenv()} is not executed or the environment variable type is undefined. Refer to error SBL_EINVALID description in \texttt{sys\_initenv()}.

4.3 Pre-defined Environment Variables

Following is the list of the environment variables supported for the system. Some of these environments may not be available based on the SoC for which PSPBoot is compiled.

\begin{itemize}
\item \textbf{CPUFREQ} \hspace{1cm} CPU Frequency (Read Only).
\item \textbf{SYSFREQ} \hspace{1cm} System Frequency (Read Only)
\item \textbf{MEMSZ} \hspace{1cm} System Memory Size. If this variable is found to be unset at boot-time, it will be set to the value taken from \texttt{sysconf.h}
\item \textbf{FLASHSZ} \hspace{1cm} System Flash Size. If this variable is found to be unset at boot-time, it will be set to the value taken from \texttt{sysconf.h}
\item \textbf{MODETTY0} \hspace{1cm} Serial Port 0 Working Mode.
\end{itemize}
The mode parameters comprise of baud rate, parity, data bits, stop bits and flow-control mechanism. For a port configuration of 9600 baud, no parity, 8 data bits, 1 stop bit and hardware flow-control, it should be configured as: 9600,n,8,1,hw. If this variable is found to be unset at boot-time, it will be set to a default value of 9600,n,8,1,hw.

**MODETTY1** Serial Port 1 Working Mode. Format is the same as for MODETTY0.

**PROMPT** Bootloader shell prompt. This variable is read at boot-time only. If this variable if found to be unset at boot-time it will be set to default value of “psbl”

**BOOTCFG** Boot Configuration. (find more details in documentation of ‘boot’ command)

**HWA_0** Hardware Address for the first Ethernet MAC interface.

The delimiters for the octects can be any of “.-:_” (excluding the double quotes), for example: 00.e0.a6.66.39.54

**HWA_1** Hardware Address for the second Ethernet MAC interface.

**HWA_RNDIS** Hardware Address, for RNDIS interface. (Target side)

**HWA_3** Hardware Address, for Ethernet-over-ATM interface.

**HWA_HRNDIS** Hardware Address, for RNDIS interface. (Host side)

**IPA** System IP Address.

**IPA_SVR** TFTP/FTP Server IP Address.

**IPA_GATEWAY** Target’s Gateway IP address.

**SUBNET_MASK** Target’s Gateway IP address.

**BLINE_MAC0** Bootline ascii string for VxWorks with initialization parameters for CPMAC-0 interface.

These parameters hold boot device name, unit number, internet addresses for the host and the interface, network mask, user name and password.

A sample BLINE_MAC0 string is:
cpmac(0,0)host:file h=192.138.139.235 e=192.138.139.81:FFFFFFF00 u=titan pw=titan

**BLINE_MAC1** Bootline parameters for CPMAC-1 interface.
A sample BLINE_MAC1 string is:
cpmac(1,0)host:file h=192.138.139.235 e=192.138.139.
82:FFFFFF00 u=titan pw=titan

BLINE_RNDIS  Bootline parameters for RNDIS interface.
A sample BLINE_RNDIS string is:
rndis(2,0)host:file h=192.138.139.235 e=192.138.139.
83:FFFFFF00 u=titan pw=titan

BLINE_ATM     Bootline for ATM interface.
BLINE_ESWITCH Bootline for Eswitch.

USB_PID      Product ID of USB device. (16-bit value)
USB_VID      Vendor ID of the USB device. (16-bit value)
USB_EPPOLLI  USB Endpoint Polling Interval, in milliseconds. This is the polling rate for device interrupts.
USB_SERIAL   Serial number of USB device.
REMOTE_USER  User Name on the remote system (for FTP)
REMOTE_PASS  User password on remote system (for FTP)
REMOTE_DIR   Directory on the remote system (for FTP)
LINK_TIMEOUT Time in seconds to wait for the link to come up before starting DHCP.

MAC_PORT     The Active MAC port number. Valid values: 0 and 1
PATH         The path in FFS to search for application to execute. This variable should consist of directories in FFS separated by ‘:’ Example /bin:/dev/ttyS0. This variable is read at boot time only. If at boot-time this variable is found to be undefined, it takes a default value of “/bin:/dev/ttyS0”

HOSTNAME     The hostname for the Target.

