Reprinted from the

Proceedings of the Linux Symposium

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.

Camcorder multimedia framework with Linux and GStreamer

W. H. Lee, E. K. Kim, J. J. Lee , S. H. Kim, S. S. Park
SWL, Samsung Electronics
woonghee.lee@samsung.com

Abstract

Along with recent rapid technical advances, user expectations for multimedia devices have been changed from basic functions to many intelligent features. In order to meet such requirements, the product requires not only a powerful hardware platform, but also a software framework based on appropriate OS, such as Linux, supporting many rich development features.

In this paper, a camcorder framework is introduced that is designed and implemented by making use of open source middleware in Linux. Many potential developers can be referred to this multimedia framework for camcorder and other similar product development. The overall framework architecture as well as communication mechanisms are described in detail. Furthermore, many methods implemented to improve the system performance are addressed as well.

1 Introduction

It has recently become very popular to use the internet to express ourselves to everyone in the world. In addition to blogs, the emerging motion video service provided by such companies as YouTube and Metacafe help us to use internet in this way. The question is, how can we record the video content we want to express?

Digital camcorders, cameras and even mobile phones can be used for making movies. But the quality generated by mobile phones or digital cameras is generally not as good as that of a digital camcorder. If users want to make higher quality content they must use digital camcorders.

In this paper we introduce a camcorder multimedia framework with Linux and GStreamer. We take into account portability and reusability in the design of this framework. To achieve portability and reusability we adopt a layered and modular architecture.

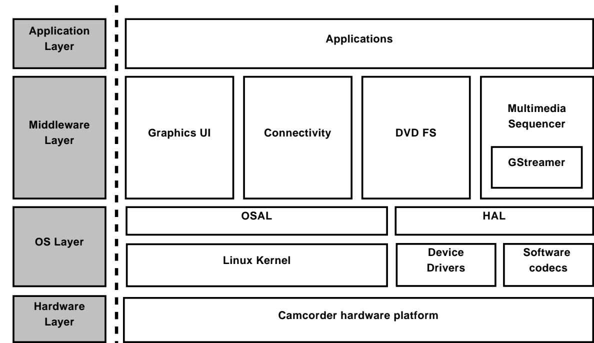

Figure 1: Architecture diagram of camcorder multimedia framework

The three software layers on any hardware platform are application, middleware, and OS. The architecture and functional operation of each layer is discussed. Additionally, some design and implementation issues are addressed from the perspective of system performance.

The overall software architecture of a multimedia framework is described in Section 2. The framework design and its operation are introduced in detail in Section 3. Development environments, implementation, and performance issues are represented and discussed in Section 4. Finally, we present some concluding remoarks in Section 5.

2 Multimedia Framework Overview

The three layers of the multimedia framework are application, middleware and OS. The architecture of the multimedia framework is shown in Figure 1.

2.1 Application layer

There are many programs, such as players and recorders for movie and still pictures, User Interface (UI) managers, USB control, navigator and camera manager in

the application layer. The player and recorder are similar to the general media player and recorder present in standard platforms, however they have additional features in order to support DVD media. The UI manager interacts with camcorders through the keypad, sound and display. USB control manages USB connections. Users can browse DVD titles and select a particular clip with the navigator module. All camera specific functions such as Image Stabilizer and Auto Focusing are implemented inside the camera manager. Because the application layer outside of this paper's scope, we will not discuss it further.

2.2 Middleware layer

The Middleware layer is categorized into four functional groups: multimedia, connectivity, UI, and DVDFS (DVD File System). The multimedia module includes the DVD sequencer, GStreamer and many media specific plugins. USB specific functions are implemented in the connectivity module and are further broken into three major functional blocks: USB Mass Storage (UMS), Digital Print Solution (DPS) and PC Camera (PC-Cam). The UI module includes FLTK and Nano-X. DVDFS is the module that controls the DVD disk file system.

2.3 OS layer

The OS layer plays an important role in system management, OS services, and hardware abstraction. It consists of OSAL (OS Abstraction Layer), HAL (Hardware Abstraction Layer), Linux kernel, device drivers and software codecs. The OSAL provides the middleware an abstraction eliminating the OS dependency. The HAL also provides the middleware with an abstraction eliminating the hardware dependency. The role of the Linux kernel is the management of the system and the support of the general OS environment. The device drivers are used to control the hardware. The multimedia codecs not supported by hardware in the platform are implemented by software.

2.4 Hardware layer

In this paper, the hardware layer represents the camcorder hardware platform including a camcorder specific multimedia SoC chip and its supporting board. The SoC chip supports multimedia oriented operations such as MPEG-2 coding, multiplexing and de-multiplexing of the DVD stream, and IO operations to the DVD disc. In this paper, an application product is assumed to be a camcorder. The supporting board has NOR flash, SDRAM, DVD loader, LCD screen, key pad, camera module and so on.

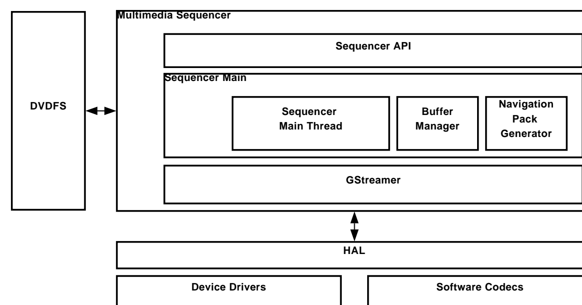

Figure 2: Structure of the multimedia sequencer

3 Architecture design

3.1 Multimedia subsystem

3.1.1 Sequencer

The multimedia sequencer is a middleware module that has functions related to the multimedia control, such as playback and recording.

