Application Report

Feb 2012

TMS320DM81XX Graphics Control Using Frame Buffer Driver

Video Surveillance Applications

ABSTRACT

TMS320DM81xx devices support three graphics pipelines which can be connected to any of the Video compositors to blend graphics with video before sending them to video encoders. A region-based graphics processor is used to compose one or more rectangular graphics regions to create a display plane which is given as input for the video compositor. The graphics pipelines supported through FBDev interface supports different color formats, display parameters such as height and width of display graphics, position of the display graphics, bits-per-pixel, buffer management through memory mapped (mmaped) on FBDev , scaling on graphics pipeline through FBDev etc.

Video Surveillance Applications 1
ABSTRACT 1
1 Overview 1
2 Memory Requirement and Allocation 1
3 Usage 2
4 Sample Graphics application using FBDev 2
5 System Components 3
6 Code & References 3

1 Overview

TMS320DM81xx device has three graphics pipeline for rendering GUI by connecting each of three graphics pipelines to video compositors. Frame Buffer driver 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.

2 Memory Requirement and Allocation

Frame Buffer Driver and other VPSS kernel driver communicate with Video compositors by setting up a Proxy Server and client on A8 and VPSS M3. For communication between the Proxy Server and Client a shared buffer is used which contains the configuration structures for frame buffer driver and other video processing subsystem modules. This shared buffer address is marked as HDVPSS_SHARED_ADDR with size as HDVPSS_DESC_SIZE in the memory map configuration file (e.g. dvr_rdk/mcfw/src_bios6/cfg/ti816x/config_1G_256MLinux.bld). The same address and size should be reflected in target file system environment variables too(e.g. dvr_rdk\bin\ti816x\env.sh)

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. The VRAM size can be set while booting kernel as given below,

setenv bootargs console=ttyO2,115200n8 mem=242M noinitrd root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp ***vram=40M***

Individual VRAM size for /dev/fb0, /dev/fb1 and /dev/fb2 can be set while inserting FBDev kernel module as follows

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

3 Usage

Given below are the basic steps to open FBDev nodes and draw GUI in display.

1) Boot kernel with at least 16M VRAM size.

setenv bootargs console=ttyO2,115200n8 mem=242M noinitrd root=/dev/nfs nfsroot=nfs-server/home,nolock ip=dhcp ***vram=40M***

2) Run ./init.sh (To insert syslink kernel module)

./init.sh

3) Run ./load.sh (To load M3 binaries and vpps.ko, ti81xxfb.ko ti81xxhdmi.ko)

./load.sh

Note: Edit load.sh to edit vram size while inserting FBDev module.

4) Run A8 side graphics application

./run.sh

4 Sample Graphics application using FBDev

In this section we will discuss the steps to develop a sample graphics application using FBDev.

1) Open Frame Buffer device by open call , with the device name and mode of operation. The driver will expose three software channels (/dev/fb0, /dev/fb1, /dev/fb2) for the graphics pipeline.

fd = open("/dev/fb0", O_RDWR);

2) Get fix screen information using FBIOGET_FSCREENINFO ioctl. Fix screen information gives fixed information(not-changing) like panning step for horizontal and vertical direction, line length, memory mapped start address and length etc.

ret = ioctl(fd, FBIOGET_FSCREENINFO, &fixinfo);

Note: This call is just for confirming buffer size and length and not really required.

3) Get variable screen information using FBIOGET_VSCREENINFO ioctl . Variable screen information gives information like size of the image, bites per pixel, virtual size of the image etc.

ret = ioctl(fd, FBIOGET_VSCREENINFO, &varinfo);

4) User can change variable screen information like resolution, bits per pixel etc. and Set variable screen information using FBIOPUT_VSCREENINFO ioctl.

ret = ioctl(fd, FBIOPUT_VSCREENINFO, &varinfo);

5) Calculate the FBDev buffer with the following formula

*buffersize = varinfo.xres*varinfo.yres*(varinfo.bits_per_pixel/8);*

6) FBDev buffers are mapped from Kernel space to user space by 'mmap'

*grpx->buf = (unsigned char *)mmap (0, buffersize, (PROT_READ|PROT_WRITE), MAP_SHARED, fd, 0);*

7) Now the user application can fill the graphics buffer pointed by 'grpx->buf' with size 'buffersize'

8) User can get and set various region parameters like transparency enable/disable, color, alpha-blending, scaling by IOCTL calls. Shown below is an example of graphics scaling done using IOCTL call

```
status = ioctl(fd, TIFB_GET_SCINFO, &scparams);

scparams.inwidth    = inWidth;
scparams.inheight  = inHeight;

scparams.outwidth  = outWidth;
scparams.outheight = outHeight;

status = ioctl(fd, TIFB_SET_SCINFO, &scparams);

ret = ioctl(fd, TIFB_GET_PARAMS, &regp);
regp.scalaren = TI81XXFB_FEATURE_ENABLE;
ret = ioctl(fd, TIFB_SET_PARAMS, &regp);
```

9) To exit application, 'munmap' and 'close' the driver.

ret = munmap(grpx->buf, (int)grpx->buf_size);

ret = close(grpx->fd);

10) User can another FB device(/dev/fb1 or /dev/fb2) by following steps 1 to 9 mentioned above.

5 System Components

The various system components version that on which this has been validated are as follows:

1. DVR RDK Version: 2.00.00.00
2. LSP: 04.00.01.13/ 04.00.02.14
3. HDVPSS 01.00.01.36

6 Code & References

Please refer to the following examples for further details

1) dvr_rdk/demos/graphic/graphic.c
2) dvr_rdk/demos/graphic/graphic.h

Please refer to the following documentation for further information

1) Linux VPSS User Guide :
http://processors.wiki.ti.com/index.php/DM816X_AM389X_VPSS_Video_Driver_User_Guide_PSP_04.00.01.13