TFTPCFG      Configure the timeout and retransmission count of TFTP client. The configuration is of the form:
              <Time-out (seconds)>: <Retransmissions Count>
              For Example, TFTPCFG value of 6:5 denotes that the timeout is 6 seconds and the retransmissions count is 5. If this variable is not set or not set properly, the default configuration is taken. The default time-out is 5 seconds and the default retransmissions count is 5. Set of valid values for time-out and retransmissions count is positive integers greater than zero. Care should be taken while setting the TFTPCFG variable. It should be set taking into account the timeout and retransmission values at the server side. A mismatch might result in one of the sides waiting for transfer while the other times out.
4.4 Adding Environment Variables

4.4.1 Adding new pre-defined environment variables

The bootloader supports pre-defined environment variables. Every environment is referenced by a unique index, common across the system. This indexing is supported by using the enumerator data type of C.

The file, env.c should be common across PSP bootloader and OS and any other application running on the system. This is since the environment file system is a shared global resource.

Following are the steps for adding a new pre-defined environment variable.

NOTE: Steps 1 and 2 are for adding a new environment of the PSP Bootloader type.

1) Add the environment variable name in the enumeration data type, ENV_VARS, in file env.c. Care should be taken that the new variable name is added to the end of the ENV_VARS, immediately before the enumerator env_vars_end. This allows intact pre-defined environment variables and assigning a new unique value for the added variable.

To use the added environment variable with the APIs sys_setenv(), sys_getenv(), sys_unsetenv(), use the new enumerator name as a string.

2) Add the new environment enumerator name in the array, env_ns, in env.c, using the macro _ENV_ENTRY. This macros provides a lookup for the respective name string for the environment. This entry in the env_ns can be added anywhere and there is no positioning requirement. The last entry in this array should always be _ENV_ENTRY(0).

This support is mainly for the shell support for set and print of environments.

4.4.2 Adding new dynamic environment variables

Addition of dynamic variables is fairly simple. A new dynamic variable can be added by using the setenv command of the bootloader and supplying a variable name and value. Dynamic variables are to be used with caution as they are less space efficient than pre-defined ones. They are meant for ease of use in development stage. Once the system is in deployment stage, it is highly recommended to convert all the finalized environment variables into pre-defined ones.
5  FLASH FILE SYSTEM (FFS)

5.1  API List

5.1.1  ffs_init

Name: ffs_init – Initialize the Flash File System

Synopsis:
#include <ffs.h>

int ffs_init(void);

Description:
The ffs_init routine is used to initialize the FFS before using it.

Return Value:
This always return with 0, indicating successful operation.

5.1.2  ffs_fopen

Name: ffs_fopen – open a file on FFS

Synopsis:
#include <ffs.h>

FFS_FILE *ffs_fopen(const char *filename, const char *type);

   filename: File name to open (string)
   type:     Mode of file operation (string)

Description:
The ffs_fopen routine initializes the data structures required for file operations. Two fundamental kinds of access are provided: read and write. `type' must begin with either of "r" or "w", to select either for read or write operation, respectively.

This returns a file pointer for usage on every subsequent reference to the opened file.

Append mode is not supported.

Returns:
Upon successful execution, a file pointer of type FFS_FILE will be returned. On failure, it returns with NULL.

5.1.3  ffs_fread

Name: ffs_fread – read array elements from an FFS file

Synopsis:
#include <ffs.h>
size_t ffs_fread(void *ptr, size_t size, size_t nitems, FFS_FILE *stream);

ptr: Memory pointer, to copy read data to.
size: Size of each element to read.
nitems: Number of elements to read.
stream: File pointer, identifying the file to read from.

Description:
`ffs_fread' reads `nitems' number of elements, each of size `size', to the memory pointed to by `stream'. `ffs_fread' may copy fewer elements than `nitems' if an error, or end of file occurs. `ffs_fread' also advances the file position indicator in it's FCB by the actual number of elements read.

Returns:
It returns the number of elements read. Upon error or EOF, it returns 0, indicating zero elements read.

5.1.4 ffs_fwrite

Name: ffs_fwrite – write array elements to an FFS file

Synopsis:
#include <ffs.h>
size_t
ffs_fwrite(const void *ptr, size_t size, size_t nitems, FFS_FILE *stream);

ptr: Memory pointer, to copy data from.
size: Size of each element for write.
nitems: Number of elements to write.
stream: File pointer, identifying the file to write to.

Description:
`ffs_fwrite' attempts to copy `nitems' number of elements, each of size `size', starting from memory location `ptr' to the file, `stream'. `ffs_fwrite' may write fewer elements than requested if an error occurs. `ffs_fwrite' will also advance the file position indicator in it's FCB by the actual number of elements written.

Returns:
Number of elements written. On an error, it returns 0.