The architecture of the multimedia sequencer is described in Figure 2.

The multimedia sequencer is configured using interfaces such as the sequencer API layer, sequencer main module, and GStreamer multimedia engine. The GStreamer plugins call the device drivers or the software codecs through the HAL APIs. A developer can create and maintain the plugin codes easily using the HAL APIs. The DVDFS module is used for reading and writing a DVD disc in the sequencer. It is used in both the DVDSrc plugin and the buffer manager block. For playback, the DVDSrc plugin reads the stream data stored in a DVD disc using the DVDFS. For recording, the buffer manager uses the DVDFS for writing the recorded stream data to the DVD disc.

In the sequencer API layer, there are functions related to creation, initialization and control of the sequencer.

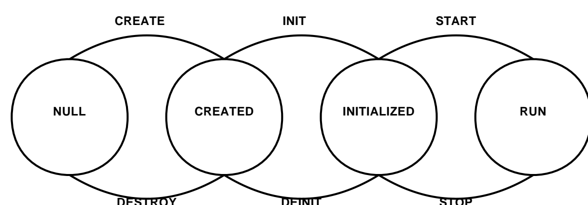

Figure 3: State diagram of multimedia sequencer

The application can create, destroy, initialize, and de-initialize the multimedia sequencer. There are three modes and two types in the multimedia sequencer. The modes are categorized as DVD-Video, DVD+VR and DVD-VR, and the two types are playback and record. For example, if the application initializes the multimedia sequencer using DVD-Video mode and playback type, then the sequencer is set for the DVD-Video player. The sequencer API also supplies control functions. Using these control functions, the application can start, stop, pause and resume the sequencer. And the application can register callback function pointers to the sequencer using the init function. The sequencer can send information to the application at any time using this registered callback function.

In the main layer of the sequencer, the three function blocks are the thread block, buffer manager and navigation pack generator. In the main thread block, the thread routine reads and processes messages in a queue. When the sequencer API is called by the application, the API function is converted to a message. This converted message is sent back to the message queue in the main thread block of the sequencer.

When the sequencer is operating as a DVD recorder, the buffer manager and the navigation pack generator are activated. In the DVD recorder, VOB data has to be recorded on the DVD disc. VOB data contains the presentation data and part of the navigation data. VOB data may be divided into CELL's, which are made up of VOBU's. The video and audio data is packed and recorded as a VOBU unit. The navigation pack is stored at the start of each VOBU data. Because CELL information has to be stored in the navigation packs of VOBU's in the CELL, buffering of CELL data is necessary. The buffering manager controls the VOBU data and calls the navigation pack generator to make navigation pack data of VOBU's in the CELL.

The four states of the multimedia sequencer are NULL, CREATED, INITIALIZED and RUN. The states can change as described in Figure 3. Each state can be managed and set to in the multimedia sequencer based on the state of the GStreamer core.

3.1.2 Interface between GStreamer and sequencer

In this section, we describe the interface between GStreamer and the multimedia sequencer. The GStreamer pipeline configuration for DVD playback and recording is also discussed.

When the multimedia sequencer is initialized, the gst_init() function is called for GStreamer framework initialization.[1] After the GStreamer framework is initialized, shared libraries for plugins are loaded using the gst_plugin_load_file() function. After loading plugin libraries, the sequencer creates a pipeline using gst_pipeline_new() function.

To monitor the state of the GStreamer framework, the sequencer creates an event loop thread that checks messages from the GStreamer framework. In this thread, the sequencer gets a bus from the created pipeline and checks the GstMessage from the GStreamer framework using gst_bus_poll() function. To send the GstMessage to the application, the sequencer uses the registered callback function.

When the application calls the start() function of the sequencer, the loaded GStreamer plugins are registered using gst_element_factory_make() function and then the pipeline is configured. After the pipeline is configured, the properties of plugins are set using g_object_set() function.

To change the GStreamer state, the sequencer calls the gst_element_set_state() function. When the playback or recording operation is started, the state of GStreamer is changed to GST_STATE_PLAYING. To pause the GStreamer framework, the state is changed to GST_STATE_PAUSED.

GObject signals are used to communicate between the sequencer and the plugin. In GStreamer, the GObject signal is already defined for notifying the application of plugin events. In order to register the signal in the sequencer, the g_signal_connect() function is called in the sequencer.[2] After the plugin sends the signal to sequencer, the signal handler function in the sequencer is called. The detailed structure of GStreamer pipelines for

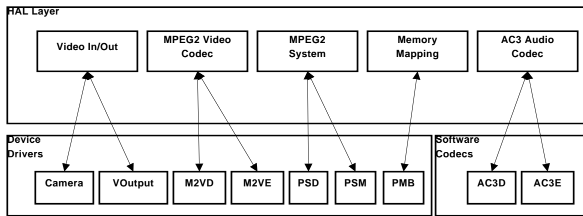

Figure 4: HAL layer with device drivers

playback and record are discussed in sections 3.1.4 and 3.1.5.

3.1.3 Hardware abstraction layer

There are some hardware IP blocks for DVD camcorders in the hardware layer. These hardware IP's can be accessed and controlled by device drivers. These device drivers are wrapped by the HAL layer described in Figure 4. The GStreamer elements related to the device drivers are implemented using the HAL layer API's.

