AM35x-OMAP35x-PSP 04.02.00.07

User Guide

AM35x-OMAP35x-PSP 04.02.00.07 UserGuide

Read This First

Important changes

1. The NAND driver in Linux kernel now uses 1-bit hardware ECC. The x-loader and u-boot have been updated to use same ECC scheme.

   It is, therefore, mandatory to update the x-loader and u-boot when using the Linux kernel from this release. Filesystem residing on the NAND partition will need to be re-flashed as well.

   See detailed procedure in the User Guide
2. The serial console device has been renamed `ttyOn` (from `ttySn`).

   This impacts the `bootargs` passed to the kernel and console definition in the `/etc/inittab` of existing filesystems.

   See section **Using the Correct Console Device** in this document.
3. List of parameters passed to create the JFFS2 image via `mkfs.jffs2` have changed.

   See section **Creating JFFS2 filesystem** in this document.
4. The steps for flashing the JFFS2 image to NAND from u-boot have changed.

   See section **Flashing from u-boot** in this document.
5. The Beagle and BeagleXM are added to list of supported platforms.
6. In Linux kernel (2.6.37), the entry corresponding to the root filesystem appearing in `/proc/mounts` has changed. This may cause failures in the init scripts.

   For example, when using NFS location for root filesystem, the DHCP script included in the filesystem may need to be changed.

   Click **here** for more details.

About this Manual

This document describes how to install and work with Texas Instruments' Platform Support Package (PSP) for OMAP35x, AM/DM37x, AM3517 platforms running Linux. This PSP provides a fundamental software platform for development, deployment and execution on. It abstracts the functionality provided by the hardware. The product forms the basis for all application development on these platforms.

In this context, the document contains instructions to:

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

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

Installation

System Requirements

Hardware Requirements:

- For AM/DM37x
  - OMAP3 Processor Module with AM37x ES1.2
  - OMAP3EVM Main Board (Rev G)
  - BeagleXM (Rev A1 and later)
- For AM3505/ AM3517
  - AM3517 Processor Module with ES1.2 Si
  - AM3517EVM (Rev C)
  - AM3517 Application board (Rev C)
- For OMAP35x
  - OMAP3 Processor Module with OMAP35x ES3.1 or later
  - OMAP3EVM Main Board (Rev G)
  - Beagle (Rev D and later)

Software Requirements:

- CodeSourcery ARM tool chain version 2009-q1

Important

This release has been system tested on the following platforms -

1. AM/DM37x Support : OMAP3 EVM (rev.G) and AM/DM37x Processor board with ES1.2 Si
2. AM3517 Support : AM3517 EVM (Rev C), Application board (Rev C) and ES1.2 Si

Only basic testing has been done on other platforms.

Package Contents

Extract the contents of release package with the following command:

```
$ tar -xvfz  AM35x-OMAP35x-LINUX-PSP-MM.mm.pp.bb.tgz
```

This creates a directory AM35x-OMAP35x-LINUX-PSP-MM.mm.pp.bb with the following contents:

```
\---AM35x-OMAP35x-LINUX-PSP-MM.mm.pp.bb
   |    Software-manifest.html
   +----docs
   |    |----DataSheet-MM.mm.pp.bb.pdf
   |    |----ReleaseNotes-MM.mm.pp.bb.pdf
   |    |----am3517
   |    |    `----UserGuide-MM.mm.pp.bb.pdf
   |    |----omap3530
   |    |    `----UserGuide-MM.mm.pp.bb.pdf
   +----host-tools
   |    |----linux
   |    |    `----signGP
   |    |----src
   |    |    `----signGP.c
```

```
+----images
|    |----boot-strap
|    |    |----am3517
|    |    |    `----x-load.bin.ift
|    |    |----omap3530
|    |    |    `----x-load.bin.ift
|    |    |----beagle
|    |    |    `----x-load.bin.ift
|    |----kernel
|    |    |----am3517
|    |    |    `----uImage
|    |    |----omap3530
|    |    |    `----uImage
|    |    |----beagle
|    |    |    `----uImage
|    |----u-boot
|    |    |----am3517
|    |    |    `----u-boot.bin
|    |    |----omap3530
|    |    |    `----u-boot.bin
|    |    |----beagle
|    |    |    `----uImage
+----scripts
|    |----am3517
|    |    |----Readme.txt
|    |    |----initenv-micron.txt
|    |    `----reflash-micron.txt
|    |----omap3530
|    |    |----Readme.txt
|    |    |----initenv-micron.txt
|    |    `----reflash-micron.txt
|    |    |----initenv-samsung.txt
|    |    `----reflash-samsung.txt
+----src
     |----boot-strap
     |    |----ChangeLog-MM.mm.pp.bb
     |    |----ShortLog
     |    |----Unified-patch-MM.mm.pp.bb.gz
     |    |----diffstat-MM.mm.pp.bb
     |    |----x-loader-patches-MM.mm.pp.bb.tar.gz
     |    `----x-loader-MM.mm.pp.bb.tar.gz
     |----examples
     |    |----examples.tar.gz
     |----kernel
     |    |----Readme.txt
     |    |----ChangeLog-MM.mm.pp.bb
     |    |----ShortLog
```

```
        |    |----Unified-patch-MM.mm.pp.bb.gz
        |    |----diffstat-MM.mm.pp.bb
        |    |----kernel-patches-MM.mm.pp.bb.tar.gz
        |    `----linux-MM.mm.pp.bb.tar.gz
        |----u-boot
             |----Readme.txt
             |----ChangeLog-MM.mm.pp.bb
             |----ShortLog
             |----Unified-patch-MM.mm.pp.bb.gz
             |----diffstat-MM.mm.pp.bb
             |----u-boot-patches-MM.mm.pp.bb.tar.gz
             `----u-boot-MM.mm.pp.bb.tar.gz
```

Important

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

Environment Setup

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

   For example, in bash:

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

2. Add location of u-boot tools to the PATH environment variable (required for `mkimage` utility that is built as part of u-boot)

   For example, in bash:

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

Note

Actual instructions and the path setting will depend upon your shell and location of the tools

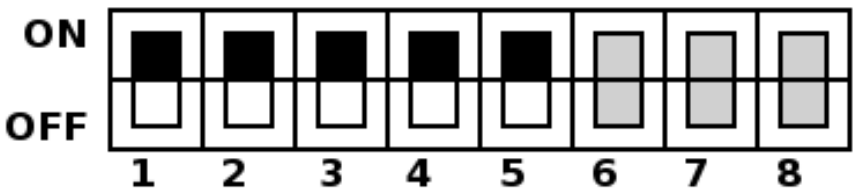
Selecting boot mode

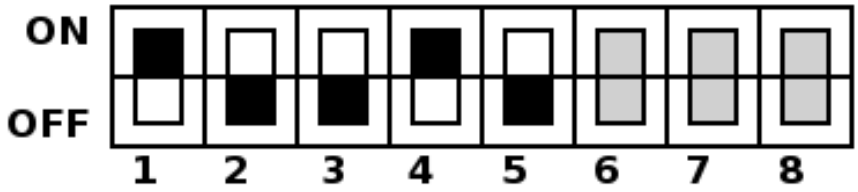
OMAP3EVM (OMAP35x, AM/DM37x)

The boot mode is selected by DIP switch SW4 on the main board. This selection identifies the location from where the x-loader and u-boot binaries are loaded for execution. The switch positions differ across the boards populated with Samsung OneNAND and Micron NAND parts.

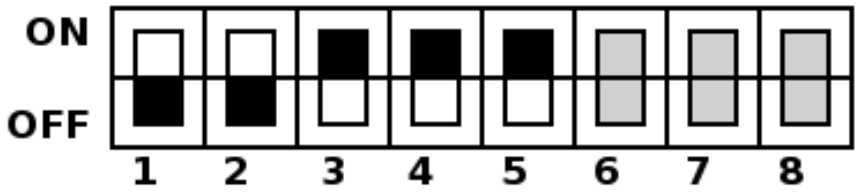
EVM populated with Samsung OneNAND

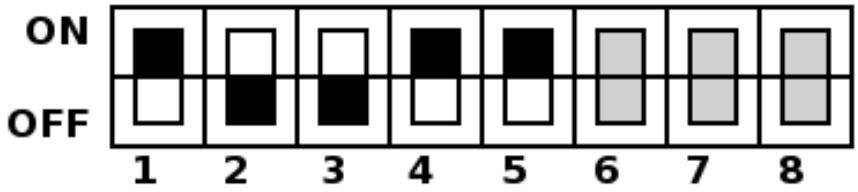
To boot from OneNAND, use either of following switch settings:





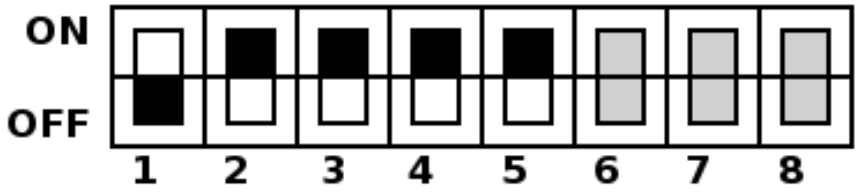
To boot from MMC (on EVM with Samsung OneNAND), use either of following switch settings:

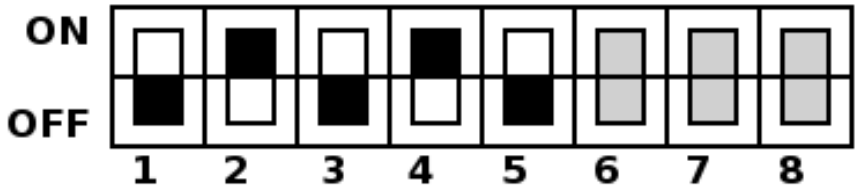




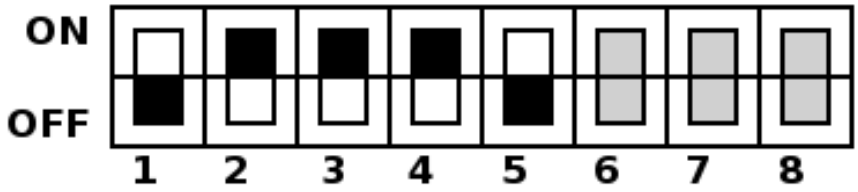
EVM populated with Micron NAND

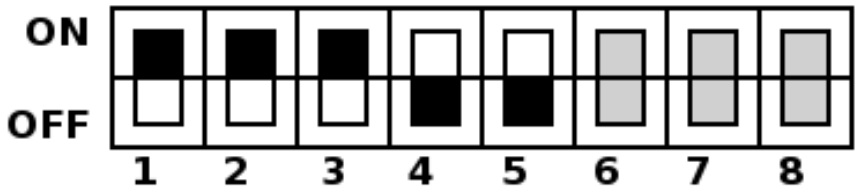
To boot from NAND, use either of following switch settings:





To boot from MMC (on EVM with Micron NAND), use either of following switch settings:





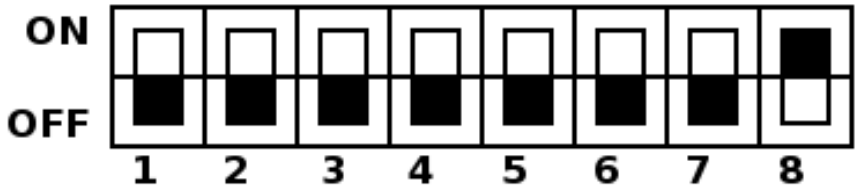
Note

Position of switches SW4-6, SW4-7 and SW4-8 is **Don't Care**. These are grayed in the illustrations above.

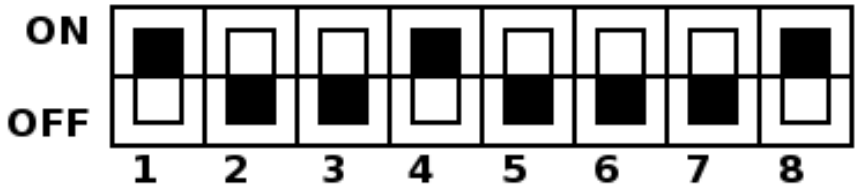
AM3517EVM

The boot mode is selected by DIP switch S7 on the main board. This selection identifies the location from where the **x-loader** and **u-boot** binaries are loaded for execution.

To boot from NAND, use this switch setting:



To boot from MMC, use this switch setting:



Beagle/ BeagleXM

On Beagle board by default system boots up in NAND boot mode, the boot mode is selected by User push button "S1" on the main board to force the system for MMC boot mode. Press the S1 push button and keep it holding while pressing and releasing User reset button S2. In case of BeagleXM, the only MMC boot mode supported, the switch is connected to GPIO4 only and doesn't function as a boot mode selector.

x-loader

Introduction

x-loader is the primary boot loader. It is loaded by ROM boot loader into the internal RAM. x-loader is responsible for initializing the external memory and loading the u-boot from the selected boot device.

x-loader supports boot from NAND, MMC/SD and OneNAND.

Compiling x-loader

1. Change to base of the x-loader directory.

   ```
   $ cd ./x-load
   ```

2. Remove the intermediate files generated during build. This step is not necessary when building for the first time.

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

3. Choose build configuration corresponding to the target platform

   - For OMAP3EVM:

     ```
     $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3evm_config
     ```

   - For AM3517EVM:

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517evm_config
```

- For Beagle/ BeagleXM:

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3beagle_config
```

4. Initiate the build

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

On successful completion, file `x-loader.bin` will be created in the current directory.

Signing the x-loader binary

The generated `x-loader.bin` needs to be signed before it can be used by the ROM bootloader. The `signGP` tool required for signing is available in the release package under the folder - host-tools/linux. To sign the x-loader binary:

```
$ signGP x-load.bin
```

This creates `x-load.bin.ift` in the current directory.

Selecting ECC scheme

x-loader supports following ECC schemes which need to be selected at compile time.

- 1-bit hardware ECC

  This is the default scheme. To explicitly select this scheme:

  ```
  $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm PLATFORM_RELFLAGS+='-DONE_BIT_ERROR_CORRECT'
  ```

- 4-bit hardware ECC

  To select this scheme:

  ```
  $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm PLATFORM_RELFLAGS+='-DFOUR_BIT_ERROR_CORRECT'
  ```

- 8-bit hardware ECC

  To select this scheme:

  ```
  $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm PLATFORM_RELFLAGS+='-DEIGHT_BIT_ERROR_CORRECT'
  ```

- 1-bit software ECC

  To select this scheme:

  1. Edit the file `include/configs/<board_config>.h`
  2. Undefine `ECC_HW_ENABLE`
     - Remove any existing definition (if any) for this macro
     - Add this line:

     ```
     #undef ECC_HW_ENABLE
     ```

U-boot

Compiling U-boot

1. Change to base of the u-boot directory.

   ```
   $ cd ./u-boot
   ```

2. Remove the intermediate files generated during build. This step is not necessary when building for the first time.

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

3. Choose build configuration corresponding to the target platform
   - For OMAP3EVM:

     ```
     $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3_evm_config
     ```

   - For AM3517EVM:

     ```
     $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517_evm_config
     ```

   - For Beagle/ BeagleXM:

     ```
     $ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3_beagle_config
     ```

4. Initiate the build

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

   On successful completion, file `u-boot.bin` will be created in the current directory.

U-Boot Features

The generic u-boot manual is located here [1]. It describes all generic u-boot commands.

This section describes platform specific features supported in the current release.

OneNAND Support

OneNAND is supported only on older OMAP3EVMs with OneNAND parts. The default configuration under U-boot enables Micron NAND support. To enable OneNAND the u-boot binary has to be re-compiled after enabling OneNAND Support (instead of NAND) in the config file.

Marking a bad block

To forcefully mark a block as bad:

```
OMAP3_EVM # onenand markbad <offset>
```

For example, to mark block 32 (assuming erase block size of 128Kbytes) as bad block - offset = blocknum * 128 * 1024:

```
OMAP3_EVM # onenand markbad 0x400000
```

Erasing OneNAND

To erase OneNAND blocks in the address range:

```
OMAP3_EVM # onenand erase <stoffaddr> <endoffaddr>
```

This command skips bad blocks (both factory or user marked) encountered within the specified range.

For example, to erase blocks 32 through 34:

```
OMAP3_EVM # onenand erase 0x00400000 0x00440000
```

Important
If the erase operation fails, the block is marked bad and the command aborts. To continue erase operation, the command needs to be re-executed for the remaining blocks in the range.

Writing to OneNAND

To write len bytes of data from a memory buffer located at addr to the OneNAND block offset:

```
OMAP3_EVM # onenand write <addr> <offset> <len>
```

If a bad block is encountered during the write operation, it is skipped and the write operation continues from next 'good' block.

For example, to write 0x40000 bytes from memory buffer at address 0x80000000 to OneNAND - starting at block 32 (offset 0x400000):

```
OMAP3_EVM # onenand write 0x80000000 0x400000 0x40000
```

Important
If the write fails on ECC check, the block where the failure occurred is marked bad and write operation is aborted. The command needs to be re-executed to complete the write operation. The offset and length for writing have to be page aligned else the command will abort.

Reading from OneNAND

To read len bytes of data from OneNAND block at offset to memory buffer located at addr:

```
OMAP3_EVM # onenand read <addr> <offset> <len>
```

If a bad block is encountered during the read operation, it is skipped and the read operation continues from next 'good' block

For example, to read 0x40000 bytes from OneNAND - starting at block 32 (offset 0x400000) to memory buffer at address 0x80000000:

```
OMAP3_EVM # onenand read 0x80000000 0x400000 0x40000
```

Important
If the read fails on ECC check, the block where the failure occurred is marked bad and read operation is aborted. The command needs to be re-executed to complete the read operation. But, the data in just marked bad block is irrecoverably lost. The offset and length for reading have to be page aligned else the command will abort.

Scrubbing OneNAND

This command operation is similar to the erase command, with a difference that it doesn't care for bad blocks. It attempts to erase all blocks in the specified address range.To scrub OneNAND blocks in the address range:

```
OMAP3_EVM # onenand scrub <stoffaddr> <eoffaddr>
```

If a bad block is encountered during the read operation, it is skipped and the read operation continues from next 'good' block

For example, to read 0x40000 bytes from OneNAND - starting at block 32 (offset 0x400000) to memory buffer at address 0x80000000:

```
OMAP3_EVM # onenand read 0x80000000 0x400000 0x40000
```

Important

- The command does not check whether the block is a user marked or factory marked bad block. This command fails on a factory marked bad block.
- If the erase operation fails, the block is marked as bad and the command aborts. The command needs to be re-executed for the remaining blocks in the range.

NAND Support

Micron Nand part (OMAP35x and AM3517) and Hynix Nand part (AM/DM37x) are supported.

Note
The following sub-sections illustrate the usage of NAND specific commands on OMAP3EVM. The same set of commands should work on AM3517, Beagle and BeagleXM as well.

NAND Layout

The NAND part on the EVM has been configured in the following manner. The addresses mentioned here are used in the subsequent NAND related commands.

```
 +------------+-->0x00000000-> X-loader start
 |            |
 |            |-->0x0007FFFF-> X-loader end
 |            |-->0x00080000-> U-Boot start
 |            |
 |            |-->0x001BFFFF-> U-Boot end
 |            |-->0x001C0000-> ENV start
 |            |
 |            |
 |            |-->0x0027FFFF-> ENV end
 |            |-->0x00280000-> Linux Kernel start
 |            |
 |            |
 |            |
 |            |
 |            |-->0x0077FFFF-> Linux Kernel end
 |            |-->0x00780000-> Filesystem start
 |            |
 |            |
 |            |
```

```
 |             |
 |             |
 |             |
 |             |
 |             |
 +-------------+-->0x10000000-> Filesystem end
```

Marking a bad block

To forcefully mark a block as bad:

```
OMAP3_EVM # nand markbad <offset>
```

For example, to mark block 32 (assuming erase block size of 128Kbytes) as bad block - offset = blocknum * 128 * 1024:

```
OMAP3_EVM # nand markbad 0x400000
```

Viewing bad blocks

To view the list of bad blocks:

```
OMAP3_EVM # nand bad
```

Note
The user marked bad blocks can be viewed by using this command only after a reset.

Erasing Nand

To erase NAND blocks in the address range or using block numbers:

```
OMAP3_EVM # nand erase <stoffaddr> <len>
```

This commands skips bad blocks (both factory or user marked) encountered within the specified range.

For example, to erase blocks 32 through 34:

```
OMAP3_EVM # nand erase 0x00400000 0x40000
```

Writing to Nand

To write len bytes of data from a memory buffer located at addrto the NAND block offset:

```
OMAP3_EVM # nand write <addr> <offset> <len>
```

If a bad block is encountered during the write operation, it is skipped and the write operation continues from next 'good' block.

For example, to write 0x40000 bytes from memory buffer at address 0x80000000 to NAND - starting at block 32 (offset 0x400000):

```
OMAP3_EVM # nand write 0x80000000 0x400000 0x40000
```

Reading from Nand

To read len bytes of data from NAND block at offset to memory buffer located at addr:

```
OMAP3_EVM # nand read <addr> <offset> <len>
```

If a bad block is encountered during the read operation, it is skipped and the read operation continues from next 'good' block.

For example, to read 0x40000 bytes from NAND - starting at block 32 (offset 0x400000) to memory buffer at address 0x80000000:

```
OMAP3_EVM # nand read 0x80000000 0x400000 0x40000
```

Selecting ECC algorithm

NAND flash memory, although cheap, suffers from problems like bit flipping which lead to data corruption. However by making use of some error correction coding (ECC) techniques it is possible to workaround this problem.

The ECC algorithm to be used can be be selected by the command `nandecc`:

```
# nandecc [ hw <hw_type> | sw | bch4_sw | bch8_sw ]

Usage:

   sw            - Set software ECC for NAND
   hw <hw_type> - Set hardware ECC for NAND
                  <hw_type> - 1 for Kernel/FileSystem ECC layout
                              2 for X-loader/U-boot ECC layout
   bch4_sw       - Set 4-bit BCH ECC for NAND
   bch8_sw       - Set 8-bit BCH ECC for NAND

  (hw 1 is set as the default nandecc)
```

NOR Support (Only on AM3517EVM)

Intel Strata Flash (8 Mbytes) is supported, Please refer to the Application Note available at AM35x-NOR-Flash-Support-ApplicationNote [2] for more details.

MUSB Host Support

The u-boot supports USB Mass storage class (MSC) on the MUSB port. It can be used to load any file from USB MSC device.

Note
Ensure that USB MSC device is connected to the MUSB port before issuing any of the commands described in this section

To initialize the USB subsystem:

```
OMAP3_EVM # usb start
```

All the connected devices will, now, get recognized.

To view all connected USB devices in a tree form:

```
OMAP3_EVM # usb tree
```

To view filesystem information of MSC device:

```
OMAP3_EVM # fatinfo usb D:P
```

This command shows filesystem information of a partition on the MSC device.

Note
Substitute D with the storage device number and p with the partition number on the device.

To load a file from MSC device:

```
OMAP3_EVM # fatload usb D:P <addr>ADDR> <file-name>
```

This command reads specified file from MSC device and writes its contents at the specified address.

Note
Substitute D with the storage device number and p with the partition number on the device

Flashing from u-boot

OneNAND

Saving Environment variables

User can use standard u-boot command to save environment variable.

For Example, To set bootargs and save them to the environment, the following commands could be used:

```
OMAP3_EVM # setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2'
OMAP3_EVM # saveenv
```

Flashing x-loader

To flash MLO (x-load.bin.ift) to the OneNAND Flash, execute the commands listed below:

```
OMAP3_EVM # mw.b 0x80000000 0xFF 0x100000
OMAP3_EVM # tftp 0x80000000 MLO
OMAP3_EVM # onenand erase 0x0 0x50000
OMAP3_EVM # onenand write 0x80000000 0x0 0x50000
```

Flashing U-boot

To flash u-boot.bin to the OneNAND Flash, execute the commands listed below:

```
OMAP3_EVM # mw.b 0x80000000 0xFF 0x100000
OMAP3_EVM # tftp 0x80000000 u-boot.bin
OMAP3_EVM # onenand erase 0x80000 0x1C0000
OMAP3_EVM # onenand write 0x80000000 0x80000 0x1C0000
```

Flashing Linux kernel

To flash uImage to the OneNAND Flash execute the commands listed below:

```
OMAP3_EVM # mw.b 0x80000000 0xFF 0x100000
OMAP3_EVM # tftp 0x80000000 uImage
OMAP3_EVM # onenand erase 0x280000 <kernel image size>
OMAP3_EVM # onenand write 0x80000000 0x280000 <kernel image size>
```

Flashing JFFS2 filesystem

To flash final.jffs2 to the OneNAND Flash execute the commands listed below:

```
OMAP3_EVM # mw.b 0x80000000 0xFF <file system size>
OMAP3_EVM # tftp 0x80000000 <file system image>
OMAP3_EVM # onenand erase 0x780000 <file system size>
OMAP3_EVM # onenand write 0x80000000 0x780000 <file system size>
```

NAND

Saving Environment variables

The default ECC scheme is 1-bit hardware ECC with Kernel/FileSystem ECC layout. Before saving the environment to NAND flash, **always** select this scheme. This will ensure that U-boot can read the environment from NAND without any ECC mismatch.

For example, to set bootargs and save them to the environment, the following commands could be used:

```
OMAP3_EVM # nandecc hw 1
OMAP3_EVM # setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2'
OMAP3_EVM # saveenv
```

Flashing x-loader

To flash MLO (x-load.bin.ift) to the NAND Flash, execute the commands listed below:

```
OMAP3_EVM # tftp 0x80000000 MLO
OMAP3_EVM # nand erase 0x0 0x50000
OMAP3_EVM # nandecc hw 2
OMAP3_EVM # nand write 0x80000000 0x0 0x50000
```

Flashing U-boot

To flash u-boot.bin to the NAND Flash, execute the commands listed below:

```
OMAP3_EVM # tftp 0x80000000 u-boot.bin
OMAP3_EVM # nand erase 0x80000 0x1C0000
OMAP3_EVM # nandecc hw 2
OMAP3_EVM # nand write 0x80000000 0x80000 0x1C0000
```

Flashing Linux kernel

To flash uImage to the NAND Flash execute the commands listed below:

```
OMAP3_EVM # tftp 0x80000000 uImage
OMAP3_EVM # nand erase 0x280000 <kernel image size>
OMAP3_EVM # nandecc hw 1
OMAP3_EVM # nand write 0x80000000 0x280000 <kernel image size>
```

Flashing JFFS2 filesystem

To flash final.jffs2 to the NAND Flash execute the commands listed below:

```
OMAP3_EVM # mw.b 0x80000000 0xFF <file system size>
OMAP3_EVM # tftp 0x80000000 <file system image>
OMAP3_EVM # nand erase 0x780000 <file system size>
OMAP3_EVM # nandecc hw 1
OMAP3_EVM # nand write 0x80000000 0x780000 <file system size>
```

Note

The image size should be upward aligned to NAND page size which is 2KiB (i.e. 0x800). For example, if the image size is 0x19B8004 the size to be passed to the NAND write command should be 0x19B8800.

NOR

Please refer to the Application notes available under - [2]

Linux Kernel

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

Compiling Linux Kernel

1. Change to the base of the Linux kernel source directory.

   $ `cd ./kernel`
2. Remove the intermediate files generated during build. This step is not necessary when building for the first time.

   $ `make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean`
3. Choose build configuration corresponding to the target platform
   - For OMAP3EVM:

     $ `make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3_evm_defconfig`
   - For AM3517EVM:

     $ `make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517_evm_defconfig`
   - For Beagle/ BeagleXM:

     $ `make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm omap3_beagle_defconfig`
4. Initiate the build

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

   On successful completion, file uImage will be created in the sub-directory `arch/arm/boot`. Copy this file to the root directory of your TFTP server.

Note

For the kernel image (uImage) to be built, location of `mkimage` utility must be included in the path. This utility is generated in tools sub-directory of u-boot sources when building u-boot.

Selecting NAND ECC scheme for Linux Kernel

Kernel is built with 1-bit hardware ECC scheme as a default ECC scheme. Currently, ECC scheme cannot be selected via menuconfig. Following steps should be followed to change default settings:

1. Edit files:
   - `arch/arm/mach-omap2/board-flash.c`
   - `drivers/mtd/nand/omap2.c`
   - `Replace the macro "OMAP_ECC_HAMMING_CODE_HW" with either "OMAP_ECC_BCH4_CODE_HW" or "OMAP_ECC_BCH8_CODE_HW" based on ECC requirement.`

Modifying Kernel Configuration

This table lists some of the key features and drivers supported by the default kernel configurations for all platforms supported in the release:

Driver	OMAP3EVM	BeagleBoard	AM3517EVM
Serial Port	Y	Y	Y
Ethernet	Y	Y	Y
MMC/SD	Y	Y	Y
Video Display (fbdev, v4l2)	Y	Y	Y
Video Capture	TVP514x, MT9T111	TVP514x, MT9T111	TVP514x
OneNAND	Y		
NAND	Y	Y	Y
NOR			Y[1]
Touchscreen	Y		Y
Keypad	Y		Y
WDT	Y	Y	Y
RTC	Y	Y	Y
Mentor USB (in OTG mode)	Y	Y	Y
USB EHCI	Y	Y	Y
Power Management	Y	Y	Y[2]

1 NOR Flash (PC28F640P30B85) is available on AM3517 Application Board

2 AM3517 supports basic suspend resume only

To view and modify the kernel configuration interactively:

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

Using the Correct Console Device

In the new kernel version, the serial console device has been renamed `ttyOn` (from `ttySn`). The serial port associated with UART0 is now `ttyO0` and so on...

The bootargs passed to kernel and definition of the terminal is the `/etc/inittab` needs must be updated to reflect this change.

Here is an illustration of changes required for `ttyS0`:

- Change the definition of `console` in bootargs

  from

  `console=ttyS0,115200n8`

  to

  `console=ttyO0,115200n8`

- Change the definition of `terminal` in `/etc/inittab` of the target filesystem

  from

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

  to

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

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

Creating JFFS2 filesystem

Since jffs2 is used only on flash devices, a standard Linux distribution does not have the tools to make a jffs2 file system. The Internet site http://sources.redhat.com/jffs2 explains where and how to obtain the latest source to MTD code and file systems. mkfs.jffs2 is the tool needed to build a jffs2 file system and the source is in this code. All of the MTD code will be downloaded but only the code to build mkfs.jffs2 is built.

The source code is maintained in a CVS tree. CVS is version control software and is usually installed when the Linux distribution is installed. CVS does not work across firewalls so a direct connection to the Internet is required. From the command line enter the following:

```
[root@localhost user]# cd /home/user/build
[root@localhost build]# cvs
 -d :pserver:anoncvs@cvs.infradead.org:/home/cvs login CVS password:
anoncvs
[root@localhost build]# cvs
 -d :pserver:anoncvs@cvs.infradead.org:/home/cvs co mtd
```

There should now be a mtd directory under /home/user/build. The source to mkfs.jffs2 is found in the util directory under mtd. To build mkfs.jffs2 simply change directory to mtd/util and enter “make mkfs.jffs2”. When the build is complete, copy the mkfs.jffs2 file to the /sbin directory; that is where the other utilities that make file systems reside.

```
[root@localhost build]# cd mtd/util
[root@localhost util]# make mkfs.jffs2
[root@localhost util]# cp mkfs.jffs2 /sbin
```

With the mkfs.jffs2 utility built and in place it is now time to make the jffs2 file system from of the target directory. Change to the /home/user directory and enter the mkfs.jffs2 command below. Probably the most important argument to the utility is the erase block size. For the NAND part on the EVM board the erase block is 128k Consult either

u-boot, the kernel, or the data manual for the flash part used to find the erase block size.

```
[root@localhost util]# cd /home/user
[root@localhost util]# mkfs.jffs2 -lqn —e 128 -r target -o
/tftpboot/rd-jffs2.bin
```

By building the file in the /tftpboot directory, the step of copying it over is eliminated. The file must now be written into flash at a specific location. In u-boot, the flash file system will get flashed to the physical address 0x780000 and will be mounted by the kernel from /dev/mtdblock<partition-number>, partition-number starts from 0, refer flash layout for exact number. The steps to perform this are:

Unprotect the flash area where the file system will reside.

Erase that area. Download the rd-jffs2.bin file. Write it to flash. Modify bootargs to support a JFFS2 file system as root on /dev/mtdblock<partition-number>. Save the environment variables.

Booting Linux Kernel

Boot from NAND

Select the boot mode as defined in section Selecting boot mode. Power on EVM and wait for u-boot to come up.

When kernel image and filesystem are flashed on the NAND device:

```
$ nand read.i 0x80000000 280000 500000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2 ip=dhcp'
$ bootm 0x80000000
```

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

```
$ nand read.i 0x80000000 280000 500000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd rw
root=/dev/nfs nfsroot=/mnt/nfs,nolock ip=dhcp'
$ bootm 0x80000000
```

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

```
$ setenv autoload no
$ dhcp
$ setenv serverip <Server IP Address>
$ tftp 0x80000000 uImage
$ tftp 0x82000000 ramdisk.gz
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 root=/dev/ram0
initrd=0x82000000,40M ramdisk_size=32768 ip=dhcp'
$ bootm 0x80000000
```

Boot from OneNAND

Select the boot mode as defined in section Selecting boot mode. Power on EVM and wait for u-boot to come up.

When kernel image and filesystem are flashed on the OneNAND device:

```
$ onenand read 0x80000000 0x280000 0x0220000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2 ip=dhcp'
$ bootm 0x80000000
```

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

```
$ onenand read 0x80000000 0x280000 0x0220000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd rw
root=/dev/nfs nfsroot=/mnt/nfs,nolock ip=dhcp'
$ bootm 0x80000000
```

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

```
$ setenv autoload no
$ dhcp
$ setenv serverip <Server IP Address>
$ tftp 0x80000000 uImage
$ tftp 0x82000000 ramdisk.gz
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 root=/dev/ram0
initrd=0x82000000,40M ramdisk_size=32768 ip=dhcp'
$ bootm 0x80000000
```

Please note that steps for NAND boot mode are applicable to all platforms, where NAND Flash is supported. Please make sure to use right console number, e.g. in case of AM3517EVM, we use ttyO2, so please change ttyO0 => ttyO2.

Boot from MMC

Select the boot mode as defined in section Selecting boot mode. Power on EVM and wait for u-boot to come up.

When kernel image and filesystem (ramdisk) are available on the MMC card:

```
$ mmc init
$ fatload mmc 0 0x80000000 uImage
$ fatload mmc 0 0x82000000 ramdisk.gz
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 root=/dev/ram0
initrd=0x82000000,40M ramdisk_size=32768 ip=dhcp'
$ bootm 0x80000000
```

When kernel image is available on the MMC card and NFS mounted filesystem is being used:

```
$ mmc init
$ fatload mmc 0 0x80000000 uImage
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd rw
root=/dev/nfs nfsroot=/mnt/nfs,nolock ip=dhcp'
$ bootm 0x80000000
```

When kernel image is available on the MMC card and filesystem on the NAND device is used:

```
$ mmc init
$ fatload mmc 0 0x80000000 uImage
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2 ip=dhcp'
$ bootm 0x80000000
```

Boot from NOR

Configure the EVM in XIP boot mode as mentioned in the "Selecting boot mode" section above, power on the EVM and wait for u-boot to come up.

When kernel image and filesystem (ramdisk) are available on the MMC card:

```
$ cp.b 0x80000000 0x08100000 0x400000
$ cp.b 0x82000000 0x08500000 0x2000000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 root=/dev/ram0
initrd=0x82000000,40M ramdisk_size=32768 ip=dhcp'
$ bootm 0x80000000
```

When kernel image is available on the MMC card and NFS mounted filesystem is being used:

```
$ cp.b 0x80000000 0x08100000 0x400000
$ cp.b 0x82000000 0x08500000 0x2000000
$ setenv bootargs 'mem=128M console=ttyO0,115200n8 noinitrd rw
root=/dev/nfs nfsroot=/mnt/nfs,nolock ip=dhcp'
$ bootm 0x80000000
```

Note

Once the Linux kernel boots, login as "root". No password is required.

Important

In the latest kernel (2.6.37) the entry corresponding to the root filesystem appearing in `/proc/mounts` has changed. Unless updated, the init scripts in the filesystem that use this entry to determine the device on which root filesystem is mounted will fail.

For example, the script `/etc/udhcpc.d/50default` doesn't send DHCP request if the root filesystem is network mounted. Although, the filesystem is successfully mounted, boot process my stall after showing errors from udhcpc.

The issue is fixed by updating the function `root_is_nfs()` as shown below:

Original code:

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

Updated code:

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

When using an existing filesystem, review the init scripts to locate similar instances and make appropriate changes.

Audio Driver

→ Audio Driver UserGuide

Video Display Driver

→ Display Driver UserGuide

Video Capture Driver

- → OMAP35x, AM/DM37x Video Capture Driver UserGuide
- → AM3517 Capture Driver UserGuide

USB Driver

→ USB Driver UserGuide

MMC Driver

→ MMC Driver UserGuide

Ethernet Driver

→ Ethernet Driver UserGuide

Power Management

- → OMAP35x/DM37x Power Management UserGuide
- → AM35x Power Management UserGuide

Power Management IC

→ Power Management IC UserGuide

TI HECC CAN Controller

→ TI HECC CAN Controller

References

[1] http://www.denx.de/wiki/view/DULG/UBoot
[2] http://processors.wiki.ti.com/index.php/AM35x-NOR-Flash-Support-ApplicationNote

UserGuideAudioDriver PSP 04.02.00.07

```
Audio Driver
```

Introduction

Audio module is available inside PMIC IC TPS65950 (in OMAP35xEVM, AM37xEVM, BEAGLE, BEAGLEXM) and AIC23 (in AM3517EVM), contains audio analog inputs and outputs. It is connected to the main processor through the TDM / I2S interface and used to transmit and receive audio data. The TPS65950 / AIC23 audio codec is connected via Multi-Channel Buffered Serial Port (McBSP) interface, a communication peripheral, to the main processor.

