# Community Linux PSP for DaVinci devices

# Community Linux PSP for DaVinci devices


## Document License



## PSP Overview

Linux Platform Support Package (PSP) provides support for Linux kernel, U-Boot, UBL and utilities to flash boot software on the EVM. The latest PSP package can be obtained from TI's Technology and Software Publicly Available (TSPA) download site [1]. This package includes the following components:

**Note:** Specific version information of each component is included in the PSP Release Notes for the release. PSP drivers features and performance guide provides an overview, features, constraints and performance for each of the drivers included in the PSP release.

- **DaVinci Linux kernel.** This Linux kernel is based on a kernel version available from the DaVinci GIT tree [2]. The pre-built kernel binary included with the PSP package is built with Sourcery G++ Lite 2009q1-203 for ARM GNU/Linux [3] from CodeSourcery [4].
- **U-Boot.** This U-Boot is based on an U-Boot version available from the U-Boot GIT tree [5]. The pre-built U-Boot binary included with this release is built with Sourcery G++ Lite 2009q1-203 for ARM GNU/Linux [3] from CodeSourcery [4]
- **User Boot Loader (UBL).** This is the primary boot software which copies U-Boot to external RAM and starts it. This software requires Code Composer Studio (CCStudio) version 5.0 for building it.
- **Flash writers.** This software requires Code Composer Studio (CCStudio) version 5.0 for building.

You can copy any portions of the PSP that need to be run on a Microsoft Windows host (such as CCStudio projects for building the UBL and flash writers) to a Microsoft Windows host. Alternately, you can install the PSP package in a disk partition that can be accessed both from a Microsoft Windows and Linux host.

Most of the components of the PSP need to be untar-ed or extracted in order to be used. The following files and directories are provided in the PSP:

```
----DaVinci-PSP-SDK-#.#.#.#
|-- Software-manifest.html
|-- docs
|   |-- FeaturesPerformanceGuide-#.#.#.#.pdf
|   |-- GPLv2.pdf
|   |-- ReleaseNotes-#.#.#.#.pdf
|   `-- UserGuide-#.#.#.#.pdf
|-- host-tools
|   |-- linux
|   |-- src
|   `-- windows
```

```
|-- images
|   |-- boot-strap
|   |   |-- ccs
|   |   |   |-- dm36x
|   |   |   |   |-- NANDWriter_DM36x.out
|   |   |   |   `-- NANDEraser_DM36x.out
|   |   |   `-- omapl1x8
|   |   |   |   |-- NANDWriter_ARM.out
|   |   |   |   |-- NORWriter_ARM.out
|   |   |   |   `-- SPIWriter_OMAP-L138.out
|   |   `-- serial_flash
|   |   |   |-- dm36x
|   |   |   |   |-- sfh_DM36x.exe
|   |   |   |   |-- ubl_DM36x_NAND.bin
|   |   |   |   `-- ubl_DM36x_SDMMC.bin
|   |   |   `-- omapl1x8
|   |   |   |   |-- sfh_OMAP-L138.exe
|   |   |   |   |-- ubl_OMAPL138_NAND.bin
|   |   |   |   |-- ubl_OMAPL138_NOR.bin
|   |   |   |   |-- ubl_OMAPL138_SDMMC.bin
|   |   |   |   `-- ubl_OMAPL138_SPI_MEM.bin
|   |-- examples
|   |   |-- dm36x
|   |   |   |-- edma
|   |   |   |   `-- edma_test.ko
|   |   |   |-- fbdev
|   |   |   |   |-- blend
|   |   |   |   |-- fbdev_display_bands
|   |   |   |   `-- fbdev_display_rgb
|   |   |   `-- mc
|   |   |   |   |-- display_lcd
|   |   |   |   |-- mc_enumerate
|   |   |   |   |-- mt9p031_ccdc_aew
|   |   |   |   |-- mt9p031_ccdc_af
|   |   |   |   |-- mt9p031_ccdc_file
|   |   |   |   |-- mt9p031_ccdc_prv_file
|   |   |   |   |-- mt9p031_ccdc_prv_rsz_file
|   |   |   |   |-- mt9p031_ccdc_prv_rsz_loopback
|   |   |   |   |-- mt9p031_ccdc_prv_rsz_loopback_mmap
|   |   |   |   |-- previewer_resizer_ss
|   |   |   |   |-- previewer_ss
|   |   |   |   |-- resizer_ss
|   |   |   |   |-- tvp514x_ccdc_crop_loopback
|   |   |   |   |-- tvp514x_ccdc_file
|   |   |   |   |-- tvp514x_ccdc_loopback
|   |   |   |   |-- tvp514x_ccdc_loopback_mmap
|   |   |   |   |-- tvp514x_ccdc_prv_file
```

```
|   |   |   |   |-- tvp514x_ccdc_prv_rsz_file
|   |   |   |   |-- tvp7002_ccdc_file
|   |   |   |   |-- tvp7002_ccdc_loopback
|   |   |   |   |-- tvp7002_ccdc_loopback_mmap
|   |   |   |   |-- tvp7002_ccdc_prv_file
|   |   |   |   |-- tvp7002_ccdc_prv_rsz_file
|   |   |   |   |-- v4l2_colors
|   |   |   |   |-- v4l2_colors_hd
|   |   |   |   `-- v4l2_display
|   |   `-- omapl1x8
|   |   |   |-- edma
|   |   |   |   `-- edma_test.ko
|   |   |   |-- vpif
|   |   |   |   |-- vpif_mmap_loopback
|   |   |   |   |-- vpif_userptr_loopback_cmem
|   |   |   |   |-- vpif_display
|   |   |   |   |-- vpif_userptr_loopback_sd
|   |   |   |   `-- vpif_mmap_loopback_sd
|   |   |   |-- gpio
|   |   |   |   `-- gpio_test.ko
|   |   |   `-- mcbsp
|   |   |   |   `-- mcbsp_test.ko
|   |-- kernel
|   |   |-- dm36x
|   |   |   |-- modules\
|   |   |   `-- uImage
|   |   |-- omapl1x8
|   |   |   |-- modules\
|   |   |   `-- uImage
|   |-- u-boot
|   |   |-- dm36x
|   |   |   `-- u-boot.bin
|   |   |-- omapl1x8
|   |   |   `-- u-boot.bin
|   `-- utils
|   |   |-- dm36x
|   |   `-- omapl1x8
|-- scripts
`-- src
    |-- boot-strap
    |   `-- flash-utils-#.#.#.#.tar.gz
    |-- examples
    |   `-- examples-#.#.#.#.tar.gz
    |-- kernel
    |   |-- ChangeLog-#.#.#.#
    |   |-- ShortLog
    |   |-- Unified-patch-#.#.#.#.gz
```

```
|   |-- diffstat-#.#.#.#
|   |-- kernel-patches-#.#.#.#.tar.gz
|   `-- linux-#.#.#.#.tar.gz
|-- u-boot
|   |-- ChangeLog-#.#.#.#
|   |-- ShortLog
|   |-- Unified-patch-#.#.#.#.gz
|   |-- diffstat-#.#.#.#
|   |-- u-boot-#.#.#.#.tar.gz
|   `-- u-boot-patches-#.#.#.#.tar.gz
`-- utils
```

## Host platform Requirements

Building and running all of the PSP components requires both a Windows and a Linux machine.

The Windows machine is required for running CCStudio 5.0. CCStudio is required for building the User Boot Loader (UBL) and Flash writers. CCStudio is also used to burn the boot images (UBL, U-Boot) into the flash using the flash writers provided in the PSP package.

Linux host is required:

- for compiling UBL, Serial flash writers, U-Boot and Linux kernel.
- to host the TFTP server required for downloading kernel and file system images from U-Boot using Ethernet.
- to host the NFS server to boot the EVM with NFS as root filesystem

### Host Software Requirements

- CCstudio 5.0 (needed only to rebuild UBL/flash writers or to re-flash UBL/U-Boot image on the EVM)
- TI Code Generation Tools 4.5.1 and above for TMS470Rx (needed only to rebuild UBL/flash writers)
- CodeSourcery tool chain for ARM
- MONO Framework
- Serial console terminal application
- TFTP and NFS servers.

# Getting Started Quickly

Get started with setting up the EVMs for → **OMAP-L138, DA850 or AM18x** or → **DM36x**.

To help you get started quickly, pre-built binaries for the UBL, U-Boot, Linux kernel, and flash writers are provided in the `images` directory under PSP installation.

In order to create your own applications running on Linux or to rebuild U-Boot or the Linux kernel provided with the PSP package, you will need to install the CodeSourcery tools for cross compilation.

## Running PSP Components on OMAP-L138

**Booting U-Boot** provides information on setting up the EVM to boot U-Boot from various boot media.
**Booting the Linux kernel** provides information on booting Linux on the EVM.
**Re-flashing boot images** provides information on re-flashing the boot software (UBL, U-Boot) on the EVM.

### Using Linux Kernel Drivers

→ **Linux drivers usage** has specific usage information on various Linux drivers and features.

**Loading Linux kernel modules** provides information on how to use various kernel features and drivers as loadable kernel modules.

## Running PSP Components on DM36x

**Booting DM36x Out of the Box** provides information on setting up the EVM to boot U-Boot and Linux from various boot media.
**Re-flashing boot images** provides information on re-flashing the boot software (UBL, U-Boot) on the EVM.

## Building PSP Components

**Building Software Components for OMAP-L1** provides procedures for rebuilding the following software components used on the OMAP-L1 processors or to flash software to the board.

- **Linux kernel**
- **U-Boot**
- **ARM User Boot Loader**
- **SPI Flash writer**
- **NAND Flash writer**
- **NOR Flash writer for OMAP-L138**

**Building Software Components for DaVinci devices** provides procedures for rebuilding the following software components used on the DaVinci processors or to flash software to the board.

- **Linux kernel**
- **U-Boot**
- **User Boot Loader for DM36x**
- **NAND Flash writer for DM36x**

**Configuring Linux Kernel** provides information on how to reconfigure the Linux kernel to include and exclude various drivers and kernel features.

## Additional topics

- The **additional procedures** topic documents some additional useful procedures aside from the usual usage procedures.
- The → **Writing V4L2 Media Controller Applications on Dm36x Video Capture** is an introduction to Media Controller and tutorial for writing applications for Video Capture

## References

[1] http://software-dl.ti.com/dsps/dsps_public_sw/psp/LinuxPSP/DaVinci_03_21/03_21_00_03/index_FDS.html
[2] http://git.kernel.org/?p=linux/kernel/git/khilman/linux-davinci.git;a=summary
[3] http://www.codesourcery.com/sgpp/lite/arm/portal/release858
[4] http://www.codesourcery.com
[5] http://git.denx.de/?p=u-boot.git;a=summary

# GSG: Booting the OMAP-L138/AM18x Out of the Box

^ Up to main **Getting Started Guide for OMAP-L1** Table of Contents

^ Up to main **Getting Started Guide for AM18x** Table of Contents

**IMPORTANT**

By default, DA850/OMAP-L138/AM18xx EVM comes up with CPU operating at 300MHz. UBL has to be re-built to boot at a higher CPU frequency. You can also use the Linux CPUFreq driver to change the operating point after Linux has booted up.

## Booting the EVM

**NOTE**: The EVM (beta version and later) is configured at manufacturing to allow the board to boot up Linux out of the box without having to build or install any additional software. For earlier versions of EVM (alpha or pre-alpha), user still needs to flash U-Boot and UBL to flash. Please refer to the topic on flashing the boot images.

Use the following steps to boot the board:

1. Refer to the usage information available on LogicPD [1] website.
2. Make sure that the SOM module is properly plugged into the EVM baseboard.
3. Connect the serial cable included with the kit to the serial port of the EVM board and connect the other end to the serial port of a workstation.
4. Make sure the the boot switch pins on switch S7 are in the correct position:

| S7:8 (BOOT[4]) | S7:7 (BOOT[3]) | S7:6 (BOOT[2]) | S7:5 (BOOT[1]) |
|---|---|---|---|
| OFF | OFF | OFF | OFF |

5. Run a terminal session on your workstation and configure it to connect to the serial port with the following settings
   - Bits per second: 115,200
   - Data bits: 8
   - Parity: None
   - Stop bits: 1
   - Flow control: None

- Transmit delay: 0 msec/char, 100 msec/line

6. Upon powering up the board, you should see U-Boot start up and then messages indicating that the Linux kernel is starting up. When the boot sequence has completed, you should see a login prompt. **NOTE**
   Initial kits that ship may power up to a U-Boot prompt rather than automatically boot Linux. If you have such a kit, you will need to set up U-Boot to boot Linux.
7. Log in as root (no password).
   - At the Linux prompt, you can use existing Linux commands to test various features and use examples from the "examples" folder under the "images" directory.
   - While U-Boot is booting, you can press any key to interrupt the automatic boot of the Linux kernel. This will leave you in U-Boot, where you can type commands in the U-Boot command shell.

**NOTE:** The AM18x EVM / eXperimenter's Kit comes with a demo that should boot out of the box according to the instructions in this video [2]. If you changed the u-boot default environmental variables and want to restore them, please follow the instructions on this page or use the following u-boot commands to set them:

```
setenv bootcmd 'sf probe 0;sf read 0xc0700000 0x80000 0x220000;bootm 0xc0700000'
setenv bootargs mem=32M console=ttyS2,115200n8 root=/dev/mmcblk0p1 rw rootdelay=3 ip=off
setenv bootfile "uImage"
saveenv
```

## What's next?

Please continue on to the Installing the SDK software for OMAP-L1 or to Installing_the_SDK_software for AM18x section of the Getting Started Guide

## References

[1] http://logicpd.com/products/development-kits/zoom-omap-l138-evm-development-kit
[2] http://e2e.ti.com/videos/m/digital_signal_processors/181433.aspx

# Booting DM36x Out of the Box

**NOTE**: The EVM (beta version and later) is configured at manufacturing to allow the board to boot up Linux out of the box without having to build or install any additional software. For earlier versions of EVM (alpha or pre-alpha), user still needs to flash U-Boot and UBL to flash.

Use the following steps to boot the board:

1. Connect the serial cable included with the kit to the serial port of the EVM board and connect the other end to the serial port of a workstation.
2. Make sure the the boot switch pins on switch SW4 are in the correct position:
3. For NAND boot mode:

| S4:1 (BITSEL2) | S4:2 (BITSEL1) | S4:3 (BITSEL0) | S4:4 (AECFG2) | S4:5 (AECFG1) | S4:6 (AECFG0) |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | OFF | OFF |

5. For MMC/SD boot mode:

| S4:1 (BITSEL2) | S4:2 (BITSEL1) | S4:3 (BITSEL0) | S4:4 (AECFG2) | S4:5 (AECFG1) | S4:6 (AECFG0) |
|---|---|---|---|---|---|
| OFF | ON | OFF | OFF | OFF | OFF |

7. Run a terminal session on your workstation and configure it to connect to the serial port with the following settings
   - Bits per second: 115,200
   - Data bits: 8
   - Parity: None
   - Stop bits: 1
   - Flow control: None
   - Transmit delay: 0 msec/char, 100 msec/line
8. Upon powering up the board, you should see U-Boot start up and then messages indicating that the Linux kernel is starting up. When the boot sequence has completed, you should see a login prompt.
9. Log in as root (no password).
   - At the Linux prompt, you can use existing Linux commands to test various features and use examples from the "examples" folder under the "images" directory.
   - While U-Boot is booting, you can press any key to interrupt the automatic boot of the Linux kernel. This will leave you in U-Boot, where you can type commands in the U-Boot command shell.

# GSG: DA8x/OMAP-L1/AM1x DVEVM Additional Procedures

^ Up to main **Getting Started Guide for OMAP-L1** Table of Contents

## Flashing images

### DA850/OMAP-L138/AM18xx

On the OMAP-L138 (or AM18xx) SoC, the ARM boots first. On boot-up, the ARM runs the ARM UBL in AIS file format. The purpose of the ARM UBL is to initialize the PLLs, mDDR, and other hardware. Once done, it copies the U-Boot into mDDR and starts it.

U-Boot is an open source boot loader and is responsible for booting the Linux kernel.

#### Flashing images to SPI Flash

There are two ways to flash images to SPI Flash:

- Serial flasher

1. See this page for instructions on using the command line serial flashing utility, which requires only a UART cable: Serial Boot and Flash Loading Utility for OMAP-L138

- CCS

1. The embedded emulation that comes with the EVM is XDS100 version 1, that does not support the ARM. So to connect to the ARM, an external emulator is necessary. Also, to use an external emulator, you need the full version of CCS, not the one that comes with the evm. If you do not have an external emulator, please use the Serial Flasher above.
2. Set the boot pins to emulation boot mode. This is done by setting switch S7 on the EVM according to the following table:
   For LogicPD EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | OFF | OFF | ON |

   For Spectrum Digital EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | ON | ON | ON |

   - For **03.20.xx.xx** Releases:

   1. Start CCStudio and connect to the ARM. See details at: How to connect to the OMAP-L138/C6748 EVM board using CCS?. Ensure that you have the latest ARM GEL file from [[1]] for LogicPD EVM and [[2]]for Spectrum Digital EVM correctly specified.
   2. Execute the GEL function "Full EVM-->SPI1_PINMUX" incase of LogicPD and "Test_setup-->SPI_setup" on Spectrum Digital. In case of CCSv3, this appears under the `GEL` menu. In case of CCSv4, this appears under `Scripts` menu.
   3. Load the SPI flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x8/` directory of the PSP installation (`spiflash-writer.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer

4. Run the SPI flasher program. You will be prompted for the input file type and the file path. For booting from SPI, an ARM AIS image and U-Boot are required.
   - To burn the ARM AIS image, for OMAP-L138/AM18xx, type `armais` as the image type. When prompted for a file name, provide the path to `arm-spi-ais.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x8/` directory of the PSP installation.
   - To burn U-Boot, run the SPI flasher program again, and type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file. A pre-built image is located in the `images/u-boot/omapl1x8/` directory of the PSP installation.

- For **03.21.xx.xx** Releases:

1. Start CCStudio v5 and connect to the ARM. Ensure that you have the latest ARM GEL file from [[1]] for LogicPD EVM and [[2]]for Spectrum Digital EVM correctly specified.
2. Execute the GEL function "Full EVM-->SPI1_PINMUX" incase of LogicPD and "Test_setup-->SPI_setup" on Spectrum Digital.This appears under the Scripts menu.
3. Load the SPI flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/boot-strap/ccs/omapl1x8/` directory of the PSP installation (`SPIWriter_OMAP-L138.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer
4. Run the SPI flasher program. You will be prompted for the file path of ARM AIS image and U-Boot. For booting from SPI, an ARM AIS image and U-Boot are required.
   - First it prompts this message "Will you be writing UBL image ? (y or Y) ", Entering 'N' will skip the flashing of UBL. Y will allow to flash UBL.
   - To burn the ARM AIS image, for OMAP-L138/AM18xx . When prompted for a file name, provide the path to ubl_OMAPL138_SPI_MEM.bin file. A pre-built image is located in the images /boot-strap/serial_flash/omapl1x8/ directory of the PSP installation. Entering 'none' will skip this step.
   - To burn U-Boot. When prompted for a file name, provide the path to the u-boot.bin file. A pre-built image is located in the images/u-boot/omapl1x8/ directory of the PSP installation. Entering 'none' will skip this step.

7. Once the SPI flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from SPI flash

## Flashing images to NAND Flash

Follow these steps to flash images to NAND Flash:

1. The embedded emulation that comes with the EVM is XDS100 version 1, that does not support the ARM. So to connect to the ARM, an external emulator is necessary. Also, to use an external emulator, you need the full version of CCS, not the one that comes with the evm. If you do not have an external emulator, you cannot load the NAND Flash using these instructions.
2. Set the boot pins to emulation boot mode. This is done by setting switch S7 on the EVM according to the following table:
   For LogicPD EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | OFF | OFF | ON |

   For Spectrum Digital EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | ON | ON | ON |

- For **03.20.xx.xx** Releases:

1. Start CCStudio and connect to the ARM. See details at: How to connect to the OMAP-L138/C6748 EVM board using CCS?. Ensure that you have the latest ARM GEL file from LogicPD [3]/ Spectrum Digital [2] correctly specified.
2. Execute the GEL function "Full EVM-->EMIFA_NAND_PINMUX" for LogicPD and "Test_setup-->NAND_setup" for Spectrum Digital. In case of CCSv3, this appears under the `GEL` menu. In case of CCSv4, this appears under `Scripts` menu.
3. Load the NAND flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x8` directory of the PSP installation (`nand-writer.out`), or you can build your own by following the steps in the section on Rebuilding the NAND Flash writer
4. Run the NAND flasher program. You will be prompted for the input file type and the file path. For booting from NAND, an ARM AIS image and U-Boot are required.
   - To burn the ARM AIS image, for OMAP-L138/AM18xx, type `armais` as the image type. When prompted for a file name, provide the path to `arm-nand-ais.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x8/` directory of the PSP installation.
   - To burn U-Boot, type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file.

- For **03.21.xx.xx** Releases:

1. Start CCStudio v5 and connect to the ARM. Ensure that you have the latest ARM GEL file from LogicPD [3]/ Spectrum Digital [2] correctly specified.
2. Execute the GEL function "Full EVM-->EMIFA_NAND_PINMUX" for LogicPD and "Test_setup-->NAND_setup" for Spectrum Digital. This appears under `Scripts` menu.
3. Load the NAND flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/boot-strap/ccs/omapl1x8` directory of the PSP installation (`NANDWriter_OMAP-L138.out`), or you can build your own by following the steps in the section on Rebuilding the NAND Flash writer
4. Run the NAND flasher program. You will be prompted for the file path of ARM AIS and U-BOOT. For booting from NAND, an ARM AIS image and U-Boot are required.
   - CCS will prompt this message "Do you want to global erase NAND flash", Entering 'n' or 'N' will not erase the NAND flash.
   - To burn the ARM AIS image,for OMAP-L138/AM18xx. When prompted for a file name, provide the path to `ubl_OMAP-L138_NAND.bin` file. A pre-built image is located in the images/boot-strap/serial_flash/omapl1x8/ directory of the PSP installation. Entering 'none' will skip this step.
   - To burn U-Boot, When prompted for a file name, provide the path to the u-boot.bin file. Entering 'none' will skip this step.