In the HAL layer, there are wrapper functions for video input and output. The device driver functions of camera and VOutput are wrapped in the HAL. The device drivers for MPEG2 video are M2VD and M2VE. These device drivers are wrapped up in the HAL as the MPEG2 video codec. For MPEG2 system layer, there are some hardware IP's such as MPEG2 PS demuxer and muxer. These short names for these device drivers are PSD and PSM. These device drivers are wrapped in the HAL layer as MPEG2 system. Because the hardware IP requires a physical memory buffer for input data, the user space program has to be able to access the physical memory buffer area. For this reason we use memory mapped IO for mapping a physical memory area in device driver into a virtual memory area in user space. An additional benefit of using memory mapped IO is that memory copy operations can be avoided. In device drivers for hardware IP's, the physical memory area can be assigned as the input and output buffer. But, if an element is not connected to such device drivers, the element has to use a device driver that enables the physical memory area mapping. For this reason the PMB device driver is implemented for memory mapping in the HAL layer. The operation of PMB will be described in detail in section 4.2.2. The AC3 audio encode and decoder codecs are needed for a DVD camcorder. These software codecs and named as AC3D and AC3E. The audio codecs are wrapped in the HAL layer as AC3 audio codec.

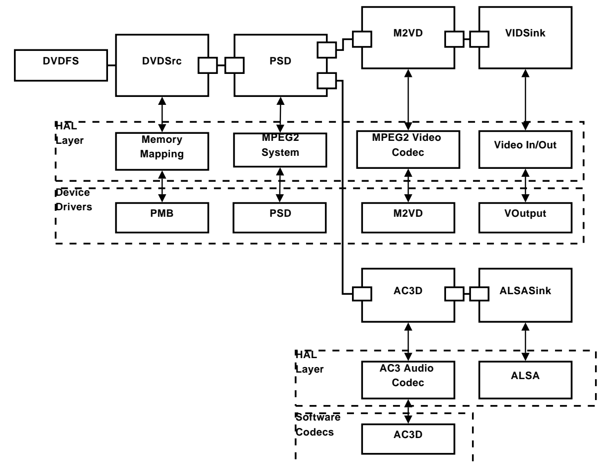

Figure 5: Playback pipeline

3.1.4 Playback pipeline

In this section, the structure and HAL interface of the playback pipeline are discussed. Its diagram is shown in Figure 5.

The DVDSrc element reads the stream data from DVD disc using DVDFS and stores the stream data in the stream buffer. Because the PSD element requires physical memory for the input buffer, the input stream buffer has to be assigned to a physical memory area. This physical buffer is assigned by the PMB device driver through the memory mapping HAL layer and is mapped into the virtual memory address in the DVDSrc element. The DVDSrc element stores the stream data into the physical memory buffer using the mapped virtual memory address. The base addresses of physical and virtual memory and the memory size are sent to the PSD element using GstCaps. The input stream buffer memory is configured and managed as a ring buffer in the DVDSrc element. The address of the input data is sent to the PSD element through the GstBuffer.

The PSD element parses the input stream data and stores the parsed data into the video and audio stream buffers that assigned by the PSD device driver through the MPEG2 system HAL layer. This parsing operation is processed using the PSD device driver through the MPEG2 system HAL layer. The stream buffer for the parsed data is located at a physical memory area and the

address of this stream buffer is sent to the M2VD element through the GstBuffer.

The M2VD element is used for MPEG2 video decoding. In the M2VD element, the M2VD device driver is connected through the HAL layer of the MPEG2 video codec. The address of the video stream buffer in the physical memory area is sent from PSD element. The M2VD element decodes the input video stream data and stores the decoded frame data into the physical memory buffer. This physical memory buffer is assigned by the M2VD device driver through the HAL layer of MPEG2 video codec. After decoding the video stream into the reconstructed frame, the M2VD element sends the physical buffer address of output buffer to the VIDSink element through GstBuffer. Then, the reconstructed frame can be displayed on either LCD or TV.

For audio, the stream data is sent from the PSD to the AC3D element. Because the AC3 audio decoder is a software codec, it can access the input buffer by virtual memory address. The PSD sends the memory mapped virtual address to the AC3D element using GstBuffer. The AC3D element decodes the AC3 audio stream data to PCM data by AC3D software codec through the HAL layer of AC3 Audio codec. The decoded PCM data is sent to the ALSASink element. This ALSASink element is provided from the GStreamer base plugin.

Audio and video synchronization is controlled using the system clock provided from GStreamer. AV synchronization is implemented based on the time stamp data embedded in the stream data. The time stamp data, so called PTS (Presentation Time Stamp), is extracted from the navigation pack in VOBU data and converted to the GStreamer time format. All audio packs have the PTS, so the audio decoder can send the PTS to next element without additional calculation. In the case of video pack, however, the PTS only exists in the first video pack of GOP. Hence, the time information of the video packs has to be calculated using the video frame rate. In GStreamer, the calculated time information is embedded in GstBuffer structure and passed through the GStreamer pipeline. And then the audio and video synchronization can be achieved automatically in the GStreamer core.

