Video4Linux2: Path to a Standardized Video Codec API

Kamil Dębski
Linux Platform Group
Samsung Poland R&D Centre

Contents

- Hardware accelerated video codecs
  - Special needs and quirks
- Current solutions for HW codecs
- Video4Linux2 memory-to-memory devices
- V4L2 Codec API
- MFC 5.1 and its quirks
- Using a V4L2 codec API from user space
  - Decoding use case
  - Encoding use case
- How to get started with a video codec driver
- Summary

Hardware accelerated video codecs

- **Hardware module integrated in the SoC or as a separate chip**

- **Characteristics**
  - Memory intensive operations
  - High bandwidth required for HD content
  - Highly optimised for speed
    - Sometimes at the cost of complicated interface
  - Preference for low power usage

Current solutions

- **OpenMAX based**
  - Usually proprietary kernel interface
  - Vendors often provided closed source OpenMAX libraries
- **Open source OpenMAX IL libraries**
  - Bellagio
  - LIM OpenMAX
  - Vendors prefer to use own implementations
- **Problems**
  - Even if kernel code is open source the library can be closed source
  - Quite complicated, different implementations
  - Library, not a kernel interface

OpenMAX

- **OpenMAX Bellagio with support for Exynos4 using the Video4Linux2 Codec API**
  - http://omxil.git.sourceforge.net/git/gitweb.cgi?p=omxil/components;a=shortlog;h=refs/heads/samsung
  - http://omxil.git.sourceforge.net/git/gitweb.cgi?p=omxil/omxil;a=shortlog;h=refs/heads/samsung

Current solutions

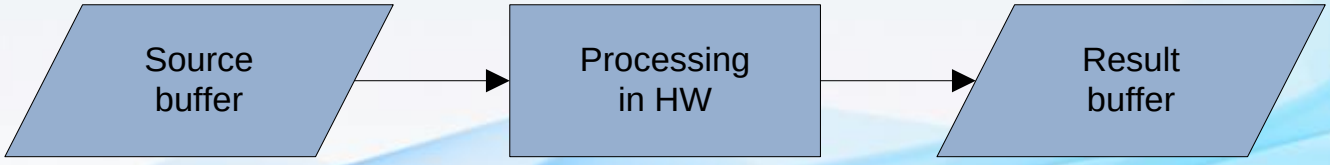
- **Distributed Codec Engine by TI**
  - Uses the syslink/rcm shim layer interface to connect the application to the codec
  - It is a library, not a kernel interface
- **VA API by Intel**
  - Library with many backends
- **VDPAU by NVidia**
  - Library
  - Proprietary driver

Video4Linux2

- **Provides support for TV tuners, webcams, video capture and output devices**
- **Support for video codecs has been added by Hans Verkuil**
- **In 2.6.35 support for memory-to-memory (m2m) devices has been added**
  - Enabled support for video filters
    - Example devices FIMC, G2D
  - Made possible to add video codecs acting as filters
  - Prior to the introduction of the m2m framework the Video4Linux only supported hardware codecs where only either source or sink was memory
- **After the introduction of the m2m devices the codec API has been extended**