7. Once the NAND flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from NAND flash.

## Flashing images to NOR Flash

Follow these steps to boot from NOR Flash:

1. The embedded emulation that comes with the EVM is XDS100 version 1, that does not support the ARM. So to connect to the ARM, an external emulator is necessary. Also, to use an external emulator, you need the full version of CCS, not the one that comes with the evm. If you do not have an external emulator, you cannot load the NOR Flash using these instructions.
2. Obtain the latest ARM GEL file from LogicPD [3]/ Spectrum Digital [2] . Run the CCStudio Setup tool and ensure that the ARM GEL file is correctly specified.
3. Set the boot pins to emulation boot mode. This is done by setting switch S7 on the EVM according to the following table:
   For Logic PD

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | OFF | OFF | ON |

   For Spectrum Digital

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | ON | ON | ON |

- For **03.20.xx.xx** Releases:

1. Start CCStudio and connect to the ARM. See details at: How to connect to the OMAP-L138/C6748 EVM board using CCS?.
2. Execute the GEL function "Full EVM-->EMIFA_NOR_PINMUX" for Logic PD and "Test_setup-->NOR_setup" for Spectrum Digital. In case of CCSv3, this appears under the `GEL` menu. In case of CCSv4, this appears under `Scripts` menu.
3. Load the NOR flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x8` directory of the PSP installation (`norflash-writer.out`), or you can build your own by following the steps in the section on Rebuilding the NOR Flash writer.
4. Run the NOR flasher program. You will be prompted for the input file type and the file path. For booting from NOR, an ARM AIS image and U-Boot are required.
   - To burn the ARM AIS image, for OMAP-L138/AM18xx, type `armais` as the image type. When prompted for a file name, provide the path to `arm-nor-ais.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x8/` directory of the PSP installation.
   - To burn U-Boot, type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file built for NOR.

- For **03.21.xx.xx** Releases:

1. Start CCStudio v5 and connect to the ARM.
2. Execute the GEL function "Full EVM-->EMIFA_NOR_PINMUX" for Logic PD and "Test_setup-->NOR_setup" for Spectrum Digital. This appears under `Scripts` menu.
3. Load the NOR flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/boot-strap/ccs/omapl1x8` directory of the PSP installation (`NORWriter_OMAP-L138.out`), or you can build your own by following the steps in the section on Rebuilding the NOR Flash writer.
4. Run the NOR flasher program. You will be prompted for the input file paths. For booting from NOR, an ARM AIS image and U-Boot are required.

- To burn the ARM AIS image, for OMAP-L138/AM18xx. When prompted for a file name, provide the path to ubl_OMAP-L138_NOR.bin file. A pre-built image is located in the `images/boot-strap/serial_flash/omapl1x8/` directory of the PSP installation. Entering 'none' will skip this step.
- To burn U-Boot, when prompted for a file name, provide the path to the `u-boot.bin` file built for NOR. Entering 'none' will skip this step.

8. Once the NOR flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from NOR flash

## DA830/OMAP-L137

On the OMAP-L137 and DA830 SoCs, the DSP boots first, on boot-up, the DSP runs the DSP UBL in AIS file format. The purpose of the DSP UBL is to wake up the ARM and start the ARM UBL which is present in raw binary form. The purpose of the ARM UBL is to initialize the PLLs, SDRAM, and other hardware. Once done, it copies the U-Boot into SDRAM and starts it. U-Boot is an open source boot loader and is responsible for booting the Linux kernel.

### Flashing images to SPI Flash

There are two ways to flash images to SPI Flash:

- Serial flasher

1. See this page for instructions on using the command line serial flashing utility, which requires only a UART cable: Serial Boot and Flash Loading Utility for OMAP-L137

- CCS

1. Setup the EVM in "emulation debug" mode by setting SW2 switch as follows:

#### For EVM revisions A and B:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 1 |

#### For EVM revisions after B:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 0 |

2. Start CCStudio and connect to the DSP. Once done, connect to the ARM. Ensure that you have the latest DSP GEL file from Spectrum Digital Support site [4] correctly specified.
3. Load the SPI flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x7/` directory of the PSP installation (`spiflash_writer.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer
4. Run the SPI flasher program. You will be prompted for the input file type and the file path. Three files are required: DSP AIS, ARM UBL and U-Boot.
   - To burn the DSP AIS image, type `dspais` as the image type. When prompted for a file name, provide the path to `dsp-spi-ais.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x7/` directory of the PSP installation.
   - To burn the ARM UBL image, type `armubl` as the image type. When prompted for a file name, provide the path to `ubl-spi.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x7/`

directory of the PSP installation.

- To burn U-Boot, run the SPI flasher program again, and type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file. A pre-built image is located in the `images/u-boot/omapl1x7/` directory of the PSP installation.

5. Once the SPI flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from SPI flash.

### Flashing images to NAND Flash

1. Setup the EVM in "emulation debug" mode by setting SW2 switch on base board as follows:

#### For EVM revisions A and B:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 1 |

#### For EVM revisions after B:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 0 |

2. On the User Interface card, set the SW1 switch as follows:

| Pin # | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Position | 1 | 0 | 1 | 1 |

3. Start CCStudio and connect to the DSP. Once done, connect to the ARM. Ensure that you have the latest DSP GEL file from Spectrum Digital Support site [4] correctly specified.
4. Execute the GEL function `Setup_EMIFA_PinMux()`. In case of CCSv3, this appears under the `GEL` menu. In case of CCSv4, this appears under `Scripts` menu.
5. Load the NAND flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x7/` directory of the PSP installation (`nand_writer.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer
6. Run the NAND flasher program. You will be prompted for the input file type and the file path. Three files are required: DSP AIS, ARM UBL and U-Boot.
   - To burn the DSP AIS image, type `dspais` as the image type. When prompted for a file name, provide the path to `dsp-nand-ais.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x7/` directory of the PSP installation.
   - To burn the ARM UBL image, type `armubl` as the image type. When prompted for a file name, provide the path to `ubl-nand.bin` file. A pre-built image is located in the `images/boot-strap/omapl1x7/` directory of the PSP installation.
   - To burn U-Boot, run the SPI flasher program again, and type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file. A pre-built image is located in the `images/u-boot/omapl1x7/` directory of the PSP installation.
7. Once the NAND flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from NAND flash

# AM17xx

On the AM17xx SoC, the ARM boots first, on boot-up, the ARM runs the ARM UBL in AIS file format. The purpose of the ARM UBL is to initialize the PLLs, SDRAM, and other hardware. Once done, it copies the U-Boot into SDRAM and starts it. U-Boot is an open source boot loader and is responsible for booting the Linux kernel.

## Flashing images to SPI Flash

1. Setup the EVM in "emulation debug" mode by setting SW2 switch as follows:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 0 |

2. Start CCStudio and connect to the ARM. Ensure that you have the latest ARM GEL file from Spectrum Digital Support site [5] correctly specified.
3. Load the SPI flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x7/` directory of the PSP installation (`spiflash_writer.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer
4. Run the SPI flasher program. You will be prompted for the input file type and the file path. Two files are required: ARM AIS and U-Boot.
   - To burn the ARM AIS image, type `armais` as the image type. When prompted for a file name, provide the path to `arm-spi-ais.bin` file. A pre-built image is located in the `images/boot-strap/am17xx/` directory of the PSP installation.
   - To burn U-Boot, run the SPI flasher program again, and type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file. A pre-built image is located in the `images/u-boot/omapl1x7/` directory of the PSP installation.
5. Once the SPI flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from SPI flash

## Flashing images to NAND Flash

1. Setup the EVM in "emulation debug" mode by setting SW2 switch on the base board as follows:

| Pin # | 7 | 2 | 1 | 0 | 3 |
|---|---|---|---|---|---|
| Position | 1 | 1 | 1 | 1 | 0 |

2. On the User Interface card, set the SW1 switch as follows:

| Pin # | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Position | 1 | 0 | 1 | 1 |

3. Start CCStudio and connect to the ARM. Ensure that you have the latest ARM GEL file from Spectrum Digital Support site [5] correctly specified.
4. Execute the GEL function `Setup_EMIFA_PinMux()`. In case of CCSv3, this appears under the `GEL` menu. In case of CCSv4, this appears under `Scripts` menu.
5. Load the NAND flasher tool on to the ARM. You can either use the pre-built binary shipped in the `images/utils/omapl1x7/` directory of the PSP installation (`nand_writer.out`), or you can build your own by following the steps in the section on Rebuilding the SPI Flash writer
6. Run the NAND flasher program. You will be prompted for the input file type and the file path. Two files are required: ARM AIS and U-Boot.

- To burn the ARM AIS image, type `armais` as the image type. When prompted for a file name, provide the path to `arm-nand-ais.bin` file. A pre-built image is located in the `images/boot-strap/am17xx/` directory of the PSP installation.
- To burn U-Boot, run the NAND flasher program again, and type `uboot` as the image type. When prompted for a file name, provide the path to the `u-boot.bin` file.

7. Once the NAND flash has been written with the all the required files, disconnect CCStudio and power off the EVM. Proceed to boot from NAND flash

# Flashing Boot Images on Linux Without CCS

This procedure mirrors that found for the Hawkboard [6]. It requires the Serial boot and Flash Package available here [7] and a compiled u-boot binary. The AIS boot image that is generated makes use of built-in ROM functions and does not use a UBL for booting.

Note: The following tools require the Mono Framework to run on a Linux system. Refer to your Linux distribution's package management instructions for information on how to check if Mono is installed or to install/update it.

## OMAP-L138

1. Prepare a UART AIS boot image using the command-line HexAIS_OMAP-L138.exe and the attached INI file:  . Note that there are different versions of the INI files for the D800K002 ROM, as that ROM revision (which was part of the first silicon revision) does not correctly handle mDDR initialization. Also, each file has mDDR configuration setup for the either 132MHz or 150MHz operation. Comment out the one you do not want to use and un-comment the one you do intend to use. Alternatively, you can use the AISGen GUI tool to perform this step.

```
host$ mono ./HexAIS_OMAP-L138.exe -ini OMAP-L138_EVM_uart.ini -o
u-boot_uart.ais
```

3. Prepare a SPI AIS boot image using the command-line HexAIS_OMAP-L138.exe and the attached INI file. Alternatively, you can use the AISGen GUI tool to perform this step. This file should be placed in the /tftproot directory of your TFTP server so that it can be transferred to the EVM under u-boot using the Ethernet port.

```
host$ mono ./HexAIS_OMAP-L138.exe -ini OMAP-L138_EVM_spi.ini -o
u-boot_spi.ais
```

5. Set the EVM to boot in UART2 boot mode. This is done by setting switch S7 on the EVM according to the following table:

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | OFF | OFF | ON | ON |

6. Start the command-line slh_OMAP-L138.exe tool to boot the board from your host PC.

```
host$ mono ../slh_OMAP-L138.exe -waitForDevice -v
AISUtils/u-boot_uart_L138.ais
```

8. Power up the EVM. The UART boot process will start. It may take as long as 15 seconds to complete.

```
-----------------------------------------------------
   TI Serial Loader Host Program for OMAP-L138
   (C) 2010, Texas Instruments, Inc.
   Ver. 1.65
-----------------------------------------------------


Platform is Windows.
Attempting to connect to device COM1...
Press any key to end this program at any time.

Entering AIS Parser

Waiting for the OMAP-L138...
(AIS Parse): Read magic word 0x41504954.
(AIS Parse): Waiting for BOOTME... (power on or reset target now)
(AIS Parse): BOOTME received!
(AIS Parse): Performing Start-Word Sync...
(AIS Parse): Performing Ping Opcode Sync...
(AIS Parse): Processing command 0: 0x5853590D.
(AIS Parse): Performing Opcode Sync...
(AIS Parse): Executing function...
(AIS Parse): Processing command 1: 0x5853590D.
(AIS Parse): Performing Opcode Sync...
(AIS Parse): Executing function...
(AIS Parse): Processing command 2: 0x58535901.
(AIS Parse): Performing Opcode Sync...
(AIS Parse): Loading section...
(AIS Parse): Loaded 156384-Byte section to address 0xC1080000.
(AIS Parse): Processing command 3: 0x58535906.
(AIS Parse): Performing Opcode Sync...
(AIS Parse): Performing jump and close...
(AIS Parse): AIS complete. Jump to address 0xC1080000.
(AIS Parse): Waiting for DONE...
(AIS Parse): Boot completed successfully.

Operation completed successfully.
```

10. When boot is completed, start a serial terminal program (Hyperterminal, minicom) to connect to the EVM. Press "enter" to see the u-boot prompt.
11. Erase first 256 KB of the serial flash

```
U-Boot > sf probe 0
U-Boot > sf erase 0 40000
```

13. Setup the u-boot enviroment for TFTP download.

```
U-Boot > setenv serverip 172.24.156.199
U-Boot > dhcp
```

15. Use TFTP to transfer the SPI u-boot image generated earlier.

```
U-Boot > tftpboot 0xc0700000 u-boot_spi.ais
```

17. Write the SPI AIS u-boot image to the start of the SPI flash.

```
U-Boot > sf write c0700000 0 40000
```

19. Power off the EVM, and change the boot mode switches to SPI0 master boot. This is done by setting switch S7 on the EVM according to the following table:

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | OFF | OFF | OFF | OFF |

20. Power on the EVM and the u-boot should start.

# Booting U-Boot

U-Boot is an open source boot loader and is responsible for booting the Linux kernel.

Connect a serial cable from the serial port on the EVM to the COM port on the host machine. Set up the serial terminal software as described in Booting the EVM out of the box.
**Note:** Boot images may not have been pre-flashed on the EVM for all boot modes. In this case, follow the procedures in Flashing images to flash the required boot images.

## DA850/OMAP-L138/AM18xx

### Booting from SPI Flash

In order to boot from SPI flash, which has been written with the boot images, set the SW7 switch on the base board as follows:

For LogicPD EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | OFF | OFF | OFF | OFF |

For Spectrum Digital EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | OFF | ON | ON | OFF |

### Booting from NAND Flash

In order to boot from NAND flash, which has been written with the boot images, set the SW7 switch on the base board as follows:
For LogicPD EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | OFF | OFF | OFF |

For Spectrum Digital EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|

| Position | OFF | OFF | OFF | OFF | ON | ON | ON | OFF |
|---|---|---|---|---|---|---|---|---|

### Booting from NOR Flash

In order to boot from NOR flash, which has been written with the boot images, set the SW7 switch on the base board as follows:
For LogicPD EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | ON | ON | OFF |

For Spectrum Digital EVM

| Pin# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Position | OFF | OFF | OFF | OFF | ON | OFF | OFF | OFF |

## DA830/OMAP-L137/AM17xx

### Booting from SPI Flash

- Set the SW2 switch on the DSK board as follows. (X indicates the setting is 'don't care')

| **Pin#** | **7** | **2** | **1** | **0** | **3** |
|---|---|---|---|---|---|
| **Position** | 0 | 1 | 0 | 1 | X |

### Booting from NAND Flash

- Set the SW2 switch on the DSK board as follows. (X indicates the setting is 'don't care')

| **Pin#** | **7** | **2** | **1** | **0** | **3** |
|---|---|---|---|---|---|
| **Position** | 0 | 1 | 1 | 1 | X |

On the User Interface card, set the SW1 switch as follows:

| **Pin #** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| **Position** | 1 | 0 | 1 | 1 |

# Booting the Linux kernel using U-Boot

Booting the kernel requires a valid kernel image (`uImage`) and a target filesystem. A pre-built kernel image is included in the `images/kernel` directory of the PSP installation.

**Booting Linux kernel using U-Boot** describes various methods to boot Linux kernel.

# Using extended memory available on OMAP-L138 EVM board

The OMAP-L138 eXperimenter board has 64MB of memory and the OMAP-L138 EVM has 128 MB available. Out of this 128 MB, some amount is used for Linux and the rest can be used for DSP or as DSP/ARM shared memory. Sometimes it is convenient to allow Linux to use discontinuous blocks of memory. Kernel parameter `mem=` can be

used for this purpose. One example is shown here:

```
U-Boot> setenv bootargs console=ttyS2,115200n8 root=/dev/ram0 rw
initrd=0xc1180000,4M ip=dhcp mem=32M@0xc0000000 mem=64M@0xc4000000
```

If you plan on using Codec Engine (or DSPLink and CMEM alone), you need to make sure that the insmod command for the cmemk.ko kernel module uses the "allowOverlap=1" option (see the CMEM documentation available in the LinuxUtils package). Failure to use this option will result in an error message, since the kernel still reports its memory usage to CMEM as a contiguous 96M range.

# Creating bootable SD card for OMAP-L138 EVM board

The UBL on OMAP-L138 expects the u-boot descriptor to be present in sectors 1 to 25 of the SD card. Sector 0 is used for storing DOS partition information. Hence the u-boot descriptor is stored from sectors 1 till 24. The descriptor is 512 bytes in size and is replicated in each of these sectors. U-Boot binary starts at Sector 117.

The SD card shall be re-partitioned and formated to create some room for storing u-boot. Use fdisk utility (as super user) to delete the existing partition and create a new one.

```
host$ fdisk /dev/mmcblk0 (device name might change if USB card reader
is used)
```

- Delete the existing partitions with 'd' command.
- Create a new partition with 'n' command, followed by 'p' command
- Mark the first cylinder as 20. Typical cylinder size is 32KBytes. So starting the first cylinder at 20 provides us about 600Kbytes for storing UBL and u-boot. If the fdisk utility displays a different cylinder size, make sure that you are leaving atleast 500K space before the first cylinder.
- Leave the last cylinder to default value (or) any other value depending on the partition size requirements.
- Save and exit with 'w' command

Using the **uflash** utility, place the u-boot binary on the SD card. Copy the u-boot.bin to uflash directory,

```
host$ ./uflash -d /dev/mmcblk0 -b u-boot.bin -p OMAPL138 -vv
        u-boot Size 252087
        U-Boot Magic Number     : a1aced66
        U-Boot Entry Point      : c1080000
        U-Boot Number of Blocks : 000001ec
        U-Boot Starting Block   : 00000075
        Load U-Boot Address     : c1080000
        Writing U-Boot Signature
        Writing U-Boot
        Done...
```

# SD Card Writer Utility (uflash)

This is a Linux command line tool specific to TI's Davinci platforms, for flashing UBL/U-Boot to SD card.

## Building uflash

1. Use your Linux host to extract the uflash source code from the `src/utils/mmcsd-writer-#.#.#.#.tar.gz` tarball from the OMAP-L138 Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.
2. Use the native Linux host gcc compiler to build uflash

```
host$ gcc uflash.c -o uflash
```

# Loading Linux Kernel Modules

Many of the kernel features can be built as run-time loadable modules so that they are not part of the kernel image, but can be inserted into the kernel at run-time to increase the running kernel's functionality.

To understand how to configure some features as modules and how to build them, refer to Rebuilding the Linux Kernel and Driver configuration in the Linux kernel.

Pre-built binaries for features configured by default as kernel modules are included in the PSP package in the `images/kernel/omapl1x7/modules/lib` directory for OMAP-L137 and in the `images/kernel/omapl1x8/modules/lib` directory for OMAP-L138. To use these modules, copy the contents of this directory to the /lib directory of your root file system.

On the target Linux command prompt use command `modprobe` command to load the module. `rmmod` command can be used to remove the module. Below is an example of loading the USB file storage gadget module. Similar steps can be followed for any driver module.

## Loading USB 2.0

To load the USB file backed storage gadget:

1. Insert the g_file_storage.ko module with the following command where `/dev/blockdevX` is the storage device acting as the actual storage space. Replace it with the path to the actual block device name acting as physical storage, such as a MMC/SD card.

```
target$ modprobe g_file_storage file=/dev/blockdevX
```

3. Remove the module by using

```
target$ rmmod g_file_storage
```

# DSP wakeup in U-Boot

On the OMAP-L138 SoC, the U-Boot wakes up the DSP by default. The DSP reset vector is set to `0x80000000`, and after wakeup it executes the `idle` instruction. To prevent the DSP from being woken up by U-Boot, you can do the following:

1. Set the `dspwake` environment variable to "no".
2. Save the environment space.
3. Reset the board.

This feature has been added to help DSP users wake up the DSP quickly, since on the OMAP-L138 EVM only the on-board emulation can access the DSP.

# Modifying SPI Frequency in U-Boot

On DA850/OMAP-L138/AM18xx EVMs, in U-Boot, SPI flash has been configured to work at 30MHz. But in some situations one may have to modify the SPI frequency. For example:

- When operating at different voltages.
- When customer has a different SPI flash which operates at different frequency.

In such cases, SPI frequency can be changed in U-Boot by modifying the `CONFIG_SF_DEFAULT_SPEED` variable in `u-boot/include/configs/da850evm.h` file. After this modification, **rebuild u-boot** and **flash the new u-boot** binary file to SPI flash.

# Restoring factory default U-Boot environment variables

To restore factory default U-Boot environment variables, the existing environment variables need to be erased from flash. You will need to know the offset in flash were environment variables are stored and the size of flash dedicated to storing the environment variables.

These values are represented by `CONFIG_ENV_OFFSET` and `CONFIG_ENV_SIZE` respectively in U-Boot EVM configuration file.

For OMAP-L138 (or DA850, AM18xx) SoCs, this file is `include/configs/da850_evm.h` inside the U-Boot source tree.
For OMAP-L137 (or DA830, AM17xx) SoCs, this file is `include/configs/da830_evm.h` inside the U-Boot source tree.

Follow this procedure based on the flash U-Boot stores its environment variables on. The size and offset values given below are to considered representative only.

## SPI flash

On SPI flash, environment variables are stored at offset 256 KBytes into the flash and environment variables sector size is 64 KBytes.

1. On the U-Boot command prompt:

```
U-Boot> sf probe 0
```

3. Erase environment sector length bytes (64 KBytes) from environment sector offset (256 KBytes) as follows:

```
U-Boot> sf erase 40000 10000
```

5. Reset the board and it will come up with default environment variables.
6. Save the new set of environment variables.

```
U-Boot> saveenv
```

### NAND Flash

On the NAND flash, env variables are stored at offset 0 (zero) and environment variables sector size is 128 KBytes.

1. Erase environment sector length bytes (128 KBytes) from environment sector offset 0 (zero) as follows:

```
U-Boot> nand erase 0 20000
```

3. Reset the board and it will come up with default environment variables.
4. Save the new set of environment variables.

```
U-Boot> saveenv
```

### NOR Flash

On the NOR flash, env variables are stored at the 4th sector of NOR flash and environment variables sector size is 128 KBytes.

1. Unprotect the 4th sector using:

```
U-Boot> protect off 60060000 +20000
```

3. Erase environment sector length bytes (128 KBytes) from environment sector offset 60060000 as follows:

```
U-Boot> erase 60060000 +20000
```

5. Reset the board and it will come up with default environment variables.
6. Save the new set of environment variables.

```
U-Boot> saveenv
```

## Restoring MAC address on SPI Flash

On DA850/OMAP-L138/AM18x EVMs, Logic-PD would be pre-flashing the MAC address to the SPI flash. If SPI flash is completely erased then even the MAC address gets erased. MAC address can be written to SPI flash from U-Boot. The following commands need to be entered from U-Boot prompt.