McBSP provides a full-duplex direct serial interface between the main processor (AM/DM37x, OMAP35x and AM3517) and other devices in the system such as the TPS65950 / AIC23 codec. It provides a direct interface to industry standard codecs, analog interface chips (AICs) and other serially connected A/D and D/A devices:

- Inter-IC Sound (I2S) compliant devices
- Pulse Code Modulation (PCM) devices
- Time Division Multiplexed (TDM) bus devices.

The TPS65950 / AIC23 audio module is controlled by internal registers that can be accessed by the high speed I2C control interface.

This user manual defines and describes the usage of user level and platform level interfaces of the ALSA SoC Audio driver.

References

1. ALSA SoC Project Homepage [http://www.alsa-project.org/main/index.php/ASoC [1]]
2. ALSA Project Homepage [http://www.alsa-project.org/main/index.php/Main_Page [2]]
3. ALSA User Space Library [http://www.alsa-project.org/alsa-doc/alsa-lib/ [3]]
4. Using ALSA Audio API [http://www.equalarea.com/paul/alsa-audio.html [4]] Author: Paul Davis
5. Using alsamixer interface [http://linux.die.net/man/1/alsamixer [5]]
6. TPS65950: Integrated Power Management IC with 3 DC/DC's, 11 LDO's, Audio Codec, USB HS Transceiver, Charger. [http://focus.ti.com/docs/prod/folders/print/tps65950.html [6]
7. TLV320AIC23B - Low-Power Stereo CODEC with HP Amplifier. Literature Number: SLWS106H. [[tlv320aic23b.pdflfocus.ti.com/lit/ds/symlink/ tlv320aic23b.pdf [7]]

Acronyms & Definitions

Audio Driver: Acronyms

Acronym	Definition
ALSA	Advanced Linux Sound Architecture
ALSA SoC	ALSA System on Chip
DMA	Direct Memory Access
I2C	Inter-Integrated Circuit
McBSP	Multi-channel Buffered Serial Port
PCM	Pulse Code Modulation
TDM	Time Division Multiplexing
OSS	Open Sound System
I2S	Inter-IC Sound

Features

This section describes the features supported by ALSA SoC Audio driver.

- Supports AIC23 audio codec (on AM3517 only) and TPS65950 audio codec (on AM/DM37x,OMAP35x only) in ALSA SoC framework.
- Multiple sample rates support (8KHz, 16KHz, 22.05KHz, 32KHz, 44.1KHz, 48KHz, 64KHz, 88.2KHz and 96KHz - AM3517; 8 KHz, 11.025 KHz, 12 KHz, 16 KHz, 22.05 KHz, 24 KHz, 32 KHz, 44.1 KHz and 48 KHz - AM/DM37x,OMAP35x) for both capture and playback.
- Supports audio in stereo mode.
- Supports simultaneous playback and record (full-duplex mode).
- Start, stop, pause and resume feature.
- Supports mixer interface for audio codecs.
- Supports MMAP mode for both playback and capture.
- McBSP is configured as slave and TPS65950 / AIC23 Codec is configured as master.

Important (for AM/DM37x,OMAP35x ONLY)

- Audio capture channels AUXL and AUXR are by default disabled for AM/DM37x,OMAP35x ASoC driver. To enable,hit the <space> key on channels "Analog Left AUXL" & "Analog Right AUXR" in the alsamixer interface under Capture tab. Or use the following 'amixer' commands:

```
amixer cset name='Analog Left AUXL Capture Switch' 1
amixer cset name='Analog Right AUXR Capture Switch' 1
```

- Audio playback (on Headset channel) is muted by default. On alsamixer interface use the key "m" to unmute "HeadsetL Mixer AudioL1" & "HeadsetR Mixer AudioR1" channels under the Playback tab.

Or use the following amixer commands:

```
amixer cset name='HeadsetL Mixer AudioL1' on
amixer cset name='HeadsetR Mixer AudioR1' on
```

Important (for AM3517 ONLY)

By default Line is enabled under [Capture] tab in alsamixer instead of Mic.To use microphone input instead, enable Mic setting under [Capture] tab and unmute (using the "m" key) the Mic Boost setting under [Playback]

- When Line is enabled (default) under [Capture] tab in alsamixer, connect an audio player source to Line In Codec 1 jack on the application board.

- When Mic is enabled under [Capture] tab in alsamixer (and Mic boost is un-muted), connect a headset microphone to the Mic In Codec 1 jack on the application board.

Important
Please note that enabling any line-in inputs necessitates connecting an audio playback source's output ; connecting a (non-preamplified) microphone input (like the one from a head-set jack) might not work.

ALSA SoC Architecture

Introduction

The overall project goal of the ALSA System on Chip (ASoC) layer is to provide better ALSA support for embedded system on chip processors and portable audio codecs. Currently there is some support in the kernel for SoC audio, however it has some limitations:

- Currently, codec drivers are often tightly coupled to the underlying SoC cpu. This is not really ideal and leads to code duplication.
- There is no standard method to signal user initiated audio events e.g. Headphone/Mic insertion, Headphone/Mic detection after an insertion event.
- Current drivers tend to power up the entire codec when playing (or recording) audio. This is fine for a PC, but tends to waste a lot of power on portable devices. There is also no support for saving power via changing codec oversampling rates, bias currents, etc.

Design

The ASoC layer is designed to address these issues and provide the following features:

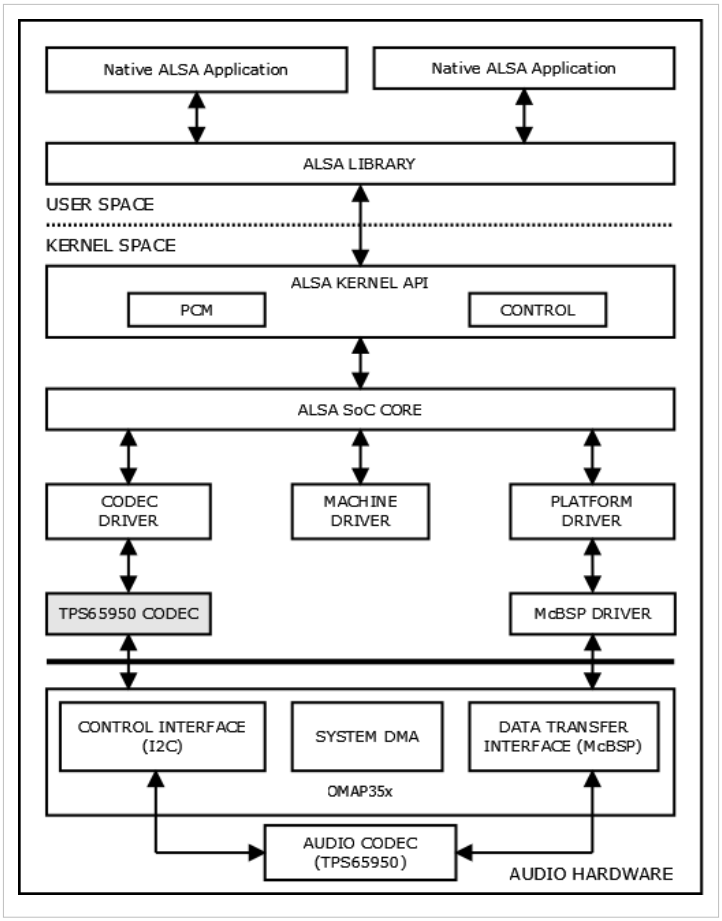
- **Codec independence**: Allows reuse of codec drivers on other platforms and machines.
- **Easy I2S/PCM audio interface setup** between codec and SoC. Each SoC interface and codec registers it's audio interface capabilities with the core and are subsequently matched and configured when the application hw params are known.
- **Dynamic Audio Power Management (DAPM)**: DAPM automatically sets the codec to it's minimum power state at all times. This includes powering up/down internal power blocks depending on the internal codec audio routing and any active streams.
- **Pop and click reduction**: Pops and clicks can be reduced by powering the codec up/down in the correct sequence (including using digital mute). ASoC signals the codec when to change power states.

To achieve all this, ASoC basically splits an embedded audio system into three components:

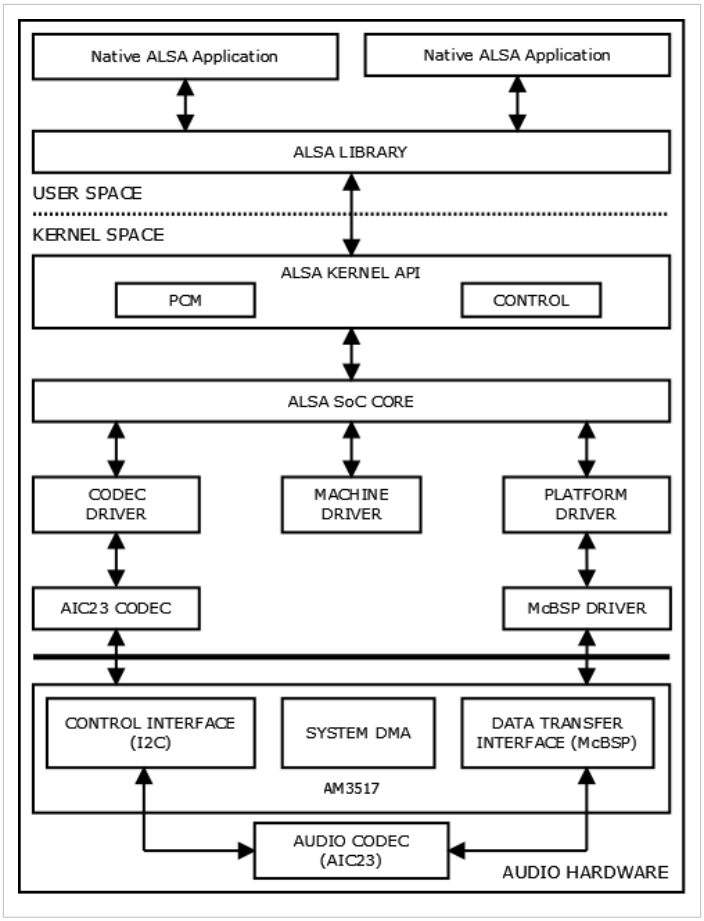
- **Codec driver**: The codec driver is platform independent and contains audio controls, audio interface capabilities, codec dapm definition and codec IO functions.
- **Platform driver**: The platform driver contains the audio dma engine and audio interface drivers (e.g. I2S, AC97, PCM) for that platform.
- **Machine driver**: The machine driver handles any machine specific controls and audio events i.e. turning on an amp at start of playback.

Following architecture diagram shows all the components and the interactions among them:

AM/DM37x, OMAP35x:


Native ALSA Application
Native ALSA Application
ALSA LIBRARY
USER SPACE
KERNEL SPACE
ALSA KERNEL API
PCM
CONTROL
ALSA SoC CORE
CODEC DRIVER
MACHINE DRIVER
PLATFORM DRIVER
TPS65950 CODEC
McBSP DRIVER
CONTROL INTERFACE (I2C)
SYSTEM DMA
DATA TRANSFER INTERFACE (McBSP)
OMAP35x
AUDIO CODEC (TPS65950)
AUDIO HARDWARE

AM3517:



Configuration

To enable/disable audio support, start the *Linux Kernel Configuration* tool:

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

Select *Device Drivers* from the main menu.

```
...
    ...
    Power management options --->
    [*] Networking support --->
    '''Device Drivers --->'''
    File systems --->
    Kernel hacking --->
    ...
    ...
```

Select *Sound card support* as shown here:

```
...
    ...
    Multimedia devices --->
    Graphics support --->
'''<*> Sound card support --->'''
```

```
[*] HID Devices --->
[*] USB support --->
    ...
    ...
```

Select *Advanced Linux Sound Architecture* as shown here:

```
--- Sound card support
'''<*> Advanced Linux Sound Architecture --->'''
< > Open Sound System (DEPRECATED) --->
```

Select *ALSA for SoC audio support* as shown here:

```
...
    ...
[*] ARM sound devices  --->
[*] SPI sound devices  --->
[*] USB sound devices  --->
'''<*> ALSA for SoC audio support --->'''
```

For AM/DM37x,OMAP35x, select *SoC Audio support for OMAP3EVM board* as shown here:

```
--- ALSA for SoC audio support
<*> SoC Audio for the Texas Instruments OMAP chips
'''<*> SoC Audio support for OMAP3EVM board'''
< > Build all ASoC CODEC drivers
```

For AM3517, select *SoC Audio support for OMAP3517 / AM3517 EVM* as shown here:

```
--- ALSA for SoC audio support
<*> SoC Audio for the Texas Instruments OMAP chips
'''<*> SoC Audio support for OMAP3517 / AM3517 EVM'''
< > Build all ASoC CODEC drivers
```

Make sure that McBSP support is enabled. To check the same, select *System Type* from the main menu.

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

Select *TI OMAP Common Features* as shown here:

```
[*] MMU-based Paged Memory Management Support
    ARM system type (TI OMAP) --->
    '''TI OMAP Common Features --->'''
    TI OMAP2/3/4 Specific Features  --->
    *** Processor Type ***
```

```
-*- Support ARM V6K processor extensions
    ...
    ...
```

McBSP support should be selected:

```
...
    ...
[ ] Multiplexing debug output
[*] Warn about pins the bootloader didn't set up
-*- McBSP support
< > Mailbox framework support
< > Export OMAP IOMMU internals in DebugFS
    System timer (Use 32KHz timer) --->
    ...
    ...
```

Application Interface

This section provides the details of the Application Interface for the ALSA Audio driver.

Application developer uses ALSA-lib, a user space library, rather than the kernel API. The library offers 100% of the functionality of the kernel API, but adds major improvements in usability, making the application code simpler and better looking.

The online-documentation for the same is available at:

http://www.alsa-project.org/alsa-doc/alsa-lib/ [3]

Device Interface

The operational interface in `/dev/` contains three main types of devices:

- PCM devices for recording or playing digitized sound samples,
- CTL devices that allow manipulating the internal mixer and routing of the card, and,
- MIDI devices to control the MIDI port of the card, if any.

Device Interface

Name	Description
/dev/snd/controlC0	Control devices (i.e. mixer, etc)
/dev/snd/pcmC0D0c	PCM Card 0 Device 0 Capture device
/dev/snd/pcmC0D0p	PCM Card 0 Device 0 Playback device

Proc FS Interface

The `/proc/asound` kernel interface is used as a status and configuration interface. A lot of useful information about the sound system can be found in the `/proc/asound` subdirectory.

Please refer to the below table for different proc entries in `/proc/asound` for Audio driver:

Proc Interface

Name	Description
cards	List of registered cards
version	Version and date the driver was built on
devices	List of registered ALSA devices
pcm	The list of allocated PCM streams
cardX/ (X = 0-7)	The card specific directory
cardX/pcm0p	The directory of the given PCM playback stream
cardX/pcm0c	The directory of the given PCM capture stream

Sys FS Interface

The `/sys` FS interface is used to configure settings for the audio interface and to extract information.

Please refer to the below table for list of various `/sys` entries for the audio driver:

Proc Interface

Name	Description
`sys/devices/platform/soc-audio/TWL4030` `sys/devices/platform/soc-audio/TLV320AIC23`	Information on the codec device. e,g. codec_reg attribute provides the register dump
`/sys/devices/platform/omap-pcm-audio`	Interface for the platform pcm(ALSA) device
`/sys/devices/platform/omap-mcbsp-dai.x`	Interface for the McBSP device; x is 0 for AM3517 and 1 for AM/DM37x & OMAP35x

Commonly Used APIs

Some of the commonly used APIs to write an ALSA based application are:

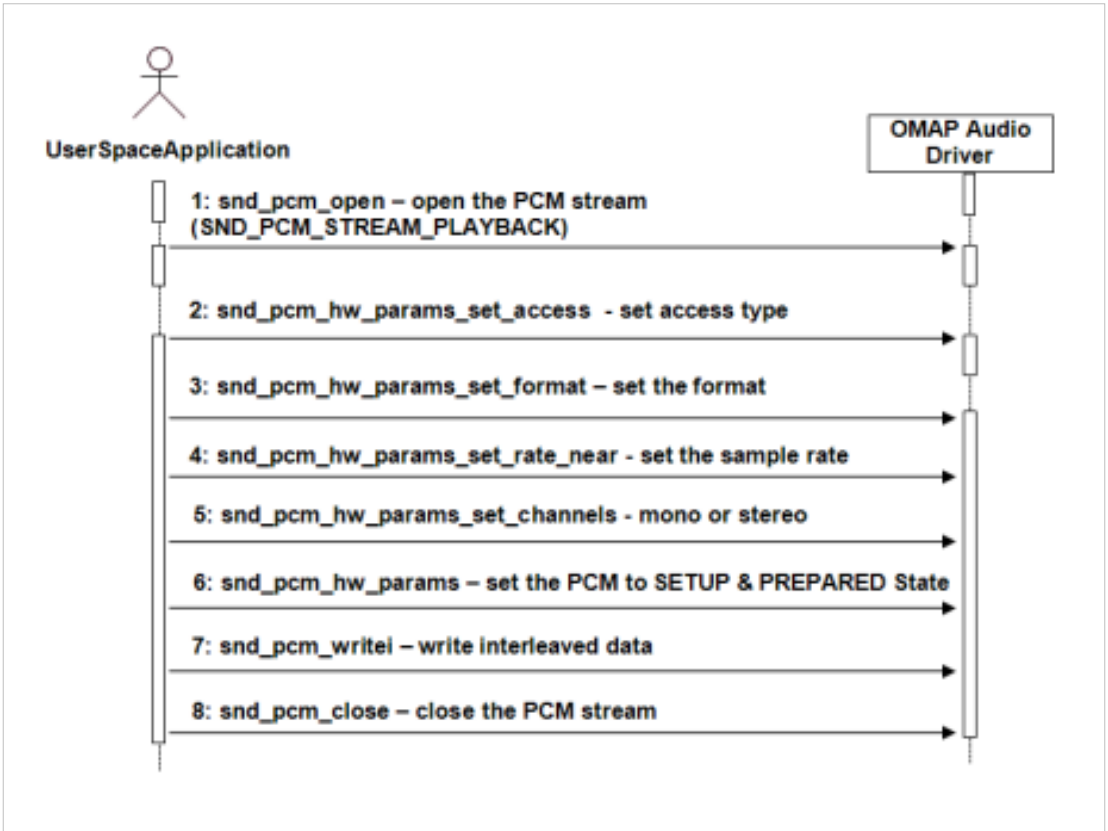
Commonly Used APIs

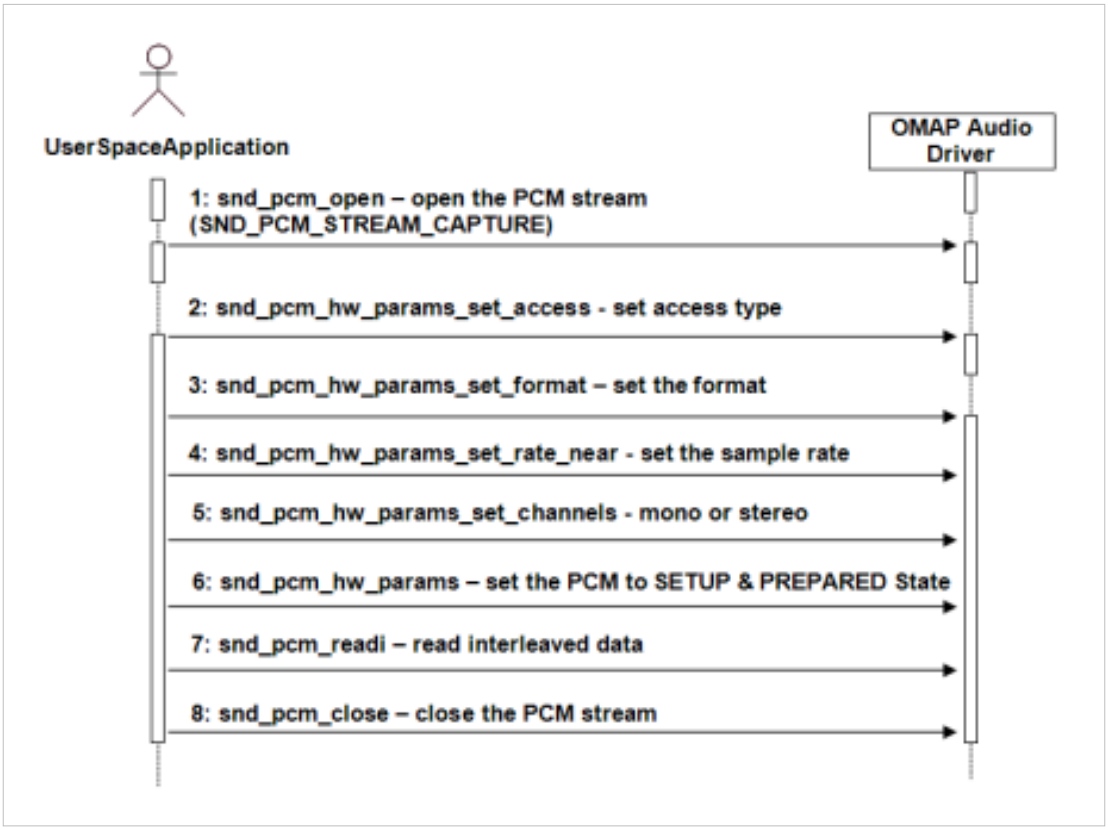
Name	Description
snd_pcm_open	Opens a PCM stream
snd_pcm_close	Closes a previously opened PCM stream
snd_pcm_hw_params_any	Fill params with a full configuration space for a PCM
snd_pcm_hw_params_test_ <<parameter>>	Test the availability of important parameters like number of channels, sample rate etc. For e.g. snd_pcm_hw_params_test_format, snd_pcm_hw_params_test_rate etc.
snd_pcm_hw_params_set_ <<parameter>>	Set the different configuration parameters. For e.g. snd_pcm_hw_params_set_format, snd_pcm_hw_params_set_rate etc.
snd_pcm_hw_params	Install one PCM hardware configuration chosen from a configuration space

snd_pcm_writei	Write interleaved frames to a PCM
snd_pcm_readi	Read interleaved frames from a PCM
snd_pcm_prepare	Prepare PCM for use
snd_pcm_drop	Stop a PCM dropping pending frames
snd_pcm_drain	Stop a PCM preserving pending frames

User Space Interactions

This section depicts the sequence of operations for a simple playback and capture application.





Sample Applications

This chapter describes the audio sample applications provided along with the package. The source for these sample applications are available in the Examples directory of the Release Package folder.

The applications arecord, aplay and amixer are also available in the alsa-utils package available at ftp:// ftp. alsa-project.org/pub/utils/alsa-utils-1.0.24.2.tar.bz2

The source needs to be cross-compiled using the Code-Sourcery tool-chain (i,e. arm-none-linux-gnueabi- prefixed) and needs libasound (currently tested with /usr/lib/libasound.so.2) to run on these platforms.

Writing a simple audio application

Writing an audio application involves the following steps:

- Opening the audio device
- Set the parameters of the device
- Receive audio data from the device or deliver audio data to the device
- Close the device

These steps are explained in detail in this section.

Note

User space ALSA libraries can be downloaded from this link http:// www. alsa-project. org/ main/ index. php/ Download [8]. User needs to build and install them before he starts using the ALSA based applications.

A minimal playback application

This program opens an audio interface for playback, configures it for stereo, 16 bit, 44.1kHz, interleaved conventional read/write access. Then its delivers a chunk of random data to it, and exits. It represents about the simplest possible use of the ALSA Audio API, and isn't meant to be a real program.

Opening the audio device

To write a simple PCM application for ALSA, we first need a handle for the PCM device. Then we have to specify the direction of the PCM stream, which can be either playback or capture. We also have to provide some information about the configuration we would like to use, like buffer size, sample rate, pcm data format. So, first we declare:

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

#define BUFF_SIZE 4096

int main (int argc, char *argv[])
{
  int err;
  short buf[BUFF_SIZE];
  int rate = 44100; /* Sample rate */
  unsigned int exact_rate; /* Sample rate returned by */
  /* Handle for the PCM device */
  snd_pcm_t *playback_handle;
  /* Playback stream */
  snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
  /* This structure contains information about the hardware and can be
used to specify the configuration to be used for */
  /* the PCM stream. */
  snd_pcm_hw_params_t *hw_params;
```

The most important ALSA interfaces to the PCM devices are the "plughw" and the "hw" interface. If you use the "plughw" interface, you need not care much about the sound hardware. If your sound card does not support the sample rate or sample format you specify, your data will be automatically converted. This also applies to the access type and the number of channels. With the "hw" interface, you have to check whether your hardware supports the configuration you would like to use. Otherwise, user can use the default interface for playback by:

```
/* Name of the PCM device, like plughw:0,0 */
  /* The first number is the number of the soundcard, the second number
 is the number of the device. */

  static char *device = "default"; /* playback device */
```

Now we can open the PCM device:

```
/* Open PCM. The last parameter of this function is the mode. */
  if ((err = snd_pcm_open (&playback_handle, device, stream, 0))
< 0) {
    fprintf (stderr, "cannot open audio device (%s)\n", snd_strerror
```

```
(err));
    exit (1);
  }
```

Setting the parameters of the device

Now we initialize the variables and allocate the `hwparams` structure:

```
/* Allocate the snd_pcm_hw_params_t structure on the stack. */
  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameters (%s)\n",
snd_strerror (err));
    exit (1);
  }
```

Before we can write PCM data to the soundcard, we have to specify access type, sample format, sample rate, number of channels, number of periods and period size. First, we initialize the hwparams structure with the full configuration space of the soundcard:

```
/* Init hwparams with full configuration space */
  if ((err = snd_pcm_hw_params_any (playback_handle, hw_params)) <
0) {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n", snd_strerror (err));
    exit (1);
  }
```

Now configure the desired parameters. For this example, we assume that the soundcard can be configured for stereo playback of 16 Bit Little Endian data, sampled at 44100 Hz. Therefore, we restrict the configuration space to match this configuration only. The access type specifies the way in which multi-channel data is stored in the buffer. For INTERLEAVED access, each frame in the buffer contains the consecutive sample data for the channels. For 16 Bit stereo data, this means that the buffer contains alternating words of sample data for the left and right channel.

```
/* Set access type. */
  if ((err = snd_pcm_hw_params_set_access (playback_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n", snd_strerror
(err));
    exit (1);
  }

  /* Set sample format */
  if ((err = snd_pcm_hw_params_set_format (playback_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format (%s)\n", snd_strerror
(err));
    exit (1);
  }

  /* Set sample rate. If the exact rate is not supported by the
hardware, use nearest possible rate. */
```

```
  exact_rate = rate;
  if ((err = snd_pcm_hw_params_set_rate_near (playback_handle,
hw_params, &exact_rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate (%s)\n", snd_strerror
(err));
    exit (1);
  }

  if (rate != exact_rate) {
    fprintf(stderr, "The rate %d Hz is not supported by your
hardware.\n ==> Using %d Hz instead.\n", rate, exact_rate);
  }

  /* Set number of channels */
  if ((err = snd_pcm_hw_params_set_channels (playback_handle,
hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count (%s)\n", snd_strerror
(err));
    exit (1);
  }
```

Now we apply the configuration to the PCM device pointed to by `pcm_handle` and prepare the PCM device.

```
  /* Apply HW parameter settings to PCM device and prepare device. */
  if ((err = snd_pcm_hw_params (playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n", snd_strerror
(err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  if ((err = snd_pcm_prepare (playback_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
snd_strerror (err));
    exit (1);
  }
```

Writing data to the device

After the PCM device is configured, we can start writing PCM data to it. The first write access will start the PCM playback. For interleaved write access, we use the function:

```
/* Write some junk data to produce sound. */
  if ((err = snd_pcm_writei (playback_handle, buf, BUFF_SIZE/2)) !=
BUFF_SIZE/2) {
    fprintf (stderr, "write to audio interface failed (%s)\n",
snd_strerror (err));
    exit (1);
  } else {
```

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

After the PCM playback is started, we have to make sure that our application sends enough data to the soundcard buffer. Otherwise, a buffer under-run will occur. After such an under-run has occurred, `snd_pcm_prepare` should be called.

Closing the device

After the data has been transferred, the device needs to be closed by calling:

```
  snd_pcm_close (playback_handle);

  exit (0);
}
```

A minimal record application

This program opens an audio interface for capture, configures it for stereo, 16 bit, 44.1kHz, interleaved conventional read/write access. Then its reads a chunk of random data from it, and exits. It isn't meant to be a real program.

Note that it is not possible to use one pcm handle for both playback and capture. So you have to configure two handles if you want to access the PCM device in both directions.

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

#define BUFF_SIZE 4096

int main (int argc, char *argv[])
{
  int err;
  short buf[BUFF_SIZE];
  int rate = 44100; /* Sample rate */
  int exact_rate; /* Sample rate returned by */
  snd_pcm_t *capture_handle;

  /* This structure contains information about the hardware and can be
used to specify the configuration to be used for */
  /* the PCM stream. */
  snd_pcm_hw_params_t *hw_params;

  /* Name of the PCM device, like hw:0,0 */
  /* The first number is the number of the soundcard, the second number
 is the number of the device. */
  static char *device = "default"; /* capture device */

  /* Open PCM. The last parameter of this function is the mode. */
  if ((err = snd_pcm_open (&capture_handle, device,
SND_PCM_STREAM_CAPTURE, 0)) < 0) {
```

```
    fprintf (stderr, "cannot open audio device (%s)\n", snd_strerror
(err));
    exit (1);
  }

  memset(buf,0,BUFF_SIZE);

  /* Allocate the snd_pcm_hw_params_t structure on the stack. */
  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameter structure
(%s)\n", snd_strerror (err));
    exit (1);
  }

  /* Init hwparams with full configuration space */
  if ((err = snd_pcm_hw_params_any (capture_handle, hw_params)) < 0)
 {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n", snd_strerror (err));
    exit (1);
  }

  /* Set access type. */
  if ((err = snd_pcm_hw_params_set_access (capture_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n", snd_strerror
(err));
    exit (1);
  }

  /* Set sample format */
  if ((err = snd_pcm_hw_params_set_format (capture_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format (%s)\n", snd_strerror
(err));
    exit (1);
  }

  /* Set sample rate. If the exact rate is not supported by the
hardware, use nearest possible rate. */
  exact_rate = rate;
  if ((err = snd_pcm_hw_params_set_rate_near (capture_handle,
hw_params, &exact_rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate (%s)\n", snd_strerror
(err));
    exit (1);
  }
```

```
  if (rate != exact_rate) {
    fprintf(stderr, "The rate %d Hz is not supported by your
hardware.\n ==> Using %d Hz instead.\n", rate, exact_rate);
  }

  /* Set number of channels */
  if ((err = snd_pcm_hw_params_set_channels(capture_handle, hw_params,
2)) < 0) {
    fprintf (stderr, "cannot set channel count (%s)\n", snd_strerror
(err));
    exit (1);
  }

  /* Apply HW parameter settings to PCM device and prepare device. */
  if ((err = snd_pcm_hw_params (capture_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n", snd_strerror
(err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  if ((err = snd_pcm_prepare (capture_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
snd_strerror (err));
    exit (1);
  }

  /* Read data into the buffer. */
  if ((err = snd_pcm_readi (capture_handle, buf, 128)) != 128) {
    fprintf (stderr, "read from audio interface failed (%s)\n",
snd_strerror (err));
    exit (1);
  } else {
    printf ("snd_pcm_readi successful\n");
  }

  snd_pcm_close (capture_handle);

  exit (0);
}
```

ALSA UserSpace Library (Useful Commands)

For each utility the `--help` option provides a comprehensive list of command-line options for arecord, aplay and amixer. Some of the more useful ones are listed below :

aplay

- `aplay <<file_name>> &` : plays a wave file with the sample-rate/pcm-word size info available in the wav header.
- `aplay -l` : prints information on all the audio cards available
- `aplay <<file_name>> -N &`: playback in non-blocking mode
- `aplay <<file_name>> -M &`: playback in memory-mapped (m-mapped) mode
- `aplay --buffer_size=<<buffer_size>> <<file_name>> &`: playback using different buffer-sizes
- `aplay -r <<sample_rate>> -c <<no_of_channels>> <<file_name>> -f <<sample_format>>&` : play using different sample rates and no. of channels or sample formats

Recognized sample-formats are: S8 U8 S16_LE S16_BE U16_LE U16_BE S24_LE S24_BE U24_LE U24_BE S32_LE S32_BE U32_LE U32_BE FLOAT_LE FLOAT_BE FLOAT64_LE FLOAT64_BE IEC958_SUBFRAME_LE IEC958_SUBFRAME_BE MU_LAW A_LAW IMA_ADPCM MPEG GSM SPECIAL S24_3LE S24_3BE U24_3LE U24_3BE S20_3LE S20_3BE U20_3LE U20_3BE S18_3LE S18_3BE U18_3LE

arecord

- `arecord -f cd <<file_name>> &`: record in the "CD" (i,e. 16 bit LE, 44100 hz, stereo) format
- `arecord -l`:prints information on all the audio cards available
- `arecord -r <<sample_rate>> -c <<no_of_channels>> -f <<sample_format>> <<file_name>> &`: record in the specified sample-rate and no. of channels or in the sample format indicated.

The sample-formats are as indicated for aplay.

Some recognized shortcuts are:

- `-f cd` (16 bit little endian, 44100, stereo)
- `-f cdr` (16 bit big endian, 44100, stereo)
- `-f dat` (16 bit little endian, 48000, stereo)
- `arecord <<file_name>> -N &`: record in non-blocking mode
- `arecord <<file_name>> -M &`:record in memory-mapped (m-mapped) mode
- `arecord --buffer_size=<<buffer_size>> <<file_name>> &`: record using different buffer-sizes

Loopback command -

`arecord -f cd | aplay &`: record in "CD" format (see below for details) and play the recorded data i,e. software loopback

amixer

- `amixer controls`:show all controls for given card
- `amixer contents`:show contents of all controls for given card

References

[1] http://www.alsa-project.org/main/index.php/ASoC
[2] http://www.alsa-project.org/main/index.php/Main_Page
[3] http://www.alsa-project.org/alsa-doc/alsa-lib/
[4] http://www.equalarea.com/paul/alsa-audio.html
[5] http://linux.die.net/man/1/alsamixer
[6] http://focus.ti.com/docs/prod/folders/print/tps65950.html
[7] http://focus.ti.com/lit/ds/symlink/
[8] http://www.alsa-project.org/main/index.php/Download

UserGuideDisplayDrivers PSP 04.02.00.07

Display Driver

Introduction

Display Sub-System hardware integrates one graphics pipeline, two video pipelines, and two overlay managers (one for digital and one for analog interface). Digital interface is used for LCD and DVI output and analog interface is used for TV out.

The primary functionality of the display driver is to provide interfaces to user level applications and management of Display Sub-System hardware.

This section defines and describes the usage of user level interfaces of Video Display Driver.

NOTE: Please note that the AM3517 and AM/DM37x Display Sub-System module is same as OMAP3x, so terms have been used inter-changeably and referred as OMAP3x in this document.

References

1. Video for Linux Two Home Page [http://linux.bytesex.org/v4l2/ [1]]
2. Video for Linux Two API Specification [[http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf [2]]

Acronyms & Definitions

Display Driver: Acronyms

Acronym	Definition
V4L2	Video for Linux Two
DSS	Display SubSystem
NTSC	National Television System Committee
PAL	Phase Alternating Line
LCD	Liquid Crystal Display
DVI	Digital Visual Interface

Hardware Overview

The display subsystem provides the logic to display a video frame from the memory frame buffer (either SDRAM or SRAM) on a liquid-crystal display (LCD) panel or a TV set. The display subsystem integrates the following elements

- Display controller (DISPC) module
- Remote frame buffer interface (RFBI) module
- Serial display interface (SDI) complex input/output (I/O) module with the associated phased-locked loop (PLL)
- Display serial interface (DSI) complex I/O module and a DSI protocol engine
- DSI PLL controller that drives a DSI PLL and high-speed (HS) divider
- NTSC/PAL video encoder

Features

The Display driver supports the following features:

- Supports LCD display interface at VGA resolution (480*640)
- Supports TV display interface at NTSC/PAL resolutions (both S-Video out and Composite out is supported)
- Supports DVI digital interface (mode selection via boot argument).
- Supports Graphics pipeline and two video pipelines. Graphics pipeline is supported through fbdev and video pipelines through V4L2.
- Supported color formats: On OSD (Graphics pipeline): RGB565, RGB888, ARGB and RGBA. On Video pipelines: YUV422 interleaved, RGB565, RGB888.
- Configuration of parameters such as height and width of display screen, bits-per-pixel etc.
- Supports setting up of OSD and Video pipeline destinations (TV or LCD) through syfs interface.
- Supports buffer management through memory mapped and user pointer buffer exchange for application usage (mmaped).
- Supports rotation - 0, 90, 180 and 270 degrees on LCD and TV output
- Supports destination and source colorkeying on Video pipelines through V4L2.
- Supports alpha blending through ARGB pixel format on Video2 pipeline and RGBA and ARGB format on graphics pipeline and global alpha blending

Architecture

This chapter describes the Driver Architecture and Design concepts

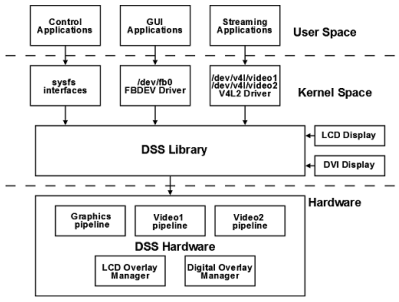
Driver Architecture

OMAP3x display hardware integrates one graphics pipeline, two video pipelines, and two overlay managers (one for digital and one for analog interface). Digital interface is used for LCD and DVI output and analog interface is used for TV out.