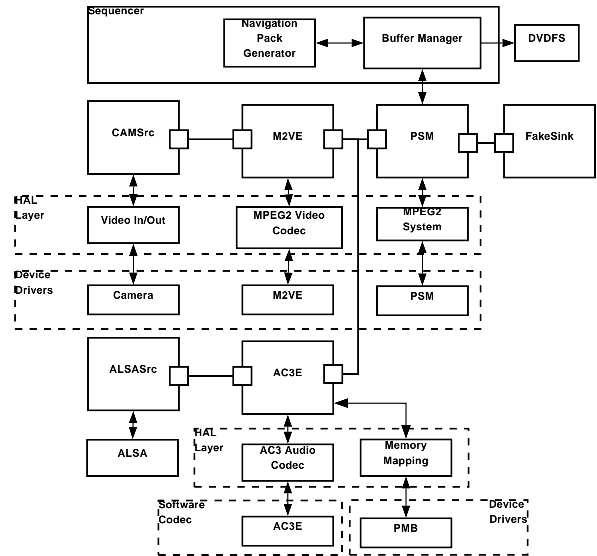

Figure 6: Record pipeline

3.1.5 Record pipeline

In this section, the structure and HAL interface of the record pipeline are discussed. Its diagram is shown in Figure 6.

The CAMSrc element is connected to the camera device driver through the video in/out HAL layer. The input frame data is stored in the physical address area that is assigned by camera device driver. The physical address of the input frame data is sent to the M2VE element.

The M2VE element encodes the input frame data and sends the physical address of the encoded video stream data to the PSM element.

The ALSASrc element sends the captured PCM data to the AC3E element. In the AC3E element, the PCM data is encoded by the AC3E software codec through the HAL layer of AC3 audio codec. Because the PSM device driver requires the physical memory address of the input audio stream buffer, the AC3E element uses the PMB device driver for memory mapping. The encoded audio data is stored in the virtual memory address by the AC3E element and the memory mapped physical address is sent to the next PSM element.

The PSM element packetizes the input video and audio stream data, and generates video and audio packs. In order to satisfy the MPEG2 system layer and the DVD standard, the PSM element uses the buffer manager in the multimedia sequencer. For synchronization

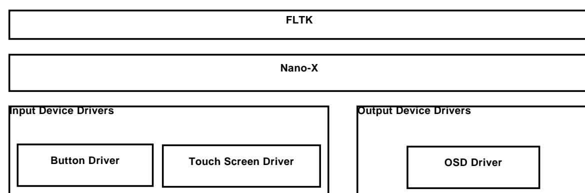

Figure 7: Structure of graphics subsystem

of recorded stream, the PSM element uses the GstCollectPads that is provided from GStreamer.

In PSM, the VOBU information from the packetized data is sent to the buffer manager. The buffer manager generates the navigation pack for the VOBU using the navigation pack generator. After CELL data is ready in the buffer manager, the CELL data is recorded to the DVD disc using DVDFS module. Because the disc write operation is processed in the buffer manger and DVDFS module, we use the FakeSink as a dummy element for the generation of a complete pipeline

3.2 Graphics subsystem

The structure of the graphics subsystem is depicted in Figure 7. The FLTK (Fast Light Tool Kit)[3] is a lightweight version of GTK+, and Nano-X[4] is a lightweight X window system. They are generally used for the graphics subsystem of embedded platforms due to their small size. They are supported by input and output device drivers. The input device can be a keyboard and a pointer like touchpad, and the output device can be on screen display (OSD) or the frame buffer.

3.2.1 Input and output device drivers

To support input devices, the button and the touch screen devices are used in the camcorder platform. The button driver takes care of the press, release and long press events from each key or button. The touch screen driver handles of click, double click and drag events.

The target platform has video output and OSD layers. The video output layer is used for displaying video frame data from camera module, and the OSD is used for displaying camcorder information like icons, numbers and menus. In the graphics subsystem, only the OSD layer is of interest. The OSD implementation is similar to that of the Linux frame buffer, however, it needs an additional process. It should be set the palette of the specified OSD and its property as followings:

- Width and height
- Bit per pixel
- Screen color number
- Pixel format
- Frame buffer address
- Scale
- Chroma-key information

Additionally, the video output and OSD have different color formats, YUV and RGB respectively. Because the color format to be displayed on LCD or TV is YUV, the RGB data from OSD must be converted to YUV.

3.2.2 Nano-X

Nano-X is a lightweight X windows system developed by the Nano-X open source project. Although it implements a lightweight X window system, it offers lots of functionality so that it is sufficient for embedded devices such as digital cameras and camcorders.

Nano-X is implemented based on MicroGUI, which is a portable graphics engine and composed of Win32-like Microwindows API and X-like Nano-X API.

Nano-X generally supports keyboards and pointers such as mice as input devices, however buttons and a touch screen are used for the input devices of camcorders. The drivers of button and touch screen devices should work like those of keyboard and pointer devices.

3.2.3 FLTK

FLTK is a lightweight version of GTK+. FLTK takes an important role in creating and exploiting widgets, handling various events in the windows and widgets, and supporting OpenGL window to implement OpenGL's applications on the FLTK. Its role is similar to GTK+'s.

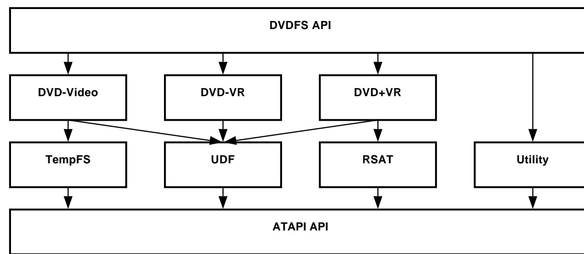

Figure 8: Architecture of DVDFS

FLTK is implemented in C++ so that we can create various types of windows and widgets, and handle the special events by sub-classing the widget and window class. FLTK supports FLUID (UI Designer Toolkit) to help UI designers create an UI applications easily.

Because all applications in our framework are implemented in C, application programmers can not create or extend the widgets and windows implemented by C++. So the parts of FLTK implemented by C++ should be wrapped in API's for the mixed C/C++ development.

