TI81XX VPSS Video Driver User Guide

DM81xx AM38xx Video Driver User Guide

Linux PSP

IMPORTANT

TI81xx refers to DM816x, DM814x and DM813X.

Video Driver

Read This First

This section show the major updates since last release, please review this section before any SW activities.

- This User Guide is for PSP 04.04.00.01
- This User Guide supports DM813X(DM385 called in other places) family.
- VPSS capture driver function is added. Please refer VPSS Captuer Driver user Guide.
- VPSS capture driver requires IO EXPANSION Daughter Card. If Daugther card is not connected with base EVM, capture function does not work and below error message is printed out when loading vpss.ko which can be ignored for the display function.

```
2C: Transfer failed at vps_ti814/6x_select_video_decoder 184 with error code: -121
```

- The build dependency of syslink has been removed.
- The build procedures are changed as below, the defconfig is up to which platform was used :

```
$make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- ti8168_evm_defconfig
```

or

```
   $make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- ti8148_evm_defconfig
```

or

```
   $make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- dm385_evm_defconfig
```

```
   $make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- uImage
   $make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- modules
```

- The addresses of notifyk.vpssm3_sva and vpss.sbufaddr mentioned in this userguide apply only to the M3 firmware inside PSP package. For the M3 firmware from TI SDK package, please check SDK document to get right address for notifyk.vpssm3_sva and vpss.sbufaddr

- The usage of loading M3 firmware described in this userguide does not apply for M3 firmware coming from TI SDK packagae. Please check SDK document to get details information.
- For the DM814X/AM387X platform, two different M3 firmwares, ti814x_hdvpss_512M.xem3 and ti814x_hdvpss_1G.xem3 are provided to support 512M DDR3 EVM and 1GB DDR3 EVM. The 512M.xem3 can be used on 512MB PG1.0 EVM only while 1G.xem3 can only be used over 1GB DDR3 PG2.0 EVM.
- If ti814x_hdvpss_512M.xem3 is planned to be used over PG2.1 silicon, DMM_LISA_MAP_0/1/2/3 must be reconfigured with PG1_0_DMM_LISA_MAP_0/1/2/3 in the u-boot. The default DMM_LISA_MAP_0/1/2/3 setting does not support ti814x_hdvpss_512M.xem3.
- A new bootargs parameter is required to use notify driver in kernel. This address 0xA00000000 is valid for DM816X/AM389X, DM814X/AM387x(PG2.1 EVM) and DM813x platforms and 0x8DB00000 is for DM814X/AM387X(PG1.0 EVM) only.

```
$ notifyk.vpssm3_sva=0xA0000000
```

for ti814x_hdvpss_512M.xem3

```
  notifyk.vpssm3_sva=0x8DB00000
```

- M3 firmware loader program is changed from procmgrapp to slaveloaer

```
$ ./slaveloader startup VPSS-M3 ti816x_hdvpss.xem3
```

or

```
 $ ./slaveloader startup VPSS-M3 ti813x_hdvpss.xem3
```

or

```
 $ ./slaveloader startup VPSS-M3 ti814x_hdvpss_1G.xem3
```

or

```
 $ ./slaveloader startup VPSS-M3 ti814x_hdvpss_512M.xem3
```

- The sharing buffer address is changed to 0xA0200000 and it is valid for DM816X/AM389X, DM814X/AM387x(PG 2.1 EVM) and DM813X platforms. The address is 0x8DE00000 for PG1.0 DM814X/AM387X EVM.
- The On-Chip HDMI Driver is changed to ti81xxhdmi.ko and hdmi_mode parameter is no long valid, please check on-chip HDMI user guide.
- The following sysfs entries have been added to support EDID(read only)

```
$ /sys/devices/platform/vpss/display0/edid
```

- The following sysfs entries have been removed

```
$ /sys/devices/platform/vpss/system/automode
 $ /sys/devices/platform/vpss/system/pllclks
```

- The following syfs entries have been modified to support polarity of the DE(ACT_VID)/FID/HS/VS signal

```
$ /sys/devices/platform/vpss/display0/output
 $ /sys/devices/platform/vpss/display1/output
```

- Supports customized display resolution over On-Chip HDMI interface.
- cropping, positioning is supported at /dev/video3 node
- V4L2 display nodes are /dev/video1, /dev/video2, /dev/video3
- V4L2 capture nodes are /dev/video0, /dev/video4, /dev/video5 and /dev/video6.
- Above nodes are created only when ti81xxvo.ko is loaded followed by ti81xxvin.ko
- V4L2 capture streaming is supported only on /dev/video0 node with TVP7002 as decoder.

Installation Guide

Platform Buffer Address

VPSS-M3 Firmware	Notify Buffer Default Value(notifyk.vpssm3_sva)	Sharing Buffer Default Value(sbufaddr)
ti816x_hdvpss.xem3	0xA0000000	0xA0200000
ti814x_hdvpss_512M.xem3	0x8DB00000	0x8DE00000
ti814x_hdvpss_1G.xem3	0xA0000000	0xA0200000
ti813x_hdvpss.xem3	0xA0000000	0xA0200000

Introduction

This is the TI81XX Linux VPSS(Fbdev and V4L2 Display and Capture) Driver user installation guide. This documents the steps to get the VPSS Driver worked up on TI81XX based EVM(Simulator is not tested). This document details the steps in load syslink, M3 BIOS VPSS firmware and VPSS symbol. This driver has the following dependencies outside of Kernel.

- TI81XX PSP 04.04.00.01 release for TI814x, DM837x, TI816x, DM389x, DM385x and DM813x
- Syslink_02_10_02_17 release (slaverloader)
- HDVPSS 01.00.01.37 release

Note: Only the above mentioned version of the components are verified and tested.

Preliminary Work

- Build the Linux Uboot and Kernel following the installation guide of TI816x PSP 04.00.02.14 release or TI814X PSP 04.01.00.08 release
- M3 BIOS VPSS Firmware(ti816x_hdvpss.xem3, ti814x_hdvpss_1G/512M.xem3 and ti813x_hdvpss.xem3) is loaded by the slaveloader, which is the user space program from syslink.
- M3 BIOS VPSS Firmware(ti816x_hdvpss.xem3, ti814x_hdvpss_1G/512M.xem3 and ti813x_hdvpss.xem3) is the software running over the VPSS M3 Processor.

Build Linux VPSS, Fbdev, V4L2 Display and V4L2 Capture Drivers

Though build dependency over syslink has been removed, VPSS driver still relies on the slaveloader to load M3 firmware, VPSS, Fbdev, V4L2 Display and V4L2 capture Drivers supports dynamic build only. The pre-build uImage inside of PSP release package supports the prebuit VPSS/Fbdev/V4L2 modules.

NOTE:

1. Applications are not required to rebuild the uImage or VPSS/FBDev/V4L2 module unless VPSS/FBDev/V4L2 drivers are changed.

- Enable the notify in menuconfig(by default this option is on)

```
$ make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- ti8168_evm_defconfig
$ make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- uImage
$ make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- menuconfig
```

select Device Drivers

```
   Userspace binary formats  --->
   Power management options  --->
   [*] Networking support  --->
 " Device Drivers  --->    "
   CBUS support  --->
   File systems  --->
```

Select Sys_Link

```
  [ ] Auxiliary Display support  --->
   < > Userspace I/O drivers  --->
  [ ] Staging drivers  --->
  "[*] Sys_Link  --->    "
```

Select SysLink Notify

```
 --- Sys_Link
" <*>   SysLink Notify  "
```

- Enable the VPSS and Fbdev in menuconfig

select Device Drivers

```
   Userspace binary formats  --->
   Power management options  --->
   [*] Networking support  --->
 " Device Drivers  --->    "
   CBUS support  --->
   File systems  --->
```

select graphics support:

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

Select TI81XX Video Processing Subsystem (EXPERIMENTAL)

```
   < > Fujitsu MB862xx GDC support
   < > E-Ink Broadsheet/Epson S1D13521 controller support
   < > OMAP frame buffer support (EXPERIMENTAL)
 " <M> TI81XX Video Processing Subsystem (EXPERIMENTAL)  ---> "
   [ ] Backlight & LCD device support  --->
```

Setup the VPSS configuraiton.

```
 --- TI81XX Video Processing Subsystem (EXPERIMENTAL)
   (50)  VRAM size (MB)
   [*]   Debug support
    <>   TI81XX frame buffer support (EXPERIMENTAL)  --->
```

select TI81XX frame buffer support (EXPERIMENTAL)

```
 --- TI81XX Video Processing Subsystem (EXPERIMENTAL)
   (50)  VRAM size (MB)
   [*]   Debug support
   " <M>   TI81XX frame buffer support (EXPERIMENTAL)  ---> "
```

```
--- TI81XX frame buffer support (EXPERIMENTAL)
   [*]   Debug support for TI81XX FB
   (3)   Number of framebuffers
```

- Enable V4L2 Display Driver in Menuconfig

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 ***
[ ]   Media Controller API (EXPERIMENTAL)
"<*>   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  --->
    ...
    ...
```

Select TI81XX V4L2-Display driver

```
[ ]   Autoselect pertinent encoders/decoders and other helper chi
      Encoders/decoders and other helper chips  --->
"<M>   TI81XX V4L2-Display driver"
< >   CPiA2 Video For Linux
< >   Philips SAA7134 support
```

- Enable V4L2 Capture Driver in Menuconfig

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 ***
[ ]   Media Controller API (EXPERIMENTAL)
"<*>   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  --->
    ...
    ...
```

Select TI81XX V4L2-Capture driver

```
[ ]   Autoselect pertinent encoders/decoders and other helper chi
      Encoders/decoders and other helper chips  --->
<>   TI81XX V4L2-Capture driver
"<M>   TI81XX V4L2-Capture driver"
< >   CPiA2 Video For Linux
< >   Philips SAA7134 support
```

- Build module for the drivers

```
$ make ARCH=arm CROSS_COMPILE=PATH_TO_TOOLCHAIN/bin/arm-none-linux-gnueabi- modules
```

- vpss.ko(VPSS library) is generated under drivers/video/ti81xx/vpss， ti81xxfb.ko(VPSS fbdev driver) is generated under drivers/video/ti81xx/ti81xxfb, ti81xxvo.ko(VSPSS V4L2 display driver) is generated under drivers/media/video/ti81xx and ti81xxvin.ko(VSPSS V4L2 capture driver) is generated under drivers/media/video/ti81xx

Prerequisites

The following files will be available after the above steps have been executed:

1. syslink kernel driver module, referred as "syslink.ko"
2. loader user space program, referred as "slaveloader"
3. M3 BIOS Firmware binary, referred as "ti816x_hdvpss.xem3/ti814x_hdvpss_512M.xem3/ti814x_hdvpss_1g.xem3/ti813x_hdvpss.xem3"
4. VPSS kernel driver module, referred as "vpss.ko"
5. FBDev kernel driver module, referred as "ti81xxfb.ko"
6. V4L2 display kernel module, referred as "ti81xxvo.ko"

Note: Item 1-3 can be downloaded from PSP download Link.

Load VPSS, Fbdev, and V4L2 Display Driver Modules

- After seeing Linux booting log at Hyper Terminal or Tera Term, type "root" to log int.

```
$ cd to the folder where those Prerequisites are stored in the NFS or RamDisk
```

- Note:

The addresses of notifyk.vpssm3_sva and vpss.sbufaddr mentioned in this userguide apply only to the M3 firmware inside PSP package. For the M3 firmware from TI SDK package, please check SDK document to get right address for notifyk.vpssm3_sva and vpss.sbufaddr
The usage of loading M3 firmware described in this userguide does not apply for M3 firmware coming from TI SDK packagae. Please check SDK document to get details information.