The primary functionality of the display driver is to provide interfaces to user level applications and management to OMAP3x display hardware.

This includes, but is not limited to:

- GUI rendering through the graphics pipeline.
- Static image or video rendering through two video pipelines.
- Connecting each of three pipelines to either LCD or TV output so the display layer is presented on the selected output path.
- Image processing (cropping, rotation, mirroring, color conversion, resizing, and etc).



OMAP3x Display Subsystem Architecture

Software Design Interfaces

Above figure (OMAP3x Display Subsystem Architecture) shows the major components that makes up the DSS software sub-system

- **Display Library**

This is a HAL/functional layer controlling the bulk of DSS hardware. It exposes the number of APIs controlling the overlay managers, clock, and pipelines to the user interface drivers like V4L2 and FBDEV. It also exposes the functions for registering and de-registering of the various display devices like LCD and DVI to the DSS overlay managers.

- **SYSFS interfaces**

The SYSFS interfaces are mostly used as the control path for configuring the DSS parameters which are common between FBDEV and V4L2 like the alpha blending, color keying, etc. It is also used for switching the output of the pipeline to either LCD or Digital overlay manager. In future sysfs entries might also be used to switch the modes like NTSC, PAL on TV and 480P, 720P on DVI outputs.

- **Frame Buffer Driver**

This driver is registered with the FBDEV subsystem, and is responsible for managing the graphics layer frame buffer. Driver creates /dev/fb0 as the device node. Application can open this device node to open the driver and negotiate parameters with the driver through frame buffer ioctls. Application maps driver allocated buffers in the application memory space and fills them for the driver to display.

- **Video Applications & V4L2 subsystem**

Video applications (camera, camcorder, image viewer, etc.) use the standard V4L2 APIs to render static images and video to the video layers, or capture/preview camera images.
This driver is responsible for managing the video layers frame-buffers. It is a V4L2 compliant driver with some additions to implement special software requirements that target OMAP3x hardware features . This driver conforms to the Linux driver model. For using the driver, application should create the device nodes /dev/video1 and /dev/video2 device nodes for two video layers. Application can open the driver by opening these device nodes and negotiate the parameters by V4L2 ioctls. Initially application can request the driver to allocate number of buffers and MMAPs these buffers. Then the application can fill up these buffers and pass them to driver for display by using the standard V4L2 streaming ioctls.

Usage

Opening and Closing of Driver

The device can be opened using open call from the application, with the device name and mode of operation as parameters. Application can open the driver only in blocking mode. Non-blocking mode of open is not supported.

- **V4L2 Driver**

The driver will expose two software channels (/dev/video1 and /dev/video2), one for each video pipeline. Both of these channels supports only blocking mode of operations. These channels can only be opened once.

```
/* Open a video Display logical channel in blocking mode */
fd = open ("/dev/video1", O_RDWR);
if (fd == -1) {
    perror("Failed to open display device\n");
    return -1;
}
/* Closing of channel */
close (fd);
```

- **FBDEV Driver**

The driver will expose one software channels (/dev/fb0) for the graphics pipeline. The driver cannot be opened multiple times. Driver can be opened only once.

```
/* Open a graphics Display logical channel in blocking mode */
fd = open ("/dev/fb0", O_RDWR);
if (fd == -1) {
    perror("failed to open display device\n");
    return -1;
}
/* Closing of channels */
close (fd);
```

Command Line arguments

V4L2 Driver

V4L2 driver supports set of command line argument for, default number of buffers, their buffer size, enable/disable VRFB buffer allocation and debug option for both the video pipelines.

V4L2 driver uses the VRFB buffers for rotation. Because of the limitation of the VRFB engine these buffers are quite big in size. Please refer to the Buffer Management section for required and allocated size of the VRFB buffers. VRFB buffers are allocated by driver during vidioc_reqbufs ioctl if the rotation is enabled and freed during vidioc_streamoff. But under heavy system load, memory fragmentation may occur and VFRB buffer allocation may fail. To address this issue V4L2 driver provides command line argument to allocate the VRFB buffers at driver init time and buffers will be freed when driver is unloaded.

Below is the list of arguments which V4L2 driver supports -

V4L2 Driver Command Line Arguments

Argument	Description
video1_numbuffers	Number of buffers to be allocated at init time for Video1 device
video2_numbuffers	Number of buffers to be allocated at init time for Video2 device
video1_bufsize	Size of the buffer to be allocated for video1 device
video2_bufsize	Size of the buffer to be allocated for video2 device
vid1_static_vrfb_alloc	Static allocation of the VRFB buffer for video1 device
vid2_static_vrfb_alloc	Static allocation of the VRFB buffer for video2 device
debug	Enable debug messaging

For dynamic build of the driver, these argument are specified at the time of inserting the driver. For static build of the driver, these argument can be specified along with boot time arguments. Following example shows how to specify command line argument for static and dynamic build.

Insert the dynamically built module with following parameters:

```
# insmod omap_vout.ko video1_numbuffers=3 video2_numbuffers=3
video1_bufsize=644000 video2_bufsize=644000
vid1_static_vrfb_alloc=y vid2_static_vrfb_alloc=y
```

Below is the sample example of bootargs for statically compiled driver from bootloader:

```
# setenv bootargs 'console=ttyS0,115200n8 mem=128M root=/dev/nfs
noinitrd nfsroot=nfs-server/home,nolock ip=dhcp
omap_vout.video1_numbuffers=3 omap_vout.video2_numbuffers=3
omap_vout.video1_bufsize=64400 omap_vout.video2_bufsize=64400
omap_vout.vid1_static_vrfb_alloc=y omap_vout.vid2_static_vrfb_alloc=y'
```

- **NOTE:** The entire command should be entered in a single line.

FBDEV Driver

FBDEV driver supports set of command line argument for enabling/setting rotation angle, enable/disable VRFB rotation, default mode, size of vram and debug option. These command line arguments can only be used with boot time arguments as FBDEV driver only supports static build.

Below is the list of arguments which Fbdev driver supports -

Fbdev Driver Command Line Arguments

Argument	Description
mode	Default video mode for specified displays
vram	VRAM allocated memory for a framebuffer, user can individually configure VRAM buffers for each plane/device node
vrfb	Use VRFB rotation for framebuffer
rotate	Default rotation applied to framebuffer
test	Draw test pattern to framebuffer whenever framebuffer settings change
debug	Enable debug messaging

Following example shows how to specify 90 degree rotation in boot time argument:

```
# setenv bootargs console=ttyS0,115200n8 mem=128M noinitrd
root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp omapfb.rotate=1
omapfb.vrfb=y
```

Following example shows how to specify size of framebuffer in boot time argument:

```
setenv bootargs console=ttyS0,115200n8 mem=128M noinitrd root=/dev/nfs
nfsroot=nfs-server/home,nolock ip=dhcp vram=20M omapfb.vram=0:20M
```

Following example shows how to specify mode for framebuffer in boot time argument:

```
# setenv bootargs console=ttyS0,115200n8 mem=128M noinitrd
root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp
omapfb.mode=dvi:720x480@60
```

- **NOTE:** The entire command should be entered in a single line.

DSS Library

There are few arguments which allows control over core DSS functionality.

DSS Library Command Line Arguments

Argument	Description
def_disp	Name of default display, to which all overlays will be connected
debug	Enable debug messages

Following example shows how to specify default display to particular output in boot time argument:

```
# setenv bootargs console=ttyS0,115200n8 mem=128M noinitrd
root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp
omapdss.def_disp="dvi" omapdss.debug=y
```

- **NOTE:** The entire command should be entered in a single line.

Buffer Management

Memory requirement for V4L2 and FBDEV driver

Driver	Without Rotation	With Rotation
Fbdev Driver	A single buffer of size 480*640*2 bytes, can be changed through bootargs	A single buffer of size 2048*640*2 bytes, can be changed through bootargs
V4L2 Driver	Single buffer of 1280*720*4 bytes and Number of buffers is configurable using VIDIOC_REQBUFS ioctl and command line argument	Same requirement as without rotation. Additionally allocates one buffer of size 3686400 bytes for each context. Number of context are same as the number of buffers allocated using REQBUFS, which is not more than four.

- **V4L2 Driver**

Memory Mapped buffer mode and User pointer buffer mode are the two memory allocation modes supported by driver. In Memory map buffer mode, application can request memory from the driver by calling VIDIOC_REQBUFS ioctl. In this mode, maximum number of buffers is limited to VIDEO_MAX_FRAME (defined in driver header files) and is limited by the available memory in the kernel. If driver is not able to allocate the requested number of buffer, it will return the number of buffer it is able to allocate.

The main steps that the application must perform for buffer allocation are:

1. **Allocating Memory**

Ioctl: VIDIOC_REQBUFS

This is a necessary ioctl for streaming IO. It has to be called for both drivers buffer mode and user buffer mode. Using this ioctl, driver will identify whether driver buffer mode or user buffer mode will be used.

It takes a pointer to instance of the `v4l2_requestbuffers` structure as an argument.

User can specify the buffer type (V4L2_BUF_TYPE_VIDEO_OUTPUT), number of buffers, and memory type (V4L2_MEMORY_MMAP, V4L2_MEMORY_USERPTR)at the time of buffer allocation. In case of driver buffer mode, this ioctl also returns the actual number of buffers allocated in count member of v4l2_requestbuffer structure.

It can be called with zero number of buffers to free up all the buffers already allocated. It also frees allocated buffers when application changes buffer exchange mechanism. Driver always allocates buffers of maximum image size supported. If application wants to change buffer size, it can be done through video1_buffsize and video2_buffsize command line arguments.

When rotation is enabled, driver also allocates buffer for the VRFB virtual memory space along with the mmap or user buffer. It allocates same number of buffers as the mmap or user buffers. Maximum number of buffers, which can be allocated, is 4 when rotation is enabled.

```
/* structure to store buffer request parameters */
struct v4l2_requestbuffers reqbuf;
reqbuf.count = numbuffers;
reqbuf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
reqbuf.memory = V4L2_MEMORY_MMAP;
ret = ioctl(fd , VIDIOC_REQBUFS, &reqbuf);
if(ret < 0) {
    printf("cannot allocate memory\n");
    close(fd);
    return -1;
}
```

1. **Getting physical address**

Ioctl: VIDIOC_QUERYBUF

This ioctl is used to query buffer information like buffer size and buffer physical address. This physical address is used in m-mapping the buffers. This ioctl is necessary for driver buffer mode as it provides the physical address of buffers, which are used to mmap system call the buffers.

It takes a pointer to instance of v4l2_buffer structure as an argument. User has to specify the buffer type (V4L2_BUF_TYPE_VIDEO_OUTPUT), buffer index, and memory type (V4L2_MEMORY_MMAP)at the time of querying.

```
/* allocate buffer by VIDIOC_REQBUFS */
/* structure to query the physical address
of allocated buffer */
struct v4l2_buffer buffer;
/* buffer index for querying -0 */
buffer.index = 0;
buffer.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
buffer.memory = V4L2_MEMORY_MMAP;
if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < 0) {
    printf("buffer query error.\n");
    close(fd);
    exit(-1);
}
/*The buffer.m.offset will contain the physical address returned from
driver*/
```

1. **Mapping Kernel space address to user space**

Mapping the kernel buffer to the user space can be done via mmap. User can pass buffer size and physical address of buffer for getting the user space address

```
/* allocate buffer by VIDIOC_REQBUFS */
/* query the buffer using VIDIOC_QUERYBUF */
/* addr hold the user space address */
unsigned int addr;
addr = mmap(NULL, buffer.size, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
buffer.m.offset);
/* buffer.m.offset is same as returned from VIDIOC_QUERYBUF */
```

- **FBDEV Driver**

FBDEV driver supports only memory mapped buffers. Driver allocates one physically contiguous buffers, which can support 480X640 resolution for 16 bpp format.

Following steps are required to map buffers in application memory space.

1. **Getting fix screen information**

`FBIOGET_FSCREENINFO` ioctl is used to get the not-changing screen information like physical address of the buffer, size of the buffer, line length.

```
/* Getting fix screen information */
struct fb_fix_screeninfo fix;
ret = ioctl(fd, FBIOGET_FSCREENINFO, &fix);
```

```
if(ret < 0) {
    printf("Cannot get fix screen information\n");
    close(fd);
    exit(0);
}
printf("Line length = %d\n",fix.line_length);
printf("Physical Address = %x\n",fix.smem_start);
printf("Buffer Length = %d\n",fix.smem_len);
```

1. **Getting Variable screen information**

FBIOGET_VSCREENINFO ioctl is used to get the variable screen information like resolution, bits per pixel etc...

```
/* Getting fix screen information */
struct fb_var_screeninfo var;
ret = ioctl(fd, FBIOGET_VSCREENINFO, &var);
if(ret < 0) {
    printf("Cannot get variable screen information\n");
    close(fd);
    exit(0);
}
printf("Resolution = %dx%d\n",var.xred, var.yres);
printf("bites per pixel = %d\n",var.bpp);
```

1. **Mapping Kernel space address to user space**

Mapping the kernel buffer to the user space can be done via mmap system call.

```
/* addr hold the user space address */
unsigned int addr, buffersize;
/* Get the fix screen info */
/* Get the variable screen information */
buffersize = fix.line_length * var.yres;
addr = mmap(NULL, buffersize, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
0);
/* buffer.m.offset is same as returned from VIDIOC_QUERYBUF */
```

Rotation

Rotation is implemented with use of Rotation Engine module in Virtual Rotation Frame Buffer module in OMAP3x. Rotation engine supports rotation of an image with degree 0, 90, 180 and 270. There are 12 contexts available for rotating an image and there are four virtual memory space associated with each context. To rotate an image, image is written to 0 degree virtual memory for a context and rotated image can read back from the virtual memory for that angle of the same context.

For using Rotation Engine, User has to allocate physical memory and provide address of the memory to the rotation engine. The buffer size for this physical buffer should be large enough to store the image to be rotated. When program writes to the virtual address of the context, rotation engine write to this memory space and when program reads image from virtual address, rotation engine reads image from this buffer with rotation angle.

- **V4L2 Driver**

V4L2 driver supports rotation by using rotation engine in the VRFB module. Driver allocates physical buffers, required for the rotation engine, when application calls VIDIOC_REQBUFS ioctl. Therefore, when this ioctl is

called driver allocates buffers for storing image and allocates buffers for the rotation engine. It also programs VRFB rotation engine when this ioctl is called. At the time of enqueing memory mapped buffer, driver copies entire image from mmaped buffer to buffer for the rotation engine using DMA. DSS is programmed to take image from VRFB memory space when rotation is enabled. So DSS always gets rotated image. Maximum four buffers can be allocated using REQBUFS ioctl when rotation is enabled.

Driver provides ioctl interface for enabling/disabling and changing the rotation angle. These ioctls are VIDIOC_S_CTRL/VIDIOC_G_CTRL as drive allocates buffer for VRFB during REQBUFS ioctl, application has to enable/set the rotation angle before calling REQBUFS ioctl. After enabling rotation, application can change the rotation angle. Rotation angle cannot be changed while streaming is on.

Following code shows how to set rotation angle to 90 degree:

```
struct v4l2_control control;
int degree = 90;
control.id = V4L2_CID_ROTATE;
control.value = degree;
ret = ioctl(fd, VIDIOC_S_CTRL, &control);
if (ret < 0) {
    perror("VIDIOC_S_CTRL\n");
    close(fd);
    exit(0);
}
/* Rotation angle is now set to 90 degree. Application can now do
streaming to see rotated image*/
```

NOTE: Rotation value must be set using VIDIOC_S_CTRL before setting any format using VIDIOC_S_FMT as VIDIOC_S_FMT uses rotation value for calculating buffer formats. Also VIDIOC_S_FMT ioctl must be called after changing the rotation angle to change parameters as per the new rotation angle.

- **FBDEV Driver**

FBDEV driver supports rotation by using rotation engine in the VRFB module. For using this feature of the driver, rotation has to be enabled. Application can enable rotation by enabling/setting rotation angle in boot time argument of the kernel for FBDEV driver. Applications can thus use the FBIOPUT_VSCREENINFO ioctl to set the rotation angle. Applications have to set the 'rotate' field in the fb_var_screeninfo structure equal to the angle of rotation (0, 90, 180 or 270) and call this ioctl. Frame buffer driver also supports the rotation through sysfs entry. Any one of the two method can be used to configure rotation.

Constraint: While doing rotation x-resolution virtual should be equal to x-resolution. y-resolution virtual should be greater than or equal to yresolution. Please note that VRFB rotation engine requires alignment of 32 bytes in horizontal size and 32 lines in vertical size. So while doing rotation x-resolution should be 32 byte aligned and y resolution and y resolution virtual should be 32 lines aligned. For example for 360X360 required resolution with 16bpp no of bytes per line comes to 360*2=720. Which is not 32 byte aligned. While no of lines comes to 360 which is also ot 32 lines aligned. So actual resolution should be set to 368X368. But if same resolution is required for 32bpp then no of bytes per line comes to 360*4 that is 1440. Which is 32 byte aligned so actual resolution should be set to 360X368. Also the maximum y-res virtual possible is 2048 because of VRFB limitation when rotation enabled.

var.rotate variable should not be modified when rotation is not selected through command line arguments else behaviour is unexpected

NOTE: By default frame buffer driver allocates the buffer for single VGA (480x640) frame considering 0 degree rotation. Please refer to the section Command line arguments

Following code listings demos how to set the rotation in frame buffer driver using ioctl and sysfs entry:

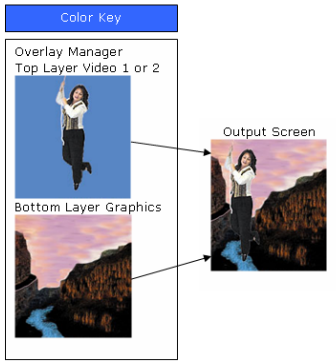
```
struct fb_var_screeninfo var;
/* Set the rotation through ioctl. */
/* Get the Variable screen info through "FBIOGET_VSCREENINFO" */
var.rotate = 1; /* To set rotation angle to 90 degree */
if (ioctl(fd, FBIOPUT_VSCREENINFO, &var)<0) {
    perror("Error:FBIOPUT_VSCREENINFO\n");
    close(fd);
    exit(4);
}
```

Setting the rotation through sysfs where 0 - 0 degree, 1 - 90 degree, 2 - 180 degree and 3 - 270 degree respectively:

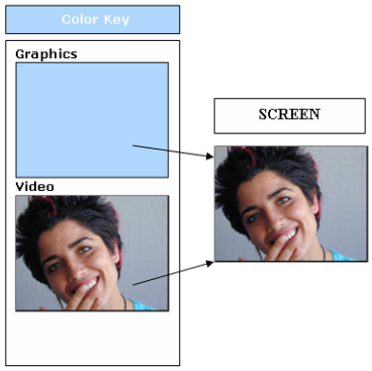
```
# echo 1 > /sys/class/graphics/fb0/rotate
```

Color Keying

There are two types of transparent color keys: Video source transparency and graphics destination transparency key. The encoded pixel color value is compared to the transparency color key. For CLUT bitmaps, the palette index is compared to the transparency color key and not to the palette value pointed out by the palette index.



Video Source Color keying



Video Destination color keying

Constraint: The video source transparency color key and graphics destination transparency color key cannot be active at the same time.

User can Color Keying either through V4L2 Driver IOCTL interface or SYSFS interface.

Video source transparency color key value allows defining a color that the matching pixels with that color in the video pipelines are replaced by the pixels in graphics pipeline. It is limited to RGB formats only and non-scaling cases.

The Graphics destination color key allows defining a color that the nonmatching pixels in the graphics pipelines prevent video overlay. The destination transparency color key is applicable only in the graphics region when graphics and video overlap. Otherwise, the destination transparency color key is ignored.

Both the keying mechanism cannot be used simultaneously. All color key related IOCTLs are not pipeline oriented. Following example shows how to use source and destination color key using both the interfaces -

- **Using V4L2 IOCTL**

Enable Source Color Keying:

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_G_FBUF");
    close(fd);
    exit(1);
}
/* Set SRC_COLOR_KEYING if device supports that */
if(framebuffer.capability & V4L2_FBUF_CAP_SRC_CHROMAKEY) {
    framebuffer.flags |= V4L2_FBUF_FLAG_SRC_CHROMAKEY;
    framebuffer.flags &= ~V4L2_FBUF_FLAG_CHROMAKEY;
    ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
    if (ret < 0) {
        perror ("VIDIOC_S_FBUF");
        close(fd);
        exit(1);
    }
}
```

Disabling the Source Color Keying:

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_G_FBUF");
    close(fd);
    exit(1);
}
if(framebuffer.capability & V4L2_FBUF_CAP_SRC_CHROMAKEY) {
    framebuffer.flags &= ~V4L2_FBUF_FLAG_SRC_CHROMAKEY;
    ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
    if (ret < 0) {
        perror ("VIDIOC_S_FBUF");
        close(fd);
        exit(1);
    }
}
```

Enableing destination color keying:

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_G_FBUF");
    close(fd);
    exit(1);
}
/* Set SRC_COLOR_KEYING if device supports that */
if(framebuffer.capability & V4L2_FBUF_CAP_CHROMAKEY) {
    framebuffer.flags |= V4L2_FBUF_FLAG_CHROMAKEY;
    framebuffer.flags &= ~V4L2_FBUF_FLAG_SRC_CHROMAKEY;
    ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
    if (ret < 0) {
        perror ("VIDIOC_S_FBUF");
        close(fd);
        exit(1);
    }
}
```

Disabling destination color keying:

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_G_FBUF");
    close(fd);
    exit(1);
}
if(framebuffer.capability & V4L2_FBUF_CAP_CHROMAKEY) {
    framebuffer.flags &= ~V4L2_FBUF_FLAG_CHROMAKEY;
    ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
    if (ret < 0) {
        perror ("VIDIOC_S_FBUF");
        close(fd);
        exit(1);
    }
}
```

Below program listing shows how to set the chromakey value. Please note that chroma key value should be set before enabling the chroma keying. Overlay manager should not be changed between the setting up of chroma key and enabling the chroma keying.

```
struct v4l2_format fmt;
u8 chromakey = 0xF800; /* Red color RGB565 format */
fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
```

```
    close(fd);
    exit(0);
}
fmt.fmt.win.chromakey = chromakey;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
    close(fd);
    exit(0);
}
```

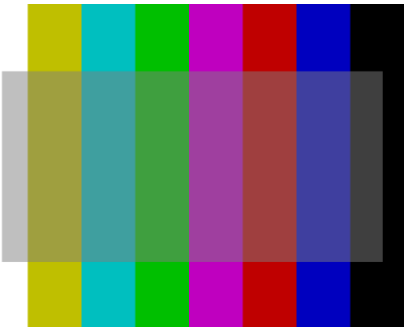
The code snippet below illustrates how to get the chromakey value.

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
    close(fd);
    exit(0);
}
printf("Croma value read is %d\n", fmt.fmt.win.chromakey);
```

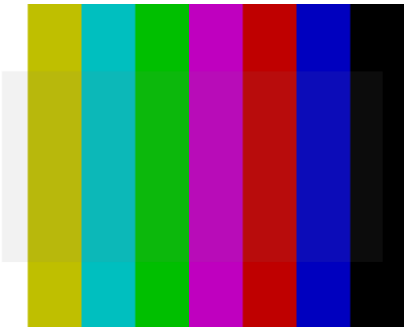
- **SYSFS Interface**

Alpha Blending

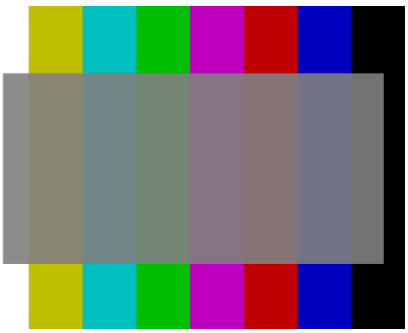
Alpha blending is a process of blending a foreground color with a background color and producing a new blended color. New blended color depends on the transparency factor referred to as alpha factor of the foreground color. If the alpha factor is 100% then blended image will have only foreground color. If the alpha factor is 0% blended image will have only back ground color. Any value between 0 to 100% will blend the foreground and background color to produce new blended color depending upon the alpha factor.



Alpha blending with almost 50% transparency



Alpha blending with almost 100% transparency



Alpha blending with almost 0% transparency

Overlay manager of DSS is capable of supporting the alpha blending. This is done by displaying more than one layer (video and graphics) to the same output device, TV or LCD. Overlay manager supports normal mode and alpha mode of operation.

In normal mode graphics plane is at bottom on top of it is video1 and video2 is on top of video1. While in alpha mode video1 plane is at bottom, video2 is on top of video1, and graphics plane is above video2. Alpha mode is selectable on any of the output device TV or LCD.

Video2 and graphics layer of the DSS is capable of supporting alpha blending. Two types of alpha blending is supported global and pixel alpha blending. ARGB and RGBA formats of the video2 and graphics pipeline supports pixel based alpha blending. In which A represent the alpha value for each pixel. Thus, each pixel can have different alpha value. While global alpha is the constant alpha factor for the pipeline for all the pixels. Both can be used in conjunction.

Both V4L2 and Frame buffer driver supports alpha blending based on pixel format for video2 and graphics pipeline respectively. Global alpha blending is also supported through V4L2 and SYSFS interface. Before using any of the alpha blending methods alpha blending needs to be enabled on the selected output device either through V4L2 ioctl or SYSFS interface. Alpha blending will be enabled on the output device to which video pipeline is connected

Following program listing will enable alpha blending through V4L2 driver ioctl -

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_S_FBUF");
    close(fd);
    return 0;
}
framebuffer.flags |= V4L2_FBUF_FLAG_LOCAL_ALPHA;
```

```
framebuffer.flags &= ~(V4L2_FBUF_FLAG_CHROMAKEY |
V4L2_FBUF_FLAG_SRC_CHROMAKEY);
ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_S_FBUF");
    close(fd);
    return 0;
}
```

Following program listing will disable alpha blending through V4L2 driver ioctl -

```
struct v4l2_framebuffer framebuffer;
ret = ioctl (fd, VIDIOC_G_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_S_FBUF");
    close(fd);
    return 0;
}
framebuffer.flags &= ~V4L2_FBUF_FLAG_LOCAL_ALPHA;
ret = ioctl (fd, VIDIOC_S_FBUF, &framebuffer);
if (ret < 0) {
    perror ("VIDIOC_S_FBUF");
    close(fd);
    return 0;
}
```

Following program listing will enable/disable alpha blending through SYSFS entry -

```
# echo 0/1 >
/sys/devices/platform/omapdss/manager<index>/alpha_blending_enabled

Where,
      0/1   => 0 - Disable, 1 - Enable.
      index => 0 - LCD Manager 1 - TV Manager.
```

- **Global Alpha belnding**

User can configure global alpha value either through V4L2 IOCTL (V4L2_BUF_TYPE_VIDEO_OVERLAY) or SYSFS interfac.

Below programlisting shows how to set the global alpha value for video2 pipeline.

```
struct v4l2_format fmt;
u8 global_alpha = 128;
fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
    close(fd);
    exit(0);
}
fmt.fmt.win.global_alpha = global_alpha;
```

```
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
    close(fd);
    exit(0);
}
```

- **Pixel Based Alpha belnding**

V4L2 Driver

V4l2 driver supports alpha blending through ARGB pixel format as well as global alpha value.

To set the pixel alpha value set ARGB format by setting format type to V4L2_PIX_FMT_RGB32. Call VIDIOC_S_FMT ioctl of the driver to set it to ARGB format.

```
struct v4l2_format fmt;
/* Set the video type*/
fmt.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
/* Set the width and height of the picture*/
fmt.fmt.pix.width = 400;
fmt.fmt.pix.height = 400;
/* Set the format to ARGB */
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB32;
/* Call set format Ioctl */
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    exit(0);
}
```

Note: RGBA format is not supported.

FBDEV Driver

Pixel alpha value is supported through 32 bpp. Setting the offsets correctly will set the pixel format as ARGB or RGBA.

Below program listing shows how to set ARGB pixel format.

```
struct fb_var_screeninfo var;
/* Get variable screen information. Variable screen information
* gives information like size of the image, bites per pixel,
* virtual size of the image etc. */
ret = ioctl(fd, FBIOGET_VSCREENINFO, &var);
if (ret < 0) {
    perror("Error reading variable information.\n");
    close(fd);
    exit(3);
}
/* Set bits per pixel and offsets*/
var.red.length= 8;
var.green.length = 8;
```

```
var.blue.length = 8;
var.transp.length= 8;
var.transp.offset = 24;
var.red.offset = 16;
var.green.offset =8;
var.blue.offset = 0;
var.bits_per_pixel = 32;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &var)<0) {
    perror("Error:FBIOPUT_VSCREENINFO\n");
    close(fd);
    exit(4);
}
```

User can set the global alpha value for graphics pipeline using sysfs entry, as shown below -

```
#echo <global alpha value> >
/sys/devices/platform/omapdss/overlay<index>/global_alpha

Where,
      index => 0 - GFX Overlay 1 - Vid1 Overlay 2 - Vid2 Overlay
```

NOTE: Before using the global alpha or pixel based alpha Alpha blending needs to be enabled using either sysfs or V4L2 ioctl interface.

Buffer Format

Buffer format describes the pixel format in the image. It also describes the memory organization of each color component within the pixel format. In all buffer formats, blue value is always stored in least significant bits, then green value and then red value.

V4L2 Driver

Video layer supports following buffer format: YUYV, UYVY, RGB565, RGB24 (packed and unpacked). The corresponding v4l2 defines for pixel format are V4L2_PIX_FMT_YUYV, V4L2_PIX_FMT_UYVY, V4L2_PIX_FMT_RGB565, V4L2_PIX_FMT_RGB24 (packed), V4L2_PIX_FMT_RGB32. (For video1 and video2 V4L2_PIX_FMT_RGB32 corresponds to RGB24 unpacked).

Buffer format can be changed using VIDIOC_S_FMT ioctl with type as V4L2_BUF_TYPE_VIDEO_OUTPUT and appropriate pixel format type.

Following example shows how to change pixel format to RGB565

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB565;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    exit(0);
}
```

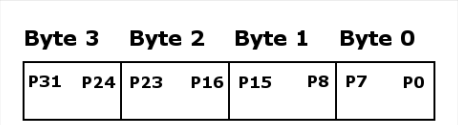
FBDEV Driver

Graphics layer supports following buffer format: RGB24(un-packed) ARGB, RGBA and RGB565. Buffer format can be changed in FBDEV driver by using bpp, red, green, and blue fields of fb_vscreeninfo structure and ioctl FBIOPUT_VSCREENINFO. Application needs to specify bits per pixel and length and offset of red, green and blue component. Bits-per-pixel and color depth in the pixel aren't quite the same thing. The display controller supports color depths of 1, 2, 4, 8, 12, 16, 24 and 32 bits. Color depth and bits-per-pixel are the same for depths of 1, 2, 4, 8, and 16 bits, but for a color depth of 12 bits the pixel data is padded to 16 bits-per-pixel, and for a color depth of 24 bits the pixel data is padded to 32 bits-per-pixel. So application has to specify bits per pixel 16 and 32 for the color depth 12 and 24. To specify exact color depth, red, green and blue member of the fb_varscreeninfo can be used.

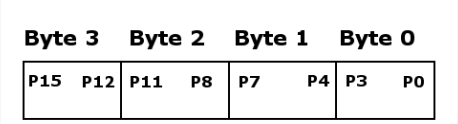
Following example shows how to set 12 and 24 bits per pixels.

```
struct fb_varscreeninfo var;
var.bpp = 16;
var.red.length = var.green.length = var.blue.length = 4;
var.red.offset = 8;
var.green.offset = 4;
var.blue.offset = 0;
ret = ioctl(fd, FBIOPUT_VSCREENINFO, &var);
if (ret < 0) {
    perror("FBIOPUT_VSCREENINFO\n");
    close(fd);
    exit(0);
}
```

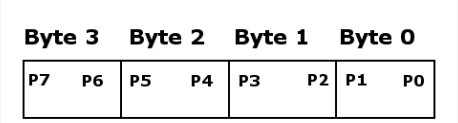
Buffer Formats



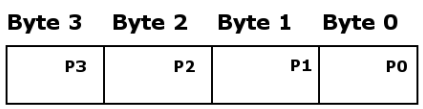
1-BPP Data Memory Organization



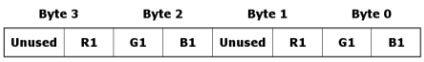
2-BPP Data Memory Organization



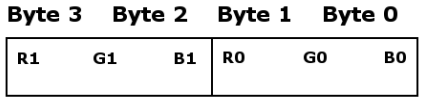
4-BPP Data Memory Organization



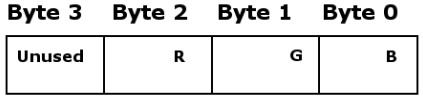
8-BPP Data Memory Organization



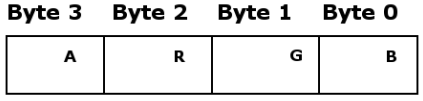
12-BPP Data Memory Organization



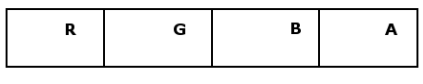
16-BPP Data Memory Organization



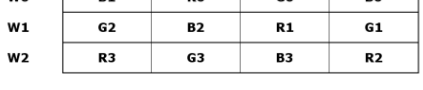
24-BPP Data Memory Organization



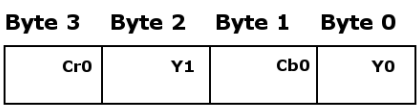
ARGB 32-BPP Data Memory Organization



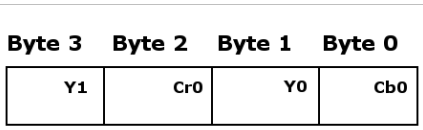
RGBA 32-BPP Data Memory Organization



24-BPP Packed Data Memory Organization



UYVY 4:2:2 Data Memory Organization



YUV2 4:2:2 Data Memory Organization

Display Window

The video pipelines can be connected to either an DVI output LCD output or a TV output either through boot time parameter or through SYSFS interface. Although the display Driver computes a default display window whenever the image size or cropping is changed, an application should position the display window via the VIDIOC_S_FMT I/O control with the V4L2_BUF_TYPE_VIDEO_OVERLAY buffer type. When a switch from LCD to TV or from TV to LCD happens, an application is expected to adjust the display window. V4L2 driver only supports change of display window.

Following example shows how to change display window size.

```
struct v4l2_format fmt;
Fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
fmt.fmt.win.w.left = 0;
fmt.fmt.win.w.top = 0;
fmt.fmt.win.w.width = 200;
fmt.fmt.win.w.height = 200;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    exit(0);
}
/* Display window size and position is changed now */
```

Cropping

The V4L2 Driver allows an application to define a rectangular portion of the image to be rendered via the VIDIOC_S_CROP Ioctl with the V4L2_BUF_TYPE_VIDEO_OUTPUT buffer type. When application calls VIDIOC_S_FMT ioctl, driver sets default cropping rectangle that is the largest rectangle no larger than the image size and display windows size. The default cropping rectangle is centered in the image. All cropping dimensions are rounded down to even numbers. Changing the size of the cropping rectangle will in general also result in a new default display window. As stated above, an application must adjust the display window accordingly.

Following example shows how to change crop size.

```
struct v4l2_crop crop;
crop.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
```

```
crop.c.left = 0;
crop.c.top = 0;
crop.c.width = 320;
crop.c.height = 320;
ret = ioctl(fd, VIDIOC_S_CROP, &crop);
if (ret < 0) {
    perror("VIDIOC_S_CROP\n");
    close(fd);
    exit(0);
}
/* Image cropping rectangle is now changed */
```

Scaling

Video pipe line contains scaling unit which is used when transferring pixels from the system memory to the LCD panel or the TV set. The scaling unit consists of two scaling blocks: The vertical scaling block followed by the horizontal scaling block. The two scaling units are independent: Neither of them, only one, or both can be used simultaneously.

As scaling unit is on video pipeline, scaling is only supported in V4L2 driver. Scaling is not explicitly exposed at the API level. Instead, the horizontal and vertical scaling factors are based on the display window and the image cropping rectangle. The horizontal scaling factor is computed by dividing the width of the display window by the width of the cropping rectangle. Similarly, the vertical scaling factor is computed by dividing the height of the display window by the height of the cropping rectangle.

Down-scaling is limited upto factor 0.5 and the up-scaling factor to 8 in the software, while hardware supports from 0.25x to 8x both horizontally and vertically . The display Driver makes sure the limits are never exceeded.

The code snippet below illustrates how to scale image by factor of 2.

```
struct v4l2_format fmt;
struct v4l2_crop crop;
/* Changing display window size to 200x200 */
fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
fmt.fmt.win.w.left = 0;
fmt.fmt.win.w.top = 0;
fmt.fmt.win.w.width = 200;
fmt.fmt.win.w.height = 200;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    exit(0);
}
/* Changing crop window size to 400x400 */
crop.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
crop.c.left = 0;
crop.c.top = 0;
crop.c.width = 400;
crop.c.height = 400;
ret = ioctl(fd, VIDIOC_S_CROP, &crop);
```

```
if (ret < 0) {
    perror("VIDIOC_S_CROP\n");
    close(fd);
    exit(0);
}
/* Image should be now scaled by factor 2 */
```

Color look table

The graphics pipeline supports the color look up table. The CLUT mode uses the encoded pixel values from the input image as pointers to index the 24-bit-wide CLUT value: 1-BPP pixels address 2 entries, 2-BPP pixels address 4 entries, 4-BPP pixels address 16 entries, and 8-BPP pixels address 256 entries.

Driver supports 1, 2, 4 and 8 bits per pixel image format using color lookup table. FBIOPUTCMAP and FBIOGETCMAP can be used to set and get the color map table. When CLUT is set, the driver makes the hardware to reload the CLUT.

Following example shows how to change CLUT.

