Android多核心嵌入式多媒體系統設計與實作

Linux Video Driver Architecture

賴槿峰 (Chin-Feng Lai)
Assistant Professor, institute of CSIE, National Ilan University
Oct 6th 2012

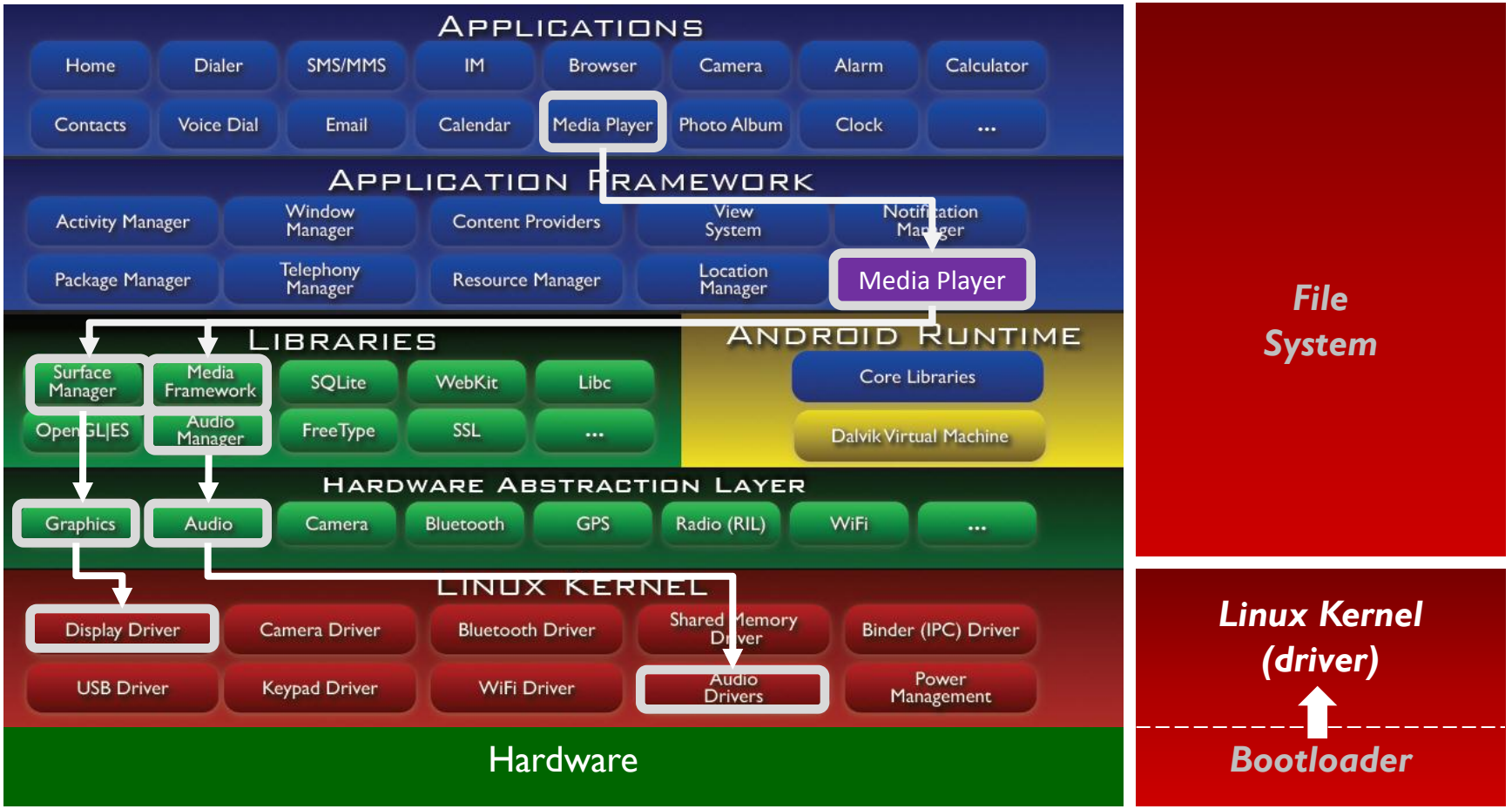
Outline

- Video Device Driver Introduction
- Video Codec Format
- Raw Data Format
- Framebuffer Driver
- Video for Linux Two, V4L2 Driver
- OMAP3 Display SubSystem (DSS)
- LAB

- ***Video Device Driver Introduction***
- ***Video Codec Format***
- ***Raw Data Format***
- ***Framebuffer Driver***
- ***Video for Linux Two, V4L2 Driver***
- ***OMAP3 Display SubSystem (DSS)***
- ***LAB***

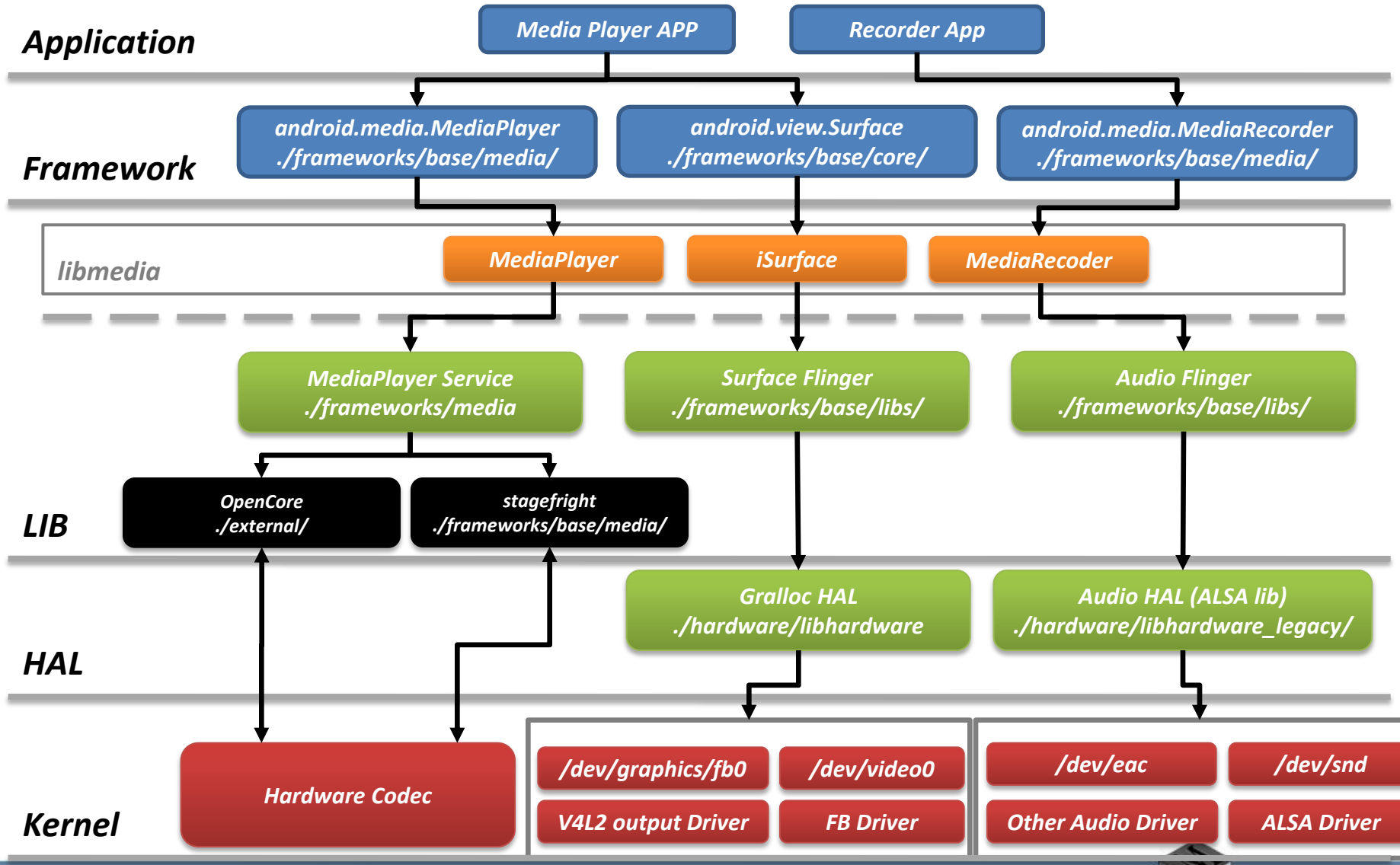
Android Multimedia System Introduction

- Android system and Embedded system



MMN Lab.

Android Multimedia System Introduction



MMN Lab.

Video Device driver introduction

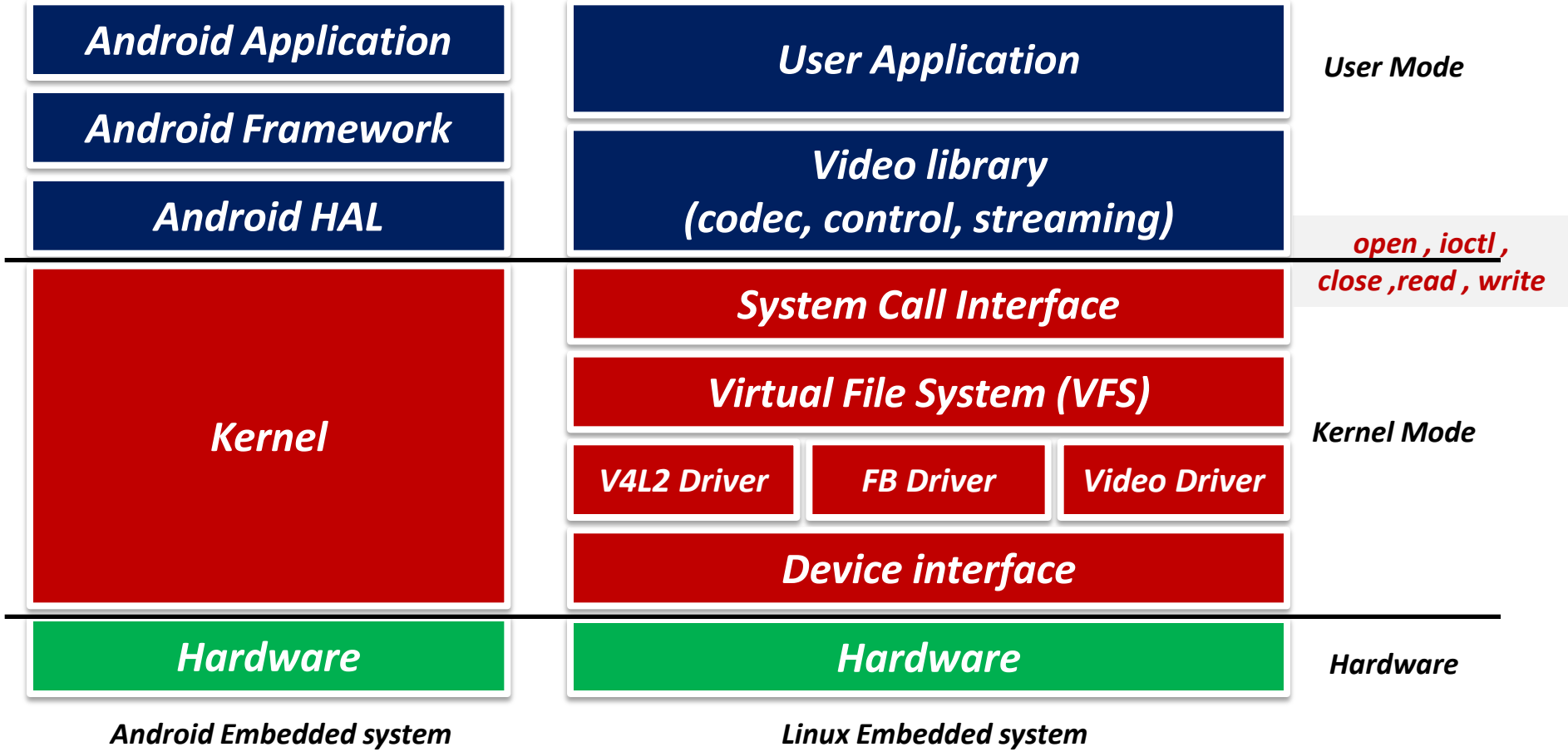
- Linux Video Device Driver is a middle role of user program and hardware
- The Video Driver is responsible for pushing decoded raw data to display interface
- The development of Linux video driver can classify by two group , the one is the video for Linux series(V4L 、V4L2) and Framebuffer

Video Device driver introduction

- Video for Linux series driver provide flexible use with capture/playback interface , also support many camera type and display interface
- Frame buffer driver drive display interface as a block of memory , user program use simple memory control command to read/write data from display device

Video Device driver introduction

- Linux Driver Architecture - Video



MMN Lab.