- **Load Syslink Module**

```
$ insmod syslink.ko
```

- **Load VPSS M3 Firmware(dependent on hardware platform)**

```
$ ./slaveloader startup VPSS-M3 ti816x_hdvpss.xem3
```

or

```
$ ./slaveloader startup VPSS-M3 ti814x_hdvpss_512M.xem3
```

or

```
$ ./slaveloader startup VPSS-M3 ti814x_hdvpss_1G.xem3
```

or

```
$ ./slaveloader startup VPSS-M3 ti813x_hdvpss.xem3
```

- **Load VPSS Module**

```
 $ insmod vpss.ko
```

- **Load FBDev Module**

```
$ insmod ti81xxfb.ko
```

- **Load V4L2 Display Module**

```
$ insmod ti81xxvo.ko
```

- Now Fbdev, and V4L2 display module are loaded and application can be executed from here.

NOTE:

```
HDMI Kernel module is required to view the output from On-Chip HDMI, see next section.

Application need develop their own software component to support the external display device connected with DVO2 VENC.

Add debug=1 to enable the debug function of vpss.ko,ti81xxfb.ko, and ti81xxvo.ko

Application need assign the memroy size when loading ti81xxfb.ko if accessing /dev/fb1 and /dev/fb2 nodes

   $ insmod ti81xxfb.ko vram=0:XXM,1:YYM,2:ZZM
```

Load On-Chip HDMI Module

On-Chip HDMI hardware is controlled through another kernel driver, which should be loaded after vpss.ko is loadded. Please refer | HDMI user guide [2]for details.

- Load HDMI Module

```
$ insmod ti81xxhdmi.ko
```

Introduction

Video Processing Sub-System hardware integrates theee graphics pipeline, five video pipelines, two capture ports and 4 video compositors. Video compositors are connected to four VENCs to support various output format and mode.

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

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

References

1. Video for Linux Two Home Page [http://linux.bytesex.org/v4l2/ [3]]
2. Video for Linux Two API Specification [[http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf [4]]

Acronyms & Definitions

Display Driver: Acronyms

Acronym	Definition
V4L2	Video for Linux Two
VPSS	Vidoe Processing SubSystem
HDMI	Hight Definition Multimedia Interface
HDCOMP	High Definition Component
DVO	Digital Video Output
VENC	Video Encoder
NTSC	National Television System Committee
PAL	Phase Alternating Line
SD	Standard Definition
SDK	Software Development Kit
VCOMP	Video Compositor
COMP	Compsoitor
EDID	Extended display identification data
VIP	Video Input Port

Display Port and VENC name

Display Port Name	Type	VENC Name	Port Num for Pin	Sysfs Name
HDMI/DVO1	Digital	HDMI	VOUT1	Display0
DVO2	Digital	DVO2	VOUT0	Display1
SD	Analog	SD	N/A	Display2
HDCOMP *	Analog	HDCOMP	N/A	Display3

Note*: HDCOMP is not available on DM814X/AM387X Platform.

Hardware Overview

The video processing subsystem provides the functions to display a video frame from the memory frame buffer to external display device, such as LCD,TV; or capture a video frame from external camera to memory. The video processing subsystem integrates the following main elements

- Graphics processing modules
- Video Capture processing modules
- Video display processing modules
- Video Compositor modules

Features

The VPSS driver supports the following features:

- Supports 3 Graphics pipelines through FBDev interface
- Supports 3 Video pipelines through V4L2 interface
- Supports 2 VIP capture instances of VPSS through V4L2 interface.
- Each of the VIP capture instances can either behave as 2 8-bit port or 1 16/24 bit port.
- Supported color formats on Graphics pipeline: RGB565, ARGB1555, RGBA5551, ARGB4444, RGBA4444, ARGB6666,RGBA6666,RGB888, ARGB8888 and RGBA8888, BMP 1/2/4/8.
- Support Color formats on Video pipeline; 422 Interleave, 420 semi-planar and 422 semi-planar.
- Supported color formats on VIP port are YUV422 interleaved, YUV422 semi-planar, YUV420 semi-planar and RGB888.
- Supports cropping and scaling for YUV color formats on V4L2 capture driver. Only downscaling is supported.
- Configuration of display parameters such as height and width of display graphics, position of the display graphics, bits-per-pixel etc.
- Supports setting up of Graphics pipelines/Video pipelines and VENCs destination(HDMI, HDCOMP, DVO2 or SD) through syfs interface.
- Supports buffer management through memory mapped (mmaped) on both FBDev and V4l2.
- Supports user pointer buffer exchange for application usage on V4L2 only.
- Supports global alpha blending on RGB format, pixel alpha blending on ARGB/RGBA format and palette alpha blending on BMP formats.
- Supports colorkeying on graphics pipelines through FBDev.
- Supports boundbox on graphcis pipelines through FBDev
- Supports display priority on graphics pipelines through FBDev
- Supports scaling on graphics pipeline through FBDev
- Supports stenciling on graphics pipeline through FBDev
- supports global alpha blending on video pipeline through V4L2
- Supports colorkeying on video pipeline through V4L2
- Supports various display resolutions on HDMI/HDCOMP VENCs through sysfs
- Supports various Video PLL frequency through sysfs
- Supports various VENC clock source through sysfs
- Supports various output for VENC through sysfs.
- Supports reshuffle display order through sysfs
- Supports customized timing for VENC through sysfs

Architecture

This chapter describes the Driver Architecture and Design concepts

Driver Architecture

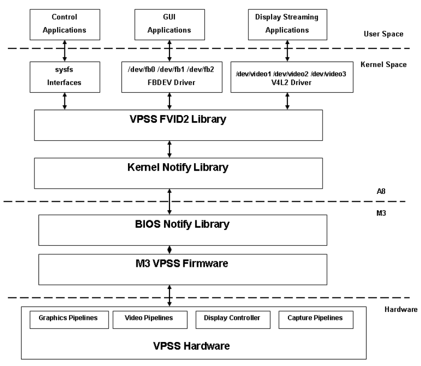
TI81XX Video Processing hardware integrates three graphics pipeline, five video pipelines, 2 capture ports and 4 video compositors. Video compositors are connected to four VENCs to support various output format and mode.

The primary functionality of the VPSS driver is to provide interfaces to user level applications and management to video processing subsystem hardware.

This includes, but is not limited to:

- GUI rendering through graphics pipelines.

- Video rendering through video pipelines.
- Connecting each of three graphics pipelines to four compositors so the display layer is presented on the selected output path.
- Connecting each of three video pipelines to four compositors so the display layer is presented on the selected output path.
- Image/Video processing(alpha blending, colorkeying, cropping)



Video Processing Subsystem Architecture

Software Design Interfaces

Above figure (Video Processing Subsystem Architecture) shows the major components that makes up the VPSS software sub-system

- **M3 VPSS Firmware**

This is a firmware running over processor Cotex M3 controlling the Video Processing subsystem hardware.

- **Syslink Library**

This is a funcional layer controlling the inter-processor communction between Cotex A8 and Cotex M3.

- **VPSS FVID2 Library**

This is a HAL/functional layer controlling the Firmware running over the M3. It exposes the number of APIs controlling the video compositors, VENCs, graphics/video pipelines to the user interface drivers like V4L2 and FBDEV. It translates the V4L2/Fbdev data structures and ioctl to FVID2 data structures and command, and call the Linux Syslink IPC notify function to pass FVID2 data structures and commands to Firmware running over the M3 processor.

- **SYSFS interfaces**

The SYSFS interfaces are mostly used as the control path for configuring the VPSS parameters which are common between FBDEV and V4L2 like the venc modes etc. It is also used for switching the output of the pipeline to different video compositors.

- **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, /dev/fb1 and /dev/fb2 as the device nodes. Application can open these device nodes 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.

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.

- **FBDEV Driver**

The driver will expose three software channels (/dev/fb0, /dev/fb1, /dev/fb2) for the graphics pipeline.

```
/* 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);

- **V4L2 Driver**

The driver will expose three software channels (/dev/video1, /dev/video2 and /dev/video3), one for each video pipeline. All of these channels supports only blocking mode of operations.

```
/* 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);
```

Command Line arguments

Command Line Arguments

Argument	Description
vram	Total framebuffer memory reserverd for the Fbdev driver

Following example shows how to specify total framebuffer size as boot time argument

```
setenv bootargs console=ttyO2,115200n8 mem=128M noinitrd root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp vram=50M
```

FBDEV Driver

FBDEV driver supports set of command line argument for size of vram and debug option. These argument are only specified at the time of inserting the driver since FBDev driver only supports dynamic build.

Below is the list of arguments which Fbdev driver supports -

Fbdev Driver Command Line Arguments

Argument	Description
vram	VRAM allocated memory for a framebuffer, user can individually configure VRAM buffers for each plane/device node
debug	Enable Fbdev driver debug messaging

Following example shows how to specify size of framebuffer:

```
$ insmod ti81xxfb.ko vram=0:16M,1:16M,2:6M
```

Following example shows how to enable debug of Fbdev:

```
$ insmod ti81xxfb.ko debug=1
```

V4L2 Driver

V4L2 driver supports set of command line argument for default number of buffers, their buffer size, and debug option for both the video pipelines. These argument are only specified at the time of inserting the driver since V4L2 driver only supports dynamic build.

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
video3_numbuffers	Number of buffers to be allocated at init time for Video3 device
video1_bufsize	Size of the buffer to be allocated for video1 device, no bigger than 4MB
video2_bufsize	Size of the buffer to be allocated for video2 device, no bigger than 4MB
video3_bufsize	Size of the buffer to be allocated for video3 device, no bigger than 4MB
video3_memtype	memory type for video 3 device(0:non-tiled memory, 1:tiled memory). video1 and video2 device supports non-tiled memory only
debug	Enable debug messaging

Insert the dynamically built module with following parameters:

```
# insmod ti81xxvo.ko video1_numbuffers=3 video2_numbuffers=3 video1_bufsize=4147200 video2_bufsize=4147200
```

Following example shows how to enable debug of V4L2:

```
$ insmod ti81xxvo.ko debug=1
```

VPSS Driver

VPSS driver supports set of command line argument for mode, tied venc, venc clock source. These argument are only specified at the time of inserting the driver since VPSS only supports dynamic build.

Below is the list of arguments which VPSS driver supports -

VPSS Driver Command Line Arguments

Argument	Description
clksrc	clock source for hdmi/dvo2/hdcomp VENC. By default, hdmi and dvo2 VENC share the d clock input while hdcomp uses the a clock input. hdmi:dclk/dclkdiv2/dclkdiff,hdcomp:dclk/dclkdiv2/dclkdiff/aclk/aclkdiv2/aclkdiff,dvo2:dclk/dclkdiv2/dclkdiff/aclk/aclkdiv2/aclkdiff
debug	Enable VPSS Driver debug messaging
i2c_mode	choose the control source for the external i2c-based video devices, by default it is A8. when i2c_mode is set to 1, M3 controls external i2c-based video devices and it requires a special M3 fimrware, which is not part of PSP release.
mode	Default video mode for specified VENC(hdmi, hdcomp,dvo2,sd) if mode is not assigned during the boot time, driver uses the following mode: hdmi: 1080p-60, hdcomp: 1080i-60, dvo2:1080p-60, sd:ntsc
tiedvencs	OR value of the VENCs(1:hdmi venc, 2:hdcomp venc, 4:dvo2 venc, 8:sd venc)
sbufaddr	Physical DDR address of the sharing buffer between Processor A8 and M3. This sharing buffer is used as payload to store FVID2 structurs communicated bewteen A8 and M3. The buffer must be non-cacheable DDR from Ducati side and should not be accessed by any other software components. This buffer should be predefined in the M3 Fimrware.
sbufsize	Size of the sharing buffer between processor A8 and M3.
timeout	timeout value(ms) to be set in driver to determine how long driver should wait the response from M3. If it is set to zero, driver waits infinitely. By default, this is set to 2000 ms

NOTE: In TI81xx Hardware platform, there are two independent HD clock sources(d clock and a clock) with three HD VENCs. Therefore two of three VENCs must share the same clock source. The argument of clksrc is designed to achieve this function.

Following example shows how to specify HDMI VENC mode as module argument:

```
$ insmod vpss.ko mode=hdmi:1080p-60
```

Following example shows how to tie hdmi(1) venc and dvo2(4) venc together as module argument:

- **NOTE:** Application should make sure the tied vencs have the same timing. VPSS driver does not check for this.

