Simplify Graphical User Interface and **Video Integration** for i.MX 6 Series Processors

AMF-SHB-T1035

Tore Slotfeldt | Graphics Ecosystem Manager
Original Slides Created by Daniele Dall'Acqua

M A R . 2 0 1 5

freescale™

External Use

Session Introduction

- This session will introduce the audience to the problem of the composition of video elements with graphic elements and how this can be simplified using i.MX 6 series hardware components.

- This presentation will show simple methods for effectively using these hardware components, supported by several useful software code examples, based on Gstreamer Multimedia framework.

Session Objectives

- After completing this session you will be able to:
  - Effectively use i.MX 6 hardware component software API for video & graphic element composition.
  - Create a simple multimedia player application, based on Gstreamer, that can be integrated with any graphical framework.

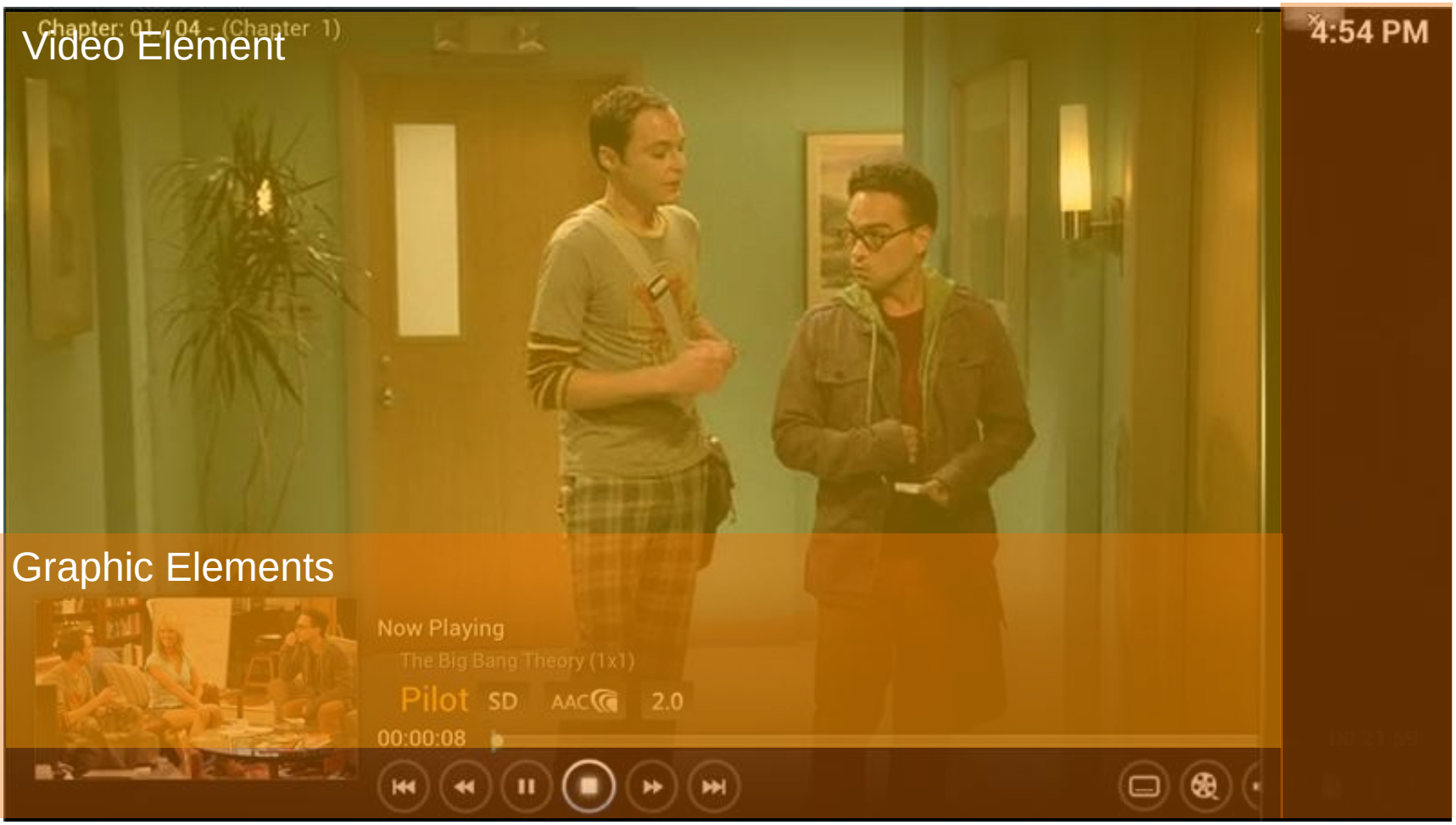
Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