- *Video Device Driver Introduction*
- ***Video Codec Format***
- *Raw Data Format*
- *Framebuffer Driver*
- *Video for Linux Two, V4L2 Driver*
- *OMAP3 Display SubSystem (DSS)*
- *LAB*

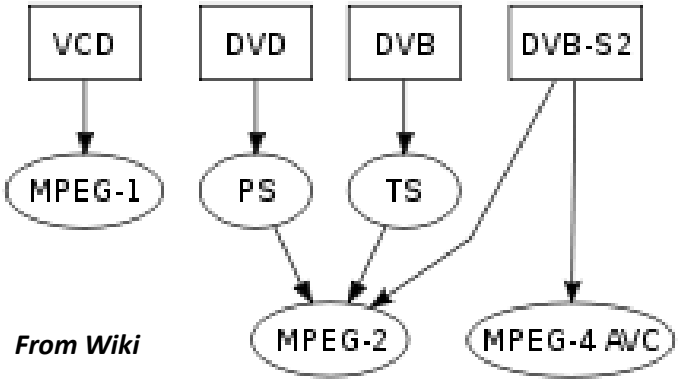
Video Codec Format

- multimedia File knowledge
  - Organization
    - ***ISO / IEC***
    - ***ITU-T***

v • d • e	Multimedia compression formats	[hide]
Video compression	**ISO/IEC**	MJPEG • Motion JPEG 2000 • MPEG-1 • MPEG-2 (Part 2) • MPEG-4 (Part 2/ASP • Part 10/AVC) • HEVC
	ITU-T	H.120 • H.261 • H.262 • H.263 • H.264 • HEVC
	Others	AMV • AVS • Bink • CineForm • Cinepak • Dirac • DV • Indeo • Microsoft Video 1 • OMS Video • Pixlet • RealVideo • RTVideo • SheerVideo • Smacker • Sorenson Video & Sorenson Spark • Theora • VC-1 • VP3 • VP6 • VP7 • VP8 • WMV
Audio compression	**ISO/IEC**	MPEG-1 Layer III (MP3) • MPEG-1 Layer II • MPEG-1 Layer I • AAC • HE-AAC • MPEG-4 ALS • MPEG-4 SLS • MPEG-4 DST
	ITU-T	G.711 • G.718 • G.719 • G.722 • G.722.1 • G.722.2 • G.723 • G.723.1 • G.726 • G.728 • G.729 • G.729.1
	Others	AC3 • AMR • AMR-WB • AMR-WB+ • Apple Lossless • ATRAC • DRA • DTS • FLAC • GSM-FR • GSM-EFR • iLBC • Monkey's Audio • MT9 • μ-law • Musepack • Nellymoser • OptimFROG • OSQ • RealAudio • RTAudio • SD2 • SHN • SILK • Siren • Speex • TwinVQ • Vorbis • WavPack • WMA • True Audio
Image compression	**ISO/IEC/ITU-T**	JPEG • JPEG 2000 • JPEG XR • lossless JPEG • JBIG • JBIG2 • PNG • WBMP
	Others	APNG • BMP • DjVu • EXR • GIF • ICER • ILBM • MNG • PCX • PGF • TGA • QTVR • TIFF
Media containers	**General**	3GP and 3G2 • ASF • AVI • Bink • DMF • DPX • EVO • FLV • GXF • Matroska • MPEG-PS • **MPEG-TS** (M2TS) • MP4 • MXF • M2V (Packetized elementary stream • Elementary stream) • Ogg • QuickTime • RealMedia • RIFF • Smacker • VOB • WebM
	Audio only	AIFF • AU • WAV

Video Codec Format

- MPEG - Moving Picture Experts Group.
  - A working group of ISO/IEC in charge of the development of international standards for compression, decompression, processing and coded representation of moving pictures audio and their combination
  - MPEG –1, MPEG –1 Part 2
  - MPEG –2/H.262, MPEG –2 part 2
  - MPEG –4 ASP, MPEG –4 Part 2
  - MPEG –4 AVC/H.264, MPEG –4 Part 10



From Wiki

Video Codec Format

- MPEG 1
  - The standard for storage and retrieval of moving pictures and audio on storage media. Approved Nov. 1992
  - defined in ISO/IEC-11172-1.
  - –VCD
  - –MP2, MPEG-1 Audio Layer II
  - The standard consists of the following
    1. Systems (storage and synchronization of video, audio, and other data together)
    2. Video (compressed video content)
    3. Audio (compressed audio content)
    4. Conformance testing (testing the correctness of implementations of the standard)
    5. Reference software (example software showing how to encode and decode according to the standard)

Video Codec Format

- MPEG 2
  - MPEG-2 is an extension of the MPEG-1 international standard for digital compression of audio and video signals
  - MPEG-2 is directed at broadcast formats at higher data rates
  - Supports a wide range of bit rates and provides for multichannel surround sound coding
  - ISO/IEC 13818-2 Part1~Part11(or ITU-T H.262)
  - Broadcast TV, cable/satellite TV, HDTV, video
  - services on networks (e.g., ATM)
  - 4~9 Mbits/s, interlaced video, and scalable coding
  - DVD
  - MP3, MPEG-2 Audio Layer III

Video Codec Format

- MPEG 4
  - MPEG-4 absorbs many of the features of MPEG-1 and MPEG-2 and other related standards
  - MPEG-4 is still a developing standard and is divided into a number of parts(Part1~Part28)
  - Key Part -

1. MPEG-4 part 2, Advanced Simple Profile
   - used by codecs such as DiVX, Xvid, Nero Digital and 3ivx, Quicktime 6
2. MPEG-4 part 10, MPEG-4 AVC (Advanced Video Coding)/H.264
   - used by x264 encoder, Nero Digital AVC, Quicktime 7, and high-definition video media like Blu-ray Disc

Video Codec Format

- H264
  - H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding) is a standard for video compression
  - H.264/MPEG-4 AVC is a block-oriented motion-compensation-based codec standard
  - H.264 is designed to provide lower bit rates than mpeg-2 to fit a wide variety of networks and systems
  - To fit network transport , H.264 provide HAL layer (Network Abstraction Layer) and VCL layer (Video Coding Layer)
  - Three Profile
    - Baseline profile
    - Main profile
    - Extension profile

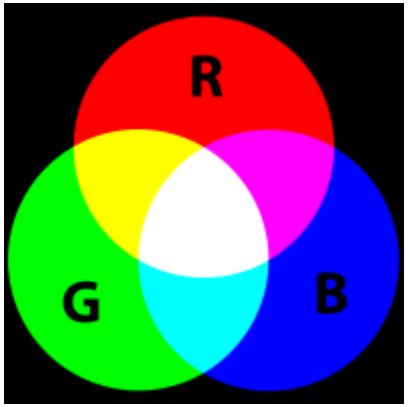
- *Video Device Driver Introduction*
- *Video Codec Format*
- ***Raw Data Format***
- *Framebuffer Driver*
- *Video for Linux Two, V4L2 Driver*
- *OMAP3 Display SubSystem (DSS)*
- *LAB*

Raw Data Format

- RGB color space
  - RGB888
  - RGB565
- YUV color space (YCbCr)
  - YCbCr 4:2:0
  - YCbCr 4:2:2
  - YCbCr 4:4:4

Raw Data Format

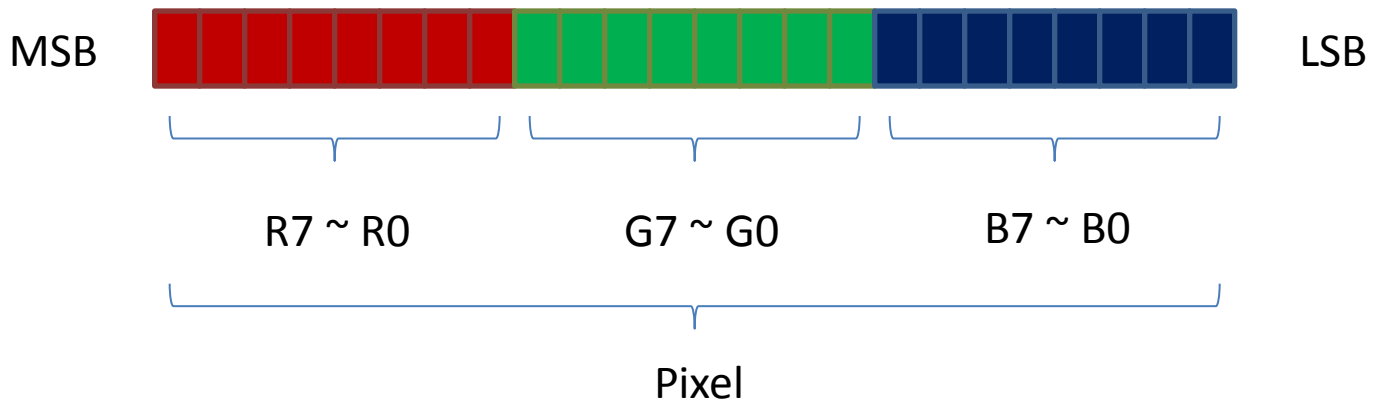
- The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems
- The name of the model comes from the initials of the three additive primary colors, red, green, and blue



RGB color space

Raw Data Format

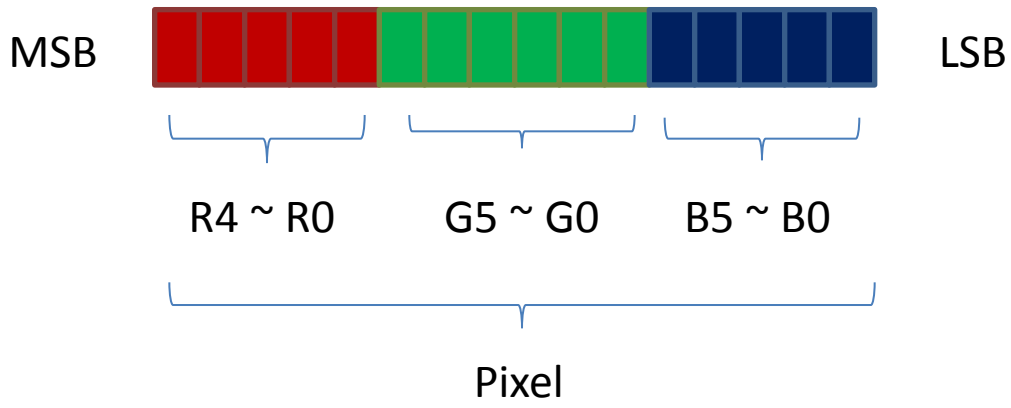
- 24ibt RGB888
  - R7 R6 R5 R4 R3 R2 R1 R0 G7 G6 G5 G4 G3 G2 G1 G0 B7 B6 B5 B4 B3 B2 B1 B0



  - How much memory we have to allocate ?
    - If the resolution is 1024 x 768
    - We need 1024 x 768 x 24 (bits) / 8 = 1024 x 768 x 3 byte

Raw Data Format

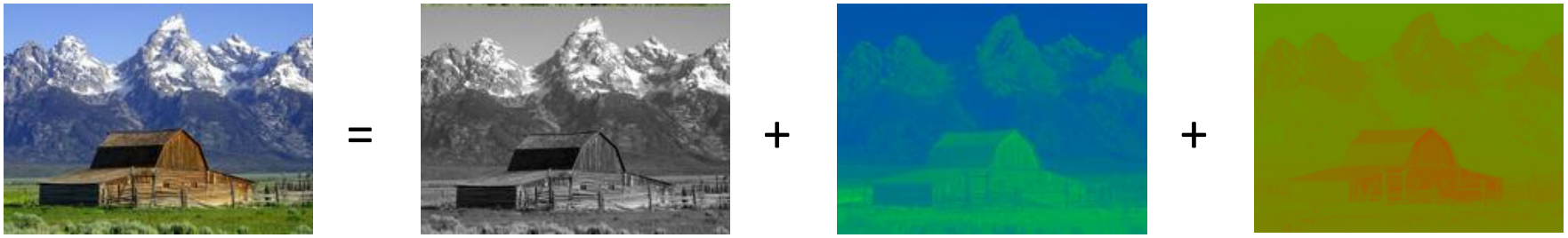
- 16ibt RGB565
  - R4 R3 R2 R1 R0 G5 G4 G3 G2 G1 G0 B4 B3 B2 B1 B0



  - How much memory we have to allocate ?
    - If the resolution is 1024 x 768
    - We need 1024 x 768 x 16 (bits) / 8 = 1024 x 768 x 2 byte

Raw Data Format

- YUV Color Space
- The scope of the terms YUV, Y'UV ,YCbCr, YPbPr
  - Y means Luminance、Luma
  - U means Chrominance
  - V means Chroma
- YUV and Y'UV were used for a specific analog encoding of color information in television systems



Original ***Luminance*** ***Chrominance*** ***Chroma***

Raw Data Format

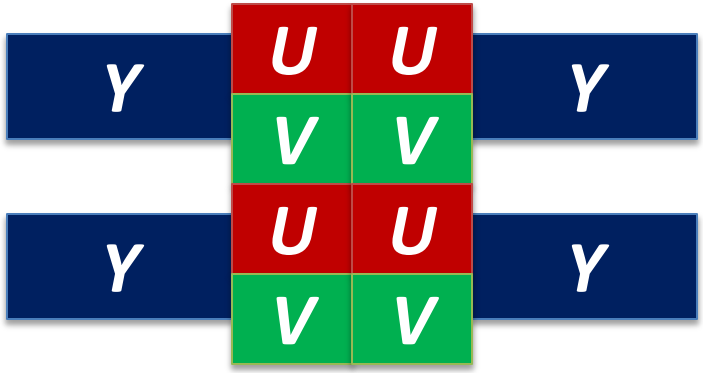
- YCbCr was used for digital encoding of color information
- The most concept subsample, different to pixel concept
- Conversion to/from RGB

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Raw Data Format

- YUV444
- Y3 Y2 Y1 Y0 U3 U2 U1 U0 V3 V2 V1 V0
  - (Y3 U3 V3) (Y2 U2 V2) (Y1 U1 V1) (Y0 U0 V0)
  - Per subsample Y1U1V1 is 3 byte

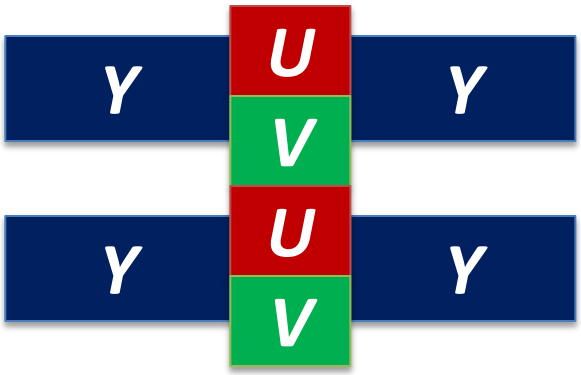


Per Subsample

  - How much memory we have to allocate ?
    - If the resolution is 1024 x 768
    - We need 1024 x768 x 3 byte

Raw Data Format

- YUV422
- Y3 Y2 Y1 Y0 U1 U0 V1 V0
  - (Y3 ½ U1 ½ V1) (Y2 ½ U1 ½ V1) (Y1 ½ U0 ½ V0) (Y0 ½ U0 ½ V0)
  - Per subsample is 1+1/2+1/2 = 2 byte



Per Subsample

  - How much memory we have to allocate ?
    - If the resolution is 1024 x 768
    - We need 1024 x768 x (1+1/2+1/2) = 1024 x 768 x2 byte

Raw Data Format

- YUV420
- Y3 Y2 Y1 Y0 U0 V0
  - (Y3 ¼ U0 ¼ V0) (Y2 ¼ U0 ¼ V0) (Y1 ¼ U0 ¼ V0) (Y0 ¼ U0 ¼ V0)
  - Per subsample is 1+1/4+1/4 = 3/2 byte

Per Subsample

  - How much memory we have to allocate ?
    - If the resolution is 1024 x 768
    - We need 1024 x768 x (1+1/4+1/4) = 1024 x 768 x 3/2 byte

- *Video Device Driver Introduction*
- *Video Codec Format*
- *Raw Data Format*
- ***Framebuffer Driver***
- *Video for Linux Two, V4L2 Driver*
- *OMAP3 Display SubSystem (DSS)*
- *LAB*

