TI81XX PSP User Guide

TI81XX PSP User Guide

Linux PSP

IMPORTANT
TI81XX refers to TI816X, TI814X and TI813X.

About this Manual

This document describes how to install and work with Texas Instrument's Linux Platform Support Package (PSP) for the TI81XX platform. PSP provides a fundamental software platform for development, deployment and execution on TI81xx platform. It abstracts the functionality provided by the hardware and forms the basis for all application development on this platform.

Software Overview

To begin developing applications, you need to install PSP package and the toolchain.

Package Contents

Important
The values of MM, mm, pp and bb in this illustration will vary across the releases and actually depends on individual component versions

Extract the contents of release package on your Linux host with the following command:

```
$ tar -xvzf  TI81XX-LINUX-PSP-MM.mm.pp.bb.tgz
```

This creates a directory **TI81XX-LINUX-PSP-MM.mm.pp.bb** with the following contents:

```
|-- docs
|   |-- TI81XX_PSP_MM.mm.pp.bb_Feature_Performance_Guide.pdf
|   |-- TI81XX_PSP_MM.mm.pp.bb_Release_Notes.pdf
|   |-- TI81XX_PSP_AUDIO_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_EDMA_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_EMAC_Boot.pdf
|   |-- TI81XX_PSP_ETHERNET_Switch_User_Guide.pdf
|   |-- TI81XX_PSP_Flashing_Tools_Guide.pdf
|   |-- TI81XX_PSP_GPIO_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_HDMI_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_HDMI_Sii9022a_Transmitter_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_IOMMU_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_McSPI_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_MMC_SD_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_NAND_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_NOR_Driver_User_Guide.pdf
```

```
|   |-- TI81XX_PSP_PCI_Express_Endpoint_Boot_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_PCI_Express_Endpoint_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_PCI_Express_Root_Complex_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_PM_AVS_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_PM_CLOCK_FRAMEWORK_User_Guide.pdf
|   |-- TI81XX_PSP_PM_DVFS_User_Guide.pdf
|   |-- TI81XX_PSP_PM_SUSPEND_RESUME_User_Guide.pdf
|   |-- TI81XX_PSP_UBOOT_User_Guide.pdf
|   |-- TI81XX_PSP_USB_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_User_Guide.pdf
|   |-- TI81XX_PSP_VIDEO_CAPTURE_Driver_User_Guide.pdf
|   |-- TI81XX_PSP_VPSS_Video_Driver_User_Guide.pdf
|   `-- TI81XX_PSP_WDT_Driver_User_Guide.pdf
|-- host-tools
|   |-- DM814x_gel.zip
|   |-- DM816x.gel
|   |-- mksd-ti814x.sh
|   |-- mksd-ti816x.sh
|   |-- nand-flash-writer.out
|   |-- norflash-writer.out
|   |-- spi-flash-writer.out
|   |-- src
|   |   |-- nandflash-MM.mm.pp.bb.tar.gz
|   |   |-- norflash-MM.mm.pp.bb.tar.gz
|   |   |-- spiflash-MM.mm.pp.bb.tar.gz
|   |   `-- switch-config-MM.mm.pp.bb.tar.gz
|   `-- switch-config
|-- images
|   |-- examples
|   |   `-- pcie
|   |       |-- boot.scr
|   |       `-- saBootApp
|   |-- kernel
|   |   |-- ti813x
|   |   |   `-- uImage
|   |   |-- ti814x
|   |   |   `-- uImage
|   |   `-- ti816x
|   |       `-- uImage
|   `-- u-boot
|       |-- ti813x
|       |   |-- nand
|       |   |   |-- u-boot.bin
|       |   |   `-- u-boot.min.nand
|       |   `-- sd
|       |       |-- MLO
|       |       `-- u-boot.bin
```

```
|       |-- ti814x
|       |   |-- nand
|       |   |   `-- u-boot.min.nand
|       |   `-- sd
|       |       |-- MLO
|       |       `-- u-boot.bin
|       `-- ti816x
|           |-- nand
|           |   `-- u-boot.noxip.bin
|           |-- sd
|           |   `-- MLO
|           `-- u-boot.bin
|-- src
|   |-- examples
|   |   |-- examples.tar.gz
|   |   `-- pcie
|   |       |-- bootscript.txt
|   |       |-- Makefile
|   |       |-- README
|   |       `-- saBootApp.c
|   |-- kernel
|   |   |-- ChangeLog-MM.mm.pp.bb
|   |   |-- diffstat-MM.mm.pp.bb
|   |   |-- kernel-patches-MM.mm.pp.bb.tar.gz
|   |   |-- linux-MM.mm.pp.bb.tar.gz
|   |   |-- Readme.txt
|   |   |-- ShortLog
|   |   `-- Unified-patch-MM.mm.pp.bb.gz
|   `-- u-boot
|       |-- ChangeLog-MM.mm.pp.bb
|       |-- diffstat-MM.mm.pp.bb
|       |-- Readme.txt
|       |-- ShortLog
|       |-- u-boot-MM.mm.pp.bb.tar.gz
|       |-- u-boot-patches-MM.mm.pp.bb.tar.gz
|       `-- Unified-patch-MM.mm.pp.bb.gz
`-- TI81XXPSP_Software_Manifest.doc
```

Few points to note about the above structure:

Assuming the release package is extracted inside directory represented as $TI81XX-PSP-DIR:

- U-Boot source tarball is u-boot-MM.mm.pp.bb.tar.gz needs to be extracted on Linux build host. This will create U-Boot source base for TI81XX

```
$ cd $TI81XX-PSP-DIR/TI81XX-LINUX-PSP-MM.mm.pp.bb/src/u-boot
$ tar -xvzf u-boot-MM.mm.pp.bb.tar.gz
```

- Similarly kernel source tarball linux-MM.mm.pp.bb.tar.gz from TI816x-LINUX-PSP-MM.mm.pp.bb/src/kernel directory needs to be extracted to have kernel source directory setup for building kernel and/or modules

```
$ cd $TI81XX-PSP-DIR/TI81XX-LINUX-PSP-MM.mm.pp.bb/src/kernel
$ tar -xvzf linux-MM.mm.pp.bb.tar.gz
```

- Various Video and Audio example sources are included inside $TI81XX-PSP-DIR/examples/examples.tar.gz
- Additionally, Watchdog timer and PCIe Endpoint boot applications are provided in respective directories inside $TI816X-PSP-DIR/examples

Toolchain

To build U-Boot and the Linux kernel, you will need to download and install CodeSourcery ARM tool chain version 2009-q1. For additional documentation on installing the CodeSourcery tools, please visit https://sourcery.mentor.com/sgpp/portal/release858 [1].

To help you get started quickly the PSP package comes with pre-built binaries. However, after making any changes to U-Boot and Linux Kernel you need to cross-compile them using the CodeSourcery toolchain and use the new binaries that are generated

In this context, the document contains instructions to:

- Install the release
- Build the sources contained in the release

The document also provides detailed description of drivers and modules specific to this platform - as implemented in the PSP.