```
$ insmod vpss.ko tiedvencs=5
```

Following example shows how to set up the sharing buffer address

- **NOTE:** If sbufaddr is not set, driver sets buffer base address to 0xA0200000 .

```
$insmod vpss.ko sbufaddr=0xA0200000
```

Following example shows how to set up the sharing buffer size. This size can not be over 2MB.

- **NOTE:** If sbufsize is not set, driver sets buffer size equal to 2MB.

```
$insmod vpss.ko sbufsize=1024K
```

Below is the list of arguments of clksrc supports

VPSS Driver Clock Source Arguments

Argument	Description
dclk	clk1x/dvo1_clk/dvo2_clk/vbi_clk_hd uses hd_venc_d_clk as clock source
dclkdiv2	clk1x/dvo1_clk/dvo2_clk/vbi_clk_hd used hd_venc_d_clk/2 as clock source
dclkdiff	clk1x uses hd_venc_d_clk/2 as clocksource, while dvo1_clk/dvo2_clk/vbi_clk_hd used hd_venc_d_clk as clock source
aclk	clk1x/dvo2_clk/vbi_clk_hd uses hd_venc_a_clk as clock source
aclkdiv2	clk1x/dvo2_clk/vbi_clk_hd uses hd_venc_a_clk/2 as clock source
aclkdiff	clk1x uses hd_vencc_a_clk/2 as clock source while dvo2_clk/vbi_clk_hd uses hd_venc_a_clk as clock source

Following example show how to choose a clock as the source clock of DVO2 VENC.

```
$ insmod vpss.ko clksrc=dvo2:aclk
```

Following example show how to enable debug of VPSS driver

```
$ insmod vpss.ko debug=1
```

Buffer Management

FBDEV Driver

Memory requirement for FBDEV driver

Driver	FB0	FB1	FB2
Fbdev Driver	A single buffer of size 1920*1080*4*2 bytes, can be changed through bootargs	0 bytes if app does not assign size for this node through bootargs	0 bytes if app does not assign size for this node through bootargs

FBDEV driver supports only memory mapped buffers. Driver allocates one physically contiguous buffers for each node, which can support up to 1920x1080 resolution 32 bpp format. By default, driver only allocates the memory for FB0(dev/fb0) node. Application need set the memory size if accessing /dev/fb1 and /dev/fb2 when loading FBDev modules.

Set Frame Buffer size for FB Nodes

```
$ insmod ti81xxfb.ko vram=0:XXM,1:YYM,2:ZZM
```

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

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("FBIOGET_FSCREENINFO\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);

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;`

if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0) {

```
   printf("FBIOGET_VSCREENINFO\n");
   close(fd);
   exit(0);
```

}

printf("Resolution = %dx%d\n", var.xred, var.yres); printf("bites per pixel = %d\n", var.bpp);

Mapping Kernel space address to user space `mmap` 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);

V4L2 Driver

Memory requirement for V4L2 driver

Driver	video1	video2	video3
V4L2 Driver	three buffers of size 4MB each, can be changed by module parameters.	three buffers of size 4MB each, can be changed by module parameters..	three buffers of size 720x576*2 bytes each, can be changed by module parameters.

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.

NOTE: Memoy Mapped buffer mode only support non-tiled memmory allocation with MAX size 4MB. User pointer buffer mode must be use if applicaiton want to use either tiled memory or buffer size bigger than 4MB.

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

- **Allocating Memory**

`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.

`/* 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;
```

`}`

- **Getting physical address**

`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*/`

- **Mapping Kernel space address to user space**

`mmap`

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 */`

Transparency Keying

The encoded pixel color value is compared to the application defined transparency color key(RGB888) based the LSB bits mask set by the application to determine whether the pixel is to be transparent or not. There are four types of the bit mask offered by the hardware: No Masking, whole pixel color value is compared to the transparency color key; Mask[0]: MSB[7:1] of the encoded pixel color value is compared to the corresponding 7 bits of transparency color key; MASK[1:0]: MSB[7:2] of the encoded pixel color value is compared to the corresponding bits of transparency color key; MAKS[2:0]: MSB[7:3] of the encoded pixel color value is compared to the corresponding bits of transparency color key.

User can enable/disable color key function, set color keying and mask type through Fbdev Driver private IOCTL interface, V4L2 IOCTL or Sysfs. FBDev is used to configure the transparency of graphics pipeline while V4L2/Syfs is used to configure the transparency of video pipeline.

Using fbdev IOCTL

- Enable Transparency Keying:

`struct ti81xxfb_region_params regp;`

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");

```
        close(fd);
```

exit(1); }

/* Enable the Transparency Keying */ regp.transen = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {

```
    perror ("TIFB_SET_PARAMS.\n");
    close(fd);
    exit(1);
```

}

- Disable Transparency Keying:

`struct ti81xxfb_region_params regp;`

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");

```
        close(fd);
```

exit(1); }

/* Disable the Transparency Keying */ regp.transen = TI81XXFB_FEATURE_DISABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {

```
    perror ("TIFB_SET_PARAMS.\n");
    close(fd);
    exit(1);
```

}

- Set Transparency Keying Value(RGB888):

`struct ti81xxfb_region_params regp; u8 key_rgb888; /*RGB888 format*/`

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");

```
        close(fd);
```

exit(1); } /* Set the Transparency Keying */ regp.transcolor = key_rgb888; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {

```
    perror ("TIFB_SET_PARAMS.\n");
    close(fd);
    exit(1);
```

}

- Set Transparency Keying Mask Type:

`struct ti81xxfb_region_params regp; struct ti81xxfb_transparancy_type trans_type;`

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");

```
       close(fd);
```

exit(1); }

/* Set the Transparency Keying Mask type */ regp.transtype= trans_type; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {

```
   perror ("TIFB_SET_PARAMS.\n");
   close(fd);
   exit(1);
```

}

Using V4L2 IOCTL

- **NOTE:**: /dev/video3 does not support transparency.
- Enable the transparency 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 Transparency 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);
   }
```

`}`

- Set the Transparency Keying value(RGB888,).

`struct v4l2_format fmt; u8 chromakey = R>>16 | G>>8 | B; /* Chromakey is RGB888 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) { }`

- Get the Transparency Keying 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);`

Using Sysfs

Sysfs is only valid for video devices(not for graphcis devices).

- **NOTE:** /dev/video3(/sys/devices/platform/vpss/video2) does not support transparency.
- Enable/Disable the Transparency keying

```
echo 0/1 > /sys/devices/platform/vpss/video#/trans_key_enabled
       where 0/1: 0 - > disable, 1 - > enable
       #: 0-1 video device
```

- Set the Transparency Keying Value (RGB888 format, hex only)

```
echo  0xRRBBGG> /sys/devices/platform/vpss/video#/trans_key_value
      where #: 0-1 video device
```

- Set the Transparency Keying Mask type

number of LSB bits to maks when checking for pixel color keying

```
echo  0/1/2/3 > /sys/devices/platform/vpss/video#/trans_key_type
       where 0: no maksing, 1: mask 1 LSB bit,2: mask 2 LSB bits;3: mask 3 LSB bits
       #: 0-1 video device
```

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.

Both Graphics pipelines and Video pipelines of the VPSS are capable of supporting alpha blending. Graphics pipeline supports three types of alpha blending, global, palette and pixel alpha blending while Video pipeline only support global blending. RGB formtas of graphics pipeline support global alpha blending, BMP formats of graphcis pipeline support palette alpha blending, ARGB and RGBA formats of 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.

FBDev IOCTL

- Disable alpha blending

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");
```

```
        close(fd);
```

```
exit(1); } /*Disable Alpha Blending*/ regp.blendtype = TI81XXFB_BLENDING_NO; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
   perror ("TIFB_SET_PARAMS.\n");
   close(fd);
   exit(1);
```

```
}
```

- **Global Alpha blending**

User can configure global alpha value either through FBDEV ioctl.

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

```
struct ti81xxfb_region_params regp; u8 alpha; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");
```

```
        close(fd);
```

```
exit(1); } /*Set Global Alpha Blending*/ regp.blendtype = TI81XXFB_BLENDING_GLOBAL; regp.blendalpha = alpha; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
   perror ("TIFB_SET_PARAMS.\n");
   close(fd);
   exit(1);
```

```
}
```

- **Pixel Based Alpha blending**

FBDEV driver supports alpha blending through ARGB/RGBA pixel format

Below programlisting shows how to set the pxiel alpha value for graphics pipeline.

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0)
{ perror("TIFB_GET_PARAMS\n");
```

```
       close(fd);
```

```
exit(1); } /*Set Pixel Alpha Blending*/ regp.blendtype =
TI81XXFB_BLENDING_PIXEL; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
   perror ("TIFB_SET_PARAMS.\n");
   close(fd);
   exit(1);
```

```
}
```

- **Palette Based Alpha blending**

FBDEV driver supports alpha blending through CLUT data, data must be BMP format.

Below programlisting shows how to set the palette alpha value for graphics pipeline.

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0)
{ perror("TIFB_GET_PARAMS\n");
```

```
       close(fd);
```

```
exit(1); } /*Set Palette Alpha Blending*/ regp.blendtype =
TI81XXFB_BLENDING_PALETTE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
   perror ("TIFB_SET_PARAMS.\n");
   close(fd);
   exit(1);
```

```
}
```

V4L2 IOCTLs

- **NOTE:**: /dev/video3 does not support alpha blending.
- Eenable alpha blending

```
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) { }
```

- Disable alpha blending

```
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) { }
```

- Set the global alpha value

```
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) { }
```

Using Sysfs

This section is for video device only(not for graphics device)

- **NOTE:** /dev/video3(/sys/devices/platform/vpss/video2) does not support alpha blending.
- Enable/disable alpha blending

```
# echo 0/1 > /sys/devices/platform/vpssss/video#/alpha_blending_enabled

Where,
      0/1   => 0 - Disable, 1 - Enable.
      #: 0-1 Video device
```

- Set the global alpha value

```
# echo 0-255> /sys/devices/platform/vpss/video#/global_alpha

Where,
      #: 0-1 Video device
```

Scaling

FBDEV IOCTL

Graphics pipelines support 0.25x-4x scaling with a step size of 0.01. The scalar has 5tap/8phase polyphase horizontal filter and 4taps/8phase polyphase vertical filter. Meanwhile, When the target display of the graphics pipeline conncted to is in a interlaced scan mode, graphics pipeline could also performs anti-flicker filtering using the including scalar. This can be achieved by seting scaling ratio 1.