1. Modify the DDR content at address c0000000. MAC address is printed on a sticker pasted on SoC module of EVM.

```
U-Boot> mm.b c0000000
c0000000: fb ? 00
c0000001: ef ? 08
c0000002: fd ? ee
c0000003: df ? 03
c0000004: ff ? 6a
c0000005: ff ? c4
c0000006: bf ? q
```

3. Make sure that modification is correct:

```
U-Boot> md.b c0000000
c0000000: 00 08 ee 03 6a c4 bf ff f3 fd fb ff fe fd bf ef
....j...........
```

```
c0000010: ff ff fb fd 3f fd ff ff bb df 7f fd f7 ff 77 ff
....?.........w.
c0000020: 7f ee f7 fd 3f fd fb fd ed ef bf ce 77 bf fb ff
....?.......w...
c0000030: 6f bc b7 fe ff de fd ff 7b fd ff ff b7 fd ff ff
o.......{.......
```

5. Initialize SPI flash from u-boot.

```
U-Boot> sf probe 0
```

7. Write the MAC address from DDR to SPI flash:

```
U-Boot> sf write c0000000 7f0000 6
```

9. Reset the EVM. U-Boot/Kernel should read the MAC address which was written to the flash.

# Enabling Write-Back Cache on DA830/OMAP-L137

On the DA830 and OMAP-L137 SoCs silicon revisions 1.1 and earlier, the ARM cache in write-back mode is not functional. This has been resolved starting silicon revision 2.0. To maintain backward compatibility with affected silicon revisions, the Linux kernel keeps write-back mode disabled in the default configuration. Users of silicon revisions 2.0 and higher can enable write-back cache from kernel configuration.

```
System Type  --->
    [ ] Force write through D-cache
```

# Message logging on UART in ARM UBL

One may wish to enable messaging logging on the UART console. This feature can be enabled by supplying DEVICE_UART<n>_FOR_DEBUG, where <n> is the UART number. Presently UART0, UART1 and UART2 can be enabled for logging. However, UART<n> may be available for logging depending on the PINMUXing with core peripheral like NAND/SPI/NOR etc and the BOOT mode one wishes to have. Please check this and the settings in device.c/UARTInit()

For example if one wishes to send the console debug messages over UART2 the compiler flag needs to supplied as below (shown for NAND boot mode):

```
Options=-g -o3 -fr"$(Proj_dir)\..\nand" -fs"$(Proj_dir)\..\nand"
-i"$(Proj_dir)\..\include" -d"UBL_NAND" -d"USE_IN_ROM"
-d"DEVICE_UART2_FOR_DEBUG" -me -mv5e --abi=ti_arm9_abi
```

## Linux Functional Test Bench

The Linux Functional Test Bench (LFTB) is the set of tools used to verify the various driver features. The test bench is included in the `test-suite` directory of the PSP installation. Use the `tar` command to extract the LFTB package.

Information on how to use LFTB and its various features is included in the LFTB package itself.

## What's next?

Please continue on to the **Additional information** section of the OMAP-L1 Getting Started Guide.

### References

[1] http://www.logicpd.com/products/development-kits/zoom-omap-l138-experimenter-kitlLogicPD
[2] http://support.spectrumdigital.com/boards/evmam1808/revb/
[3] http://www.logicpd.com/products/development-kits/zoom-omap-l138-experimenter-kit
[4] http://support.spectrumdigital.com/boards/evmomapl137/revd/
[5] http://support.spectrumdigital.com/boards/evmam1707
[6] http://elinux.org/Hawkboard#Booting
[7] http://sourceforge.net/projects/dvflashutils/files/OMAP-L138/v2.11/OMAP-L138_FlashAndBootUtils_2_11.tar.gz/download

# OMAP-L1 Linux Drivers Usage

Before starting to use the drivers please read information on how to configure and rebuild the Linux kernel.

## Ethernet

To test the ethernet interface, you need a valid IP address. To set a static IP address:

```
target$ ifconfig eth0 <static IP address>
```

To obtain an IP address dynamically using DHCP server on the network use:

```
target$ udhcpc
```

To test network connectivity, use ping command to the the network gateway

```
target$ ping <IP address of network gateway>
```

## Audio

ALSA commands `aplay` and `arecord` can be used to playback and record audio respectively. ALSA mixer commands can be used to adjust the playback and capture volume. The command `amixer contents` lists all the params that can be controlled by mixer. To record and playback the content enter:

```
target$ arecord -f dat | aplay -f dat
```

```
To know the current volume settings enter:
target$ amixer cget numid=1
```

To change the playback volume, the following command can be used

```
target$ amixer cset numid=1 80,80 # Sets the volume to 80
target$ amixer cset numid=2 80,80 #This is also used to adjust playback
```

```
volume
```

## Character LCD

Character LCD can be tested by writing some characters/strings to the LCD device:

```
target$ echo "Hello, World" > /dev/lcd
```

The above command will display the string "Hello, World" on the character LCD.

The LCD device can be controlled by writing the following escape sequences to the LCD device (`echo <escape sequence> > /dev/lcd`):

**NOTE**

Pressing `"Ctrl-v", "Ctrl-["` on keyboard will generate the `"^["` escape sequence. In some file systems, the escape sequences may not appear on the serial console but the input will still reach the character LCD.

| Escape Sequence | Functionality |
|---|---|
| `^[[2J` | Clear the display |
| `^[[LI` | Reinitialize display |
| `^[[H` | Cursor to home |
| `^[[LD` | Display ON |
| `^[[Ld` | Display OFF |
| `^[[LC` | Cursor ON |
| `^[[Lc` | Cursor OFF |
| `^[[LB` | Blink ON |
| `^[[Lb` | Blink OFF |
| `^[[Ll` | Shift Cursor Left |
| `^[[Lr` | Shift Cursor Right |
| `^[[LL` | Shift Display Left |
| `^[[LR` | Shift Display Right |

## RTC

Use the `hwclock` [1] utility to test RTC. The RTC device node created is `/dev/rtc0`

## McBSP

The McBSP test setup on the EVM comprises of two EVMs connected via the audio expansion slot. One EVM will act as the master and the other as slave. There is a sample kernel module provided in the examples included in the PSP package which demonstrates the working of McBSP peripheral.

For McBSPs to be interfaced via the audio expansion slot, S7-2 should be turned ON. Note that on doing this audio will not work anymore.

Refer to the README file present with the McBSP example to know more about the module parameters that have to be passed to the sample module to configure the peripheral.

# Power Management

## CPUFreq

The cpufreq driver [2] in kernel allows scaling of clock speed of the CPU on the fly to save power. CPU consumes lower power at lower frequencies.

Please see CPUFreq User's Guide [3] for a comprehensive list of supported features.

**Operating point**

An operating point is a voltage frequency pair that defines a specific power state that the SoC has been characterized for. To see the list of available OPPs (frequencies):

```
target$ cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
300000 200000 96000
target$
```

**Governors**

CPUfreq governor [4] is responsible to making the decision on the OPP to be used among the available OPPs. You can change the governor being used using sysfs entries.

To see the list of governers (the output of this command depends on the governors chosen at kernel configuration step):

```
target$ cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
ondemand userspace performance
target$
```

To see the governor being used currently:

```
target$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
userspace
target$
```

To change governor at runtime:

```
target$ echo performance >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
target$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance
target$
```

**Frequency and Voltage scaling**

When using the 'ondemand' governor, the kernel takes care of frequency transitions based on processor load. When using 'userspace' governor, the frequency transitions can be initiated from command line. For example:

To set the frequency to 96 MHz

```
target$ echo 96000 >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

**NOTE**

When using cpufreq only peripherals on a clock domain asynchronous with PLL0 will function correctly. The drivers in this release are not updated to take care of clock input changes. ASYNC3 domain has been configured to

derive from PLL1, so is immune to CPU frequency changes.

To see the PLL0 frequency changes occurring because of frequency transitions:

```
target$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

To see the CVDD voltage changes occurring because of CPU load:

```
target$ cat /sys/class/regulator/regulator.2/microvolts
1150000
target$
```

## CPUIdle

The cpuidle driver [5] in Linux kernel allows CPU to idle efficiently by moving to a state with lower power consumption during idle. Once built in cpuidle driver works automatically and does not require any user intervention.

OMAP-L138 cpuidle driver implements two states

```
target$ ls /sys/devices/system/cpu/cpu0/cpuidle
state0  state1
target$
```

cpuidle state0 keeps ARM in WFI mode. state1 in addition puts the DDR2/mDDR in power-down mode. Details regarding DDR2/mDDR power down mode are documented in DDR2/mDDR Memory Controller User's Guide section 2.10.

```
target$ cat /sys/devices/system/cpu/cpu0/cpuidle/state0/desc
Wait for interrupt
target$ cat /sys/devices/system/cpu/cpu0/cpuidle/state1/desc
WFI and DDR Self Refresh
target$
```

To find usage of state0 and state1 so far since boot-up:

```
target$ cat /sys/devices/system/cpu/cpu0/cpuidle/state0/usage
6736
target$ cat /sys/devices/system/cpu/cpu0/cpuidle/state1/usage
92724
target$
```

## Suspend-to-RAM

Suspend-to-RAM [6] allows the system to save power by freezing all tasks, asking all drivers to move the devices to a low power state (by disabling the peripheral clock using its LPSC), cutting the clock to external RAM and finally moving the SoC to DeepSleep mode. This user-initiated transtition saves the maximum power among all the power saving mechanisms available.

To put the system into Suspend-to-RAM state use `rtcwake` command.

```
rtcwake -d /dev/rtc0 -s 20 -m mem
```

In the above example, the system remains in suspended state for 20 seconds before coming back up.

**IMPORTANT**

If you are using MMC/SD card in your system, before suspending the system the MMC/SD card has to be

unmounted. If the system is suspended without unmounting the card, the behaviour is undefined, the system may hang and not resume at all. This is a known behaviour for MMC/SD cards in Linux kernel.

## Video Port Interface (VPIF)

The Video port interface (VPIF) driver supports NTSC and PAL standards for both display and capture. Currently the driver supports Composite and S-video capture and display.

The bootargs to be set for NTSC display and capture are

```
u-boot> setenv bootargs console=ttyS2,115200n8 noinitrd rw
ip=<ipaddr> root=/dev/nfs nfsroot=<nfs path> ,nolock
mem=32M@0xc0000000 mem=64M@0xc4000000 vpif_capture.ch0_bufsize=692224
vpif_display.ch2_bufsize=692224
```

The bootargs to be set for PAL display and capture are

```
u-boot> setenv bootargs console=ttyS2,115200n8 noinitrd rw
ip=<ipaddr> root=/dev/nfs nfsroot=<nfs path> ,nolock
mem=32M@0xc0000000 mem=64M@0xc4000000 vpif_capture.ch0_bufsize=831488
vpif_display.ch2_bufsize=831488
```

**NOTE**

The entire bootargs has to be entered in a single line.

Example test applications for standalone display and capture-display loopback are included in PSP package under `src/examples/vpif`. Please refer to the included `README` file for usage instructions.

## Touchscreen

The touchscreen device node is created as `/dev/input/touchscreen0`. Before using the touchscreen, it needs to be calibrated. Use the command `ts_calibrate` to do that. To test the touchscreen functionality `ts_test` can be used.

Note that the SoC does not have a touchscreen controller integrated. The touchscreen functionality is provided by on-board components like TPS65070 or TSC2004.

## Timers

Timers in Linux kernel are used to provide clock tick to the kernel and as watchdog timer.

**DA830/OMAP-L137/AM17xx**

The DA830/OMAP-L137/AM17xx SoCs have two 64-bit timers.

- Timer 0 is configured as two 32-bit timers by kernel at boot-up.
  - Bottom half (12) is used as clockevent and free-run counter.
  - Top half (34) left unused
- Timer 1 will be configured as watchdog timer if support is enabled.

**DA850/OMAP-L138/AM18xx**

The DA850/OMAP-L138/AM18xx SoCs have four 64-bit timers.

- Timer 0 is configured as two 32-bit timers by kernel at boot-up.
  - Bottom half (12) is used as clockevent.
  - Top half (34) is used as free-run counter.

- Timer 1 will be configured as watchdog timer if support is enabled.
- Timer 2 is left untouched.
- Timer 3 is left untouched.

### References

[1] http://linux.die.net/man/8/hwclock
[2] http://lxr.linux.no/linux+v2.6.32/Documentation/cpu-freq/
[3] http://lxr.linux.no/linux+v2.6.32/Documentation/cpu-freq/user-guide.txt
[4] http://lxr.linux.no/linux+v2.6.32/Documentation/cpu-freq/governors.txt
[5] http://lxr.linux.no/linux+v2.6.32/Documentation/cpuidle/
[6] http://lxr.linux.no/#linux+v2.6.32/Documentation/power

# UG: DaVinci PSP Installation on DM36x EVM

## Flashing DM36x EVM using NAND Writer utilities (CCS environment)

For flashing the DaVinci DM36x EVM using NAND Writer utilities, refer to the following instructions. The pre-built binaries for NAND Writer, UBL and U-Boot can be downloaded from **DaVinci GIT Linux Kernel Releases**

- System Requirements – CCS 3.3.38 or above installed on Windows XP with Service Pack 2
- Use the Configuration and gel file to set up CCS from the Spectrum Digital Website [1]
- Make sure the Boot Mode / Configuration Select Switch SW4 is set for the NAND boot mode and 8-bit AEMIF configuration -

| S4:1 (AECFG0) | S4:2 (AECFG1) | S4:3 (AECFG2) | S4:4 (BITSEL0) | S4:5 (BITSEL1) | S4:6 (BITSEL2) |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | OFF | OFF |

- Also make sure the Board Configuration Select Switch SW5 is set for the Vcore=1.35 Volts configuration -

| S5:1 (NAND/ONE NAND) | S5:2 (EXTRA1) | S5:3 (EXTRA2) | S5:4 (EXTRA3) | S5:5 (VCORE ADJ) | S5:6 (NTSC/PAL) |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | ON | OFF |

- Connect the XDS510 to the TI JTAG
- Setup CCS 3.3 with the CCS configuration and GEL files corresponding to the DM36x EVM
- Open CCS 3.3, Alt+C to connect to the device. Ctrl+R to reset from CCS
- If the EVM has been loaded with older flash utilities then run the NAND eraser binary that is now supplied as part of the release. This is essential if the EVM has been flashed with utilities from older LSP releases. This is done so that the entire NAND is erased and we want the Bad Block Tables to be re written
- Load the NANDEraser_DM36x.out from the release and run it (Press F5). The eraser will ask for an input. Input “0x02000000”. Run the eraser again. This time input “0x02004000”. The program need not be run the second time if the 4K NAND from SAMSUNG that is going to be tested as part of the release is being flashed. The need to re run the eraser is dependent on whether the NAND being flashed is internally 1 or 2 devices.
- Load the NAND programmer - NANDWriter_DM36x.out and run it
- The NAND programmer will ask for input for the UBL binary. Choose the UBL_DM36x_NAND.bin file from the release.

- NAND programmer will then ask for the U-Boot binary. Choose the binary u-boot-dm365-evm.bin. Set 0x81080000 for the Application Entry Point and set 0x81080000 for the Application Load Address
- Turn off the board and disconnect the CCS from the device. Alt+C and disconnect XDS510 from the device
- Connect the RS232 cable between your PC and EVM
- You will see console messages, UBL identifying the U-Boot magic number and giving control to U-Boot

# MMC/SD boot mode and SD flashing tool

For booting DM36x via SD card and flashing the EVM for MMC/SD boot mode, refer to **SD card boot and flashing tool for DM36x**.

# SPI boot mode and SPI flashing tool

For booting DM36x via SPI flash and flashing the EVM for SPI boot mode, refer to **SPI flash boot and flashing tool for DM36x**. The link also describes how to setup the kernel for SPI FLASH on the DM36x

# Flashing DM36x EVM (PG 1.2 Silicon or newer) using Serial flash utilities

For flashing the DaVinci DM36x EVM using Serial Flash utilities, refer to the following instructions. The pre-built binaries for Serial Flash, UBL and U-Boot can be downloaded from **DaVinci GIT Linux Kernel Releases**.

- Connect the RS232 cable between your PC and EVM .
- Make sure the Boot Mode / Configuration Select Switch SW4 is set for the UART boot mode and 8-bit AEMIF configuration -

| S4:1 (BITSEL2) | S4:2 (BITSEL1) | S4:3 (BITSEL0) | S4:4 (AECFG2) | S4:5 (AECFG1) | S4:6 (AECFG0) |
|---|---|---|---|---|---|
| OFF | ON | ON | OFF | OFF | OFF |

- Also make sure the Board Configuration Select Switch SW5 is set for the Vcore=1.35 Volts configuration -

| S5:1 (NAND/ONE NAND) | S5:2 (EXTRA1) | S5:3 (EXTRA2) | S5:4 (EXTRA3) | S5:5 (VCORE ADJ) | S5:6 (NTSC/PAL) |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | ON | OFF |

- There will be a serial flasher for DM36x "sfh_DM36x.exe" and a UBL "ubl_DM36x_nand.bin" in the release package
- Use the above utilities and a U-Boot binary meant for DM365 to flash the NAND on the DM36x EVM.
- Issue the following command in CYGWIN or DOS.

```
    sfh_DM36x.exe -nandflash  <UBL binary file> <binary
application file>
```

- Turn off the board
- Once this is done, change to NAND boot mode
- Make sure the Boot Mode / Configuration Select Switch SW4 is set for the NAND boot mode and 8-bit AEMIF configuration -

| S4:1 (BITSEL2) | S4:2 (BITSEL1) | S4:3 (BITSEL0) | S4:4 (AECFG2) | S4:5 (AECFG1) | S4:6 (AECFG0) |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | OFF | OFF |

- Power cycle the EVM. You will see console messages, UBL identifying the U-Boot magic number and giving control to U-Boot
- NOTE: The serial utility has an issue where the utility just hangs and is unable to flash the NAND on the DM36x EVM. This is an issue with the core drivers inherited from the flash utilities repository in the GFORGE server. It affects other SOCs as well. Resetting the EVM and running the serial flasher again results in success.

# Booting the DM36x EVM using pre-built binaries

After flashing the EVM for UBL and U-Boot binaries, using one of the above methods (using CCS utilities or Serial flash utilities), the EVM can now be loaded with Kernel and Filesystem images for the complete out-of-box experience.

Refer to → **Booting DM36x Out of the Box** for information on booting the EVM.

## Booting Linux Kernel from a SDRAM (via TFTP download to SDRAM)

- Copy the Linux Kernel image (uImage-dm365-evm.bin) from the images/dm365-evm/ directory of the DaVinci PSP installation to the /tftpboot (TFTP root) directory of your host workstation. If you do not have a TFTP server configured please see the Setting up a TFTP Server.
- Set the following U-Boot ENV variables to download the Linux Kernel image to the SDRAM location

```
setenv bootfile uImage-dm365-evm.bin
setenv serverip <ipaddress-of-your-host-workstation>
setenv bootcmd 'dhcp; bootm'
saveenv
```

- These commands will download the uImage-dm365-evm.bin to the default SDRAM location - 0x80700000. Note that U-Boot will get its IP Address from DHCP on each reboot.

```
Hit any key to stop autoboot:  0
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 44
*** Unhandled DHCP Option in OFFER/ACK: 46
*** Unhandled DHCP Option in OFFER/ACK: 150
*** Unhandled DHCP Option in OFFER/ACK: 44
*** Unhandled DHCP Option in OFFER/ACK: 46
*** Unhandled DHCP Option in OFFER/ACK: 150
DHCP client bound to address 192.168.1.197
TFTP from server 192.168.1.25; our IP address is 192.168.1.197
Filename 'uImage-dm365-evm.bin'.
Load address: 0x80700000
Loading:
#################################################################

#################################################################
          ############
done
Bytes transferred = 2066356 (1f87b4 hex)
```

```
## Booting kernel from Legacy Image at 80700000 ...
   Image Name:   Arago/2.6.31+2.6.32-rc1-r26+gitr
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2066292 Bytes =  2 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

## Booting Linux Kernel from NAND flash

To boot the Linux Kernel image from NAND flash, it has to be downloaded first and copied to the NAND partition.

- Download the Linux Kernel image (uImage-dm365-evm.bin) from the TFTP server and copy to the NAND partition

```
tftp 0x80700000 uImage-dm365-evm.bin
nand erase 0x400000 0x200000
nand write 0x80700000 0x400000 0x200000
```

- The above commands will download the image and write to the NAND partition

```
DM365 EVM > tftp 0x80700000 uImage-dm365-evm.bin
TFTP from server 192.168.1.25; our IP address is 192.168.1.197
Filename 'uImage-dm365-evm.bin'.
Load address: 0x80700000
Loading:
#################################################################

#################################################################
          ############
done
Bytes transferred = 2066356 (1f87b4 hex)
```

```
DM365 EVM > nand erase 0x400000 0x200000

NAND erase: device 0 offset 0x400000, size 0x200000
Erasing at 0x400000 -- 100% complete.
OK
```

```
DM365 EVM > nand write 0x80700000 0x400000 0x200000

NAND write: device 0 offset 0x400000, size 0x200000
 2097152 bytes written: OK
```

- Setup the bootcmd ENV variable to boot the Linux Kernel image from NAND partition

```
setenv bootcmd 'nboot 0x80700000 0 0x400000; bootm'
```

- On each reboot, the Linux Kernel image will be read from NAND partition and copied to SDRAM location

```
Hit any key to stop autoboot:  0

Loading from NAND 1GiB 3,3V 8-bit, offset 0x400000
   Image Name:   Arago/2.6.31+2.6.32-rc1-r26+gitr
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2066292 Bytes =  2 MB
   Load Address: 80008000
   Entry Point:  80008000
## Booting kernel from Legacy Image at 80700000 ...
   Image Name:   Arago/2.6.31+2.6.32-rc1-r26+gitr
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2066292 Bytes =  2 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

## Setting up the target filesystem

DaVinci PSP 03.xx release uses Arago filesystem. The latest filesystem images can be downloaded from **DaVinci GIT Linux Kernel Releases**.