Installation

Prerequisites

Before you begin with the installation of the package please make sure you have met the following system requirements:

- Windows machine
- Host machine running a version of Linux such as RedHat Enterprise Linux or Ubuntu.
- TI816X/AM389x EVM or TI814X EVM or TI813X or J5Eco EVM

The Windows machine is used for running CCSv4/v5, which will be used to build flash writers and also to burn the boot images (U-Boot) onto the flash using the flash writers provided.

The Linux host is used for the following:

- Compiling U-Boot and the Linux kernel.
- Hosting the NFS server to boot the EVM with NFS as root filesystem

You can use either the Windows machine or the Linux host for

- Hosting the TFTP server required for downloading kernel and filesystem images from U-Boot using Ethernet.
- Running a serial console terminal application

Environment Setup

After you have installed the CodeSourcery toolchain you need to setup the environment in the Linux host.

1. Set the environment variable PATH to contain the binaries of the CodeSourcery cross-compiler tool-chain.

   For example, in bash:

   ```
   $ export PATH=/opt/toolchain/2009-q1/bin:$PATH
   ```

2. Add the location of U-Boot tools directory to the PATH environment variable (required for mkimage utility that is built as part of U-Boot build process and is needed to generate uImage when building the kernel)

   For example, in bash:

   ```
   $ export PATH=/opt/u-boot/tools:$PATH
   ```

Note

Actual commands to be used for setting the environment variables will depend upon your shell and location of the tools

Flashing Tools

TI816X

On TI816X EVM, the Cortex-A8 core boots up first. On boot-up, the ARM core runs the U-Boot image which need to be present in the flash memory of the EVM.

The flash-writers are a part of the PSP package. Instructions on how to use the flash-writers in CCSv4/v5 can be found in **PSP Flashing Tools Guide** (TI81XX_PSP_Flashing_Tools_Guide.pdf).

TI814X

On TI814X EVM, the Cortex-A8 core boots up first. On boot-up, the ARM core runs the U-Boot image which need to be present in the flash memory of the EVM.

The flash-writers are a part of the PSP package. Instructions on how to use the flash-writers in CCSv4/v5 can be found in **PSP Flashing Tools Guide** (TI81XX_PSP_Flashing_Tools_Guide.pdf).

U-Boot

TI816X, TI814X, TI813X

Please refer to **U-Boot User Guide** for details about U-Boot in the context of TI816X, TI814X and TI813X.

Linux Kernel

This chapter describes the steps required to build and configure the Linux kernel. It also provides basic steps to boot kernel on the EVM.

Compiling Linux Kernel

Change to the base of the Linux source directory.

Choose default kernel configuration for your platform.

TI816X

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm ti8168_evm_defconfig
```

TI814X

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm ti8148_evm_defconfig
```

TI813X

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm dm385_evm_defconfig
```

Initiate the build.

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm uImage
```

Note

For the kernel image (uImage) to be built, mkimage utility must be included in the path. mkimage utility is generated (under tools folder of U-Boot) while building the U-Boot binary.

On successful completion, file uImage will be created in the directory `./arch/arm/boot`.

Copy this file to the root directory of your TFTP server.

Modifying Kernel Configuration

TI816X

This section describes the steps to configure the kernel for TI816X EVM and illustrates related configuration items for reference.

Start with the default configuration for your platform

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm ti8168_evm_defconfig
```

To view configuration interactively:

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm menuconfig
```

From the onscreen menu, select System Type and press ENTER:

```
   General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
  System Type  --->
  Bus support  --->
  Kernel Features  --->
  ...
  ...
```

Select ARM system type

```
ARM system type (TI OMAP)  --->
TI OMAP Common Features  --->
TI OMAP2/3/4 Specific Features  --->
```

You will be presented with various ARM based SoCs. Select OMAP as

```
(X) TI OMAP
```

After pressing ENTER, you will be presented earlier menu. Select "TI OMAP2/3/4 Specific Features" and press ENTER. Select various options as shown below (marked with '[*]') by pressing Y on each item after selecting it.

```
[*] Typical OMAP configuration
[ ] TI OMAP2
[ ] TI OMAP3
[ ] TI OMAP4
[*] TI 81XX
[*] TI816X support
...
   *** TI81XX Board Type ***
[*] TI8168 Evaluation Module
```

Note that the "TI816X support" option will be shown only if you enable "TI 81XX". Similarly, "TI8168 Evaluation Module" option will be presented only after selecting "TI816X support"

Choose Exit successively to return to previous menu(s) and eventually back to the shell. Make sure to save the changes you have made when prompted at exit.

Some of the key drivers which may be enabled in the default configuration are:

- Serial port
- SATA
- Ethernet
- PCI Express Root Complex
- MMC/SD
- I2C
- GPIO
- USB
- Video Display
- Audio
- NAND
- Touchscreen
- PCI Express Root Complex
- PCI Express Boot Driver
- PCI Express EP Driver (mutually exclusive with RC driver)
- WDT

If you are looking for some particular functionality please double-check that it is enabled in the config options.

Also, please refer individual driver documentation about enabling particular driver and various configurations and dependencies as applicable.

TI814X

This section describes the steps to configure the kernel for TI814X EVM and illustrates related configuration items for reference.

Start with the default configuration for your platform

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm ti8148_evm_defconfig
```

To view configuration interactively:

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm menuconfig
```

From the onscreen menu, select System Type and press ENTER:

```
   General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
System Type  --->
Bus support  --->
Kernel Features  --->
...
...
```

Select ARM system type

```
ARM system type (TI OMAP)  --->
TI OMAP Common Features  --->
TI OMAP2/3/4 Specific Features  --->
```

You will be presented with various ARM based SoCs. Select OMAP as

```
(X) TI OMAP
```

After pressing ENTER, you will be presented earlier menu. Select "TI OMAP2/3/4 Specific Features" and press ENTER. Select various options as shown below (marked with '[*]') by pressing Y on each item after selecting it.

```
[*] Typical OMAP configuration
[ ] TI OMAP2
[ ] TI OMAP3
[ ] TI OMAP4
[*] TI 81XX
[ ] TI816X support
[*] TI814X support
...
*** TI81XX Board Type ***
[*] TI8148 Evaluation Module
```

Note that the "TI814X support" option will be shown only if you enable "TI81XX". Similarly, "TI8148 Evaluation Module" option will be presented only after selecting "TI814X support"

Choose Exit successively to return to previous menu(s) and eventually back to the shell. Make sure to save the changes you have made when prompted at exit.

Some of the key drivers which may be enabled in the default configuration are:

- Serial port

- SATA
- Ethernet
- I2C
- GPIO
- Audio
- MMC/SD
- NAND
- PCI Express Root Complex
- PCI Express Boot Driver
- PCI Express EP Driver (mutually exclusive with RC driver)
- WDT
- USB

If you are looking for some particular functionality please double-check that it is enabled in the config options.