Memory-to-memory devices

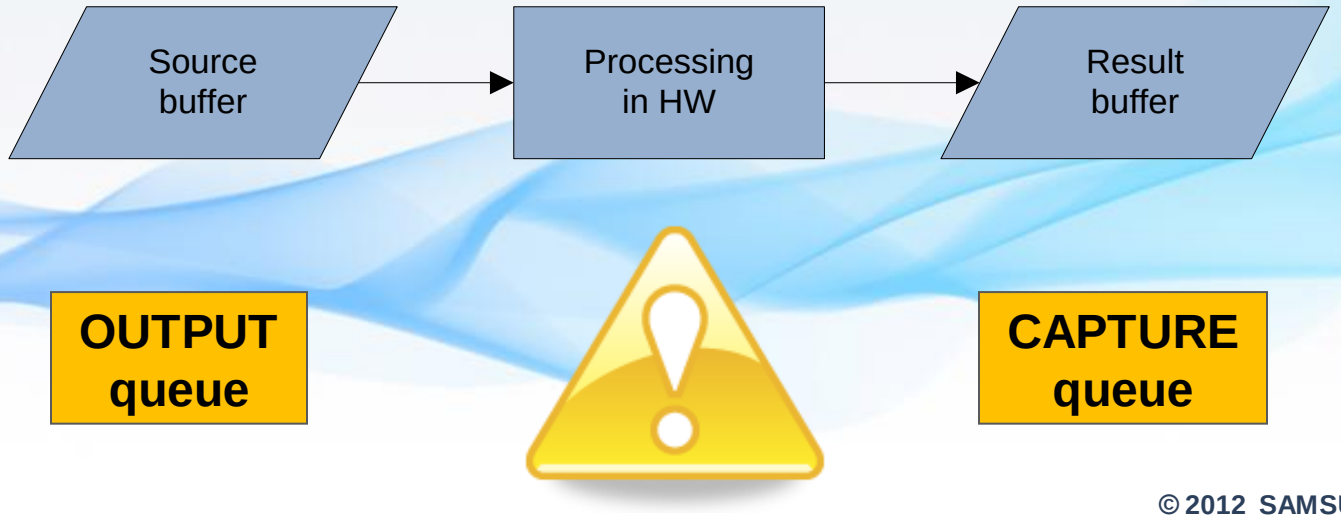
- **Have properties of both CAPTURE and OUTPUT nodes**
- **Both the source of the data and result of the processing is memory**
- **Simple devices can use the m2m framework**
  - Here simple means that one source buffer produces one result buffer (an 1:1 relationship)



- **For more complex devices such as the MFC video codec the m2m framework cannot be used**

Memory-to-memory devices

- **Have properties of both CAPTURE and OUTPUT nodes**
- **Both the source of the data and result of the processing is memory**
- **Simple devices can use the m2m framework**
  - Here simple means that one source buffer produces one result buffer (an 1:1 relationship)



Multiplanar API

- **New special pixel formats that indicate use of multiple planes**
- **Each buffer can contain many planes e.g. :**
  - Separate plane for each colour component
  - Extra data associated with the buffer (e.g. face detection results, highlight warning)
- **MFC was one of the inspirations that led to the introduction of multiplanar API**
  - Y and CbCr components had to be placed in two different part of memory
  - Additional padding constrains (NV12MT)

Video4Linux2 Codec API

- **Memort-to-memory support**
- **Multiplanar API**
- **Videobuf2 subsystem handles many details of buffer management**
- **The API extension**
  - Added support for more codecs – H.264, H.263, MPEG4
  - Add new controls to modify encoding parameters

Multi Format Codec v5.1

- **Available in Exynos3[1] / Exynos4[2]**
- **Capable of decoding and encoding 1080p content of up to 30 fps**
- **Support for H.264, H.263, MPEG4 enc/dec and MPEG1/2, VC1, XVID decoding**
- **Hardware quirks:**
  - Two AXI bus masters
  - Buffer address restrictions
  - Complicated colour format - NV12MT
  - Clock gating – separate clocks for each memory bank IOMMU

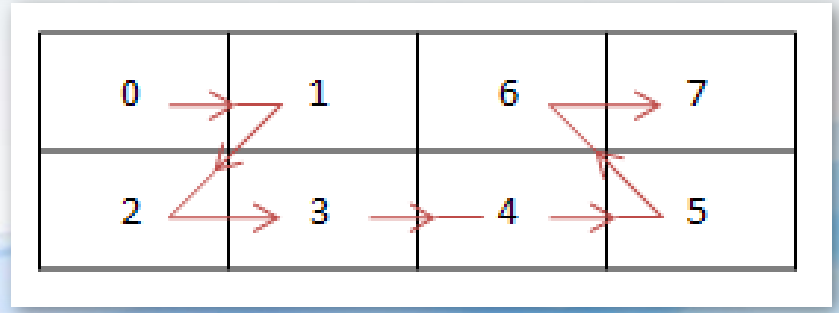
[1] aka S5PC110/S5PV210 [2] aka S5PC210/S5PV310