Below shows how to enable the scaling feature

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");
```

```
       close(fd);
```

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

regp.scalaren = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n");

```
       close(fd);
```

exit(1); }

Below shows how to disable the scaling feature

`struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");`

```
       close(fd);
```

`exit(1); } regp.scalaren = TI81XXFB_FEATURE_DISABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n"); exit(1); }`

NOTE
TIFB_SET_SCINFO ioctl must be called before enabling the scaling feature. See Custom_IOCTLs[5] for details.

V4L2 IOCTL

This function is not available for V4L2 Driver.

Stenciling

Graphics pipelines supports a 1-bit for each pixel of a image that has "stenciling" feature enabled. This bit is used to force the alpha value of the pixel to ZERO in order to "mask"-off the pixel(such as transparent).

Video pipelines do not support stenciling function. Below shows how to enable the stenciling feature

`struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");`

```
       close(fd);
```

`exit(1); } regp.stencilingen = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n"); exit(1); }`

Below show how to disable the stenciling feature

`struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");`

```
       close(fd);
```

`exit(1); } regp.stencilingen = TI81XXFB_FEATURE_DISABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n"); exit(1); }`

NOTE
TIFB_ALLOC and TIFB_SET_STENC ioctl must be called before enabling the scaling feature. See Section 5.1.3 for details.

Bound Box Blending

Graphics piplelines support overwriting alpha values of piels that make up a 1-pixel wide boundary box of a image with a semi-transparnt value(0x80,default) so the flickering around the edges can be minimized.

Video pipelines does not support bound box function.

Below shows how to enable boundbox feature and change the bound box alpha value `struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");`

```
       close(fd);
```

`exit(1); } regp.bbalpha = 0x40; regp.bben = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n"); exit(1); }`

Below shows how to disable boundbox feature `struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) { perror("TIFB_GET_PARAMS\n");`

```
       close(fd);
```

`exit(1); } regp.bben = TI81XXFB_FEATURE_DISABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) { perror("TIFB_SET_PARAMS\n"); exit(1); }`

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.

FBDEV Driver

In all buffer formats, blue value is always stored in least significant bits, then green value and then red value. The alpha value stored is up to the RGBA or ARGB format.

Graphics pipeline supports following buffer format: RGB565, ARGB1555, RGBA5551, ARGB4444,RGBA4444, ARGB6666, RGBA6666, RGB888, ARGB8888 and RGBA8888. Moreover graphics pipeline supports 1/2/4/8 BMP format. Buffer format can be changed in FBDEV driver by using bpp, red, green, ,blue and transparency fields of fb_vscreeninfo structure and ioctl FBIOPUT_VSCREENINFO. Application needs to specify bits per pixel and length and offset of red, green, blue and transparnecy component to let driver know the right pixel format.

Extra care must be taken by the application when dealing with 1/2/4 BMP format. Graphis pipeline support 8 different 1b BMP formats, 4 different 2b BMP formats and 2 different 4b BMP formats. By default, if bits_per_pixel was assigned to 1, TI81XXFB_BMP1_OFF0 format will be used by the driver. if bits_per_pixel was assigned to 2, TI81XXFB_BMP2_OFF0 will be used by the driver. If bits_per_pixel was assigend to 4, TI81XXFB_BMP1_L will be used by the driver. If default value is not meeting application's requirements, please use nonstd filed in the fb_vscreeninfo structure to set the right 1/2/4 BMP format. Do not forget to set nonstd to 0 when switching to different pixel format.

Following example shows how to ARGB666 bits per pixels.

`struct fb_varscreeninfo var; if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0) {`

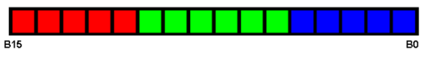
```
   perror("FBIOGET_VSCREENINFO\n");
   close(fd);
   exit(0);
```

`} var.bpp = 24; var.red.length = var.green.length = var.blue.length = var.transp.length = 4; var.transp.offset = 18; var.red.offset = 12; var.green.offset = 6; var.blue.offset = 0; if (ioctl(fd, FBIOPUT_VSCREENINFO, &var) < 0) {`

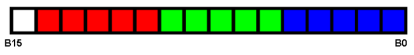
```
   perror("FBIOPUT_VSCREENINFO\n");
   close(fd);
   exit(0);
```

`}`

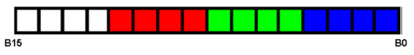
Buffer Formats



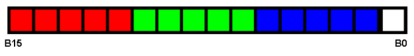
RGB565 Data Memory Organization



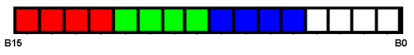
ARGB1555 Data Memory Organization



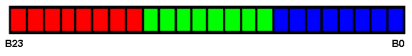
ARGB444 Data Memory Organization



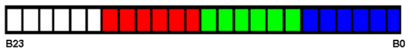
RGBA5551 Data Memory Organization



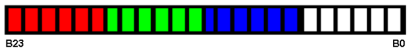
RGBA4444 Data Memory Organization



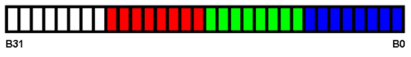
RGB888 Data Memory Organization



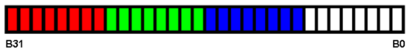
ARGB6666 Data Memory Organization



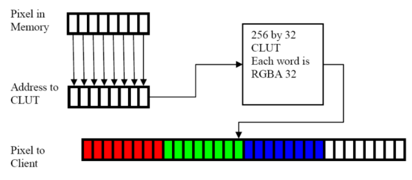
RGBA6666 Data Memory Organization



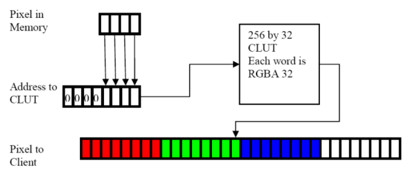
ARGB8888 Data Memory Organization



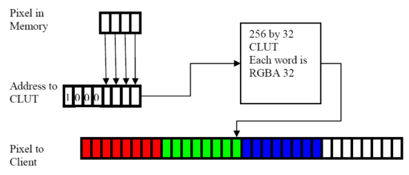
RGBA8888 Data Memory Organization



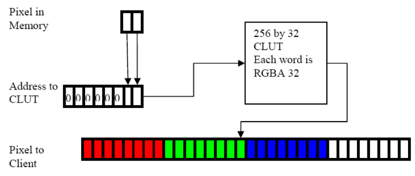
BPP8 Data Memory Organization



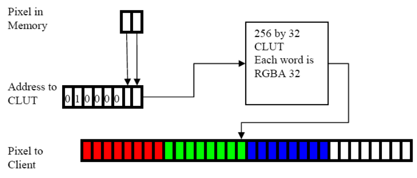
BPP4 LOWER Data Memory Organization



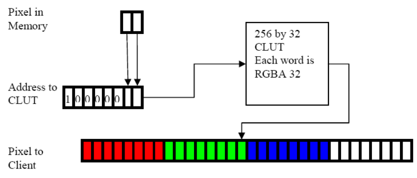
BPP4 UPPER Data Memory Organization



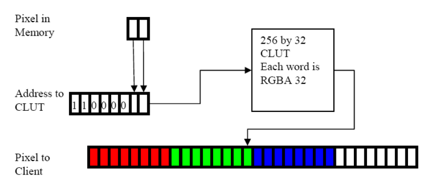
BPP2 OFFSET0 Data Memory Organization



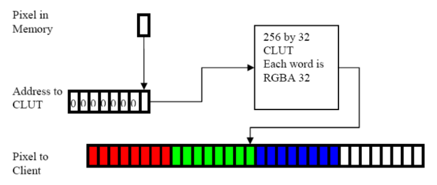
BPP2 OFFSET1 Data Memory Organization



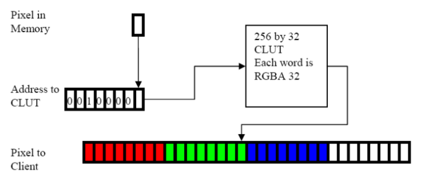
BPP2 OFFSET2 Data Memory Organization



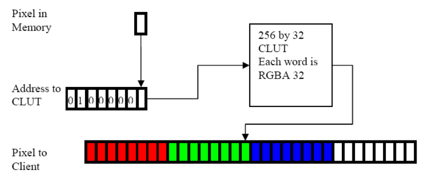
BPP2 OFFSET3 Data Memory Organization



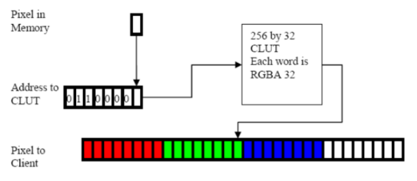
BPP1 OFFSET0 Data Memory Organization



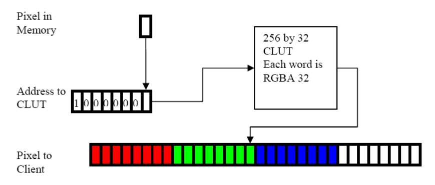
BPP1 OFFSET1 Data Memory Organization



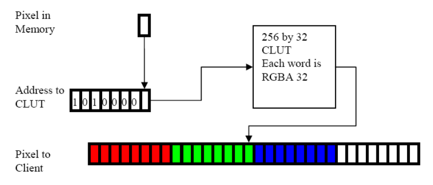
BPP1 OFFSET2 Data Memory Organization



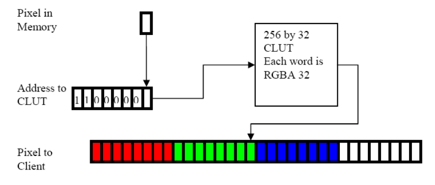
BPP1 OFFSET3 Data Memory Organization



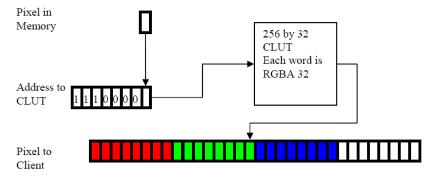
BPP1 OFFSET4 Data Memory Organization



BPP1 OFFSET5 Data Memory Organization



BPP1 OFFSET6 Data Memory Organization



BPP1 OFFSET7 Data Memory Organization

V4L2 Driver

video layer supports following buffer format: YUYV and Y/UV. The corresponding v4l2 defines for pixel format are V4L2_PIX_FMT_YUYV, V4L2_PIX_FMT_NV12, V4L2_PIX_FMT_NV16. Buffer format can be changed using VIDIOC_S_FMT ioctl with type as V4L2_BUF_TYPE_VIDEO_OUTPUT and appropriate pixel format type.

Video Deviceo Buffer Format

Buffer Format	/dev/video1	/dev/video2	/dev/video3
V4L2_PIX_FMT_YUYV	Y	Y	Y
V4L2_PIX_FMT_NV12	N	N	Y
V4L2_PIX_FMT_NV16	N	N	Y

- **NOTE:**: For the NV12/NV16 format, even the Y and UV(CbCr) components are stored separated, the buffer address of Y and UV(CbCr) component should be continuous or driver can not handle these two formats.

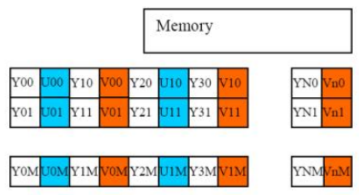
Following example shows how to change pixel format to YUYV