TI813X

This section describes the steps to configure the kernel for TI813X EVM and illustrates related configuration items for reference.

Start with the default configuration for your platform

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm dm385_evm_defconfig
```

To view configuration interactively:

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm menuconfig
```

From the onscreen menu, select System Type and press ENTER:

```
   General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
System Type  --->
Bus support  --->
Kernel Features  --->
...
...
```

Select ARM system type

```
ARM system type (TI OMAP)  --->
TI OMAP Common Features  --->
TI OMAP2/3/4 Specific Features  --->
```

You will be presented with various ARM based SoCs. Select OMAP as

```
(X) TI OMAP
```

After pressing ENTER, you will be presented earlier menu. Select "TI OMAP2/3/4 Specific Features" and press ENTER. Select various options as shown below (marked with '[*]') by pressing Y on each item after selecting it.

```
[*] Typical OMAP configuration
[ ] TI OMAP2
[ ] TI OMAP3
```

```
[ ] TI OMAP4
[*] TI 81XX
[ ] TI816X support
[*] TI814X support
...
*** TI81XX Board Type ***
[*] TI8148 Evaluation Module
[*] DM385 Evaluation Module
```

Note that the "TI814X support" option will be shown only if you enable "TI 81XX". Similarly, "TI8148 Evaluation Module" and "DM385 Evaluation module" options will be presented only after selecting "TI814X support"

Choose Exit successively to return to previous menu(s) and eventually back to the shell. Make sure to save the changes you have made when prompted at exit.

Some of the key drivers which may be enabled in the default configuration are:

- Serial port
- SATA
- Ethernet Switch
- I2C
- Audio
- MMC/SD
- NAND
- WDT

Auto detection of Kernel Load Address and Runtime RAM Base Determination

Note: This section is only applicable to following releases of Linux Kernel
TI816X: 04.00.02.14 onwards, which includes PSP 04.04.00.xx

TI814X: 04.01.00.07 onwards, which includes PSP 04.04.00.xx

TI813X: PSP 04.04.00.xx

By default, the TI81XX kernel is built to be loaded at physical address 0x80008000 in RAM and take start of RAM as 0x80000000.

It is possible to load the kernel at a different location in RAM with kernel automatically determining RAM base depending upon the load address.

This is achieved by the combination of two features added the kernel:

1. Run time PHYS_OFFSET determination: This feature enables determining physical to virtual translations dynamically depending upon the position of kernel in system memory.
2. Auto calculation of address for uncompressed kernel: This feature enables the compressed kernel entry code (part of kernel zImage) to determine the address to uncompress the kernel depending upon the location it is loaded (started from).

Subsequent sub-sections assume that you have already downloaded and extracted the kernel snapshot from above mentioned link/page or set up git working directory with latest PSP kernel from repository associated with the above page. It is also assumed that the environment for kernel build is set up and 'mkimage' executable utility from U-Boot is available in the PATH (this utility gets built with U-Boot and available inside 'tools' directory form U-Boot source base).

Configuring the Kernel

Note: The run time physical offset support is treated as EXPERIMENTAL feature currently. The default kernel configurations for TI816X EVM have this feature enabled in latest source. You can follow steps similar to those described below and build the kernel to enable this support for any custom board as well as to view/disable this feature as per the need.

Generate default config for the Board

For TI816X EVM:

```
BUILDHOST$ make ARCH=arm ti8168_evm_defconfig
```

For TI814X EVM:

```
BUILDHOST$ make ARCH=arm ti8148_evm_defconfig
```

For TI813X EVM:

```
BUILDHOST$ make ARCH=arm dm385_evm_defconfig
```

The above steps should configure the kernel to include run time physical to virtual translation support (CONFIG_ARM_PATCH_PHYS_VIRT=y) and auto detection of load address (CONFIG_AUTO_ZRELADDR=y).

If you desire to view the exact configuration options to be controlled to enable or disable these features, read the next steps.

Enter the configuration menu

```
BUILDHOST$ make ARCH=arm menuconfig
```

You will be presented with configuration menu as shown below:

```
[*] Patch physical to virtual translations at run time (EXPERIMENTAL)
General setup  --->
[*] Enable loadable module support  --->
...
Kernel Features  --->
Boot options  --->
```

Enable/Disable runtime physical to virtual translation feature

Ensure that the "Patch physical to virtual translations..." option is highlighted and press 'n' key (without quotes) to disable it (the default kernel configuration done in the earlier step had enabled this option).

```
[ ] Patch physical to virtual translations at runtime (EXPERIMENTAL)
General setup  --->
[*] Enable loadable module support  --->
...
```

Pressing 'y' on this option will (again) enable it.

Enable/Disable auto calculation of decompressed kernel location

Now navigate to "Boot options --->" option by using DOWN arrow key and press SPACE to enter the sub-menu shown below.

Here, using DOWN ARROW key, navigate to the option "Auto calculation..." and press 'y' to enable or 'n' to disable it. You should see a '*' in the box '[ ]' in front of that option when 'y' is pressed (enable) otherwise the box will be empty (disabled) when 'n' is pressed.

```
(0) Compressed ROM boot loader base address
(0) Compressed ROM boot loader BSS address
()  Default kernel command string
[ ] Kernel Execute-In-Place from ROM
[ ] Kexec system call (EXPERIMENTAL)
[*] Auto calculation of the decompressed kernel image address
```

Using RIGHT arrow key, highlight the 'Exit' option on the menu page and press ENTER key subsequently till you are asked to save the configuration. Now ensure that 'Yes' is highlighted and press ENTER again to save and exit the configuration.

Building the Kernel

Use following command to build the kernel:

```
BUILDHOST$ make ARCH=arm zImage
```

Notice that we are building the standard zImage (compressed kernel image) instead of usual uImage. This is so because the uImage built as part of kernel build will have the load address and entry point @0x80008000 by default while, in this case we desire to load the kernel at different load address. Hence we build a zImage and then pass the desired address to 'mkimage' to get uImage as shown below.

Here we use 0x90008000 as the kernel load address and entry point.

Note: You can chose any other address as per your requirement and available RAM space but please refer next section **Points to Note/Restrictions** for various important points and constraints.

Use 'mkimage' to generate uImage. Note that the command is run from the kernel source base after zImage is built successfully.

```
BUILDHOST$ mkimage -A arm -O linux -T kernel -C none -a 0x90008000 -e 0x90008000 -n 'Linux-2.6.37' -d arch/arm/boot/zImage uImage
    Image Name:   Linux-2.6.37
    Created:      Mon Oct 29 21:05:21 2011
    Image Type:   ARM Linux Kernel Image (uncompressed)
    Data Size:    2570504 Bytes = 2510.26 kB = 2.45 MB
    Load Address: 90008000
    Entry Point:  90008000
```

You can build the zImage once and just execute the 'mkimage' step for different load address requirements with same kernel.

Alternate Method: Single Step Build

You can still achieve the above in single step as part of kernel build by passing `zreladdr-y=<desired-load-address>` to `make` when building the uImage.