3.3 DVD file system

Nowadays there are various types of DVD-Disc, such as DVD-ROM, DVD-R, DVD-RW, DVD-RAM, DVD+R and DVD+RW. Each of them has different physical characteristics. Moreover, the formats for storing data on DVD-Disc can be categorized into three different specifications: DVD-Video[5], DVD-VR and DVD+VR. The camcorder software has to consider all of the types of media and DVD formats. DVDFS provides simple APIs for DVD recording and playback. The application and middleware can record and play the DVD contents without worrying about what types of DVD media and formats are used.

The architecture of DVDFS is shown in Figure 8.

Modules of DVDFS can be described as follows:

- **DVDFS API**-The unified user API set.
- **DVD-Video**-Module for DVD-Video format
- **DVD-VR**-Module for DVD-VR format
- **DVD+VR**-Module for DVD+VR format
- **Tempfs**-Temporary File System. It manages a temporary table which is used in the finalization on DVD-Video format.

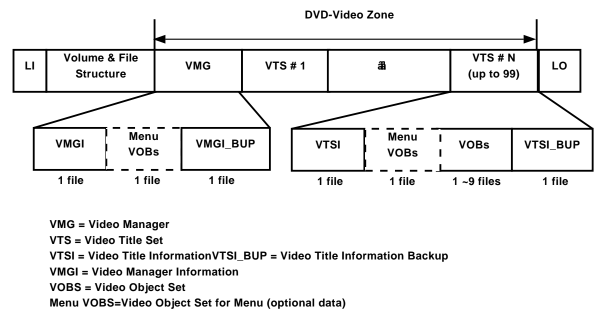

Figure 9: Layout of DVD-Video format

- **UDF**-Module for Universal Disc Format[6]
- **RSAT**-Module for Reserved Sector Allocation Table. It is used by DVD+VR module to translate the mapping information of first reserved area.
- **Utility**-Module for formatting, getting disc info.
- **ATAPI API**-Wrapper of the OS specific ATAPI.

DVD-Video, unlike other formats, does not support realtime recoding, so it needs to support Tempfs. We will focus on how to record through DVD-Video format in following sections.

3.3.1 DVD-Video format

DVD-Video format is a basic specification and most widely used for the distribution of movies. In the beginning it only considered pre-pressed disc like DVD-ROM, so the order of files is physically predetermined as showin in Figure 9. For that reason it is not suitable for real-time recording devices such as camcorders, recorders and so on.

3.3.2 Real-time recording on DVD-R

DVD-R is a write-once media. Basically, it has to be written by sequential-recording methods. This means that recording is only permitted at the next address of the written block. Therefore, DVD-R provides the incremental write mode. In this mode, the media can be divided into several areas and these areas are called RZone. Sequential-recording is performed in each RZone. DVDFS adopts this method to reserved areas

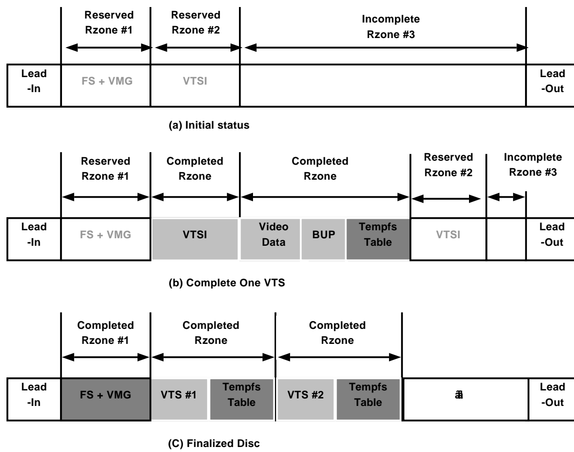

Figure 10: Real-time recording DVD-Video with DVD-R

for file system data and video information files which are determined after recording video data.

The management of RZone is the most important process in recording data on DVD-R. Disc status during the recording operation on DVD-Video and its sequence are described in Figure 10.

(a) depicts the initial status of disc. RZone #1 is reserved for file system and VMG. RZone #2 is reserved for VTSI. The remained area is automatically regarded as RZone #3, and it is used for recording movie data.

(b) depicts the status of the completed VTS. When user closes a movie data recorded, VTSI_BUP is generated and written continuously. Then, Tempfs table is written at the end of VTSI_BUP. It contains all data of file location, file size and time information. Once RZone is closed, it can not be managed any more. Then, DVDFS need to reserve another RZone for new VTSI.

(c) depicts the final status. Firstly, the volume and file structures of UDF are written to RZone #1. At that time, the last Tempfs table is used for constructing the UDF data structures. Then, VMG data is written on the end of UDF. Finally, the border will be closed.

3.3.3 Real-time recording on DVD-RW

In case of the rewritable media, formatting creates addressable blocks which can be overwritten. It takes a long time to format a whole disc. To avoid this problem, quick format is used. Formatting is performed in

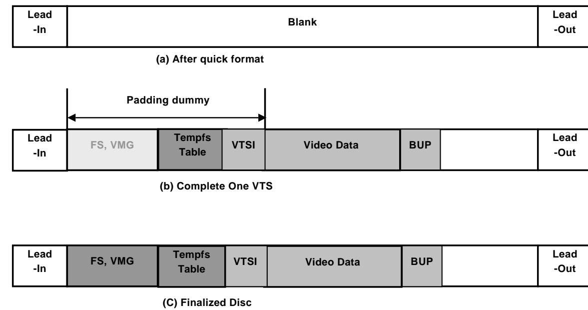

Figure 11: Real-time recording DVD-Video with DVD-RW

the small part of disc, and other parts are only used for sequential recording. Once a block is written, it turns into an over-writable block.