```
struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
    perror("VIDIOC_S_FMT\n");
    close(fd);
    exit(0);
}
```

Buffer Formats

- YUYV422 Interleaved Format

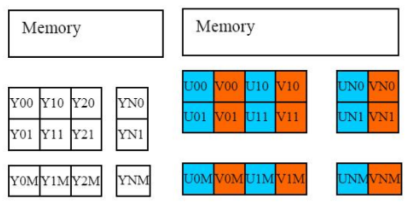
YUYV422 is expected to be packed with Y0 in the lowest byte followed by U(Cb) followed by Y1 finally V(Cr) in the upper most byte.


Memory
Y00 U00 Y10 V00 Y20 U10 Y30 V10
Y01 U01 Y11 V01 Y21 U11 Y31 V11
YN0 Vn0
YN1 Vn1
Y0M U0M Y1M V0M Y2M U1M Y3M V1M
YNM VnM

YUYV422 Interleaved Data Memory Organization

- YUV422 Semi-planar Format

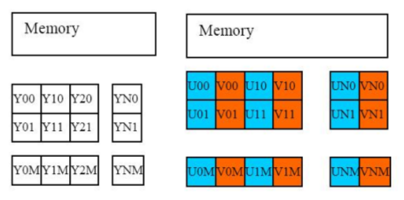
Y and UV are stored separately. UV is expected to be packed with U(Cb) in the lower byte and V(Cr) in the upper byte.



YUV422 semi-planar Data Memory Organization

- YUV420 Semi-planar Format

Y and UV are stored separately. UV is expected to be packed with U(Cb) in the lower byte and V(Cr) in the upper byte.



YUV420 semi-planar Data Memory Organization

Display Position of Graphics

FBDev IOCTL

The display position of graphics pipeline is able to changed within the resolution defined by the VENCs which graphics pipiline is connected.

Following example shows how to change display position graphics.

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0)
{ perror("TIFB_GET_PARAMS\n");
```

```
       close(fd);
```

```
exit(1); } /*Set the Display position*/ regp.pos_x = 10; regp.pos_y = 20; if
(ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
   perror ("TIFB_SET_PARAMS\n");
   close(fd);
   exit(1);
```

```
}
```

V4L2 IOCTL

The video pipelines can be connected to various output interface 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 one display interface to the onther 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 */
```

- **NOTE:**: the width and height should match the width and height of the VIDIOC_S_FMT I/O control with the V4L2_BUF_TYPE_VIDEO buffer type if cropping is no used; the width and height should match the width and height of the VIDIOC_S_CROP I/O control with V4L2_BUF_TYPE_VIDEO buffer type if cropping is used, otherwise driver reports error back to caller.

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 */
```

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 32-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[256]={0xFF, 0x00, 0x00, 0xFF}; unsigned short g[256]={0x00, 0xFF, 0x00, 0xFF}; unsigned short b[256]={0x00, 0x00, 0xFF, 0x00}; unsigned short t[256]={0xFF, 0xFF, 0xFF, 0xFF}; cmap.len = 256; cmap.red = r; cmap.green = g; cmap.blue = b; cmap.transp = t; 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. TI81XX supports 256*32(RGBT) color-map, so the len of the fb_cmap should be 256, or driver returns error.

FBIO_BLANK

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

FBIO_WAITFORVSYNC

This I/O control is used to put an application to sleep until next vertical sync interval of the display.

Supported Custom IOCTLs

TIFB_GET_PARAMS

Ioctl returns the parameters of the current graphics features.

Data Structure: `enum ti81xxfb_status {`

```
TI81XXFB_FEATURE_DISABLE = 0,
TI81XXFB_FEATURE_ENABLE
```

`};`

enum ti81xxfb_blending_type {

```
/*alpha blending disable*/
TI81XXFB_BLENDING_NO = 0,
/* global alpha blending*/
TI81XXFB_BLENDING_GLOBAL,
/* PALETTE alpha blending, the alpha value in CLUT is used for blending*/
TI81XXFB_BLENDING_PALETTE,
 /*PIXEL alpha blending, valid for ARGB/RGBA format only, the embedded alpha value is used*/
TI81XXFB_BLENDING_PIXEL
```

};

enum ti81xxfb_transparancy_type {

```
/*when performing color comparison, no MASK*/
TI81XXFB_TRANSP_LSPMASK_NO = 0,
/*MASK the LSB bit0 for each color component*/
TI81XXFB_TRANSP_LSPMASK_1,
/*MASK the LSB bit[1:0] for each color component*/
TI81XXFB_TRANSP_LSPMASK_2,
/*MASK the LSB bit[2:0] for each color component*/
TI81XXFB_TRANSP_LSMMASK_3
```

};

struct ti81xxfb_region_params {

```
__u16                          ridx;
__u16                          pos_x;
__u16                          pos_y;
__u16                          priority;
/*enum ti81xxfb_status*/
__u32                           firstregion;
/*enum ti81xxfb_status*/
__u32                         lastregion;
/*enum ti81xxfb_status*/
__u32                           scalaren;
```

```
/*enum ti81xxfb_status*/
__u32                          stencilingen;
/*enum ti81xxfb_status*/
__u32                        bben;
/*enum ti81xxfb_status*/
__u32                          transen;
/*enum ti81xxfb_blending_type*/
__u32                        blendtype;
/*enum ti81xxfb_transparancy_type*/
__u32                         transtype;
__u32                    transcolor;
__u8                          bbalpha;
__u8                     blendalpha;
__u8                        reserved[2];
```

};

- **NOTE:** firstregion and lastregion must be set to 1, these two fields are reserved for future usages.

Usage: `struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {`

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

`}`

TIFB_SET_PARAMS

Ioctl set the current graphics features.

Data Structure: `enum ti81xxfb_status {`

```
TI81XXFB_FEATURE_DISABLE = 0,
TI81XXFB_FEATURE_ENABLE
```

`};`

enum ti81xxfb_blending_type {

```
/*alpha blending disable*/
TI81XXFB_BLENDING_NO = 0,
/* global alpha blending*/
TI81XXFB_BLENDING_GLOBAL,
/* PALETTE alpha blending, the alpha value in CLUT is used for blending*/
TI81XXFB_BLENDING_PALETTE,
 /*PIXEL alpha blending, valid for ARGB/RGBA format only, the embedded alpha value is used*/
TI81XXFB_BLENDING_PIXEL
```

};

enum ti81xxfb_transparancy_type {

```
/*when performing color comparison, no MASK*/
TI81XXFB_TRANSP_LSPMASK_NO = 0,
/*MASK the LSB bit0 for each color component*/
```

```
TI81XXFB_TRANSP_LSPMASK_1,
/*MASK the LSB bit[1:0] for each color component*/
TI81XXFB_TRANSP_LSPMASK_2,
/*MASK the LSB bit[2:0] for each color component*/
TI81XXFB_TRANSP_LSMMASK_3
```

};

struct ti81xxfb_region_params {

```
__u16                         ridx;
__u16                         pos_x;
__u16                         pos_y;
__u16                         priority;
/*enum ti81xxfb_status*/
__u32                          firstregion;
/*enum ti81xxfb_status*/
__u32                        lastregion;
/*enum ti81xxfb_status*/
__u32                          scalaren;
/*enum ti81xxfb_status*/
__u32                          stencilingen;
/*enum ti81xxfb_status*/
__u32                        bben;
/*enum ti81xxfb_status*/
__u32                          transen;
/*enum ti81xxfb_blending_type*/
__u32                        blendtype;
/*enum ti81xxfb_transparancy_type*/
__u32                         transtype;
__u32                    transcolor;
__u8                          bbalpha;
__u8                     blendalpha;
__u8                        reserved[2];
```

};

- **NOTE:** firstregion and lastregion must be set to 1, these two fields are reserved for future usages.

Usage:

- Enable Global Alpha Blending with alpha value (0x40)

```
struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0)
{
```

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

```
} regp.blendtype = TI81XXFB_BLENDING_GLOBAL; regp.blendalpha = 0x40; if
(ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {
```

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

- Enable Transparency without masking, the RGB color key is 0x123456

struct ti81xxfb_region_params regp; if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

} regp.transen = TI81XXFB_FEATURE_ENABLE; regp.transtype = TI81XXFB_TRANSP_LSPMASK_NO; regp.transcolor = 0x123456; if (ioctl(fd, TIFB_SET_PARAMS, &regp) < 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

TIFB_QUERY_MEMINFO

Returns the size and type of the frame buffer

Data Structure: enum ti81xxfb_mem_mode {

```
TI81XXFB_MEM_NONTILER = 0,
TI81XXFB_MEM_TILER_PAGE ,
TI81XXFB_MEM_TILER_8,
TI81XXFB_MEM_TILER_16,
TI81XXFB_MEM_TILER_32
```

}; struct ti81xxfb_mem_info {

```
__u32 size;
/*ti81xxfb_mem_mode*/
__u32  type;
```

};

NOTE: Only TI81XXFB_MEM_NONTILER is supported currently.

Usage: struct ti81xxfb_mem_info mi; if (ioctl(fb, TIFB_QUERY_MEM, &mi)) {

```
perror("TIFB_QUERY_MEM.\n");
close(fd);
exit(1);
```

}

TI81XX_SETUP_MEM

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

Data Structure:

enum ti81xxfb_mem_mode {

```
TI81XXFB_MEM_NONTILER = 0,
TI81XXFB_MEM_TILER_PAGE ,
TI81XXFB_MEM_TILER_8,
TI81XXFB_MEM_TILER_16,
TI81XXFB_MEM_TILER_32
```

`}; struct ti81xxfb_mem_info {`

```
__u32 size;
/*ti81xxfb_mem_mode*/
__u32  type;
```

`};`

NOTE: Only TI81XXFB_MEM_NONTILER is supported currently.

Usage: `struct ti81xxfb_mem_info mi; if (ioctl(fb, TIFB_QUERY_MEM, &mi)) {`

```
perror("TIFB_QUERY_MEM.\n");
close(fd);
exit(1);
```

`}`

mi.size = <Expected size of buffer> mi.type = <Expected type of buffer> if (ioctl(fb, TIFB_SETUP_MEM, &mi)) {

```
perror("TIFB_SETUP_MEM.\n");
close(fd);
exit(1);
```

}

TIFB_SET_SCINFO

Allow users to config the scalar ratio

Data Structure:

struct ti81xxfb_coeff {

```
__u16     horcoeff[TI81XXFB_COEFF_HOR_TAP][TI81XXFB_COEFF_PHASE];
__u16     vercoeff[TI81XXFB_COEFF_VER_TAP][TI81XXFB_COEFF_PHASE];
```

}; **NOTE:**
each coefficient is 10 bit wide, so MSB[15:10] will be truncated by the driver.

`struct ti81xxfb_scparams {`

```
__u16                  inwidth;
__u16                  inheight;
__u16                  outwidth;
__u16                  outheight;
struct ti81xxfb_coeff  *coeff
```

`};` **NOTE:**

- 0.25x-4x scaling is supported for both horizontal and vertical direction.
- if 1.0 is selected, hardware performs anti-flicker filtering.
- If app wants to use preloaded coefficients, please set coeff = NULL.