```
struct fb_cmap cmap;
unsigned short r[4]={0xFF,0x00, 0x00, 0xFF};
unsigned short g[4]={0x00, 0xFF, 0x00, 0xFF};
unsigned short b[4]={0x00, 0x00, 0xFF, 0x00};
cmap.len = 4;
cmap.red = r;
cmap.green = g;
cmap.blue = b;
if (ioctl(fd, FBIOPUTCMAP, &cmap)) {
    perror("FBIOPUTCMAP\n");
    close(fd);
    exit(3);
}
```

Streaming

V4L2 driver supports the streaming of the buffer. To do streaming minimum of three buffers should be requested by the application by using VIDIOC_REQBUFS ioctl. Once driver allocates the requested buffers application should call VIDIOC_QUERYBUF and mmap to get the physical address of the buffers and map the kernel memory to user space as explained earlier. Following are the steps to enable streaming.

1. Fill the buffers with the image to be displayed in the proper format
2. Queue buffers to the driver queue using VIDIOC_QBUF ioctl
3. Start streaming using VIDIOC_STREAMON ioctl
4. Call VIDIOC_DQBUF to get the displayed buffer
5. Repeat steps 1, 2, 4 and 5 in a loop for the frame count to be displayed
6. Call VIDIOC_STREAMOFF ioctl to stop streaming

Following example shows how to do streaming with V4l2 driver

```
/* Initially fill the buffer */
struct v4l2_requestbuffers req;
struct v4l2_buffer buf;
struct v4l2_format fmt;
```

```
/* Fill the buffers with the image */
/* Enqueue buffers */
for (i = 0; i < req.count; i++) {
    buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
    buf.index = i;
    buf.memory = V4L2_MEMORY_MMAP;
    ret = ioctl(fd, VIDIOC_QBUF, &buf);
    if (ret < 0) {
        perror("VIDIOC_QBUF\n");
        for (j = 0; j < req.count; j++){
            /* Unmap all the buffers if call fails */
            exit(0);
        }
        printf("VIDIOC_QBUF = %d\n",i);
    }
}
/* Start streaming */
a = 0;
ret = ioctl(fd, VIDIOC_STREAMON, &a);
if (ret < 0) {
    perror("VIDIOC_STREAMON\n");
    for (i = 0; i < req.count; i++)
        /* Unmap all the buffers if call fails */
    exit(0);
}
/* loop for streaming with 500 Frames*/
for(i = 0 ;i < LOOPCOUNT ;i ++) {
    ret = ioctl(fd, VIDIOC_DQBUF, &buf);
    if(ret < 0){
        perror("VIDIOC_DQBUF\n");
        for (j = 0; j < req.count; j++){
            /* Unmap all the buffers if call fails */
        }
        exit(0);
    }
    /* Fill the buffer with new data
    fill(buff_info[buf.index].start, fmt.fmt.pix.width,
fmt.fmt.pix.height,0);
    /Queue the buffer again */
    ret = ioctl(fd, VIDIOC_QBUF, &buf);
    if(ret < 0){
        perror("VIDIOC_QBUF\n");
        for (j = 0; j < req.count; j++){
             /* Unmap all the buffers if call fails */
        }
        exit(0);
    }
```

```
}
/* Streaming off */
ret = ioctl(fd, VIDIOC_STREAMOFF, &a);
if (ret < 0) {
    perror("VIDIOC_STREAMOFF\n");
    for (i = 0; i < req.count; i++){
        /* Unmap all the buffers if call fails */
    exit(0);
}
```

Software Interfaces

Frame-Buffer Driver Interface

Application Interface

open ()
To open a framebuffer device
close ()
To close a framebuffer device
ioctl ()
To send ioctl commands to the framebuffer driver.
mmap ()
To obtain the framebuffer region as mmap'ed area in user space.

Supported Standard IOCTLs

FBIOGET_VSCREENINFO, FBIOPUT_VSCREENINFO
These I/O controls are used to query and set the so-called variable screen info. This allows an application to query or change the display mode, including the color depth, resolution, timing etc. These I/O controls accept a pointer to a struct fb_var_screeninfo structure. The video mode data supplied in the fb_var_screeninfo struct is translated to values loaded into the display controller registers.

FBIOGET_FSCREENINFO
This I/O control can be used by applications to get the fixed properties of the display, e.g. the start address of the framebuffer memory. This I/O control accepts a pointer to a struct fb_fix_screeninfo.

FBIOGETCMAP, FBIOPUTCMAP
These I/O controls are used to get and set the color-map for the framebuffer. These I/O controls accept a pointer to a struct fb_cmap structure.

FBIO_BLANK
This I/O control is used to blank or unblank the framebuffer console.

Supported Custom IOCTLs

OMAPFB_WAITFORVSYNC

This ioctl can be used to put an application to sleep until next vertical sync interval of the display.

OMAPFB_GET_VRAM_INFO Ioctl returns the configured/allocated vram information.

OMAPFB_QUERY_MEM

Returns the size and type of the frame buffer.

Data Structure:

```
#define OMAPFB_MEMTYPE_SDRAM            0
#define OMAPFB_MEMTYPE_SRAM             1
#define OMAPFB_MEMTYPE_MAX              1
struct omapfb_mem_info {
 __u32 size;
 __u8  type;
 __u8  reserved[3];
};
```

Usage:

```
struct omapfb_mem_info mi;
if (ioctl(fb, OMAPFB_QUERY_MEM, &mi)) {
 perror("Error: OMAPFB_QUERY_MEM.\n");
 exit(1);
}
printf("size - %d\n", mi.size);
printf("type - %d\n", mi.type);
```

OMAPFB_SETUP_MEM

Allows user to setup the frame buffer memory, like size and type.

Data Structure:

```
#define OMAPFB_MEMTYPE_SDRAM            0
#define OMAPFB_MEMTYPE_SRAM             1
#define OMAPFB_MEMTYPE_MAX              1
struct omapfb_mem_info {
 __u32 size;
 __u8  type;
 __u8  reserved[3];
};
```

Usage:

```
struct omapfb_mem_info mi;
mi.size = <Expected size of buffer>
mi.type = <Expected type of buffer>
if (ioctl(fb, OMAPFB_SETUP_MEM, &mi)) {
 perror("Error: OMAPFB_SETUP_MEM.\n");
 exit(1);
}
```

OMAPFB_QUERY_PLANE

Query the plane (gfx) and returns the omapfb_plane_info information -

Data Structure:

```
struct omapfb_plane_info {
 __u32 pos_x;
 __u32 pos_y;
 __u8  enabled;
 __u8  channel_out;
 __u8  mirror;
 __u8  reserved1;
 __u32 out_width;
 __u32 out_height;
 __u32 reserved2[12];
};
```

Usage:

```
struct omapfb_plane_info pi;
if (ioctl(fb, OMAPFB_QUERY_PLANE, &pi)) {
 perror("Error: OMAPFB_QUERY_PLANE.\n");
 exit(1);
}
```

OMAPFB_SETUP_PLANE

TBD.

Data Structures

fb_var_screeninfo

This structure is used to query and set the so-called variable screen information. This allows an application to query or change the display mode, including the color depth, resolution, timing etc.

fb_fix_screeninfo

This structure is used by applications to get the fixed properties of the display, e.g. the start address of the framebuffer memory, framebuffer length etc.

fb_cmap

This structure is used to get/set the color-map for the framebuffer.

V4L2 Driver Interface

Application Interface

open()

To open a video device

close()

To close a video device

ioctl()

To send ioctl commands to the display driver.

mmap()

To memory map a driver allocated buffer to user space

Supported Standard IOCTLs

This section describes the standard V4L2 IOCTLs supported by the Display Driver.

NOTE: Standard IOCTLs that are not listed here are not supported. The Display Driver handles the unsupported ones by returning EINVALerror code.

VIDIOC_QUERYCAP

This is used to query the driver's capability. The video driver fills a v4l2_capability struct indicating the driver is capable of output and streaming.

VIDIOC_ENUM_FMT

This is used to enumerate the image formats that are supported by the driver. The driver fills a v4l2_fmtdesc struct.

VIDIOC_G_FMT

This is used to get the current image format or display window depending on the buffer type. The driver fills the information to a *v4l2_format* struct.

VIDIOC_TRY_FMT

This is used to validate a new image format or a new display window depending on the buffer type. The driver may change the passed values if they are not supported. Application should check what is granted.

VIDIOC_S_FMT

This is used to set a new image format or a new display window depending on the buffer type. The driver may change the passed values if they are not supported. Application should check what is granted if *VIDIOC_TRY_FMT* is not used first.

VIDIOC_CROPCAP

This is used to get the default cropping rectangle based on the current image size and the current display panel size. The driver fills a *v4l2_cropcap* struct.

VIDIOC_G_CROP

This is used to get the current cropping rectangle. The driver fills a *v4l2_crop* struct.

VIDIOC_S_CROP

This is used to set a new cropping rectangle. The driver fills a *v4l2_crop* struct. Application should check what is granted.

VIDIOC_REQBUFS

This is used to request a number of buffers that can later be memory mapped. The driver fills a *v4l2_request* buffers struct. Application should check how many buffers are granted.

VIDIOC_QUERYBUF

This is used to get a buffer's information so mmap can be called for that buffer. The driver fills a *v4l2_buffer* struct.

VIDIOC_QBUF

This is used to queue a buffer by passing a *v4l2_buffer* struct associated to that buffer.

VIDIOC_DQBUF

This is used to dequeue a buffer by passing a *v4l2_buffer* struct associated to that buffer.

VIDIOC_STREAMON

This is used to turn on streaming. After that, any *VIDIOC_QBUF* results in an image being rendered.

VIDIOC_S_CTRL, VIDIOC_G_CTRL, VIDIOC_QUERYCTRL

These ioctls are used to set/get and query various V4L2 controls like rotation, mirror and background color. Currently only rotation is supported.

VIDIOC_STREAMOFF

This is used to turn off streaming.

SYSFS Software Interfaces

User can control all dynamic configuration of DSS core and Fbdev functionality thorugh SYSFS interface.

Frame-buffer Driver sysfs attributes

Following attributes are available for user control -

```
#
# ls -1 /sys/class/graphics/fb0/
bits_per_pixel
blank
console
cursor
dev
device
mirror
mode
modes
name
overlays
overlays_rotate
pan
phys_addr
power
rotate
rotate_type
size
state
stride
subsystem
uevent
virt_addr
virtual_size
#
```

Frame-buffer Driver sysfs attributes

Acronym	Definition
bits_per_pixel	Allows user to control bits per pixel configuration, currently the supported values are 16, 24 and 32. `# echo 16/24/32 > /sys/class/graphics/fb0/ bits_per_pixel`

blank	Allows user to control lcd display blanking configuration independently.
	`# echo 0/4 > /sys/class/graphics/fb0/blank`
	
	`Values only 0(FB_BLANK_UNBLANK) and 4(FB_BLANK_POWERDOWN) is supported`

rotate	Allows user to control rotation through this entry,
	`# echo 0/1/2/3 > /sys/class/graphics/fb0/rotate`
	
	`0 - 0 degree, 1 - 90 degree, 2 - 180 degree and 3 - 270 degree respectively.`

rotate_type	Allows user to control rotation type through this entry,
	`# echo 0/1 > /sys/class/graphics/fb0/rotate_type`
	
	`0 - DMA based rotation,`
	`1 - VRFB based rotation.`
	`Currently only VRFB based rotation is supported.`

virtual_size	Allows user to configure xres_virtual and yres_virtual parameters of frame-buffer,
	`# cat /sys/class/graphics/fb0/virtual_size`
	`480,640`

virt_addr	Readonly entry, displays virtual address of the frame-buffer memory.
phys_addr	Readonly entry, displays physical address of the frame-buffer memory.

DSS Library sysfs attributes

DSS library provides/exports following attributes, which explained in detail below -

```
#
# ls -1 /sys/devices/platform/omapdss/
bus
display0
display1
display2
driver
manager0
manager1
microamps_requested_vdda_dac
modalias
overlay0
overlay1
overlay2
power
subsystem
uevent
#
```

DSS Library: display0/1/2

In all total 3 output displays are supported on EVM,

```
#
# ls -1 /sys/devices/platform/omapdss/display0/
bus
driver
enabled
microamps_requested_vdvi
mirror
name
power
rotate
subsystem
tear_elim
timings
uevent
update_mode
wss
#
```

DSS Library-display0/1/2: sysfs attributes

Acronym	Definition
enabled	User can enable/disable the display through this entry
timings	Displays the timing configuration for specific display panel
name	Shows name of the display panel/output

DSS Library: Manager0/1

In all total 2 managers are supported on EVM,

```
#
# ls -1 /sys/devices/platform/omapdss/manager0/
alpha_blending_enabled
default_color
display
name
trans_key_enabled
trans_key_type
trans_key_value
#
```

DSS Library-Manager0/1: sysfs attributes

Acronym	Definition
alpha_blending_enabled	User can enable/disable Alpha-blending through this entry.
display	Allows user to control the output display, user can set the output to any of the display
trans_key_enabled	User can enable/disable Transparency key keying through this entry
trans_key_type	User can control the Transparency key type here.
trans_key_value	User can configure Transparency color keying value through this entry.

DSS Library: Overlay0/1/2

In all total 3 Overlays/Planes/Pipelines are supported on EVM,

```
#
# ls -1 /sys/devices/platform/omapdss/overlay0/
enabled
global_alpha
input_size
manager
name
output_size
position
screen_width
#
```

DSS Library-display0/1/2: sysfs attributes

Acronym	Definition
enabled	User can enable/disable overlay through this entry.
global_alpha	User can configure global alpha value through this entry.
manager	Allows control over manager <-> overlay interface, user can configure any overlay to any of the manager

DSS2 SYSFS Examples

Please refer DSS2 SYSFS examples [3] for examples of using sysfs interface to change display settings.

Miscellaneous Configurations

The default setup/configuration is -

```
GFX   => - -  \        DVI
               \
Vid1  => - - => => => LCD
               /
Vid2  => _ _  /        TV
```

User can control/configure the various interfaces like, overlay <=> manager <=> display. This section demonstrate/explains the dynamic switching of output using above interfaces.

Switching output from LCD to DVI

Follow the steps below to switch output from LCD to DVI interface:

- Disable LCD display

```
# echo 0 > /sys/devices/platform/omapdss/display0/enabled
```

- Disable manager link to display

```
# echo "" > /sys/devices/platform/omapdss/manager0/display
```

- Configure the framebuffer driver for target display panel size

```
# fbset –fb /dev/fb0 –xres $w –yres $h –vxres $w –vyres $h
```

- Configure manager to DVI display interface

```
# echo "dvi" > /sys/devices/platform/omapdss/manager0/display
```

- Enable DVI display

```
# echo 1 > /sys/devices/platform/omapdss/display2/enabled
```

NOTE: Similar steps must be followed for switching from DVI to LCD.

NOTE: Please note that the user can read the panel configuration through sysfs entry "/sys/devices/platform/omapdss/display<index>/timings"

Switching Overlay0 (GFX) output from LCD to TV

Follow below steps to switch output from LCD to TV interface -

- Disable GFX overlay

```
# echo 0 > /sys/devices/platform/omapdss/overlay0/enabled
```

- Disable GFX overlay link to LCD manager

```
# echo "" > /sys/devices/platform/omapdss/overlay0/manager
```

- Disable LCD display output

```
# echo 0 > /sys/devices/platform/omapdss/display0/enabled
```

- Configure the framebuffer driver for target display panel size

```
# fbset –fb /dev/fb0 –xres $w –yres $h –vxres $w –vyres $h
```

- Switch GFX overlay to TV manager

```
# echo "tv" > /sys/devices/platform/omapdss/overlay0/manager
```

- Enable TV display interface

```
# echo 1 > /sys/devices/platform/omapdss/display1/enabled
```

- Enable GFX overlay

```
# echo 1 > /sys/devices/platform/omapdss/overlay0/enabled
```

NOTE: Similar steps must follow for other (Video 1 & 2) overlays.

Cloning Overlay0 (GFX) output to both LCD and TV

Follow below steps to clone GFX overlay output to both LCD and TV -

- Disable GFX overlay

```
# echo 0 > /sys/devices/platform/omapdss/overlay0/enabled
```

- Disable Video1 overlay

```
# echo 0 > /sys/devices/platform/omapdss/overlay1/enabled
```

- If fb1 is configured, disable link between Frame-buffer and overlay1

```
# echo "" > /sys/class/graphics/fb1/overlays
```

- Attach overlay 0 and overlay 1 to FB0

```
# echo "0,1" > /sys/class/graphics/fb0/overlays
```

- Configure overlay1

```
# echo "$w,$h" > /sys/devices/platform/omapdss/overlay0/output_size
```

- Set the overlay1 manager to TV

```
# echo "tv" > /sys/devices/platform/omapdss/overlay0/manager
```

- Enable both overlay (overlay 0 and 1)

```
# echo "1" > /sys/devices/platform/omapdss/overlay0/enabled
# echo "1" > /sys/devices/platform/omapdss/overlay1/enabled
```

- Enable TV out

```
# echo "1" > /sys/devices/platform/omapdss/display1/enabled
```

Driver Configuration

V4L2 video driver

To V4L2 video driver start the Linux Kernel Configuration tool.

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

- Select "Device Drivers" from the main menu.

```
    ...
    ...
    Kernel Features  --->
    Boot options  --->
    CPU Power Management  --->
    Floating point emulation  --->
    Userspace binary formats  --->
    Power management options  --->
[*] Networking support  --->
"    Device Drivers  --->"
    ...
```

```
    ...
```

- Select "Multimedia support" from the menu.

```
    ...
    ...
    Sonics Silicon Backplane  --->
    Multifunction device drivers  --->
[*] Voltage and Current Regulator Support  --->
"<*> Multimedia support  --->"
    Graphics support  --->
<*> Sound card support  --->
[*] HID Devices  --->
[*] USB support  --->
    ...
    ...
```

- Select "Video For Linux" from the menu.

```
    ...
    ...
    *** Multimedia core support ***
"<*>   Video For Linux"
[*]     Enable Video For Linux API 1 (DEPRECATED)
< >   DVB for Linux
    ...
    ...
```

- Select "Video capture adapters" from the same menu. Press <ENTER> to enter the corresponding sub-menu.

```
    ...
    ...
[ ]   Customize analog and hybrid tuner modules to build  --->
"[*]   Video capture adapters  --->"
[ ]   Memory-to-memory multimedia devices  --->
[ ]   Radio Adapters  --->
[ ]   DAB adapters
    ...
    ...
```

- Select "OMAP2/OMAP3 V4L2-DSS drivers" from the menu.

```
    ...
    ...
-*-   VPSS System module driver
<*>   VPFE Video Capture Driver
<*>     DM6446 CCDC HW module
"<*>   OMAP2/OMAP3 V4L2-Display driver"
< >   CPiA2 Video For Linux
    ...
    ...
```

- In case of modular build choose "M" for "OMAP2/OMAP3 V4L2-Display driver", as shown below -

```
    ...
    ...
-*-   VPSS System module driver
<*>   VPFE Video Capture Driver
<*>     DM6446 CCDC HW module
"<M>   OMAP2/OMAP3 V4L2-Display driver"
< >   CPiA2 Video For Linux
    ...
    ...
```

All dependencies are being handled through standard Linux Kconfig rules; after building the modules it will create "omap-vout.ko" file under "<kernel home>/drivers/media/video/omap/" directory.

NOTE: Please note that based on Kconfig rule, other dependent module may get build as module, so please make sure that you insert all other dependent modules into the kernel before using omap-vout.ko.

Framebuffer driver

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

- Select "Device Drivers" from the main menu.

```
...
    ...
    Kernel Features  --->
    Boot options  --->
    CPU Power Management  --->
    Floating point emulation  --->
    Userspace binary formats  --->
    Power management options  --->
[*] Networking support  --->
"   Device Drivers  --->"
    ...
    ...
```

- Select "Graphics support" from the menu.

```
    ...
    ...
    Sonics Silicon Backplane  --->
    Multifunction device drivers  --->
[*] Voltage and Current Regulator Support  --->
<*> Multimedia support  --->
"    Graphics support  --->"
<*> Sound card support  --->
[*] HID Devices  --->
[*] USB support  --->
    ...
    ...
```

- Select "Support for frame buffer devices" from the menu.

```
    ...
    ...
<M> Lowlevel video output switch controls
"<*> Support for frame buffer devices  --->"
< > Virtual Frame Buffer support (ONLY FOR TESTING!)
< > E-Ink Broadsheet/Epson S1D13521 controller support
< > Fujitsu MB862xx GDC support
    ...
    ...
```

- Select "OMAP2/3 Display Subsystem support (EXPERIMENTAL)" from the same menu.

```
    ...
    ...
< > Fujitsu MB862xx GDC support
< > E-Ink Broadsheet/Epson S1D13521 controller support
[ ] Check bootloader initialization
"-*- OMAP2/3 Display Subsystem support (EXPERIMENTAL)  --->"
[*] Backlight & LCD device support  --->
    ...
    ...
```

- Configure default VRAM size to the required/expected size of buffer, the default is 4MB.

Go inside option "OMAP2/3 Display Subsystem support"

```
    ...
    ...
--- OMAP2/3 Display Subsystem support (EXPERIMENTAL)
"(4)   VRAM size (MB)"
[*]   Debug support
    ...
    ...
```

- Select "DPI support" from the menu.

```
    ...
    ...
[ ]   Debug support
[*]   DPI support
[ ]   RFBI support
    ...
    ...
```

- Select "VENC support" from the menu.

```
    ...
    ...
[ ]   Debug support
[ ]   RFBI support
"[*]   VENC support"
```

```
          OMAP2_VENC_OUT_TYPE (Use S-Video output interface)  --->
[ ]   SDI support
    ...
    ...
```

The default TV out interface is S-Video. Other option available is Composite.
NOTE: On OMAP3EVM-1 (< Rev-E) SW1.6 is used to select between S-Video and Composite outputs. **ON :- S-Video, OFF :- Composite**

- Select DSI Clock as a source clock for DSS/DISPC

```
    ...
    ...
  [ ]   SDI support
  [*]   DSI support
      [*]     Use DSI PLL for PCLK (EXPERIMENTAL)
  [ ]   Fake VSYNC irq from manual update displays
```

NOTE: Please note that by default DSI clock is enabled as a source clock for DSS/DISPC.

- Select "OMAP2/3 frame buffer support" from the same menu. Press <ENTER> to enter the corresponding sub-menu.

```
    ...
    ...
[ ]   Fake VSYNC irq from manual update displays
(1)   Minimum FCK/PCK ratio (for scaling)
"<*>   OMAP2/3 frame buffer support (EXPERIMENTAL)  --->"
      OMAP2/3 Display Device Drivers  --->
    ...
    ...
```

- Value for Number of framebuffers can be changed here.

```
--- OMAP2/3 frame buffer support (EXPERIMENTAL)
[*]   Debug support for OMAP2/3 FB
"(1)   Number of framebuffers"
```

NOTE: If this value is set as 1, the graphics pipeline of the DSS is controlled by the FBDEV interface and both video pipelines by the V4L2 interface.
If this value is set as 2, the graphics pipeline and one video pipeline is controlled by the FBDEV interface and one video pipeline by the V4L2 interface.
If this value is set as 3, all 3 pipelines are controlled by the FBDEV interface.

- Select the supported display panels, exit one level up to the menu "OMAP2/3 Display Subsystem support" and go inside "OMAP2/3 Display Device Drivers"

```
    ...
    ...
(1)   Minimum FCK/PCK ratio (for scaling)
<*>   OMAP2/3 frame buffer support (EXPERIMENTAL)  --->
"OMAP2/3 Display Device Drivers  --->"
    ...
    ...
```

- Select the supported display panels, as shown below

```
<*> Generic Panel
<*> Sharp LS037V7DW01 LCD Panel
<*> Sharp LQ043T1DG01 LCD Panel
< > Taal DSI Panel
    ...
    ...
```

NOTE: In case of OMAP3EVM "Sharp LS037V7DW01 LCD Panel" is used and in case of AM3517EVM "Sharp LQ043T1DG01 LCD Panel" is being used.

- In case of modular build choose "M" for "OMAP2/3 Display Subsystem support (EXPERIMENTAL)", as shown below -

```
    ...
    ...
< > Fujitsu MB862xx GDC support
< > E-Ink Broadsheet/Epson S1D13521 controller support
[ ] Check bootloader initialization
"<M> OMAP2/3 Display Subsystem support (EXPERIMENTAL)  --->"
[*] Backlight & LCD device support  --->
    ...
    ...
```

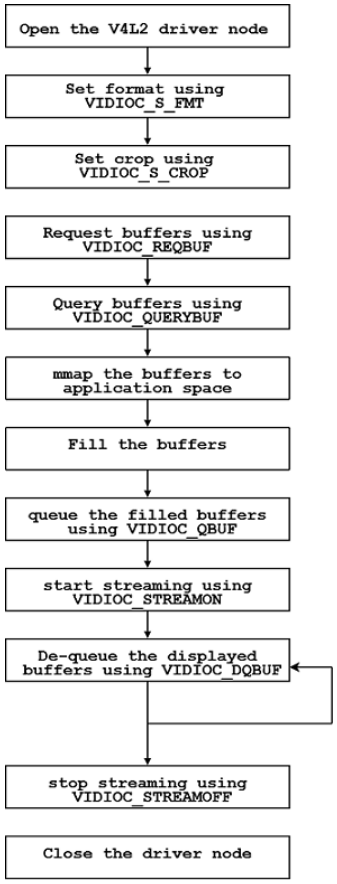
All dependencies are being handled through standard Linux Kconfig rules; after building the modules it will create various .ko files under "<kernel home>/drivers/video/omap2/" directory.

NOTE: Please note that based on Kconfig rule, other dependent module may get build as module, so please make sure that you insert all other dependent modules into the kernel before using omapfb.ko.

Sample Application Flow

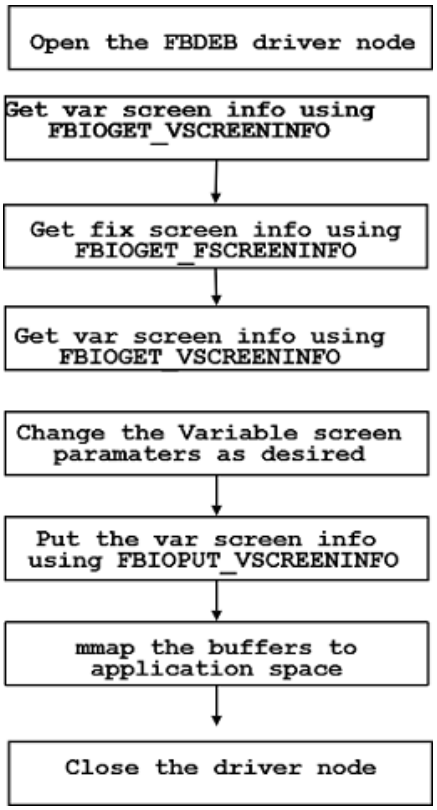
This chapter describes the application flow using the V4l2 and FBDEV drivers.

V4L2-Display Application Flow



Application for v4l2 driver using MMAP buffers

Fbdev-Display Application Flow



Application for FBDEV driver

References

[1] http://linux.bytesex.org/v4l2/

[2] http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf

[3] http://processors.wiki.ti.com/index.php?title=DSS2_SYSFS_Examples

UserGuideOmap35xCaptureDriver PSP 04.02.00.07

1. **Please note that with this release we have migrated to new Media-controller framework, so the User interface has been changed. Please refer to below UserGuide for complete details about media-controller usage.**
2. **Also, Please note that Media-controller is not a replacement for standard V4L2 framework, this framework is plug-in to the existing V4L2 framework to allow user to create/enable/disable the link and set/get/enum format for the given link/entity.**
3. Since this is new framework which is still under experiment, there could be issues with it. Please refer to Arago [1] for any updates or bug fixes.

Introduction

The camera ISP is a key component for imaging and video applications such as video preview, video record, and still-image capture with or without digital zooming.

The camera ISP provides the system interface and the processing capability to connect RAW image-sensor modules and video decoders to the OMAP35x device.

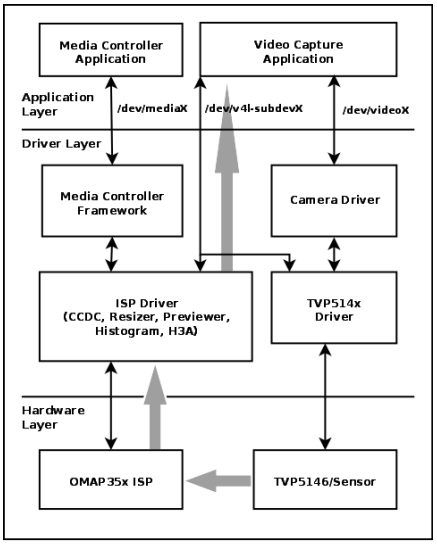
The capture module consists of the following interfaces:

- One S-video SD input in BT.656 format.
- One Composite SD input in BT.656 format.
- One Camera sensor interface in YUV format

BT656 video inputs are connected to one TVP5146 decoder and sensor is directly interfaced to OMAP35x CCDC. The application must create the link before starting streaming on the streaming device node of the link.

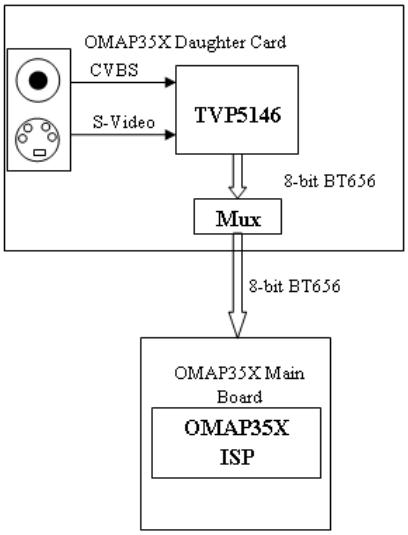
NOTE: Only one input can be captured or selected at any given point of time.

The following figure shows the basic block diagram of capture interface for TVP5146 video decoder.



Capture Driver Component Overview

The following figure shows the physical connection and inputs for TVP5146 decoder.



Capture Physical Input Interface

The Media-Controller framework has been adopted for OMAP3 ISP capture module. The Media-Controller framework exports hardware topology to the User space application as mesh or network of devices. Then User independently can set/get/enum formats at each entity and allows user to set/get/enumerate the link in current topology.

Since Media-controller is just a plug-in to the existing V4L2 framework, the V4L2 Capture driver model is still being used for streaming and all standard V4L2 interfaces. The V4L2 driver model is widely used across many platforms in the Linux community. V4L2 provides good streaming support and support for many buffer formats. It also has its own buffer management mechanism that can be used.

References

- OMAP35x Camera Interface Subsystem (ISP) TRM

Author: Texas Instruments, Inc. Literature Number: SPRUFA2

- OMAP35x Memory Management Units (MMUs)TRM

Author: Texas Instruments, Inc. Literature Number: SPRUFF5

- Video for Linux Two API Specification

Author: Michael H Schimek Version: 0.23

- Media-Controller Framework

<Linux Kernel>/Documentation/video4linux/

Acronyms & Definitions

Capture Driver: Acronyms

Acronym	Definition
MMDC	Mass Market Daughter Card/Customer Daughter Card
3A	Auto White Balance, Auto Focus, Auto Exposure
API	Application Programming Interface
CCDC	Input interface block of ISP
DMA	Direct Memory Access
I/O	Input & Output
IOCTL	Input & Output Control
MMU	Memory Management Unit
V4L2	Video for Linux specification version 2
MC	Media-Controller
YUV	Luminance + 2 Chrominance Difference Signals (Y, Cr, Cb) Color Encoding

Features

The ISP Capture Driver provides the following features:

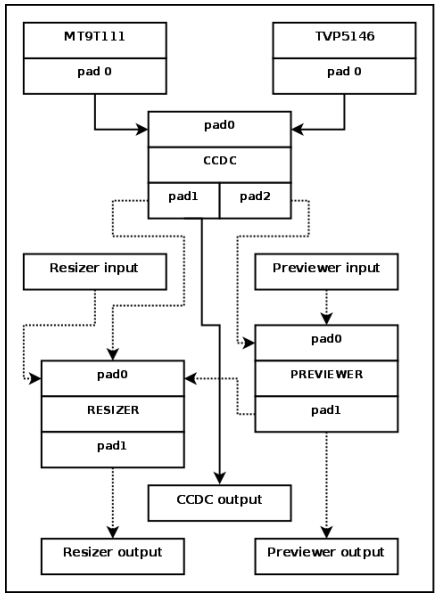
- Supports multi software channel of capture and a corresponding device node (/dev/videoX) is created.
- Supports multi sub-device nodes and corresponding sub device node (/dev/v4l2-subdevX) is created.
- Supports single I/O instance and multiple control instances.
- Supports buffer access mechanism through memory mapping and user pointers.
- Supports dynamic switching among input interfaces with some necessary restrictions wherever applicable.
- Supports NTSC and PAL standard on Composite and S-Video interfaces.
- Supports 8-bit BT.656 capture in UYVY and YUYV interleaved formats.
- Supports standard V4L2 IOCTLs to get/set various control parameters like brightness, contrast, saturation, hue and auto gain control.
- TVP5146 (TVP514x) decoder driver module can be used statically or dynamically (insmod and rmmod supported).
- In USERPTR mode of operation both malloc'd and IO mapped buffers are supported.
- The camera ISP driver supports both static into kernel and modular build.

Media-Controller interface

OMAP3 ISP Camera driver has been migrated to the newly added Media-Controller (now onwards will refer as MC) framework into V4L2 sub-system. Basically MC exposes complex media device topology to the User Spaces a graph pf building blocks called **Entities** and they are connected to each other through **Pads**. User now has a capability to activate/deactivate links directly from User Space application.

Architecture

Following block diagram shows basic architecture of the MC on OMAP3 ISP Module -



Media-Controller Framework for OMAP3 ISP
(cut down version)

The system architecture diagram illustrates the software components that are relevant to the Camera Driver. Some components are outside the scope of this design document.

NOTE: Currently only MT9T111/TVP5146 ==> CCDC ==> Memory path has been validated.

The following is a brief description of each component in the figure.

As shown in the above architecture diagram, media topology for OMAP3 ISP is consist of various entities, like CCDC, Previewer, Resizer, external peripherals (TVP5146 & MT9T111), etc... Each entity will have one or more input/output pads depending on functionality.

MT9T111/TVP5146: The MT9T111/TVP5146 Driver is platform and board independent. Driver registers to the MC framework as an sub-device entity and sub-device to V4L2 layer. This is V4L2 compliant driver with addition of MC support. MT9T111 is parallel sensor interface whereas TVP5146 is BT656 interface, both uses 8-bit data bus connected to ISP.

CCDC: CCDC is a HW block in Camera ISP which acts as a data input port. It receives data from the sensor/decoder through parallel or serial interface. The CCDC driver (not a independent platform driver) registers to MC framework as media entity with various input/output pads. It is configured by the ISP driver based on the sensor/decoder attached and desired output from the camera driver.

CCDC output: This is streaming V4L2 device node, where standard V4L2 application would work.

Resizer: Resizer is HW block in Camera ISP which works in 2 ways, one-shot mode (memory-to-memory) and in on-the-fly mode (ccdc/previewer-->resizer path). Resizer driver (not a independent platform driver) registers to MC framework as a media entity with 2 pads (one input and another output). The image format at the input and output pads may differ and based on that scaling ratio is being configured.

Resizer input/output: These are streaming device nodes, where standard V4L2 application would work.

Previewer: Previewer is HW block in Camera ISP which works in 2 ways, one-shot mode (memory-to-memory) and in on-the-fly mode (ccdc-->previewer path). Previewer driver (not a independent platform driver) registers to MC framework as a media entity with 2 pads (one input and another output). The image format at the input and output pads may differ and based on that scaling ratio is being configured.

Previewer input/output: These are streaming device nodes, where standard V4L2 application would work.

Software Design Interfaces

Opening and Closing of driver

The device can be opened using open call from the application, with the device name and mode of operation as parameters. Based on underneath hardware and corresponding media topology, there could be multiple media device nodes (/dev/mediaX). In case of OMAP3 ISP, only one media topology is possible and is being exported by /dev/media0.

Example:

```
/* Open a media device in blocking mode */
fd = open("/dev/media0", O_RDWR);
if (fd == -1) {
    perror("failed to open media device\n");
    return -1;
}
/* closing of channel */
close (fd);
```

Enumerate available Media Entities

This IOCTL is used to enumerate the information of available entities (both video-device and subdevice). It includes information like <TBD>.

Ioctl: *MEDIA_IOC_ENUM_ENTITIES*

It takes pointer to *media_entity_desc* structure. Application provides the id number for which it requires the information, in *id* member of *media_entity_desc* structure. Entities can be enumerated by oring the id with the *MEDIA_ENTITY_ID_FLAG_NEXT* flag. The driver will return information about the entity with the smallest id strictly larger than the requested one (next entity), or the &EINVAL; if there is none.

Example:

```
/* Define place holder for entities to enumerate */
struct media_entity_desc entity[20];
int index = 0;
...
...
do {
      memset(&entity[index], 0, sizeof(entity));
      /* Set the id parameter to enumerate */
      entity[index].id = index | MEDIA_ENT_ID_FLAG_NEXT;

      ret = ioctl(media_fd, MEDIA_IOC_ENUM_ENTITIES,
&entity[index]);
      if (ret < 0) {
                  break;
      }else {
            /*
             * Application should pick up respective entity and
corresponding
             * device/sub-device nodes for configuration and streaming.
             */
```

```
            if (!strcmp(entity[index].name, "mt9t111 2-003c")) {
                  entity_mt9t111 =  entity[index].id;
            }
            else if (!strcmp(entity[index].name, "tvp514x 3-005c")) {
                  entity_tvp5146 =  entity[index].id;
            }
                else if (!strcmp(entity[index].name, "OMAP3 ISP CCDC"))
 {
                  entity_ccdc =  entity[index].id;
                }
            printf("[%d]:%s\n", entity[index].id, entity[index].name);
      }
      index++;
}while (ret == 0);

printf("Total number of available entities: %u\n", index);
```

Enumerate available Links & Pads

This IOCTL is used to enumerate the information of available links with all entities within media topology. It includes information like <TBD>.

Ioctl: *MEDIA_IOC_ENUM_LINKS*

It takes pointer to *media_links_enum* structure. Application provides the entity id number for which it requires the information, in *entity* member of *media_links_enum* structure which he would have known from *MEDIA_IOC_ENUM_ENTITIES*. Please make sure that you allocate memory space for pads/links for *pads* and *links* member of *media_links_enum* structure.

Example:

```
struct media_links_enum links;
int index = 0, i;

/*
 * Details from MEDIA_IOC_ENUM_ENTITIES is being used here
 */
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 < num_entities; index++) {

  links.entity = entity[index].id;

  ret = ioctl(media_fd, MEDIA_IOC_ENUM_LINKS, &links);
  if (ret < 0) {
      break;
  } else {
    /* Display pads info */
    if(entity[index].pads)
```

```
      printf("pads for entity %d=", entity[index].id);

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

    /* Now display link info */
    for(i = 0; i < entity[index].links; i++) {
      printf("[%d:%d]===>[%d:%d]", links.links->source.entity,
                    links.links->source.index,
                    links.links->sink.entity,
                    links.links->sink.index);
      if(links.links->flags & MEDIA_LINK_FLAG_ENABLED)
        printf("\tACTIVE\n");
      else
        printf("\tINACTIVE \n");
      links.links++;
    }
  }
}
```

Enable/Setup/Activate Links

This IOCTL is used to setup/enable/activate links with specified entities within media topology.
Ioctl: *MEDIA_IOC_SETUP_LINK*
It takes pointer to *media_link_desc* structure. Application provides the all required information which would have been extracted using enumeration of media device while calling this ioctl.
It take source and sink entity id, media pad type and pad index as an input argument.

Example:

```
/*
 * Details from MEDIA_IOC_ENUM_ENTITIES and MEDIA_IOC_ENUM_LINKS is
being used here
 */
struct media_link_desc link;

memset(&link, 0, sizeof(link));

link.flags |=  MEDIA_LINK_FLAG_ENABLED;
link.source.entity = entity_tvp5146;
link.source.index = 0;
link.source.flags = MEDIA_PAD_FLAG_OUTPUT;

link.sink.entity = entity_ccdc;
link.sink.index = 0;
```

```
link.sink.flags = MEDIA_PAD_FLAG_INPUT;

ret = ioctl(media_fd, MEDIA_IOC_SETUP_LINK, &link);
if(ret) {
      printf("failed to setup/enable/activate link between tvp5146 and
ccdc\n");
      goto cleanup;
}
printf("Link between [tvp5146]===>[ccdc] enabled...\n");
```

V4L2 Sub-Device Interface

All sub-devices in the media topology will have separate sub-device node (/dev/v4l-subdevX) exported to the

Software Design Interfaces

Opening and Closing of driver

The device can be opened using open call from the application, with the device name and mode of operation as parameters. Based on underneath hardware and corresponding media topology, there could be multiple sub-device nodes (/dev/v4l-subdevX). In case of OMAP3 ISP, we have in all total 8 sub-devices being exported by /dev/v4l-subdevX.

Example:

```
/* Open a sub-device 0 in blocking mode */
fd = open("/dev/v4l-subdev0", O_RDWR);
if (fd == -1) {
    perror("failed to open subdev device\n");
    return -1;
}
/* closing of channel */
close (fd);
```

Set the format at each pad

Before starting streaming on corresponding channel, user must set the format at each pad of every entity of the configured link.

Ioctl: *VIDIOC_SUBDEV_S_FMT*
It takes pointer to *v4l2_subdev_format* structure. Application provides the all required information like,

Media bus format type, possible values/options are

```
  - V4L2_SUBDEV_FORMAT_TRY
  - V4L2_SUBDEV_FORMAT_ACTIVE
```

Data format code, possible codes options are -

Below example illustrates how user can set the format at pad level -

```
  - V4L2_MBUS_FMT_UYVY8_2X8
  - V4L2_MBUS_FMT_VYUY8_2X8
```

NOTE: Please note that currently we are supporting only 2 codes mentioned above but theoretically there various formats supported by driver. Please refer to the file *include/linux/v4l2-mediabus.h* for all supported media bus formats.

Example:

```
struct v4l2_subdev_format fmt;

/*
 * Set the required format
 */
memset(&fmt, 0, sizeof(fmt));
fmt.pad = 0;
fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
fmt.format.code = V4L2_MBUS_FMT_UYVY8_2X8;
fmt.format.width = <width>;
fmt.format.height = <height>;
fmt.format.colorspace = V4L2_COLORSPACE_SMPTE170M;
fmt.format.field = V4L2_FIELD_INTERLACED;
ret = ioctl(ccdc_fd, VIDIOC_SUBDEV_S_FMT, &fmt);
if(ret) {
    printf("failed to set format on pad %x\n", fmt.pad);
}
```

Get the format at each pad

This IOCTL is used to get the format set/configured at specified pad of underneath sub-device.
Ioctl: *VIDIOC_SUBDEV_G_FMT*
It takes pointer to *v4l2_subdev_format* structure. Application provides pad number and corresponding media type.
It take source and sink entity id, media pad type and pad index as an input argument.

Example:

```
struct v4l2_subdev_format fmt;

/*
 * Get the format used at pad 0 of underneath sub-device
 */
memset(&fmt, 0, sizeof(fmt));
fmt.pad = 0;
fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
ret = ioctl(ccdc_fd, VIDIOC_SUBDEV_G_FMT, &fmt);
if(ret) {
    printf("failed to set format on pad %x\n", fmt.pad);
}
```

V4L2 Streaming Interface

Software Design Interfaces

Opening and Closing of driver

The device can be opened using open call from the application, with the device name and mode of operation as parameters. Application should open the driver in blocking mode. In this mode, DQBUF IOCTL will not return until an empty frame is available.

```
/* Open a video capture logical channel in blocking mode */
fd = open("/dev/video0", O_RDWR);
if (fd == -1) {
    perror("failed to open Capture device\n");
    return -1;
}
/* closing of channel */
close (fd);
```

Buffer Management

ISP Capture driver can work with physically non-contiguous buffers. It uses the ISP MMU to capture data to buffers scattered to a set of page frames. Hence, in user pointer mode the application can allocate buffers in user space, which need not be physically contiguous, and pass this directly to driver for capture operation. The only restriction for the user buffer is that, the buffer should be aligned to 32 bytes boundary.
The driver supports both memory usage modes:

1. Memory map buffer mode
2. User Pointer mode

In Memory map buffer mode, application can request memory from the driver by calling *VIDIOC_REQBUFS* IOCTL. In user buffer mode, application needs to allocate memory using some other mechanism in user space like *malloc* or *memalign*. In driver buffer mode, maximum number of buffers is limited to *VIDEO_MAX_FRAME* (defined in driver header files) and is limited by the available memory in the kernel.
The main steps that the application must perform for buffer allocation are:

1. Allocating Memory
2. Getting Physical Address
3. Mapping Kernel Space Address to User Space

Allocating Memory

This IOCTL is used to allocate memory for frame buffers. This is the necessary IOCTL for streaming IO. It has to be called for both driver buffer mode and user buffer mode. Using this IOCTL, driver will know whether driver buffer mode or user buffer mode will be used.
Ioctl: VIDIOC_REQBUFS
It takes a pointer to instance of *v4l2_requestbuffers* structure as an argument. User should specify buffer type as *V4L2_BUF_TYPE_VIDEO_CAPTURE*, number of buffers, and memory type *V4L2_MEMORY_MMAP*, *V4L2_MEMORY_USERPTR* at the time of buffer allocation.
Constraint: This IOCTL can be called only once from the application. This IOCTL is necessary IOCTL.

Example:

```
/* structure to store buffer request parameters */
struct v4l2_requestbuffers reqbuf;
```

```
reqbuf.count = numbuffers;
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_REQBUFS, &reqbuf);
if (ret < 0) {
    printf("cannot allocate memory\n");
    close(fd);
    return -1;
}
printf("Number of buffers allocated = %d\n", reqbuf.count);
```

Getting Physical Address

This IOCTL is used to query buffer information like buffer size and buffer physical address. This physical address is used in mmapping the buffers. This IOCTL is necessary for driver buffer mode as it provides the physical address of buffers, which are used to mmap system call the buffers.

Ioctl: VIDIOC_QUERYBUF

It takes a pointer to instance of *v4l2_buffer* structure as an argument. User has to specify buffer type as *V4L2_BUF_TYPE_VIDEO_CAPTURE*, buffer index, and memory type *(V4L2_MEMORY_MMAP)* at the time of querying.

Example:

```
/* allocate buffer by VIDIOC_REQBUFS */
/* structure to query the physical address of allocated buffer */
struct v4l2_buffer buffer;
buffer.index = 0; /* buffer index for quering -0 */
buffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buffer.memory = V4L2_MEMORY_MMAP;
if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < -1) {
    printf("buffer query error.\n");
    close(fd);
    exit(-1);
}
The buffer.m.offset will contain the physical address returned from
driver.
```

Mapping Kernel Space Address to User Space

Mapping the kernel buffer to the user space can be done via mmap. This is only required for MMAP buffer mode. User can pass buffer size and physical address of buffer for getting the user space address.

Example:

```
/* allocate buffer by VIDIOC_REQBUFS */
/* query the buffer using VIDIOC_QUERYBUF */
/* addr hold the user space address */
int addr;
addr = mmap(NULL, buffer.size,PROT_READ | PROT_WRITE, MAP_SHARED, fd,
buffer.m.offset);
/* buffer.m.offset is same as returned from VIDIOC_QUERYBUF */
```

Query Capabilities

This IOCTL is used to verify kernel devices compatibility with V4L2 specification and to obtain information about individual hardware capabilities. In this case, it will return capabilities provided by ISP capture driver and current decoder driver.

Ioctl: VIDIOC_QUERYCAP

Capabilities can be video capture *V4L2_CAP_VIDEO_CAPTURE* and streaming *V4L2_CAP_STREAMING*. It takes pointer to *v4l2_capability* structure as an argument. Capabilities can be accessed by capabilities field in the *v4l2_capability* structure.

Example:

```
struct v4l2_capability capability;
ret = ioctl(fd, VIDIOC_QUERYCAP, &capability);
if (ret < 0) {
    printf("Cannot do QUERYCAP\n");
    return -1;
}
if (capability.capabilities & V4L2_CAP_VIDEO_CAPTURE) {
    printf("Capture capability is supported\n");
}
if (capability.capabilities & V4L2_CAP_STREAMING) {
    printf("Streaming is supported\n");
}
```

Input Enumeration

This IOCTL is used to enumerate the information of available inputs (analog interface). It includes information like name of input type and supported standards for that input type.

Ioctl: VIDIOC_ENUMINPUT

It takes pointer to *v4l2_input* structure. Application provides the index number for which it requires the information, in index member of *v4l2_input* structure. Index with value zero indicates first input type of the decoder. It returns combination of the standards supported on this input in the std member of *v4l2_input* structure.

Example:

```
struct v4l2_input input;
i = 0;
while(1) {
      input.index = i;
      ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
      if (ret < 0)
      break;
      printf("name = %s\n", input.name);
      i++;
}
```

Set Input

This IOCTL is used to set input type (analog interface type).

Ioctl: VIDIOC_S_INPUT

This IOCTL takes pointer to integer containing index of the input which has to be set. Application will provide the index number as an argument.

0 - Composite input, 1 - S-Video input.

Example:

```
int index = 1; /*To set S-Video input*/
struct v4l2_input input;
ret = ioctl(fd, VIDIOC_S_INPUT, &index);
if (ret < 0) {
    perror("VIDIOC_S_INPUT\n");
    close(fd);
    return -1;
}
input.index = index;
ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
if (ret < 0) {
    perror("VIDIOC_ENUMINPUT\n");
    close(fd);
    return -1;
}
printf("name of the input = %s\n",input.name);
```

Get Input

This IOCTL is used to get the current input type (analog interface type).

Ioctl: VIDIOC_G_INPUT

This IOCTL takes pointer to integer using which the detected inputs will be returned. It will return the software managed input detected during open system call. Application will provide the index number as an output argument.

Example:

```
int input;
struct v4l2_input input;
ret = ioctl(fd, VIDIOC_G_INPUT, &input);
if (ret < 0) {
    perror("VIDIOC_G_INPUT\n");
    close(fd);
    return -1;
}
input.index = index;
ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
if (ret < 0) {
    perror("VIDIOC_ENUMINPUT\n");
    close(fd);
    return -1;
}
printf("name of the input = %s\n", input.name);
```

Standard Enumeration

This IOCTL is used to enumerate the information regarding video standards. This IOCTL is used to enumerate all the standards supported by the registered decoder.

Ioctl: VIDIOC_ENUMSTD

This IOCTL takes a pointer to *v4l2_standard* structure. Application provides the index of the standard to be enumerated in the index member of this structure. It provides information like standard name, standard ID defined at V4L2 header files (few new standards are included in the respective decoder header files, which were not available in standard V4L2 header files), and numerator and denominator values for frame period and frame lines. It takes index as an argument as a part of *v4l2_standard* structure.

Index with value zero provides information for the first standard among all the standards of all the registered decoders. If the index value exceeds the number of supported standards, it returns an error.

Example:

```
struct v4l2_standard standard;
i = 0;
while(1) {
      standard.index = i;
      ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
      if (ret < 0)
            break;
      printf("name = %s\n", std.name);
      printf("framelines = %d\n", std.framelines);
      printf("numerator = %d\n",
      std.frameperiod.numerator);
      printf("denominator = %d\n",
      std.frameperiod.denominator);
      i++;
}
```

Standard Detection

This IOCTL is used to detect the current video standard set in the current decoder.

Ioctl: VIDIOC_QUERYSTD

It takes a pointer to *v4l2_std_id* instance as an output argument. Driver will call the current decoder's function internally (which has been initialized) to detect the current standard set in hardware. Support of this IOCTL depends on decoder device, whether it can detect a standard or not.

Note: This IOCTL should be called by the application so that the camera driver can configure ISP properly with the detected decoder standard. Standard IDs are defined in the V4L2 header files

Example:

```
v4l2_std_id std;
struct v4l2_standard standard;
ret = ioctl(fd, VIDIOC_QUERYSTD, &std);
if (ret < 0) {
      perror("VIDIOC_QUERYSTD\n");
      close(fd);
      return -1;
}
while(1) {
```

```
      standard.index = i;
      ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
      if (ret < 0)
            break;
      if (standard.std & std) {
            printf("%s standard detected\n", standard.name);
            break;
      }
      i++;
}
```

Set Standard

This IOCTL is used to set the standard in the decoder.

Ioctl: VIDIOC_S_STD

It takes a pointer to *v4l2_std_id* instance as an input argument. If the standard is not supported by the decoder, the driver will return an error Standard IDs are defined in the V4L2 header files (few new standards are included in respective decoder header files, which were not available in standard V4L2 header files).

Note: Application need not call this IOCTL as the decoder can auto detect the current standard. This is required only when the application needs to set a particular standard. In this case, the decoder driver auto detect function is disabled. Auto detect can be enabled again only by closing and re-opening the driver.

Example:

```
v4l2_std_id std = V4L2_STD_NTSC;
ret = ioctl(fd, VIDIOC_S_STD, &std);
if (ret < 0) {
      perror("S_STD\n");
      close(fd);
      return -1;
}
while(1) {
      standard.index = i;
      ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
      if (ret < 0)
            break;
      if (standard.std & std) {
            printf("%s standard is selected\n");
            break;
      }
      i++;
}
```

Get Standard

This IOCTL is used to get the current standard in the current decoder.

Ioctl: VIDIOC_G_STD

It takes a pointer to *v4l2_std_id* instance as an output argument. Standard IDs are defined in the V4L2 header files

Example:

```
v4l2_std_id std;
ret = ioctl(fd, VIDIOC_G_STD, &std);
if (ret < 0) {
      perror("G_STD\n");
      close(fd);
      return -1;
}
while(1) {
      standard.index = i;
      ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
      if (ret < 0)
            break;
      if (standard.std & std) {
            printf("%s standard is selected\n");
            break;
      }
      i++;
}
```

'Format Enumeration

This IOCTL is used to enumerate the information of pixel formats. The driver supports only two pixel form at -8-bit UYVY interleaved and 8-bit YUYV interleaved.

Ioctl: VIDIOC_ENUM_FMT

It takes a pointer to instance of *v4l2_fmtdesc* structure as an output parameter. Application must provide the buffer type in the type argument of *v4l2_fmtdesc* structure as *V4L2_BUF_TYPE_VIDEO_CAPTURE* and index member of this structure as zero.

Example:

```
struct v4l2_fmtdesc fmt;
i = 0;
while(1) {
      fmt.index = i;
      ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmt);
      if (ret < 0)
            break;
      printf("description = %s\n",fmt.description);
      if (fmt.type == V4L2_BUF_TYPE_VIDEO_CAPTURE)
            printf("Video capture type\n");
      if (fmt.pixelformat == V4L2_PIX_FMT_YUYV)
            printf("V4L2_PIX_FMT_YUYV\n");
      i++;
}
```

Set Format

This IOCTL is used to set the format parameters. The format parameters are line offset, storage format, pixel format, and so on. This IOCTL is one of the necessary IOCTL. If it is not set, it uses the following default values:

- Default storage format - V4L2_FIELD_INTERLACED

This IOCTL expects proper width and height members of the *v4l2_format* structure from application as per the standard selected. Please note that, *V4L2_FIELD_INTERLACED* is the only storage format supported.
The application can decide the buffer pixel format using pixelformat member of this IOCTL. The current driver supports - 8-bit UYVY interleaved and 8-bit YUYV interleaved formats.
The desired pitch of the buffer can be set by using the bytesperline member. The pitch should be at least one line size in bytes. When changing the pitch, the application should also modify the sizeimage member accordingly - sizeimage should be at least pitch * image height.
The driver allocates buffer of size sizeimage member of the *v4l2_format* structure passed through this IOCTL for both mmap buffer and user pointer mode. Driver validates the provided buffer size along with the other members and uses this buffer size for calculating offsets for storing video data.
This IOCTL is a necessary IOCTL for the user buffer mode because driver will know the buffer size for user buffer mode. If it not called for the user buffer mode, driver assumes the default buffer size and calculates offsets accordingly.
Ioctl: VIDIOC_S_FMT
It will take pointer to instance of v4l2_format structure as an input parameter. If the type member is *V4L2_BUF_TYPE_VIDEO_CAPTURE*, it checks pixel format, pitch value, and image size. It returns an error, if the parameters are invalid.
Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
/* for  NTSC standard */
fmt.fmt.pix.width = 720;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
      perror("VIDIOC_S_FMT\n");
      close(fd);
      return -1;
}
```

Get Format

This IOCTL is used to get the current format parameters.
Ioctl: VIDIOC_G_FMT
It takes a pointer to instance of *v4l2_format* structure as an input parameter. Driver provides format parameters in the structure pointer passed as an argument. *v4l2_format* structure contains parameters like pixel format, image size, bytes per line, and field type. For type *V4L2_BUF_TYPE_VIDEO_CAPTURE*, the *v4l2_pix_format* structure of fmt union is filled.
Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
      perror("VIDIOC_G_FMT\n");
      close(fd);
      return -1;
}
if (fmt.fmt.pix.pixelformat == V4L2_PIX_FMT_YUYV)
      printf("8-bit UYVY pixel format\n");
printf("Size of the buffer = %d\n", fmt.fmt.pix.sizeimage);
printf("Line offset = %d\n", fmt.fmt.pix.bytesperline);
if (fmt.fmt.pix.field == V4L2_FIELD_INTERLACED)
      printf("Storate format is interlaced frame format");
```

Try Format

This IOCTL is used to validate the format parameters provided by the application. It checks parameters and returns the correct parameter, if any parameter is incorrect. It returns error only if the parameters passed are ambiguous.

Ioctl: VIDIOC_TRY_FMT

It takes a pointer to instance of v4l2_format structure as an input/output parameter If the type member is *V4L2_BUF_TYPE_VIDEO_CAPTURE*, it checks pixel format, pitch value, and image size. It returns errors to the application, if the parameters are invalid.

Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
fmt.fmt.pix.sizeimage = size;
fmt.fmt.pix.bytesperline = pitch;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
ret = ioctl(fd, VIDIOC_TRY_FMT, &fmt);
if (ret < 0) {
      perror("VIDIOC_TRY_FMT\n");
      close(fd);
      return -1;
}
```

Query Control

This IOCTL is used to get the information of controls that is, brightness, contrast, and so on supported by the current decoder.

Ioctl: VIDIOC_QUERYCTRL

This IOCTL takes a pointer to the instance of *v4l2_queryctrl* structure as the argument and returns the control information in the same pointer. Application provides the control ID in the *v4l2_queryctrl* id member in this structure. This control ID is defined in V4L2 header file, for which information is needed. If the control command specified by Id is not supported in current decoder, driver will return an error.

Example:

```
struct v4l2_queryctrl ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ret = ioctl(fd, VIDIOC_QUERYCTRL, &ctrl);
if (ret < 0) {
      perror("VIDIOC_QUERYCTRL \n");
      close(fd);
      return -1;
}
printf("name = %s\n", ctrl.name);
printf("min = %d max = %d step = %d default = %d\n",
ctrl.minimum, ctrl.maximum, ctrl.step, ctrl.default_value);
```

Set Control

This IOCTL is used to set the value for a particular control in current decoder. To set the control value, this IOCTL can also be called when streaming is on.

Ioctl: VIDIOC_S_CTRL

It takes a pointer to instance of *v4l2_control* structure as an input parameter. Application provides control ID and control values in the *v4l2_control* id and value member in this structure. If the control command specified by Id is not supported in the current decoder and if value of the control is out of range, driver returns an error. Otherwise, it sets the control in the registers.

Example:

```
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ctrl.value = 100;
ret = ioctl(fd, VIDIOC_S_CTRL, &ctrl);
if (ret < 0) {
      perror("VIDIOC_S_CTRL\n");
      close(fd);
      return -1;
}
```

Get Control

This IOCTL is used to get the value for a particular control in the current decoder.

Ioctl: VIDIOC_G_CTRL

It takes a pointer to instance of *v4l2_control* structure as an output parameter. Application provides the control ID of id member in this structure. If the control command specified by Id is not supported in the current decoder, driver returns an error. Otherwise, it returns the value of the control in the value member of the *v4l2_control* structure.

Example:

```
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ret = ioctl(fd, VIDIOC_G_CTRL, &ctrl);
if (ret < 0) {
      perror("VIDIOC_G_CTRL\n");
      close(fd);
      return -1;
}
```

```
printf("value = %x\n", ctrl.value);
```

Queue Buffer

This IOCTL is used to enqueue the buffer in buffer queue. This IOCTL will enqueue an empty buffer in the driver buffer queue. This IOCTL is one of necessary IOCTL for streaming IO. If no buffer is enqueued before starting streaming, driver returns an error as there is no buffer available. So at least one buffer must be enqueued before starting streaming. This IOCTL is also used to enqueue empty buffers after streaming is started.

Ioctl: VIDIOC_QBUF

This IOCTL takes a pointer to instance of *v4l2_buffer* structure as an argument. Application has to specify the buffer type *(V4L2_BUF_TYPE_VIDEO_CAPTURE)*, buffer index, and memory type *V4L2_MEMORY_MMAP* or *V4L2_MEMORY_USERPTR* at the time of queuing. For the user pointer buffer exchange mechanism, application also has to provide buffer pointer in the m.userptr member of *v4l2_buffer* structure.

Driver will enqueue buffer in the driver's incoming queue. It will take pointer to instance of v4l2_ buffer structure as an input parameter.

Example:

```
struct v4l2_buffer buf;
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.type = V4L2_MEMORY_MMAP;
buf.index = 0;
ret = ioctl(fd, VIDIOC_QBUF, &buf);
if (ret < 0) {
      perror("VIDIOC_QBUF\n");
      close(fd);
      return -1;
}
```

Dequeue Buffer

This IOCTL is used to dequeue the buffer in the buffer queue. This IOCTL will dequeue the captured buffer from buffer queue of the driver. This IOCTL is one of necessary IOCTL for the streaming IO. This IOCTL can be used only after streaming is started. This IOCTL will block until an empty buffer is available.

Note: The application can dequeue all buffers from the driver - the driver will not hold the last buffer to itself. In this case, the driver will disable the capture operation and the capture operation resumes when a buffer is queued to the driver again.

Ioctl: VIDIOC_DQBUF

It takes a pointer to instance of v4l2_buffer structure as an output parameter. Application has to specify the buffer type *V4L2_BUF_TYPE_VIDEO_CAPTURE* and memory type *V4L2_MEMORY_MMAP* or *V4L2_MEMORY_USERPTR* at the time of dequeueing.

If this IOCTL is called with the file descriptor, with which *VIDIOC_REQBUF* is not performed, driver will return an error. Driver will enqueue buffer, if the buffer queue is not empty.

Example:

```
struct v4l2_buffer buf;
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.type = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_DQBUF, &buf);
if (ret < 0) {
      perror("VIDIOC_DQBUF\n");
```

```
      close(fd);
      return -1;
}
```

Stream On

This IOCTL is used to start video capture functionality.
Ioctl: VIDIOC_STREAMON
If streaming is already started, this IOCTL call returns an error.
Example:

```
v4l2_buf_type buftype = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMON, &buftype);
if (ret < 0) {
      perror("VIDIOC_STREAMON \n");
      close(fd);
      return -1;
}
```

Stream Off

This IOCTL is used to stop video capture functionality.
Ioctl: VIDIOC_STREAMOFF
If streaming is not started, this IOCTL call returns an error.
Example:

```
v4l2_buf_type buftype = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMOFF, &buftype);
if (ret < 0) {
      perror("VIDIOC_STREAMOFF \n");
      close(fd);
      return -1;
}
```

Driver Configuration

Configuration Steps

To enable capture driver support in the kernel, start *Linux Kernel Configuration* tool.

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

- Select *Device Drivers* from the main menu.

```
...
...
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
```

```
[*] Networking support  --->
Device Drivers  --->
...
...
```

- Select *Multimedia support* from the menu.

```
...
...
Sonics Silicon Backplane  --->
Multifunction device drivers  --->
[*] Voltage and Current Regulator Support  --->
<*> Multimedia support  --->
Graphics support  --->
<*> Sound card support  --->
[*] HID Devices  --->
[*] USB support  --->
...
...
```

- Select *Media Controller API* from the menu

```
...
...
--- Multimedia support
     *** Multimedia core support ***
[*]   Media Controller API (EXPERIMENTAL)
<*>   Video For Linux
...
...
```

- Select *Video For Linux* from the menu.

```
...
...
*** Multimedia core support ***
[*]   Media Controller API (EXPERIMENTAL)
<*>   Video For Linux
[*]     Enable Video For Linux API 1 (DEPRECATED)
< >   DVB for Linux
...
...
```

- Select *V4L2 sub-device userspace API* from the menu.

```
...
...
*** Multimedia core support ***
[*]   Media Controller API (EXPERIMENTAL)
<*>   Video For Linux
[*]     Enable Video For Linux API 1 (DEPRECATED)
```

```
[*]     V4L2 sub-device userspace API (EXPERIMENTAL)
< >   DVB for Linux
...
...
```

- Select *Video capture adapters* from the same menu. Press <ENTER> to enter the corresponding sub-menu.

```
...
...
[ ]   Customize analog and hybrid tuner modules to build  --->
[*]   Video capture adapters  --->
[ ]   Radio Adapters  --->
[ ]   DAB adapters
...
...
```

- Select *OMAP3 Camera Support* from the menu.

```
...
...
< >   SR030PC30 VGA camera sensor support
<*>   OMAP 3 Camera support (EXPERIMENTAL)
[ ]     OMAP 3 Camera debug messages
...
...
```

- De-Select *Autoselect pertinent encoders/decoders and other helper chips* from the same menu option. After De-selecting this

option, new option *Encoders/decoders and other helper chips* will drop down.

```
...
--- Video capture adapters
[ ]   Enable advanced debug functionality
[ ]   Enable old-style fixed minor ranges for video devices
[ ]   Autoselect pertinent encoders/decoders and other helper chips
Encoders/decoders and other helper chips  --->
< >   Virtual Video Driver
< >   CPiA Video For Linux
```

- Go inside option *Encoders/decoders and other helper chips*.

```
...
--- Video capture adapters
[ ]   Enable advanced debug functionality
[ ]   Enable old-style fixed minor ranges for video devices
[ ]   Autoselect pertinent encoders/decoders and other helper chips
        Encoders/decoders and other helper chips --->
< >   Virtual Video Driver
< >   CPiA Video For Linux
```

- Select TVP514x Video decoder driver from the menu.

```
...
...
< > Philips SAA7171/3/4 audio/video decoders
< > Philips SAA7191 video decoder
<*> Texas Instruments TVP514x video decoder
< > Texas Instruments TVP5150 video decoder
...
...
```

- Select *Aptina MT9T111 VGA CMOS IMAGE SENSOR* from the menu.

```
...
...
< > Aptina MT9V113 VGA CMOS IMAGE SENSOR
<*> Aptina MT9T111 VGA CMOS IMAGE SENSOR
< > TCM825x camera sensor support
...
...
```

The selection between MT9T111 camera sensor and TVP5146 video decoder is done through media-controller interface dynamically.

Installation

NOTE: Please note that with this release older revision of OMAP3EVM (<Rev-E) is not supported.

Driver built statically

If the OMAP35x Camera driver and TVP514x driver are built statically into the kernel, it is activated during boot-up. There is no special procedure to install the driver.

Driver built as loadable module

Currently modular build is not supported in this release.

```
Sample Applications
```

This chapter describes the sample application provided along with the package. The binary and the source for these sample application can are available in the Examples directory of the Release Package folder.

Introduction

Writing a capture application involves the following steps:

- Opening media device
- Enumerate/Create/Setup the links
- Opening underneath sub device
- Set the format at each pad underneath link
- Opening the capture device.
- Set the parameters of the device.
- Allocate and initialize capture buffer
- Receive video data from the device.
- Close the device.

Hardware Setup

Following are the steps required to run the capture sample application:

- If you are using OMAP3EVM (>=Rev-G) revision board, TVP5146 decoder is present on board. If camera sensor module is being used, then please interface it to port J31.
- In care of Analog input, cConnect a DVD player/camera generating a NTSC video signal to the S-Video or Composite jack of the daughter card or EVM.
- Run the sample application after booting the kernel. Please use *-h* or *--help* for available command line options.

NOTE: Please note that, the software doesn't support older revision of EVM's (<=Rev D).

Applications

Following are the list of capture sample application provided with the release:

- **MMAP Loopback Application (saMmapLoopback.c):**

This sample application using driver allocated buffers to capture video data from any one of the active inputs and displays the video in the active display (LCD / DVI / TV) using display driver. Application is ported to media-controller framework and automatically creates, setup the link; configures the format at each pad of the link and start streaming.

- **USERPTR Loopback Application (saUserPtrLoopback.c):**

This sample application using user allocated buffers to capture video data from any one of the active inputs and displays the video in the active display (LCD / DVI / TV) using display driver. Application is ported to media-controller framework and automatically creates, setup the link; configures the format at each pad of the link and start streaming.

References

[1] http://arago-project.org/git/projects/?p=linux-omap3.git;a=summary

UserGuideAM3517CaptureDriver PSP 04.02.00.07

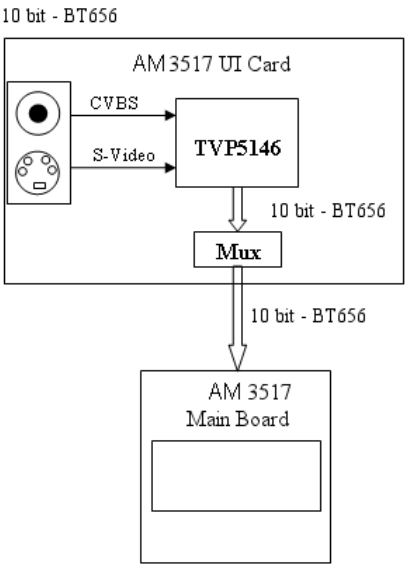
Introduction

The Capture Module is a key component for still-image capture applications. The capture module provides the system interface and the processing capability to connect RAW image-sensor modules and video decoders to the AM3517 device.

The capture module consists of the following interfaces:

- One S-video SD input in BT.656 format.
- One Composite SD input in BT.656 format.

The following figure shows the top view of physical connection and inputs for TVP5146 decoder.



Capture Physical Input Interface

Both these video inputs are connected to one TVP5146 decoder and the application can select between these two inputs using standard V4L2 interface.

NOTE: Only one input can be captured or selected at any given point of time.

The V4L2 Capture driver model is used for capture module. The V4L2 driver model is widely used across many platforms in the Linux community. V4L2 provides good streaming support and support for many buffer formats. It also has its own buffer management mechanism that can be used.

References

- AM3517 VPSS TRM

Author: Texas Instruments, Inc.

- Video for Linux Two API Specification

Author: Michael H Schimek Version: 0.23

Acronyms & Definitions

Capture Driver: Acronyms

Acronym	Definition
API	Application Programming Interface
CCDC	Input interface block of Capture module
DMA	Direct Memory Access
I/O	Input & Output
IOCTL	Input & Output Control
V4L2	Video for Linux specification version 2
YUV	Luminance + 2 Chrominance Difference Signals (Y, Cr, Cb) Color Encoding

Features

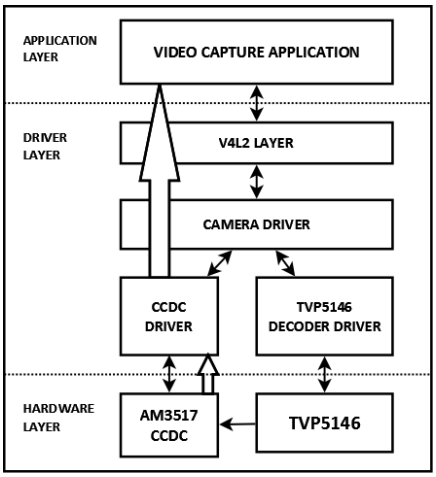
The Capture Driver provides the following features:

- Supports one software channel of capture and a corresponding device node (/dev/video0) is created.
- Supports single I/O instance and multiple control instances.
- Supports buffer access mechanism through memory mapping and user pointers.
- Supports dynamic switching among input interfaces with some necessary restrictions wherever applicable.
- Supports NTSC and PAL standard on Composite and S-Video interfaces.
- Supports 10-bit BT.656 capture in UYVY and YUYV interleaved formats.
- Supports standard V4L2 IOCTLs to get/set various contro parameters like brightness, contrast, saturation, hue and auto gain control.
- In USERPTR mode of operation both malloced and IO mapped buffers are supported.
- Both VPFE Master capture driver and TVP5146 (TVP514x) decoder driver module can be used statically or dynamically (insmod and rmmod supported).

Architecture

Overview

The following figure shows the basic block diagram of capture interface.



Capture Driver Component Overview

The system architecture diagram illustrates the software components that are relevant to the Camera Driver. Some components are outside the scope of this design document. The following is a brief description of each component in the figure.

Camera Applications:

Camera applications refer to any application that accesses the device node that is served by the Camera Driver. These applications are not in the scope of this design. They are here to present the environment in which the Camera Driver is used.

V4L2 Subsystem:

The Linux V4L2 subsystem is used as an infrastructure to support the operation of the Camera Driver. Camera applications mainly use the V4L2 API to access the Camera Driver functionality. A Linux 2.6 V4L2 implementation is used in order to support the standard features that are defined in the V4L2 specification.

Video Buffer Library:

This library comes with V4L2. It provides helper functions to cleanly manage the video buffers through a video buffer queue object.

Camera Driver:

The Camera Driver allows capturing video through an external decoder. It is a V4L2-compliant driver with addition of an AM3517 Capture hardware feature. This driver conforms to the Linux driver model for power management. The camera driver is registered to the V4L2 layer as a master device driver. Any slave decoder driver added to the V4L2 layer will be attached to this driver through the new V4L2 master-slave interface layer. The current implementation supports only one slave device.

Decoder Driver:

The Camera Driver is designed to be AM3517 VPFE module dependent, but platform and board independent. It is the decoder driver that manages the board connectivity. A decoder driver must implement the new V4L2 master-slave interface. It should register to the V4L2 layer as a slave device. Changing a decoder requires implementation of a new decoder driver; it does not require changing the Camera Driver. Each decoder driver exports a set of IOCTLs to the master device through function pointers.

CCDC library:
CCDC is a HW block, where it acts as a data input/entry port. It receives data from the sensor/decoder through parallel interface. The CCDC library exports API to configure CCDC module. It is configured by the master driver based on the sensor/decoder attached and desired output from the camera driver.