5.1.5 ffs_fseek

Name: ffs_fseek – set file position

Synopsis:
#include <ffs.h>
int ffs_fseek(FFS_FILE *stream, bit32 offset, int ptrname);

    stream:  FFS file pointer.
    offset:  New position offset.
    ptrname: Mode of `offset' usage.

Description:
`ffs_fseek' sets the file position indicator in it's FCB, defined by `offset'. The different referencing modes are:

    FFS_SEEK_SET: `offset' is the absolute file position desireable.
    FFS_SEEK_CUR: `offset' is relative to the current file position.
    FFS_SEEK_END: `offset' is relative to the current end of file. `offset' can be either positive or negative.

Returns:
Upon successful operation, it returns 0. On failure, it returns EOF.

5.1.6 ffs_fclose
Name: ffs_fclose - close a file

Synopsis:
#include <ffs.h>
int ffs_fclose(FFS_FILE *stream);

    stream: FFS file pointer.

Description:
`ffs_fclose' closes an FFS file `stream' if already open, after ensuring that any pending data is returned.

Returns:
When successful, it returns 0. Otherwise, it returns EOF.

5.1.7 ffs_remove
Name: ffs_remove - delete an FFS file

Synopsis:
#include <ffs.h>
int ffs_remove(const char *filename);

    filename: File to be removed (string)

Description:
`ffs_remove' deletes a file from the FFS file system. Following a remove, the file will no longer be accessible.

Returns:
On success, it returns 0. Else, it returns -1.
6 ENVIRONMENT AND FFS SUPPORT FOR OPERATING SYSTEMs

Since PSPBoot is meant to cater to various Operating Systems (like VxWorks, Linux and WinCE), a set of files for the environment variables and flash file system, will be provided which require to be pulled in and compiled with the respective compiler suite of the particular OS.

Here is the list of files to be compiled in for Environment and FFS support. These files can be soft-linked into the OS VoB from the psp_boot VoB. psp_boot VoB will contain a directory named ‘export’ which will contain links (within ‘psp_boot’ VoB) to files required to be compiled in the OS kernel for environment and FFS support. The directory ‘inc’ within the ‘export’ directory contains the header files containing the interfaces for environment and FFS access.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Required to Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>inc\psbl\debug.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>psbl\kernel\env.c</td>
<td>ENV</td>
</tr>
<tr>
<td>inc\psbl\env.h</td>
<td>ENV</td>
</tr>
<tr>
<td>inc\psbl\errno.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>inc\psbl\flashop.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>psbl\kernel\flashop_xx.c [Board dependent]</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>inc\psbl\hw.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>platform.h [To be provided by the OS]</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>Platform.c [To be provided by the OS]</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>Psbl\kernel\shell.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>inc\psbl\stddef.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>inc\psbl\sysconf.h</td>
<td>ENV and FFS</td>
</tr>
<tr>
<td>Psbl\kernel\fcb.c</td>
<td>FFS</td>
</tr>
<tr>
<td>Psbl\kernel\ffs.c</td>
<td>FFS</td>
</tr>
<tr>
<td>Psbl\kernel\ffs_extra.c</td>
<td>FFS</td>
</tr>
<tr>
<td>Psbl\kernel\ffs_extra.h</td>
<td>FFS</td>
</tr>
<tr>
<td>Psbl\kernel\fcb.h</td>
<td>FFS</td>
</tr>
<tr>
<td>inc\psbl\mod.h</td>
<td>FFS</td>
</tr>
<tr>
<td>inc\psbl\ffs.h</td>
<td>FFS</td>
</tr>
<tr>
<td>inc\psbl\stdio.h</td>
<td>FFS</td>
</tr>
</tbody>
</table>

These files should be included in the OS sources and compiled into the OS kernel. For initializing the Environment support, `sys_initenv()` should be called. This call should be made before Environment/Flash File System support is used.

The file `env.c` has an external dependency on the OS to provide it with mutual exclusion primitives for serializing Flash access (Read or Write). The functions `int enter_critical_section()` and `int exit_critical_section()` should be defined by the OS (preferably in `platform.c`). These primitives when defined, will ensure that there is only a single process reading or writing to the flash at any point of time. Please note that this support is available only for Environment variables and not for flash.
file system. If such a (mutual exclusion) facility is not required, then these routines can be defined to be empty routines.

The file `platform.h` is used to provide any OS dependent defines for compiling the PSPBoot files. For Example, for VxWorks, `platform.h` should define `sys_printf` (print function in PSPBoot) to `printf`. (print function in vxWorks).

The files `platform.c` and `platform.h` will not be provided in PSPBoot VoB and will have to be maintained along with the OS sources.