Usage: scale input image to 2x(horizontal) and 1.5x(vertical) with app's own coefficients `struct ti81xxfb_scparams scprms; struct ti81xxfb_coeff coeff; struct ti81xxfb_region_params regp;`

scprms.inwidth = x; scprms.inheight = y; scprms.outwidth = 2*x; scprms.outheight = 1.5*y, scprms.coeff = &coeff;

if (ioctl(fd, TIFB_SET_SCINFO, &scprms) < 0) {

```
perror("TIFB_SET_SCINFO.\n");
close(fd);
exit(1);
```

}

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

}

regp.scalaren = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp)< 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

Usage: scale input image to 2x(horizontal) and 1.5x(vertical) with preloaded coefficients `struct ti81xxfb_scparams scprms; struct ti81xxfb_region_params regp;`

scprms.inwidth = x; scprms.inheight = y; scprms.outwidth = 2 * x; scprms.outheight = 1.5 * y, scprms.coeff = NULL;

if (ioctl(fd, TIFB_SET_SCINFO, &scprms) < 0) {

```
perror("TIFB_SET_SCINFO.\n");
close(fd);
exit(1);
```

} if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

}

regp.scalaren = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp)< 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

Usage: Anti-Flicker Implementation

`struct ti81xxfb_scparams scprms; struct ti81xxfb_region_params regp;`

scprms.inwidth = x; scprms.inheight = y; scprms.outwidth = x; scprms.outheight = y, scprms.coeff = NULL;

if (ioctl(fd, TIFB_SET_SCINFO, &scprms) < 0) {

```
perror("TIFB_SET_SCINFO.\n");
close(fd);
exit(1);
```

}

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

}

regp.scalaren = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp)< 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

TIFB_GET_SCINFO

Allow users to get current scalar ratio

Data Structure:

`struct ti81xxfb_scparams {`

```
__u16                   inwidth;
__u16                   inheight;
__u16                   outwidth;
__u16                   outheight;
struct ti81xxfb_coeff   *coeff;
```

`};` **NOTE:**

coeff is not used when called TIFB_GET_SCINFO ioctl.

Usage: `struct ti81xxfb_scparams scprms; if (ioctl(fd, TIFB_GET_SCINFO, &scprms) {`

```
perror("TIFB_GET_SCINFO.\n");
close(fd);
exit(1);
```

`}`

TIFB_ALLOC

This ioctl is used to allocate a memory from reserved pool and return the physical addres back to app. This memory can be used by the stenciling data buffer.

USAGE: allocate a buffer `unsigned long size; unsigned long phy_addr; unsigned long temp;`

size = 8*1024; temp = size; if(ioctl(fd, TIFB_ALLOC, &size)) {

```
perror("TIFB_ALLOC.\n");
close(fd);
exit(1);
```

} phy_addr = size;

TIFB_FREE

This ioctl is to free a memory allocated by the TIFB_ALLOC.

USAGE `unsigned long phy_addr; void virt_addr; unsigned long size;`

munmap(virt_addr, stensize); if (ioctl(fd,TIFB_FREE, &phy_addr)) {

```
perror(" TIFB_FREE.\n");
close(fd);
exit(1);
```

} **NOTE:**

If the memory is used by the stenciling, please disable stenciling feature before freeing it.

TIFB_SET_STENC

Allow user to config stenciling.

Data Structure: `struct ti81xxfb_stenciling_params {`

```
__u32              pitch;
__u32              paddr;
```

`};` **NOTE:**

pitch must be 16 byte alignment.

USAGE: how to enable the stenciling feature `struct ti81xxfb_stenciling_params stenprms; struct fb_var_screeninfo var; struct ti81xxfb_region_params regp; unsigned long size; unsigned long temp;`

if (ioctl(fd, FBIOGET_VSCREENINFO, &var)) {

```
perror("FBIOGET_VSCREENINFO.\n");
close(fd);
exit(1);
```

}

stenprms.pitch = var.xres >> 3; if (stenprms.pitch & 0xF)

```
   stenprms.pitch += 0xF - (stenprms.pitch & 0xF);
```

size = stenprms.pitch * var.yres;

temp = size; if(ioctl(fd, TIFB_ALLOC, &size)) {

```
perror("TIFB_ALLOC.\n");
close(fd);
exit(1);
```

}

stenprms.paddr = size; size = temp;

virt_addr = (unsigned char *)mmap(0, size, (PROT_READ | PROT_WRITE), MAP_SHARED, fd, stenprms.paddr); if ((int)stenbuf == -1) {

```
perror("sten MMAP failed.\n");
close(fd);
exit(1);
```

}

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

}

regp.stencilingen = TI81XXFB_FEATURE_ENABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp)< 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

USAGE: how to disable the stenciling feature `struct ti81xxfb_stenciling_params stenprms; struct ti81xxfb_region_params regp; unsigned long size;`

if (ioctl(fd, FBIOGET_VSCREENINFO, &var)) {

```
perror("FBIOGET_VSCREENINFO.\n");
close(fd);
exit(1);
```

}

stenprms.pitch = var.xres >> 3; if (stenprms.pitch & 0xF)

```
   stenprms.pitch += 0xF - (stenprms.pitch & 0xF);
```

size = stenprms.pitch * var.yres;

if (ioctl(fd, TIFB_GET_PARAMS, &regp) < 0) {

```
perror("TIFB_GET_PARAMS\n");
close(fd);
exit(1);
```

}

regp.stencilingen = TI81XXFB_FEATURE_DISABLE; if (ioctl(fd, TIFB_SET_PARAMS, &regp)< 0) {

```
perror("TIFB_SET_PARAMS\n");
close(fd);
exit(1);
```

}

munmap(virt_addr, size); if (ioctl(fd,TIFB_FREE, &stenprms.paddr)) {

```
perror(" Failed to free memory.\n");
close(fd);
exit(1);
```

}

OMAPFB Compatible IOCTLS

TI81XX Fbdev driver also supports the following OMAPFB IOCTLs.

OMAPFB_SETUP_MEM

check OMAPFB userguide for the usage

OMAPFB_QUERY_MEM

check OMAPFB userguide for the usage

OMAPFB_SETUP_PLANE

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:

Only pos_x/pos_y/enabled/out_width/out_height are valid for this ioctl. pos_x/pos_y change the display starting location. enabled bit enable/disable the corresponding graphcis port. out_width/out_height enable the scaling feature of the graphics ports. if both out_width/out_height are set to zero, driver treat is as non-scaling case.

NOTE:

application need make sure the pos_x/pos_y and out_width/out_height do not out of the current display frame boundary.

OMAPFB_QUERY_PLANE

OMAPFB_GET_VRAM_INFO

check OMAPFB userguide for the usage

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 VPSS, Fbdev and V4L2 Display 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
mode
modes
name
rotate
pan
phys_addr
rotate
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 1, 2, 4, 8, 16, 24 and 32. `# echo 1/2/4/8/16/24/32 > /sys/class/graphics/fb0/ bits_per_pixel`

blank	Allows user to control lcd display blanking configuration independently.
	`# echo 0/1  > /sys/class/graphics/fb0/blank`
	
	`Values only 0(FB_BLANK_UNBLANK) and 4(FB_BLANK_NORMAL) 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.

VPSS Library sysfs attributes

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

```
#
# ls -1 /sys/devices/platform/vpss/
display0
display1
display2
display3
graphics0
graphics1
graphics2
modalias
subsystem
system
uevent
#
```

Note: display3 is not available on DM814X/AM387X Platform

VPSS Library: system

Config Display Controller

```
#
# ls -1 /sys/devices/platform/vpss/system/
tiedvencs
#
```

VPSS Library-system: sysfs attributes

Acronym	Definition
tiedvencs	tie multiple vencs together: hdmi id = 1, hdcomp id = 2, dvo2 id = 4, sd id = 8 `# echo XX > /sys/devices/platform/vpss/system/tiedvencs` where XX: OR value of the venc id.

NOTE:

- By Default, VPSS driver automatically configures the PLL clock based on the display mode setuped by the application for individual VENC without considering the PLL clock is shared or not with other VENC. So it is application's responsiblity to make sure that PLL clock and clock source of VENC are coupled well to supported the desired display mode.
- hdcomp is not available on DM814X/AM387X Platform

VPSS Library: display

display Output

Arguments	Definition
single	An 10-bit wide, single channel CCIR656 video stream
double	Two 10-bit wide video streams in CCIR656 format. The video data is YUV422 format
triple	Three channel 10-bit video streams with embedded sync SAV/EAV. The video format can be YCbCr 444 or RGB
doublediscrete	Two channel 10-bit video streams with dedicated HS/VS/FID/ACTVID sync signals. The video data format is YUV 422. The first channel is Y; and CbCr are multiplexed on the second channel
triplediscrete	Three channel 10-bit video streams with dedicated HS/VS/FID/ACTVID sync signals. The three channel component digital video can be RGB or YUV 444 format

VPSS Library: display0

Configure the HDMI VENC, which is connected to the on-chip HDMI module.

```
#
# ls -1 /sys/devices/platform/vpss/display0/
clksrc
edid
enabled
mode
order
output
timings
#
```

VPSS Library-display0: sysfs attributes

Acronym	Definition
clksrc	Set the right clock source for the VENC `echo dclk/dclkdiv2/dclkdiff > /sys/devices/platform/vpss/display0/clksrc`
edid	read EDID information only.
enabled	Enable/Disable the VENC. `# echo 0/1 > /sys/devices/platform/vpss/display0/enabled`
mode	Set the right VENC mode,default 1080p-60.Check drivers/video/ti81xx/vpss/sysfs.h to get name of the supported modes `# echo mode_name > /sys/devices/platform/vpss/display0/mode`

order	reshuffle display order
	`echo A,B/C/D/E > /sys/devices/platform/vpss/display0/order`
	`A: enable(1)/disable(0) global reorder. if 0 is set, then only B value is required, C,D and E should not be used.`
	`B: 0-3: display order of video(0:lowest, 3:highest)`
	`C: 0-3: display order of graphics0`
	`D: 0-3: display order of graphics1`
	`E: 0-3: display order of graphics2`

output	Set the right VENC output `echo single/double/doublediscrete/triple/triplediscrete,rgb888/yuv444p/yuv422spuv,(0/1)/(0/1)/(0/1)/(0/1) > /sys/devices/platform/vpss/display0/output` the first part is the sync mode, the second part is output data format, the last part is the polarity of the DE/FID/HS/VS, 1: ACTIVE_LOW(inverted),0:ACTIVE_HIGH(non_inverted).

timings	Displays the timing configuration.
	`echo 148500,1920/88/148/44,1080/4/36/5,1 > /sys/devices/platform/vpss/display0/timings`
	`where:`
	`148500: display pixel clock(KHz)`
	`1920: display width`
	`88: horizontal front porch`
	`148: horizontal back porch`
	`44: horizontal sync width`
	`1080: display height`
	`4: vertical front porch`
	`36: vertical back porch`
	`5: vertical sync width`
	`1: progressive output`

VPSS Library: display1

Configure DVO2 VENC, which is able to connect to exteral display device, such as LCD pannel, HDMI Transmitter etc. This document does not cover any such information related to the external display device.

```
#
# ls -1 /sys/devices/platform/vpss/display1/
clksrc
enabled
mode
output
timings
#
```

VPSS Library-display1: sysfs attributes

Acronym	**Definition**

clksrc	Set the right clock source for the VENC
	`echo aclk/aclkdiv2/aclkdiff/dclk/dclkdiv2/dclkdiff > /sys/devices/platform/vpss/display1/clksrc`
	DM814X/AM387X Platform
	`echo aclk/aclkdiv2/aclkdiff > /sys/devices/platform/vpss/display1/clksrc`

enabled	Enable/Disable the VENC. `# echo 0/1 > /sys/devices/platform/vpss/display1/enabled`
mode	Set the right VENC mode, default 1080p-60.Check drivers/video/ti81xx/vpss/sysfs.h to get name of the supported modes `# echo mode_name > /sys/devices/platform/vpss/display1/mode`

order	reshuffle display order
	`echo A,B/C/D/E > /sys/devices/platform/vpss/display1/order`
	`A: enable(1)/disable(0) global reorder. if 0 is set, then only B value is required, C,D and E should not be used.`
	`B: 0-3: display order of video(0:lowest, 3:highest)`
	`C: 0-3: display order of graphics0`
	`D: 0-3: display order of graphics1`
	`E: 0-3: display order of graphics2`

output	Set the right VENC output `echo single/double/doublediscrete/triple/triplediscrete,rgb888/yuv444p/yuv422spuv,(0/1)/(0/1)/(0/1)/(0/1) > /sys/devices/platform/vpss/display1/output` the first part is the sync mode, the second part is output data format, the last part is the polarity of the DE/FID/HS/VS, 1: ACTIVE_LOW(inverted),0:ACTIVE_HIGH(non_inverted).