Software Design Interfaces

Opening and Closing of driver

The device can be opened using open call from the application, with the device name and mode of operation as parameters. Application should open the driver in blocking mode. In this mode, DQBUF IOCTL will not return until an empty frame is available.

```
/* Open a video capture logical channel in blocking mode */
fd = open("/dev/video0", O_RDWR);
if (fd == -1) {
   perror("failed to open Capture device\n");
   return -1;
}
/* closing of channel */
close (fd);
```

Buffer Management

Capture driver only works with physically contiguous buffers and buffer address should be aligned to 32 bytes boundary. The driver supports both memory usage modes:

- Memory map buffer mode
- User Pointer mode

In Memory map buffer mode, application can request memory from the driver by calling VIDIOC_REQBUFS IOCTL. In user buffer mode, application needs to allocate memory using some other mechanism in user space like mmap from other driver. In driver buffer mode, maximum number of buffers is limited to VIDEO_MAX_FRAME (defined in driver header files) and is limited by the available memory in the kernel.
The main steps that the application must perform for buffer allocation are:

1. Allocating Memory
2. Getting Physical Address
3. Mapping Kernel Space Address to User Space

- **Allocating Memory**

This IOCTL is used to allocate memory for frame buffers. This is the necessary IOCTL for streaming IO. It has to be called for both driver buffer mode and user buffer mode. Using this IOCTL, driver will know whether driver buffer mode or user buffer mode will be used.
Ioctl: VIDIOC_REQBUFS
It takes a pointer to instance of v4l2_requestbuffers structure as an argument. User should specify buffer type as (V4L2_BUF_TYPE_VIDEO_CAPTURE), number of buffers, and memory type (V4L2_MEMORY_MMAP, V4L2_MEMORY_USERPTR) at the time of buffer allocation.

Constraint: This IOCTL can be called only once from the application. This IOCTL is necessary IOCTL.

Example:

```
/* structure to store buffer request parameters */
struct v4l2_requestbuffers reqbuf;
reqbuf.count = numbuffers;
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_REQBUFS, &reqbuf);
if (ret < 0) {
    printf("cannot allocate memory\n");
    close(fd);
    return -1;
}
printf("Number of buffers allocated = %d\n", reqbuf.count);
```

- **Getting Physical Address**

This IOCTL is used to query buffer information like buffer size and buffer physical address. This physical address is used in mmapping the buffers. This IOCTL is necessary for driver buffer mode as it provides the physical address of buffers, which are used to mmap system call the buffers.
Ioctl: VIDIOC_QUERYBUF
It takes a pointer to instance of v4l2_buffer structure as an argument. User has to specify buffer type as (V4L2_BUF_TYPE_VIDEO_CAPTURE), buffer index, and memory type (V4L2_MEMORY_MMAP) at the time of querying.
Example:

```
/* allocate buffer by VIDIOC_REQBUFS */
/* structure to query the physical address of allocated buffer */
struct v4l2_buffer buffer;
buffer.index = 0; /* buffer index for quering -0 */
buffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buffer.memory = V4L2_MEMORY_MMAP;
if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < -1) {
    printf("buffer query error.\n");
    close(fd);
    exit(-1);
}
The buffer.m.offset will contain the physical address returned from
driver.
```

- **Mapping Kernel Space Address to User Space**

Mapping the kernel buffer to the user space can be done via mmap. This is only required for MMAP buffer mode. User can pass buffer size and physical address of buffer for getting the user space address.
Example:

```
/* allocate buffer by VIDIOC_REQBUFS */
/* query the buffer using VIDIOC_QUERYBUF */
/* addr hold the user space address */
int addr;
addr = mmap(NULL, buffer.size,PROT_READ | PROT_WRITE, MAP_SHARED, fd,
buffer.m.offset);
/* buffer.m.offset is same as returned from VIDIOC_QUERYBUF */
```

Query Capabilities

This IOCTL is used to verify kernel devices compatibility with V4L2 specification and to obtain information about individual hardware capabilities. In this case, it will return capabilities provided by capture driver and current decoder driver.

Ioctl: VIDIOC_QUERYCAP

Capabilities can be video capture (V4L2_CAP_VIDEO_CAPTURE) and streaming (V4L2_CAP_STREAMING). It takes pointer to v4l2_capability structure as an argument. Capabilities can be accessed by capabilities field in the v4l2_capability structure.

Example:

```
struct v4l2_capability capability;
ret = ioctl(fd, VIDIOC_QUERYCAP, &capability);
if (ret < 0) {
    printf("Cannot do QUERYCAP\n");
    return -1;
}
if (capability.capabilities & V4L2_CAP_VIDEO_CAPTURE) {
    printf("Capture capability is supported\n");
}
if (capability.capabilities & V4L2_CAP_STREAMING) {
    printf("Streaming is supported\n");
}
```

Input Enumeration

This IOCTL is used to enumerate the information of available inputs (analog interface). It includes information like name of input type and supported standards for that input type.

Ioctl: VIDIOC_ENUMINPUT

It takes pointer to v4l2_input structure. Application provides the index number for which it requires the information, in index member of v4l2_input structure.

Index with value zero indicates first input type of the decoder. It returns combination of the standards supported on this input in the std member of v4l2_input structure.

Example:

```
struct v4l2_input input;
i = 0;
while(1) {
    input.index = i;
    ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
    if (ret < 0)
        break;
    printf("name = %s\n", input.name);
    i++;
}
```

Set Input

This IOCTL is used to set input type (analog interface type).
Ioctl: VIDIOC_S_INPUT
This IOCTL takes pointer to integer containing index of the input which has to be set. Application will provide the index number as an argument.

```
0 - Composite input,
       1 - S-Video input.
```

Example:

```
int index = 1; /*To set S-Video input*/
struct v4l2_input input;
ret = ioctl(fd, VIDIOC_S_INPUT, &index);
if (ret < 0) {
    perror("VIDIOC_S_INPUT\n");
    close(fd);
    return -1;
}
input.index = index;
ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
if (ret < 0) {
perror("VIDIOC_ENUMINPUT\n");
    close(fd);
    return -1;
}
printf("name of the input = %s\n",input.name);
```

Get Input

This IOCTL is used to get the current input type (analog interface type).
Ioctl: VIDIOC_G_INPUT
This IOCTL takes pointer to integer using which the detected inputs will be returned. It will return the software managed input detected during open system call. Application will provide the index number as an output argument.
Example:

```
int input;
struct v4l2_input input;
ret = ioctl(fd, VIDIOC_G_INPUT, &input);
if (ret < 0) {
    perror("VIDIOC_G_INPUT\n");
    close(fd);
    return -1;
}
input.index = index;
ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
if (ret < 0) {
    perror("VIDIOC_ENUMINPUT\n");
    close(fd);
    return -1;
```

```
}
printf("name of the input = %s\n", input.name);
```

Standard Enumeration

This IOCTL is used to enumerate the information regarding video standards. This IOCTL is used to enumerate all the standards supported by the registered decoder.

Ioctl: VIDIOC_ENUMSTD

This IOCTL takes a pointer to v4l2_standard structure. Application provides the index of the standard to be enumerated in the index member of this structure. It provides information like standard name, standard ID defined at V4L2 header files (few new standards are included in the respective decoder header files, which were not available in standard V4L2 header files), and numerator and denominator values for frame period and frame lines.

It takes index as an argument as a part of v4l2_standard structure. Index with value zero provides information for the first standard among all the standards of all the registered decoders. If the index value exceeds the number of supported standards, it returns an error.

Example:

```
struct v4l2_standard standard;
i = 0;
while(1) {
    standard.index = i;
    ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
    if (ret < 0)
        break;
    printf("name = %s\n", std.name);
    printf("framelines = %d\n", std.framelines);
    printf("numerator = %d\n",
           std.frameperiod.numerator);
    printf("denominator = %d\n",
           std.frameperiod.denominator);
    i++;
}
```

Standard Detection

This IOCTL is used to detect the current video standard set in the current decoder.

Ioctl: VIDIOC_QUERYSTD

It takes a pointer to v4l2_std_id instance as an output argument. Driver will call the current decoder's function internally (which has been initialized) to detect the current standard set in hardware. Support of this IOCTL depends on decoder device, whether it can detect a standard or not.

Note: This IOCTL should be called by the application so that the camera driver can configure Capture module properly with the detected decoder standard.

Standard IDs are defined in the V4L2 header files

Example:

```
struct v4l2_std_id std;
struct v4l2_standard standard;
ret = ioctl(fd, VIDIOC_QUERYSTD, &std);
if (ret < 0) {
    perror("VIDIOC_QUERYSTD\n");
```

```
    close(fd);
    return -1;
}
while(1) {
    standard.index = i;
    ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
    if (ret < 0)
        break;
    if (standard.std & std) {
        printf("%s standard detected\n",
               standard.name);
        break;
    }
    i++;
}
```

Set Standard

This IOCTL is used to set the standard in the decoder.
Ioctl: VIDIOC_S_STD
It takes a pointer to v4l2_std_id instance as an input argument. If the standard is not supported by the decoder, the driver will return an error Standard IDs are defined in the V4L2 header files (few new standards are included in respective decoder header files, which were not available in standard V4L2 header files).
Note: Application need not call this IOCTL as the decoder can auto detect the current standard. This is required only when the application needs to set a particular standard. In this case, the decoder driver auto detect function is disabled. Auto detect can be enabled again only by closing and re-opening the driver.
Example:

```
struct v4l2_std_id std = V4L2_STD_NTSC;
ret = ioctl(fd, VIDIOC_S_STD, &std);
if (ret < 0) {
    perror("S_STD\n");
    close(fd);
    return -1;
}
while(1) {
    standard.index = i;
    ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
    if (ret < 0)
        break;
    if (standard.std & std) {
        printf("%s standard is selected\n");
        break;
    }
    i++;
}
```

Get Standard

This IOCTL is used to get the current standard in the current decoder.
Ioctl: VIDIOC_G_STD
It takes a pointer to v4l2_std_id instance as an output argument. Standard IDs are defined in the V4L2 header files
Example:

```
struct v4l2_std_id std;
ret = ioctl(fd, VIDIOC_G_STD, &std);
if (ret < 0) {
    perror("G_STD\n");
    close(fd);
    return -1;
}
while(1) {
    standard.index = i;
    ret = ioctl(fd, VIDIOC_ENUMSTD, &standard);
    if (ret < 0)
        break;
    if (standard.std & std) {
         printf("%s standard is selected\n");
         break;
    }
    i++;
}
```

Format Enumeration

This IOCTL is used to enumerate the information of pixel formats. The driver supports only two pixel form at -8-bit UYVY interleaved and 8-bit YUYV interleaved.
Ioctl: VIDIOC_ENUM_FMT
It takes a pointer to instance of v4l2_fmtdesc structure as an output parameter. Application must provide the buffer type in the type argument of v4l2_fmtdesc structure as V4L2_BUF_TYPE_VIDEO_CAPTURE and index member of this structure as zero.
Example:

```
struct v4l2_fmtdesc fmt;
i = 0;
while(1) {
    fmt.index = i;
    ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmt);
    if (ret < 0)
        break;
    printf("description = %s\n",fmt.description);
    if (fmt.type == V4L2_BUF_TYPE_VIDEO_CAPTURE)
        printf("Video capture type\n");
    if (fmt.pixelformat == V4L2_PIX_FMT_YUYV)
        printf("V4L2_PIX_FMT_YUYV\n");
    i++;
}
```

Set Format

This IOCTL is used to set the format parameters. The format parameters are line offset, storage format, pixel format, and so on. This IOCTL is one of the necessary IOCTL. If it is not set, it uses the following default values:

- Default storage format - V4L2_FIELD_INTERLACED

This IOCTL expects proper width and height members of the v4l2_format structure from application as per the standard selected. Please note that, V4L2_FIELD_INTERLACED is the only storage format supported.
The application can decide the buffer pixel format using pixelformat member of this IOCTL. The current driver supports - 8-bit UYVY interleaved and 8-bit YUYV interleaved formats. The desired pitch of the buffer can be set by using the bytesperline member. The pitch should be at least one line size in bytes. When changing the pitch, the application should also modify the sizeimage member accordingly - sizeimage should be at least pitch * image height. The driver allocates buffer of size sizeimage member of the v4l2_format structure passed through this IOCTL for both mmap buffer and user pointer mode. Driver validates the provided buffer size along with the other members and uses this buffer size for calculating offsets for storing video data.
This IOCTL is a necessary IOCTL for the user buffer mode because driver will know the buffer size for user buffer mode. If it not called for the user buffer mode, driver assumes the default buffer size and calculates offsets accordingly.
Ioctl: VIDIOC_S_FMT
It will take pointer to instance of v4l2_format structure as an input parameter. If the type member is V4L2_BUF_TYPE_VIDEO_CAPTURE, it checks pixel format, pitch value, and image size. It returns an error, if the parameters are invalid.
Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
/* for  NTSC standard */
fmt.fmt.pix.width = 720;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    return -1;
}
```

Get Format

This IOCTL is used to get the current format parameters.
Ioctl: VIDIOC_G_FMT
It takes a pointer to instance of v4l2_format structure as an input parameter. Driver provides format parameters in the structure pointer passed as an argument. v4l2_format structure contains parameters like pixel format, image size, bytes per line, and field type.
For type V4L2_BUF_TYPE_VIDEO_CAPTURE, the v4l2_pix_format structure of fmt union is filled.

Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
```

```
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_G_FMT\n");
    close(fd);
    return -1;
}
if (fmt.fmt.pix.pixelformat == V4L2_PIX_FMT_YUYV)
    printf("8-bit UYVY pixel format\n");
printf("Size of the buffer = %d\n", fmt.fmt.pix.sizeimage);
printf("Line offset = %d\n", fmt.fmt.pix.bytesperline);
if (fmt.fmt.pix.field == V4L2_FIELD_INTERLACED)
    printf("Storate format is interlaced frame format");
```

Try Format

This IOCTL is used to validate the format parameters provided by the application. It checks parameters and returns the correct parameter, if any parameter is incorrect. It returns error only if the parameters passed are ambiguous.
Ioctl: VIDIOC_TRY_FMT
It takes a pointer to instance of v4l2_format structure as an input/output parameter. If the type member is V4L2_BUF_TYPE_VIDEO_CAPTURE, it checks pixel format, pitch value, and image size. It returns errors to the application, if the parameters are invalid.
Example:

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
fmt.fmt.pix.sizeimage = size;
fmt.fmt.pix.bytesperline = pitch;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
ret = ioctl(fd, VIDIOC_TRY_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_TRY_FMT\n");
    close(fd);
    return -1;
}
```

Query Control

This IOCTL is used to get the information of controls that is, brightness, contrast, and so on supported by the current decoder.
Ioctl: VIDIOC_QUERYCTRL
This IOCTL takes a pointer to the instance of v4l2_queryctrl structure as the argument and returns the control information in the same pointer. Application provides the control ID in the v4l2_queryctrl id member in this structure. This control ID is defined in V4L2 header file, for which information is needed.
If the control command specified by Id is not supported in current decoder, driver will return an error.
Example:

```
struct v4l2_queryctrl ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ret = ioctl(fd, VIDIOC_QUERYCTRL, &ctrl);
```

```
if (ret < 0) {
    perror("VIDIOC_QUERYCTRL \n");
    close(fd);
    return -1;
}
printf("name = %s\n", ctrl.name);
printf("min = %d max = %d step = %d default = %d\n",
       ctrl.minimum, ctrl.maximum, ctrl.step, ctrl.default_value);
```

Set Control

This IOCTL is used to set the value for a particular control in current decoder. To set the control value, this IOCTL can also be called when streaming is on.

Ioctl: VIDIOC_S_CTRL

It takes a pointer to instance of v4l2_control structure as an input parameter. Application provides control ID and control values in the v4l2_control id and value member in this structure. If the control command specified by Id is not supported in the current decoder and if value of the control is out of range, driver returns an error. Otherwise, it sets the control in the registers.

Example:

```
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ctrl.value = 100;
ret = ioctl(fd, VIDIOC_S_CTRL, &ctrl);
if (ret < 0) {
    perror("VIDIOC_S_CTRL\n");
    close(fd);
    return -1;
}
```

Get Control

This IOCTL is used to get the value for a particular control in the current decoder.

Ioctl: VIDIOC_G_CTRL

It takes a pointer to instance of v4l2_control structure as an output parameter. Application provides the control ID of id member in this structure. If the control command specified by Id is not supported in the current decoder, driver returns an error. Otherwise, it returns the value of the control in the value member of the v4l2_control structure.

Example:

```
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_CONTRAST;
ret = ioctl(fd, VIDIOC_G_CTRL, &ctrl);
if (ret < 0) {
    perror("VIDIOC_G_CTRL\n");
    close(fd);
    return -1;
}
printf("value = %x\n", ctrl.value);
```

Queue Buffer

This IOCTL is used to enqueue the buffer in buffer queue. This IOCTL will enqueue an empty buffer in the driver buffer queue. This IOCTL is one of necessary IOCTL for streaming IO. If no buffer is enqueued before starting streaming, driver returns an error as there is no buffer available. So at least one buffer must be enqueued before starting streaming. This IOCTL is also used to enqueue empty buffers after streaming is started.

Ioctl: VIDIOC_QBUF

This IOCTL takes a pointer to instance of v4l2_buffer structure as an argument. Application has to specify the buffer type (V4L2_BUF_TYPE_VIDEO_CAPTURE), buffer index, and memory type (V4L2_MEMORY_MMAP or V4L2_MEMORY_USERPTR) at the time of queuing.

For the user pointer buffer exchange mechanism, application also has to provide buffer pointer in the m.userptr member of v4l2_buffer structure. Driver will enqueue buffer in the driver's incoming queue. It will take pointer to instance of v4l2_ buffer structure as an input parameter.

Example:

```
struct v4l2_buffer buf;
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.type = V4L2_MEMORY_MMAP;
buf.index = 0;
ret = ioctl(fd, VIDIOC_QBUF, &buf);
if (ret < 0) {
    perror("VIDIOC_QBUF\n");
    close(fd);
    return -1;
}
```

Dequeue Buffer

This IOCTL is used to dequeue the buffer in the buffer queue. This IOCTL will dequeue the captured buffer from buffer queue of the driver. This IOCTL is one of necessary IOCTL for the streaming IO. This IOCTL can be used only after streaming is started. This IOCTL will block until an empty buffer is available.

Note: The application can dequeue all buffers from the driver - the driver will not hold the last buffer to itself. In this case, the driver will disable the capture operation and the capture operation resumes when a buffer is queued to the driver again.

Ioctl: VIDIOC_DQBUF

It takes a pointer to instance of v4l2_buffer structure as an output parameter. Application has to specify the buffer type (V4L2_BUF_TYPE_VIDEO_CAPTURE) and memory type (V4L2_MEMORY_MMAP or V4L2_MEMORY_USERPTR) at the time of dequeueing.

If this IOCTL is called with the file descriptor, with which VIDIOC_REQBUF is not performed, driver will return an error. Driver will enqueue buffer, if the buffer queue is not empty.

Example:

```
struct v4l2_buffer buf;
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.type = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_DQBUF, &buf);
if (ret < 0) {
    perror("VIDIOC_DQBUF\n");
    close(fd);
    return -1;
```

```
}
```

Stream On

This IOCTL is used to start video capture functionality.
Ioctl: VIDIOC_STREAMON
If streaming is already started, this IOCTL call returns an error.
Example:

```
v4l2_buf_type buftype = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMON, &buftype);
if (ret < 0) {
    perror("VIDIOC_STREAMON \n");
    close(fd);
    return -1;
}
```

Stream Off

This IOCTL is used to stop video capture functionality.
Ioctl: VIDIOC_STREAMOFF
If streaming is not started, this IOCTL call returns an error.
Example:

```
struct v4l2_buf_type buftype = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMOFF, &buftype);
if (ret < 0) {
    perror("VIDIOC_STREAMOFF \n");
    close(fd);
    return -1;
}
```

Driver Configuration

To enable V4L2 capture driver support in the kernel:

Start Linux Kernel Configuration tool.

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

- Select Device Drivers from the main menu.

```
...
...
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
```

```
...
```

- Select Multimedia support from the menu.

```
...
...
Sonics Silicon Backplane  --->
Multifunction device drivers  --->
<*> Multimedia support  --->
Graphics support  --->
<*> Sound card support  --->
[*] HID Devices  --->
[*] USB support  --->
...
...
```

- Select Video For Linux from the menu.

```
...
...
*** Multimedia core support ***
<*>   Video For Linux
[*]     Enable Video For Linux API 1 (DEPRECATED)
< >   DVB for Linux
...
...
```

- Select Video capture adapters from the same menu.

```
...
...
[ ]   Customize analog and hybrid tuner modules to build  --->
[*]   Video capture adapters  --->
[*]   Radio Adapters  --->
[ ]   DAB adapters
...
...
```

- Select "VPFE Video Capture Driver" from the menu. After selecting this option sub-menu will appear and automatically will select "DM6446 CCDC HW module" and "VPSS System module driver"

```
...
...
      Encoders/decoders and other helper chips  --->
-*-   VPSS System module driver
<*>   VPFE Video Capture Driver
<*>     DM6446 CCDC HW module
<*>   OMAP2/OMAP3 V4L2-Display driver
< >   CPiA2 Video For Linux
...
...
```

- Selection of TVP5146 Video Decoder driver -

De-select option Autoselect pertinent encoders/decoders and other helper chips and go inside Encoders/decoders and other helper chips

```
--- Video capture adapters
...
...
[ ]   Autoselect pertinent encoders/decoders and other helper chips
    Encoders/decoders and other helper chips  --->
< >   Virtual Video Driver
...
...
```

- Select TVP5146 Video Decoder driver from the menu.

```
*** Audio decoders ***
...
...
< >   Philips SAA7191 video decoder
<*>   Texas Instruments TVP514x video decoder
< >   Texas Instruments TVP5150 video decoder
...
...
```

Sample Applications

This chapter describes the sample application provided along with the package. The binary and the source for these sample application can are available in the Examples directory of the Release Package folder.

Introduction

Writing a capture application involves the following steps:

- Opening the capture device.
- Set the parameters of the device.
- Allocate and initialize capture buffer
- Receive video data from the device.
- Close the device.

Hardware Setup

Following are the steps required to run the capture sample application:

- Connect the AM3517 User interface card module containing the TVP5146 deocoder to the AM3517 main board.
- Make sure that switch settings on UI Card, S11.1 and S11.2 are turned ON.
- Connect a DVD player/camera generating a NTSC video signal to the S-Video or Composite jack of the daughter card.
- Run the sample application after booting the kernel.

Applications Source

Please refer to the examples.tar.gz file in Release package for the provided application sources.

UserGuideUsbDriver PSP 04 02 00 07

USB Driver

Introduction

TI OMAP35x, AM3517, AM/DM37x have a host cum gadget controller MUSB OTG, an EHCI and its companion OHCI controller. There are three USB ports which are to be controlled by either EHCI or OHCI controller individually.

Ports has to be connected to high speed USB phy to act as an EHCI port and similarly it should be connected to full speed USB phy to act as an OHCI port. We don't need these USB phys in TLL mode of operations.

In ES2.0/2.1 silicon all the three port can either be configured in PHY mode or in TLL mode at a time. This limitation got resolved in ES3.0/3.1 silicon where PHY/TLL mode selection can be done on per port basis.

Supported features

The salient features of the MUSB OTG controller are:

Common feature

- High/full speed operation as USB peripheral.
- High/full/low speed operation as Host controller.
- The host controller for a multi-point USB system (when connected via hub).
- USB On-The-Go compliant USB controller.
- 15 Transmit and 15 Receive Endpoints other than the mandatory Control Endpoint 0.
- Double buffering FIFO.
- Support for Bulk split and Bulk combine
- Support for high bandwidth Isochronous transfer

OMAP35x, AM/DM37x

- 16 Kilobytes of Endpoint FIFO RAM for USB packet buffering.
- Dual Mode HS DMA controller with 8 channels.

AM3517

- 32 Kilobytes of Endpoint FIFO RAM for USB packet buffering.
- CPPI4.1 DMA controller with 15 Rx and 15 Tx channels.

References

- OMAP35x, AM/DM37x Technical Reference Manual
- AM3517 Technical Reference Manual

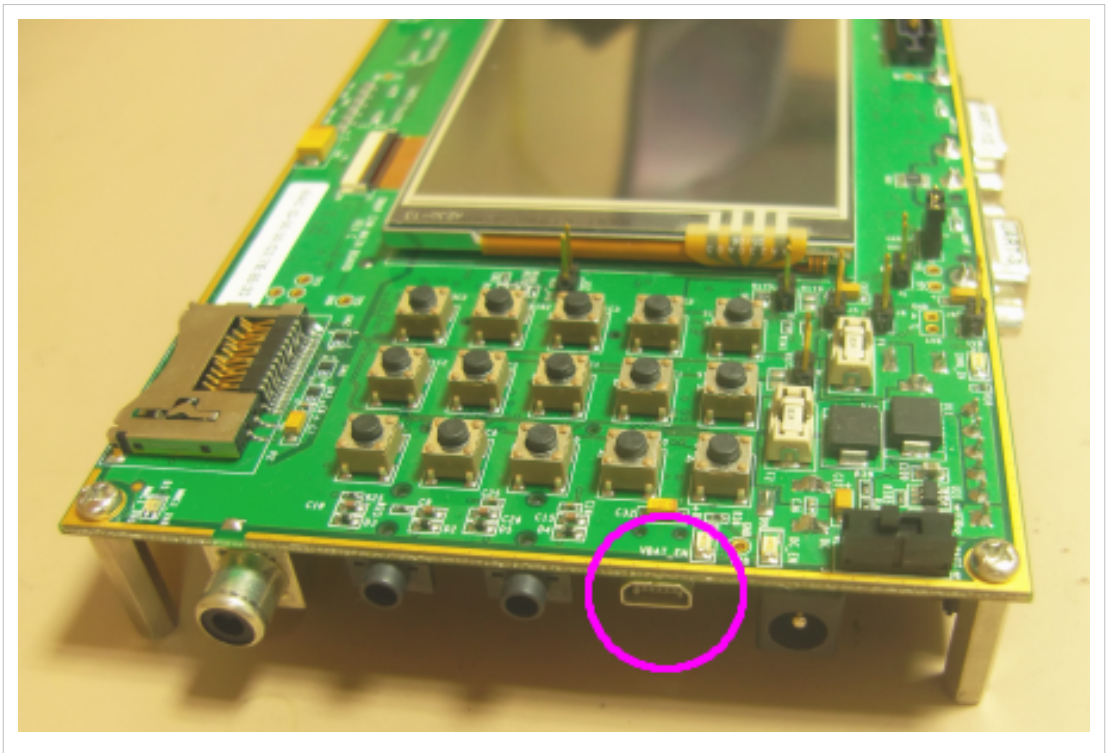
Hardware Overview

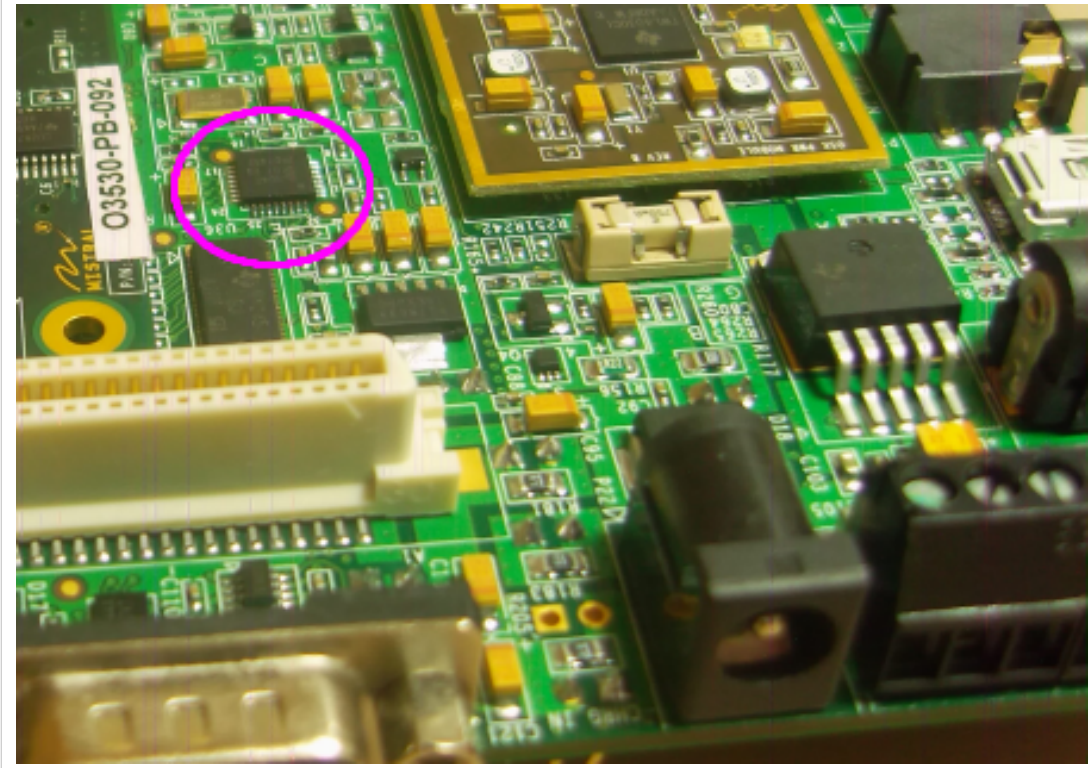
OMAP35x, AM/DM37x

The OMAP35x MUSB OTG controller sits on the L3 and L4 interconnect.It can be an L3 master while performing DMA transfers and an L4 target when host CPU/DMA engine is the master.

OMAP3EVM-1 (<=Rev-E) has an OTG compliant USB PHY from NXP (ISP 1504)and OMAP3EVM-2 (>=Rev-E) has NXP USB PHY ISP1507.

HS USB port2 is connected to SMSC USB83320 high speed PHY on Mistral/Multimedia daughter card (MMDC) attached to OMAP3EVM-1 (<=Rev-E) whereas on OMAP3EVM-2 (>=Rev-E) it is connected to SMSC USB3320 PHY. Port1 and Port3 are not available either on MMDC attached to OMAP3EVM-1 (<=Rev-E) or OMAP3EVM-2 (>=Rev-E).





AM3517

AM3517 HS USB port1 is connected to SMSC3320 high speed PHY and available on baseboard while port2 is connected to same SMSC PHY but is available on UI card. Port2 will be functional only if LCD is not in use.This limitation is due to shared pinmux.

Features

The MUSB OTG and EHCI driver supports a significant subset of all the features provided by the USB controller. The following section discusses the features supported in this release.

Supported feature on MUSB OTG port

- Can be built in-kernel (part of vmlinux) as well as a driver module (musb_hdrc.ko).
- Audio Class in Host mode.
- Video Class in Host mode.
- Mass Storage Class in Host mode.
- Mass Storage Class in Gadget mode.
- Hub Class in Host mode.
- Human Interface Devices (HID) in Host mode.
- Communication Device Class (CDC) in Gadget mode.
- Remote Network Driver Interface Specification (RNDIS) Gadget support.
- OTG support which includes support for Host Negotiation Protocol (HNP) and Session Request Protocol (SRP).

Supported feature on EHCI port

- Can be built in-kernel (part of vmlinux) as well as a driver module (ehci_hcd.ko).
- Human Interface Devices (HID) via a high speed hub.
- Mass Storage Class.
- Audio Class.
- Video Class.
- Hub Class.

Not supported

- OHCI is not supported as OHCI port is not available either on OMAP35x, AM/DM37x or AM3517 EVMs

Driver configuration

The MUSB OTG controller is used in Host and Gadget modes while EHCI is used only in Host mode.The following section shows the configuration options for USB and its associated class drivers.

USB phy selection for MUSB OTG port

Please select NOP USB transceiver for MUSB support.

```
Device Drivers --->
USB support --->
*** OTG and related infrastructure ***
[ ] GPIO based peripheral-only VBUS sensing 'transceiver'
[ ] Philips ISP1301 with OMAP OTG
[ ] TWL4030 USB Transceiver Driver
[*] NOP USB Transceiver Driver
```

USB controller in host mode

MUSB OTG Host Configuration

```
Device Drivers --->
USB support --->
<*> Support for Host-side USB
    *** Miscellaneous USB options ***
[*] USB device filesystem
[*] USB device class-devices (DEPRECATED)
    *** USB Host Controller Drivers ***
<*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
          *** Platform Glue Layer ***
<>     TUSB6010
<*>     OMAP2430 and onwards
<>     AM35x
          Driver Mode (USB Host) --->
[ ] Disable DMA (always use PIO)
[*] Use System DMA for Mentor DMA workaround
[*] Enable debugging messages
```

- Please enable "Use System DMA for Mentor DMA workaround" on OMAP35x/AM37x as a workaround to Mentor DMA issues.

EHCI Configuration

Port-2 will automatically be selected for OMAP35x, AM/DM37x EVM and would be configured in PHY mode.Port-1 will automatically be selected for AM3517 EVM while port-2 will be enabled only if LCD is not configured.

```
Device Drivers --->
USB support --->
<*> Support for Host-side USB
    *** Miscellaneous USB options ***
[*] USB device filesystem
[*] USB device class-devices (DEPRECATED)
<*> EHCI HCD (USB2.0) Support
[ ] Root hub transaction translators
[*] Improved Transaction Translator scheduling
    *** USB Host Controller Drivers ***
```

MUSB OTG controller in gadget mode

Configuration

Please do not disable support for host side usb as this will disable EHCI host interface also. Gadget option in driver mode will appear only when gadget support is also selected. Please enable gadget support as given below.

```
Device Drivers --->
USB support --->
  <*> USB Gadget Support --->
      [ ] Debugging messages (DEVELOPMENT)
      [ ] Debugging information files (DEVELOPMENT)
      [ ]   Debugging information files in debugfs (DEVELOPMENT)
      (2) Maximum VBUS power usage (2-500mA) NEW
  USB Peripheral Controller (Inventra HDRC Peripheral(TI,ADI, ...))
--->
  <M> USB Gadget Drivers
  <M> File-backed Storage Gadget
```

Please make sure that Inventra HDRC is selected as USB peripheral controller which will appear only when "USB Peripheral (gadget stack)" is selected in driver mode as shown below so after selecting Gadget Support go back to driver mode option to select "USB Peripheral (gadget stack)" and then come back again to select Inventra HDRC as USB peripheral controller.

```
Device Drivers --->
USB support --->
<*> Support for Host-side USB
    *** Miscellaneous USB options ***
[*] USB device filesystem
[*] USB device class-devices (DEPRECATED)
    *** USB Host Controller Drivers ***
```

```
<*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
          **** Platform Glue Layer ***
<>     TUSB6010
<*>     OMAP2430 and onwards
<>     AM35x
          Driver Mode (USB Peripheral (gadget stack)) --->
[ ] Disable DMA (always use PIO)
[*] Use System DMA for Mentor DMA workaround
[*] Enable debugging messages
```

MUSB OTG controller in OTG mode

OTG Configuration

Both Host and Gadget driver should be selected for OTG support.If gadget driver is build as module then the host side module will be initialized only after gadget module is inserted after bootup.

If "Rely on targeted peripheral list" is also selected then make sure to update `drivers/usb/core/otg_whitelist.h` with the desired supported device class identification ids.

OTG option in driver mode will appear only when gadget support is also selected.Please enable gadget support as given below.