For example, to build the uImage with load address 0x90008000 considered in earlier case:

```
BUILDHOST$ make ARCH=arm zreladdr-y=0x90008000 uImage
```

The above command will build the kernel zImage and then uImage by internally invoking 'mkimage' with the specified load address (passed as zreladdr-y).

Note: In any case, you need to rebuild the uImage for specific load address.

Points to Note/Restrictions

- Load address and entry point parameter passed to 'mkimage' must be same, that is, values passed for '-a' and '-e' must be same.
- The auto address calculation of load address is only possible when using compressed kernel image (zImage) and thus, the uImage must be built from zImage - using uncompressed kernel image (Image) will not support this feature.
- The run time physical RAM base calculation support (called as p2v) adds a string "p2v8" to module version and thus the modules built without p2v support cannot be loaded on the kernel built with p2v support enabled and vice verse. This prevents incompatible modules being loaded mistakenly.
- This feature may not be used with XIP kernels.
- Run time physical offset determination feature is an EXPERIMENTAL feature presently.
- The kernel compressed entry code which determines the address for loading uncompressed kernel considers that the compressed kernel image (zImage) is loaded within first 128MB of the RAM, this also means that the RAM base is determined by masking least significant (LS) 27 bits of the load address.
- There must be at least 32KiB space available *after* the determined RAM base (see above point) and *before* the kernel load address specified when preparing uImage.

Refer **Choosing Load Address and Determination of RAM Base** section below for details and examples.

Choosing Load Address and Determination of RAM Base

As mentioned earlier, the load address must be chosen such that it leaves at least 32KB space below it from the RAM base. This is required since the kernel uses this region of RAM below the image to store page tables (in normal execution as well as during decompress).

Also note that since the RAM base is calculated by truncating to the nearest 128MiB boundary (masking LS 27 bits of the load address), the LS bits getting masked need to evaluate to at least 32K offset from the RAM base (that is, 0x8000 minimum).

- Following are some examples of correct load addresses (with automatically calculated RAM base) followed by a few incorrect ones:

Correct Load Addresses

Load Address	Calculated RAM Base
0x90008000	0x90000000
0x90010000	0x90000000
0x91000000	0x90000000
0x98100000	0x98000000
0x98104000	0x98000000
0x80008000	0x80000000
0x88100000	0x88000000
0x86008000	0x80000000

- Following addresses would result into a non bootable kernel (kernel will hang/halt during boot):

Incorrect Load Addresses

Load Address	Problem Area
0x90000000	Calculated RAM base = 0x90000000 (masking to 0xF8000000), which means the 32KiB requirement for load address is not met
0x98000000	RAM base = 0x98000000, 32KiB requirement not met
0x98004000	RAM base = 0x98000000, only 16KiB available below the load address

Modifying Pin Mux settings

TI816X

On TI816X device, the default pin multiplexing is set to Mode 0 (FUNCTION 1) configuration on reset. If you desire to use a particular pin to any other function than Mode 0 or override any pin mode which was already set in U-Boot, the kernel needs to be modified and rebuilt.

You can change default mux mode by adding specific mux entry in the beginning of `board_mux` array in `arch/arm/mach-omap2/board-ti8168evm.c` or calling `omap_mux_init_signal()` during initialization (e.g., in device specific initialization function called from `omap2_init_devices()` in `arch/arm/mach-omap2/devices.c`).

The names of multiplexed signals are specified in arch/arm/mach-omap2/mux81xx.c file in kernel source directory.

e.g., To set up `mtsi_dclk` pin (mode 0) to `gmii1_rxclk` (mode 1) on TI816X, add

```
TI816X_MUX(MTSI_DCLK,OMAP_MUX_MODE1 | OMAP_PULL_UP)
```

to `board_mux` structure in board file or call

```
omap_mux_init_signal("gmii1_rxclk", OMAP_PULL_UP)
```

from module specific init function (in `arch/arm/mach-omap2/devices.c`). Build the kernel after this change.

To disable pull up/down you will need to set bit-3, e.g., use

```
omap_mux_init_signal("gmii1_rxclk", 1 << 3)
```

One could also pass mode0_name.desired_mode_name format string to set specific pin to desired functionality:

```
omap_mux_init_signal("mtsi_dclk.gmii1_rxclk", 1 << 3)
```

The API approach is useful if you desire to set the pin-mux run time depending upon the board/hardware detected without need of maintaining separate kernel binaries.

Alternatively, you can set particular pins by passing respective pinmux details to kernel command line in following format

```
omap_mux=<mode0_name>.<signal_name>=<value>,<mode0_name>.<signal_name>=<value>
```

E.g., to set tsi1_dclk (mode 0) pin to vout1_b_cb_c3 and disable pull up/down (set bit 3), append following to kernel command line passed from the boot loader:

```
omap_mux=tsi1_dclk.vout1_b_cb_c3=0x8
```

You can pass configuration for multiple pins separated by comma.

For details about this boot parameter, refer Documentation/kernel-parameters.txt in kernel source directory.

TI814X/TI813X

On TI814X/TI813X devices, the pins are tri-stated and set to Mode 0 or any other mode as required for specific module (e.g., MMC) in U-Boot. If you desire to use a particular pin to any other function than Mode 0 or override any other pin mode which was already set in U-Boot, the kernel needs to be modified and rebuilt.

You can change default mux mode by adding specific mux entry in the beginning of `board_mux` array in `arch/arm/mach-omap2/board-ti8148evm.c` or calling `omap_mux_init_signal()` during initialization (e.g., in device specific initialization function called from `omap2_init_devices()` in `arch/arm/mach-omap2/devices.c`).

The names of multiplexed signals are specified in arch/arm/mach-omap2/mux814x.c file in kernel source directory.

e.g., for setting `xref_clk0` pin (mode 0) to `usb1_drvvbus` (mode 7 or FUNCTION 8), add

```
TI814X_MUX(XREF_CLK0, OMAP_MUX_MODE7)
```

to `board_mux` structure in board file or call

```
omap_mux_init_signal("xref_clk0.usb1_drvvbus", 0)
```

Note: The string passed above should be mode0_name.desired_mode_name format.

Note: On TI813X devices,

- some pins do not have mode0, in such case, the mode0 name for corresponding pin of TI814X is retained and suffixed with "_ti813x" to be accessible in the mode0.mode format.
- some pins have some extra modes/functions added to existing modes on dm814x or replaced with new functions in such cases the pin mode0 name is suffixed with "_ti813x" to differentiate from the existing pin.

Refer arch/arm/mach-omap2/mux814x.c file for specific details.

e.g, omap_mux_init_signal("xref_clk2_ti813x.sata_act1_led_mux0", 0);

The API approach is useful if you desire to set the pin-mux run time depending upon the board/hardware detected without need of maintaining separate kernel binaries.

Note: On TI814X/TI813X devices, the bit positions for pull up/down are different and hence use macros TI814X_PULL_DIS (to disable pull up/down) or TI814X_PULL_UP to select pull up.