Figure 11 illustrates the sequence of recording operations on DVD-RW. After quick format, DVDFS reserves a space by writing dummy data. This area will contain file system, VMG, VTSI, and Tempfs table. Video data is written after that padded data.

Another difference between recording of DVD-R and that of DVD-RW is the location of Tempfs table. Because the over-writing is possible in DVD-RW, the Tempfs table is written on a fixed area

3.4 Connectivity subsystem

3.4.1 Architecture of USB Connectivity

The Linux kernel supports USB devices using the USB gadget framework.[7] It is made up of the Peripheral controller drivers, Gadget drivers and Upper layers. The Peripheral controller drivers manage and control the USB hardware IP. The Gadget drivers are the logical layers divided by their function. They can be a file system (Gadget file system), a network (Ethernet over USB), a serial or a MIDI. Upper layers are the supporting layers for user applications. They execute the specific functions such as UMS, DPS, Ethernet or serial.

The USB connectivity architecture is depicted in Figure 12. USB controller driver is the Peripheral controller driver in the USB gadget framework. It has to fit well with the API of the USB gadget framework. This makes it easier to implement or to use Gadget drivers. Although there are many kinds of Gadget drivers, only

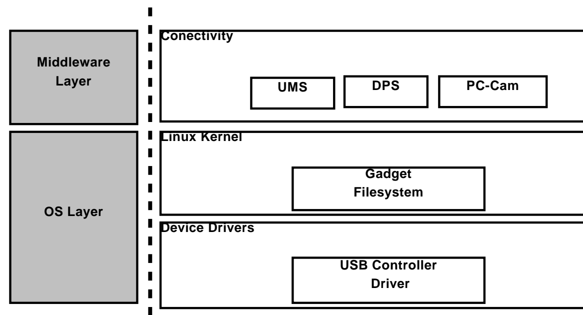

Figure 12: Architecture of USB connectivity

the Gadget file system is used in our camcorder framework. In the middleware layer, the connectivity module supplies UMS, DPS and PC-Cam.

3.4.2 Blocks of USB connectivity

UMS transfers files to computer. This requires some implementations such as:

- **USB Mass Storage Class Bulk-Only Transport** - It sends commands through CBW(Command Block Wrapper) and gets the result of CBW by receiving CSW(Command Status Wrapper).[8]
- **SCSI protocol** - It is a protocol included in CBW. It has commands like READ and WRITE.[9]
- **Block device control function** - It enables UMS to get block device information such as media type and number of sectors from block device. And it must be able to read and write from/to the block device.

DPS enables the camcorder to connect to a printer directly and to print images. The connected printer must have the function of PictBridge. This needs some implementations such as:

- **Picture Transfer Protocol**-It is a protocol for digital cameras to send images to other devices like PC and printers.[10]
- **PictBridge**-It is an industry standard written by CIPA (Camera & Image Products Association). It includes the methods for the device discovery and sending the information of images.[11]

PC-Cam enables a camcorder to send the captured images and sounds. Data transfer is divided into two parts - video and audio. Video data is compressed by JPEG and transferred to PC. The methods for transferring audio data observe the definition of USB Audio Class.[12]

3.5 OS

3.5.1 OS abstraction layer

The original purpose of OSAL was to remove the OS dependency from software. If middleware and application have such a dependency on the OS, they can't be reused on different OS's. However, it causes some overhead to cover up the difference between many OS's. Therefore, we designed the OSAL to achieve its original purpose and to reduce overhead simultaneously. To meet this goal, the number of categories and functions of OSAL are limited to the minimum as far as possible. The categories consist of task, semaphore, message queue, mutex, timer and system timer. The functions of OSAL are limited to creating, deleting and a few action functions.

3.5.2 Porting Linux kernel

In order to support the camcorder platform, the Linux kernel needs to be ported with board specific code and device drivers. The first work is to select the appropriate ARM core code from many versions of codes already existing in the Linux kernel source. Because our platform has ARM 11 core, the ARMv6 code of Linux kernel source was selected. The second is to include machine specific code. This code initializes and controls the peripherals of the SoC Chip such as Timer, Clock, DMA, IO mapping and so on. Finally, device drivers not existing in the standard kernel source need to be created. They support many kinds of general and camcorder specific devices. They are almost based on open source and managed by HAL layer and GStreamer elements in middleware.

4 Implementation

4.1 Development environments with emulator

The emulator is a useful tool for development and testing of applications on a PC. It is helpful when a real target is not available. It provides an emulated camcorder

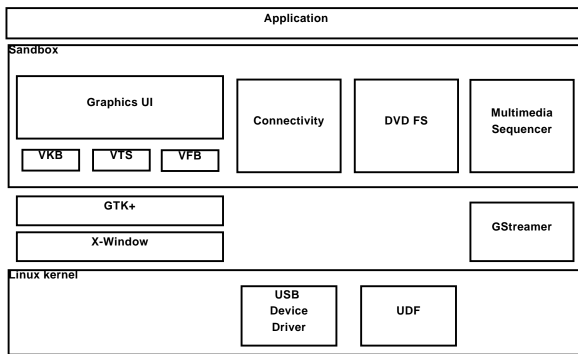

Figure 13: Architecure of emulator

framework. The user interface of the emulator includes keyboards, buttons, menu toolbar, touch screen and so on. It is based on X Window, GTK+ and sandbox modified from scratchbox[13] . The architecture of emulator is shown in Figure 13.