Framebuffer Driver

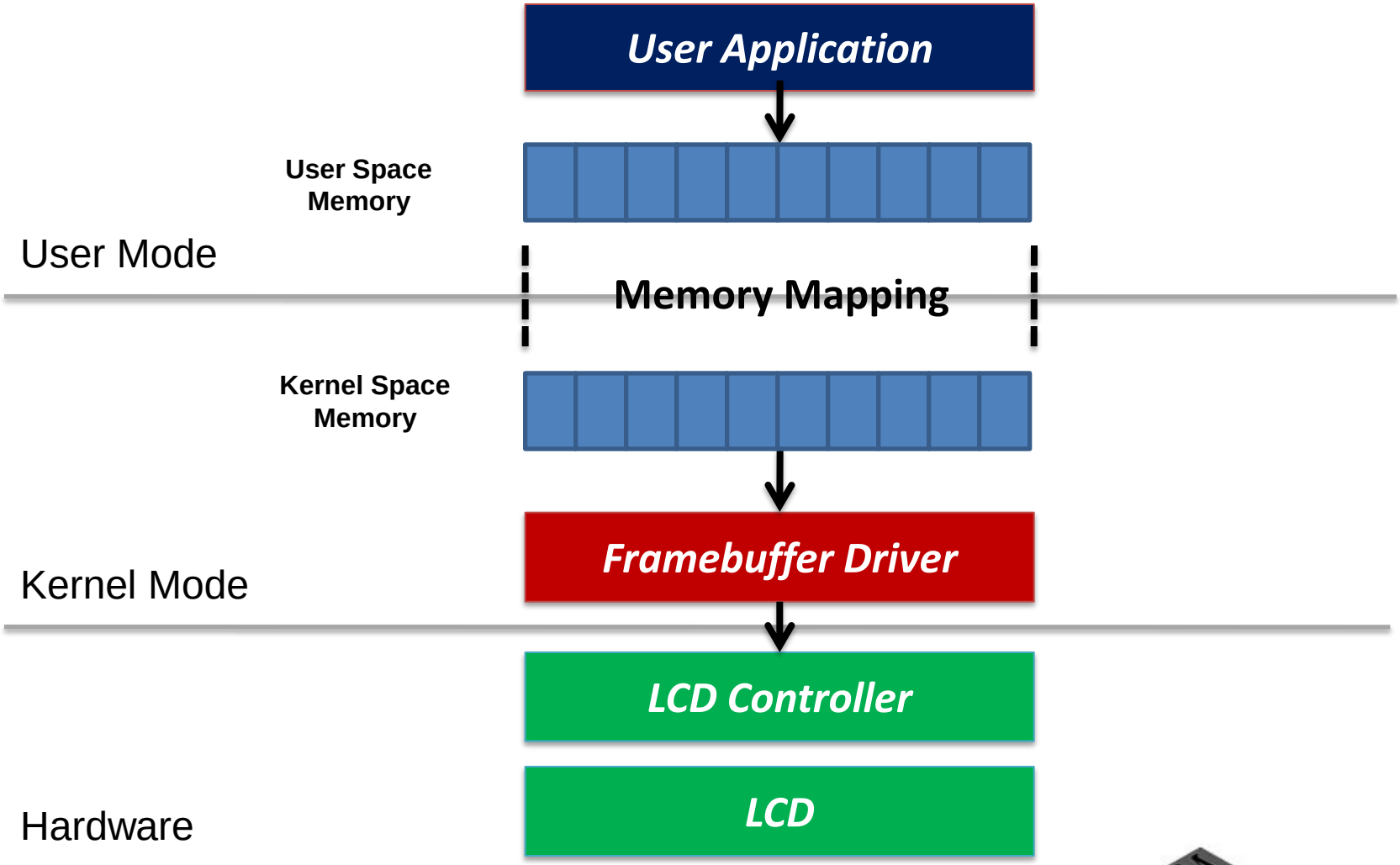
- What is Framebuffer
  - A framebuffer is a video output device that drives a ***video display from a memory buffer*** containing a complete frame of data
  - Use ***mmap*** to allocate a block of memory space in kernel, and create virtual memory in user space, user space directly access framebuffer memory
  - The information in the buffer typically consists of color values for every pixel (point that can be displayed) on the screen
  - Typically use /dev/fbx (x=any number) as framebuffer device
  - Support various video input signal

Framebuffer Driver

- What is Framebuffer
  - The total amount of the memory required to drive the framebuffer depends on
    - ***Resolution***
    - ***Color depth***
    - ***Palette size***
  - Framebuffers are commonly accessed via a memory mapping directly to memory space
  - ***Double/Tripple*** buffering technology makes display of frames more smooth

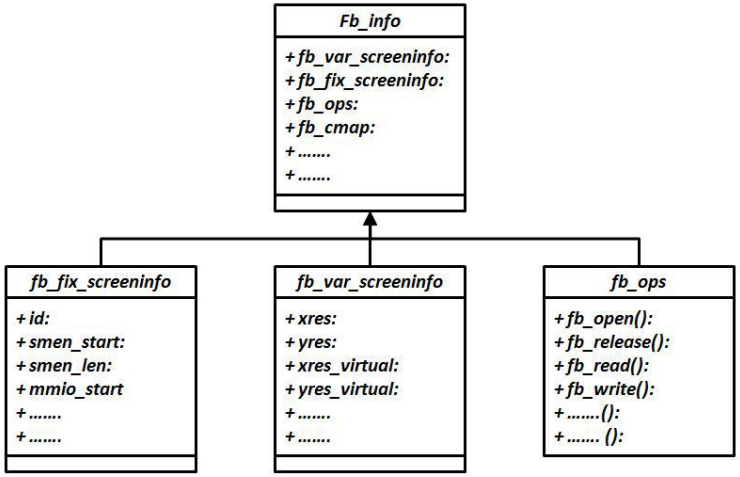
Framebuffer Driver

- Framebuffer Architecture



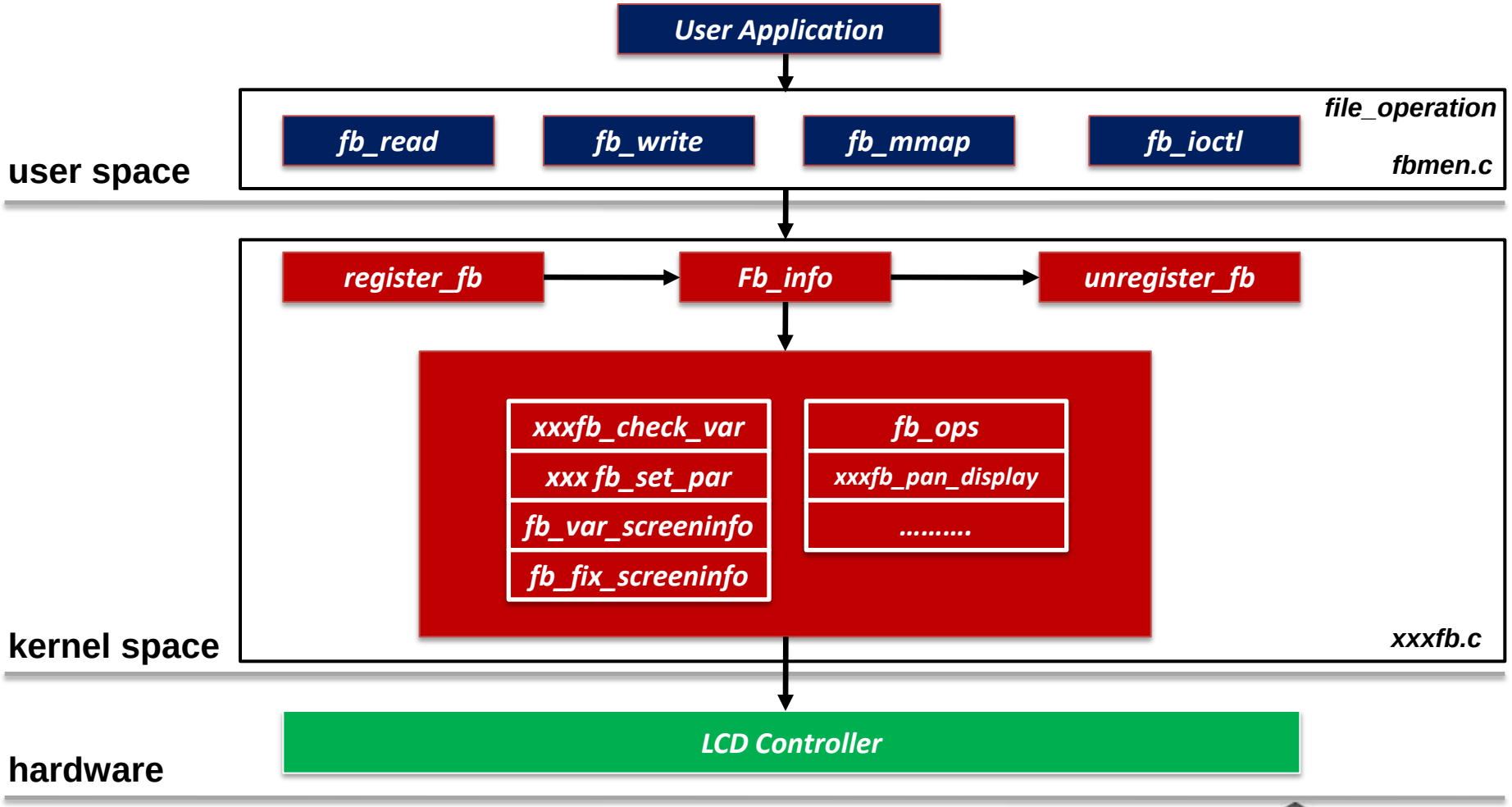
Framebuffer Driver

- Define
  - linux/include/linux/fb.h
  - linux/drivers/video/fbmem.c
  - linux/drivers/video/omap2/ (only OMAP platform)
  - Major number：29
- Internal data structure of framebuffer
  - struct fb_info
    - Framebuffer information structure
  - struct fb_var_screeninfo
    - User defined properties
  - struct fb_fix_screeninfo
    - Hardware properties



Framebuffer Driver

- Framebuffer Driver programming arcticture



Framebuffer Driver

- fb_info – framebufer information structure

```
struct fb_info {
        int node;
        int flags;
        struct fb_var_screeninfo var;      /* Current var                */
        struct fb_fix_screeninfo fix;      /* Current fix                */
        struct fb_monspecs monspecs;       /* Current Monitor specs      */
        struct work_struct queue;          /* Framebuffer event queue    */
        struct fb_pixmap pixmap;           /* Image hardware mapper      */
        struct fb_pixmap sprite;           /* Cursor hardware mapper     */
        struct fb_cmap cmap;               /* Current cmap               */
        struct list_head modelist;         /* mode list                  */
        struct fb_videomode *mode;         /* current mode               */

        struct backlight_device *bl_dev; /* assigned backlight device    */
        struct mutex bl_curve_mutex;       /* Backlight level curve      */
}
```


Framebuffer Driver

- fb_var_screeninfo - used to describe the features of a video card or LCD Controller that are user defined

```
struct fb_var_screeninfo {
    __u32 xres;                 /* visible resolution            */
    __u32 yres;
    __u32 xres_virtual;         /* virtual resolution            */
    __u32 yres_virtual;
    __u32 xoffset;              /* offset from virtual to visible */
    __u32 yoffset;              /* resolution                    */

    __u32 bits_per_pixel;       /* guess what                    */
    __u32 grayscale;            /* != 0 Graylevels instead of colors */

    struct fb_bitfield red;     /* bitfield in fb mem if true color, */
    struct fb_bitfield green;   /* else only length is significant */
    struct fb_bitfield blue;
    struct fb_bitfield transp;  /* transparency                  */
    .....
    .....
```


Framebuffer Driver

- fb_var_screeninfo(cont.) - Timing: All values in pixclocks, except pixclock

```
    ……..
    ……..
    __u32 pixclock;              /* pixel clock in ps (pico seconds)     */
    __u32 left_margin;           /* time from sync to picture            */
    __u32 right_margin;          /* time from picture to sync            */
    __u32 upper_margin;          /* time from sync to picture            */
    __u32 lower_margin;
    __u32 hsync_len;             /* length of horizontal sync            */
    __u32 vsync_len;             /* length of vertical sync              */
    __u32 sync;                  /* see FB_SYNC_*                        */
    __u32 vmode;                 /* see FB_VMODE_*                       */
    __u32 rotate;                /* angle we rotate counter clockwise */
    __u32 reserved[5];           /* Reserved for future compatibility */
}
```


Framebuffer Driver

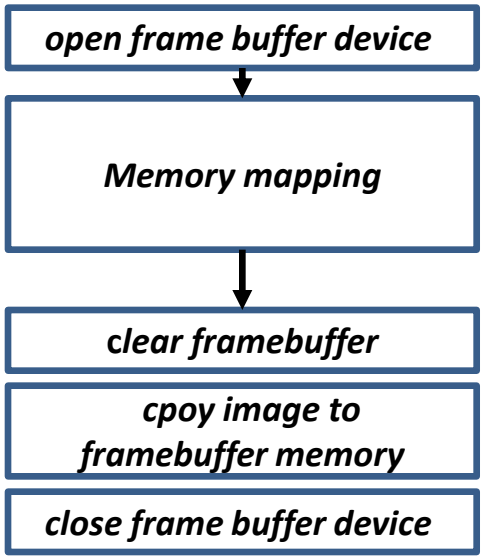
- fb_fix_screeninfo - Fixed information about the video hardware, such as the start address and size of frame buffer

```
struct fb_fix_screeninfo {
        char id[16];                    /* identification string eg "TT Builtin" */
        unsigned long smem_start;       /* Start of frame buffer mem             */
                                        /* (physical address)                    */
        __u32 smem_len;                 /* Length of frame buffer mem            */
        __u32 type;                     /* see FB_TYPE_*                         */
        __u32 type_aux;                 /* Interleave for interleaved Planes     */
        __u32 visual;                   /* see FB_VISUAL_*                       */
        __u16 xpanstep;                 /* zero if no hardware panning           */
        __u16 ypanstep;                 /* zero if no hardware panning           */
        __u16 ywrapstep;                /* zero if no hardware ywrap             */
        __u32 line_length;              /* length of a line in bytes             */
        unsigned long mmio_start;       /* Start of Memory Mapped I/O            */
                                        /* (physical address)                    */
        __u32 mmio_len;                 /* Length of Memory Mapped I/O           */
        __u32 accel;                    /* Indicate to driver which              */
                                        /*  specific chip/card we have           */
        __u16 reserved[3];              /* Reserved for future compatibility     */
};
```


Framebuffer Driver

- framebuffer Programming on linux

```
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <linux/fb.h>

int main(void){
   int fb;
   unsigned char* fb_mem;

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

   fb_mem = mmap (NULL, 320*240*2,
                  PROT_READ|PROT_WRITE,
                        MAP_SHARED, fb, 0);

   memset(fb_mem, 0, 320*240*2);

   memcpy(fb_mem, (unsigned char *)ScreenBitmap,
                                 320*240*2);
   close(fb);

}
```