Memory Management in MFC

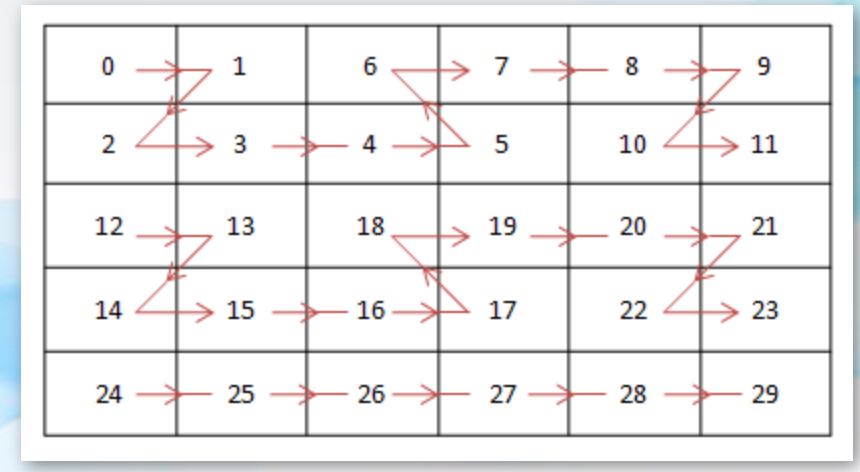
- **Need to allocate contiguous memory areas**
- **Necessary to define various memory banks**
  - MFC bank 1 – firmware, stream buffers, chroma buffers
  - MFC bank 2 – luma buffers
- **Troublesome without IOMMU**
  - with IOMMU one still needs to care about memory banks
- **Solution CMA**
  - Provides regular kernel API – dma_alloc_coherent
  - Mainlined in kernel 3.5
  - Enables definition of memory banks
  - Memory reserved for hardware module can be reused by user space applications when module is idle

NV12MT Colour Format

- YUV 4:2:0
- Luma and chroma separated in two buffers
- Chroma U and V components interleaved
- Image divided into 64x32 tiles
- Specific order of tiles in memory „Z” shaped