Alternatively, you can set particular pins by passing respective pinmux details to kernel command line in following format

```
omap_mux=<mode0_name>.<signal_name>=<value>,<mode0_name>.<signal_name>=<value>
```

E.g., to set mmc1_cmd_mux0 (mode 0) pin to gpio0_0 and enable pull down, append following to kernel command line passed from the boot loader:

```
omap_mux=mmc1_cmd_mux0.gpio0_0=0
```

For pull up (set bit 17):

```
omap_mux=mmc1_cmd_mux0.gpio0_0=0x20000
```

You can pass configuration for multiple pins separated by comma.

For details about this boot parameter, refer Documentation/kernel-parameters.txt in kernel source directory.

Distinguish between chip revisions run time

TI814X

Sometimes a kernel module/driver may require to do specific handling depending upon TI814X chip revision. To achieve this, the exported kernel API omap_rev() can be used as shown in following example:

- Ensure that plat/cpu.h file is included in your driver file

```
#include<plat/cpu.h>
```

- Use following kind of block to handle specific configuration for chip revisions which or PG 2.0 and onwards:

```
...
if (omap_rev() != TI8148_REV_ES1_0)
 /* Handle non PG 1.0 device configurations */
else
 /* Do the configuration for PG 1.0 devices */
...
```

You can find all the supported chip revision macros by searching for "TI8148_REV_ES" (without quotes) in arch/arm/plat-omap/include/plat/cpu.h file from kernel source.

Using The Correct Console Device

TI816X

On the latest kernel packaged in this release, the serial device names are changed from ttySx to ttyOx. Thus, the serial port associated with UART2 is now referred as ttyO2.

- This means you will need to update 'bootargs' as passed to kernel to use ttyS2 as console. For example, change

```
console=ttyS2,115200n8
```

to,

```
console=ttyO2,115200n8
```

- The subsequent example 'bootargs' use the new console name and will not work with kernel from older releases.
- Similarly, you will need to update the `/etc/inittab` file in the filesystem used for kernel by replacing lines with 'ttySx' by 'ttyOx'. For example, replace following

```
S:2345:respawn:/sbin/getty 115200 ttyS2
```

by,

```
S:2345:respawn:/sbin/getty 115200 ttyO2
```

TI814X, TI813X

On the latest kernel packaged in this release, the serial device names are changed from ttySx to ttyOx. Thus, the serial port associated with UART0 is now referred as ttyO0.

- This means you will need to update 'bootargs' as passed to kernel to use ttyS0 as console. For example, change

```
console=ttyS0,115200n8
```

to,

```
console=ttyO0,115200n8
```

- The subsequent example 'bootargs' use the new console name and will not work with kernel from older releases.
- Similarly, you will need to update the `/etc/inittab` file in the filesystem used for kernel by replacing lines with 'ttySx' by 'ttyOx'. For example, replace following

```
S:2345:respawn:/sbin/getty 115200 ttyS0
```

by,

```
S:2345:respawn:/sbin/getty 115200 ttyO0
```

Note: The character 'O' in the serial port names stands for "**O**MAP UART Port".

Updating the DHCP Script

The latest kernel has changed the entry for root filesystem name as appear in /proc/mounts file. This might impact boot process if some of the init scripts in your filesystem read this entry to determine the root device type.

The existing scripts checking /proc/mounts may not work as intended.

One such case is the /etc/udhcpc.d/50default script in the filesystem, which skips sending DHCP requests if the root device is a network mounted filesystem.

If you face any issue where the filesystem is mounted successfully but hangs after showing udhcpc related prints. The issue is most likely with incompatibility between latest kernel and the udhcpc script.

To fix this, open the /etc/udhcpc.d/50default in your TI81XX filesystem and check if a check like the one shown below exists:

```
root_is_nfs() {
      grep -qe '^/dev/root.*\(nfs\|smbfs\|ncp\|coda\) .*' /proc/mounts
}
```

Update the above fragment by replacing the check as below:

```
root_is_nfs() {
      grep -qe 'nfs\|smbfs\|ncp\|coda.*' /proc/mounts
}
```

The above change will be required even when you want to (manually) run 'udhcpc' client after booting.

Booting Linux Kernel

TI816X

Kernel along with root filesystem can either be booted from on board storage device or can be fetched over the Ethernet to RAM using TFTP and booted from there. Also, the root filesystem can be formatted as JFFS2 or UBIFS, flashed and then mounted. Please refer U-Boot User Guide for details about flashing and supported storage devices.

Following sections describe various kernel boot options possible.

Note: The offsets and MTD partition numbers used in examples below may vary depending upon actual partition layout used on particular storage device. Also, selecting multiple storage device support in kernel (e.g., NAND & SPI) may change the effective partition number to be used as root partition.

Boot from NAND

Make sure the Boot Mode/Configuration Select Switch is set for the NAND boot mode as described in the **U-Boot User Guide**

Power on EVM and wait for U-Boot to come up on the serial console.

When kernel uImage and JFFS2 or UBIFS filesystem are flashed on the NAND device:

```
TI8168_EVM# nand read.i 0x81000000 280000 500000


For JFFS2 file system:
TI8168_EVM# setenv bootargs 'mem=128M console=ttyO2,115200n8 noinitrd root=/dev/mtdblock7 rw rootfstype=jffs2 ip=dhcp'


For UBIFS file system:
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 noinitrd ip=off mem=256M rw ubi.mtd=7,2048 rootfstype=ubifs
root=ubi0:rootfs init=/init'


TI8168_EVM# bootm 0x81000000
```

When kernel image is flashed on the NAND device, and NFS mounted filesystem is being used:

```
TI8168_EVM# nand read.i 0x81000000 280000 500000
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs nfsroot=172.24.179.98:/nfs_root,nolock rw mem=128M'
TI8168_EVM# bootm 0x81000000
```

Boot from NOR

Make sure the Boot Mode/Configuration Select Switch is set for the NOR boot mode as described in **U-Boot User Guide**.

Power on EVM and wait for U-Boot to come up on the serial console.

Assuming kernel uImage and JFFS2 filesystem are flashed on the NOR device @0x08060000 (Partition 2) and @0x08460000 (Partition 3) respectively:

```
TI8168_EVM# cp.w 0x08060000 0x81000000 0x200000
TI8168_EVM# setenv bootargs 'mem=128M console=ttyO2,115200n8 noinitrd root=/dev/mtdblock3 rw rootfstype=jffs2 ip=dhcp'
TI8168_EVM# bootm 0x81000000
```

Boot from SPI

Make sure the Boot Mode/Configuration Select Switch is set for the SPI boot mode as described in **U-Boot User Guide**.

Power on EVM and wait for U-Boot to come up on the serial console.