Framebuffer Driver

- Get the desktop picture

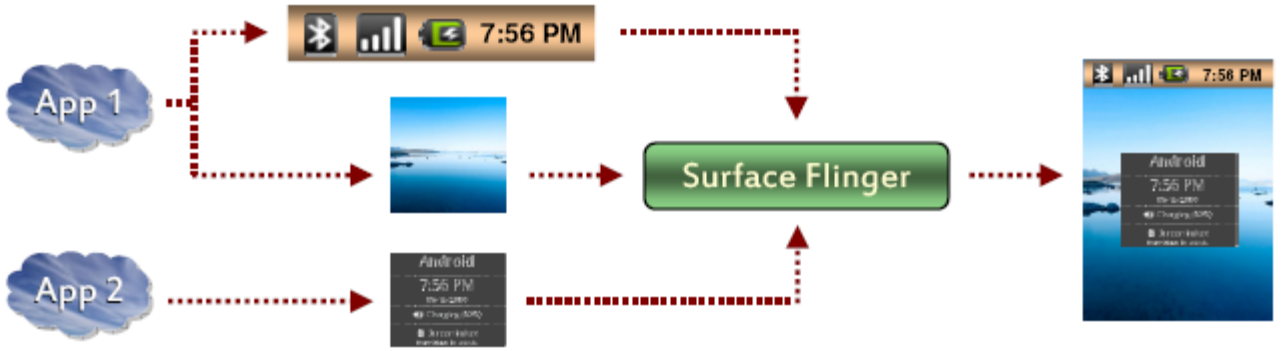
```
dd if=/dev/fb0 of=fbdata bs=1024 count=768 (if the resolution is 1024x768)
```

- Put the fbdata to framebuffer

```
dd if=fbdata of=/dev/fb0 bs=1024 count=768
```


Framebuffer Driver – Android SurfaceFlinger

- SurfaceFlinger
  - Proviides a system-wide surface “composer” to render all the surfaces in a frame buffer
    - Can combiined 2D and 3D surffaces
    - Can use OpenGL ES and 2D hardware accellerattor for its composiittiions



From esmertec

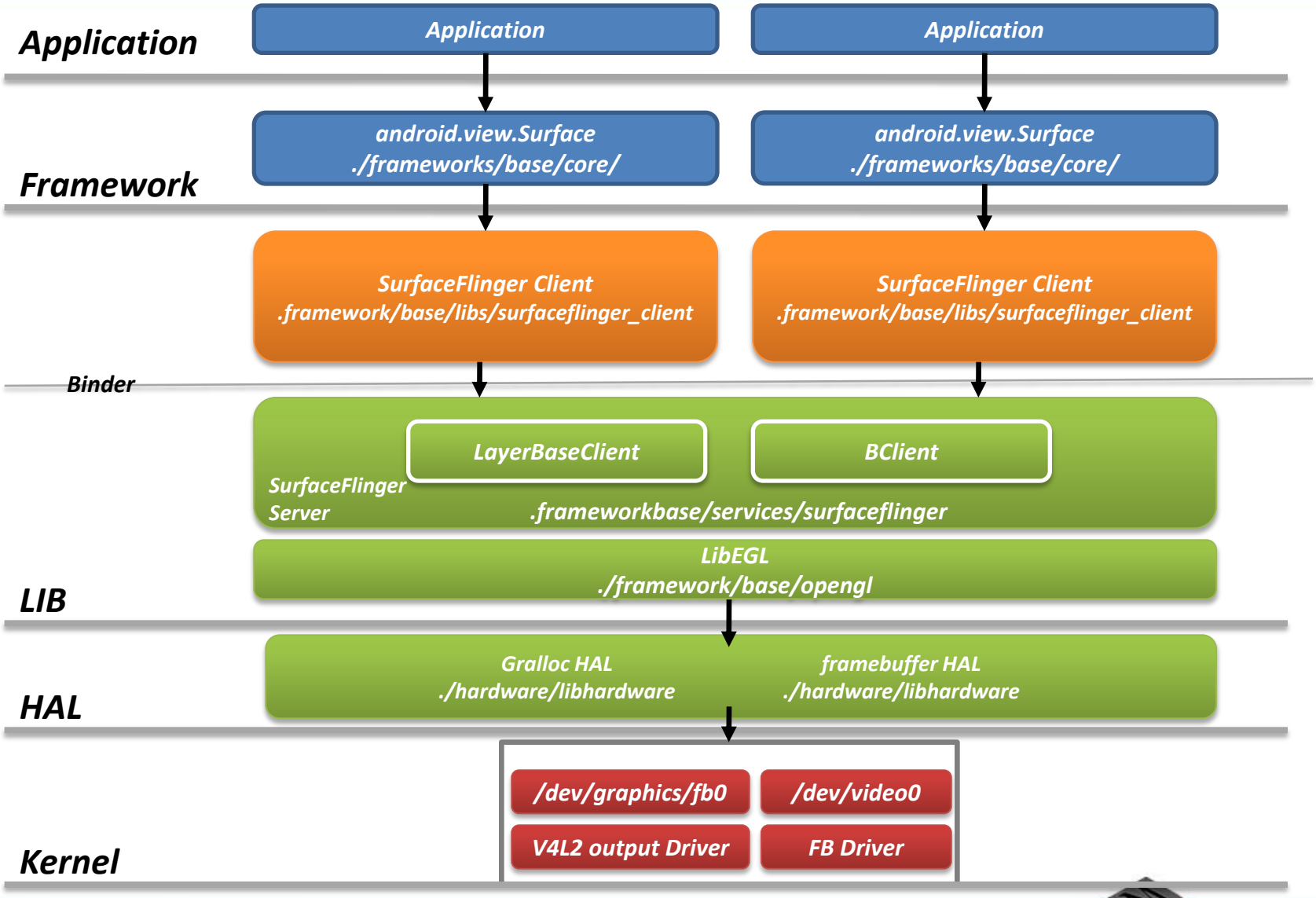
Framebuffer Driver – Android SurfaceFlinger

- SurfaceFlinger
  - ./frameworks/base/libs/surfaceflinger/
  - ./frameworks/base/libs/ui
  - ./frameworks/base/libs/surfaceflinger_client
  - ./frameworks/base/core/java/Android/view/
  - ./frameworks/base/core/jni/Android_view_Surface.cpp
  - Android Surface Sequence
    1. new surfaceflingerclient(), it will createconnection(), and new a client
    2. createsurface, it will new layer or layerblur or layerdim by z- blend order
    3. createoverlay, if layer support overlay
    4. register buffer
    5. draw something on canvas(line, text, bitmap, rect…) which attach above buffer
    6. post buffer

Framebuffer Driver – Android SurfaceFlinger

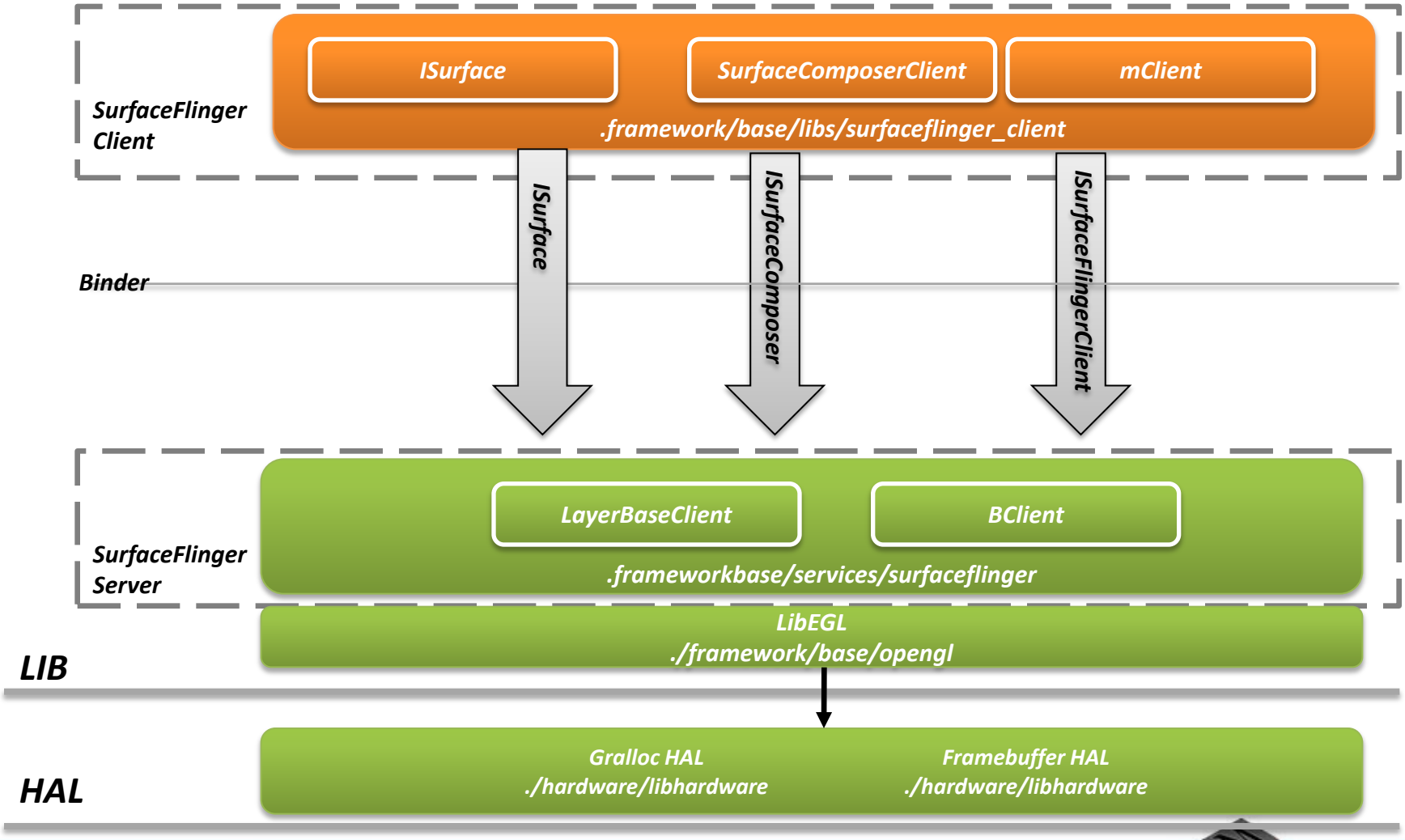
- Android framebuffer
  - Android relies on the standard frame buffer device (***/dev/fb0*** or ***/dev/graphics/fb0***) and driver as described in the ***linux/fb.h*** kernel header file
  - Android makes two requirements of the driver
    - Using the ***FBIOGET_FSCREENINFO*** and ***FBIOGET_VSCREENINFO*** Input / Output Control (ioctl) calls
    - using ***FBIOPUT_VSCREENINFO*** ioctl to attempt to create a virtual display twice the size of the physical screen and to set the pixel format to ***rgb_565***

Framebuffer Driver – Android SurfaceFlinger



Framebuffer Driver – Android SurfaceFlinger

- SurfaceFlinger Server and Client

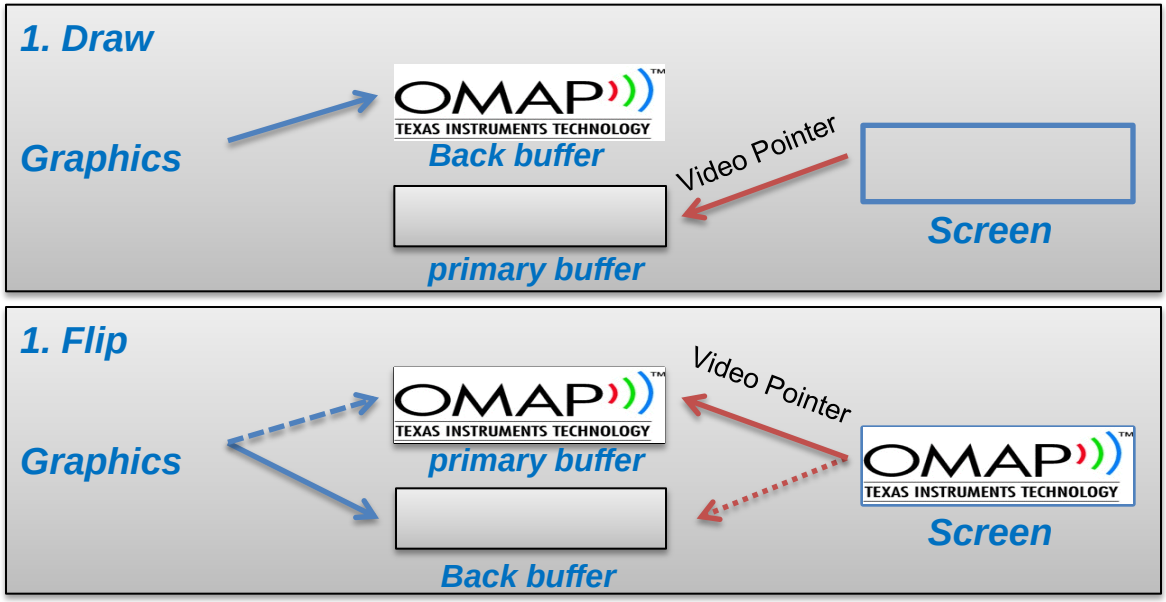


Framebuffer Driver – Android SurfaceFlinger

- EGL Display Surface
  - LibEGL compose all frame layer created by surfaceclient and transfer data to HAL layer(Gralloc)
  - Use ***copyFrontToBack*** and ***swapBuffers*** to flipping(double buffering)
  - Programming
    1. postFrameBuffer(***./framework/base/services/surfaceflinger***)
    2. Flip(***./framework/base/services/surfaceflinger***)
    3. eglSwapBuffers(***./framework/base/opengl***)
    4. queueBuffer(***./framework/base/libs/surfaceflinger_client***)
    5. fbpost(***./hardware/libhardware/modules/gralloc***)

Framebuffer Driver – Android SurfaceFlinger

- EGL Display Surface
  - copyFrontToBack
    - copy Primary Surface data to Back Buffer
  - swapBuffer
    - use to flip videopointer