There are two types of filesystems generated by Arago build system:

- Minimal base filesystem:

```
arago-base-image-arago.ext2.gz
arago-base-image-arago.jffs2
arago-base-image-arago.tar.gz
```

- "Demo" filesystem for DM36x EVM

```
arago-demo-image-dm365-evm.ext2.gz
arago-demo-image-dm365-evm.jffs2
arago-demo-image-dm365-evm.tar.gz
```

### Booting with RAMDISK as a root filesystem

- Download the Arago minimal base filesystem image (RAMDISK: arago-base-image-arago.ext2.gz) from the TFTP server and copy to the NAND partition

```
tftp 0x82000000 arago-base-image-arago.ext2.gz
nand erase 0xC00000 0x300000
nand write 0x82000000 0xC00000 0x300000
```

- The above commands will download the Arago base filesystem image and write to the NAND partition

```
DM365 EVM > tftp 0x82000000 arago-base-image-arago.ext2.gz
operating at 100M full duplex mode
```

```
  TFTP from server 192.168.1.25; our IP address is 192.168.1.199
  Filename 'arago-base-image-arago.ext2.gz'.
  Load address: 0x82000000
  Loading:
#################################################################

#################################################################

#################################################################
           ####################
  done
  Bytes transferred = 3148726 (300bb6 hex)
```

```
  DM365 EVM > nand erase 0xC00000 0x300000

  NAND erase: device 0 offset 0xc00000, size 0x300000
  Erasing at 0xc00000 -- 100% complete.
  OK
```

```
  DM365 EVM > nand write 0x82000000 0xC00000 0x300000

  NAND write: device 0 offset 0xc00000, size 0x300000
   3145728 bytes written: OK
```

- Setup the bootcmd ENV variable to boot the Linux Kernel image from NAND partition and also load Arago base filesystem (RAMDISK) into SDRAM from NAND partition

```
  setenv bootcmd 'nand read 0x82000000 0xC00000 0x300000; nboot
0x80700000 0 0x400000; bootm'
```

- Setup the bootargs ENV variable to boot with RAMDISK as the root filesystem

```
  setenv bootargs mem=116M console=ttyS0,115200n8 root=/dev/ram0 rw
initrd=0x82000000,4M ip=dhcp
```

- On each reboot, the Linux Kernel and Arago base filesystem images (RAMDISK) will be read from the corresponding NAND partition and copied to corresponding SDRAM location

```
  Hit any key to stop autoboot:  0

  NAND read: device 0 offset 0xc00000, size 0x300000
   3145728 bytes read: OK

  Loading from NAND 1GiB 3,3V 8-bit, offset 0x400000
     Image Name:   Arago/2.6.31+2.6.32-rc1-r26+gitr
     Image Type:   ARM Linux Kernel Image (uncompressed)
     Data Size:    2066292 Bytes =  2 MB
     Load Address: 80008000
     Entry Point:  80008000
  ## Booting kernel from Legacy Image at 80700000 ...
     Image Name:   Arago/2.6.31+2.6.32-rc1-r26+gitr
```

```
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2066292 Bytes =  2 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

...
RAMDISK: gzip image found at block 0
VFS: Mounted root (ext2 filesystem) on device 1:0.
Freeing init memory: 128K
INIT: version 2.86 booting
Please wait: booting...
Starting udev
Populating dev cachetar: short write
Remounting root file system...
Mon Oct 19 13:53:00 UTC 2009
INIT: Entering runlevel: 5
Starting telnet daemon.
Starting syslogd/klogd: done
 _____                    _____           _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                   |___|

Arago Project http://arago-project.org dm365-evm ttyS0
Arago 2009.09 dm365-evm ttyS0
dm365-evm login:
```

### Booting with NFS as a root filesystem

- The "demo" filesystem for DM36x EVM (arago-demo-image-dm365-evm.tar.gz) should be exported from the NFS server on your host Linux workstation. Perform the following steps (You only need to perform these steps once)
- Log in with a user account on the host Linux workstation
- Perform the following commands to prepare a location for the Arago target file system. For example:

```
host $ cd /home/<useracct>
host $ mkdir -p workdir/filesys
host $ cd workdir/filesys
```

- Switch user to root on the host Linux workstation

```
host $ su root
```

- Perform the following commands to create a copy of the target file system with permissions set for writing to the shared area as <useracct>. Substitute you user name for <useracct>.

```
  host $ cp -a
<release-location>/images/dm365-evm/arago-demo-image-dm365-evm.tar.gz
 .
  host $ tar xvfz arago-demo-image-dm365-evm.tar.gz
  host $ chmod 777 -R /home/<useracct>/workdir/filesys
```

- Edit the /etc/exports file on the host Linux workstation (not the exports file on the target file system). Add the following line for exporting the filesys directory, substituting your user name for <useracct>. Use the full path from root; ~ may not work for exports on all file systems.

```
  /home/<useracct>/workdir/filesys
*(rw,no_root_squash,no_all_squash,sync)

  NOTE: Make sure you do not add a space between the * and the ( in the
 above command.
```

- Still as root, use the following commands to make the NFS server aware of the change to its configuration and to invoke an NFS restart.

```
  host $ /usr/sbin/exportfs -av
  host $ /sbin/service nfs restart

  NOTE: Use exportfs -rav to re-export all directories. Use
/etc/init.d/nfs status to verify that the NFS status is running.
```

- Setup the following ENV variables for NFS at the U-Boot commandline prompt

```
  setenv nfshost <Linux Host IP address>
  setenv rootpath <directory to mount>
```

- Setup the bootargs ENV variable to boot with NFS target filesystem as the root filesystem

```
  setenv bootargs mem=116M console=ttyS0,115200n8 root=/dev/nfs rw
nfsroot=$(nfshost):$(rootpath) ip=dhcp
```

- Here is an example of the console dump

```
  DM365 EVM > setenv nfshost 192.168.1.25
  DM365 EVM > setenv rootpath /opt/workdir/filesys
  DM365 EVM > setenv bootargs mem=116M console=ttyS0,115200n8
root=/dev/nfs rw nfsroot=$(nfshost):$(rootpath) ip=dhcp
  DM365 EVM > print
  ...
  nfshost=192.168.1.25
  rootpath=/opt/workdir/filesys
  bootargs=mem=116M console=ttyS0,115200n8 root=/dev/nfs rw
nfsroot=192.168.1.25:/opt/workdir/filesys ip=dhcp
  DM365 EVM >
```

```
  ...
  eth0: attached PHY driver [Generic PHY] (mii_bus:phy_addr=1:01,
id=221613)
  Sending DHCP requests .
  PHY: 1:01 - Link is Up - 100/Full., OK
  IP-Config: Got DHCP answer from 0.0.0.0, my address is 192.168.1.197
  IP-Config: Complete:
       device=eth0, addr=192.168.1.197, mask=255.255.255.0,
gw=192.168.1.1,
       host=192.168.1.197, domain=xxx.com,
       bootserver=0.0.0.0, rootserver=192.168.1.25,
rootpath=/opt/workdir/filesys
  Looking up port of RPC 100003/2 on 192.168.1.25
  Looking up port of RPC 100005/1 on 192.168.1.25
  VFS: Mounted root (nfs filesystem) on device 0:14.
  Freeing init memory: 128K
  INIT: version 2.86 booting
  Please wait: booting...
  Starting udev
  Populating dev cache
  Remounting root file system...
  NET: Registered protocol family 10
  INIT: Entering runlevel: 5
  Starting telnet daemon.
  Starting syslogd/klogd: done
  Starting thttpd.
   _____                     _____            _          _
  |  _  |___ ___ ___ ___    |  _  |___ ___  |_|___ ___| |_
  |     |  _| .'| . | . |   |   __|  _| . | | | -_|  _|  _|
  |__|__|_| |__,|_  |___|   |__|  |_| |___|_| |___|___|_|
                |___|                   |___|

  Arago Project http://arago-project.org dm365-evm ttyS0
  Arago 2009.09 dm365-evm ttyS0
  dm365-evm login:
```

# Video Input/Output Source Configuration

The video capture (VPFE) and display (VPBE) drivers in DaVinci PSP release allow input and output sources to be configured from the bootargs ENV variable in U-Boot.

- VPFE input source can be configured via vpfe_capture.interface settings in the bootargs (replace if it is already there, append to current bootargs otherwise). The VPFE capture driver on DM36x also allows dynamic switching of TVP5146 and TVP7002 video input decoder sources.

```
  vpfe_capture.interface=2
  for TVP7002 (default)
```

```
  vpfe_capture.interface=1
  for Micron sensor – MT9T031
```

```
  vpfe_capture.interface=0
  for TVP5146
```

- Default VPFE input standard is Composite and mode is NTSC. The input standard can be switched to Component or S-Video using S_INPUT IOCTL and mode can be switched to PAL or any other resolutions (720P-60, 1080I-30 etc) using the S_STD IOCTL.
- Buffer allocation for the VPFE capture driver can be configured using vpfe_capture.bufsize settings in the bootargs (replace if it is already there, append to current bootargs otherwise).

```
  vpfe_capture.bufsize=4147200
  for 1080I-30 (1920*1080*2=4147200)
```

```
  vpfe_capture.bufsize=1843200
  for 720P-60 (1280*720*2=1843200)
```

```
  vpfe_capture.bufsize=829440
  for PAL (720*576*2=829440) – default
```

```
  vpfe_capture.bufsize=691200
  for NTSC (720*480*2=691200)
```

- Default VPBE ouput is Composite and mode is NTSC. The output/mode can be switched to different output/mode using the davinci_enc_mngr settings in the bootargs (replace if it is already there, append to current bootargs otherwise). Currently there is an issue with the sequencing of driver loading. The Video display drivers seem to initialize before the I2C probing of video encoders/decoders. Due to this, configuring the video display output/mode settings from the bootargs are known to fail. The same configuration can be achieved at run time using the sysfs attributes - /sys/class/davinci_display/ch0/output and /sys/class/davinci_display/ch0/mode.

```
  davinci_enc_mngr.ch0_output=COMPOSITE davinci_enc_mngr.ch0_mode=pal
```

```
  davinci_enc_mngr.ch0_output=COMPONENT
davinci_enc_mngr.ch0_mode=720P-60
```

```
  davinci_enc_mngr.ch0_output=COMPONENT
davinci_enc_mngr.ch0_mode=1080I-30
```

```
  davinci_enc_mngr.ch0_output=LCD davinci_enc_mngr.ch0_mode=480x272
  for LCD mounted on the DM36x EVM
```

# Building PSP Components

## Rebuilding the Linux kernel

The rebuild the Linux kernel, follow the steps below.

### Preparing your Environment

1. If you have not already done so, install the CodeSourcery tools on your Linux host. For additional documentation on installing the CodeSourcery tools, please visit the CodeSourcery website [2].
2. If U-Boot is not built yet, build U-Boot first, as building Linux Kernel requires `mkimage` utility, which gets built along with U-Boot. Refer to Rebuilding U-Boot for steps to build U-Boot. Once built, mkimage will be present under the tools folder of U-Boot source (`/home/<user>/DaVinci-PSP-SDK-xx.xx.xx.xx/src/u-boot/uboot-xx.xx.xx.xx/tools`). Ensure that this path is added to the `$PATH` variable by adding the following

   **host$** export PATH=home/<user>/DaVinci-PSP-SDK-xx.xx.xx.xx/src/u-boot/uboot-xx.xx.xx.xx/tools:$PATH
4. to the ~/.bashrc file. You can then
5. *source*
6. the ~/.bashrc file with the following command

   **host$** source ~/.bashrc
8. Use your Linux host to extract source files for building the target Linux kernel from the `src/kernel/linux-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory (/home/<user>/DaVinci-PSP-SDK-xx.xx.xx.xx/src/kernel for default path installation). Use the `tar` command to extract the sources.

   **host$** tar zxf linux-#.#.#.#.tar.gz

**NOTE**

Patches from the kernel-patches-#.#.#.#.tar.gz file have already been applied on Linux Kernel. This is the list of patches which have been developed on top of the base Linux kernel version. You can find information about the base Linux kernel version from the release notes accompanying the PSP release.

11. Change directory (`cd` command) to the top-level directory of the Linux kernel source files obtained in the previous step.

    **host$** cd linux-xx.xx.xx.xx

### Basic Configuration of the Kernel

Based on what chip and development board you are using, you will need to configure the kernel for different options. Most of these setings are build into the Make file provided by TI, and you can switch between builds using some simple make options.

1. Clean your installation of previous build settings

   **host$** make distclean ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
3. Configure the kernel according to the default configuration provided. The steps below assume that the CodeSourcery tools are already present in your `$PATH` variable. Information on how to add the tools to your `$PATH` can be obtained form the *Getting Started Guide* for the CodeSourcery tools.
   - For DM36x (DM365 and DM368)

     **host$** make davinci_dm365_defconfig ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

5. Modify the kernel options you would like to run. This may take some planning, but the default settings are a good place to start. For more information on these settings, look further down the page at Driver configuration in the Linux kernel or see detaile instructions for | a different processors root file system [3]]. there are two menu configs that are commmon to run:

   **host$** make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig

   (or)

   **host$** make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- xconfig

## Building uImage

Run the following command to build the kernel image:

**host$** make uImage ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

The compiled `uImage` is copied into the `arch/arm/boot` directory under the kernel tree.

Once the kernel has been compiled, **if you are using tftp boot**, use the following commands to copy the uImage to a place where U-Boot can use TFTP to download it to the EVM. These commands assume you are using the default TFTP root area, which is /tftpboot. If you use another TFTP root location, please change /tftpboot to your own TFTP root location:

**host$** cp arch/arm/boot/uImage /tftpboot

**Note**
If the uImage build fails with a similar message as follows:

```
"mkimage" command not found - U-Boot images will not be built
```

it could be that the path to the u-boot uImage script has not been added to the command search path. Include the path "{u-boot source path}/tools" to your PATH environment variable.

## Building Modules

To build all features configured as modules (M), issue the following command:

**host$** make modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

## Installing modules to Target File System

To install the compiled modules into the target root file system, issue the following command:

**host$** make modules_install INSTALL_MOD_PATH=<root fs path> ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

where the `<root fs path>` is the path of your target root file system on the host machine (/home/user/workdir/filesys for example).

See Loading Linux Kernel Modules for more information on using kernel modules after you build them.

## Rebuilding U-Boot

Follow these steps to rebuild U-Boot:

1. If you have not already done so, install the CodeSourcery tools on your Linux host. For documentation on installing the CodeSourcery tools, please visit the CodeSourcery website [2].
2. Use your Linux host to extract source files for building U-Boot from the `src/u-boot/u-boot-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.
   **Note:** Patches from the uboot-patches-#.#.#.#.tar.gz file have already been applied to U-Boot. This is the list of patches which have been developed on top of the base version. You can find information about the base U-Boot version from the release notes accompanying the PSP release.
3. Go to the u-boot directory created when you extracted the files.
4. Run the following commands on your Linux host to build U-Boot. **Note:** The steps below assume that the CodeSourcery tools are already present in your `$PATH` variable. Information on how to add the tools to your `$PATH` can be obtained form the *Getting Started Guide* for the CodeSourcery tools.

```
host$ make distclean CROSS_COMPILE=arm-none-linux-gnueabi-
```

- for DM36x (DM365 and DM368) EVM

```
host$ make davinci_dm365evm_config
CROSS_COMPILE=arm-none-linux-gnueabi-
```

```
host$ make all CROSS_COMPILE=arm-none-linux-gnueabi-
```

8. The compiled `u-boot.bin` file will be created in the same directory.
9. To change the default options, the EVM configuration file needs to be edited.
   - for DM36x (DM365 and DM368) EVM are specified in the include file `include/configs/davinci_dm365evm.h`

   To change U-Boot environment area location or size:
   - `CONFIG_ENV_SIZE` Configures the environment variable size.
   - `CONFIG_ENV_OFFSET` Configures the environment variable offset.

## Building the User Boot Loader (UBL)

Follow the steps mentioned below for releases **before the 03.21.xx.xx** release:

The latest source code for UBL can be downloaded from Arago GIT server. Once the source is extracted (from source tarball), following steps can be followed to build UBL.

1. Start CCStudio v3.3.
2. From the menus, choose **Project->Open**.
3. Browse to the extracted UBL source and open the `UBL.pjt` project. Project file can be found under `<flash-utils-install-dir>\DM36x\CCS\UBL\` folder of the extracted source.
4. To build for different ARM and DDR frequencies, select the "Config" option suitably in CCStudio window.
5. From the menus, choose **Project->Build**.
6. When the build is complete, the binary file (UBL_DM36x_NAND*.bin) suitable for flashing is generated in the Project Root directory `(<flash-utils-install-dir>\DM36x\CCS\UBL\)`

**For 03.21.xx.xx Release:**

Use your Linux host to extract the UBL code from the `src/boot-strap/flash-utils-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources. If the extracted files are not accessible from your Microsoft Windows host, copy all the extracted files to a location that is accessible.

When using **CCStudiov5**:

1. Start CCStudiov5.
2. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**. Browse to `DM36x/CCS` directory inside the extracted UBL source and select the UBL folder.
3. From the list of Discovered projects:
   - Select UBL_NAND for NAND boot mode.
   - Select UBL_SDMMC for MMC/SD boot mode.
4. From the menus, choose **Project->Build Project**
5. When build is complete:
   - The binary file (ubl_DM36x_NAND.bin) suitable for flashing is created in the directory `(<flash-utils-install-dir>\DM36x\CCS\UBL\UBL_NAND)`

When using **Cygwin or Linux Host**:

1. For a particular platform, the extracted package will consist of a 'Common' directory and a '<PlatformName>' directory.
2. Enter into the '<PlatformName>/GNU' directory.

```
cd '<PlatformName>/GNU'
```

3. If you wish to rebuild everything, simply run 'make' at this point.
4. When build is complete:
   - The binary files (ubl_DM36x_xxx.bin) for supported ARM and DDR frequencies, suitable for flashing is created in the directory `(<flash-utils-install-dir>\DM36x\GNU\ubl)`.
   - The serial flasher (sfh_DM36x.exe) which can be used for flashing NAND is created in the directory `(<flash-utils-install-dir>\DM36x\GNU)`. **Note**
     Currently, serial flasher built from the source code will not work. Till this problem is resolved, please use the serial flasher available in pre-built binaries folder.
5. If you wish to clean-up already built components, run 'make clean' from the path you wish to clean.

## Building the NAND Flash writer

NAND flash writer is used to flash the UBL and U-Boot images to NAND flash. The flash writer also supports flashing a given image at a chosen offset.

Use the following steps to build NAND Flash writer using **CCStudio v5**:

1. Use your Linux host to extract the flash utilities code from the `src/boot-strap/flash-utils-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources. If the extracted files are not accessible from your Microsoft Windows host, copy all the extracted files to a location that is accessible.
2. Start CCStudio v5.
3. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**. Browse to `DM36x/CCS` directory inside the extracted source and select the NANDWriter folder.
4. From the list of Discovered projects, select NANDWriter.

5. From the menus, choose **Project->Build Project**. The built image `NANDWriter_DM36x.out` is placed in the `ccsv4/Debug` under the top directory.

To flash images to NAND Flash, see Flashing images to NAND Flash.
To boot U-Boot from NAND Flash, see Booting from NAND Flash.
To boot the Linux kernel from NAND Flash using U-Boot, see Booting the Linux kernel from NAND Flash.

# NAND Partitions

The NAND writer writer the UBL to Block 1 and U-boot to block 25. We reserve 24 blocks for UBL becuase RBL will search for a valid UBL header from blocks 1 to 24

- Block 1 through 24 : UBL
- Block 25 through 30 : U-Boot
- Block 31 to 32 : Environment

# Building and using the MT9P031 CMOS image sensor driver

This section contains the information on compiling and installing the CMOS sensor driver, as well as the use of an application demo to capture video at 1080p30 (1920x1080 30fps) and display up to 1080I60.

The demo has a semi-auto exposure and semi-auto white balance algorithm, which only requires one-time calibration under unchanged lighting condition. The calibration is simple and easy. It is performed by pointing the camera to a white scene and pressing one key. This algorithm is contributed to the open source by Leopard Imaging, Inc.

The demo is tested on DM36x EVM with LI-5M02 rev2 [4] image sensor board from Leopard Imaging,Inc.

1. Building the driver

The CMOS sensor driver is available at arago git tree [5] in this directory : drivers/media/video/davinci/. The major part of the driver is in mt9p031.c.

- Getting the latest linux git tree

```
git clone git://arago-project.org/git/projects/linux-davinci.git
linux-davinci
```

- building the kernel with the driver enabled

Follow the steps in section “Building the Kernel” on this page. Make sure “CONFIG_SOC_CAMERA_MT9P031=y” is included in linux-davinci/arch/arm/configs/davinci_dm365_deconfig. If not, it can be configured using:

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig
```

check “device drivers -> Multimedia Support -> Video for Linux” and “device drivers -> Multimedia Support -> Video Capture Adapters -> mt9p031 support”. It can be built in the kernel or dynamically loaded as a module.

- To load the driver using the dynamically-loadable modules, copy the modules (.ko files) to the target file system. Execute the following command on the DM36x to load the CMOS sensor driver (If the driver is built inside the kernel, this step can be ignored):

```
insmod mt9p031.ko
```

- Execute the following command on the DM36x to unload the CMOS sensor Driver:

```
rmmod mt9p031.ko
```

2 . Building the demo application

The latest sample application code for DaVinci PSP release can be downloaded/cloned from examples-davinci Git tree [6].

- Getting the examples:

```
git clone git://arago-project.org/git/projects/examples-davinci.git
examples-davinci
```

- Go to the root directory of sample application. (the demo application is capture_prev_rsz_onthe_fly_bayer_mew)

```
cd examples-davinci/imp-prev-rsz/dm365/
```

- Export the Linux Kernel directory and the LINUXLIBS_DIR. Linuxlibs is available in Davinci PSP release.

```
export KERNEL_DIR=<kernel directory>
export LINUXLIBS_DIR=<linuxlibs directory>
```

- Clean any object files previously created

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- clean
```

- Build the application

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

- Copy the application to the target file system.

```
cp capture_prev_rsz_onthe_fly_bayer_mew <target filesys addr>/opt/
```

3. Running the demo

- To run the demo, bootargs should be set as follows. The bootargs is for booting with NFS.

```
setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=dhcp
root=/dev/nfs nfsroot=<linux host IP>:<nfs host
directory>,nolock mem=60M
video=davincifb:vid0=OFF:vid1=OFF:osd0=1920x1080x4,32400K
dm365_imp.oper_mode=0 vpfe_capture.interface=1
vpfe_capture.bufsize=4147200 davinci_enc_mngr.ch0_mode=1080I-30
```

the meaning of the parameters:

- dm365_imp.oper_mode

select the image mode, 0:default, 1:continuous,2: XOR mode

- vpfe_capture.interface

select the vpfe input interface, 0:tvp5146 composite input. 1:mt9p031 CMOS sensor bayer input. 2:tvp7002 component input.

-vpfe_capture.bufsize

make sure the buffer size is equal to or larger than : 1920*1080*2 = 4147200 bytes

-davinci_enc_mngr.ch0_mode

Select the display mode, 1080I-30 standards for 1920*1080 interleaved @30fps

- Make sure the loadmodules_hd.sh and binary files are under /opt in the file system. After boot up the file system, log in as root and enter the commands below:

```
cd /opt
```

```
./loadmodules_hd.sh

./capture_prev_rsz_onthe_fly_bayer_mew –i1 –m4 –h50

or ./capture_prev_rsz_onthe_fly_bayer_mew –i1 –m4 –h60

or ./capture_prev_rsz_onthe_fly_bayer_mew –i1 –m10 –h50

or ./capture_prev_rsz_onthe_fly_bayer_mew –i1 –m10 –h60
```

For the arguments that follows “./capture_prev_rsz_onthe_fly_bayer_mew”:

-i1: select input type is CMOS sensor input

-m4/m10: select standard is 720P-30/1080P-30

-h50/h60: select 50Hz/60Hz anti-flicker

- semi-auto exposure and semi-auto white balance

When the loop-back video is on the TV screen, point the camera to a white or grey area, such as a piece of white paper, a white wall, or the professional 18% grey card, type 'w' key, a square frame will show up in the center of the video, make sure the frame covers a uniform white/grey area, type 'w' key again, it will ajust sensor exposure time and gain, IPIPE white balance gains and RGB matrix to get a decent quality video.

# MT9P031 CMOS Sensor Driver Overview

This section provides a brief overview of the driver and its features. The driver is under GPL license, contributed to the open source community by Leopard Imaging, Inc.

1. Overview

The MT9P031 CMOS sensor driver can work as built-in driver or loadable module. It mainly handles the communication and configuration of the CMOS sensor. And it fits into the V4l2 structure as a video capture device.