timings	Displays the timing configuration.
	`echo 148500,1920/88/148/44,1080/4/36/5,1 > /sys/devices/platform/vpss/display1/timings`
	`where:`
	`148500: display pixel clock(KHz)`
	`1920: display width`
	`88: horizontal front porch`
	`148: horizontal back porch`
	`44: horizontal sync width`
	`1080: display height`
	`4: vertical front porch`
	`36: vertical back porch`
	`5: vertical sync width`
	`1: progressive output`

VPSS Library: display2

Configure the SD VENC(Composite output)

```
#
# ls -1 /sys/devices/platform/vpss/display2/
clksrc
enabled
mode
name
order
output
timings
#
```

VPSS Library-display2: sysfs attributes

Acronym	Definition
clksrc	not supported
enabled	Enable/Disable the VENC. `# echo 0/1 > /sys/devices/platform/vpss/display2/enabled`
mode	Set the right VENC mode:ntsc/pal are supported (default ntsc). `# echo ntsc/pal > /sys/devices/platform/vpss/display2/mode`

order	reshuffle display order
	`echo A,B/C/D/E > /sys/devices/platform/vpss/display2/order`
	`A: enable(1)/disable(0) global reorder. if 0 is set, then only B value is required, C,D and E should not be used.`
	`B: 0-3: display order of video(0:lowest, 3:highest)`
	`C: 0-3: display order of graphics0`
	`D: 0-3: display order of graphics1`
	`E: 0-3: display order of graphics2`

output	set the output for SD VENC `echo composite/svideo > /sys/devices/platform/vpss/display2/output`
timings	Displays the timing configuration.(Not available now)

VPSS Library: display3

Configure the HDCOMP VENC(component output)
Note: display3 is not available on DM814X/AM387X Platform

```
#
# ls -1 /sys/devices/platform/vpss/display3/
clksrc
enabled
mode
order
output
timings
#
```

VPSS Library-display3: sysfs attributes

Acronym	Definition
clksrc	Set the right clock source for the VENC `echo aclk/aclkdiv2/aclkdiff/dclk/dclkdiv2/dclkdiff > /sys/devices/platform/vpss/display3/clksrc`
enabled	Enable/Disable the VENC. `# echo 0/1 > /sys/devices/platform/vpss/display3/enabled`
mode	Set the right VENC mode:1080p-60/1080p-30/1080i-60/720p-60 are supported (default 1080i-60). `# echo 1080p-60/1080p-30/1080i-60/720p-60 > /sys/devices/platform/vpss/display3/mode`

order	reshuffle display order
	`echo A,B/C/D/E > /sys/devices/platform/vpss/display3/order`
	`A: enable(1)/disable(0) global reorder. if 0 is set, then only B value is required, C,D and E should not be used.`
	`B: 0-3: display order of video(0:lowest, 3:highest)`
	`C: 0-3: display order of graphics0`
	`D: 0-3: display order of graphics1`
	`E: 0-3: display order of graphics2`

output	Set the right VENC output `echo component/svideo/composite, rgb888/yuv444p/yuv422spuv > /sys/devices/platform/vpss/display3/output` the first part is the output mode, the second part is output data format.

timings	Displays the timing configuration.
	`echo 148500,1920/88/148/44,1080/4/36/5,1 > /sys/devices/platform/vpss/display3/timings`
	`where:`
	`148500: display pixel clock(KHz)`
	`1920: display width`
	`88: horizontal front porch`
	`148: horizontal back porch`
	`44: horizontal sync width`
	`1080: display height`
	`4: vertical front porch`
	`36: vertical back porch`
	`5: vertical sync width`
	`1: progressive output`

VPSS Library: graphics0/1/2

In all total 3 graphics pipelines are supported on hardware,

```
#
# ls -1 /sys/devices/platform/vpss/graphics0/
enabled
nodes
#
```

VPSS Library-graphics0/1/2: sysfs attributes

Acronym	Definition
enabled	User can enable/disable overlay through this entry. `# echo 0/1 > /sys/devices/platform/vpss/graphics0/enabled`

nodes	User can configure which vencs(hdmi/hdcomp/dvo2/sd) this graphics pipeline connected to.
	`One graphics pipeline can be connected to multiple video compositor as long as those VENCs associated with video compositor are in the same timing.`
	`# echo XX:outputs > /sys/devices/platform/vpss/graphics0/nodes`
	where XX: 1-4 number of vencs connected, outputs: see below table.

Graphics Notes input/output table

device	output

fb0	hdmi(display0)
	dvo2(display1)
	hdcomp(display3)
	sd(display2)

fb1	hdmi(display0)
	dvo2(display1)
	hdcomp(display3)
	sd(display2)

N/A	hdmi(display0)
	dvo2(display1)
	hdcomp(display3)
	sd(display2)

Note: hdcomp(display3) is not available on DM814X/AM387X Platform.

VPSS Library: Video0/1/2

In all total 3 video pipelines are supported on hardware,

```
#
# ls -1 /sys/devices/platform/vpss/video0/
alpha_blending_enabled
default_color
enabled
global_alpha
nodes
trans_key_enabled
trans_key_type
trans_key_value
#
```

VPSS Library-video0/1/2: sysfs attributes

Acronym	Definition

alpha_blending_enabled	User can enable/disable alpha blending for the video pipeline. Not available for video2.
	`echo 0/1 > /sys/devices/platform/vpss/video0/alpha_blending_enabled`
	`     where 0/1: 0 -> disable, 1 -> enable`

default_color	Not available
enabled	Not available
global_alpha	User can set the global alpha blending value for the video pipeline. Not available for video2. `echo 0-255 > /sys/devices/platform/vpss/video0/global_alpha`

nodes	User can configure how many vencs(hdmi/hdcomp/dvo2/sd) this video pipeline connected to.
	`One video pipeline can be connected to multiple video VENCs as long`
	`as those VENCs associated with video compositor are in the same timing.`
	`# echo input:output1,output2,output3 > /sys/devices/platform/vpss/video0/nodes`
	see below table for details

trans_key_enabled	User can enable/disable transparency keying for the video pipeline. Not available for video2.
	`echo 0/1 > /sys/devices/platform/vpss/graphics0/trans_key_enabled`
	`     where 0/1: 0 -> disable, 1 -> enable`

trans_key_type	User can set transparency keying mask type for the video pipeline. Not available for video2.
	`echo 0/1/2/3 > /sys/devices/platform/vpss/graphics0/trans_key_enabled`
	`     where 0/1/2/3 defines how many LSB bit to be masking when performing keying`

trans_key_value	User can set transparency keying value(RGB888, hex value only) for the video pipeline. Not available for video2.
	`echo 0xRRGGBB > /sys/devices/platform/vpss/graphics0/trans_key_value`
	`     where 0xRRGGBB: R in the upper byte, B in the lower byte.`

Video Notes input/output table

device	input		
	vcompmux	**hdcompmux**	**sdmux**

video0 (/dev/video1)	hdmi(display0)
	dvo2(display1)
	hdcomp(display3)
		hdmi(display0)
		dvo2(display1)
		hdcomp(display3)
			sd(display2)

video1(/dev/video2)	hdmi(display0)
	dvo2(display1)
	hdcomp(display3)
		hdmi(display0)
		dvo2(display1)
		hdcomp(display3)
			sd(display2)

video2(/dev/video3)	N/A	N/A	sd(display2)

Note: hdcomp(display3) is not available on DM814X/AM387X Platform.

Miscellaneous Configurations

The default setup/configuration is -

```
fb0  -- => graphics0 - - => hdmi_venc(display0)->on_chip_hdmi output


* DM816X/AM389X, DM813X Platform
fb1  -- => graphics1 - - => hdcomp_venc(display3)->component output
* DM814X/AM387X Platform
fb1  -- => graphics1 - - => dvo2_venc(display1) -> LCD pannel or external display device


fb2  -- => graphics2 - - => sd_venc(display2)->composite ouput
```

```
/dev/video1  -- => video0 -- => vcompmux  -- => hdmi_venc(display0)->on_chip_hdmi output


* DM816X/AM389X, DM813X Platform
/dev/video2  -- => video1 -- => hdcompmux -- => hdcomp_venc(display3)->component output
* DM814X/AM387X Platform
```

```
/dev/video2  -- => video1 -- => hdcompmux -- => dvo2_venc(display1)-> LCD Pannel or external dispaly device


/dev/video3  -- => video2 -- =>   sdmux   -- => sd_venc(display2)->composite ouput
```

User can control/configure the various graphics/video pipelines to different video compositors and VENCs. This section demonstrate/explains the switching of output using above interfaces.

NOTE:
The connections between fb nodes and graphics pipelines are fixed and not able to change.
The connections between /dev/video nodes and video pipelines are fixed and not able to change.

Switching fb0(graphics0) output from HDMI VEVC(display0) to DVO2 VENC(display1)

Follow the steps below to switch graphics0(fb0)output from HDMI to DVO2 VENC

- Disable graphics 0

```
#echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

- Switch output VENC

```
# echo 1:dvo2 > /sys/devices/platform/vpss/graphics0/nodes
```

- Enable graphics 0

```
#echo 1 > /sys/devices/platform/vpss/graphics0/enabled
```

NOTE:
Similar steps must be followed for switching other graphcis pipeline to different compositors and VENCs.

Switching /dev/video1(video0) from vcompmux - HDMI VEVC(display0) to vcompmux - DVO2 VENC(display1)

```
# echo vcompmux:dvo2 > /sys/devices/platform/vpss/video0/nodes
```

- **NOTE:**

Application should make sure /dev/video1 device node is closed when performing the above step
Similar steps must be followed for switching other video pipeline to different input/output configurations.

Switching /dev/video1(video0) from vcompmux - HDMI VEVC(display0) to hdcompmux - DVO2 VENC(display1)

```
# echo hdcompmux:dvo2 > /sys/devices/platform/vpss/video0/nodes
```

- **NOTE:**

Application should make sure /dev/video1 device node is closed when performing the above step
Similar steps must be followed for switching other video pipeline to different input/output configurations.

Cloning fb0(graphics0) output to both HDMI VENC(display0) and DVO2 VENC(display1)

Follow below steps to clone GFX overlay output to both HDMI and DVO2 VENC.

NOTE:
Appliation should make sure that both VENCs have the same timing.
If one graphics plane is connected to multiple vencs, those vencs must tied.
The similar steps must be following when cloning other graphics pipeline to multiple VENCs

- Disable graphics0 pipeline

```
#echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

- Disable hdmi VENC

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

- Disable dvo2 venc

```
#echo 0 > /sys/devices/platform/vpss/display1/enabled
```

- Tited the VENCs

```
# echo 5 > /sys/devices/platform/vpss/system/tiedvenc
```

- Switch the both HDMI and DVO2 VENCs

```
# echo 2:hdmi,dvo2 > /sys/devices/platform/vpss/graphics0/nodes
```

- Enable graphics 0 pipeline

```
#echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

Cloning /dev/video1(video0) output to both vcompmux-HDMI VENC(display0) and vcompmux-DVO2 VENC(display1)

Follow below steps to clone video0 output to both HDMI and DVO2 VENC through vcompmux.

NOTE:
Appliation should make sure that both VENC has the same timing.
If one video pipeline is connected to multiple vencs, those vencs must tied.

- Disable hdmi VENC

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

- Disable dvo2 venc

```
#echo 0 > /sys/devices/platform/vpss/display1/enabled
```

- Tited the VENCs

```
# echo 5 > /sys/devices/platform/vpss/system/tiedvenc
```

- Switch the both HDMI and DVO2 VENCs

```
# echo vcompmux:hdmi,dvo2 > /sys/devices/platform/vpss/video0/nodes
```

NOTE: Application must make sure that /dev/video0 is closed when performing above steps.
The similar stpes must be following when cloning other video pipelines to multiple VENCs

Change the mode on HDMI VENC(display0)

Follow below steps to change the VENC mode on HDMI, assuming that Graphics0(fb0) is connected to HDMI

- Disable graphics0 pipeline

```
#echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

- Disable hdmi VENC

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

- Set the new VENC mode

```
#echo 720p-60 > /sys/devices/platform/vpss/display0/mode
```

- Enable hdmi VENC

```
#echo 1 > /sys/devices/platform/vpss/display0/enabled
```

- Enable graphics0 pipeline

```
#echo 1 > /sys/devices/platform/vpss/graphics0/enabled
```

NOTE:
The similar stpes must be following when change mode of other VENCs.
application should disable all the pipeline connected to the HDMI venc before performing the above steps.