Flip

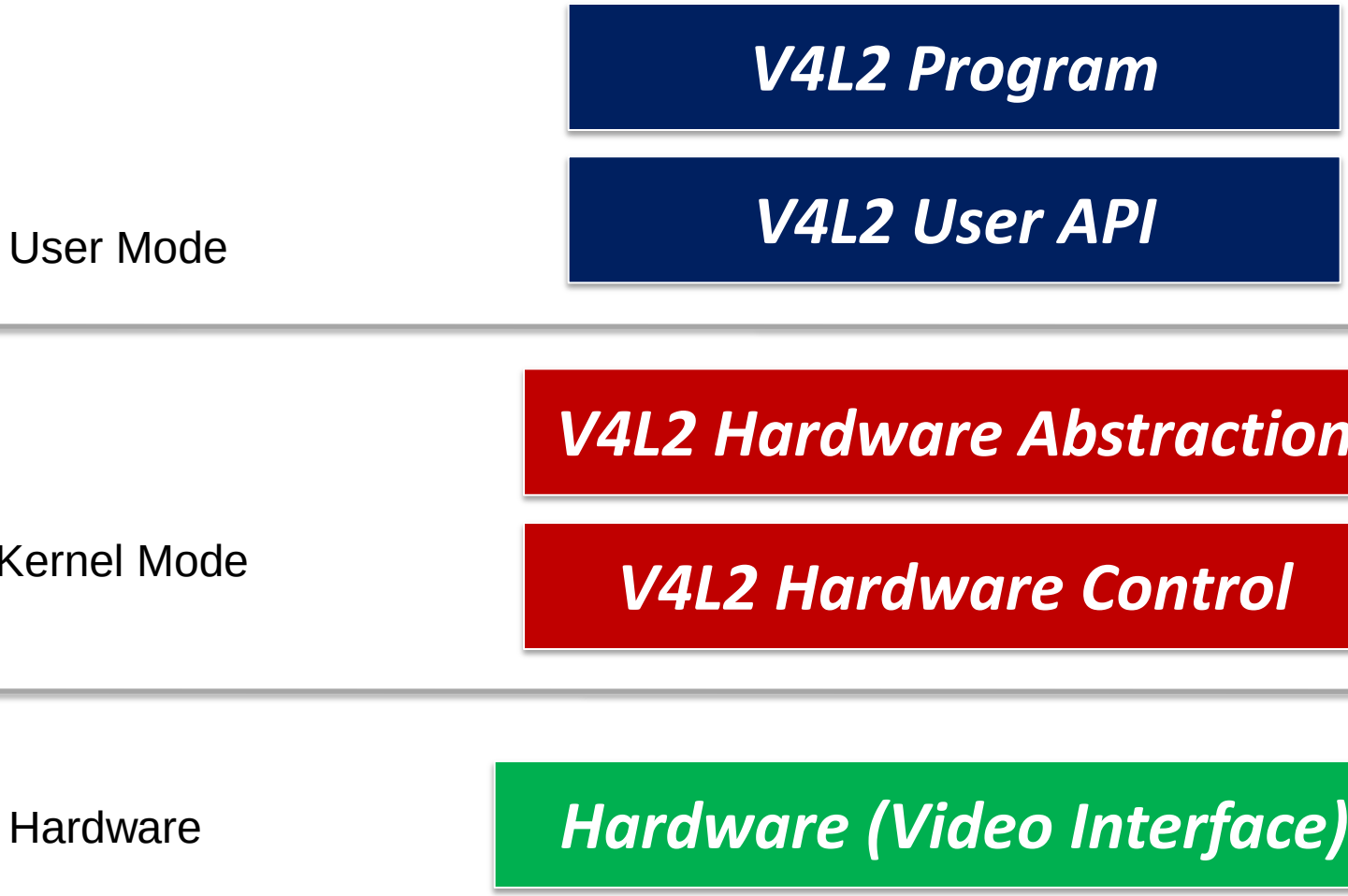
- *Video Device Driver Introduction*
- *Video Codec Format*
- *Raw Data Format*
- *Framebuffer Driver*
- ***Video for Linux Two, V4L2 Driver***
- *OMAP3 Display SubSystem (DSS)*
- *LAB*

Video for Linux Two, V4L2 Driver

- V4L
  - Video4Linux is a video capture application programming interface for Linux and proposed by Alan Cox
  - V4L support many USB webcams , TV tuners, and other devices
  - Video4Linux is closely integrated with the Linux kernel
- V4L2
  - V4L2 is the second version of V4L ,and proposed by Bill Dirks
  - V4L2 is a new design video capture APIs , support more capture device , but incompatible with V4L
- Software Support V4L2
  - aMSN , Gstreamer , Mplayer , MythTV , OpenCV , Skype , VLC media player

Video for Linux Two, V4L2 Driver

- V4L2 Architecture



Video for Linux Two, V4L2 Driver

- V4L2 Architecture
  - User mode v4l2 library
    - User space buffer handling
    - User space device handling
  - Kernel mode v4l2 driver
    - Hardware control
      - » Exposure control 、White Balance Control、Zoom Control
    - Hardware abstraction
      - » Command interface、resolution handling、raw data format handling、buffer handling

Video for Linux Two, V4L2 Driver

- V4L2 Architecture
  - V4L2 API
    - Defined in linux/videodev2.h or videodev.h
    - Operation Function
      - » Control and read/write v4l2 device
    - Ioctl Commands
      - » To determinate driver operation mode
      - » Use ioctl function to accomplish the operation
      - » Set / get the v4l2 data structure

Video for Linux Two, V4L2 Driver

- Operation Functions
  - Open() - Open a V4L2 device
  - Close() - Close a V4L2 device
  - Ioctl() – Program a V4L2 device
  - Mmap() – Map device memory (kernel space) into user space
  - Munmap()- Unmap device memory
  - Read() – Read from V4L2 device
  - Write() – Write to V4L2 device
  - Select() – Synchronous I/O multiplexing
  - Poll() – Wait for some event on a descriptor

Video for Linux Two, V4L2 Driver

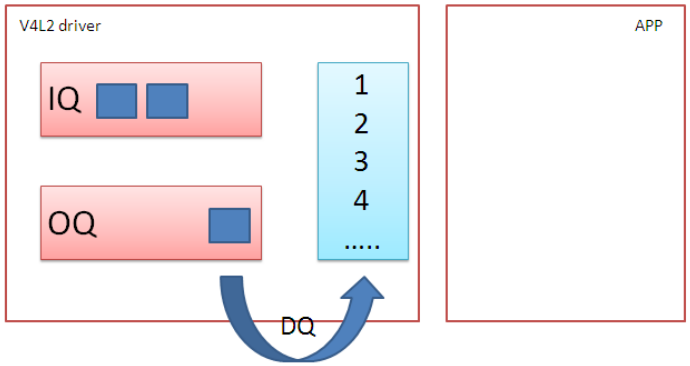
- Methods to read from or write to a device
  - Read/Write method
    - Standard read and write copy data from driver buffer to a new buffer in application process's memory space
  - Memory Mapping method
- Use Queue to capture video data
  - Use two buffers queues to manage devices drivers
    - An incoming queue(IQ):free device buffer (ready to write)
    - An outgoing queue (OQ): full data device buffer

Video for Linux Two, V4L2 Driver

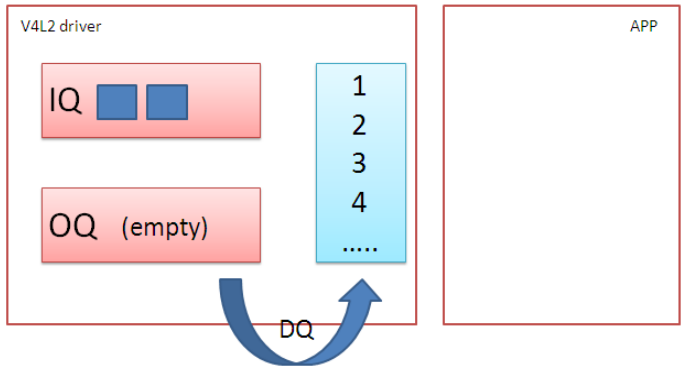
- Methods to read from or write to a device
  - Read/Write method
    - Standard read and write copy data from driver buffer to a new buffer in application process's memory space
  - Memory Mapping method
- Use Queue to capture video data
  - Use two buffers queues to manage devices drivers
    - An incoming queue(IQ):free device buffer (ready to write)
    - An outgoing queue (OQ): full data device buffer

Video for Linux Two, V4L2 Driver

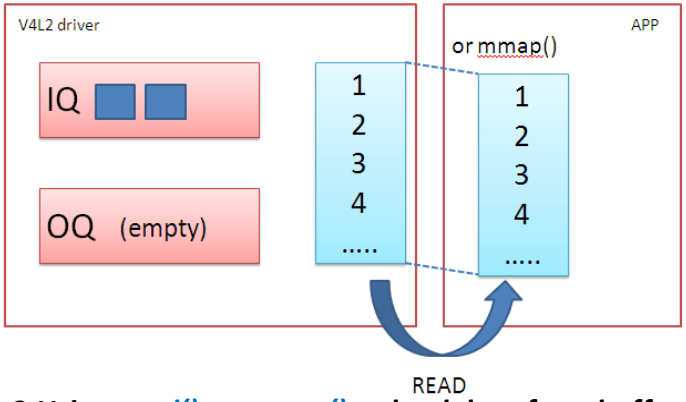
- Queue to capture video data



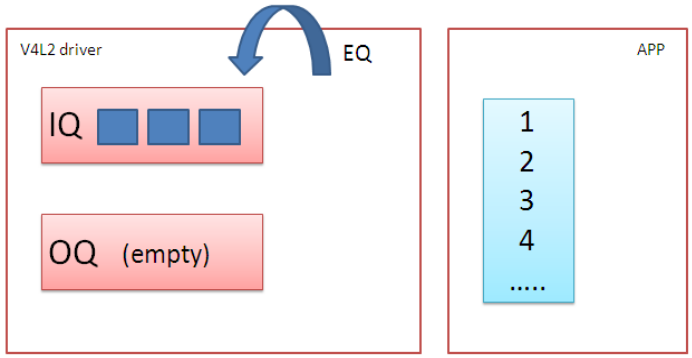
1.A device buffer use to store video image space



2.Application use VIDIOC_DQBUF , driver get a device buffer from outgoing queue



3.Using read() or mmap() to load data from buffer



4.Using VIDIOC_QBUF , driver push null device buffer to incoming queue

Video for Linux Two, V4L2 Driver

- Important V4L2 Processing APIs
  - VIDIOC_REQBUFS
    - Application requests the driver to allocate frames
  - VIDIOC_QUERYBUF
    - Application requests each allocated buffer characteristics
  - VIDIOC_QBUF
    - application requests the driver to enqueue each frame
  - VIDIOC_STREAMON
    - application starts the acquisition chain
  - VIDIOC_DQBUF
    - application waits and dequeues a frame

Video for Linux Two, V4L2 Driver

- More V4L2 Processing APIs
  - VIDIOC_CROPCAP -- Information about the video cropping and scaling abilities.
  - VIDIOC_ENUMAUDIO -- Enumerate audio inputs
  - VIDIOC_ENUMAUDOUT -- Enumerate audio outputs
  - VIDIOC_ENUM_FMT -- Enumerate image formats
  - VIDIOC_ENUM_FRAMESIZES -- Enumerate frame sizes
  - VIDIOC_ENUM_FRAMEINTERVALS -- Enumerate frame intervals
  - VIDIOC_ENUMINPUT -- Enumerate video inputs
  - VIDIOC_ENUMOUTPUT -- Enumerate video outputs
  - VIDIOC_ENUMSTD -- Enumerate supported video standards
  - VIDIOC_G_AUDIO, VIDIOC_S_AUDIO -- Query or select the current audio input and its attributes
  - VIDIOC_G_AUDOUT, VIDIOC_S_AUDOUT -- Query or select the current audio output
  - VIDIOC_G_MPEGCOMP, VIDIOC_S_MPEGCOMP -- Get or set compression parameters
  - VIDIOC_G_CROP, VIDIOC_S_CROP -- Get or set the current cropping rectangle

Video for Linux Two, V4L2 Driver

- More V4L2 Processing APIs
  - VIDIOC_G_CTRL, VIDIOC_S_CTRL -- Get or set the value of a control
  - VIDIOC_G_EXT_CTRLS, VIDIOC_S_EXT_CTRLS,
  - VIDIOC_TRY_EXT_CTRLS -- Get or set the value of several controls, try control values.
  - VIDIOC_G_FBUF, VIDIOC_S_FBUF -- Get or set frame buffer overlay parameters.
  - VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT -- Get or set the data(image) format and try a format.
  - VIDIOC_G_FREQUENCY, VIDIOC_S_FREQUENCY -- Get or set tuner or modulator radio frequency
  - VIDIOC_G_INPUT, VIDIOC_S_INPUT -- Query or select the current video input
  - VIDIOC_G_JPEGCOMP, VIDIOC_S_JPEGCOMP --
  - VIDIOC_G_MODULATOR, VIDIOC_S_MODULATOR -- Get or set modulator attributes
  - VIDIOC_G_OUTPUT, VIDIOC_S_OUTPUT -- Query or select the current video output
  - VIDIOC_G_PARM, VIDIOC_S_PARM -- Get or set streaming parameters
  - VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY -- Query or request the access priority associated with a file descriptor

Video for Linux Two, V4L2 Driver

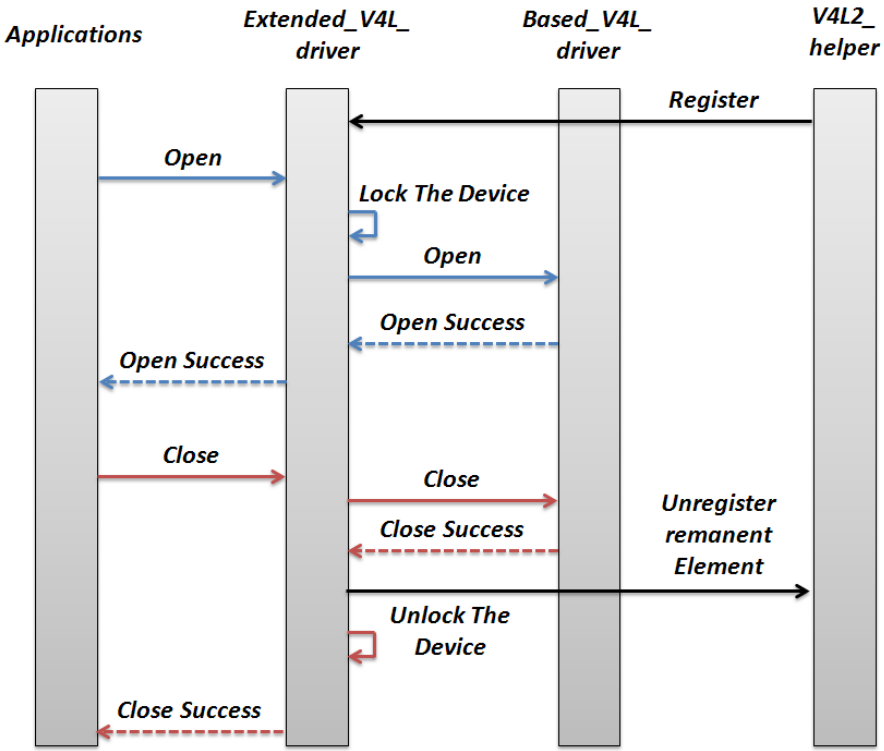
- More V4L2 Processing APIs
  - VIDIOC_G_SLICED_VBI_CAP -- Query sliced VBI capabilities
  - VIDIOC_G_STD, VIDIOC_S_STD -- Query or select the video standard of the current input
  - VIDIOC_G_TUNER, VIDIOC_S_TUNER -- Get or set tuner attributes
  - VIDIOC_LOG_STATUS -- Log driver status information
  - VIDIOC_OVERLAY -- Start or stop video overlay
  - ***VIDIOC_QBUF -- to enqueue an empty (for capturing) or filled (output) buffer in the driver's incoming queue.***
  - ***VIDIOC_DQBUF -- to dequeue a filled (for capturing) or displayed (output) buffer from the driver's outgoing queue.***
  - ***VIDIOC_QUERYBUF -- Query the status of a buffer***
  - VIDIOC_QUERYCAP -- Query device capabilities
  - VIDIOC_QUERYCTRL, VIDIOC_QUERYMENU -- Enumerate controls and menu control items
  - VIDIOC_QUERYSTD -- Sense the video standard received by the current input
  - ***VIDIOC_REQBUFS -- Streaming I/O method(Memory Mapping or User Pointer)***
  - ***VIDIOC_STREAMON, VIDIOC_STREAMOFF -- Start or stop streaming I/O***