Care should be taken **not** to define the macro `_STANDALONE` while compiling the bootloader sources. This macro is defined only when compiling PSPBoot and is used in the code to decide the context. (OS or bootloader).

Also note that it is not mandatory to use both the environment and FFS support together. For example, Linux currently uses only environment support and not FFS support.

### 7 DHCP SUPPORT

PSPBoot has DHCP client support that is used to get the system IP address, subnet mask, gateway address (router), boot-file name and the TFTP server IP address.
DHCP support is enabled using the BOOTCFG environment variable. For more details on enabling DHCP using the BOOTCFG environment variable please look at documentation on BOOTCFG in the description of `boot` command.

The DHCP client sends a broadcast request to the DHCP server(s) in the same subnet requesting IP address, gateway IP address, boot-file name and TFTP server name. The DHCP server must be configured correctly to provide this information to the DHCP client.

If the boot configuration (using BOOTCFG) requires that the boot-file be downloaded from network, DHCP server must provide the correct “boot-file name” as well as the “next server IP address” else the OS boot will fail. If DHCP server provides the “next server IP address” as 0.0.0.0 then IPA_SVR is not set and this configuration is ignored. Similarly, “boot-file name” of NULL value is ignored. In such cases, the DHCP server’s configuration should be reviewed.

The “Vendor class identifier” sent by the DHCP client in PSPBoot 1.1 is “TI-dhcp 0.1”. The hostname sent by the target during the DHCP process is taken from the environment variable HOSTNAME. If HOSTNAME variable is not present, a default value (value depending on the board for which PSPBoot is compiled) is sent.

DHCP request is sent out only when a reboot is done. The environment variable LINK_TIMEOUT is used to specify the time in seconds to wait for the link to come-up before starting the DHCP process.

References:

RFC 2131
RFC 2132
8 BOOTLOADER CUSTOMIZATION

8.1 Application Support: Extending Bootloader Functionality.

8.1.1 Introduction

A PSPBoot application is an independent program written to be executed in PSPBoot context. Applications are mainly used to extend the functionality of PSPBoot. Applications are developed and built using the application framework provided by PSPBoot. The application framework consists of defining the memory map for the application, creation of C-Runtime environment for application execution and a makefile for compiling the applications.

8.1.2 Application interface to PSPBoot

PSPBoot provides a number of pre-defined entry points into the bootloader to the applications. These entry points enable the application to use a number of bootloader features. It also reduces the size of the applications by reuse of routines that already exist in the bootloader context. These entry points are termed as SRRs (System Resident Routines). The application is compiled along with a library of SRRs (libsrr.a) that helps the application jump to the appropriate location whenever an SRR is invoked. Thus SRRs can be invoked just like any normal function. The linkage to bootloader context is taken care by the application framework through the SRR Library. The list of currently supported SRRs can be found in the file inc\apps\srr.h in the source tree.

8.1.3 Writing applications for PSPBoot

The sources for all the applications written for PSPBoot should be kept in apps\src directory. The Application will start executing at a function called mymain. This function is a must for all applications. A template application is provided in the file apps\src\template.c. This file can be taken as a starting point for all new applications.

8.1.4 Compiling applications for PSPBoot

To compile an application spanning only one C source file, the source file name should be added to the APPS_SRCS_C list in apps\makefile. For example, if the source file is app.c, the application can be compiled by issuing the command: make app.

To compile an application spanning multiple C/Assembly source files, a new rule should be created which will compile all the C and Assembly files related to the application. Also, this rule should be added as a dependency for the ‘all’ rule. To build all the applications supported, use make all.
8.1.5 Sample applications

The following sample applications are provided.

8.1.5.1 im: inspect memory

This application is used to read and modify memory.

Usage: im [-b | -h | -w] [Hexadecimal Address]

This application takes an optional hexadecimal address and displays the contents of that memory location. If no address is provided the inspection starts at 0xb400'0000. The application advances to next address when return key is pressed. Pressing the ‘-‘ key goes to the previous address. To exit the application, ‘.’ key should be pressed.

[-b] Inspect memory byte-wise.
[-h] Inspect memory half-word wise.
[-w] Inspect memory word wise. (default)

8.1.5.2 update: update bootloader using TFTP.

This application is used to update the bootloader using TFTP.

Usage: update [-est] <bootfile-name>

This application takes a bootfile and burns it at 0xb000'0000 (start of flash) after downloading it through TFTP. The bootfile should be a flat binary file for flash download. If the bootfile being downloaded is a flat binary image obtained through the VisionClick™ “convert” utility, -est should be used to indicate this.

8.1.5.3 defrag: de-fragment the Flash File System.