Basic „Z”shape



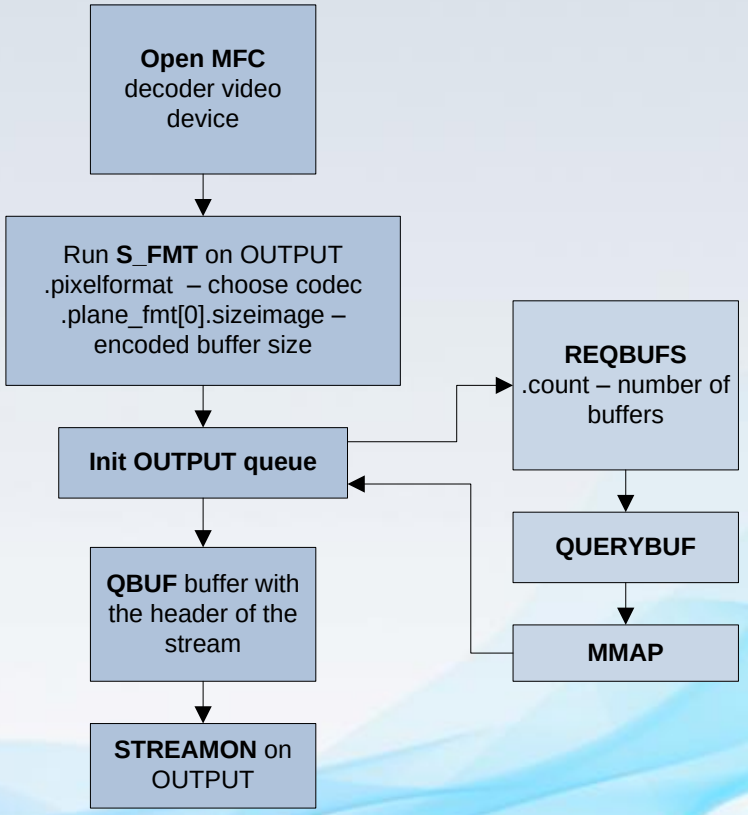
Layout example

Decoding use case

- Setting up the device and parsing header
- Buffer initialization
- Decoding frames
- Finishing decoding
- Resolution change
- Stream seek

Decoding – setup and parsing header

- **Setup**
  - Setup OUTPUT queue
    - Set format – choose codec
    - Initialize and allocate buffers
  - Queue header
  - Start streaming on OUTPUT

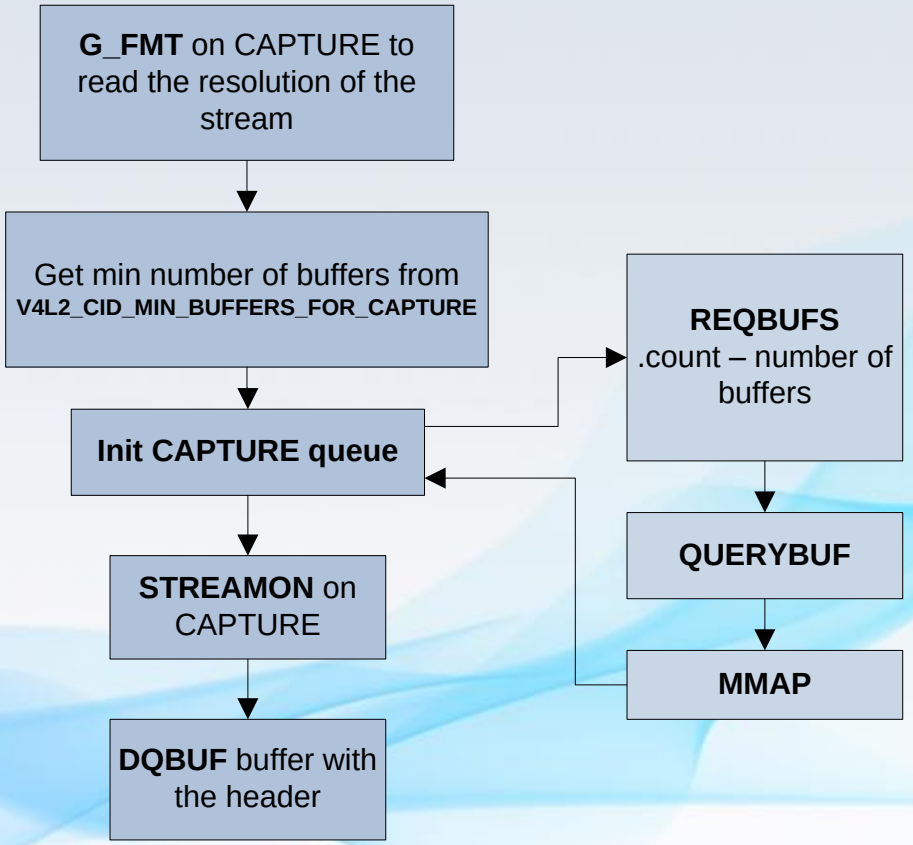


Parsing header is necessary to determine the properties of the encoded video stream. These are the frame size and the number of buffers necessary for decoding.