```
  Device Drivers --->
  USB support --->
   <*> USB Gadget Support --->
       [ ] Debugging messages (DEVELOPMENT)
       [ ] Debugging information files (DEVELOPMENT)
       [ ]   Debugging information files in debugfs (DEVELOPMENT)
       (2) Maximum VBUS power usage (2-500mA) NEW
   USB Peripheral Controller (Inventra HDRC Peripheral(TI,ADI, ...))
--->
   <M> USB Gadget Drivers
   <M> File-backed Storage Gadget
```

Please make sure that Inventra HDRC is selected as USB peripheral controller which will appear only when OTG is selected as below.

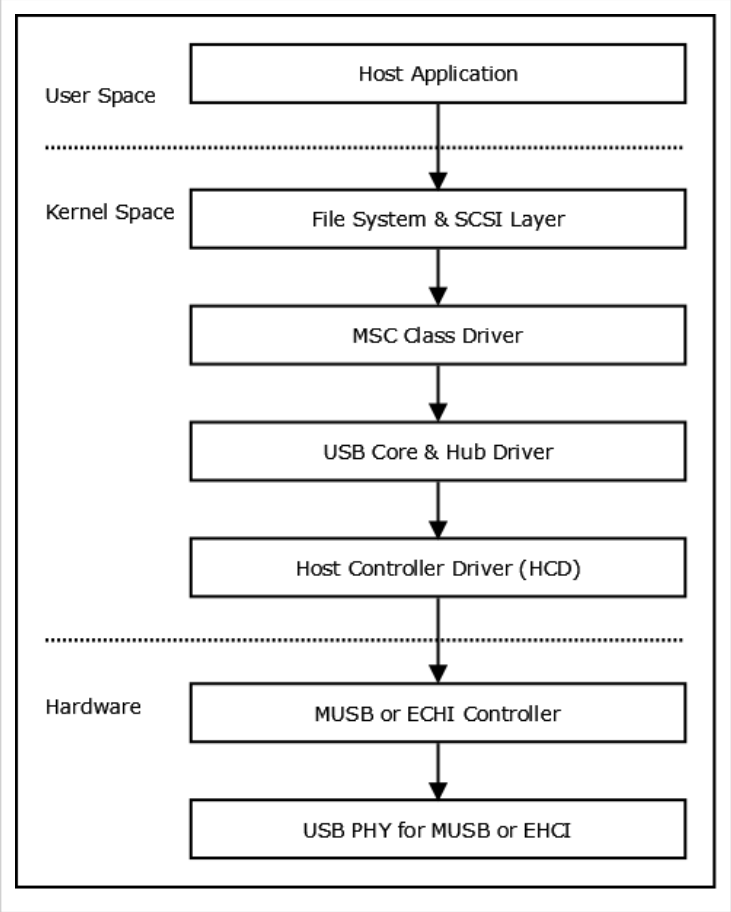
```
Device Drivers --->
USB support --->
<*> Support for Host-side USB
    *** Miscellaneous USB options ***
[*] USB device filesystem
[*] USB device class-devices (DEPRECATED)
    *** USB Host Controller Drivers ***
<*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
          **** Platform Glue Layer ***
<>     TUSB6010
<*>     OMAP2430 and onwards
<>     AM35x
          Driver Mode (Both Host and peripheral : USB OTG (On The Go)
```

```
Device) --->
 [ ] Disable DMA (always use PIO)
 [*] Use System DMA for Mentor DMA workaround
 [*] Enable debugging messages
```

Host mode applications

Mass Storage Driver

This figure illustrates the stack diagram of the system with USB Mass Storage class.



Configuration

```
    Device Drivers --->
    SCSI device support --->
     <*> SCSI device support
     [*] legacy /proc/scsi/support
     --- SCSI support type (disk, tape, CD-ROM)
     <*> SCSI disk support
    USB support --->
     <*> Support for Host-side USB
     *** Miscellaneous USB options ***
     [*] USB device filesystem
     [*] USB device class-devices (DEPRECATED)
     *** USB Host Controller Drivers ***
```

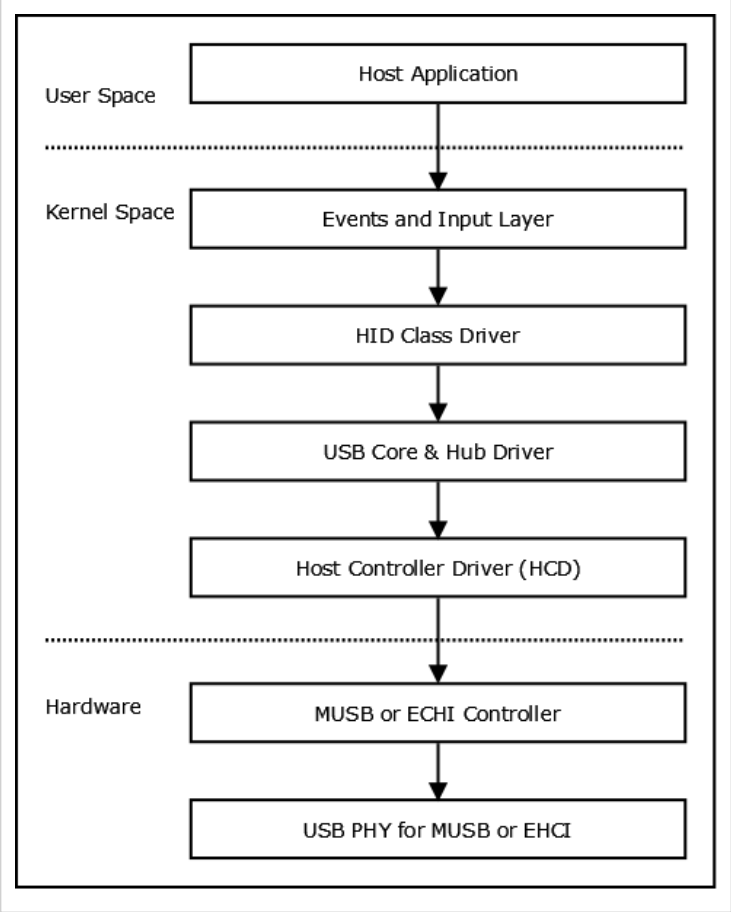
```
    <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
           **** Platform Glue Layer ***
    <>     TUSB6010
    <*>     OMAP2430 and onwards
    <>     AM35x
           Driver Mode (Both Host and peripheral : USB OTG (On The Go)
Device) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
      --- USB Device Class drivers
      <*> USB Mass Storage support
```

Device nodes

The SCSI sub system creates /dev/sd* devices with help of mdev.

USB HID Class

USB Mouse and Keyboards that conform to the USB HID specifications are supported.



Configuration

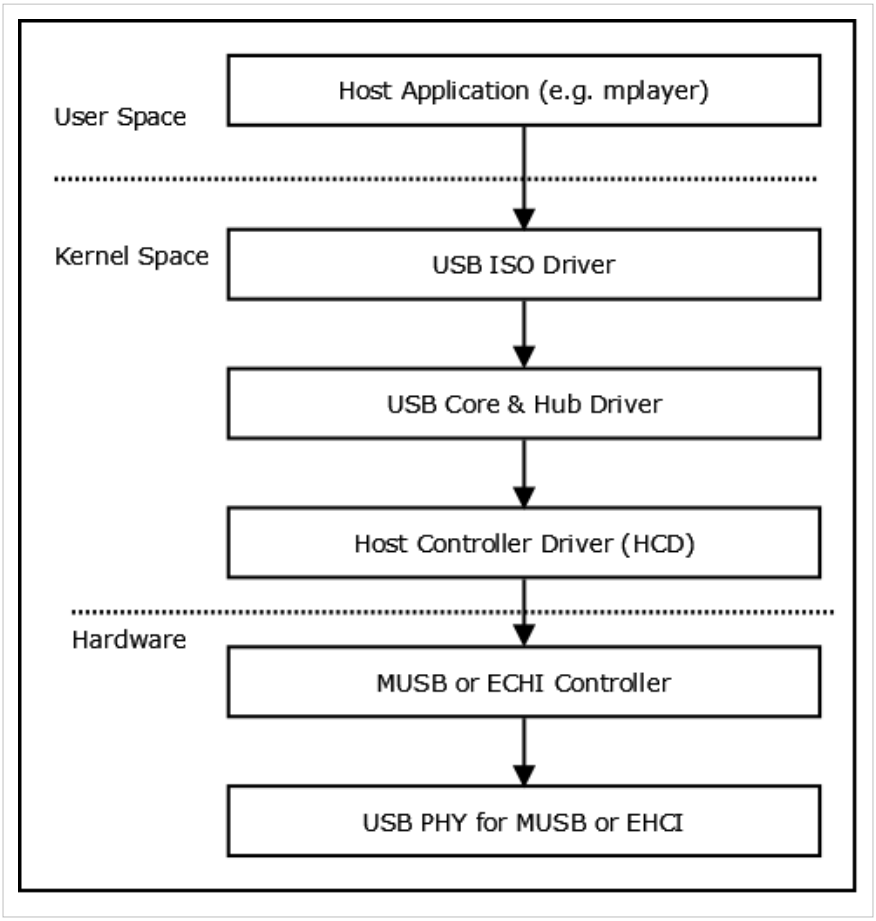
```
    Device Drivers --->
    USB support --->
     <*> Support for Host-side USB
     *** Miscellaneous USB options ***
     [*] USB device filesystem
     [*] USB device class-devices (DEPRECATED)
     *** USB Host Controller Drivers ***
    <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
           **** Platform Glue Layer ***
    <>     TUSB6010
    <*>     OMAP2430 and onwards
    <>     AM35x
           Driver Mode (Both Host and peripheral : USB OTG (On The Go)
Device) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
    HID Devices --->
     <*> Generic HID Support
              *** USB Input Devices ***
     <*> USB Human Interface Device(full HID) support
```

Device nodes

The event sub system creates /dev/input/event* devices with the help of mdev.

USB Audio

The image below shows the USB stack architecture with USB Audio/Video class.



Configuration

```
Device Drivers --->
Sound --->
 <*> Sound card support
   Advanced Linux Sound Architecture --->
   <*> Advanced Linux Sound Architecture
   [*] Dynamic device file minor number
   [*] Support old ALSA API
      USB devices --->
      <*> USB Audio/MIDI driver
USB support --->
   <*> Support for Host-side USB
   *** Miscellaneous USB options ***
   [*] USB device filesystem
   [*] USB device class-devices (DEPRECATED)
  *** USB Host Controller Drivers ***
 <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
```

```
            **** Platform Glue Layer ***
    <>      TUSB6010
    <*>      OMAP2430 and onwards
    <>      AM35x
            Driver Mode (Both Host and peripheral : USB OTG (On The Go)
Device) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
```

Resources

For testing USB Audio support we need any ALSA compliant audio player/capture application. Kindly read the Audio driver section to get more inputs on this.

USB Video

Configuration

```
  Device Drivers --->
    Multimedia devices --->
          *** Multimedia core support ***
      <*> Video for Linux
      [*] Enable Video for Linux API 1 (DEPRICATED)
      [*] Enable Video for Linux API 1 (compatible) layer
          *** Multimedia Drivers ***
      [*] Video capture adapters --->
          [*] V4L USB devices --->
              <*> USB Video Class (UVC)
    USB Support --->
      <*> Support for Host-side USB
      *** Miscellaneous USB options ***
      [*] USB device filesystem
      [*] USB device class-devices (DEPRECATED)
     *** USB Host Controller Drivers ***
    <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
            **** Platform Glue Layer ***
    <>      TUSB6010
    <*>      OMAP2430 and onwards
    <>      AM35x
            Driver Mode (Both Host and peripheral : USB OTG (On The Go)
Device) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
```

Resources

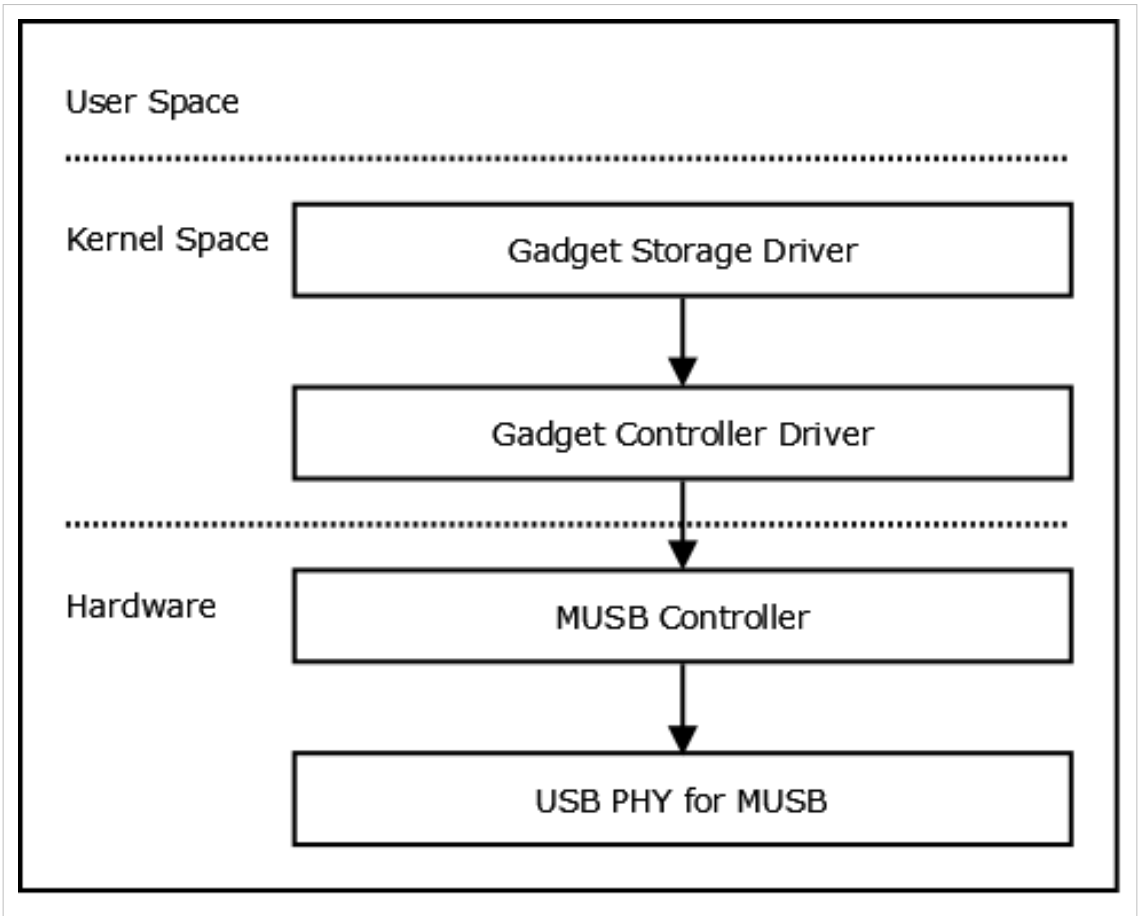
For testing USB Video support we need a user level application like mplayer to stream video from an USB camera. If you are using mplayer as the capture application, then you must export the DISPLAY to a X server. Then, execute the following command:

```
$ mplayer tv:// -tv driver=v4l2:width=320:height=240
```

Gadget Mode Applications

Mass Storage Gadget

This is the Mass storage gadget driver.



Configuration

```
Device Drivers --->
USB support --->
 <*> Support for USB Gadgets
 USB Peripheral Controller (Inventra HDRC Peripheral(TI, ...)) --->
 <M> USB Gadget Drivers
 <M> File-backed Storage Gadget
```

```
  <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
        **** Platform Glue Layer ***
  <>     TUSB6010
  <*>     OMAP2430 and onwards
```

```
    <>     AM35x
           Driver Mode (USB Peripheral (gadget stack)) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
```

Installation of File Storage Gadget Driver

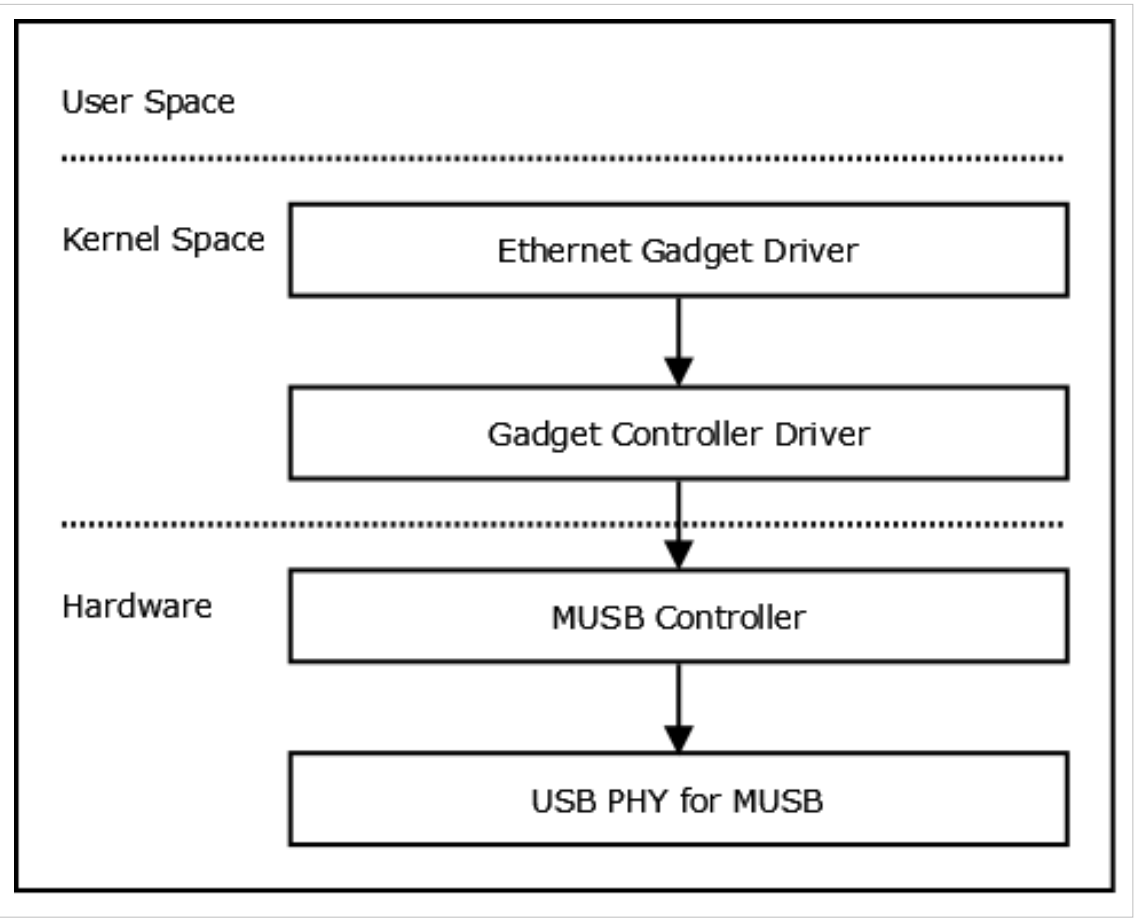
Let us assume that we are interested in exposing /dev/mmcblk0 block device to the file storage gadget driver. To that effect we need to issue the following command to load the file storage gadget driver.

```
$ insmod <g_file_storage.ko> file=/dev/mmcblk0 stall=0
```

CDC/RNDIS gadget

The CDC/RNDIS gadget driver that is used to send standard Ethernet frames using USB. Please enable "Use System DMA for Rx endpoints" to fix the flood ping hang issue with packet size of more than 16KB on OMAP35x due to Mentor DMA lockup issue.

The image below shows the USB stack architecture with CDC/RNDIS gadget.



Configuration for USB controller and CDC/RNDIS Gadget

```
Device Drivers --->
USB support --->
<*> Support for USB Gadgets
USB Peripheral Controller (Inventra HDRC Peripheral (TI, ...)) --->
<M> USB Gadget Drivers
```

```
<M> Ethernet Gadget
[*]   RNDIS support (EXPERIMENTAL) (NEW)
```

```
    <*> Inventra Highspeed Dual Role Controller (TI, ADI, ...)
           **** Platform Glue Layer ***
    <>     TUSB6010
    <*>     OMAP2430 and onwards
    <>     AM35x
           Driver Mode (USB Peripheral (gadget stack)) --->
    [ ] Disable DMA (always use PIO)
    [*] Use System DMA for Mentor DMA workaround
    [*] Enable debugging messages
```

Please do not select RNDIS support for testing ethernet gadget with Linux 2.4, IXIA and MACOS host machine.

```
 USB Peripheral Controller (Inventra HDRC Peripheral (TI, ...)) --->
 <M> USB Gadget Drivers
 <M> Ethernet Gadget
 [ ]   RNDIS support (EXPERIMENTAL) (NEW)
```

Installation of CDC/RNDIS Gadget Driver

Installing the CDC/RNDIS gadget driver is as follows:

```
 $ insmod  <path to g_ether.ko>
```

Setting up USBNet

The CDC/RNDIS Gadget driver will create a Ethernet device by the name usb0. You need to assign an IP address to the device and bring up the device. The typical command for that would be:

```
 $ ifconfig usb0  <IP_ADDR> netmask 255.255.255.0 up
```

Modular testing on MUSB

Mentor USB (MUSB) linux driver has been reorganized in v2.6.37 to support multi platform config. Modular structure has also changed due to this and thus now onward there will be below modules on MUSB

```
1. musb_hdrc.ko: The core controller module.
2. cppi41dma.ko or musbhsdma.ko: The dma controller module.
3. am35x.ko or omap2430.ko: The platform glue module.
4. g_file_storage.ko or g_ether.ko: The gadget controller module.
```

All the above four modules need to be inserted sequentially for an OTG (both host and device) configured or gadget only configured kernel. Fourth module (gadget controller) would not be needed in a host only config.

USB EHCI Electrical testing

USB EHCI electrical test is supported in software. Please use below command to perform various electrical tests.

```
$ echo 'Options' > sys/devices/platform/ehci-omap.0/portN
Where 'options' can be,
reset   --> Reset Device
t-j     --> Send TEST_J on suspended port
t-k     --> Send TEST_K on suspended port
t-pkt   --> Send TEST_PACKET[53] on suspended port
t-force --> Send TEST_FORCE_ENABLE on suspended port
t-se0   --> Send TEST_SE0_NAK on suspended port
```

USB OTG (HNP/SRP) testing

Please choose the configuration as described in driver configuration section for OTG and follow the steps below for testing.

- 1. Boot the OTG build image on two OMAP35x EVM.
- 2. If gadget driver is built as module then insert it to complete USB initialization.
- 3. Connect mini-A side of the OTG cable to one of the EVM (say EVM-1) and mini-B side on the other (say EVM-2).

In this scenario EVM-1 will become initial host or A-device and EVM-2 will become initial device or B-device. A-device will provide bus power throughout the bus communication even if it becomes peripheral using HNP.

There will not be any connect event at this point of time as Vbus power is not yet switched-on. Vbus power can be switched-on from A-device or from B-device using SRP.

- 4. Request to switch-on the Vbus power using below command on any EVM.

```
$ echo "F" > /proc/driver/musb_hdrc
```

If this command is executed on B-device then SRP protocol will be used to request A-device to switch-on the Vbus power.

- 5. Now the connect event occurs, enumeration will complete and gadget driver on B-device will be ready to use if this driver is in "Targeted Peripheral List (TPL)" of A-device.

If TPL is disabled on A-device then gadget driver will be ready to use soon after enumeration.

If TPL is enabled and gadget driver of B-device is not in TPL list of A-device then there will be an automatic trial of HNP from usb core by suspending the bus. This will cause a role switch and B-device will enumerate A-device. Now the gadget driver of A-device will be configured if it is on the TPL list of B-device.

Currently this is the only way possible for HNP testing but we have added a suspend proc entry to start HNP in other than this scenario.

- 6. Complete all the communication between A-device and B-device.
- 7. Start HNP by executing below command on host side.

```
$ echo "S" > /proc/driver/musb_hdrc
```

It will suspend the bus and role-switch will follow after that.

- 8. Repeat step 4, 5, 6 and 7 for further testing.

Software Interface

The USB driver exposes its state/control through the sysfs and the procfs interfaces. The following sections talks about these.

sysfs

sysfs attribute	Description
mode	The entry `/sys/devices/platform/musb_hdrc.0/mode` is a read-only entry. It will show the state of the OTG (though this feature is not supported) state machine. This will be true even if the driver has been compiled without OTG support. Only the states like A_HOST, B_PERIPHERAL, that makes sense for non-OTG will show up.
vbus	The entry `/sys/devices/platform/musb_hdrc.0/vbus` is a write-only entry. It is used to set the VBUS timeout value during OTG. If the current OTG state is a_wait_bcon then then urb submission is disabled.

procfs

The procfs entry `/proc/driver/musb_hdrc` is used to control the driver behaviour as well as check the status of the driver.

- 1. The following command will show the usage of this proc entry

```
$ echo "?" > /proc/driver/musb_hdrc
```

- 2. Specifically the most important usage of this entry would be to start an USB session(host mode) by issuing the following command:

$ echo "F" > /proc/driver/musb_hdrc

UserGuideMmcDriver PSP 04.02.00.07

MMC Driver

Introduction

TI DM35x/37x/AM3517 has an multimedia card high-speed/secure data/secure digital I/O (MMC/SD/SDIO) host controller, which provides an interface between microprocessor and either MMC, SD memory cards, or SDIO cards. The current version of the user guide talks about the MMC/SD controller. The MMC driver is implemented on top of host controller as a HS-MMC controller driver and supports MMC, SD, SD High Speed and SDHC cards. The salient features of the aforementioned HS-MMC host controller are:

- Full compliance with MMC/SD command/response sets as defined in

the Specification.

- Support:

1-bit or 4-bit transfer mode specifications for SD and SDIO cards 1-bit, 4-bit, or 8-bit transfer mode specifications for MMC cards

- Built-in 1024-byte buffer for read or write
- 32-bit-wide access bus to maximize bus throughput
- Single interrupt line for multiple interrupt source events
- Two slave DMA channels (1 for TX, 1 for RX)
- Designed for low power and Programmable clock generation

References

1. MMCA Homepage [http://www.mmca.org/home [1]]
2. SD ORG Homepage [http://www.sdcard.org/home [2]]

Acronyms & Definitions

Audio Driver: Acronyms

Acronym	Definition
MMC	Multimedia Card
HS-MMC	High Speed MMC
SD	Secure Digital
SDHC	SD High Capacity
SDIO	SD Input/Output

Features

The MMC/SD/SDIO driver supports following features

- The driver is built in-kernel (part of vmlinux).
- MMC cards including High Speed cards.
- SD cards including SD High Speed and SDHC cards
- Uses block bounce buffer to aggregate scattered blocks

Driver Configuration

The default kernel configuration enables support for MMC/SD(built-in to kernel).

The selection of MMC/SD/SDIO driver can be modified as follows: start Linux Kernel Configuration tool.

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

- Select Device Drivers from the main menu.

```
...
...
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

- Select MMC/SD/SDIO card support from the menu.

```
...
...
[*] USB support  --->
<*> MMC/SD/SDIO card support  --->
< > Sony MemoryStick card support (EXPERIMENTAL)  --->
...
...
```

- Select OMAP HS MMC driver

```
...
[ ] MMC debugging
[ ] Assume MMC/SD cards are non-removable (DANGEROUS)
     *** MMC/SD/SDIO Card Drivers ***
<*> MMC block device driver
[*]  Use bounce buffer for simple hosts
...
<*>   TI OMAP High Speed Multimedia Card Interface support
```

...

References

[1] http://www.mmca.org/home
[2] http://www.sdcard.org/home

UserGuideEthernetDriver PSP 04.02.00.07

Ethernet Driver

Introduction

OMAP3EVMs for AM/DM37x and OMAP35x platforms include an external SMSC ethernet controller interfaced through GPMC module. The older EVMs (till Rev.D) include SMSC 9115 and the newer EVMs include SMSC9220 with integrated EEPROM for storing the MAC address.

Features supported by this controller/driver include:

- Support for auto-negotiation
- Support for 10/100Mbps mode of operation
- PIO mode of operation (no DMA support)

AM35x platforms include integrated EMAC module with following features

- Support for auto-negotiation
- Support for 10/100Mbps Mode of operation
- DMA mode of operation
- NAPI compliant driver implementation

Driver Configuration

The default kernel configuration enables support for Ethernet driver(built-in to kernel).

The selection of Ethernet driver can be modified as follows: start Linux Kernel Configuration tool.

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

- Select Device Drivers from the main menu.

```
...
...
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

- Select Network device support from the menu.

```
...
...
< > Serial ATA and Parallel ATA drivers  --->
[ ] Multiple devices driver support (RAID and LVM)  --->
[*] Network device support  --->
[ ] ISDN support  --->
< > Telephony support  --->
...
...
```

On AM/DM37x,OMAP35x platforms choose SMSC911x driver

- Select Ethernet (10 or 100Mbit) from the menu.

```
...
...
< >   Virtual ethernet pair device
-*-   PHY Device support and infrastructure  --->
[*]   Ethernet (10 or 100Mbit)  --->
...
...
```

- Select SMSC LAN911x/LAN921x families embedded ethernet support from the menu

```
...
...
< >   OpenCores 10/100 Mbps Ethernet MAC support
< >   SMSC LAN911[5678] support
<*>   SMSC LAN911x/LAN921x families embedded ethernet support
< >   Dave ethernet support (DNET)
...
...
```

On AM35x platforms choose TI EMAC driver

- Select Ethernet (10 or 100Mbit) from the menu.

```
...
...
< >   Virtual ethernet pair device
-*-   PHY Device support and infrastructure  --->
[*]   Ethernet (10 or 100Mbit)  --->
[*]   Ethernet (1000 Mbit)  --->
...
...
```

- Select TI DaVinci EMAC/MDIO/CPDMA Support from the menu

```
...
...
< >   ASIX AX88796 NE2000 clone support
< >   SMC 91C9x/91C1xxx support
<*>   TI DaVinci EMAC Support
```

```
-*-   TI DaVinci MDIO Support
-*-   TI DaVinci CPDMA Support
< >   DM9000 support
...
```

...

UserGuidePowerMgmt PSP 04.02.00.07

Power Management

Introduction

OMAP35x and DM37x provide a rich set of power management features. These features are described in detail in the respective technical reference manuals (TRMs).

In summary, both silicons support:

- Clock control at the module and clock domain level.
- Multiple power domains i.e. one or more hardware modules sharing same power source.
- Control of scalable voltage domains.
- Independent scaling of OPPs for the VDD1 and VDD2.
- Support for transitioning power and voltage domains to retention/off.
- Wakeup on an event.

References

1. Proceedings of the Linux Symposium, June 27-30, 2007 [1]

   Authors: Venkatesh Pallipadi, Shaohua Li, Adam Belay
2. OMAP Power Management [2]

   Power Management features are being developed on `pm` branch of the `linux-omap` git tree. This page provides latest status of PM features on this branch.

Features

The power management features available in this release are based on the proposed PM interface for OMAP. This interface is described in the file `Documentation/arm/OMAP/omap_pm` in the Linux kernel sources. The features supported in this release are:

- Dynamic Tick (NO_HZ) framework.
- The *cpuidle* framework with MPU and Core transition to retention (RET) and OFF states.
  - The *menu* governor is supported.
- VDD1 OPP is now changable via frequency scaling i,e.`cpufreq` implementation:
  - VDD1 OPP can be scaled upto 720MHz (for OMAP35x) and 1GHz (for AM/DM37x).
  - When OPP1 is selected for VDD1, the VDD2 is set at OPP2.
- Support SmartReflex with automatic (hardware-controlled) mode of operation.

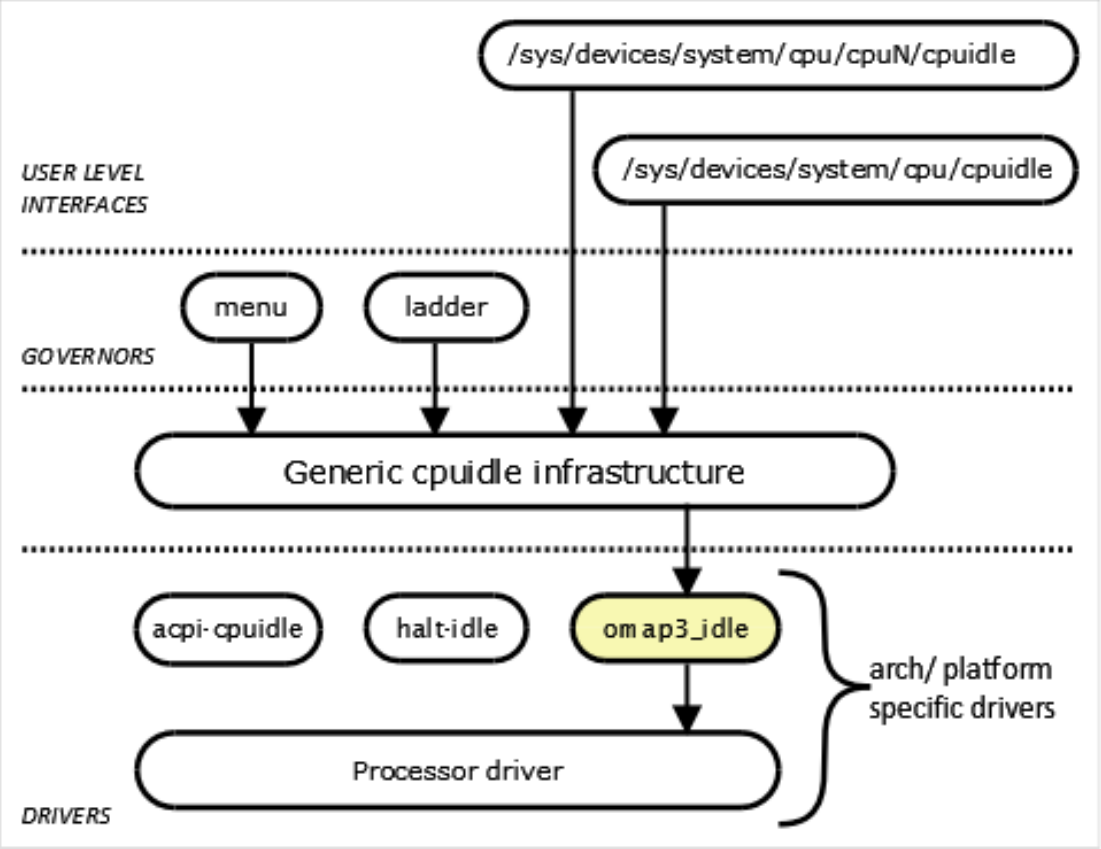
Architecture

cpuidle

The *cpuidle* framework consists of two key components:

- A governor that decides the target C-state of the system.
- A driver that implements the functions to transition to target C-state.

System Diagram



The idle loop is executed when the Linux scheduler has no thread to run. When the idle loop is executed, current 'governor' is called to decide the target C-state. Governor decides whether to continue in current state/ transition to a different state. Current 'driver' is called to transition to the selected state.

C-states

A C-state is used to identify the power state supported through the cpu idle loop. Each C-state is characterized by its:

- Power consumption
- Wakeup latency
- Preservation of processor state while in 'the' state.

The definition of C-states in the OMAP3 are a combination of the MPU and CORE states. Currently these C-states have been defined:

C-states in OMAP3

State	Description
C1	MPU WFI + Core active
C2	MPU WFI + Core inactive
C3	MPU CSWR + Core inactive
C4	MPU OFF + Core inactive
C5	MPU RET + CORE RET
C6	MPU OFF + CORE RET
C7	MPU OFF + CORE OFF

CPU Idle Governor

The current implementation supports the 'menu' governor to decide the target C-state of the system.

CPU Idle Driver

The cpuidle driver registers itself with the framework during boot-up and populates the C-sates with exit latency, target residency (minimum period for which the state should be maintained for it to be useful) and flag to check the bus activity.

In ACPI implementation, flag `CPUIDLE_FLAG_CHECK_BM` is used to specify the states requiring bus monitoring interface to be checked. In the OMAP3 implementation, this flag is used to identify the C-states that require CORE domain activity to be checked.

Once the governor has decided the target C-state, the control reaches the function `omap3_enter_idle()`. Here, the C-state is adjusted based on the value of *valid* flag corresponding to the chosen state./

Note

The value of *valid* flag for the idle states relates to the flag `enable_off_mode`. If transition to OFF mode is disabled, the idle states that require MPU to be turned OFF are made *valid*.

Performance considerations

Once idle power management is enabled, the system will transition across sleep states of varying latency. This transition can impact the runtime performance of the drivers. The flags `sleep_while_idle` and `enable_off_mode` can be used to control the run-time behavior of the *cpuidle* driver. are now accessible via `debugfs`.

Important

In previous kernel versions, these flags were accessible via `sysfs`. In this kernel version, these flags can be accessed via `debugfs`, if configuration options `CONFIG_PM_DEBUG` and `CONFIG_DEBUG_FS` are chosen.

See *Configuration* for steps to enable these configuration options.

Dynamic Tick Suppression

The dynamic tick suppression is achieved through generic Linux framework for the same. A 32K timer (HZ=128) is used by the tick suppression algorithm.

Suspend & Resume

The suspend operation results in the system transitioning to the lowest power state being supported. The drivers implement the `suspend()` function defined in the LDM. When the suspend for the system is asserted, the `suspend()` function is called for all drivers. The drivers release the clocks to reach the desired low power state. The actual transition to suspend is implemented in the function `omap3_pm_suspend()`.

Configuration

To enable/ disable power management start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

Select *Power Management support* to toggle the power management support.

```
[*] Power Management support
[ ]   Power Management Debug Support
[*] Suspend to RAM and standby
< > Advanced Power Management Emulation
```

Debugging support in Power Management

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
```

```
Device Drivers  --->
...
...
```

Select *Power Management support* from the next menu.

```
[*] Power Management support
[*]   Power Management Debug Support
[ ]       Extra PM attributes in sysfs for low-level debugging/testing
[ ]       Verbose Power Management debugging
[*] Suspend to RAM and standby
```

Enabling debug filesystem

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Kernel hacking* from the main menu.

```
File systems  --->
Kernel hacking  --->
Security options  --->
-*- Cryptographic API  --->
```

Debug Filesystem is already selected

```
[ ] Enable unused/obsolete exported symbols
-*- Debug Filesystem
[ ] Run 'make headers_check' when building vmlinux
[*] Kernel debugging
```

cpuidle

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *CPU Power Management* from the main menu.

```
...
...
System Type  --->
Bus support  --->
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
...
...
```

Select *CPU idle PM support* to enable the cpuidle driver. ...

```
< >  'conservative' cpufreq governor
[*] CPU idle PM support
```

cpufreq

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig arch=ARM
```

Select *CPU Power Management* from the main menu.

```
...
...
System Type  --->
Bus support  --->
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
...
...
```

Select *CPU idle PM support* to enable the cpuidle driver.

```
[*] CPU Frequency scaling
[ ] CPU idle PM support
```

SmartReflex

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *System Type* from the main menu.

```
...
...
[*] Enable the block layer  --->
System Type  --->
Bus support  --->
Boot options  --->
CPU Power Management  --->
...
...
```

Select *TI OMAP Common Features* from the menu.

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

```
...
...
```

Select *SmartReflex support* from the menu.

```
...
...
*** OMAP Feature Selections ***
[*] SmartReflex support
[*]     Class 3 mode of Smartreflex Implementation
...
...
```

Software Interface

The *cpuidle* framework defines a standard interface through `/sys` interface.

Mounting debug filesystem

To mount the filesystem, execute these commands:

```
$ mkdir /dbg
$ mount -t debugfs debugfs /dbg
```

Note
The commands in this document assume `/dbg` as mount point. Appropriate changes should be made if a different mount point is being used.

cpuidle

The parameters controlling *cpuidle* can be viewed via via `/sys` interface.

```
$ ls -1 /sys/devices/system/cpu/cpuidle/
current_driver
current_governor_ro
$
```

current_governor_ro lists the current governor.

```
$ cat /sys/devices/system/cpu/cpuidle/current_governor_ro
menu
$
```

current_driver lists the current driver.