In general, the emulator supports virtual devices for UI interface. The camcorder emulator supports virtual keyboard (VKB), virtual touch screen (VTS) and virtual frame buffer (VFB). Each device works as input and output devices respectively. The emulator can display the camcorder application on the PC using the X window through the VFB device and emulate its actions through VKB and VTS.

The other parts of camcorder framework are also emulated by substituting target platform dependent part. We install USB-device PCI card to the PC for emulation of USB connectivity. In the case of GStreamer emulation, the software codecs are used instead of the hardware codec.

4.2 Performance enhancement methods

4.2.1 Thread based element in GStreamer

When input data is not sufficient to be processed, the input data needs to be buffered more. Especially in the case of DVDs, because the input stream data is divided into packs, the M2VD element has to wait the input data until the input stream data is sufficient to be decoded and reconstructed into a frame. For this buffering operation, the M2VD element is threaded. The structure of the M2VD element based on the thread is similar to that of the queue element of GStreamer. In the thread based element, the size of queue is the main control point of the element. When the chain function of the thread based element is called, the element just increases the size of queue by the input data size and returns immediately. The main operation on the input data starts immediately when the size of queue is sufficient for processing. Because the memory mapped addresses are shared between the elements, this buffer size based control is possible. The immediate return of the thread based element enables the previous element to do another operation. We apply the structure of thread based elements to both M2VD and AC3D elements. These elements are linked with the source pads of PSD element. Because of the immediate response/return of the M2VD and AC3D to the PSD element we can get better decoding performance

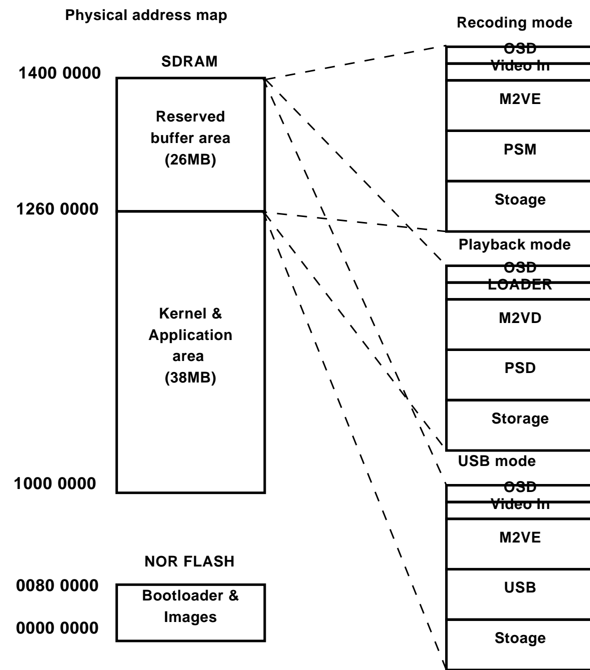

Figure 14: Reserved buffer area

4.2.2 Reserved buffer area

The memory allocation routine spends CPU time to use memory efficiently. The dynamic memory allocation, e.g., memory allocation and free in Linux kernel, are very complicated and computationally intensive. In order to eliminate these problems, the reserved buffer area is used for some device drivers.

Figure 14 shows the reserved buffer area in our camcorder framework. It's divided into small parts for the device driver according to execution modes such as

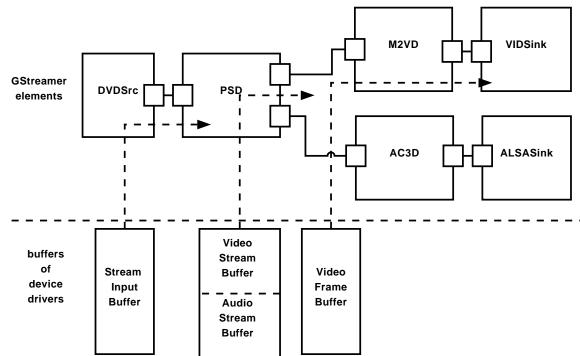

Figure 15: Memory interfaces of elements

recording, playback and USB mode. Each device driver buffer can be accessed by not only the device driver itself, but also user applications or middleware.

4.2.3 Using memory mapped IO

Normally, read and write functions are used for exchanging data between user application and kernel device drivers, but this needs to be reconsidered from a performance point of view. Firstly, they use system calls has a long call path. Secondly, the read and write data are copied by two functions, copy_from_user() and copy_to_user(), in the device driver. In order to avoid copy operations, the mmap is used instead of the read and write operations. The mmap permits us to directly access the buffer in the device driver. The camcorder framework benefits from this mmap operation a lot because it transfers a huge data frequently.

Figure 15 shows the memory interface for DVD playback. In the previous section, buffers are allocated in the reserved buffer area and accessed by their own device drivers. However, GStreamer elements can not access those buffers directly. Therefore, GStreamer elements get the access permission and virtual address by calling mmap functions of device drivers. Each element controls or manages its buffer through the virtual address.

4.2.4 Fast boot

In a PC environment, the Linux system spends more than one minute on the booting process. The booting process includes following steps; loading image, initializing the kernel and device drivers, and starting some daemons. However, such a long boot time is not suitable for commercial products. Fast boot techniques for Linux have been studied for a long time.[14] We investigated and adapted them to our camcorder framework.

- **Fast memory copy** - Generally, the memory copy routine is implemented by ARM instructions such as LDR and STR. The transfer unit is 4 bytes long in size with those instructions. However, LDM and STM can transfer data in a larger unit longer than 4 bytes. By using these instructions, the fast memory copy can be achieved.