Assuming kernel uImage is flashed on the SPI flash @0x42000 and using NFS based root filesystem:

```
TI8168_EVM# sf read 0x81000000 0x42000 0x200000
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs nfsroot=172.24.179.98:/nfs_root,nolock rw mem=128M'
TI8168_EVM# bootm 0x81000000
```

Boot from SD Card

Make sure the Boot Mode/Configuration Select Switch is set for the SD boot mode as described in **U-Boot User Guide**.

Power on EVM and wait for U-Boot to come up on the serial console.

The example below assumes kernel uImage is available in first partition of the SD card and second partition contains ext3 formatted filesystem. Please refer "Setting Up Boot Environment on SD Card" section from TI81XX U-Boot User Guide.

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# fatload mmc 0 0x81000000 uImage
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/mmcblk0p2 mem=128M rootwait'
TI8168_EVM# bootm 0x81000000
```

Boot over Network (Ethernet)

When kernel image and ramdisk image are fetched from a tftp server:

- Ensure that the EVM is connected to network with DHCP and TFTP server set up
- Set 'ethaddr' U-Boot environment variable with proper ethernet address in format 'xx:xx:xx:xx:xx:xx' (replace 'xx' with proper hexadecimal values)
- Copy kernel image and ramdisk to TFTP server's root directory.
- Execute follwing commands at U-Boot prompt. We assume kernel image name as 'uImage' and ramdisk file name as 'ramdisk.gz'

```
TI8168_EVM# setenv autoload no
TI8168_EVM# dhcp
TI8168_EVM# setenv serverip <Server IP Address>
TI8168_EVM# tftp 0x81000000 uImage
TI8168_EVM# tftp 0x82000000 ramdisk.gz
TI8168_EVM# setenv bootargs 'mem=200M console=ttyO2,115200n8 root=/dev/ram0 initrd=0x82000000,40M ramdisk_size=45000 ip=dhcp'
TI8168_EVM# bootm 0x81000000
```

- Alternatively, kernel can be made to use the same IP address as assigned to U-Boot instead of doing DHCP request again by setting U-Boot parameters as follows:

```
TI8168_EVM# print ethaddr                          <-- Check if MAC address is assigned and is unique
TI8168_EVM# setenv ethaddr <unique-MAC-address>    <-- Set only if not present already, format xx:yy:zz:aa:bb:cc
TI8168_EVM# setenv bootcmd 'dhcp;run addip; tftp 81000000 uImage;bootm'
TI8168_EVM# setenv hostname <unique-hostname>
TI8168_EVM# setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${nfsserver}:${gatewayip}:${netmask}:${hostname}:eth0:off'
TI8168_EVM# setenv autoload no
```

```
TI8168_EVM# setenv nfsserver <nfs-server-ip>         <-- Make sure the same NFS server IP is used below
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs nfsroot=<nfs-server-ip>:<path-to-nfs-share>,nolock rw mem=128M'
TI8168_EVM# setenv serverip <tftp-server-ip>
TI8168_EVM# saveenv
TI8168_EVM# boot
```

- After saving the environment variables, you need not set them again on reboot unless a change is required.
- Note that the above example uses NFS mounted root file system accessed over 'eth0' interface (as available on the base EVM)
- You will need to use 'eth1' instead of 'eth0' in case the 2nd Ethernet interface is used to connect to the network (e.g., on daughter card). For example, update 'addip' in the example above,

```
TI8168_EVM# setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${nfsserver}:${gatewayip}:${netmask}:${hostname}:eth1:off'
```

Note: You need not set 'ethaddr' for devices having valid MAC IDs set. In such cases, U-Boot will automatically detect and set the ethernt address (should show message like "Detected MACID:...").

Setting Memory Holes For System RAM

The default kernel configuration support setting up holes in system RAM. That is, the bootargs can be set to indicate kernel to map RAM regions with hole in between.

As an example, the bootargs used for network boot as shown above are modified to have two RAM regions for kernel mapping with 32MB starting form 0x80000000 followed by 96MB from 0x88200000. Note that, this change can be used for any of the other boot modes described above with 'mem' arguments as shown below:

```
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs
nfsroot=<nfs-server-ip>:<path-to-nfs-share>,nolock rw
mem=32M@0x80000000 mem=96M@0x88200000'
```

TI814X, TI813X

Kernel along with root filesystem can either be booted from on board storage device or can be fetched over the Ethernet or UART to RAM using TFTP or serial protocols like YMODEM and booted from there. Also, the root filesystem can be formatted as JFFS2 or UBIFS, flashed and mounted. Please refer to the U-Boot User Guide for details about flashing and supported storage devices.

Following sections describe various kernel boot options possible.

- **Note:** The offsets and MTD partition numbers used in examples below may vary depending upon actual partition layout used on particular storage device. Also, selecting multiple storage device support in kernel (e.g., NAND & SPI) may change the effective partition number to be used as root partition.

Boot from NAND

Make sure the Boot Mode/Configuration Select Switch is set for the NAND boot mode as described in **U-boot UserGuide** section.

Power on EVM and wait for U-Boot prompt of the 2nd stage (TI8148_EVM#) to come up on the serial console.

When kernel uImage and JFFS2 or UBIFS filesystem are flashed on the NAND device:

```
TI8148_EVM# nand read.i 0x81000000 280000 500000


For JFFS2 file system:
TI8148_EVM# setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd root=/dev/mtdblock9 rw rootfstype=jffs2 ip=dhcp'
```

```
For UBIFS file system:
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 noinitrd mem=256M rw ubi.mtd=9,2048 rootfstype=ubifs
root=ubi0:rootfs init=/init'

TI8148_EVM# bootm 0x81000000
```

When kernel image is flashed on the NAND device, and NFS mounted filesystem is being used:

```
TI8148_EVM# nand read.i 0x81000000 280000 500000
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 root=/dev/nfs nfsroot=172.24.179.98:/nfs_root,nolock rw mem=128M'
TI8148_EVM# bootm 0x81000000
```

Boot from SPI

Make sure the Boot Mode/Configuration Select Switch is set for the SPI boot mode as described in **U-boot UserGuide** section.

Power on EVM and wait for U-Boot prompt of the 2nd stage (TI8148_EVM#) to come up on the serial console.

Assuming kernel image is flashed on the SPI flash @0x42000 and NFS based root filesystem is used:

```
TI8148_EVM# sf read 0x81000000 0x42000 0x200000
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 root=/dev/nfs nfsroot=172.24.179.98:/nfs_root,nolock rw mem=128M'
TI8148_EVM# bootm 0x81000000
```

Boot from NOR

Make sure the Boot Mode/Configuration Select Switch is set for the NOR boot mode as described in **U-boot UserGuide** .

Power on EVM and wait for U-Boot to come up on the serial console.

Assuming kernel uImage and JFFS2 filesystem are flashed on the NOR device @0x08060000 (Partition 2) and @0x08460000 (Partition 3) respectively:

```
TI8168_EVM# cp.w 0x08060000 0x81000000 0x200000
TI8168_EVM# setenv bootargs 'mem=128M console=ttyO2,115200n8 noinitrd root=/dev/mtdblock3 rw rootfstype=jffs2 ip=dhcp'
TI8168_EVM# bootm 0x81000000
```

Boot over Network (Ethernet)

Note

When setting a MAC address please ensure that the LS-bit of the 1st byte is not 1 i.e. when setting the MAC address: **x**y in x**y**:ab:cd:ef:gh:jk has to be an even number. For more info this refer to the wiki page http://en.wikipedia.org/wiki/MAC_address

When kernel image and ramdisk image are fetched from a TFTP server:

- Ensure that the EVM is connected to network with DHCP and TFTP server set up
- Set 'ethaddr' U-Boot environment variable with proper ethernet address in format 'xx:xx:xx:xx:xx:xx' (replace 'xx' with proper hexadecimal values)
- Copy kernel image and ramdisk to TFTP server's root directory.
- Ensure that the "Options Negotiation" box in the tftp server settings is not checked.
- Execute following commands at U-Boot prompt. We assume kernel image name as 'uImage' and ramdisk file name as 'ramdisk.gz'

```
TI8148_EVM# setenv autoload no
TI8148_EVM# dhcp
```

```
TI8148_EVM# setenv serverip <Server IP Address>
TI8148_EVM# tftp 0x81000000 uImage
TI8148_EVM# tftp 0x82000000 ramdisk.gz
TI8148_EVM# setenv bootargs 'mem=200M console=ttyO0,115200n8 root=/dev/ram0 initrd=0x82000000,40M ramdisk_size=32768 ip=dhcp'
TI8148_EVM# bootm 0x81000000
```

- Alternatively, kernel can be made to use the same IP address as assigned to U-Boot instead of doing DHCP request again by setting U-Boot parameters as follows:

```
TI8148_EVM# print ethaddr                          <-- Check if MAC address is assigned and is unique
TI8148_EVM# setenv ethaddr <unique-MAC-address>    <-- Set only if not present already, format xn:yy:zz:aa:bb:cc
TI8148_EVM# setenv bootcmd 'dhcp;run addip; tftp 81000000 uImage;bootm'
TI8148_EVM# setenv hostname <unique-hostname>
TI8148_EVM# setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${nfsserver}:${gatewayip}:${netmask}:${hostname}:eth0:off'
TI8148_EVM# setenv autoload no
TI8148_EVM# setenv nfsserver <nfs-server-ip>         <-- Make sure the same NFS server IP is used below
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 root=/dev/nfs nfsroot=<nfs-server-ip>:<path-to-nfs-share>,nolock rw mem=128M'
TI8148_EVM# setenv serverip <tftp-server-ip>
TI8148_EVM# saveenv
TI8148_EVM# boot
```

- After saving the environment variables, you need not set them again on reboot unless a change is required.
- Note that the above example uses NFS mounted root file system accessed over 'eth0' interface (as available on the base EVM)

Note: You need not set 'ethaddr' for devices having valid MAC IDs set. In such cases, U-Boot will automatically detect and set the ethernt address (should show message like "Detected MACID:...").

Power Management

NOTE: For an overview of PM features supported, planned and implemented on each platform refer to **PSP Power management FAQ**

TI816X

Clock details

H/W Details

TI816X has two on chip oscillators which are the main source of clock, OSC0 and OSC1. By default OSC0(27 MHz) provides the input clock source for all FAPLLs.These PLLs are configured to generate higher frequencies required for interface and functional clocks of various modules on the device.Clock out from PLLs is distributed to various modules as per the hardware connections and requirements, with divider at place where lower frequencies are required.

Clock Framework Overview and Usage

For clock framework software implementation, features and usage refer to **Clock Framework User Guide**

List of available clocks

For list of available clocks,

- Refer to ti816x_clks[] in arch/arm/mach-omap2/clock816x_data.c. All the clocks with the flag CK_TI816X are available on this platform.
- Refer to "Feature performance guide" for list of clocks and their default rate/usecount.
- Run the clock browser script available here Browse clocks

Adaptive Voltage Scaling

SmartReflex-AVS is a technology that uses adaptive power supply to achieve the goal of reducing active power consumption. TI816X device has Class 2B implementation of SmartReflex and this allows dynamic AVS using Software. For more details on software implementation refer to **AVS driver User Guide**

TI814X

Note: On PG 1.X revisions of TI814X chip, only clock framework is supported, other PM features are not supported.

Clock details

H/W Details

TI814X has two on chip oscillators which are the main source of clock, OSC0 - 20 MHz and OSC1(20-30MHz). Centaurus has 13 PLLs, by default OSC0(20 MHz) provides the input clock source for all PLLs.These PLLs are configured to generate higher frequencies required for interface and functional clocks of various modules on the device.Clock out from PLLs is distributed to various modules as per the hardware connections and requirements, with divider at place where lower frequencies are required.

Clock Framework Overview and Usage

For clock framework software implementation, features and usage refer to **Clock Framework User Guide**

List of available clocks

For list of available clocks,

- Refer to ti814x_clks[] in arch/arm/mach-omap2/clock814x_data.c. All the clocks with the flag CK_TI814X are available on this platform.
- Refer to "Feature performance guide" for list of clocks and their default rate/usecount.
- Run the clock browser script available here Browse clocks

Suspend to Memory

NOTE: Lowest 1KB(1024 Bytes from the end) of OCMC RAM is used by Suspend code to execute from OCMC when DDR is placed in Self-refresh mode during suspend/resume.This memory area will be over-written by Suspend code when kernel is loaded, during SRAM initialization so drivers should avoid using the lowest 1KB of OCMC RAM

Suspending a system drives the system to a low power state there by saving power as and when desired by the user. For overview of Suspend to memory implementation, features and usage refer to **Suspend Resume User Guide**

DVFS

Note: For Release 04.04.00.01: A fix for frequency scaling of ARM not working issue(SDCM00089416 [2]) is available as a patch over the release package for 04.04.00.01. This can be downloaded from ARM PLL frequency change fix [3]

DVFS is supported on TI814x from 04.01.00.07 release onwards. For overview and detailed usage refer to **DVFS User Guide**

Supported OPPs

- Currently OPP list/ frequency table is populated with "mpu_vdd/ARM_VDD" data alone.
- Below table shows current **software Configuration**

	OPP100	OPP160
frequency(MHz)	600	1000
voltage (V)	1.10	1.35
Enabled	yes	yes

NOTES:

- If the user wants to **disable** any OPP or **change** the voltage, it can be done by editing following file:

```
arch/arm/mach-omap2/opp814x_data.c
```

- Rebuild the Kernel after modifying above file.

TI813X

Clock details

H/W Details

TI813X has two on chip oscillators which are the main source of clock, OSC0 - 20 MHz and OSC1(20-30MHz). Centaurus has 11 PLLs, by default OSC0(20 MHz) provides the input clock source for all PLLs.These PLLs are configured to generate higher frequencies required for interface and functional clocks of various modules on the device.Clock out from PLLs is distributed to various modules as per the hardware connections and requirements, with divider at place where lower frequencies are required.

Clock Framework Overview and Usage

For clock framework software implementation, features and usage refer to **Clock Framework User Guide**

List of available clocks

For list of available clocks,

- Refer to ti814x_clks[] in arch/arm/mach-omap2/clock814x_data.c. All the clocks with the flag CK_TI813X are available on this platform.
- Refer to "Feature performance guide" for list of clocks and their default rate/usecount.
- Run the clock browser script available here Browse clocks

Suspend to Memory

NOTE: Lowest 1KB(1024 Bytes from the end) of OCMC RAM is used by Suspend code to execute from OCMC when DDR is placed in Self-refresh mode during suspend/resume.This memory area will be overwritten by Suspend code when kernel is loaded, during SRAM initialization so drivers should avoid using the lowest 1KB of OCMC RAM

Suspending a system drives the system to a low power state thereby saving power as and when desired by the user. For overview of Suspend to memory implementation, features and usage refer to **Suspend Resume User Guide**

GPIO Driver

TI816X

TI816X has two GPIO modules each provides 32 dedicated general-purpose pins with input and output capabilities, total 0 - 63 pins are available for usage.

TI814X

TI814X has four GPIO modules each provides 32 dedicated general-purpose pins with input and output capabilities, total 0 - 127 pins are available for usage.

- **GPIO Driver Guide** have more details of driver usage

DCAN Driver

DCAN functionality is available only for TI814X and TI813X. Driver details can be found at DCAN Driver Guide.

- **DCAN Driver User Guide**

EDMA Driver

For details on EDMA support refer to **EDMA Driver User Guide**

Audio Driver

The audio driver in the PSP package conforms to the ASoC framework in the Linux kernel. The current driver supports audio capture and playback using the AIC3106 codec on the EVM. For more details on the audio driver refer **Audio User Guide**.

SATA Driver

The SATA subsystem supports 2 - 3Gbps SATA host ports each capable of supporting Port Multiplier and direct connect SATA devices.

SATA driver in the Linux kernel has support for

- HDD
- CD/DVD
- Port Multiplier

NOTE: In case of external RAID storage towers (via ahci), the storage tower should be configured using the windows/Linux(x86) tool before connecting it to TI8168 EVM

Feature Not Supported

- Power Management

Note: Refer **SATA FAQ** for more information.

USB Driver

The USB subsystem includes two USB(mentor) controller. There are independent usb ports for two controller musb0 and musb1 can be configured to host/device mode operation. The PG 1.0 version of **TI816x** silicon supports only point-point configuration in Mentor USB IP core and hence hub is not supported. Hence only one device can be connected to each port of USB.

USB_ID configuration for TI816X: The USB0_ID/USB1_ID pin is always configured through software in TI816X silicon revisions. This configuration is done by chosing appropriate configuration through USBx_ID menuconfig selection. more information refers to USB Configuration Page [4].

Please refer TI81XX_USB_Driver_User_Guide [5] for more details.

Ethernet Driver

Ethernet driver follows standard Linux Network Interface Architecture.

Ethernet Switch Driver

Ethernet Switch driver follows standard Linux Network Interface Architecture.

Please refer **Ethernet Switch User Guide** for more details on TI814x and TI813x.

VPSS Video Driver

VPSS Video Driver User Guide

VPSS Video Capture driver

VPSS Video Capture Driver User Guide

NOTE: V4L2 capture driver is added in PSP04.00.01.13 for TI816X class of devices and PSP 04.01.00.06 for TI814X class of devices.

HDMI Driver

For information on HDMI usage and support refer to **HDMI Driver User Guide**

Sii9022a External HDMI Transmitter Driver

Information related to features supported by the driver and usage can be found in **Sii9022a external HDMI Transmitter UserGuide**

McSPI Driver

Information related to features supported by the driver and usage can be found in **McSPI Driver Guide**

NOR Flash Support

Information related to features supported by the driver and usage can be found in **NOR Driver User Guide**

SD driver

SD Driver supports SD/SDHC/uSD cards. HSMMC peripheral (and driver) has support for 4 data lines at the max operating frequency of 48MHz. HSMMC is a slave DMA peripheral and uses EDMA to move data between SD card and system memory.

Supported Features

- SD
- SDHC
- uSD
- uSDHC

Not supported Features

- PIO mode of operation

Please refer **MMC/SD guide**.

NAND driver

Information related to features supported by the driver and usage can be found in **NAND guide**

PCI Express Root Complex Driver

Please refer the PCIe RC Driver User Guide **online**.

PCI Express Endpoint Boot Driver

Please refer the PCIe Boot Driver User Guide **online** for details about booting TI811X PCIe Endpoint connected to TI81XX or x86 Linux PCIe Root complex using PCIe boot driver (running on RC).

PCI Express Endpoint Driver

Please refer the PCIe Endpoint Driver User Guide **online**for details about using this driver on TI81XX PCIe Endpoint part of PCIe topology.

Watchdog Timer (WDT)

Information related to features supported by the driver and usage can be found in **WDT guide**.

TILER

TILER driver has dependency on Multimedia. TILER support is disabled in defconfig. TILER driver can be enabled from,

```
Device Drivers  --->
  <*> Multimedia support  --->
      <*>   TI TILER support  --->
              --- TI TILER support
              (128) Allocation granularity (2^n) (NEW)
              (4096) Allocation alignment (2^n) (NEW)
              (40)  Memory limit to cache free pages in MBytes (NEW)
              (1)   Process security (NEW)
              (0)   Use SSPtr for id (NEW)
```

```
                [ ]   Secure TILER build (NEW)
                [*]   Expose SSPtr to userspace (NEW)
```

Leave the default values as those are the one's which were validated for this release.

Following features are supported in the current TILER driver,

- Single PAT and run-time memory allocation

Features not supported,

- Dual PAT
- PAT by-pass mode

IOMMU

For features and usage information please refer to **IOMMU Driver Guide**.

Note: **IOMMU** is the same as **System MMU** from the Technical Reference Manual.

References

[1] https://sourcery.mentor.com/sgpp/portal/release858
[2] http://processors.wiki.ti.com/index.php/TI81XX_PSP_04.04.00.01_Release_Notes#SDCM00089416_Details.28fixed.29
[3] http://arago-project.org/git/projects/?p=linux-omap3.git;a=commitdiff_plain;h=10d3731856fcfe71bbea3b256248f5d4a0974c03;hp=5b847011dceb8e2d323f28aae4e664c6bef65964
[4] http://processors.wiki.ti.com/index.php/Usbgeneralpage#USB-ID_pin_configuration_selection_for_TI81XX
[5] http://processors.wiki.ti.com/index.php/TI81XX_PSP_USB_Driver_User_Guide

Article Sources and Contributors

TI81XX PSP User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=124127 *Contributors*: BradGriffis, Chanilkumar, Deepu.raj, Hemantp, Mugunthanvnm, Parth.saxena, RK, Ravibabu31, SriramAG, Yihe

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

License

License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.