- building Blocks

The CMOS sensor driver can be divided into the following blocks:

Module init:

This block handles all the initialization activities including driver registration, driver un-registration, probe and remove registration.

Sensor init:

This block handles CMOS sensor reset and enable.

Configuration and Control:

This block handles the CMOS sensor function.

Standard and Format:

This block handles standard and format supported by the CMOS sensor.

2. Features

The CMOS sensor driver supports communication and configuration of sensor through I2C bus, as well as other features such as:

Multiple Standards

Binning or non-binning mode

Set and get control status

Set, set and query standard

Set and try(check) format

Get and set sensor registers

- Standards

The CMOS sensor driver supports the following standards:

-V4L2_STD_720P_30 (1280*720 progressive 30fps), binning mode, tested

-V4L2_STD_1080P_30 (1920*1080 progressive 30fps), non-binning mode, tested

-V4L2_STD_525_60, non-tested

-V4L2_STD_625_50, non-tested

-V4L2_STD_525P_60, non-tested

-V4L2_STD_625P_50, non-tested

- Get and Set sensor control

The CMOS sensor drive can supports set and get the current control, the control type includes:

- VFLIP control

- HFLIP control

- Gain control

- Exposure and Auto-Exposure control

- Get, Set and query standard support
  The CMOS sensor driver supports set and get the standard.
- Set and try(check) format
- The CMOS sensor driver supports checking the width & height setting as well as setting different registers, such as:

- PLL config

- Enable or Disable sensor

- Bin and Skip mode - Width and Height

- Row start and Column start

- Horizontal blank and Vertical blank

- Get and Set the sensor registers

The CMOS sensor driver supports reading and setting the sensor register values.

- Default

By default, the driver initializes the CMOS sensor in V4L2_STD_1080P_30 standard, which is 1920*1080 progressive 30fps, non-binning mode.

## References

[1] http://support.spectrumdigital.com/boards/evmdm365/
[2] http://codesourcery.com
[3] http://processors.wiki.ti.com/index.php/Creating_a_Root_File_System_for_Linux_on_OMAP35x#Configure_the_Linux_Kernel_to_Support_File_Systems
[4] https://www.leopardimaging.com/DM365_5M_HD_Camera_Board.html
[5] http://arago-project.org/git/projects/linux-davinci.git
[6] http://arago-project.org/git/projects/?p=examples-davinci.git;a=summary

# GSG: Building Software Components for OMAP-L1/AM1x

^ Up to main **Getting Started Guide for OMAP-L1** Table of Contents

## Rebuilding the Linux kernel

The rebuild the Linux kernel, follow the steps below.

### Preparing your Environment

1. If you have not already done so, install the CodeSourcery tools on your Linux host. For additional documentation on installing the CodeSourcery tools, please visit the CodeSourcery website [2].
2. If U-Boot is not built yet, build U-Boot first, as building Linux Kernel requires `mkimage` utility, which gets built along with U-Boot. Refer to Rebuilding U-Boot for steps to build U-Boot. Once built, mkimage will be present under the tools folder of U-Boot source (`/home/<user>/OMAP_L138_arm_x_xx_xx_xx/DaVinci-PSP-SDK-xx.xx.xx.xx/src/u-boot/uboot-xx.x`
   Ensure that this path is added to the `$PATH` variable by adding the following

   **host$** export PATH=home/<user>/OMAP_L138_arm_x_xx_xx_xx/DaVinci-PSP-SDK-xx.xx.xx.xx/src/u-boot/uboot-xx.xx.xx.xx/tools:$PA
4. to the ~/.bashrc file. You can then
5. *source*
6. the ~/.bashrc file with the following command

   **host$** source ~/.bashrc
8. Use your Linux host to extract source files for building the target Linux kernel from the `src/kernel/linux-#.#.#.#.tar.gz` tarball from the OMAP-L138 Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory (/home/<user>/OMAP_L138_arm_x_xx_xx_xx/DaVinci-PSP-SDK-xx.xx.xx.xx/src/kernel for default path installation). Use the `tar` command to extract the sources.

   **host$** tar zxf linux-#.#.#.#.tar.gz

**NOTE**

Patches from the kernel-patches-#.#.#.#.tar.gz file have already been applied on Linux Kernel. This is the list of patches which have been developed on top of the base Linux kernel version. You can find information about the base Linux kernel version from the release notes accompanying the PSP release.

11. Change directory (`cd` command) to the top-level directory of the Linux kernel source files obtained in the previous step.

    **host$** cd linux-xx.xx.xx.xx

## Basic Configuration of the Kernel

Based on what chip and development board you are using, you will need to configure the kernel for different options. Most of these setings are build into the Make file provided by TI, and you can switch between builds using some simple make options.

1. Clean your installation of previous build settings

   **host$** make distclean ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

3. Configure the kernel according to the default configuration provided. The steps below assume that the CodeSourcery tools are already present in your `$PATH` variable. Information on how to add the tools to your `$PATH` can be obtained form the *Getting Started Guide* for the CodeSourcery tools.

   - For OMAP-L138 (or DA850, AM18xx)

     **host$** make da850_omapl138_defconfig ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
   - For OMAP-L137 (or DA830, AM17xx)

     **host$** make da830_omapl137_defconfig ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

5. Modify the kernel options you would like to run. This may take some planning, but the default settings are a good place to start. For more information on these settings, look further down the page at Driver configuration in the Linux kernel or see detaile instructions for | a different processors root file system [3]]. there are two menu configs that are commmon to run:

   **host$** make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig

   (or)

   **host$** make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- xconfig

## Building uImage

Run the following command to build the kernel image:

**host$** make uImage ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

The compiled `uImage` is copied into the `arch/arm/boot` directory under the kernel tree.

Once the kernel has been compiled, **if you are using tftp boot**, use the following commands to copy the uImage to a place where U-Boot can use TFTP to download it to the EVM. These commands assume you are using the default TFTP root area, which is /tftpboot. If you use another TFTP root location, please change /tftpboot to your own TFTP root location:

**host$** cp arch/arm/boot/uImage /tftpboot

**Note**

If the uImage build fails with a similar message as follows:

```
"mkimage" command not found - U-Boot images will not be built
```

it could be that the path to the u-boot uImage script has not been added to the command search path. Include the path "{u-boot source path}/tools" to your PATH environment variable.

### Building Modules

To build all features configured as modules (M), issue the following command:

**host$** make modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

### Installing modules to Target File System

To install the compiled modules into the target root file system, issue the following command:

**host$** make modules_install INSTALL_MOD_PATH=<root fs path> ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

where the `<root fs path>` is the path of your target root file system on the host machine (/home/user/workdir/filesys for example).

See Loading Linux Kernel Modules for more information on using kernel modules after you build them.

## Driver configuration in the Linux kernel

This section describes the procedure to configure the kernel to support various drivers.

- To begin, if you have not done basic configuration of the kernel go to that section, and do that first. Once you have compiled with base configuration, return here.

Once you base configuration is set, you can use menu based configurations to change details about your kernel's build.

```
host$  make menuconfig ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

- To select/de-select a feature, press 'space' after bringing cursor over selection box. When '*' appears in selection box, the feature is selected, when the selection box is empty, the feature is de-selected.
- To configure a particular feature as module, press 'space' until an 'M' appears in the selection box.

Note that some of the menu options which are required to be deselected here (selection box empty) may not appear at all because the internal checks which render the option invalid. In this case, it is safe to assume that the options have been automatically disabled.

### USB 2.0

USB2.0 interface is build into the default config in Dual Role mode meaning it can either be a Host or Device (OTG is not supported) automatically without need for user intervention. In the Dual Role mode support for MSC application is available in Host mode while RNDIS/CDC gadget application is available in Device mode by default. User can choose to change the application/mode supported by following the below steps if needed.

**WARNING**
When removing the support for USB2.0 from the kernel please ensure that "CPPI support" under "System Type--> TI DaVinci Implementations" is also disabled.

**Configuring for Host**

```
Device Drivers --->

    USB support --->
        <*> Support for Host-side USB
        --- Miscellaneous USB options
        [*] USB device filesystem
        --- USB Host Controller Drivers
```

```
        <*> Inventra USB Highspeed Dual Role Controller Support
        --- DA830/OMAP-L137 USB support
            Driver Mode (USB Host) --->
                (X) USB Host
```

(please see Inventra HDRC USB Controller for more details on the underlying driver)

When required to support HID devices (mouse, keyboard etc), choose the following

```
Device Drivers --->
      Input device support --->
            <*> Mouse interface
            [*] Keyboards --->
                  <*> AT keyboard

      HID Devices --->
            <*> USB Human Interface Device(full HID) support
```

When required to support MSC devices (pen drive etc), choose the following

```
Device Drivers --->
      SCSI device support --->
            --- SCSI device support
            [*] legacy /proc/scsi/support
            --- SCSI support type (disk, tape, CD-ROM)
            <*> SCSI disk support
      USB support --->
            <*> USB Mass Storage support
```

When required to support UVC, UAC class (Webcam, Speakers, Microphone etc.) choose the following

```
Device Drivers --->
      Multimedia support --->
            Video capture adapters --->
                  V4L USB devices --->
                        <*> USB Video Class (UVC)
                        [*] UVC input events device support
      Sound card support --->
            Advanced Linux Sound Architecture --->
                  USB sound devices --->
                        <*> USB Audio/MIDI driver
```

**Configuring for Gadget**

```
      Device Drivers --->
                USB support --->
                    < > Support for Host-side USB
                    <*> Inventra Highspeed Dual Role Controller
(TI, ...)
                        (*)USB Peripheral (gadget stack)
```

When required to support Ethernet gadget choose the following

```
      Device Drivers --->
            USB support --->
                  USB Gadget Support --->
                        <*> USB Gadget Drivers (Ethernet Gadget
(with CDC Ethernet support))
                              [*]     RNDIS support (EXPERIMENTAL)
```

When required to support File backed storage gadget, choose the following

```
      Device Drivers --->
            USB support --->
                  USB Gadget Support --->
                           <M> USB Gadget Drivers
                              <M> File-backed Storage Gadget
                        [*] File-backed Storage Gadget testing version
(NEW)
```

### USB 1.1

```
Device Drivers --->
            USB support --->
                  <*> Support for Host-side USB
                  --- Miscellaneous USB options
                  [*] USB device filesystem
                  --- USB Host Controller Drivers
                  <*>   OHCI HCD support
```

To support MSC, HID, UVC, UAC applications over USB 1.1 interface refer to USB2.0 procedures for the same.

### Audio

```
Device Drivers --->
    <*> Sound card support --->
        <*> Advanced Linux Sound Architecture --->
            <*> ALSA for SoC audio support --->
                <*> SoC Audio for the TI DAVINCI chip
```

For OMAP-L138 (or DA850, AM18xx) EVM board:

```
                <*> SoC Audio support for DA850/OMAP-L138/AM18xx
EVM
```

For OMAP-L137 (or DA830, AM17xx) EVM board:

```
                <*> SoC Audio support for DA830/OMAP-L137/AM17xx
EVM
```

### Graphical LCD

```
Device Drivers --->
         Graphics support  --->
             <*> Support for frame buffer devices --->
             <*> DA8xx/OMAP-L1xx/AM1xxx Framebuffer support
```

When using OMAP-L137 (or DA830, AM17xx) EVM:

```
System Type --->
         TI DaVinci Implementations  --->
             Select DA830/OMAP-L137/AM17xx UI board peripheral (LCD)
--->
```

### Character LCD

```
Device Drivers --->
         Graphics support  --->
             <*> Support for frame buffer devices --->
                < > DA8xx/OMAP-L1xx/AM1xxx Framebuffer support
         <*> Parallel port support  --->
         [*] Staging drivers  --->
             [ ]   Exclude Staging drivers from being built
             <*>       Parallel port LCD/Keypad Panel support
             (0)       Default parallel port number (0=LPT1)
             (3)       Default panel profile (0-5, 0=custom)
             [ ]       Change LCD initialization message ?
             -*-       DA8XX/OMAP-L1/AM1xxx Dummy Parallel Port
```

```
System Type --->
         TI DaVinci Implementations  --->
              Select peripherals connected to expander on UI board
(Character LCD) --->
```

### NAND

```
Device Drivers --->

    < > MMC/SD/SDIO card support  --->

    <*> Memory Technology Device (MTD)  support --->
        [*]   MTD partitioning support
        [*]   Command line partition table parsing
        <*>   Direct char device access to MTD devices
        <*>   Common interface to block layer for MTD
'translation layers'
        <*>   Caching block device access to MTD devices
        <*>   NAND Device Support --->
            <*> Support NAND on DaVinci SoC
```

**WARNING**
Please disable MMC support when NAND has to be used. They are pin multiplexed and NAND will not work when MMC is enabled.

## NOR

Note: NOR flash is supported only on OMAP-L138 (or DA850, AM18xx) EVM

```
Device Drivers --->

    < > MMC/SD/SDIO card support  --->

    <*> Memory Technology Device (MTD)  support --->
        [*]   MTD partitioning support
        [*]   Command line partition table parsing
        <*>   Direct char device access to MTD devices
        <*>   Common interface to block layer for MTD
'translation layers'
        <*>   Caching block device access to MTD devices
              RAM/ROM/Flash chip drivers --->
                 <*> Detect flash chips by Common Flash Interface
 (CFI) probe
                 <*> Support for Intel/Sharp flash chips
              Mapping drivers for chip access --->
                 <*> TI DaVinci board mappings
```

**WARNING**
Please disable MMC support when NOR has to be used. They are pin multiplexed and NOR will not work when MMC is enabled.

## SPI

```
Device Drivers --->

        [*] SPI support  --->
            <*>     SPI controller driver for DaVinci SoC

        <*> Memory Technology Device (MTD) support  --->
        [*]   MTD partitioning support
        [*]   Command line partition table parsing
        <*>   Direct char device access to MTD devices
        <*>   Common interface to block layer for MTD
'translation layers'
        <*>   Caching block device access to MTD devices
              Self-contained MTD device drivers  --->
                  <*> Support most SPI Flash chips (AT26DF, M25P,
 W25X, ...)
                  [*]   Use FAST_READ OPCode allowing SPI CLK <=
50MHz
```

### MMC/SD

```
Device Drivers --->

    <*>MMC/SD/SDIO card support  --->
        <*>   MMC block device driver
        <*>   TI DAVINCI Multimedia Card Interface support
```

### RTC

```
Device Drivers --->

    <*> Real Time Clock  --->
        <*>   TI OMAP1
```

### SATA

```
      Device Drivers --->
            <*> Serial ATA (prod) and Parallel ATA (experimental)
 drivers --->
                  [*] SATA Port Multiplier support
                  <*> AHCI SATA support
```

### McBSP

```
System Type --->
      TI DaVinci Implementations --->
                [*] DaVinci McBSP (serial API) support
                      [ ]   Support for McBSP instance 0
                      [*]   Support for McBSP instance 1
```

**WARNING**

Note: Since the McBSP peripheral has a kernel API based driver, menuconfig options are present under System Type. If sound driver is selected in menuconfig then McBSP cannot be configured. They are mutually exclusive.

### Power Management

#### CPUFreq

```
CPU Power Management  --->
      [*] CPU Frequency scaling
            <*>   'userspace' cpufreq policy governor
            Default CPUFreq governor (userspace)  --->
                  (X) usespace
Device Drivers  --->
      Multifunction device drivers  --->
            <*> TPS6507x Power Management / Touch Screen chips
      [*] Voltage and Current Regulator Support  --->
            <*>   TI TPS6507X Power regulators
```

**CPUIdle**

```
CPU Power Management  --->
      [*] CPU idle PM support
```

**Suspend-to-RAM**

```
Power management options  --->
       [*] Power Management support
       [*] Suspend to RAM and standby
```

## Ethernet

```
[*] Networking support  --->
      Networking options  --->
            [*] TCP/IP networking
            [*]   IP: kernel level autoconfiguration
            [*]     IP: DHCP support
Device Drivers --->
            [*] Network device support  --->
            [*]   Ethernet (10 or 100Mbit)  --->
                  <*>   Generic Media Independent Interface
device support
                  <*>   TI DaVinci EMAC Support
```

**Using the RMII PHY on OMAP-L138 (or DA850, AM18xx) UI card**

```
System Type --->
      TI DaVinci Implementations --->
            [*] Use RMII Ethernet PHY on DA850/OMAP-L138 EVM
Device Drivers --->
        -*- GPIO Support  --->
               <*>   PCA953x, PCA955x, TCA64xx, and MAX7310 I/O
ports
```

**WARNING**
Note: When using RMII PHY, MII ethernet PHY will not be functional. Do not plug in ethernet cables to both the PHYs.

## VPIF

```
System Type  --->
        TI DaVinci Implementations  --->
                [*] TI DA850/OMAP-L138/AM18xx Reference Platform
                      Select peripherals connected to expander on UI
board (Video Port Interface)  --->

Device Drivers --->
    <*> Multimedia support  --->
            <*>   Video For Linux
                [*]   Video capture adapters  --->
                        <*>  DaVinci Video VPIF Display
```

```
                        <*>  DaVinci Video VPIF Capture
                        -*-  DaVinci VPIF Driver
                [ ]   Autoselect pertinent encoders/decoders and other
helper chips
                      Encoders/decoders and other helper chips  --->

                        <*> Texas Instruments TVP514x video
decoder
                        -*- THS7303 Video Amplifier
                        -*- ADV7343 video encoder
```

**Note:** To run the vpif examples add the following to your bootargs

```
vpif_capture.ch0_bufsize=831488 vpif_display.ch2_bufsize=831488
```

**WARNING**
Please disable the graphical LCD frame buffer driver and the character LCD driver when VPIF display has to be used. They are pin multiplexed and VPIF display will not work when LCD is enabled.

```
Device Drivers --->
         Graphics support  --->
             < > DA8xx/OMAP-L1xx/AM1xxx Framebuffer support
```

## Touchscreen

### TSC2004

TSC2004 touchscreen controller is found on DA830 (or OMAP-L137, AM17xx) EVM when using new UI cards

```
Device Drivers --->
        Input device support  --->
                [*]   Touchscreens  --->
                        <*>   TSC2004 based touchscreens
```

### TPS65070

TPS65070 touchscreen controller is found on DA850 (or OMAP-L138, AM18xx) EVM

```
Device Drivers --->
      Multifunction device drivers  --->
            <*> TPS6507x Power Management / Touch Screen chips
        Input device support  --->
                [*]   Touchscreens  --->
                        <*>   TPS6507x based touchscreens
```

## Rebuilding U-Boot

Follow these steps to rebuild U-Boot:

1. If you have not already done so, install the CodeSourcery tools on your Linux host. For documentation on installing the CodeSourcery tools, please visit the CodeSourcery website [2].
2. Use your Linux host to extract source files for building U-Boot from the `src/u-boot/u-boot-#.#.#.#.tar.gz` tarball from the OMAP-L138 Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.
   **Note:** Patches from the uboot-patches-#.#.#.#.tar.gz file have already been applied to U-Boot. This is the list of patches which have been developed on top of the base version. You can find information about the base U-Boot version from the release notes accompanying the PSP release.
3. Go to the u-boot directory created when you extracted the files.
4. Run the following commands on your Linux host to build U-Boot. **Note:** The steps below assume that the CodeSourcery tools are already present in your `$PATH` variable. Information on how to add the tools to your `$PATH` can be obtained form the *Getting Started Guide* for the CodeSourcery tools.

```
host$ make distclean CROSS_COMPILE=arm-none-linux-gnueabi-
```

- for OMAP-L138 (or DA850, AM18xx) EVM

```
host$ make da850_omapl138_evm_config
CROSS_COMPILE=arm-none-linux-gnueabi-
```

- for OMAP-L137 (or DA830, AM17xx) EVM

```
host$ make da830_omapl137_evm_config
CROSS_COMPILE=arm-none-linux-gnueabi-
```