Decoding – buffer initialization

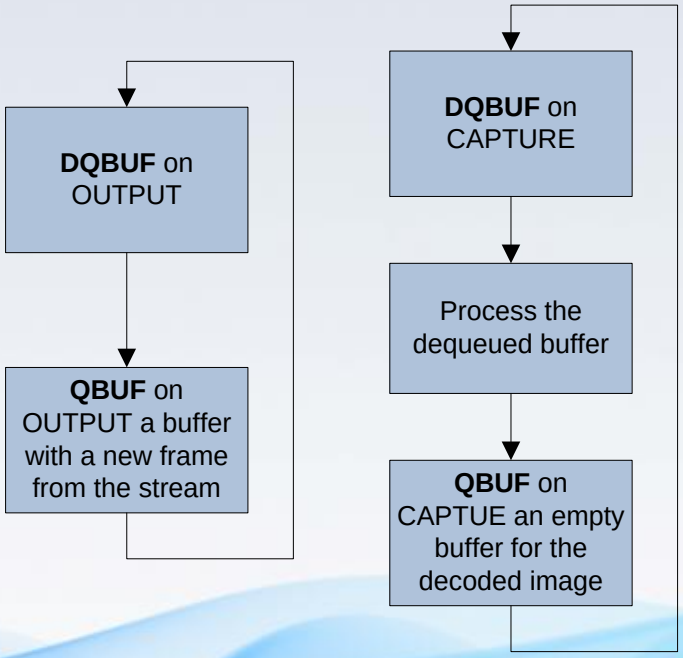
Buffer initialization

- CAPTURE queue can only be setup after the header is parsed
- Read video stream properties
  - Size
  - Minimum number of buffers
- Start streaming on CAPTURE after the buffers are mmaped and queued



Decoding – decoding frames

- Example threads



Decoding – decoding frames

Decoding frames

- Queuing OUTPUT buffers with encoded stream
- Dequeuing CAPTURE buffers with decoded frames
- Timestamp data (timestamp, timecode) is copied to a CAPTURE buffer from the corresponding OUTPUT buffer
- Frame type (I, P, B) determined by flags
- Sequence number set by the driver
- In case of a non critical error the V4L2_BUF_FLAG_ERROR flag is set, in case of a critical error DQBUF will fail

Decoding – decoding frames

Decoding frames

- One OUTPUT buffer is not equivalent to one CAPTURE buffer
  - Most of the time it is, but the are exceptions
  - In case of packed PB stream format, one packed PB OUTPUT buffer is decoded to two CAPTURE buffers
  - When using slice interface – one CAPTURE buffer can be generated from multiple OUTPUT buffers

Decoding – decoding frames

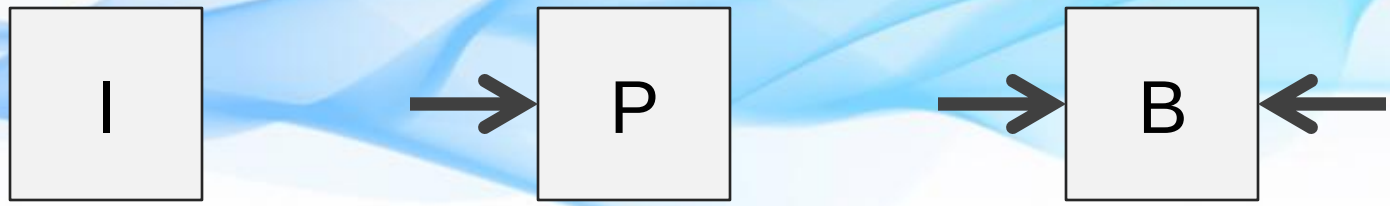
Decoding frames

- Delay before CAPTURE buffers are returned
  - A number of buffers has to be processed before first decoded buffer can be returned
    - Exception – if the user application won't write to the decoded frame buffer and order of framesm does not matter

      (for example when generating thumbnails)
  - This is a property of the stream

Encoded pictures

Slices of a picture



Picture types

Display and decoding order

- Display order

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	P	B	B	I

- Decoding/Encoding order

1	4	2	3	7	5	6	10	8	9	13	11	12
I	P	B	B	P	B	B	P	B	B	I	B	B

- Delay (keeping frames for reference)