Video for Linux Two, V4L2 Driver

- V4L2 control sequence
  - Open/close access the hardware
    - This step corresponds to the first and the last access to hardware via the extended driver
  - Control the hardware
    - This step corresponds mainly to the ioctl() entry points
  - Get video frames
    - This step will show how to managed the incoming data and provide them to the Application

Video for Linux Two, V4L2 Driver

- V4L2 control sequence

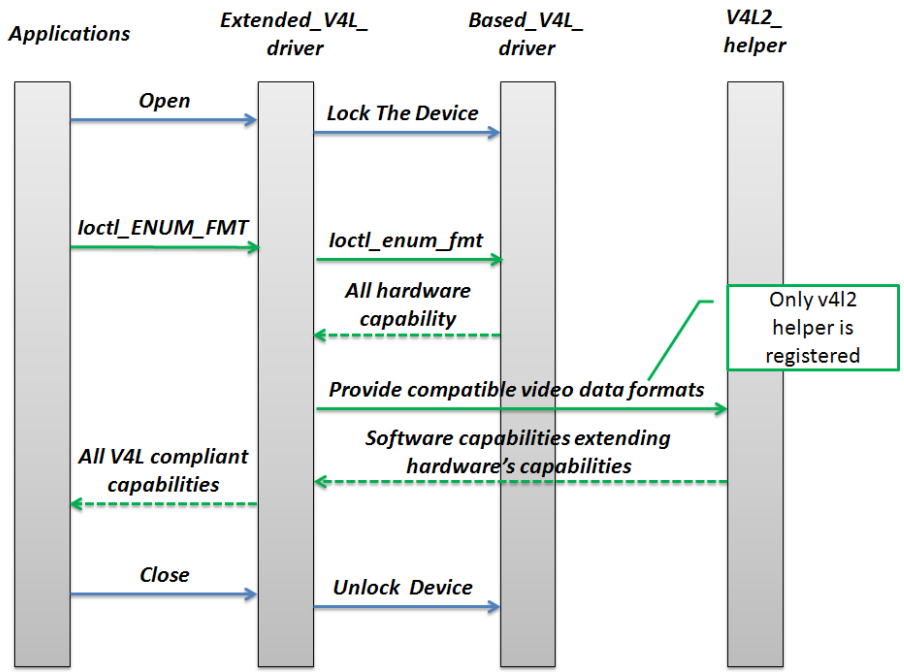


Open/close access the hardware

1. The application opens /dev/video1
2. The extended device driver requests the base driver to lock device
3. The application control hardware and get frames
4. The application closes /dev/video1
5. The extended device driver wipes out all internal allocations
6. The extended device driver requests the base driver to unlock the device

Video for Linux Two, V4L2 Driver

- V4L2 control sequence

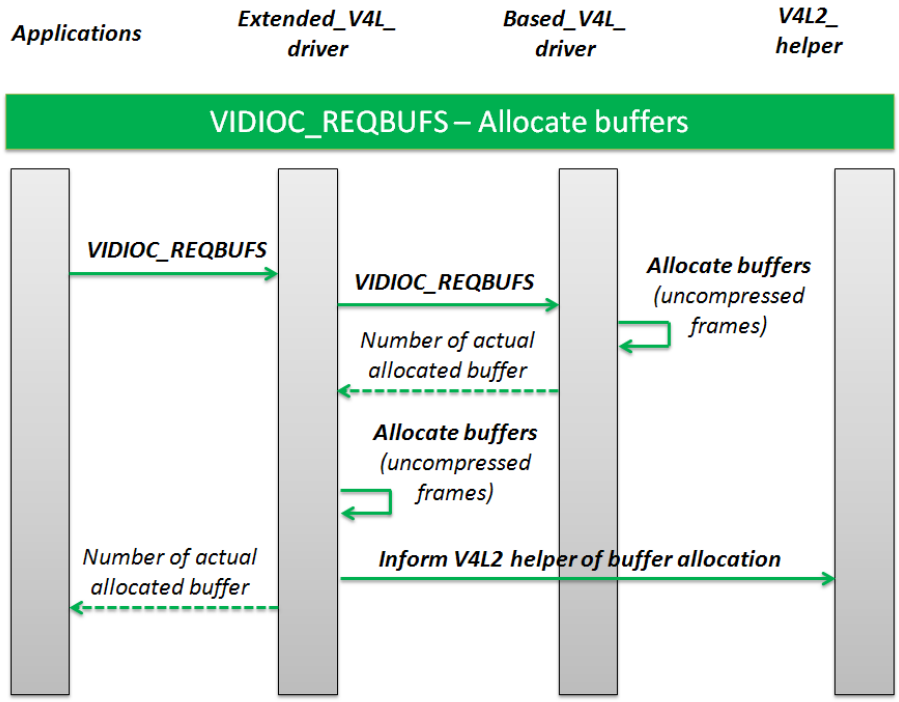


Control the harward

1. The application gets hardware capabilities with ***VIDIOC_QUERYCAP***
2. The application enumerates inputs with ***VIDIOC_ENUMINPUT***
3. The application sets selected input with ***VIDIOC_S_INPUT***
4. The application enumerates supported video standards with ***VIDIOC_ENUMSTD***

Video for Linux Two, V4L2 Driver

- V4L2 control sequence

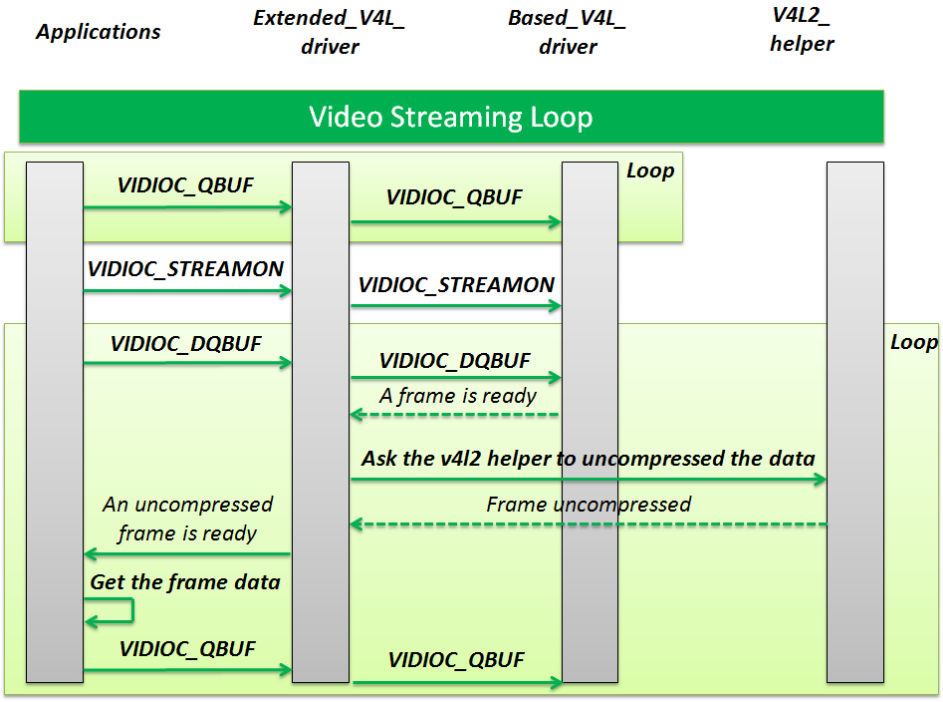


Get video frame – allocate buffer

1. The application requests the driver to allocate frames with ***VIDIOC_REQBUFS***.
2. The extended v4l driver requests the v4l helper daemon to allocate video buffers

Video for Linux Two, V4L2 Driver

- V4L2 control sequence

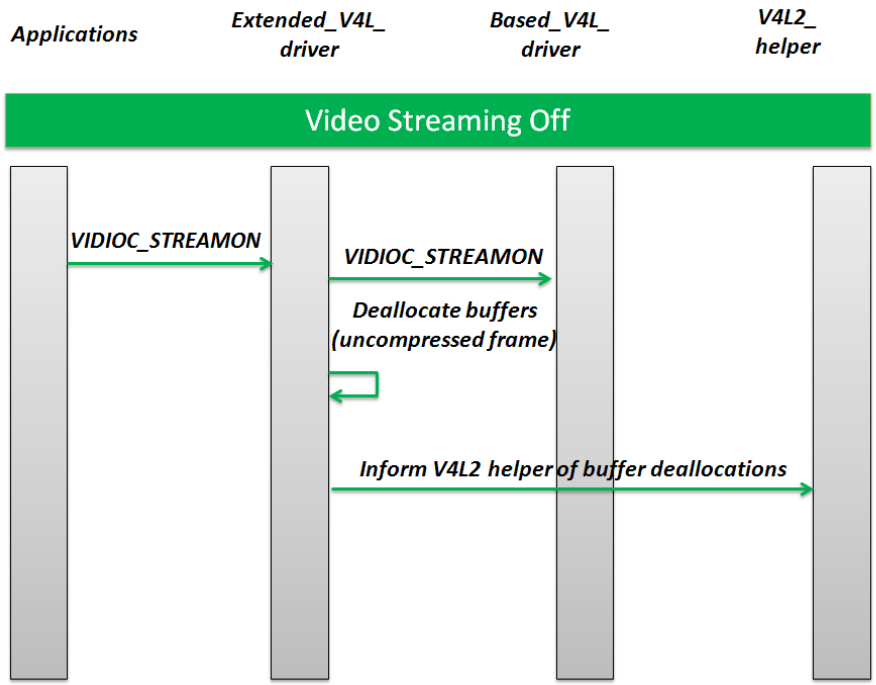


Get video frame – streaming loop

1. The application requests the driver to enqueue each frame with ***VIDIOC_QBUF.***
2. The application starts the acquisition chain with ***VIDIOC_STREAMON.***
3. The application waits and dequeues a frame with ***VIDIOC_DQBUF.***
4. As soon as the frame is processed, the application calls ***VIDIOC_QBUF*** to re-enqueue this buffer.
5. The ***VIDIOC_DQBUF/QBUF*** mechanism is repeated to get each frame sequentially

Video for Linux Two, V4L2 Driver

- V4L2 control sequence



Get video frame – streaming off

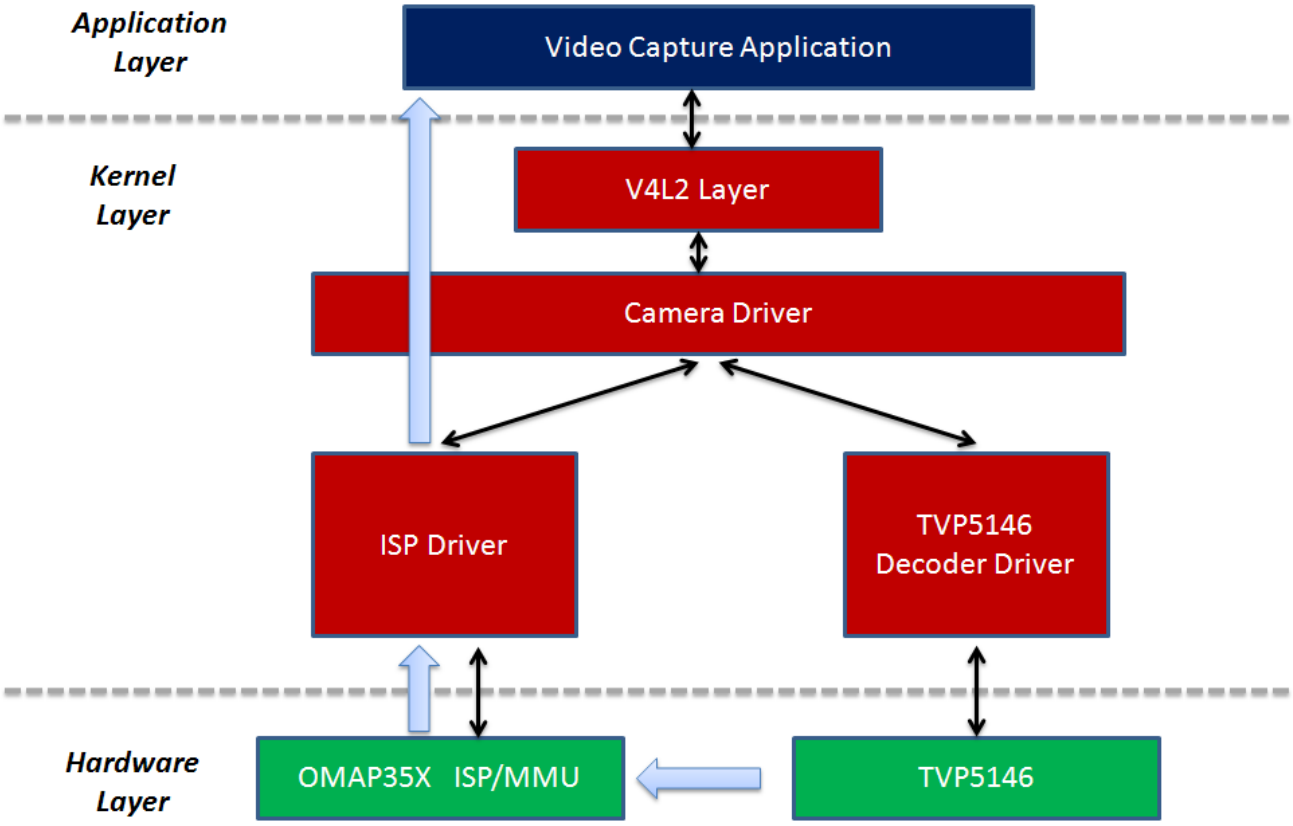
1. The application stops the acquisition with ***VIDIOC_STREAMOFF***