This application is used to de-fragment the Flash File System. De-fragmentation of the Flash File System physically frees up the space used by the deleted files in the FFS.

Usage: defrag [-h | -r | -v]

[-h] Prints help information and quit.
[-r] Just report the amount of space that can be freed and quit.
[-v] Run the tool in verbose mode

8.2 Flash Memory Manager: Allocating Sections in Flash

Flash Memory manager (FMM) is designed to facilitate easy and optimum use of flash.
Certain Intel and AMD flash types have ‘Tiny’ blocks built into them. The ‘Tiny’ blocks are small blocks – typically 4K or 8K or 16K in size – that are present so that wastage of space in a large segment due to storage limitations of flash can be avoided. These tiny blocks are typically used to store system environment variable, configuration data etc.

In PSPBoot1.1, all the allocation of flash memory – including that for Environment Variables, Flash File System and the bootloader itself – is done through the Flash Memory manager.

The FMM module mainly exposes two interfaces:

FMMSetSectionInfo: This interface is used to allocate a section in flash at a specified location in flash, overriding the Flash memory manager’s allocation algorithm. For example, bootloader is always allocated a section at the base of flash irrespective of the tiny block position using this API.

FMMAllocateSection: This interface is used to allocate a section in flash based on Flash Memory manager’s allocation algorithm. Flash memory manager always allocates the ‘tiny blocks’ first. Hence if a section (like environment space) has to be allocated ‘tiny blocks’ on a higher priority over another section (like FFS), it should be allocated space first.

To get the minimum block allocated, a size of 1 byte can be specified. To get all the currently free space allocated, a size equal to size of Flash can be requested.

Each section is identified by a unique string literal (tag). The tags for the bootloader, Environment space and FFS are fixed and should not be changed. Users are free to use their own tags (max length 10) for the user defines sections.

The allocation of all sections takes place under a single API FMMConfigSections in file fmm.c. Allocation of new sections should be done under this function.

8.3 BU specific Hooks

The following hooks are provided for BUs to add their own code at various points if initialization/execution.

1. void psbl_asmhook_init1 (void)

This is an assembly hook function that gets called immediately after the processor initialization. The SoC is not yet initialized. A sample usage of this hook is similar to the usage with TNETD7300 where the power comparator requires to be turned on as early as possible in the bring-up process to save the board from any rapid power fluctuations.

2. void psbl_asmhook_init2 (void)

This is an assembly hook function that gets called after the SoC initialization. At this stage, the processor and SoC is initialized and basic POST is executed successfully. Functionalities like extended POST, SOC workarounds and some hardware bug fixes can be performed here.
3. void psbl_chook_init3 (void)

This hook is present at the same stage as psbl_asmhook_init2 but allows the hook to be written in C language.

4. void psbl_chook_init4 (void)

This is a C function that will be called after the system is brought up and before executing the shell engine. At this stage, the whole of the system is up, with environment variables configured with the board specific information, serial driver/network drivers installed, and CPU/System frequencies configured. Any functionality that can be executed via a command can be installed here.

5. void psbl_chook_poll (void)

This is a C function that will be called periodically when the shell engine is running. This hook can be used, for example, to periodically poll the status of any device/module and execute, if any, and call a user defined function under any given condition.

8.4 EMIF configuration

EMIF configuration for PSPBoot can be done in a flexible manner for AR7, TNETV1050 and TNETV1020 platforms.
The file inc/emif.h includes a board specific include file. The board specific file contains set of definitions for different parameters for SDRAM and Asynchronous banks of the SoC. These definitions can be changed to suit any particular modifications in EMIF settings that may be required.

8.5 Miscellaneous Customizations

8.5.1 Secondary Flash Support

PSPBoot has the capability to support two discontinuous (referred to as primary and secondary) flash devices. The Flash File System is implemented to show both the devices as a single unit. Use the Makefile (\psp_boot\Makefile) option -DDUAL_FLASH to enable this support.

Notes:
1. PSPBoot requires the definition of SEC_FLASH_BASE when Dual Flash option is chosen. This definition has to be according to where the secondary flash resides. The primary flash is always assumed to start at CS0
2. PSPBoot assumes that both the flash devices are of same type. That is, the same flash driver can operate on both the devices.
3. PSPBoot assumes that the flash driver is aware of the existence of the secondary flash device.
9 KNOWN LIMITATIONS

- The PSP Bootloader depends on sead1.exe on host machine for serial download.
- The ftp command should not be used with FFS installed.
- PC application is supported only in Little Endian mode.

For a full set of known limitations for a particular release, please refer to the Release Notes of that release.