1	4	2	3	7	5	6	10	8	9	13	11	12	X
I	P	B	B	P	B	B	P	B	B	I	B	B	-

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	P	B	B	I

Decoding – finishing decoding

Finishing decoding

- After all encoded frames have been queued on the OUTPUT queue it is necessary to notify the driver that decoding is finished.
- To do this one has to queue an OUTPUT buffer with *bytesused = 0*
- This will release the buffers that were kept as reference frames so they can be dequed from the CAPTURE queue
- After the last last frame from CAPTURE is dequeued a buffer with *bytesused = 0* will be returned

Decoding – resolution change

Resolution change

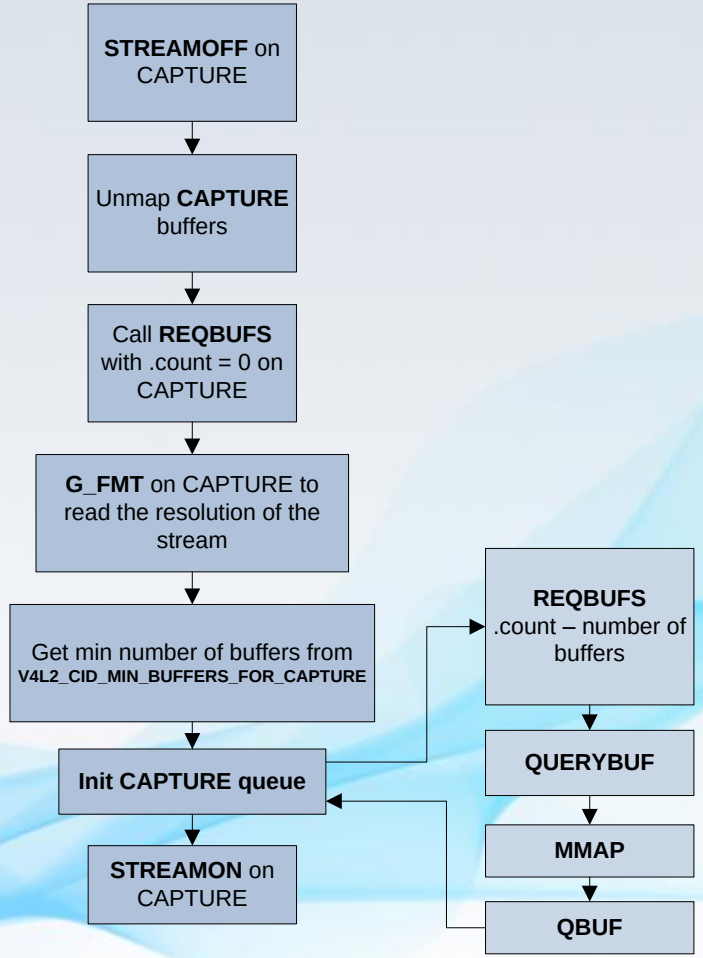
- Tricky as the resolution can be increased or decreased. The number of necessary buffers can also change.

- Indicated when a buffer with *bytesused = 0* is dequeued from CAPTURE queue.

Decoding – resolution change

Resolution change

- Following steps to be done:
  - Stop streaming on CAPTURE
  - Unmap all CAPTURE buffers
  - Run Reqbufs with *count = 0* to free memory
  - Read new stream properties – with G_FMT and V4L2_CID_MIN_BUFFERS_FOR_CAPTURE
  - Run Reqbufs with appropriate buffer count
  - Mmap new buffers
  - Queue new buffers
  - Start streaming on CAPTURE



Decoding – stream seek

Stream seek – method 1

- Necessary to flush the buffers
- Stop streaming on OUTPUT and CAPTURE
- Queue new encoded stream buffers on OUTPUT
- Queue necessary CAPTURE buffers
- Start streaming on CAPTURE and OUTPUT