Video for Linux Two, V4L2 Driver

- Video Capture Driver Architecture for omap
  - The camera module support two interface
    - S-video SD input in BT.656 format
    - Composite SD input in BT.656 format
  - Both input are connect to TVP5146 decoder are using standard V4L2 interface
  - These module support standard V4L2 device(“/dev/video”)
  - Support buffer access mechanism through memory mapping and user pointers
  - Supports standard V4L2 IOCTLs to get/set various control parameters like brightness, contrast and saturation

Video for Linux Two, V4L2 Driver

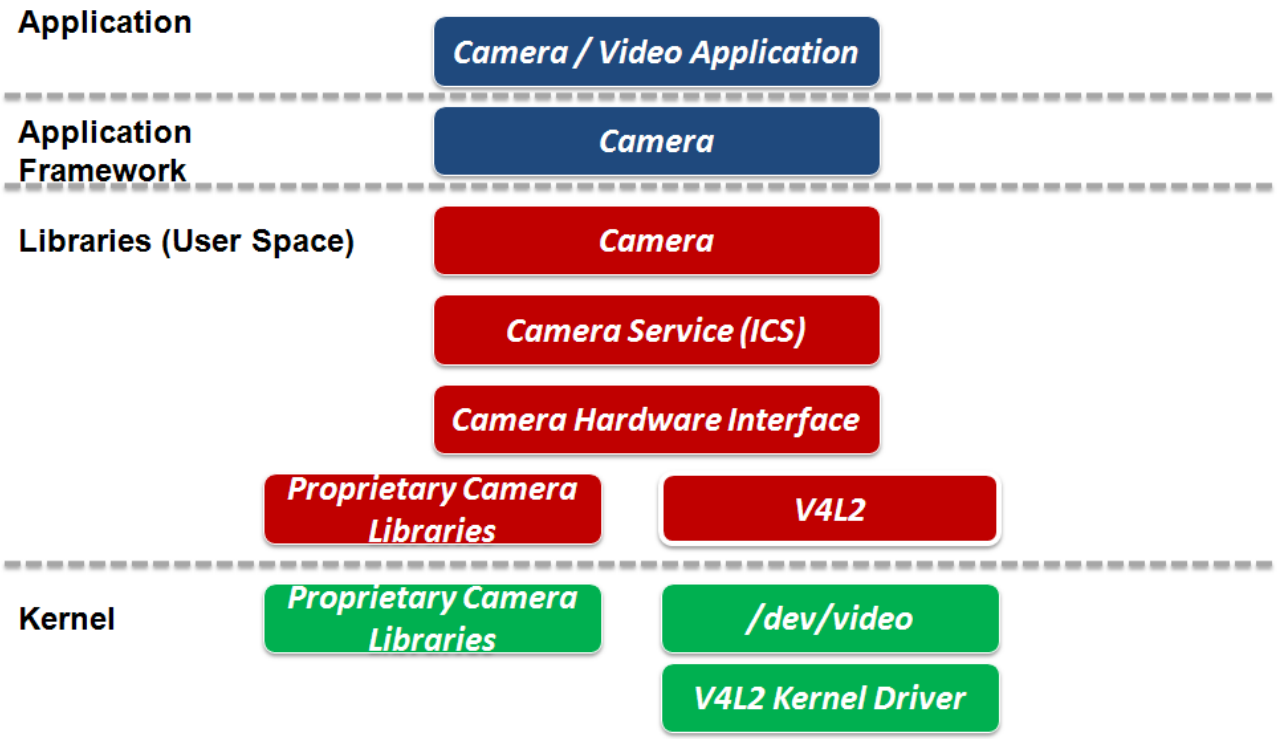
- Video Capture Driver Architecture for omap



OMAP35x ISP-Capture Interface Block-Diagram

Video for Linux Two, V4L2 Driver

- Android Framework Support Camera Interface for video streaming
- The V4L2 API was combined in Android HAL Layer
- Android Camera uses IOCTL to call V4L2 command



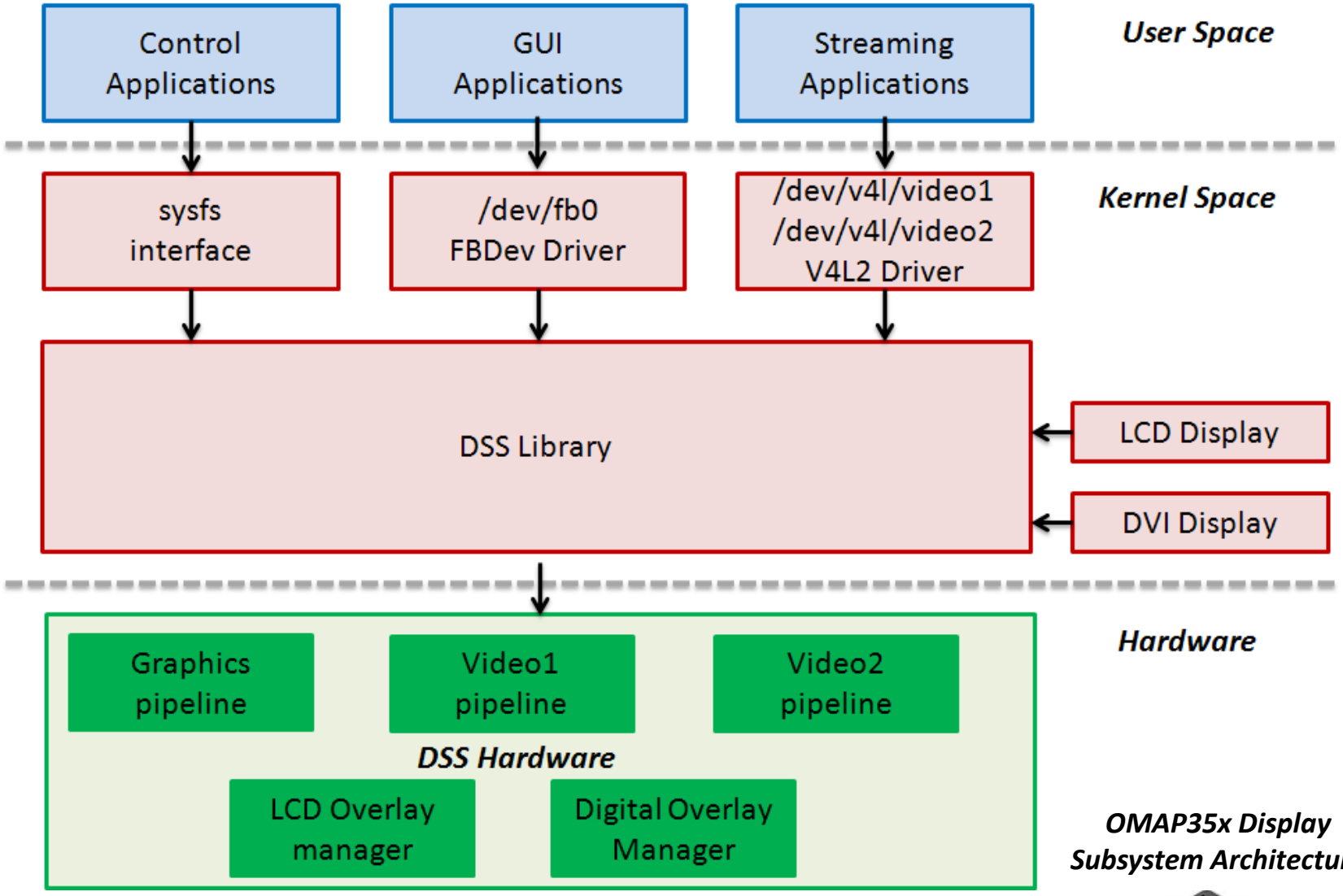
V4L2 for Android Framework

- *Video Device Driver Introduction*
- *Video Codec Format*
- *Raw Data Format*
- *Framebuffer Driver*
- *Video for Linux Two, V4L2 Driver*
- ***OMAP3 Display SubSystem (DSS)***
- *LAB*

OMAP3 Display SubSystem (DSS)

- The DSS , with full name OMAP35x Display Sub System
- The DSS integrated some related hardware , linux drivers and DSS library for flexible use
- The main functionality of display driver is to provide interface to user applications and hardware management
- The OMAP35x DSS Hardware integrated following feature to support multimedia
- One Graphics pipeline
- Two Video pipeline
- One Digital Overlay Manager (DVI、LCD)
- One Analog Overlay manager(TV out)

OMAP3 Display SubSystem (DSS)



OMAP35x Display Subsystem Architecture

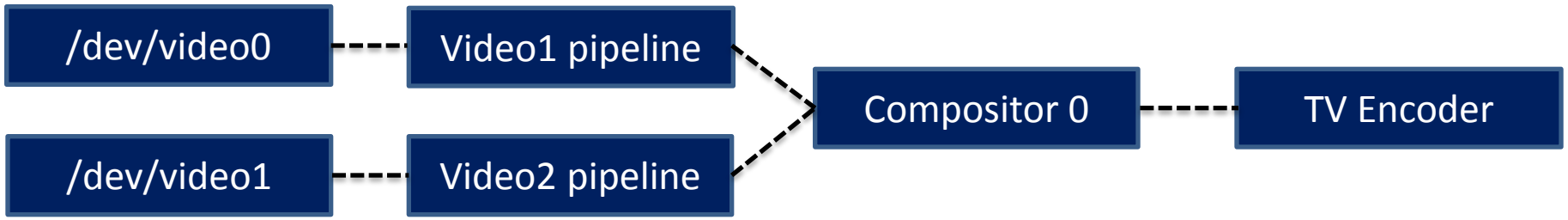
OMAP3 Display SubSystem (DSS)

- DSS support feature
  - Pixel formats supported on video plane are ***YUV***, ***UYVU***, ***RGB565***, ***RGB24P*** and ***RGB24***
  - Video pipelines is controlled through V4L2 user interface
  - Graphic pipeline is controlled through FBDEV user interface
  - Support with LCD display at VGA resolution
  - Support TV display interface at NTSC and PAL resolutions
  - Support alpha blending
  - Supported ***cropping***, ***rotation***, ***mirroring***, ***color conversion***, ***resizing***

OMAP3 Display SubSystem (DSS)

- DSS pipeline architecture

Video Pipeline



Graphics Pipeline

OMAP3 Display SubSystem (DSS)

- OMAP35x DVSDK
  - ***Linux Digital Video Software Development Kits***
  - information about how the ***audio***, ***speech*** and ***video*** clips that come with the DVEVM can be played back
  - We need Linux PSP(Platform Support Package) for TI OMAP35x
    - OMAP35x-PSP-SDK-setuplinux-02.01.03.11.bin
    - Include ***kernel-src***, ***u-boot-src***, ***utility***, ***example***
      - » ***linux-02.01.03.11***

OMAP3 Display SubSystem (DSS)

- OMAP35x DVSDK
  - ***http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/dvsdk/DVSDK_3_00/3_00_02_44/index_FDS.html***

DVSDK_3_00 Product Downloads		
Digital Video Software Development Kit [DVSDK]		
omap3530_3_00_02_44_release_notes	DVSDK Product Release Notes	264K
dvsdk_setuplinux_3_00_02_44.bin	DVSDK Product Installer	172612K
cs1omap3530_setuplinux_1_00_01-44.bin	DVSDK DSP CODECs	24100K
data_dvsdk_3_00_02_44.tar.gz	DVSDK Demonstration Multimedia Files-containing audio and video clips	358360K
data_dvsdk_low_bitrate_3_00_02_44.tar.gz	Low Bit Rate DVSDK Demonstration Multimedia Files-containing audio and video clips	137120K
Linux Platform Support Package [PSP]		
ReleaseNotes-02.01.03.11.pdf	PSP Release Notes	1276K
UserGuide-02.01.03.11.pdf	PSP User Guide	5732K
MigrationGuide-02.01.03.11.pdf	PSP Migration Guide	1164K
DataSheet-02.01.03.11.pdf	PSP Data Sheet	1584K
OMAP35x-PSP-SDK-setuplinux-02.01.03.11.bin	PSP Product Installer	195620K
DVSDK Dependencies		
xdctools_setuplinux_3_15_01_59.bin	XDC/RTSC Tools [Real Time Software Components]	152920K
bios_setuplinux_5_33_06.bin	DSP/BIOS	149620K
TI-C6x-CGT-v6.0.16.1.bin	Code Generation Tools	27952K
Extras 3D Graphics Software Development Kit and WiFi		
OMAP35x_Graphics_SDK_GettingStartedGuide.pdf	Graphics SDK Getting Started Guide	
OMAP35x_Graphics_SDK_setuplinux_3_00_00_09.bin	Graphics SDK Product Installer	286312K
v2.01.03.11-WL6.1.3.1-1.0-Linux-x86-Install	WLAN/Bluetooth driver	
Extras CODEC Add-ons		
c64xplus_mp3dec_1_31_001_production.bin	MP3 DSP CODEC	2132K
Pre-built Images		
overlay_dvsdk_3_00_02_44.tar.gz	Pre-built DVSDK Binaries/Multimedia files - this should be added to an existing PSP file system.	363836K
nfs_dvsdk_3_00_02_44.tar.gz	Complete DVSDK file system including Pre-built DVSDK Binaries [overlay] and self starting DVSDK demos - suitable for NFS mount	374300K
rootfs_dvsdk_3_00_02_44.jffs2	Complete DVSDK file system including Pre-built DVSDK Binaries [overlay] and self starting DVSDK demos - JFFS2 file system format [NAND flash]	67208K
md5sum.list	MD5 Checksums	4K
Older DVSDK 3.00 Releases		
Archived Releases	DVSDK 3.00 Archive Releases	

OMAP3 Display SubSystem (DSS)

- DSS Driver
  - userful reference - ***Documentation/arm/OMAP/DSS***
  - DSS1 - omapfb driver
    - OMAP FB driver in ***drivers/video/omap***
    - The omapfb driver implements arbitrary number of standard linux framebuffers
  - DSS2 - omap-dss driver
    - Add TV-out and multiple display support.
    - in ***arch/arm/plat-omap/dss/***
    - The DSS driver models OMAP's overlays, ***overlay managers*** and ***displays in a flexible way*** to enable non-common multi-display configuration
  - FB panel and controller drivers
    - in drivers/video/omap2
    - The drivers implement panel or controller specific functionality and are not visible to users except through omapfb driver. They register themselves to the DSS driver