- **Using the uncompressed kernel image** - The zImage is the compressed kernel image. It spends quite a long time to uncompress the image. If the uncompressed image is used, the uncompressing time can be removed.

- **Reducing kernel option** - There are many options in the Linux kernel source. Only minimum options required for booting the system up are turned on to save time.

- **Preset LPJ** - In the kernel initialization, there is a function so called LPJ to calibrate the CPU speed. It is used for busy waiting like udelay() and mdelay(). The Preset LPJ means that the calibrating code is disabled and its output variable for the delay is set with the pre-estimated value.

- **Invalidating printk** - There are a lot of messages generated in the booting phase, and consequently they affect the booting time. To solve this we can either use the 'quiet' parameter in the kernel, or turn off the CONFIG_PRINTK option. The former increases the verbose level so that messages are not printed out. And the latter invalidates the printk function completely. In other words, the prink function is converted to a null function. We invalidate the printk function using the latter.

- **Using prelink** - An application is generally built with shared libraries. Using shared libraries saves the development time and the image size as well. However, it spends a lot of time binding the library symbols at run-time. The prelink technique removes the bind operation, so that the start-up time of the application can be reduced.

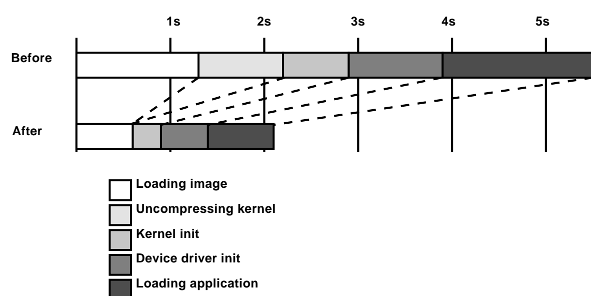

Figure 16: Result of fast boot

To analyze the fast booting process, the booting completion point needs to be defined first. Usually, it correspond to a state of record standby. In this paper, however, the completion point is defined as just before the application start-up because the application development is not completed for integration in the software framework at this time. Figure 16 shows measured data after adopting the fast boot techniques mentioned above. The booting time was about 5.5 seconds before adoption of fast boot techniques. After adoption, it reduced to about 2.1 seconds. Assuming a real product development, it will presumably take an extra time due to added components in the application.

4.2.5 Memory foot print

Because GStreamer depends on many external libraries, the total library size of the GStreamer framework is not suitable for the embedded system. In order to use the GStreamer in an embedded system, the size optimization is a significant part of development. In order to reduce GStreamer size, the external libraries actually used in the operation of GStreamer should be identified, assorted, and built with the appropriate options accordingly.

The external libraries that have dependencies with the GStreamer core are libxml, libz, libglib, libcheck and libpopt. Of the above, all external libraries except the libglib can be removed. Because the libglib is extensively used in the entire GStreamer core, it needs to be kept.

The result of optimization is described in the following Table 1. As a result of optimization, the total size of GStreamer was reduced from 2.9MB to 1.4MB approximately.

Library name	Before	After
libcheck.so	27,216	0
libpopt.so	26,928	0
libxml2.so	1,110,004	0
libz.so	74,140	0
libglib-2.0.so	622,420	622,420
libgmodule-2.0.so	10,308	10,308
libgobject-2.0.so	243,020	243,020
libgthread-2.0.so	14,684	14,684
libgstreamer-0.10.so	644,160	445,896
libgstbase-0.10.so	147,516	79,280
libgstinterfaces-0.10.so	32,412	32,392
total	2,952,808	1,448,000

Table 1: Foot print of GStreamer optimization

5 Conclusions

In order to support many multimedia devices such as cameras and camcorders using a single hardware platform equipped with a dedicated multimedia SoC, a flexible and extensible software framework is highly preferred from the development cost perspective.

GStreamer is one of the open source based multimedia frameworks. Especially, it is adopted widely in multimedia product because of a variety of its plugin functions. In this paper, we presented a multimedia framework which supports both DVD and memory camcorders. It was designed and implemented by making use of open source middleware such as GStreamer in Linux OS. Based on the GStreamer engine, we developed a multimedia framework with many plug-ins implemented by hardware as well as software codec. The emulator was also introduced as a development environment in the PC platform.

References

[1] *GStreamer Application Development Manual*, `http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual`

[2] *GStreamer Plugin Writer's Guide*, `http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/`

[3] *Fast Light Tool Kit Open Source Project*, `http://www.fltk.org/`

[4] *The Nano-X Windows System Open Source Project*,`http://www.microwindows.org/`

[5] *DVD Specification for Read-Only Disc Part3: Version 1.1*

[6] *Universal Disc Format Specification Revision 2.50*, http://www.osta.org/specs/

[7] *Linux-USB Gadget API Framework*, http://www.linux-usb.org/gadget/

[8] *USB Mass Storage Class Bulk-Only Transport*, http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf

[9] *Information Technology - SCSI Block Commands 2–3*

[10] *PIMA 15740:2000 - Picture Transfer Protocol for Digital Still Photography Devices*, http://www.i3a.org/

[11] *White Paper of CIPA DC-001-2003 Digital Photo Solutions for Imaging Devices*

[12] *Universal Serial Bus Device Class Definition for Audio Devices*

[13] *Scratchbox*, http://www.scratchbox.org/documentation/general/tutorials/explained.html

[14] *Boot Time Resources*, http://tree.ceLinuxforum.org/pubwiki/moin.cgi/BootupTimeResources