Introduction to Multimedia Support

Introduction

In the last three years, Renesas has been working hard to apply it's Open Source community participation standards that have been successfully adopted in the Linux Kernel space to User Space applications. In particular, Renesas has supported the development of a multimedia stack for the SH-MobileR series of application processors. Renesas has supported this development directly through adoption and enhancement of several free software components such as Video for Linux 2 (v4l2) http://en.wikipedia.org/wiki/V4L2e, Framebuffer Device (fbdev) http://en.wikipedia.org/wiki/Linux_framebuffer, Userspace IO (UIO) http://www.kernel.org/doc/htmldocs/uio-howto.html, OpenMAX http://en.wikipedia.org/wiki/OpenMAX, as well as the necessary kernel level code to support Renesas IP hardware components. IP components include integrated DSP and dedicated hardware for video encoding and decoding.

SH-Mobile Platform

The SH-Mobile (SuperH Mobile Application Processor) is designed to offload application processing from the LSI in order to improve performance and ease system development. SH-Mobile are 32-bit RISC application processors comprised of a system LSI and multiple function blocks.

Some sample SH-Mobile Processors and those that the development was done on are:

SH7722: SH-4 SH-MobileR	http://am.renesas.com/products/mpumcu/superh/sh7780/sh7722/sh7722_root.jsp
SH7723: SH-4A SH-MobileR2	http://am.renesas.com/products/mpumcu/superh/sh7780/sh7723/sh7723_root.jsp
SH7724: SH-4A SH-MobileR2R	http://am.renesas.com/products/mpumcu/superh/sh7780/sh7724/sh7724_root.jsp

Renesas Hardware IP Blocks

The development of the multimedia stack takes advantage of several hardware IP blocks incorporated onchip by Renesas. These are:

- VPU: Video Processing Unit
  - MPEG-4 and H.264 accelerators
  - Encoding and decoding (half-duplex)
  - H.264 slice processing
- VIO: Video Input/Output
  - CEU: Camera Engine Unit
    - YCbCr 4:2:2 or 4:2:0
    - horizontal/vertical sync
  - VEU: Video Engine Unit
    - image processing in memory
    - colorspace conversion YCbCr -> RGB -> YCbCr
    - dithering (in RGB color subtraction)
    - filtering: mirror, point symmetry, 90 degree, deblocking, median
  - BEU: Blend Engine Unit
    - image blending in memory
    - picture-in-picture
    - graphic combining
- JPU: JPEG
- LCDC: LCD controller
- VOU: Video Ouput
- SIU: Sound Input/Output
- USB

Linux Kernel Interfaces

To access the latter hardware IP blocks existing Linux kernel interfaces are used as mentioned earlier: v4l2, fbdev, and UIO.

Video for Linux 2 (v4l2)

v4l2 is the standard Linux kernel interface for video capture. It contains specific interfaces for selecting capture dimensions and color format. Applications can use read() to read frame data, or alternatively can use an ioctl() to stream frame data directly into memory buffers. Camera parameters like brightness and contrast can also be changed while the device is open.

Framebuffer Device (fbdev)

fbdev (Framebuffer device) is a standard Linux kernel interface for handling video output. It specifies routines for negotiating screen dimensions, the color map and pixel format, and for using any available acceleration functions.

Userspace IO (UIO)

UIO (Userspace IO) is a fairly recent Linux kernel mechanism for allowing device driver logic to be implemented in user space. The kernel level UIO interface itself is generic and does not have functionality specific to any kind of device. It is currently only available for char drivers (not block or network). Using UIO, the implementation of a device driver is split into two parts:

- a small kernel module which provides memory maps, and stubs for interrupt handling and IO
- a userspace device driver which uses normal read(), mmap() system calls to work with the device.

UIO: Kernel Module

On the kernel side, the module provides a struct specifying:

- the IRQ number, flags and a handler() function which acknowledges the interrupt
- an array of memory areas which can be mapped into userspace
- mmap(), open(), release() functions which are called from the corresponding file_operations members

UIO: User Space

The user space portion is a normal process which opens eg. /dev/uio0. This returns a pollable file descriptor from which the number of available events can be read. Normal system calls can be used to interact with this device:

- Can use poll() to wait until the file descriptor is ready to perform I/O.
- calling read() on this file descriptor returns the number of events (interrupts) since the last read. Both blocking and non-blocking reads are possible.
- mmap() should be used to map the memory areas described by the kernel space module.

OpenMAX

OpenMAX is a set of cross-platform APIs for multimedia codec and application portability specified by the Khronos Group. It consists of three layers:

- OpenMAX Integration Layer: A standardized media component interface to integrate and communicate with multimedia codecs implemented in hardware or software. It does not provide any interfaces for synchronized capture or playback of video and audio.
- OpenMAX Development Layer: A set of APIs that specify audio, video and imaging functions that can be implemented and optimized on new CPUs, hardware engines, and DSPs and then used for a wide range of accelerated codec functionality such as MPEG-4, H.264, MP3, AAC and JPEG.
- OpenMAX Application Layer: OpenMAX AL (Application Layer) provides acceleration of capture and presentation of audio, video, and images.

SH-Mobile Multimedia Libraries

- libshcodecs: Callback-based encoding and decoding. Some example apps contained in the source tree:
  - shcodecs-dec
  - shcodecs-enc
  - shcodecs-cap, shcodecs-capenc
- omxil-sh: Bellagio OpenMAX IL components for:
  - MP3, AAC decoding using SH7722 DSP
  - MPEG4, H.264 decoding (and encoding) using the VPU
- libuiomux: provides a kernel interfaces for various IP blocks: VPU, VEU, (2DG, BEU, URAM, MRAM, JPG). We need to provide a simple way for applications to use any of these individually, and to allow multiple applications to access different functions of a single IP block. For example, encoding and decoding of video via the VPU is half-duplex, and we want to be able to separate these tasks. Thus we provide a resource allocation layer which manages these functions, on top of the UIO user space device driver. It handles map management and event dispatch, but does not have any specific code for the devices.
  - multiplexes access to named resources, including devices which are available via UIO. It can also be queried for whether or not a resource is available on the currently running system. UIOMux consists of a user-level shared library, libuiomux, which manages a shared memory segment containing mutexes for each managed resource. This segment and its mutexes are shared amongst all processes and threads on the system, to provide system-wide locking. In this way, libuiomux can be used to manage contention across multiple simultaneous processes and threads.
  - UIOMux allows simultaneous locking of access to multiple resources, with deterministic locking and unlocking order to avoid circular waiting. Processes or threads requiring simultaneous access to more than one resource should lock and unlock them simultaneously via libuiomux. UIOMux will save and restore of memory-mapped IO registers associated with a UIO device. Registers are saved on uiomux unlock() and restored on uiomux lock(), if intervening users have used the device.
  - Map Management: The entire range mapped by the UIO device is implicitly available to all its users, so they must co-operatively share it and ensure not to overwrite each other's regions. The kernel is unable to provide more finely grained protection as these regions are not fixed in size at the time of UIO initialization; for example, the size of an image buffer depends on the requested dimensions.