A Linux multimedia platform for SH-Mobile processors

Embedded Linux Conference 2009

Conrad Parker

April 7, 2009

Abstract

A Linux multimedia platform for SH-Mobile processors

Over the past year I've been working with the Japanese semiconductor manufacturer Renesas, developing a Linux multimedia stack for the SH-MobileR series of application processors. These processors have an integrated DSP and dedicated hardware for video encoding and decoding, as well as various other nice features. In this presentation I'll introduce various free software components for building multimedia gadgets on this platform. I'll also discuss architecture-independent details about sharing UIO devices and using OpenMAX IL components.

Outline of topics

- SH-Mobile platform
- Linux kernel interfaces
- Intro to OpenMAX
- libshcodecs, omxil-sh
- Resource management, UIOMux

Demo session tomorrow evening at 6:30 PM, Imperial B

SH-Mobile processors

SH-Mobile are 32-bit RISC application processors, comprising a system LSI with multiple function blocks.

- SH7722: SH-4 up to 266MHz
- SH7723: SH-4A up to 400MHz

`http://www.renesas.com/`

IP Blocks

- VPU: Video Processing Unit
- VIO: CEU, VEU, BEU
- JPU: JPEG
- LCDC: LCD controller
- VOU: Video Output
- SIU: Sound I/O
- USB

VPU

Video Processing Unit

- MPEG-4 and H.264 accelerators
- Encoding and decoding (half-duplex)
- H.264 slice processing

CEU

Camera interface:

- YCbCr 4:2:2 or 4:2:0
- horiz/vertical sync

VEU

Video Engine Unit: image processing in memory

- colorspace conversion YCbCr → RGB → YCbCr
- dithering (in RGB color subtraction)
- filtering: mirror, point symmetry, 90degree, deblocking, median

BEU

Blend Engine Unit: image blending in memory

- picture-in-picture
- graphic combining

DSP

SH7722 also has SH4AL-DSP extended functions

- max 4 parallel operations: ALU, mult, 2xload/store
- conditional execution
- zero-overhead repeat loop control

Linux kernel interfaces

We use existing Linux kernel interfaces to provide access to various IP blocks:

- v4l2: Video for Linux 2
- fbdev: Framebuffer device
- UIO: User space IO

v4l2

v4l2 (Video for Linux 2) 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.

fbdev

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

UIO: overview

UIO (Userspace IO) is a fairly recent Linux kernel mechanism for allowing device driver logic to be implemented in userspace. 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.

UIO: enable interrupts

```
/* Enable interrupt in UIO driver */
{
        unsigned long enable = 1;

        write(uio_dev.fd, &enable, sizeof(u_long));
}
```

UIO: wait for interrupt

```
/* Wait for an interrupt */
{
        unsigned long n_pending;

        read(uio_dev.fd, &n_pending, sizeof(u_long));
}
```

OpenMAX

OpenMAX

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

- OpenMAX Integration Layer
- OpenMAX Development Layer
- OpenMAX Application Layer

OpenMAX is specified by the Khronos Group.

OpenMAX IL

OpenMAX IL (Integration Layer) is 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 DL

OpenMAX DL (Development Layer) APIs 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 AL

OpenMAX AL (Application Layer) provides acceleration of capture and presentation of audio, video, and images.

- This API is provisional: it was expected to be finalized at the end of 2008.

Implementations

A shared library with the IL core and a "reference" OpenMAX component, and some OpenMAX components which pass Khronos conformance tests.

- Bellagio is an open source implementation of OpenMAX IL, developed by STMicroelectronics and Nokia.
- TI have an OpenMAX implementation for OMAP.
- OpenCore is the multimedia framework used by the Android platform. It was developed by PacketVideo. It includes an open source implementation of OpenMAX IL.

GStreamer

GStreamer is a framework which allows applications to build an arbitrary graph of demux, decode, effects, encode, mux and I/O elements.
The scope of GStreamer is roughly equivalent to OpenMAX IL and AL combined.
There is an OpenMAX-GStreamer project which implements GStreamer plugin wrappers for Bellagio OpenMAX IL components.

Tunneling

In a real application you want to use various IP blocks together, like displaying the output of the VPU onto the LCD. OpenMAX and GStreamer are abstract interfaces which generally make buffers available for software processing. However if you happen to connect two plugins together for hardware IP blocks, then you want them to share buffers and just pass pointers around and synchronize buffer access.

GStreamer negotiation

GStreamer has great facilities for discovering plugin capabilities and negotiating data formats for exchange between plugins. GStreamer-OpenMAX can use this negotiation to trigger OpenMAX tunneling between components.

SH-Mobile multimedia libraries

- libshcodecs
- omxil-sh
- libuiomux

These libraries are available under the GNU LGPL

libshcodecs

Callback-based encoding and decoding. Example apps:

- shcodecs-dec
- shcodecs-enc
- shcodecs-cap, shcodecs-capenc

libshcodecs: Decode

To decode video data, an application provides a callback with the following prototype:

```
int vpu4_decoded (SHCodecs_Decoder * decoder,
                  unsigned char * y_buf, int y_size,
                  unsigned char * c_buf, int c_size,
                  void * user_data);
```

libshcodecs: Decode

and then supplies encoded video data to the function:

```
shcodecs_decode (decoder, input_buffer, length);
```

The decoder will process the input buffer, and call the provided callback function each time a frame is decoded. The output is given in two bitplanes of YUV 4:2:0.

libshcodecs: Decode

There are two modes of operation:

- Frame-by-frame: data passed to shcodecs_decode() corresponds to an entire encoded frame
- Streaming: pass in arbitrary amounts of an MPEG elementary stream.

When all streaming data has been passed in, call shcodecs_decoder_finalize() to flush any constructed frames remaining in the decoder.

libshcodecs: Encode

To encode video data, an application provides both input and output callback functions:

```
/* callback for acquiring an image */
int get_input(SHCodecs_Encoder * encoder, void *user_data);

/* callback for writing encoded data */
int write_output(SHCodecs_Encoder * encoder,
        unsigned char *data, int length, void *user_data);
```

libshcodecs: Encode

and then run the encoder by:

```
shcodecs_encoder_run(encoder);
```

The encoder will retrieve raw video as needed using the input callback, and will use the output callback you have provided whenever encoded video data is available.

MPlayer

Magnus Damm developed VEU support for VIDIX, MPlayer's interface for fast user-space video output drivers. This is now in upstream MPlayer.
We also have patches for:

- Video decoding using libshcodecs
- Audio deocding using AAC, MP3 DSP routines

omxil-sh

Bellagio OpenMAX IL components for:

- MP3, AAC decoding using SH7722 DSP
- MPEG4, H.264 decoding (and encoding) using the VPU

GStreamer plugins

GStreamer plugins for VPU encoding and decoding using libshcodecs are currently under development.

Resource allocation

We provide 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.

UIOMux

UIOMux 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 locking

UIOMux allows simultaneous locking of access to multiple resources, with deterministic locking and unlocking order to avoid circular waiting. Processes or threads requring 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.

Future work

Conclusion

The SH-Mobile platform has various IP blocks which are useful for multimedia processing. Linux kernel interfaces for these are in the upstream kernel, and we are developing a set of libraries and multimedia components for using them.
Demo session tomorrow evening at 6:30 PM, Imperial B

- Conrad Parker `conrad@metadecks.org`