```
$ cat /sys/devices/system/cpu/cpuidle/current_driver
omap3_idle
$
```

The *cpuidle* interface also exports information about each idle state. This information is organized in a directory corresponding to each idle state.

```
$ ls -1 /sys/devices/system/cpu/cpu0/cpuidle
state0
```

```
state1
state2
state3
state4
state5
state6
$
$ ls -1 /sys/devices/system/cpu/cpu0/cpuidle/state0
desc
latency
name
power
time
usage
```

Idle state transition

To allow/prevent the processor to enter idle states, execute these commands:

```
$ echo 1 > /dbg/pm_debug/sleep_while_idle
$ echo 0 > /dbg/pm_debug/sleep_while_idle
```

To allow/ prevent transition to OFF mode:

```
$ echo 1 > /dbg/pm_debug/enable_off_mode
$ echo 0 > /dbg/pm_debug/enable_off_mode
```

Suspend & Resume

The suspend for device can be asserted as follows:

```
$ echo -n "mem" > /sys/power/state
```

To wakeup, press a key on the OMAP3EVM keypad; or tap any on the serial console.

cpufreq & SmartReflex

Frequency scaling is achieved through `cpufreq` as `mpurate` feature is no longer supported. To get a list of all the available frequencies:

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
125000 250000 500000 550000 600000 720000 < for OMAP35x >
```

To set a particular frequency:

```
$ echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

To verify, use :

```
$ cat /proc/cpuinfo
Processor : ARMv7 Processor rev 3 (v7l)
BogoMIPS : 597.64
...
```

The SmartReflex module now acts without user intervention and the dynamic change in voltage (VDD1 and VDD2) and the current can be measured using steps detailed at http:/ / processors. wiki. ti. com/ index. php/

Measuring_Power_on_the_OMAP35x_EVM

References

[1] http://ols.108.redhat.com/2007/Reprints/pallipadi-Reprint.pdf
[2] http://elinux.org/OMAP_Power_Management

UserGuidePowerMgmt AM35x 04.02.00.07

Power Management In AM35xx

Introduction

AM3517 and AM3505 silicon provides a limited set of power management features. These features are described in detail in the AM3517/05 TRM. In summary:

- Clock control at the module and clock domain level.
- 16 power domains i.e. 16 sets of one or more hardware modules that can be independently turned on/off.
- Support for transitioning power domains to retention and wakeup on event.

References

1. OMAP Power Management [2]
    1. Power Management features are being developed on `pm` branch of the `linux-omap` git tree. This page provides latest status of PM features on this branch.

Features

The power management features available in this release are based on the proposed PM interface for OMAP. This interface is described in the filename `Documentation/arm/OMAP/omap_pm`.

Supported

This is list of features supported in this release:

- Supports Dynamic Tick framework.
- Supports the *Suspend/Resume* capability, thus allowing the system to enter into a standby state.

Architecture

Dynamic Tick Suppression

The dynamic tick suppression is achieved through generic Linux framework for the same. A 32K timer (HZ=128) is used by the tick suppression algorithm.

Suspend & Resume

The suspend operation results in the system transitioning to the lowest power state being supported. The drivers implement the `suspend()` function defined in the LDM. When the suspend for the system is asserted, the `suspend()` function is called for all drivers. The drivers release the clocks to reach the desired low power state. The actual transition to suspend is implemented in the function `omap3_pm_suspend()`.

Configuration

To enable/ disable power management start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Kernel Features  --->
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

Select *Power Management support* to toggle the power management support.

```
[*] Power Management support
[ ]   Power Management Debug Support
[*] Suspend to RAM and standby
< > Advanced Power Management Emulation
```

Power Management Debug Support

To enable power management debugging capabilities, debug file system and power management debug support need to be enabled in the configuration.

Enabling Debug Filesystem

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Kernel hacking* from the main menu.

```
File systems  --->
Kernel hacking  --->
Security options  --->
-*- Cryptographic API  --->
```

Select *Debug Filesystem* from the next menu.

```
[ ] Enable unused/obsolete exported symbols
[*] Debug Filesystem
[ ] Run 'make headers_check' when building vmlinux
[*] Kernel debugging
```

Debugging Support In Power Management

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

Select *Power Management Debug Support* from the next menu.

```
[*] Power Management support
[*]   Power Management Debug Support
[*] Suspend to RAM and standby
< > Advanced Power Management Emulation
```

Software Interface

Suspend & Resume

The suspend for device can be asserted as follows:

```
$ echo -n "mem" > /sys/power/state
```

To wakeup, tap any key on the serial console. Automatic wakeup can be done by programming the wakeup timer before asserting suspend. The wakeup timer is accessed via debug filesystem interface and this requires both debug file system support and debugging support in power management enabled in the kernel configuration as described earlier. For example, to have an auto wakeup after 3 seconds, do the following before suspend is asserted Mount debug filesystem. This is a one time configuration.

```
$ mkdir /debug
$ mount -t debugfs debugfs /debug
```

Now program the wakeup timer.

```
$ echo -n 3 > /debug/pm_debug/wakeup_timer_seconds
```

To disable auto wakeup, do the following $ echo -n 0 > /debug/pm_debug/wakeup_timer_seconds

UserGuidePmic PSP 04.02.00.07

Introduction

A Power Management IC (PMIC) is a device that contains one or more regulators (voltage or current) and often contains other susbsystems like audio codec, keypad etc. From the power management perspective, its main function is to regulate the output power from input power or simply enable/disable the output as and when required.

A PMIC can have two different types of regulators: voltage regulator or current regulator. Voltage regulators are used to enable/disable the output voltage and/ or regulate the output voltage. Current regulators preform the same functions with the output current. PMICs from TI contain two different types of voltage regulators:

- DCDC: Highly efficient and self-regulating step down DC to DC converter.
- LDO (low-dropout): DC linear voltage regulator which can operate with a very small input-output differential voltage.

The PMIC is controlled by internal registers that can be accessed by the I2C control interface.

This manual defines and describes the usage of user level and platform level interfaces of the PMIC consumer driver.

References

- Linux Voltage and Current Regulator Framework [1]
- Linux kernel documentation: Documentation/power/regulator and Documentation/ABI/testing/sysfs-class-regulator
- TPS65950 [6]: Integrated Power Management IC with 3 DC/DC's, 11 LDO's, Audio Codec, USB HS Transceiver, Charger
- TPS65023 [2]: 6-channel Power Mgmt IC with 3DC/DCs, 3 LDOs, I2C Interface and DVS, Optimized for DaVinci DSPs Literature Number: SLVS670G
- TPS65073 [3]: 5-Channel Power Management IC with 3 DC/DCs, 2 LDOs in 6x6mm QFN

Acronyms & Definitions

Acronym	Definition
PMIC	Power Management IC
VRF	Voltage Regulator Framework
LDO	Low Drop-Out

Features

This section describes the supported features and constraints of the PMIC drivers.

Features Supported

- Support for TPS65950, TPS65023 and TPS65073 PMICs in Linux Voltage Regulator Framework.
- Enabling / disabling of voltage regulators.
- Changing the output voltage, if permitted by the voltage regulator.
- Reading the status (enabled/disabled) of the voltage regulator.
- Reading other useful information about the regulator via sysfs interface.

Constraints

None

Configuration

To enable/disable VRF support, start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Check whether I2C support is enabled or not; it is required for the voltage regulator. Select *Device Drivers* from the main menu:

```
...
...
    Power management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Kernel hacking  --->
...
...
```

Then select *I2C support* as shown here:

```
...
...
    Input device support  --->
    Character devices  --->
-*- I2C support  --->
[*] SPI support  --->
    PPS Support  --->
...
...
```

Select *I2C Hardware Bus support* after selecting "I2C device interface" from the menu, as shown here:

```
...
...
< >   I2C bus multiplexing support
<*>   Autoselect pertinent helper modules
      I2C Hardware Bus support  --->
[ ]   I2C Core debugging messages
...
...
```

OMAP I2C adapter must have been automatically selected as shown here:

```
...
...
< > GPIO-based bitbanging I2C
< > OpenCores I2C Controller
-*- OMAP I2C adapter
```

```
< > PCA9564/PCA9665 as platform device
< > Simtec Generic I2C interface
< > Xilinx I2C Controller
...
...
```

Go back and select *Voltage and Current Regulator Support* from the *Device Drivers* menu.

```
...
...
   Sonics Silicon Backplane --->
-*- Multifunction device drivers --->
-*- Voltage and Current Regulator Support  --->
<*> Multimedia support  --->
    Graphics support --->
...
...
```

Make sure that the PMIC present on the EVM is selected here, which is **TI TWL4030/TWL5030/TWL6030/TPS695x0 PMIC** for AM/DM-37x and OMAP35x EVM

```
...
...
< > Maxim 8649 voltage regulator
< > Maxim 8660/8661 voltage regulator
< > Maxim MAX8952 Power Management IC
[*]   TI TWL4030/TWL5030/TWL6030/TPS695x0 PMIC
< > National Semiconductors LP3971 PMIC regulator driver
< > National Semiconductors LP3972 PMIC regulator driver
...
...
```

And **TI TPS65023 Power regulators** for AM3517 EVM

```
...
...
< > National Semiconductors LP3971 PMIC regulator driver
< > National Semiconductors LP3972 PMIC regulator driver
< > TI TPS65023 Power regulators
< > TI TPS6507X Power regulators
< > Intersil ISL6271A Power regulator
...
...
```

Only for AM/DM37x and OMAP35x EVM, come back to the Device Drivers menu and select **Multifunction device drivers** as shown below:

```
...
...
[*] Watchdog Timer Support --->
    Sonics Silicon Backplane --->
```

```
-*- Multifunction device drivers --->
-*- Voltage and Current Regulator Support --->
< > Multimedia support --->
    Graphics support --->
...
...
```

Select *Texas Instruments TWL4030/TWL5030/TWL6030/TPS659x0 Support* & *Support power resources on TWL4030 family chips* as shown here:

```
...
...
< > TPS6501x Power Management chips
< > TPS6507x Power Management / Touch Screen chips
[*] Texas Instruments TWL4030/TWL5030/TWL6030/TPS659x0 Support
[*]   Support power resources on TWL4030 family chips
< > TWL6030 PWM (Pulse Width Modulator) Support
[ ] Support STMicroelectronics STMPE
...
...
```

Application Interface

This section provides the details of the application interface for the PMIC regulator driver. Client device drivers are the ones which use PMIC regulator drivers to enable/ disable and/or regulate output voltage/current. Specifically, a client driver uses:

- Consumer driver interface: This uses a similar API to the kernel clock interface in that consumer drivers can get and put a regulator (like they can with clocks) and get/set voltage, current limit, enable and disable. This allows consumer complete control over their supply voltage and current limit.
- Sysfs interface: The linux voltage regulator framework also exports a lot of useful voltage/current/opmode data to userspace via sysfs. This could be used to help monitor device power consumption and status.

Consumer driver interface

As mentioned above, this interface provides complete control to the consumer driver over their supply voltage and/or current limit. Some of the commonly used APIs to achieve this are:

regulator_get

Get handle to a regulator

regulator_put

Release a regulator

regulator_enable

Enable a regulator

regulator_disable

Disable a regulator

regulator_is_enabled

Check status of the regulator

regulator_set_voltage

Change the output voltage

regulator_get_voltage

Fetch the current voltage

regulator_set_current_limit

Change the current limit

regulator_get_current_limit

Fetch the existing current limit

Sysfs interface

The voltage regulator interface exports following information via `sysfs`.

Here is a list of entries available under `/sys/class/regulator/regulator.x`:

name

string identifying the regulator (may be empty)

type

regulator type (voltage, current)

state

regulator enable status (enabled, disabled)

microvolts

regulator output voltage (in microvolts)

microamps

regulator output current limit (in microamps)

min_microvolts

minimum safe working output voltage setting

max_microvolts

maximum safe working output voltage setting

min_microamps

minimum safe working output current setting

max_microamps

maximum safe working output current setting

num_users

current number of users

Writing a Consumer Driver

This section describes the steps required in writing a consumer driver to regulate the output voltage and to enable/disable the regulator. User should refer to `"include/linux/regulator/consumer.h"` for the complete list of supported APIs.

Writing a consumer driver involves the following steps:

- Getting the required regulator handle: Consumer driver has to first obtain a handle to the desired regulator, by passing the correct supply.

```
int ret;
struct regulator *reg;
const char *supply = "vdd1";
int min_uV, max_uV;
reg = regulator_get(NULL, supply);
```

- Enabling it, if not already enabled: A regulator needs to be enabled before it can be used to change the output voltage. Regulator handle, obtained in the previous step, should be used now to enable it:

```
ret = regulator_enable(reg);
```

- After enabling it, user can check the status of the regulator by:

```
printk (KERN_INFO "Regulator Enabled = %d\n",
regulator_is_enabled(reg));
```

- Changing the existing voltage: Consumer driver has to pass the appropriate minimum and maximum voltage levels, as desired by the use-case, to change the output voltage.

```
ret = regulator_set_voltage(reg, min_uV, max_uV);
```

- Reading the existing voltage: Consumer driver can read back the existing voltage to check if the voltage was set properly or not.

```
printk (KERN_INFO "Regulator Voltage = %d\n",
regulator_get_voltage(reg));
```

- Disabling the regulator: Consumer driver can disable the regulator, if required. The framework ensures that the regulator is not disabled if other consumers are still using the same regulator.

```
ret = regulator_disable(reg);
```

- Releasing the regulator handle: Consumer driver can release the regulator if it is no more required.

```
regulator_put(reg);
```

After that, the handle becomes invalid and should not be used for any further operations.

References

[1] http://opensource.wolfsonmicro.com/node/15
[2] http://focus.ti.com/docs/prod/folders/print/tps65023.html
[3] http://focus.ti.com/docs/prod/folders/print/tps65073.html

Sitara AM35x CAN (HECC) Linux Driver

Overview

This wiki page provides information on using TI's CAN (HECC) Linux driver module on Sitara AM3517 EVM. The ti_hecc driver is already part of linux since 2.6.32. This document concerns the updated support targeting the linux-2.6.37 tree.

Linux Driver Information

- CAN device driver in Linux is provided as a networking driver that confirms to the socketCAN interface
- The driver is currently build-into the kernel with the right configuration items enabled (details below)
- CAN PHY on the extension application board is enabled via a i2c expander (using one of the TCA6416 IO expander chips) port. This needs to be enabled during kernel configuration.

How CAN driver fits into Linux architecture

- TI HECC CAN driver is a can "networking" driver that fits into the Linux Networking framework
- It is available as a configuration item in the linux kernel configuration as follows:

```
Linux Kernel Configuration
   Networking support
      CAN bus subsystem support
         CAN device drivers
            Platform CAN drivers with Netlink support
               CAN bit-timing calculation
               TI High End CAN controller (HECC)
```

Detailed Kernel Configuration

To enable/disable CAN driver support, start the *Linux Kernel Configuration* tool:

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

Select *Networking support* from the main menu.

```
   ...
   ...
   Power management options --->
   [*] Networking support --->
   Device Drivers --->
   File systems --->
   Kernel hacking --->
   ...
   ...
```

Select *CAN bus subsystem support* as shown here:

```
    ...
    ...
    Networking options  --->
[ ] Amateur Radio support  --->
<*> CAN bus subsystem support  --->
    IrDA (infrared) subsystem support  --->
    ...
```

Select *Raw CAN Protocol* & *Broadcast Manager CAN Protocol* as shown here:

```
    ...
    --- CAN bus subsystem support
<*> Raw CAN Protocol (raw access with CAN-ID filtering)
<*> Broadcast Manager CAN Protocol (with content filtering)
    CAN Device Drivers  --->
```

Select *CAN Device Drivers* in the above menu and then select the following options:

```
<*> Virtual Local CAN Interface (vcan)
<*> Platform CAN drivers with Netlink support
[*]   CAN bit-timing calculation
<*>   TI High End CAN Controller
< >   Microchip MCP251x SPI CAN controllers
          ...
```

Note: -> "CAN bit-timing calculation" needs to be enabled to use "ip" utility to set CAN bitrate

To enable PHY control, go back to the main-menu and select *Device Drivers* :

```
    ...
    ...
    Power management options --->
    [*] Networking support --->
    Device Drivers --->
    File systems --->
    Kernel hacking --->
    ...
    ...
```

Select *GPIO Support* as shown below:

```
    ...
     ...
-*-  I2C support  --->
     PPS support  --->
[*]  SPI support  --->
-*-  GPIO Support  --->
< >  Dallas's 1-wire support  --->
```

Select *PCA953x, PCA955x, TCA64xx, and MAX7310 I/O ports* as shown below:

```
    ...
     ...
```

```
*** I2C GPIO expanders: ***
< > Maxim MAX7300 GPIO expander
< > MAX7319, MAX7320-7327 I2C Port Expanders
<*> PCA953x, PCA955x, TCA64xx, and MAX7310 I/O ports
[ ]     Interrupt controller support for PCA953x
     ...
```

CAN Utilities

- Since CAN is a "networking" interface and uses the socket layer concepts, many utilities have been developed in open source for utilizing CAN interface.
- A good source of these utilities is the Berlios SocketCAN website [1] that offers these utilities via SVN (& HTTP interface)
- For testing CAN we commonly use the cansend /cangen and candump utilities to send and receive packets via CAN module. The utilities have been cross-compiled and made available on this page.
- To configure the CAN interface netlink standard utilities are used and this requires iproute2 utilities. The cross compiled "ip" utility is available for download on this page.

Note that if the kernel configuration for "CAN bit-timing calculation" is not enabled then each of the parameters: tq, PROP_SEG etc need to be set individually. When bit-timing calculation is enabled int he kernel then only setting the bitrate is sufficient as it calculates other parameters

ip (route2)

- "ip" utility help - ensure you are using iproute2 utility as only that supports CAN interface.

```
Usage: ip link set DEVICE type can
       [ bitrate BITRATE [ sample-point SAMPLE-POINT] ] |
       [ tq TQ prop-seg PROP_SEG phase-seg1 PHASE-SEG1
         phase-seg2 PHASE-SEG2 [ sjw SJW ] ]
       [ loopback { on | off } ]
       [ listen-only { on | off } ]
       [ triple-sampling { on | off } ]
       [ restart-ms TIME-MS ]
       [ restart ]
       Where: BITRATE       := { 1..1000000 }
              SAMPLE-POINT  := { 0.000..0.999 }
              TQ            := { NUMBER }
              PROP-SEG      := { 1..8 }
              PHASE-SEG1    := { 1..8 }
              PHASE-SEG2    := { 1..8 }
              SJW           := { 1..4 }
              RESTART-MS    := { 0 | NUMBER }
```

- check the result of `ip link help` ;ensure that the `TYPE` field contains `can`

```
Usage: ip link add link DEV [ name ] NAME
                    [ txqueuelen PACKETS ]
                    [ address LLADDR ]
                    [ broadcast LLADDR ]
                    [ mtu MTU ]
```

```
                  type TYPE [ ARGS ]
       ip link delete DEV type TYPE [ ARGS ]

       ip link set DEVICE [ { up | down } ]
                          [ arp { on | off } ]
                          [ dynamic { on | off } ]
                          [ multicast { on | off } ]
                          [ allmulticast { on | off } ]
                          [ promisc { on | off } ]
                          [ trailers { on | off } ]
                          [ txqueuelen PACKETS ]
                          [ name NEWNAME ]
                          [ address LLADDR ]
                          [ broadcast LLADDR ]
                          [ mtu MTU ]
                          [ netns PID ]
                          [ alias NAME ]
       ip link show [ DEVICE ]


  TYPE := { vlan | veth | vcan | dummy | ifb | macvlan | can }
```

- Ensure you are using the right ip utility (from iproute2)

```
<prompt> ./ip -V
ip utility, iproute2-ss090324
```

- Set the bit-rate to 125Kbits/sec with triple sampling using the following command (for device can0)

```
<prompt> ip link set can0 type can bitrate 125000 triple-sampling
 on
```

- Bring up the device using the command:

```
<prompt> ip link set can0 up
```

cansend

- Usage: cansend <device> <can_frame>.
- Send a CAN packet

```
./cansend can0 111#1122334455667788
```

candump

- Usage: candump [options] <CAN interface>+ (use CTRL-C to terminate candump)

```
Options: -t <type>   (timestamp:
(a)bsolute/(d)elta/(z)ero/(A)bsolute w date)
        -c          (increment color mode level)
        -i          (binary output - may exceed 80 chars/line)
        -a          (enable additional ASCII output)
        -S          (swap byte order in printed CAN data[] - marked
```

```
with '`' )
        -s <level>  (silent mode - 0: off (default) 1: animation
2: silent)
        -b <can>    (bridge mode - send received frames to
<can>)
        -B <can>    (bridge mode - like '-b' with disabled
loopback)
        -l          (log CAN-frames into file. Sets '-s 2' by default)
        -L          (use log file format on stdout)
        -n <count>  (terminate after receiption of <count>
CAN frames)
        -r <size>   (set socket receive buffer to <size>)
Up to 16 CAN interfaces with optional filter sets can be specified
on the commandline in the form: <ifname>[,filter]*
Comma separated filters can be specified for each given CAN interface:
 <can_id>:<can_mask> (matches when <received_can_id>
& mask == can_id & mask)
 <can_id>~<can_mask> (matches when <received_can_id>
& mask != can_id & mask)
 #<error_mask>       (set error frame filter, see
include/linux/can/error.h)
CAN IDs, masks and data content are given and expected in hexadecimal
values.
When can_id and can_mask are both 8 digits, they are assumed to be 29
bit EFF.
Without any given filter all data frames are received ('0:0' default
filter).
Use interface name 'any' to receive from all CAN interfaces.
```

- Examples:

```
candump -c -c -ta can0,123:7FF,400:700,#000000FF can2,400~7F0 can3 can8
candump -l any,0~0,#FFFFFFFF    (log only error frames but no(!) data
frames)
candump -l any,0:0,#FFFFFFFF    (log error frames and also all data
frames)
candump vcan2,92345678:DFFFFFFF (match only for extended CAN ID
12345678)
candump vcan2,123:7FF (matches CAN ID 123 - including EFF and RTR
frames)
candump vcan2,123:C00007FF (matches CAN ID 123 - only SFF and non-RTR
frames)
```

EVM Information

- AM3517 EVM User's Guide [2]
- AM3517 EVM consists of two boards - Experimenter base board and Application board.
- CAN interface is available on connectors J15 and J36 on the application board. The pin connections are as follows:

```
Pin 1 – CAN-H
Pin 2 – CAN-L
Pin 3 – Bus termination (do not connect a wire to this externally)
Pin 4 – Ground
```

- A Cable is needed to connect pins 1-1, 2-2 and 4-4 for connection between boards.
- CAN bus termination - Short pins 2 & 3 on J15 to terminate the CAN bus
- To connect two boards the following connections have to be made

```
Short pins 2 & 3 on J15 on each board
Board A J36 pins 1,2,4 connected to pins 1,2,4 of J36 on Board B
respectively
No connection on pin 3 of J36 on any board
```

- Note that at low speeds and short cable distances, it may be possible to use the CAN bus without termination installed, or with termination only at one end of the bus.
- To connect two or more AM3517 boards using CAN the bus would look like this:

```
Board A J15 pins 2 & 3 shorted to terminate bus
Board A J36 pins 1,2,4 connect to Board B J36 pins 1,2,4 respectively
Board B J15 pins 1,2,4 connect to Board C J36 pins 1,2,4 respectively
Board C J15 pins 2 & 3 connected to terminate bus
```

- If you wanted to connect even more boards follow a similar pattern using J15 and J36 to connect any boards in the middle of the chain, and use J36 to connect and J15 to terminate boards at the end of the chain.

References

[1] http://developer.berlios.de/projects/socketcan/
[2] http://support.logicpd.com/downloads/1251/

AM35x-NOR-Flash-Support-ApplicationNote

Important

Please note that Currently NOR boot mode is not supported in PSP releases.

Overview

This wiki page provides information on using NOR Flash (PC28F640P30B85) support available on Logic PD's AM3517 Application board. This document talks about how to add support for NOR Flash (PC28F640P30B85) in X-loader (Primary Boot-loader), U-Boot (secondary boot-loader), Linux Kernel and will demonstrate NOR Flash boot mode.

NOTE: Since NOR is XIP we do not need to have two stage booting process, ROM code can directly jump to U-boot start address in NOR Flash. But to keep consistency with other boot mode options (NAND, MMC, etc...) we are following up two stage process here.

Background Information

- Since NOR Flash is XIP not special configuration other than GPMC initialization required in X-Loader. But natively X-Loader is designed (or rather required) considering NAND and MMC boot mode.
- U-Boot does support and have common interfaces for CFI Complaint flash devices. Also the NOR Flash (PC28F640P30B85) available on AM3517 Application board is CFI Complaint.
- User can choose the boot medium (either NOR or NAND Flash) using user configurable switch S11. Below are the supported options through S11,

User Switches				
Switch				
1	2	3	4	**Function**
OFF	OFF	X	X	No Video Input
ON	OFF	X	X	Video Input from Camera
OFF	ON	X	X	Video Input from Expansion Connector
ON	ON	X	X	Video Input from Video Decoder
X	X	OFF	X	Boot from NAND on SOM
X	X	ON	X	Boot from NOR on App Board
X	X	X	OFF	Disable WILAN Daughter Board
X	X	X	ON	Enable WILAN Daughter Board

Note: Please note that NOR Flash is located on Application board and requires hardware modification to get NOR boot mode working. Please refer to section "EVM configuration" for more details.

- EVM Configuration

In NAND Boot mode :-

```
Set S7 switch position to

                1     2     3     4     5
              -----|-----|-----|-----|-----
               off   off   off   off   off
```

nCS0 => NAND Flash device (Boot device)

nCS2 => NOR Flash device

In NOR Flash boot mode :-

```
Set S7 switch position to

                1     2     3     4     5
              -----|-----|-----|-----|-----
                on   off   off   off   off
```

nCS0 => NOR Flash device (Boot device)

nCS2 => NAND Flash device

EVM Board modification (NOR Boot mode) -

- From SOM board remove R235 register.
- And configure switch S7 in NOR Flash boot mode (as described above)

NOTE: Please note that, once you remove R235 register from SOM board, you will not be able to boot from NAND without Application board. The GPMC_nCS0 is now routed through Application board.

NOR Flash Layout

As mentioned above, there are 2 possible options here,

- Over nCS0, where it will work as a boot medium.
- Over nCS2, as a data flash

The NOR part on the EVM has been configured in the following manner. Please note that the base address is dependent on the boot device, so below layout shows the offset and not actual base address.

Note
If NOR Flash is detected on CS0, the address would be 0x00000000 and if NOR Flash is detected on CS2, the address would be 0x08000000.

```
 +------------+->0x00000000-> X-loader start
 |            |
 |            |-->0x0001FFFF-> X-loader end
 |            |-->0x00020000-> U-Boot start
 |            |
 |            |
 |            |-->0x000BFFFF-> U-Boot end
 |            |-->0x000C0000-> ENV start
 |            |
 |            |-->0x000FFFFF-> ENV end
 |            |-->0x00100000-> Linux Kernel start
 |            |
```

```
|             |
|             |
|             |-->0x004FFFFF-> Linux Kernel end
|             |-->0x00500000-> Filesystem start
|             |
|             |
|             |
|             |
|             |
|             |
|             |
|             |
+-------------+-->0x007FFFFF-> Filesystem end
```

Adding support in X-Loader, U-Boot & Kernel (on top of PSP03.00.01.06 release)

Adding support in X-Loader

Patch: 

Adding support in U-Boot

Patch: 

Adding support in Linux Kernel

Nor Flash hook up -

Patch: 

NOTE: Please note that above patch disabled the NAND initialization in board init sequence.

Defconfig changes required to enable NOR Flash -

Patch: 

Adding support in X-Loader, U-Boot & Kernel(on top of PSP04.02.00.06 release)

Please note that support for NOR Flash is disabled in U-boot configuration, whereas in X-loader & Linux it is enabled by default.

Adding support in U-Boot

Since NOR Flash is mounted on Application board and natively u-boot doesn't support run-time detection and configuration of Flash devices, the NOR Flash support is disabled in U-boot. Please enable.

Patch: 

Usage

This section talks about default partitions defined and how user can flash the images to respective partition in NOR flash -

Default Partition configiration

To keep aligned with other flash devices supported with OMAP family of devices like, NAND or OneNAND we have defined similar partition table for NOR Flash. Please note that, first 4 blocks are of 32K and rest all blocks are 128K.

```
X-Loader      :-     0x00000000 - 0x00020000 (4 blocks = 128K)
U-Boot        :-     0x00020000 - 0x000C0000 (5 blocks = 864K)
U-Boot Env    :-     0x000C0000 - 0x00100000 (2 blocks = 1M)
Kernel        :-     0x00100000 - 0x00500000 (32 blocks = 4MB)
Data/FS       :-     0x00500000 - 0x07FFFFFF (Rest all)
```

Flashing Instruction

This section describes how to flash x-load.bin and u-boot.bin to NOR flash, please note that flashing is achieved from another uboot image, booted up from MMC/NAND Flash.

X-Loader

```
AM3517_EVM #
AM3517_EVM # protect off 0x08000000 +0x20000
Un-Protect Flash Bank # 1
.... done
AM3517_EVM # erase 0x08000000 +0x20000
Erase Flash Bank # 1
.... done
AM3517_EVM # mw.b 0x80000000 0xff 0x20000
AM3517_EVM # loadb 80000000
## Ready for binary (kermit) download to 0x80000000 at 115200 bps...
## Total Size      = 0x000042b0 = 17072 Bytes
## Start Addr      = 0x80000000
```

```
AM3517_EVM # cp.b 0x80000000 0x08000000 0x20000
Copy to Flash... done
AM3517_EVM #
```

U-Boot

```
AM3517_EVM #
AM3517_EVM # protect off 0x08020000 +0xA0000
Un-Protect Flash Bank # 1
...... done
AM3517_EVM # erase 0x08020000 +0xA0000
Erase Flash Bank # 1
...... done
AM3517_EVM # mw.b 0x80000000 0xFF 0xA0000
AM3517_EVM # loadb 0x80000000
## Ready for binary (kermit) download to 0x80000000 at 115200 bps...
## Total Size      = 0x00039128 = 233768 Bytes
## Start Addr      = 0x80000000
AM3517_EVM # cp.b 0x80000000 0x08020000 0xA0000
Copy to Flash... done
```

NOTE: Please note that early revision of Application board doesn't work with NOR flash boot mode due to HW issue. The NOR boot mode has been validated against >=Rev4 revision of Application board.

NOTE: Please untar the file to access the patch, tar file has been attached since Wiki page doesn't allow to attach .patch extension files.

NOTE: Please note that the above patches have not went under regression test, so likely to have some issues. We are working on this and will update this page once we are done with your testing. By default, NOR Flash goes to nCS2 chip select and will be able to use Data Flash.

Article Sources and Contributors

AM35x-OMAP35x-PSP 04.02.00.07 UserGuide *Source*: http://processors.wiki.ti.com/index.php?oldid=58005 *Contributors*: Hvaibhav

UserGuideAudioDriver PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57565 *Contributors*: Hvaibhav

UserGuideDisplayDrivers PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57573 *Contributors*: Hvaibhav

UserGuideOmap35xCaptureDriver PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57578 *Contributors*: Hvaibhav

UserGuideAM3517CaptureDriver PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57586 *Contributors*: Hvaibhav

UserGuideUsbDriver PSP 04 02 00 07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57588 *Contributors*: Hvaibhav

UserGuideMmcDriver PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57592 *Contributors*: Hvaibhav

UserGuideEthernetDriver PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57595 *Contributors*: Hvaibhav

UserGuidePowerMgmt PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57600 *Contributors*: Hvaibhav

UserGuidePowerMgmt AM35x 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57602 *Contributors*: Hvaibhav

UserGuidePmic PSP 04.02.00.07 *Source*: http://processors.wiki.ti.com/index.php?oldid=57604 *Contributors*: Hvaibhav

Sitara AM35x CAN (HECC) Linux Driver *Source*: http://processors.wiki.ti.com/index.php?oldid=56267 *Contributors*: Abhilashkv, Alanc, AnantGole, Kevinsc

AM35x-NOR-Flash-Support-ApplicationNote *Source*: http://processors.wiki.ti.com/index.php?oldid=58007 *Contributors*: Hvaibhav

Image Sources, Licenses and Contributors

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

Image:Omap35x-boot-onenand1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-onenand1.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-onenand2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-onenand2.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-onenand-mmc1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-onenand-mmc1.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-onenand-mmc2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-onenand-mmc2.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-nand1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-nand1.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-nand2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-nand2.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-nand-mmc1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-nand-mmc1.png *License*: unknown *Contributors*: SanjeevPremi

Image:Omap35x-boot-nand-mmc2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-boot-nand-mmc2.png *License*: unknown *Contributors*: SanjeevPremi

Image:Am3517-boot-nand.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Am3517-boot-nand.png *License*: unknown *Contributors*: SanjeevPremi

Image:Am3517-boot-mmc.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Am3517-boot-mmc.png *License*: unknown *Contributors*: SanjeevPremi

Image:Asoc architecture.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Asoc_architecture.png *License*: unknown *Contributors*: AnujAggarwal, SanjeevPremi

Image:Am3517 asoc architecture.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Am3517_asoc_architecture.png *License*: unknown *Contributors*: SanjeevPremi

Image:Alsa-usecase1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Alsa-usecase1.png *License*: unknown *Contributors*: SriramAG

Image:Alsa-usecase2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Alsa-usecase2.png *License*: unknown *Contributors*: SriramAG

Image:Display-arch.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-arch.png *License*: unknown *Contributors*: SriramAG

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

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

Image:Transparency-50.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Transparency-50.png *License*: unknown *Contributors*: SriramAG

Image:Transparency-100.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Transparency-100.png *License*: unknown *Contributors*: SriramAG

Image:Transparency-0.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Transparency-0.png *License*: unknown *Contributors*: SriramAG

Image:Display-1bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-1bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-2bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-2bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-4bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-4bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-8bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-8bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-12bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-12bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-16bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-16bpp.png *License*: unknown *Contributors*: SriramAG

Image:Display-rgb.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-rgb.png *License*: unknown *Contributors*: SriramAG

Image:Display-argb.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-argb.png *License*: unknown *Contributors*: SriramAG

Image:Display-rgba.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-rgba.png *License*: unknown *Contributors*: SriramAG

Image:Display-24bpp-packed.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-24bpp-packed.png *License*: unknown *Contributors*: SriramAG

Image:Display-uyvy.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-uyvy.png *License*: unknown *Contributors*: SriramAG

Image:Display-yuyv.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Display-yuyv.png *License*: unknown *Contributors*: SriramAG

Image:V4l2_apps_flow.png *Source*: http://processors.wiki.ti.com/index.php?title=File:V4l2_apps_flow.png *License*: unknown *Contributors*: SriramAG

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

Image:omap35x_capture_MC_overview.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x_capture_MC_overview.png *License*: unknown *Contributors*: Hvaibhav

Image:Omap35x_capture_input.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x_capture_input.png *License*: unknown *Contributors*: SriramAG

Image:omap35x-isp-MC-arch.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Omap35x-isp-MC-arch.png *License*: unknown *Contributors*: Hvaibhav

Image:Am3517_capture_tvp5146_inputs.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Am3517_capture_tvp5146_inputs.png *License*: unknown *Contributors*: SriramAG

Image:Am3517_capture_overview.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Am3517_capture_overview.png *License*: unknown *Contributors*: SriramAG

Image:usb-01.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-01.png *License*: unknown *Contributors*: AjayGupta

Image:usb-02.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-02.png *License*: unknown *Contributors*: AjayGupta

Image:usb-msc.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-msc.png *License*: unknown *Contributors*: AjayGupta, SanjeevPremi

Image:usb-hid.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-hid.png *License*: unknown *Contributors*: AjayGupta, SanjeevPremi

Image:usb-iso.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-iso.png *License*: unknown *Contributors*: AjayGupta, SanjeevPremi

Image:usb-gadget.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-gadget.png *License*: unknown *Contributors*: Hvaibhav

Image:usb-cdc.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Usb-cdc.png *License*: unknown *Contributors*: Hvaibhav

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

File:AM3517-S11-switch-conf.PNG *Source*: http://processors.wiki.ti.com/index.php?title=File:AM3517-S11-switch-conf.PNG *License*: Public Domain *Contributors*: Hvaibhav

File:x-loader-AM3517-NOR-Flash-Support2.tar.gz *Source*: http://processors.wiki.ti.com/index.php?title=File:X-loader-AM3517-NOR-Flash-Support2.tar.gz *License*: unknown *Contributors*: SanjeevPremi

File:u-boot-AM3517-NOR-Flash-Support.tar.gz *Source*: http://processors.wiki.ti.com/index.php?title=File:U-boot-AM3517-NOR-Flash-Support.tar.gz *License*: unknown *Contributors*: Hvaibhav

File:Linux-AM3517-NOR-Flash-Support.tar.gz *Source*: http://processors.wiki.ti.com/index.php?title=File:Linux-AM3517-NOR-Flash-Support.tar.gz *License*: unknown *Contributors*: Hvaibhav

File:Linux-AM3517-NOR-Flash-defconfig.tar.gz *Source*: http://processors.wiki.ti.com/index.php?title=File:Linux-AM3517-NOR-Flash-defconfig.tar.gz *License*: unknown *Contributors*: Hvaibhav

File:u-boot-AM3517-NOR-Flash-Support-04.02.00.06.tar.gz *Source*: http://processors.wiki.ti.com/index.php?title=File:U-boot-AM3517-NOR-Flash-Support-04.02.00.06.tar.gz *License*: unknown *Contributors*: Hvaibhav

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.