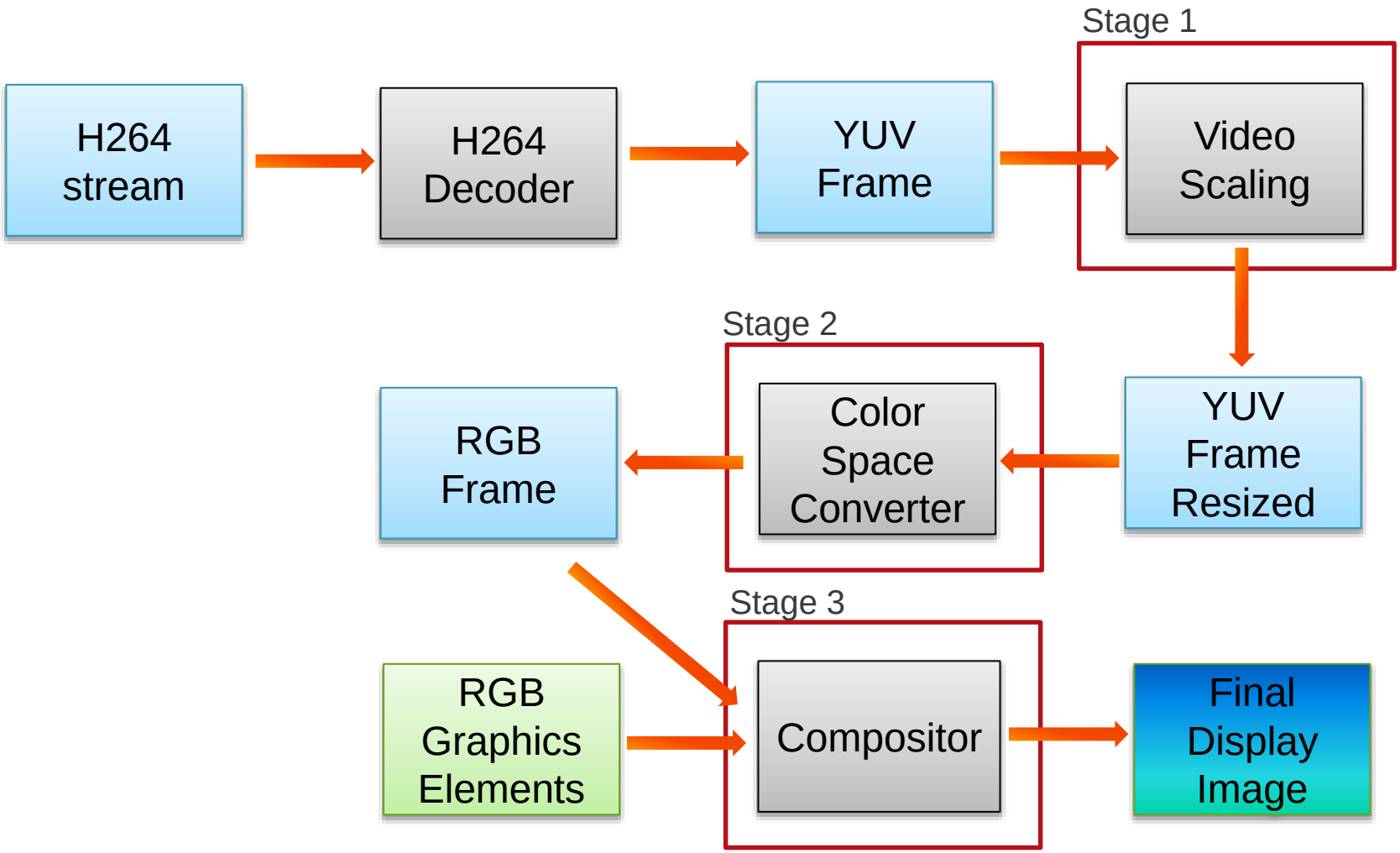
Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

Video & Graphic Composition Example



Simplified Video & Graphic Composition Flow



Stage 1: Resizing

- Before combining a video frame with a graphical element, the video frame might need to be resized, in order to meet the graphical interface specific needs.