```
host$ make all CROSS_COMPILE=arm-none-linux-gnueabi-
```

8. The compiled `u-boot.bin` file will be created in the same directory.
9. To change the default options, the EVM configuration file needs to be edited.
   - for OMAP-L138 (or DA850, AM18xx) EVM are specified in the include file `include/configs/da850_evm.h`
   - for OMAP-L137 (or DA830, AM17xx) EVM are specified in the include file `include/configs/da830_evm.h`

   To change U-Boot environment area location or size:
   - `CONFIG_ENV_SIZE` Configures the environment variable size.
   - `CONFIG_ENV_OFFSET` Configures the environment variable offset.

   Choice of Flash supported. **Note:** Only one of these Flash options should be defined at a time. Defining more than one Flash option results in a compilation error when you build U-Boot:
   - `CONFIG_USE_SPIFLASH` If this flag is defined, U-Boot supports the SPI flash on the EVM board. The environment variables are stored on the SPI flash. This option is switched on by default.
   - `CONFIG_SYS_USE_NAND` If this flag is defined, U-Boot supports U-Boot NAND flash access using the OMAP-L1 SoC. Environment variables are also stored on the NAND flash.

## Rebuilding the ARM Side User Boot Loader

Use your Linux host to extract the ARM UBL code from the `src/boot-strap/armubl-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources. If the extracted files are not accessible from your Microsoft Windows host, copy all the extracted files to a location that is accessible.

**NOTE**
By default, the ARM UBL configures the SoC to work at 300MHz, 1.2V operating point. Changing the Operating Point describes the procedure to change the default operating point.

### For OMAP-L138 (or DA850, AM18xx)

When using **CCStudiov3.3**:

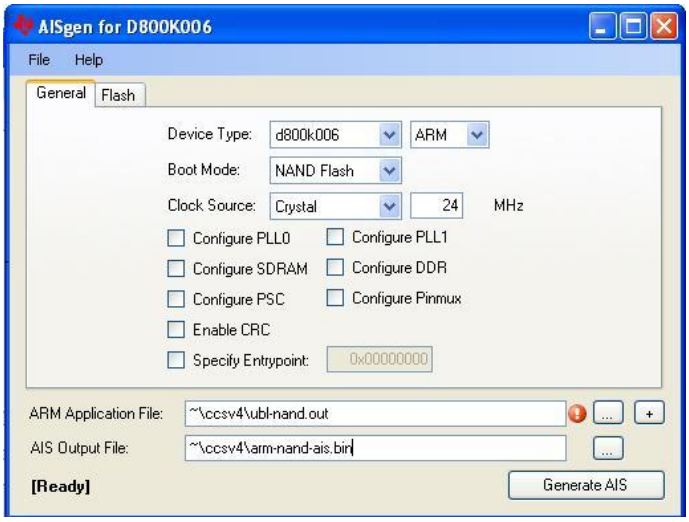
1. Start CCStudio v3.3.
2. From the menus, choose **Project->Open**.
3. Browse to the extracted ARM UBL source and open the `ubl-omapl1x8.pjt` project.
   - To build for SPI Flash, select the "Config" option as `BOOT_SPI` in CCStudio window.
   - To build for NOR Flash, select the "Config" option as `BOOT_NOR` in CCStudio window.
   - To build for NAND Flash, select the "Config" option as `BOOT_NAND` in CCStudio window.
   - To build for MMC/SD boot, select the "Config" option as `BOOT_SDMMC` in CCStudio window. **NOTE** DA850/OMAP-L138/AM18xx does not support MMC/SD boot mode, but UBL can read U-Boot from SD card and boot it.
4. From the menus, choose **Project->Build**. When the build is complete:
   - The executable ELF binary (ubl-xxx.out) file is generated in the Project Root directory `(<armubl-install-dir>\ccsv3.3\)`
5. Proceed to convert the UBL object file obtained above to AIS format file suitable for flashing.

When using **CCStudiov4**:

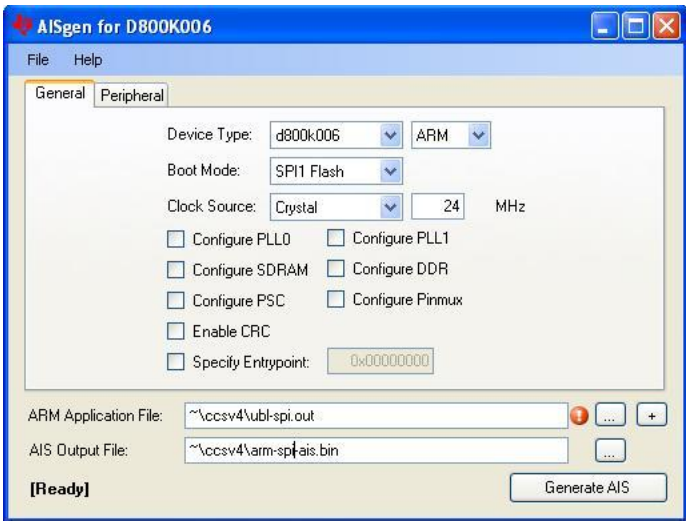
1. Start CCStudiov4.
2. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**. Browse to `ccsv4/omap-l1x8` directory inside the extracted ARM UBL source and select it.
   - To build for SPI Flash, select the "Configuration" option as `BOOT_SPI` in Project->Properties menu.
   - To build for NOR Flash, select the "Configuration" option as `BOOT_NOR` in Project->Properties menu.
   - To build for NAND Flash, select the "Configuration" option as `BOOT_NAND` in Project->Properties menu.
   - To build for MMC/SD boot, select the "Configuration" option as `BOOT_SDMMC` in Project->Properties menu.
3. From the menus, choose **Project->Build Project**
4. When build is complete:
   - The executable ELF binary (ubl-xxx.out) file is generated in the Project Root directory `(<armubl-install-dir>\ccsv4\omapl1x8\)`
5. Proceed to convert the executable ELF binary obtained above to AIS format file suitable for flashing.

#### Generating ARM AIS File For OMAP-L138 (or DA850, AM18xx)

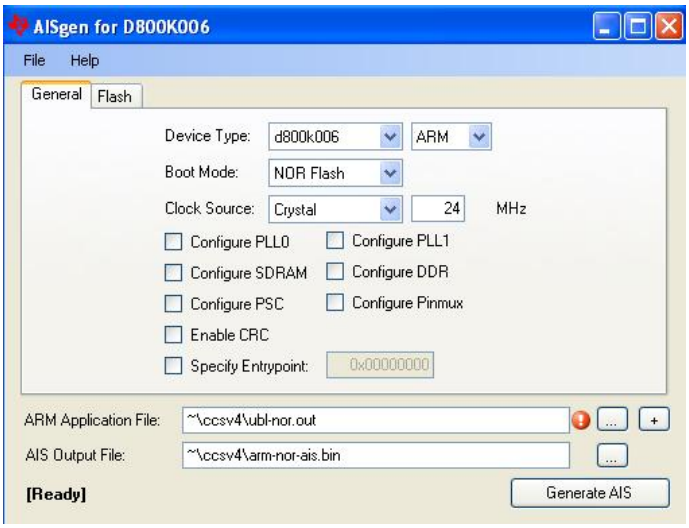
1. Obtain the AIS file format generator tool described in the Bootloader Application Report document.
2. Use the tool as shown below to obtain AIS format files ready to be flashed into NAND/SPI/NOR flash.
   For NAND Flash:




For SPI Flash:




For NOR Flash:




**Note:** Before generating the AIS image for NOR, make sure that Flash Data Width is set to 16 bit in AIS file format generator's "Flash" tab.

### For OMAP-L137 (or DA830, AM17xx)

When using **CCStudiov3.3**:

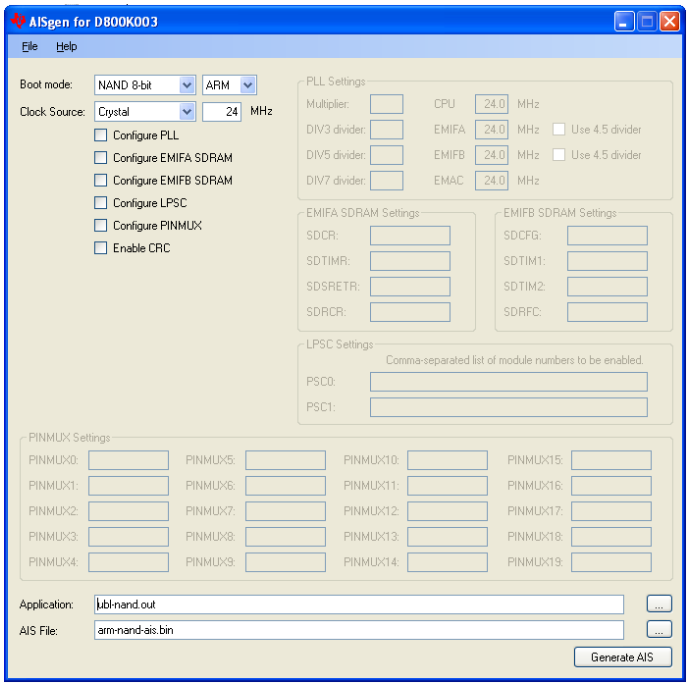
1. Start CCStudio v3.3.
2. From the menus, choose **Project->Open**.
3. Browse to the extracted ARM UBL source and open the `ubl_omapl1x7.pjt` project.
   - To build for SPI Flash, select the "Config" option as `BOOT_SPI` in CCStudio window.
   - To build for NAND Flash, select the "Config" option as `BOOT_NAND` in CCStudio window.
4. From the menus, choose **Project->Build**. When the build is complete:
   - The Intel hex format binary (ubl-xxx.bin) file is generated in the Project Root directory.
   - The executable ELF binary (ubl-xxx.out) file is generated in the Project Root directory.
   - The Project Root directory is <armubl-install-dir>\ccsv3.3\
5. The files generated by the build procedure, have to be flashed.
   - In case of DSP BOOT devices like OMAP-L137 or DA830, the Intel hex format binary file is suitable for flashing.
   - In case of ARM BOOT devices like AM17xx, proceed to convert the executable ELF binary obtained above to AIS format file suitable for flashing.

When using **CCStudiov4**:

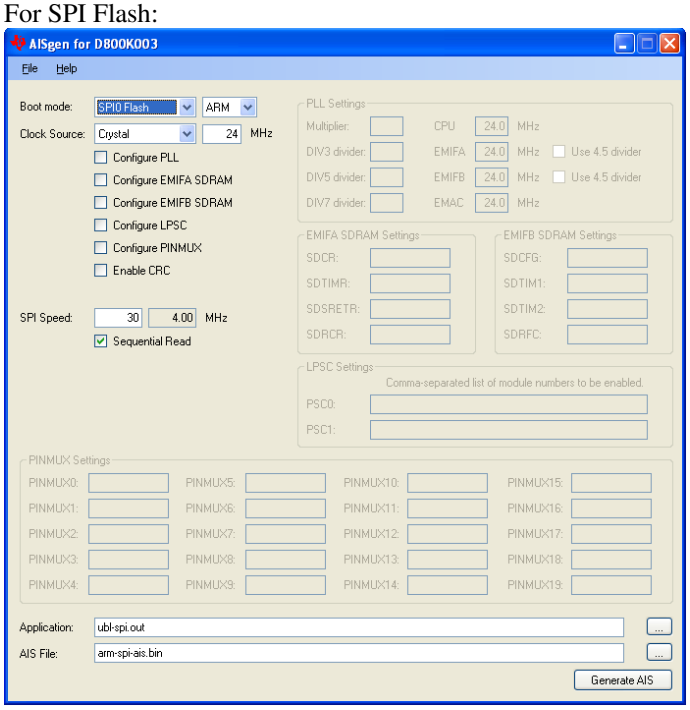
1. Start CCStudiov4.
2. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**. Browse to `ccsv4/omap_l1x7` directory inside the extracted ARM UBL source and select it.
   - To build for SPI Flash, select the "Configuration" option as `BOOT_SPI` in **Project->Properties** menu.
   - To build for NAND Flash, select the "Configuration" option as `BOOT_NAND` in **Project->Properties** menu.
3. From the menus, choose **Project->Build Project**.
4. When build is complete
   - The Intel hex format binary (ubl-xxx.bin) file is generated in the Project Root directory.
   - The executable ELF binary (ubl-xxx.out) file is generated in the Project Root directory.
   - The Project Root directory is `<armubl-install-dir>\ccsv4\omapl1x7\`
5. The files generated by the build procedure, have to be flashed.
   - In case of DSP BOOT devices like OMAP-L137 or DA830, the Intel hex format binary file is suitable for flashing.
   - In case of ARM BOOT devices like AM17xx, proceed to convert the executable ELF binary obtained above to AIS format file suitable for flashing.

#### Generating ARM AIS File For AM17xx

1. Obtain the AIS file format generator tool described in the Bootloader Application Report document.
2. Use the tool as shown below to obtain AIS format files ready to be flashed into NAND/SPI flash.
   For NAND Flash:




For SPI Flash:




## Rebuilding DSP Side User Boot Loader

For the DSP boot devices OMAP-L137 and DA830, DSP UBL is required to wake up the ARM during the boot process.

When using **CCStudiov3.3**:

1. Use your Linux host to extract the DSP UBL code from the `src/boot-strap/dspubl-#.#.#.#.tar.gz` tarball from the OMAP-L138 Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.
2. If the extracted files are not accessable from your Microsoft Windows host, copy all the extracted files to a location that is accessable.

3. Start CCStudio v3.3.
4. From the menus, choose **Project->Open**. Browse to the extracted DSP UBL source and open the `ubl-omapl1x7.pjt` project.
5. By default, the project builds DSP UBL for NAND flash. To build for SPI Flash, select the "Config" option as `BOOT_SPI` in CCStudio window.
6. From the menus, choose **Project->Build**. When the build is complete for SPI flash, the build places `ubl-spi.out` in the `spi` directory under the top level directory. Similarly, the build for NAND flash places the `ubl-nand.out` in the `nand` directory.

**CCStudiov4** specific steps to build the DSP side User Boot Loader:

1. Extract the DSP UBL code from the `src/boot-strap/dspubl-MM.mm.pp.bb.tar.gz` file in PSP installation.
2. Start CCStudiov4. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**. Browse to the extracted DSP UBL source and select the root directory where the ccsv4 project resides. By default, the project builds DSP UBL for NAND flash. To build for SPI Flash, select the "Configuration" option as `BOOT_SPI` in **Project->Properties** menu.
3. From the menus, choose **Project->Build Project**. When build is complete for SPI flash, the build places `ubl-spi.out` in `spi` directory under the top level directory. Similarly, the build for NAND flash places the `ubl-nand.out` in `nand` directory.

Once the .out file is obtained using CCStudiov3.3 or CCStudiov4, convert to AIS File format. Obtain and install the AIS file format generator tool described in Section 5 of the Using the D800K001 Bootloader document.

- Open an MS-DOS command window and change (cd command) to the AIS Generation tool installation directory.
- Execute the following command:

    For SPI:

```
hexgen -romid D800K001 -seqread -crc 1 -spiclk 0 -appln <PATH to
ubl-spi.out> -output <dsp-spi-ais.bin>
```

    For NAND:

```
hexgen -romid D800K001 -seqread -crc 1 -spiclk 0 -appln <PATH to
ubl-nand.out> -output <dsp-nand-ais.bin>
```

The resulting output file is the DSP AIS binary.

## Rebuilding the SPI Flash writer

SPI flash writer is used to flash UBL and U-Boot images to SPI flash. The flash writer also supports flashing a given image at a chosen offset.

- For **03.20.xx.xx** Releases:

1. Use the following steps to build SPI Flash writer using
2. **CCStudio v3.3**
3. :
    1. Use your Linux host to extract the SPI flash writer code from the `src/utils/spiflash-writer-#.#.#.#.tar.gz` tarball from the OMAP-L138 Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.
    2. If the extracted files are not accessable from your Microsoft Windows host, copy all the extracted files to a location that is accessable.

3. Start CCStudio v3.3.
4. From the menus, choose **Project->Open**.
5. Browse to the the extracted source directory, and open the `spiflash_writer.pjt` project.
6. Select the active configuration as SPI0 if the flash writer is being built for DA830/OMAP-L137/AM17xx or as SPI1 if it is being built for DA850/OMAP-L138/AM18xx.
7. From the menus, choose **Project->Build**. The built image `spiflash-writer.out` is placed in the `ccsv3.3/Debug` subdirectory under the project directory.

5. Use the following steps to build SPI Flash writer using
6. **CCStudio v4**
7. :

1. Extract the SPI flash writer code from `src/utils/spiflash-writer-MM.mm.bb.pp.tar.gz` file in PSP installation.
2. Start CCStudio v4. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**.
3. Browse to the the extracted source directory, select the root directory where the ccsv4 project resides.
4. Select the active configuration as SPI0 if the flash writer is being built for DA830/OMAP-L137/AM17xx or as SPI1 if it is being built for DA850/OMAP-L138/AM18xx.
5. From the menus, choose **Project->Build Project**. The built image `spiflash-writer.out` is placed in the `ccsv4/Debug` under the top directory.

- For **03.21.xx.xx** Releases:

1. Extract the SPI flash writer code from `src/boot-strap/flash-utils-03.21.00.03.tar.gz` file in PSP installation.
2. Start CCStudio v5. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**.
3. Browse to the the extracted source directory, select the SPIWriter_ARM directory where the ccsv5 project resides.
4. From the menus, choose **Project->Build Project**. The built image `SPIWriter_OMAP-L138.out` is placed in the `/src/boot-strap/flash-utils-03.21.00.03/OMAP-L138/CCS/SPIWriter/ARM` under the top directory.

To flash images to SPI Flash, see Flashing images to SPI Flash.
To boot U-Boot from SPI Flash, see Booting from SPI Flash.
To boot the Linux kernel from SPI Flash using U-Boot, see Booting the Linux kernel from SPI Flash.

## Rebuilding the NAND Flash writer

NAND flash writer is used to flash the ARM UBL and U-Boot images to NAND flash. The flash writer also supports flashing a given image at a chosen offset.

- For **03.20.xx.xx** Releases:

1. Use the following steps to build NAND Flash writer using
2. **CCStudio v3.3**
3. :

1. Use your Linux host to extract the NAND flash writer code from the `src/utils/nand-writer-#.#.#.#.tar.gz` tarball from the Linux PSP package, which is located in the DaVinci-PSP-SDK-#.#.#.# directory under the main SDK installation directory. Use the `tar` command to extract the sources.

2. If the extracted files are not accessable from your Microsoft Windows host, copy all the extracted files to a location that is accessable.
3. Start CCStudio v3.3.
4. From the menus, choose **Project->Open**.
5. Browse to the extracted source directory and open the `nand_writer.pjt` project file.
6. From the menus, choose **Project->Build**. The built image `nand-writer.out` is placed in the `ccsv3.3/Debug` subdirectory under the project directory.

5. Use the following steps to build NAND Flash writer using
6. **CCStudio v4**
7. :

- Extract the NAND flash writer code from `src/utils/nand-writer-MM.mm.bb.pp.tar.gz` file in PSP installation.
- Start CCStudio v4. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**.
- Browse to the the extracted source directory, select the root directory where the CCSv4 project resides.
- From the menus, choose **Project->Build Project**. The built image `nand-writer.out` is placed in the `ccsv4/Debug` under the top directory.

- For **03.21.xx.xx** Releases:

1. Extract the NAND flash writer code from `src/boot-strap/flash-utils-03.21.00.03.tar.gz` file in PSP installation.
2. Start CCStudio v5. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**.
3. Browse to the the extracted source directory, select the NANDWriter_ARM directory where the ccsv5 project resides.
4. From the menus, choose **Project->Build Project**. The built image `NANDWriter_OMAP-L138.out` is placed in the `/src/boot-strap/flash-utils-03.21.00.03/OMAP-L138/CCS/NANDWriter/Debug/` under the top directory.

To flash images to NAND Flash, see Flashing images to NAND Flash.
To boot U-Boot from NAND Flash, see Booting from NAND Flash.
To boot the Linux kernel from NAND Flash using U-Boot, see Booting the Linux kernel from NAND Flash.

## Rebuilding NOR Flash writer

**Note:** NOR flash writer is supported only on the EVMs for OMAP-L138 (or DA850, AM18xx).

NOR flash writer is used to flash UBL and U-Boot images to NOR flash. The flash writer also supports flashing a given image at a chosen offset.

- For **03.20.xx.xx** Releases:

1. Use the following steps to build NOR Flash writer using
2. **CCStudio v3.3**
3. :

1. Extract the NOR flash writer code from `src/utils/norflash-writer-#.#.#.#.tar.gz` directory in PSP installation.
2. Start CCStudio v3.3. From the menus, choose **Project->Open**
3. Browse to the extracted source directory and open the `ccsv3.3/norflash_writer.pjt` project file.

4. From the menus, choose **Project->Build**. The built image `norflash_writer.out` is placed in the `ccsv3.3/Debug` directory under the top level source directory.

5. Use the following steps to build NOR Flash writer using
6. **CCStudio v4**
7. :

1. Extract the NOR flash writer code from `src/utils/norflash_writer_#.#.#.#.tar.gz` file in PSP installation.
2. Start CCStudio v4. From the menus, choose **File->Import**. Choose **CCS->Existing CCS/CCE Eclipse Project**.
3. Browse to the the extracted source directory, select the root directory where the CCSv4 project resides.
4. From the menus, choose **Project->Build Project**. The built image `norflash_writer.out` is placed in the `ccsv4/Debug` under the top directory.

- For **03.21.xx.xx** Releases:

1. Extract the NOR flash writer code from `src/boot_strap/flash_utils_03.21.00.03.tar.gz` file in PSP installation.
2. Start CCStudio v5. From the menus, choose File->Import. Choose **CCS->Existing CCS/CCE Eclipse Project**.
3. Browse to the the extracted source directory, select the NORWriter_ARM directory where the ccsv5 project resides.
4. From the menus, choose **Project->Build Project**. The built image `NORWriter_OMAP_L138.out` is placed in the `/src/boot_strap/flash_utils_03.21.00.03/OMAP_L138/CCS/NORWriter/Debug` under the top directory.

## What's next?

Please continue on to the **Building the SDK** section of the OMAP-L1 Getting Started Guide.

# Getting Started Guide for OMAP-L1

This Getting Started Guide (GSG) walks you through setting up the OMAP-L137 and OMAP-L138 EVM and installing the software. You should proceed through this guide in the order given for best results. By the end of this Getting Started Guide you will have the EVM booting to Linux and the Linux host development environment configured. You should also bookmark the OMAPL1 category since new articles will continue to appear on this wiki.

**ATTENTION OMAP-L138 DEVELOPERS**: The Generally Available (GA) Linux DVSDK 4.x for the OMAP-L138 device is now available here [1]. There are various capabilities and improvements that have been added to this version. This is the **recommended** release kit for development which supersedes the SDK 1.00.xx Beta. The majority of the information below is only applicable to the SDK 1.00.xx Beta.

## EVM overview

**EVM Overview - OMAP-L138** provides an overview of the OMAP-L138 EVM kits and boards.

**EVM Overview - OMAP-L137** provides an overview of the OMAP-L137 EVM kits and boards.

## Hardware setup

For information on setting up the OMAP-L138 EVM hardware, please follow the steps in the *ZOOM OMAP-L138 eXperimenter Kit QuickStart Guide*, which is available for downloading at http://support.logicpd.com/downloads/1213/.

Spectrum Digital, Inc has a Technical Reference Guide [2] for OMAP-L137 EVM.

## Booting the EVM out of the box

This section describes how to boot the EVM board when you first set it up.

- For **OMAP-L138 (or DA850)** details are provided **here**
- For **OMAP-L137 (or DA830)** details are provided **here**

## Installing the SDK software

**OMAP-L138 DEVELOPERS** - See attention note at the top of the page.

**Installing the Software for OMAP-L1** covers how to install the SDK software and CodeSourcery Sourcery G++ tools for the OMAP-L1 processors.

As host platforms for the development software, you need both a Windows PC and a Linux machine to build and run all of the components.

- The Windows machine is used to run CCStudio 3.3/CCStudio 4, which you will use if you want to build the User Boot Loader (UBL) and Flash writers. CCStudio is also used to burn the boot images (UBL, U-Boot) into the flash using the flash writers provided.
- The Linux host is used for the following:
  - Recompiling U-Boot and the Linux kernel.
  - Hosting the TFTP server required for downloading kernel and file system images from U-Boot using Ethernet.
  - Hosting the NFS server to boot the EVM with NFS as root filesystem
  - Running a serial console terminal application

The Installing the Software for OMAP-L137 page covers how to install the MontaVista tools, SDK software, and Linux Support Pack for the OMAP-L137 processor.

**Note:** If you are migrating from the early adopter version of the SDK software you can check the differences in this document.

## Setting up target file system

**OMAP-L138 DEVELOPERS** - See attention note at the top of the page.

**Setting up OMAP-L1 Target File System** explains how to set up an NFS target file system for use on the OMAP-L1 EVMs.

For information on setting up an NFS target file system for use on the OMAP-L137 EVM please see the Setting up OMAP-L137 Target File System page.

For information on creating other types of target file systems such as SD/MMC and USB please see the Creating file systems on removable media page.

## Building software components

**Building PSP components** provides procedures for rebuilding the platform software components used on the processors or to flash software on to the EVM.

## Building the SDK

**OMAP-L138 DEVELOPERS** - See attention note at the top of the page.

**Building the OMAP-L1 SDK** provides information on building the SDK software for both OMAP-L137 and OMAP-L138.

**Note:** In general, the instructions for building the SDK for OMAP-L138 are the same as for OMAP-L137. Follow the steps on this page that are specific to your device.

The SDK software provided with this EVM contains examples for communicating between the ARM and DSP processors as well as driver modules used in that communication. It also contains development packages, such as Codec Engine, which allow for easy communication between the ARM and DSP.

## Running PSP Components

**Running PSP Components** describes the steps to boot Linux and to how to re-flash UBL and U-boot images if needed.

## Additional procedures

**OMAP-L138 DEVELOPERS** - See attention note at the top of the page.

**Linux Functional Test Bench** introduces a set of tools used to verify the various driver features.
**Loading Linux kernel modules** describes the procedure to load kernel modules into a running kernel.
**Message logging on UART in UBL** describes the procedure to enable message logging in ARM UBL.
**Modifying SPI Frequency in U-Boot** describes how to modify the SPI frequency in U-Boot.
**Restoring factory default U-Boot environment variables** describes how to revert to factory default U-Boot environment variables.
**Using extended memory available on DA850/OMAP-L138/AM18x EVM** describes how Linux can be configured to utilize the additional RAM available on the EVM board when compared to the eXperimenter board.
**Creating bootable SD card for DA850/OMAP-L138/AM18x EVM** describes the steps to write U-Boot to SD card using uflash utility.
**Creating custom boot images for OMAP-L138** describes how to create your own bootable images from scratch

and flash them to run on the ARM or DSP.

For OMAP-L138 and DA850 SoCs, **DSP wakeup in U-Boot** explains how to prevent the DSP from being woken up by U-Boot.

For DA850/OMAP-L138/AM18x EVMs, **Restoring MAC address on SPI Flash** explains the steps to Restore MAC address on SPI flash.

For OMAP-L137 and DA830 SoCs, **enabling Write-Back cache** explains how to enable write-back cache support in Linux kernel when using silicon revision 2.0 and higher.

The following pages provide additional procedures that apply to a variety of platforms, including the OMAP-L1:

- **Setting up a TFTP Server** explains how to set up a TFTP server.
- **TeraTerm Scripts** provides example TeraTerm scripts that can be used with your board.

**Enabling UART1 on AM18X/DA850/OMAP-L138 running Linux** describes steps to enable and verify the UART1/ttyS1 on AM18x/DA850/OAMP-L138 EVMs running linux.

## Additional information

- Additional hardware information can be found at the Logic PD OMAPL138 webpage: http://www.logicpd.com/products/development-kits/zoom-omap-l138-evm-development-kit
- References about some SDK components (Codec Engine, Framework Components): http://focus.ti.com/docs/toolsw/folders/print/tmdmfp.html
- The standalone version of the PSP device drivers is located here: http://focus.ti.com/docs/toolsw/folders/print/supportpkg.html
- Also, look around in this DaVinci wiki site for additional information

## References

[1] http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/dvsdk/DVSDK_4_00/latest/index_FDS.html
[2] http://support.spectrumdigital.com/boards/evmomapl137/revd/files/EVMOMAPL137_TechRef_RevD.pdf

# Writing V4L2 Media Controller Applications on Dm36x Video Capture

## Writing Media Controller Applications for Video Capture on DM36X

### Media Controller Basics

Media Controller is a framework for developing V4L2 based drivers which, compared the earlier implementation of plain V4L2 has a broader view device architecture. While the plain v4L2 had a view of the device as a plain DMA based image drabber which connects the input data to the host memory, the Media Controller takes into consideration the fact that a typical video device might consist of multiple sub-devices in form of either sensors, video decoders, video encoders working in tandem, and also the fact that image grabbing and display is not a simple DMA but might consist of smaller sub-blocks which might do some processing like resizing, format conversion etc. The document below explains these changes in design philosophy and elaborates on the nuts and bolts that make the media Controller as it is today. It also dwells upon the DM365 specifics implementation details for the VPFE capture driver.

#### Media Device

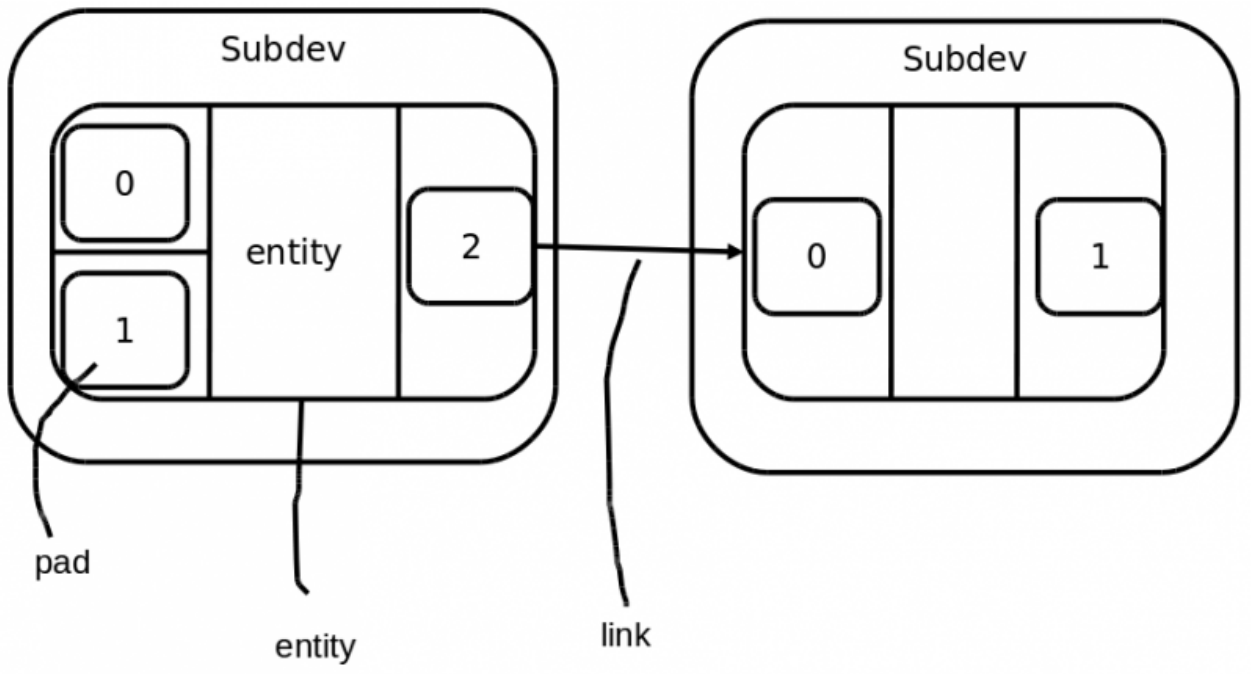
A Media device is the umbrella device under which multiple sub-entities called Media entities can be accessed, modified and worked upon. The Media Device is exposed to the user in form of a device file, which can be opened to enumerate, set and get the parameters of each of the media entities. For example, in DM365 implementation the entire VPFE capture device with its IPIPE, IPIPEIF, CCDC etc is exposed as a Media Device - /dev/media0. If there were a display driver, it would be exposed as a Media device too.

#### Media Entity

A Media entity is a sub-block of a a particular media device which usually does a particular function, can be though about as an connect-able but self-contained block which might have a register set f its own for setting the parameters and can be programed in an independent way. This could be a sub-IP, or a helping device on the same board which offloads a particular function like RAW Capture, YUV Capture, filter etc. On DM365 all the sensors, Video Decoders are media entities and the core itself has been modeled with CCDC, Previewer, Resizer, H3A, AEW as entities. These could be enumerated in the standard V4L2 way using the Media device as the enumerating device. Each of these entities has one or more input and output pads, and is connect-able to another entity through a 'link', between the pads.

#### Sub-devices

Conceptually similar to a media Entity, a subdevice is viewed as sub-block of a V4L2 Video device which is independently configurable through its own set of file operations. The file operations are exposed through V4L2 -like IOCTLS particular to subdevs.Each sub-device is exposed to the user level through device files starting with “subdev-*”. User applications need to configure V4L2 related settings like format, crop,, size parameters through these device handles for each of the sub-devices to make the work in tandem. Structurally, there is almost an one-to-one correspondence between a Media Entity and a sub-device.




## Pads

“Pads” are input and output connect-able points of a Media Entity. Depending on the number of connections the entity can have the pads are pre-fixed in the driver. Typically, a device like a sensor or a video decoder would have only an output pad since it only feeds video into the system, and a /dev/video pad would be modeled as an input pad since it is the end of the stream. The other entities like Resizer, previewer would have typically an input and an output pad and sometimes more depending on the capability.

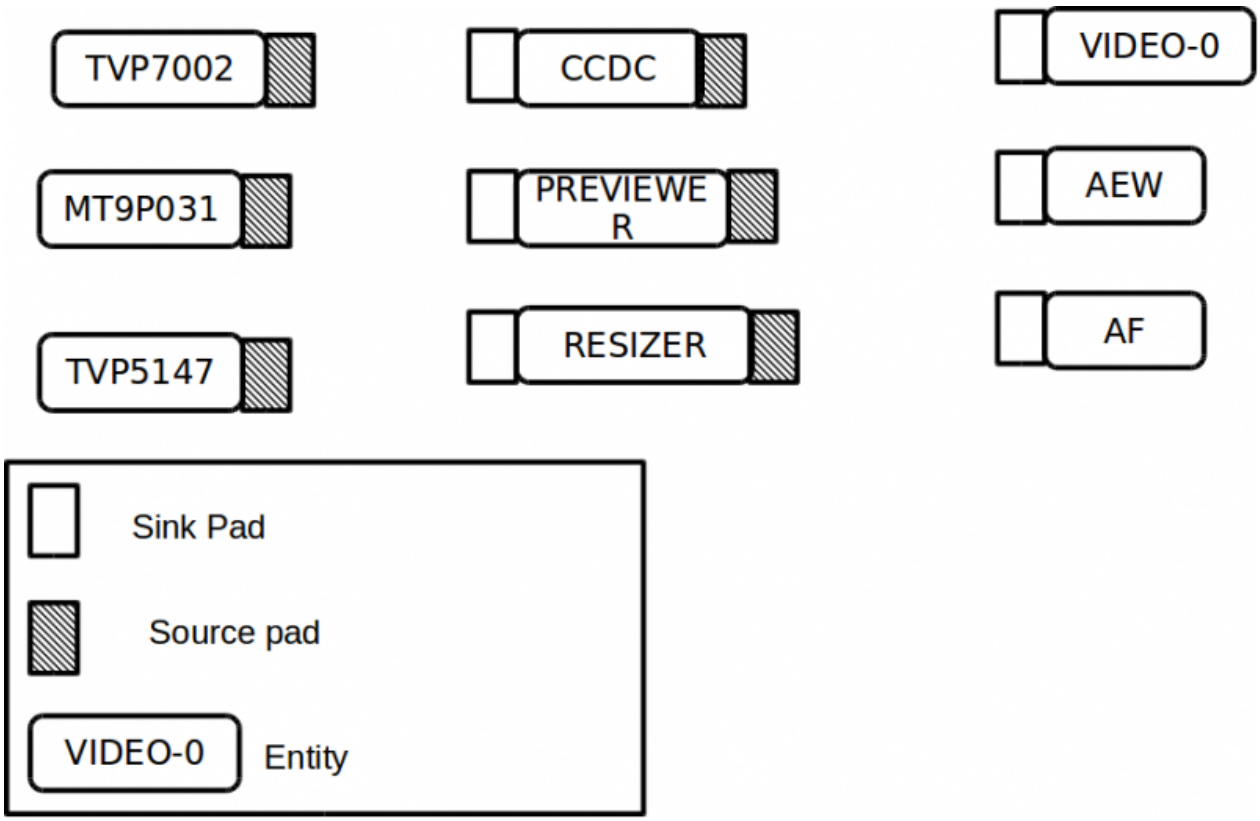
## Link

A link is a 'connection' between pads of different entities. These links can be set, get and enumerated through the Media Device. The application, for proper working of a driver is responsible for setting up of the links properly so that the driver understands the source and destination of the video data.

## Entity Graph

An entity graph is the complete setup of different entities, pads, and the links. For proper working of the software, an entity graph should be properly setup, and the driver, before it can start streaming will validate for the proper graph and the appropriate settings on the sub-device to get to know the intent of the application. Choice of if the video input is RAW BAYER or YUV video is determined by the appropriate setup of the entity graph.

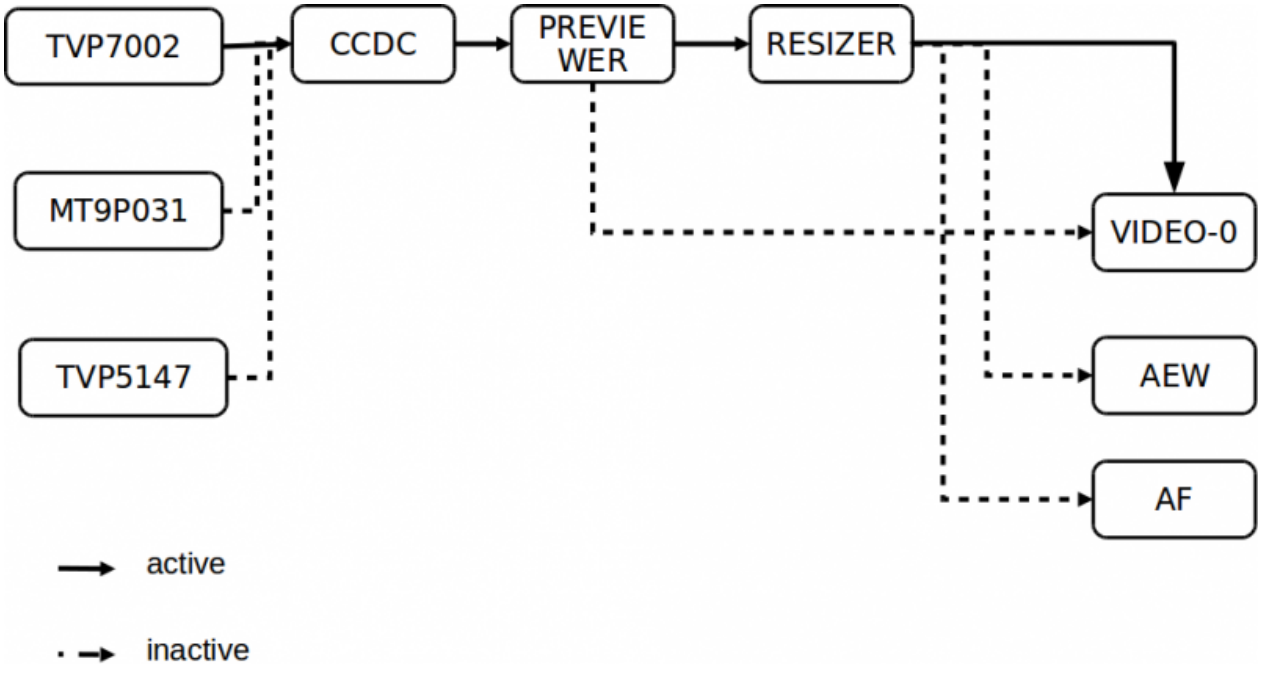
The DM365 supports the following fixed entity and pad configuration.




## Modes of Operation

### Continuous Mode Operation

Continuous Mode refers to the configuration where the input data stream is regulated by the standard capture format like NTSC/720p etc and where the data is input from an external source t be stored int DDR. Here the input data is “streamed” to the DDR and thereby the exchange of buffers between drivers and applications happen on a continuous basis regulated by the interrupts generated at every field or frame. So, the sink is invariably a video node whereas the source is an external sensor or decoder or an inbuilt ADC. The streaming mode supported here is the standard V4L2 mode of streaming.

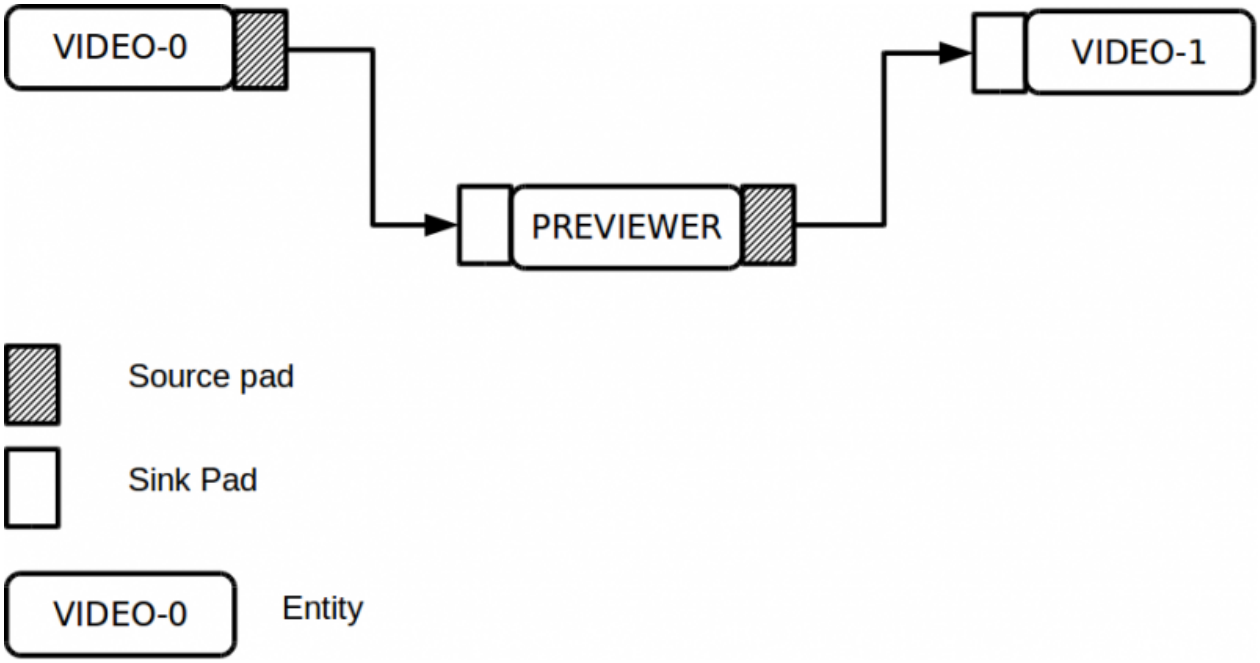


### Single Shot Mode Operation

As opposed to “continuous mode”, “single-shot mode” is referred to setup where the data input is from DDR as opposed to an external source, and the output is DDR again. So here both the source and sinks are video nodes. To facilitate this kind of interface, the input DDR source is mapped to a separate Video device, which is only used in case of single shot modes. Here the input video node supports a source media device pad, and the output video node supports a sink media device pad.

The driver here supports buffer exchanges on input and output using the standard V4L2 calls, with an exception that the present driver supports a single buffer at a time. In DM365, the list of Single shot drivers is as follows:

- Resizer
- Previewer
- Previewer + Resizer configuration




## Steps for writing a Media Controller application

Initial set of pre-defined strings and PAD identification numbers for each entity.

```
/* Media entity names */
#define E_VIDEO_CCDC_OUT_NAME      "DAVINCI VIDEO CCDC output"
#define E_VIDEO_PRV_OUT_NAME      "DAVINCI VIDEO PRV output"
#define E_VIDEO_PRV_IN_NAME      "DAVINCI VIDEO PRV input"
#define E_VIDEO_RSZ_OUT_NAME      "DAVINCI VIDEO RSZ output"
#define E_VIDEO_RSZ_IN_NAME      "DAVINCI VIDEO RSZ input"
#define E_TVP514X_NAME            "tvp514x"
#define E_TVP7002_NAME            "tvp7002"
#define E_MT9P031_NAME            "mt9p031"
#define E_CCDC_NAME            "DAVINCI CCDC"
#define E_PRV_NAME            "DAVINCI PREVIEWER"
#define E_RSZ_NAME            "DAVINCI RESIZER"
#define E_AEW_NAME            "DAVINCI AEW"
#define E_AF_NAME            "DAVINCI AF"
```

```
/* pad id's as enumerated by media device*/
#define P_RSZ_SINK      0 /* sink pad of rsz */
#define P_RSZ_SOURCE      1 /* source pad of rsz */
#define P_PRV_SINK      0
#define P_PRV_SOURCE      1
#define P_RSZ_VID_OUT      0 /* only one pad for video node */
#define P_RSZ_VID_IN      0 /* only one pad for video node */
#define P_PRV_VID_IN      0
#define P_PRV_VID_OUT      0
#define P_TVP514X      0 /* only one pad for decoder */
#define P_TVP7002      0 /* only one pad for decoder */
#define P_MT9P031      0 /* only one pad for sensor */
#define P_CCDC_SINK      0 /* sink pad of ccdc */
#define P_CCDC_SOURCE      1 /* source pad which connects video node */
#define P_VIDEO            0 /* only one input pad for video node */
#define P_AEW            0
#define P_AF            0
```

---

Open the Media Device

```
      /* 3.open media device */
      media_fd = open("/dev/media0", O_RDWR);
      if (media_fd < 0) {
            printf("%s: Can't open media device %s\n", __func__,
"/dev/media0");
            goto cleanup;
      }
```

Enumerate media-entities. Here it is a good idea to store the indices of the entities so they can be addressed using index next time.

```
      /* 4.enumerate media-entities */
      printf("4.enumerating media entities\n");
      index = 0;
      do {
            memset(&entity[index], 0, sizeof(entity));
            entity[index].id = index | MEDIA_ENT_ID_FLAG_NEXT;

            ret = ioctl(media_fd, MEDIA_IOC_ENUM_ENTITIES,
&entity[index]);
            if (ret < 0) {
                  if (errno == EINVAL)
                        break;
            }else {
                  if (!strcmp(entity[index].name,
E_VIDEO_CCDC_OUT_NAME)) {
                        E_VIDEO =  entity[index].id;
                  }
                  else if (!strcmp(entity[index].name, E_MT9P031_NAME))
```

```
 {
                        E_MT9P031 =  entity[index].id;
                  }
                  else if (!strcmp(entity[index].name, E_CCDC_NAME)) {
                        E_CCDC =  entity[index].id;
                  }
                  printf("[%x]:%s\n", entity[index].id,
entity[index].name);
            }

            index++;
      }while (ret == 0);
      entities_count = index;
      printf("total number of entities: %x\n", entities_count);
```

Enumerate all the links and pads. This is just as an information.This step is optional.

```
      /* 5.enumerate links for each entity */
      printf("5.enumerating links/pads for entities\n");

        links.pads = malloc(sizeof( struct media_pad_desc) *
entity[index].pads);
      links.links = malloc(sizeof(struct media_link_desc) *
entity[index].links);

        for(index = 0; index < entities_count; index++) {

            links.entity = entity[index].id;

            ret = ioctl(media_fd, MEDIA_IOC_ENUM_LINKS, &links);
            if (ret < 0) {
                  if (errno == EINVAL)
                        break;
            }else{
                  /* display pads info first */
                  if(entity[index].pads)
                        printf("pads for entity %x=",
entity[index].id);

                  for(i = 0;i< entity[index].pads; i++)
                  {
                        printf("(%x, %s) ",
links.pads->index,(links.pads->flags &
MEDIA_PAD_FL_INPUT)?"INPUT":"OUTPUT");
                        links.pads++;
                  }

                  printf("\n");
```

```
                /* display links now */
                for(i = 0;i< entity[index].links; i++)
                {

printf("[%x:%x]-------------->[%x:%x]",links.links->source.entity,

links.links->source.index,links.links->sink.entity,links.links->sink.index);
                                if(links.links->flags &
MEDIA_LNK_FL_ENABLED)
                                     printf("\tACTIVE\n");
                                else
                                     printf("\tINACTIVE \n");

                        links.links++;
                }

                printf("\n");
           }
      }
```

Enable the appropriate links. It is about connecting the pads in such a way that fulfills the application's need for Video Capture. If RAW capture is needed, MT9P031 can be linked, If processed YUV is needed, TVP514x link is enabled. Here we are assuming RAW BAYER and hence MT9P031.