Decoding – stream seek

Stream seek – method 2

- Use timestamp / timecode to match OUTPUT and CAPTURE buffers
- Start queueing OUTPUT buffers from the new point in the stream
- Check timestamp / timecode value to match CAPTURE buffers with the new place in the stream
- Check decoded frame type to find first I-frame

Encoding use case

- Setup encoding
- Setup buffers
- Encoding frames
- Finishing encoding

Encoding - setup

- Set image size and pixel format with S_FMT on OUTPUT
- Choose codec and set size of the buffer used for output of the encoder by calling S_FMT on CAPTURE
- Adjust encoding parameters by setting appropriate controls

Encoding – buffers setup

- Run reqbufs on both CAPTURE and OUTPUT queues
- Queue CAPTURE buffers
- Queue OUTPUT buffer with first frame
- Start streaming (VIDIOC_STREAMON)

Encoding frames

- Video frames to be encoded are queued on the OUTPUT queue
- Encoded stream chunks are dequeued from the CAPTURE queue
- If no B-frames are used one buffer on OUTPUT queue is enough
- If B-frames are used then more than one buffer has to be queued on OUTPUT queue at a time. This because encoding order is different from display order

Encoding - finishing

- To finish encoding the application has to send a special command - V4L2_ENC_CMD_STOP
- This command is sent after the last buffer to be encoded was queued
- Afterwards all the remaining CAPTURE buffers are released and can be dequeued
- After the last CAPTURE buffer was dequeued the application is notified using an V4L2_EVENT_EOS event

How to get started with a video codec driver

- **Look at existing implementations of codecs in V4L2**
  - MFC versions 5.1 and 6
    - drivers/media/platform/s5p-mfc
  - CODA codec driver by Javier Martin
    - drivers/media/platform/coda.*
- **Flick through other m2m drivers**
  - FIMC driver by Sylwester Nawrocki
    - drivers/media/platform/s5p-fimc
    - Not only m2m, also supports camera capture
  - G2D driver - 2D Graphics Acceleration Module
    - drivers/media/platform/s5p-g2d

How to get started with a video codec driver

- There is good documentation of V4L2
  - http://linuxtv.org/downloads/v4l-dvb-apis/
- Linux-media mailing list
  - http://vger.kernel.org/vger-lists.html#linux-media
- #v4l IRC channel on Freenode
- CMA might be useful for memory management
- Start off with a simple driver, get to know the V4L2 framework and then add more functionality

Summary

- **Around two years from the start of work on MFC driver untill it was merge in the mainline kernel**
- **Many discussions on the linux-media mailing list (Thanks!)**
- **Multiple features added to V4L2**
  - Memory-to-memory framework
  - Multiplanar API
  - New controls and pixel formats
  - Videobuf2
- **Contiguous Memory Allocator**
- **Some wrinkles to iron out**

Summary

- Success #1 – API merged in the mainline kernel
- Success #2 – more hardware video codecs using Vide4Linux2 (CODA codec driver by Javier Martin)
- Success #3 – ARM Chromebook uses V4L2 Codec API

Links

- **Video4Linux 2 documentation**

  http://linuxtv.org/downloads/v4l-dvb-apis/

- **MFC driver code in git.kernel.org**

  http://goo.gl/LbhAE

- **MFC decoder example code**

  http://git.infradead.org/users/kmpark/public-apps/tree/HEAD:/v4l2-mfc-example

- **MFC encoder example code**

  http://git.infradead.org/users/kmpark/public-apps/tree/HEAD:/v4l2-mfc-encoder

- **Samsung OpenMAX components in Bellagio OMX IL**

  http://omxil.git.sourceforge.net/git/gitweb.cgi?p=omxil/components;a=shortlog;h=refs/heads/samsung

  http://omxil.git.sourceforge.net/git/gitweb.cgi?p=omxil/omxil;a=shortlog;h=refs/heads/samsung

SAMSUNG

Thank you.