Change display timing on DVO2 VENC(display1)

Follow below steps to change the display timing on DVO2 VENC

- Disable DVO2 VENC

```
#echo 0 > /sys/devices/platform/vpss/display1/enabled
```

- set new display timings

```
#echo 74250,1280/110/220/40,720/5/20/5,1/3 > /sys/devices/platform/vpss/dvo2/timings
```

- Enable DVO2 VENC

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

NOTE:
The similar stpes must be followed when change mode of other VENCs.
application should disable all the pipeline connected to the HDMI venc before performing the above steps.

Reshuffle display order on HDMI VENC(display0)

Follow below steps to reshuffle the display order on HDMI VENC

- set up new display order(graphics0 has higher displar order than graphics1 & graphics2)

```
echo 1:0/3/1/1 > /sys/devices/platform/vpss/display0/order
```

NOTE:
The similar stpes must be followed when change order of other VENCs.
Reshuffling order is runing time parameter operation.

Change the clock source on HDMI VENC(display0)

Follow below steps to change the clock source on HDMI.

- Disable hdmi VENC

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

- Set the new clock source

```
#echo dclkdiff > /sys/devices/platform/vpss/display0/clksrc
```

- Enable hdmi VENC

```
#echo 1 > /sys/devices/platform/vpss/display0/enabled
```

NOTE:
The similar stpes must be following when change mode of other VENCs.
application should disable all the pipeline connected to the HDMI venc before performing the above steps.

Change the OUTPUT of HDMI VENC(display0)

Follow below steps to change the OUTPUT of HDMI VENC.

- Disable hdmi VENC

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

- Set the new digital sync format double embedded sync

```
#echo double > /sys/devices/platform/vpss/display0/output
```

- Enable hdmi VENC

```
#echo 1 > /sys/devices/platform/vpss/display0/enabled
```

NOTE:
The similar stpes must be following when change mode of other VENCs.
application should disable all the pipeline connected to the HDMI venc before performing the above steps.

Driver Configuration

Framebuffer driver

```
$ make ARCH=arm CROSS_COMPILE=$(PATH_TO_TOOLCHAIN)/bin/arm-none-linux-gnueabi- menuconfig
```

- 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  --->
< > E-Ink Broadsheet/Epson S1D13521 controller support
<M> TI81XX Viddeo Processing Subsystem support (EXPERIMENTAL)  --->
[ ] Backlight & LCD device support  --->
    ...
    ...
```

- Select TI81XX Viddeo Processing Subsystem support (EXPERIMENTAL) from the same menu.

```
    ...
    ...
<*> Support for frame buffer devices  --->
(50)   VRAM size (MB)
[*]   Debug support
<M> TI81XX frame buffer support (EXPERIMENTAL)  --->
```

- Configure default VRAM size to the required/expected size of buffer, the default is 50MB.

```
    ...
    ...
--- Select TI81XX Viddeo Processing Subsystem support (EXPERIMENTAL)
"(50)   VRAM size (MB)"
[*]   Debug support
    ...
    ...
```

- Enable/Disable Debug function of VPSS

```
[*]   Debug support
```

- Select TI81XX frame buffer support (EXPERIMENTAL) from the same menu. Press <ENTER> to enter the corresponding sub-menu.

```
    ...
    ...
[*]   Debug support for TI81XXFB
"(3)   Number of framebuffers"
    ...
    ...
```

NOTE: If this value is set as 1, only fb0(graphics0) is availabe in the fbdev.
If this value is set as 2, then graphics0(fb0) and graphics1(fb1) pipeline are available in FBDEV.
If this value is set as 3, all 3 graphics0/1/2(fb0/1/2) pipelines are available the FBDEV interface.

V4L2 video driver

```
$ make ARCH=arm CROSS_COMPILE=$(PATH_TO_TOOLCHAIN)/bin/arm-none-linux-gnueabi- menuconfig
```

- 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 ***
[ ]   Media Controller API (EXPERIMENTAL)
"<*>   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  --->
    ...
    ...
```

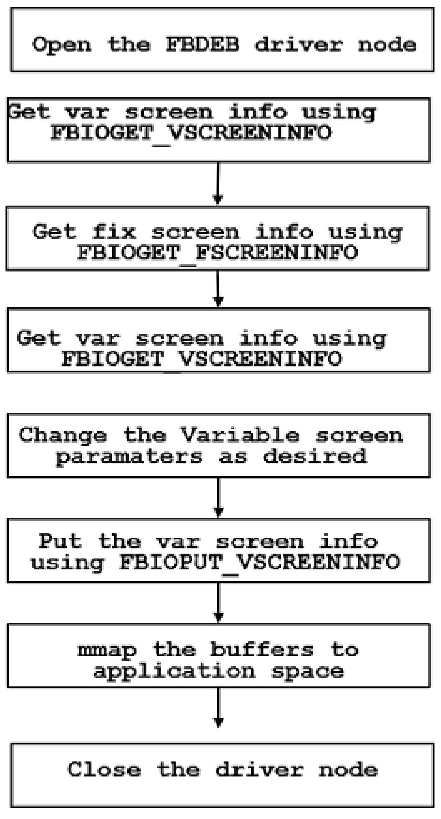
- Select TI81XX V4L2-Display driver

```
[ ]   Autoselect pertinent encoders/decoders and other helper chi
      Encoders/decoders and other helper chips  --->
"<M>   TI81XX V4L2-Display driver"
< >   CPiA2 Video For Linux
< >   Philips SAA7134 support
```

Sample Application Flow

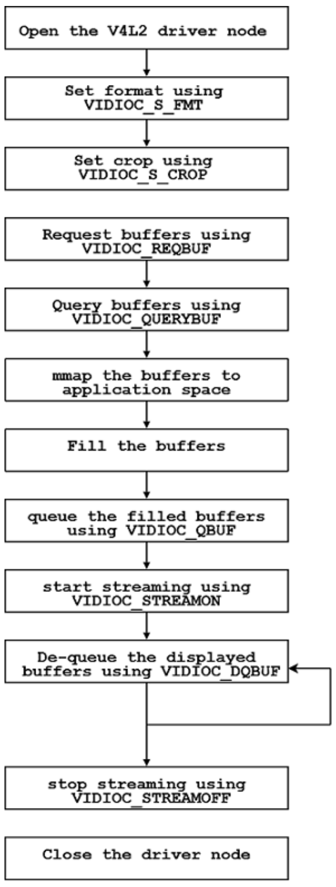
This chapter describes the application flow using the V4l2 and FBDEV drivers.

Fbdev-Display Application Flow


Open the FBDEB driver node
Get var screen info using FBIOGET_VSCREENINFO
Get fix screen info using FBIOGET_FSCREENINFO
Get var screen info using FBIOGET_VSCREENINFO
Change the Variable screen paramaters as desired
Put the var screen info using FBIOPUT_VSCREENINFO
mmap the buffers to application space
Close the driver node

Application for FBDEV driver

V4L2-Display Application Flow


Open the V4L2 driver node
Set format using VIDIOC_S_FMT
Set crop using VIDIOC_S_CROP
Request buffers using VIDIOC_REQBUF
Query buffers using VIDIOC_QUERYBUF
mmap the buffers to application space
Fill the buffers
queue the filled buffers using VIDIOC_QBUF
start streaming using VIDIOC_STREAMON
De-queue the displayed buffers using VIDIOC_DQBUF
stop streaming using VIDIOC_STREAMOFF
Close the driver node
Application for v4l2 driver

References

[1] http://creativecommons.org/licenses/by-sa/3.0/
[2] http://processors.wiki.ti.com/index.php/DM816x_AM389x_HDMI_User_Guide
[3] http://linux.bytesex.org/v4l2/
[4] http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf
[5] http://processors.wiki.ti.com/index.php/DM816X_AM389X_VPSS_Video_Driver_User_Guide_PSP_04.00.01.13#Supported_Custom_IOCTLs

Article Sources and Contributors

TI81XX VPSS Video Driver User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=96239 *Contributors*: HardikShah, Parth.saxena, Yihe

Image Sources, Licenses and Contributors

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

Image:vpss_linux_sw_arch_3.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Vpss_linux_sw_arch_3.png *License*: unknown *Contributors*: Yihe

Image:rgb565.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgb565.png *License*: unknown *Contributors*: Yihe

Image:argb1555.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Argb1555.png *License*: unknown *Contributors*: Yihe

Image:argb4444.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Argb4444.png *License*: unknown *Contributors*: Yihe

Image:rgba5551.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgba5551.png *License*: unknown *Contributors*: Yihe

Image:rgba4444.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgba4444.png *License*: unknown *Contributors*: Yihe

Image:rgb888.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgb888.png *License*: unknown *Contributors*: Yihe

Image:argb6666.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Argb6666.png *License*: unknown *Contributors*: Yihe

Image:rgba6666.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgba6666.png *License*: unknown *Contributors*: Yihe

Image:argb8888.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Argb8888.png *License*: unknown *Contributors*: Yihe

Image:rgba8888.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Rgba8888.png *License*: unknown *Contributors*: Yihe

Image:bitmap 8 bpp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_8_bpp.png *License*: unknown *Contributors*: Yihe

Image:bitmap 4 bpp lower.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_4_bpp_lower.png *License*: unknown *Contributors*: Yihe

Image:bitmap 4 bpp upper.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_4_bpp_upper.png *License*: unknown *Contributors*: Yihe

Image:bitmap 2 bpp offset0.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_2_bpp_offset0.png *License*: unknown *Contributors*: Yihe

Image:bitmap 2 bpp offset1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_2_bpp_offset1.png *License*: unknown *Contributors*: Yihe

Image:bitmap 2 bpp offset2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_2_bpp_offset2.png *License*: unknown *Contributors*: Yihe

Image:bitmap 2 bpp offset3.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_2_bpp_offset3.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset0.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset0.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset1.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset2.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset3.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset3.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset4.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset4.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset5.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset5.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset6.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset6.png *License*: unknown *Contributors*: Yihe

Image:bitmap 1 bpp offset7.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Bitmap_1_bpp_offset7.png *License*: unknown *Contributors*: Yihe

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

Image:yuv422sp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Yuv422sp.png *License*: unknown *Contributors*: Yihe

Image:yuv420sp.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Yuv420sp.png *License*: unknown *Contributors*: Yihe

Image:Fbdev_apps_flow.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Fbdev_apps_flow.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

License