```
      /* 6. enable 'mt9p031-->ccdc' link */
      printf("6. ENABLEing link [tvp7002]----------->[ccdc]\n");
      memset(&link, 0, sizeof(link));

      link.flags |=  MEDIA_LNK_FL_ENABLED;
      link.source.entity = E_MT9P031;
      link.source.index = P_MT9P031;
      link.source.flags = MEDIA_PAD_FL_OUTPUT;

      link.sink.entity = E_CCDC;
      link.sink.index = P_CCDC_SINK;
      link.sink.flags = MEDIA_PAD_FL_INPUT;

      ret = ioctl(media_fd, MEDIA_IOC_SETUP_LINK, &link);
      if(ret) {
            printf("failed to enable link between tvp7002 and ccdc\n");
            goto cleanup;
      } else
            printf("[tvp7002]----------->[ccdc]\tENABLED\n");
```

```
      /* 7. enable 'ccdc->memory' link */
      printf("7. ENABLEing link [ccdc]----------->[video_node]\n");
      memset(&link, 0, sizeof(link));
```

```
      link.flags |=  MEDIA_LNK_FL_ENABLED;
      link.source.entity = E_CCDC;
      link.source.index = P_CCDC_SOURCE;
      link.source.flags = MEDIA_PAD_FL_OUTPUT;

      link.sink.entity = E_VIDEO;
      link.sink.index = P_VIDEO;
      link.sink.flags = MEDIA_PAD_FL_INPUT;

      ret = ioctl(media_fd, MEDIA_IOC_SETUP_LINK, &link);
      if(ret) {
            printf("failed to enable link between ccdc and video
node\n");
            goto cleanup;
      } else
            printf("[ccdc]----------->[video_node]\t ENABLED\n");

      printf("*****************************************\n");
```

Now that all the links are set properly, It is time to open the capture device.

```
      /* 14.open capture device */
      if ((capt_fd = open("/dev/video0", O_RDWR | O_NONBLOCK, 0)) <=
 -1) {
            printf("failed to open %s \n", "/dev/video0");
            goto cleanup;
      }
```

Enumerate the inputs. For camera,

```
      /* 15.enumerate inputs supported by capture*/
      printf("15.enumerating INPUTS\n");
      bzero(&input, sizeof(struct v4l2_input));
      input.type = V4L2_INPUT_TYPE_CAMERA;
      input.index = 0;
      index = 0;
        while (1) {

            ret = ioctl(capt_fd, VIDIOC_ENUMINPUT, &input);
            if(ret != 0)
                  break;

            printf("[%x].%s\n", index, input.name);

            bzero(&input, sizeof(struct v4l2_input));
            index++;
            input.index = index;
        }
```

Set Camera as the input.

```
	/* 16.setting CAMERA as input */
	printf("16. setting CAMERA as input. . .\n");
	bzero(&input, sizeof(struct v4l2_input));
	input.type = V4L2_INPUT_TYPE_CAMERA;
	input.index = 0;
	if (-1 == ioctl (capt_fd, VIDIOC_S_INPUT, &input.index)) {
		printf("failed to set CAMERA with capture device\n");
		goto cleanup;
	} else
		printf("successfully set CAMERA input\n");
```

Set the FORMAT on the output PAD of MT9P031. Here we are opening the appropriate sub-device node to do this. In this example, we know the sub-device number. However, it might not be known always.

```
	/* 8. set format on pad of mt9p031 */
	mt9p_fd = open("/dev/v4l-subdev0", O_RDWR);
	if(mt9p_fd == -1) {
		printf("failed to open %s\n", "/dev/v4l-subdev0");
		goto cleanup;
	}

	printf("8. setting format on pad of mt9p031 entity. . .\n");
	memset(&fmt, 0, sizeof(fmt));

	fmt.pad = P_MT9P031;
	fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
	fmt.format.code = CODE;
	fmt.format.width = width;
	fmt.format.height = height;
	fmt.format.field = V4L2_FIELD_NONE;

	ret = ioctl(mt9p_fd, VIDIOC_SUBDEV_S_FMT, &fmt);
	if(ret) {
		printf("failed to set format on pad %x\n", fmt.pad);
		goto cleanup;
	}
	else
		printf("successfully format is set on pad %x\n", fmt.pad);
```

Similarly, set the format on the sink-pad of CCDC ( Sink : input , Source : output ) for a given entity.

```
	/* 9. set format on sink-pad of ccdc */
	ccdc_fd = open("/dev/v4l-subdev1", O_RDWR);
	if(ccdc_fd == -1) {
		printf("failed to open %s\n", "/dev/v4l-subdev2");
		goto cleanup;
	}
	/* set format on sink pad of ccdc */
	printf("12. setting format on sink-pad of ccdc entity. . .\n");
```

```
	memset(&fmt, 0, sizeof(fmt));

	fmt.pad = P_CCDC_SINK;
	fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
	fmt.format.code = CODE;
	fmt.format.width = width;
	fmt.format.height = height;
	fmt.format.field = V4L2_FIELD_NONE;

	ret = ioctl(ccdc_fd, VIDIOC_SUBDEV_S_FMT, &fmt);
	if(ret) {
		printf("failed to set format on pad %x\n", fmt.pad);
		goto cleanup;
	}
	else
		printf("successfully format is set on pad %x\n", fmt.pad);
```

Now, set the format on the Source pad of CCDC which is connected to Video node.

```
	/* 13. set format on source-pad of ccdc */
	printf("13. setting format on OF-pad of ccdc entity. . . \n");
	memset(&fmt, 0, sizeof(fmt));

	fmt.pad = P_CCDC_SOURCE;
	fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
	fmt.format.code = CODE;
	fmt.format.width = width;
	fmt.format.height = height;
	fmt.format.colorspace = V4L2_COLORSPACE_SMPTE170M;
	fmt.format.field = V4L2_FIELD_NONE;

	ret = ioctl(ccdc_fd, VIDIOC_SUBDEV_S_FMT, &fmt);
	if(ret) {
		printf("failed to set format on pad %x\n", fmt.pad);
		goto cleanup;
	}
	else
		printf("successfully format is set on pad %x\n", fmt.pad);
```

Setup the CCDC for proper configuration which might include a host of internal parameters. This is done through a private sub-device IOCTL for CCDC - VPFE_CMD_S_CCDC_RAW_PARAMS

```
	/* 10. get ccdc raw params from ccdc*/
	printf("10. getting RAW params from ccdc\n");

	bzero(&raw_params, sizeof(raw_params));

	if (-1 == ioctl(ccdc_fd, VPFE_CMD_G_CCDC_RAW_PARAMS,
&raw_params)) {
```

```
		printf("failed to get raw params, %p", &raw_params);
		goto cleanup;
	}

	/* 11. set raw params in ccdc */
	printf("11. setting raw params in ccdc\n");
	raw_params.compress.alg = CCDC_NO_COMPRESSION;
	raw_params.gain_offset.gain.r_ye = r;
	raw_params.gain_offset.gain.gr_cy = gr;
	raw_params.gain_offset.gain.gb_g = gb;
	raw_params.gain_offset.gain.b_mg = b;
	raw_params.gain_offset.gain_sdram_en = 1;
	raw_params.gain_offset.gain_ipipe_en = 1;
	raw_params.gain_offset.offset = 0;
	raw_params.gain_offset.offset_sdram_en = 1;

	/* To test linearization, set this to 1, and update the
	 * linearization table with correct data
	 */
	if (linearization_en) {
		raw_params.linearize.en = 1;
		raw_params.linearize.corr_shft = CCDC_1BIT_SHIFT;
		raw_params.linearize.scale_fact.integer = 0;
		raw_params.linearize.scale_fact.decimal = 10;

		for (i = 0; i < CCDC_LINEAR_TAB_SIZE; i++)
			raw_params.linearize.table[i] = i;
	} else {
		raw_params.linearize.en = 0;
	}

	/* csc */
	if (csc_en) {
		raw_params.df_csc.df_or_csc = 0;
		raw_params.df_csc.csc.en = 1;
		/* I am hardcoding this here. But this should
		 * really match with that of the capture standard
		 */
		raw_params.df_csc.start_pix = 1;
		raw_params.df_csc.num_pixels = 720;
		raw_params.df_csc.start_line = 1;
		raw_params.df_csc.num_lines = 480;
		/* These are unit test values. For real case, use
		 * correct values in this table
		 */
		raw_params.df_csc.csc.coeff[0] = csc_coef_val;
		raw_params.df_csc.csc.coeff[1].decimal = 1;
```

```
            raw_params.df_csc.csc.coeff[2].decimal = 2;
            raw_params.df_csc.csc.coeff[3].decimal = 3;
            raw_params.df_csc.csc.coeff[4].decimal = 4;
            raw_params.df_csc.csc.coeff[5].decimal = 5;
            raw_params.df_csc.csc.coeff[6].decimal = 6;
            raw_params.df_csc.csc.coeff[7].decimal = 7;
            raw_params.df_csc.csc.coeff[8].decimal = 8;
            raw_params.df_csc.csc.coeff[9].decimal = 9;
            raw_params.df_csc.csc.coeff[10].decimal = 10;
            raw_params.df_csc.csc.coeff[11].decimal = 11;
            raw_params.df_csc.csc.coeff[12].decimal = 12;
            raw_params.df_csc.csc.coeff[13].decimal = 13;
            raw_params.df_csc.csc.coeff[14].decimal = 14;
            raw_params.df_csc.csc.coeff[15].decimal = 15;

      } else {
            raw_params.df_csc.df_or_csc = 0;
            raw_params.df_csc.csc.en = 0;
      }

      /* vertical line defect correction */
      if (vldfc_en) {
            raw_params.dfc.en = 1;
            // correction method
            raw_params.dfc.corr_mode = CCDC_VDFC_HORZ_INTERPOL_IF_SAT;
            // not pixels upper than the defect corrected
            raw_params.dfc.corr_whole_line = 1;
            raw_params.dfc.def_level_shift = CCDC_VDFC_SHIFT_2;
            raw_params.dfc.def_sat_level = 20;
            raw_params.dfc.num_vdefects = 7;
            for (i = 0; i < raw_params.dfc.num_vdefects; i++) {
                  raw_params.dfc.table[i].pos_vert = i;
                  raw_params.dfc.table[i].pos_horz = i + 1;
                  raw_params.dfc.table[i].level_at_pos = i + 5;
                  raw_params.dfc.table[i].level_up_pixels = i + 6;
                  raw_params.dfc.table[i].level_low_pixels = i + 7;
            }
            printf("DFC enabled\n");
      } else {
            raw_params.dfc.en = 0;
      }

      if (en_culling) {

            printf("Culling enabled\n");
            raw_params.culling.hcpat_odd  = 0xaa;
            raw_params.culling.hcpat_even = 0xaa;
```

```
            raw_params.culling.vcpat = 0x55;
            raw_params.culling.en_lpf = 1;
      } else {
            raw_params.culling.hcpat_odd  = 0xFF;
            raw_params.culling.hcpat_even = 0xFF;
            raw_params.culling.vcpat = 0xFF;
      }

      raw_params.col_pat_field0.olop = CCDC_GREEN_BLUE;
      raw_params.col_pat_field0.olep = CCDC_BLUE;
      raw_params.col_pat_field0.elop = CCDC_RED;
      raw_params.col_pat_field0.elep = CCDC_GREEN_RED;
      raw_params.col_pat_field1.olop = CCDC_GREEN_BLUE;
      raw_params.col_pat_field1.olep = CCDC_BLUE;
      raw_params.col_pat_field1.elop = CCDC_RED;
      raw_params.col_pat_field1.elep = CCDC_GREEN_RED;
      raw_params.data_size = CCDC_12_BITS;
      raw_params.data_shift = CCDC_NO_SHIFT;


      if (-1 == ioctl(ccdc_fd, VPFE_CMD_S_CCDC_RAW_PARAMS,
&raw_params)) {
            printf("failed to set raw params, %p", &raw_params);
            return -1;
      } else
            printf("successfully set raw params in ccdc\n");
```

Set the format on the Video node for capture. This format is used to store into the DDR.

```
      /* 17.setting format */
      printf("17. setting format V4L2_PIX_FMT_SBGGR16\n");
      CLEAR(v4l2_fmt);
      v4l2_fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      v4l2_fmt.fmt.pix.width = width;
      v4l2_fmt.fmt.pix.height = height;
      v4l2_fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_SBGGR16;
      v4l2_fmt.fmt.pix.field = V4L2_FIELD_NONE;

      if (-1 == ioctl(capt_fd, VIDIOC_S_FMT, &v4l2_fmt)) {
            printf("failed to set format on captute device \n");
            goto cleanup;
      } else
            printf("successfully set the format\n");

      /* 15.call G_FMT for knowing picth */
      if (-1 == ioctl(capt_fd, VIDIOC_G_FMT, &v4l2_fmt)) {
            printf("failed to get format from captute device \n");
            goto cleanup;
```

```
      } else {
            printf("capture_pitch: %x\n",
v4l2_fmt.fmt.pix.bytesperline);
            capture_pitch = v4l2_fmt.fmt.pix.bytesperline;
      }
```

Request for buffers. This is the standard V4L2 procedure of doing REQ_BUFS

```
      /* 18.make sure 3 buffers are supported for streaming */
      printf("18. Requesting for 3 buffers\n");
      CLEAR(req);
      req.count = 3;
      req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      req.memory = V4L2_MEMORY_USERPTR;

      if (-1 == ioctl(capt_fd, VIDIOC_REQBUFS, &req)) {
            printf("call to VIDIOC_REQBUFS failed\n");
            goto cleanup;
      }

      if (req.count != 3) {
            printf("3 buffers not supported by capture device");
            goto cleanup;
      } else
            printf("3 buffers are supported for streaming\n");
```

Initial set of pre-queing of buffers so that as soon as we start on streaming we can do the DQ-Q cycle. A minimum of 3 buffers are needed for minimum effective running of V4L2 capture. One buffer remains with the driver for current capture, one can be held with application and the other in a queued state so that a buffer is ready for next frame capture. This has do do with the sync nature of the hardware operation where the registers are shadowed and take effect at every VSYNC.

```
      /* 19.queue the buffers */
      printf("19. queing buffers\n");
      for (i = 0; i < 3; i++) {
            struct v4l2_buffer buf;
            CLEAR(buf);
            buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            buf.memory = V4L2_MEMORY_USERPTR;
            buf.index = i;
            buf.length = buf_size;
            buf.m.userptr = (unsigned long)capture_buffers[i].user_addr;

            if (-1 == ioctl(capt_fd, VIDIOC_QBUF, &buf)) {
                  printf("call to VIDIOC_QBUF failed\n");
                  goto cleanup;
            }
      }
```

Start Streaming!!

```
      /* 20.start streaming */
      CLEAR(type);
      type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      if (-1 == ioctl(capt_fd, VIDIOC_STREAMON, &type)) {
            printf("failed to start streaming on capture device");
            goto cleanup;
      } else
            printf("streaming started successfully\n");
```

The DQ-Q cycle.

```
      while(frame_count != 5) {

            CLEAR(cap_buf);

            cap_buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            cap_buf.memory = V4L2_MEMORY_USERPTR;
try_again:
            ret = ioctl(capt_fd, VIDIOC_DQBUF, &cap_buf);
            if (ret < 0) {
                  if (errno == EAGAIN) {
                        goto try_again;
                  }
                  printf("failed to DQ buffer from capture device\n");
                  goto cleanup;
            }

            temp = cap_buf.m.userptr;
            source = (char *)temp;

            /* copy frame to a file */
            for(i=0 ; i < height; i++) {
                  fwrite(source, 1 , width*2, file);
                  source += capture_pitch;
            }

            /* Q the buffer for capture, again */
            ret = ioctl(capt_fd, VIDIOC_QBUF, &cap_buf);
            if (ret < 0) {
                  printf("failed to Q buffer onto capture device\n");
                  goto cleanup;
            }

            frame_count++;

      }
```

Once done, make sure to do Stream-off to indicate the stop for the capture.

```
      /* 21. do stream off */
      CLEAR(type);
      type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      if (-1 == ioctl(capt_fd, VIDIOC_STREAMOFF, &type)) {
            printf("failed to stop streaming on capture device");
            goto cleanup;
      } else
            printf("streaming stopped successfully\n");
```

Start with relinquishing the linkages so a new/different set of linkages can be established for the next application.

```
cleanup:
 /* 24. de-enable all the links which are active right now */
      for(index = 0; index < entities_count; index++) {

            links.entity = entity[index].id;

            links.pads = malloc(sizeof( struct media_pad_desc) *
entity[index].pads);
            links.links = malloc(sizeof(struct media_link_desc) *
entity[index].links);

            ret = ioctl(media_fd, MEDIA_IOC_ENUM_LINKS, &links);
            if (ret < 0) {
                  if (errno == EINVAL)
                        break;
            }else{

                  for(i = 0;i< entity[index].links; i++)
                  {
                              if(links.links->flags &
MEDIA_LNK_FL_ENABLED) {
                                    /* de-enable the link */
                                    memset(&link, 0,
sizeof(link));

                                  link.flags |=
~MEDIA_LNK_FL_ENABLED;
                                  link.source.entity =
links.links->source.entity;
                                  link.source.index =
links.links->source.index;
                                  link.source.flags =
MEDIA_PAD_FL_OUTPUT;

                                  link.sink.entity =
links.links->sink.entity;
                                  link.sink.index =
```

```
links.links->sink.index;
                                    link.sink.flags =
MEDIA_PAD_FL_INPUT;

                                    ret = ioctl(media_fd,
MEDIA_IOC_SETUP_LINK, &link);
                                    if(ret) {
                                          printf("failed to de-enable
link \n");
                                    }

                                }

                        links.links++;
                }
            }
      }
```

Close the file Descriptors

```
      /* 25.close all the file descriptors */
      printf("closing all the file descriptors. . .\n");
      if(capt_fd) {
            close(capt_fd);
            printf("closed capture device\n");
      }
      if(ccdc_fd) {
            close(ccdc_fd);
            printf("closed ccdc sub-device\n");
      }
      if(mt9p_fd) {
            close(mt9p_fd);
            printf("closed mt9p031 sub-device\n");
      }
      if(media_fd) {
            close(media_fd);
            printf("closed  media device\n");
      }
      if(file) {
            fclose(file);
            printf("closed the file \n");
      }
      return ret;
}
```

# Article Sources and Contributors

**Community Linux PSP for DaVinci devices** *Source*: http://processors.wiki.ti.com/index.php?oldid=56898 *Contributors*: Manjunathhadli, Sudhakar.raj

**GSG: Booting the OMAP-L138/AM18x Out of the Box** *Source*: http://processors.wiki.ti.com/index.php?oldid=36641 *Contributors*: Hezhengting, ManishKhare, Mariana, SekharNori, Sudhakar.raj, Xyvonned

**Booting DM36x Out of the Box** *Source*: http://processors.wiki.ti.com/index.php?oldid=56457 *Contributors*: Sudhakar.raj

**GSG: DA8x/OMAP-L1/AM1x DVEVM Additional Procedures** *Source*: http://processors.wiki.ti.com/index.php?oldid=56628 *Contributors*: Akshay.s, Binary0101, D-allred, Dswag89, Jeff Cobb, Mariana, SekharNori, Sudhakar.raj, Sugumar, Xyvonned

**OMAP-L1 Linux Drivers Usage** *Source*: http://processors.wiki.ti.com/index.php?oldid=56416 *Contributors*: Chaithrika, ChristinaLam, Hezhengting, SekharNori, Sudhakar.raj, Sugumar

**UG: DaVinci PSP Installation on DM36x EVM** *Source*: http://processors.wiki.ti.com/index.php?oldid=56467 *Contributors*: Alexander.stohr, Israel DaVinci, Leon Luo, Nsnehaprabha, Sandeeppaulraj, Stsongas, Sudhakar.raj

**GSG: Building Software Components for OMAP-L1/AM1x** *Source*: http://processors.wiki.ti.com/index.php?oldid=56545 *Contributors*: A0272049, Akshay.s, Alexander.stohr, Arnier, Chaithrika, ChrisRing, DanRinkes, Dswag89, FarMcKon, Loc, ManishKhare, Mariana, Prakash.pm, SekharNori, Sudhakar.raj, Sugumar, Swami, Xyvonned

**Getting Started Guide for OMAP-L1** *Source*: http://processors.wiki.ti.com/index.php?oldid=55216 *Contributors*: Arnier, BrianBarrera, Dfriedland, FrankW, Jeff Cobb, ManishKhare, Mariana, Prakash.pm, SekharNori, Stsongas, Sudhakar.raj, Xyvonned

**Writing V4L2 Media Controller Applications on Dm36x Video Capture** *Source*: http://processors.wiki.ti.com/index.php?oldid=56675 *Contributors*: Manjunathhadli

# Image Sources, Licenses and Contributors

**Image:TIBanner.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha
**File:OMAP-L138_inifiles.zip** *Source*: http://processors.wiki.ti.com/index.php?title=File:OMAP-L138_inifiles.zip *License*: unknown *Contributors*: D-allred
**Image:NandAis.jpg** *Source*: http://processors.wiki.ti.com/index.php?title=File:NandAis.jpg *License*: unknown *Contributors*: Sudhakar.raj
**Image:SpiAis.jpg** *Source*: http://processors.wiki.ti.com/index.php?title=File:SpiAis.jpg *License*: unknown *Contributors*: Sudhakar.raj
**Image:NorAis.jpg** *Source*: http://processors.wiki.ti.com/index.php?title=File:NorAis.jpg *License*: unknown *Contributors*: Sudhakar.raj
**Image:NandAisOMAPL137.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:NandAisOMAPL137.png *License*: unknown *Contributors*: Sudhakar.raj
**Image:SpiAisOMAPL137.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:SpiAisOMAPL137.png *License*: unknown *Contributors*: Sudhakar.raj
**File:Subdev.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:Subdev.png *License*: unknown *Contributors*: Manjunathhadli
**File:Entities.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:Entities.png *License*: unknown *Contributors*: Manjunathhadli
**File:Continuous.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:Continuous.png *License*: unknown *Contributors*: Manjunathhadli
**File:Singleshot.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:Singleshot.png *License*: unknown *Contributors*: Manjunathhadli

# License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.
BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

## License

## 1. Definitions

1. "**Adaptation**" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
2. "**Collection**" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.
3. "**Creative Commons Compatible License**" means a license that is listed at http://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.
4. "**Distribute**" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.
5. "**License Elements**" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.
6. "**Licensor**" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
7. "**Original Author**" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
8. "**Work**" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.
9. "**You**" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
10. "**Publicly Perform**" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
11. "**Reproduce**" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

## 2. Fair Dealing Rights

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

## 3. License Grant

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
2. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
3. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
4. to Distribute and Publicly Perform Adaptations.
5. For the avoidance of doubt:

    1. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
    2. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
    3. **Voluntary License Schemes.** The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

## 4. Restrictions

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.
2. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.
3. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
4. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

## 5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

## 6. Limitation on Liability

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## 7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

## 8. Miscellaneous

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.