OMAP3 Display SubSystem (DSS)

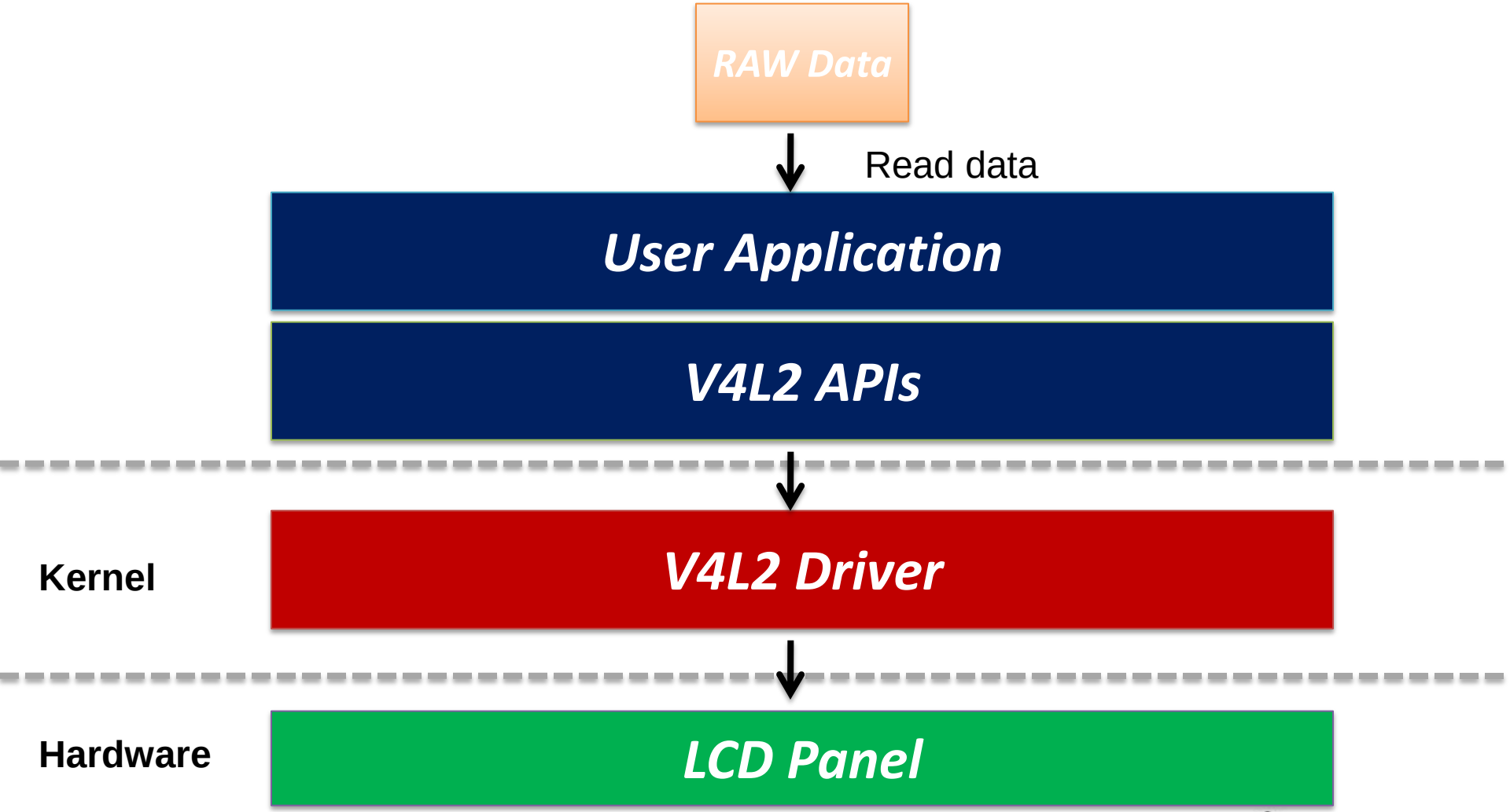
- DSS Driver – V4L2
  - arch/arm/plat-omap/dss/dss.c
    - functional layer for the DSS hardware
  - arch/arm/plat-omap/dss/venc.c
    - functional layer for the video encoder(DAC,RGB888 to NTSC/PAL)
  - drivers/media/video/omap/omap_vout.c
    - core V4L2 driver file, It also implements the necessary ioctls supported by the V4L2
  - drivers/media/video/omap/omap_voutlib.c
    - library file for the V4L2 driver, DSS provides some of the functionalities like cropping, window size and window position, This library file helps in setting all the parameters

- *Video Device Driver Introduction*
- *Video Codec Format*
- *Raw Data Format*
- *Framebuffer Driver*
- *Video for Linux Two, V4L2 Driver*
- *OMAP3 Display SubSystem (DSS*
- ***LAB***

LAB

- Purpose
  - Use V4L2 driver to implement our lab
  - Lab Programming Flow
    - Read Raw data picture via v4l2 driver
    - Use Streaming mechanism streams to local buffer
    - Push Raw data to display interface via v4l2 driver
    - Display the picture on LCD panel

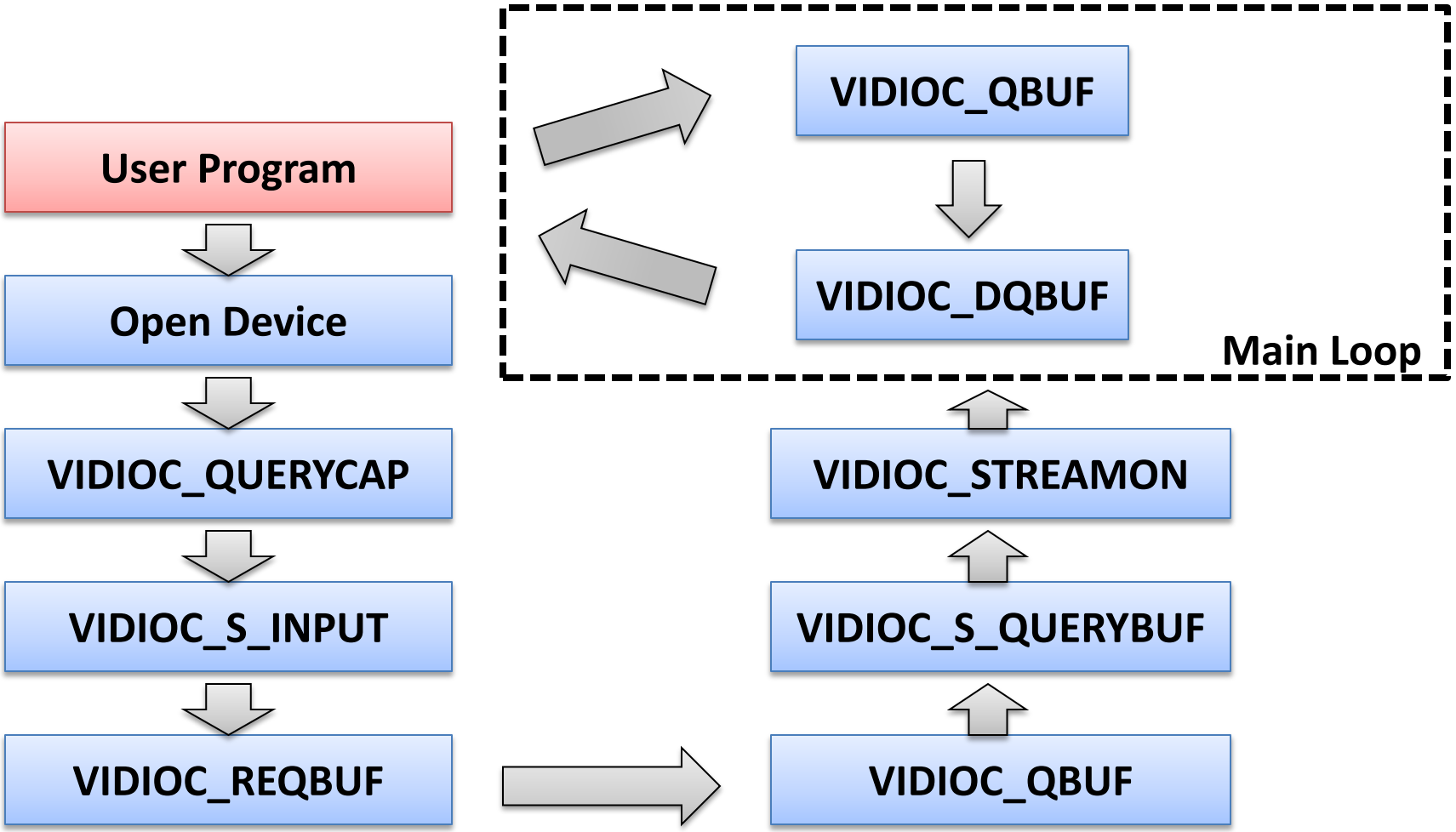
LAB



LAB

- Audio Driver LAB program tip
  - Revise ***vidoe/saVideoARGB.c***
  - Revise ***Rule.make***
    - ***KERNEL_DIR (kernel source path)***
    - ***LIB_DIR (External Library Dir)***
    - ***LIB_INC (External Headers Dir)***
  - Fill the blank with right code
  - ***make***
  - copy ***bin/**** files to target file system
    - **cp bin/* $(NFS_SYSTEM)**
  - Try & debug

V4L2 Processing Flow



LAB

- Programming Stage
  - 1. Open Stage
    - Open the vl42 device
  - 2. Set Parameter Stage
    - Use IOCTL function to set parameter
    - EX : ioctl (fd , VIDIOC_G_FMT , &fmt)
  - 3. Request & Query Buffer Stage
    - Use IOCTL function to set parameter
    - EX : ioctl (fd , REQBUFS , &req)
  - 4. Stream On Stage
    - Start the video stream
    - EX : ioctl (fd , STREAMON , &a)
  - 5. Queue & Dequeue Stage
  - 6. Stream Off Stage

LAB

- V4L2 API - Open device

```
/* Open the video2 device */
fd = open((const char *)dev_name[ch_no], mode);
if (fd <= 0) {
        printf("Cannot open = %s device\n", dev_name[ch_no]);
        exit(0);
}
```

- V4L2 API - Set output buffer type

```
fmt.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
        perror("VIDIOC_G_FMT\n");
        close(fd);
        exit(0);
}
```


LAB

- V4L2 API - Set Image size and pixel format

```
/* Set the image size and pixel format*/
fmt.fmt.pix.width = WIDTH;
fmt.fmt.pix.height = HEIGHT;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB32;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
        perror("VIDIOC_S_FMT\n");
        close(fd);
        exit(0);
}
```


LAB

- V4L2 API - Get Window Parameter

```
/* Get the window parameters before setting
 * and set only required parameters */
fmt.type = V4L2_BUF_TYPE_VIDEO_OVERLAY;
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
if (ret < 0) {
        perror("VIDIOC_G_FMT 2\n");
        close(fd);
        exit(0);
}
```


LAB

- V4L2 API - Set Window Size

```
/* Set the window size */
fmt.fmt.win.w.left =0 ;
fmt.fmt.win.w.top = 0;
fmt.fmt.win.w.width = WIDTH;
fmt.fmt.win.w.height = HEIGHT;
fmt.fmt.win.global_alpha = 255;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
if (ret < 0) {
        perror("VIDIOC_S_FMT 2\n");
        close(fd);
        exit(0);
}
```


LAB

- V4L2 API - Request the buffer

```
/* Request the buffers from the driver */
/* req.memory can be user memory also */
req.count = numbuffers;
req.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
req.memory = V4L2_MEMORY_MMAP;

ret = ioctl(fd, VIDIOC_REQBUFS, &req);
if (ret < 0) {
        perror("cannot allocate memory\n");
        close(fd);
        exit(0);
}
```

LAB

- V4L2 API - Pass the physical address to driver

```
buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
buf.index = i;
ret = ioctl(fd, VIDIOC_QUERYBUF, &buf);
if (ret < 0) {
        perror("VIDIOC_QUERYCAP\n");
        for (j = 0; j < i; j++)
                munmap(buff_info[j].start,
                                buff_info[j].length);
        close(fd);
        exit(0);
}
```


LAB

- V4L2 API - Enqueue the buffers

```
/* 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++)
                        munmap(buff_info[j].start,
                                        buff_info[j].length);
                exit(0);
        }
}
```


LAB

- V4L2 API - Start Stream on

```
/* Start stream on */
ret = ioctl(fd, VIDIOC_STREAMON, &a);
if (ret < 0) {
        perror("VIDIOC_STREAMON\n");
        for (i = 0; i < req.count; i++)
                munmap(buff_info[i].start, buff_info[i].length);
        exit(0);
}
```


LAB

- V4L2 API - Dqueue

```
/* Dqueue the already filled buffers */
ret = ioctl(fd, VIDIOC_DQBUF, &buf);
if(ret < 0){
        perror("VIDIOC_DQBUF\n");
        for (j = 0; j < req.count; j++)
                munmap(buff_info[j].start, buff_info[j].length);
        close(fd);
        exit(0);
}
```

- V4L2 API - Enqueue

```
/* Enqueue the filled buffer */
ret = ioctl(fd, VIDIOC_QBUF, &buf);
if(ret < 0){
        perror("VIDIOC_QBUF\n");
        for (j = 0; j < req.count; j++)
                munmap(buff_info[j].start,
                                buff_info[j].length);
        close(fd);
        exit(0);
}
```

LAB

- V4L2 API - Stream Off the video loop

```
/* Put off streaming
 * This will never execute as steaming is forever loop
 */
ret = ioctl(fd, VIDIOC_STREAMOFF, &a);
if (ret < 0) {
        perror("VIDIOC_STREAMOFF\n");
        for (i = 0; i < req.count; i++)
                munmap(buff_info[i].start, buff_info[i].length);
        close(fd);
        exit(0);
}
```

- V4L2 API - Memory un-mapping the buffer

```
/* Unmap the buffers */
for (i = 0; i < req.count; i++)
        munmap(buff_info[i].start, buff_info[i].length);
close(fd);
return 0;
```

Appendix

- mmap
  - mmap represent memory mapping
  - Mapping the process address to user mode virtual address
  - Let user space application directly access process memory via virtual address
  - Commonly use in store a large number of data instead of read/write

Appendix

```
#include <sys/mman.h>
void *mmap(void *addr, size_t len, int prot, int flag, int filedes, off_t off);
int munmap(void *addr, size_t len);
```

- Prot
  - PROT_EXEC
  - PROT_READ
  - PROT_WRITE
  - PROT_NONE
- Flag
  - MAP_SHARED
  - MAP_PRIVATE

Appendix

- mmap

mmap (add , len ,port ,flags , fd ,offset)