- Resizing requires calculation per pixel

- Example: EPX/Scale2×/AdvMAME2×

```
  A    --\ 1 2
C P B  --/ 3 4
  D
1=P; 2=P; 3=P; 4=P;
IF C==A AND C!=D AND A!=B => 1=A
IF A==B AND A!=C AND B!=D => 2=B
IF B==D AND B!=A AND D!=C => 4=D
IF D==C AND D!=B AND C!=A => 3=C
```

Stage 2: Color Space Conversion

- For historical reasons, video uses a different color space than graphics.
- Compressed video formats (i.e. H264, MP4) use YUV.
- Each video frame needs to be converted to RGB before being displayed in a digital system.
- Converting YUV to RGB requires complex calculations per pixel.

$$R = Y' + V\frac{1-W_R}{V_{Max}} = Y' + \frac{V}{0.877}$$

$$G = Y' - U\frac{W_B(1-W_B)}{U_{Max}W_G} - V\frac{W_R(1-W_R)}{V_{Max}W_G} = Y' - \frac{0.232U}{0.587} - \frac{0.341V}{0.587} = Y' - 0.395U - 0.581V$$

$$B = Y' + U\frac{1-W_B}{U_{Max}} = Y' + \frac{U}{0.492}$$

Stage 3: Combining, alpha & color key

- When composing the final image, combining video and graphical elements, it is required to determine which pixel has to be displayed in the location where the two overlap.

- Color Key (or Chroma Key): a method that makes a color of an element transparent during the composition

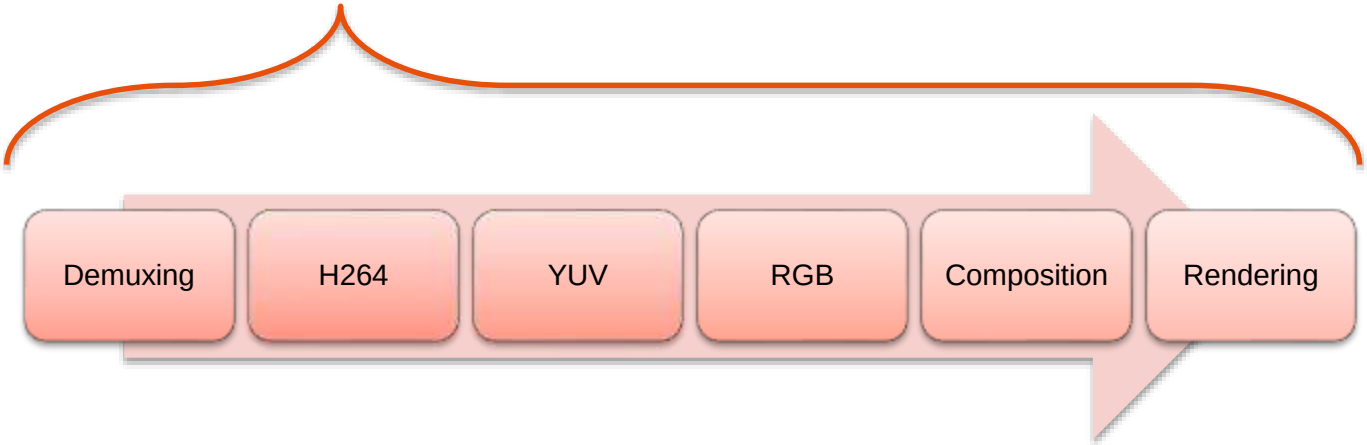
- Alpha: a value that tells in which proportion the pixel of one graphic plane weight, when it is combined with another graphic plane in the resulting pixel. It can be a global value or it can be per pixel.

Computational Weight

- Those 3 stages have impact either on CPU & memory bandwidth.
- E.G for 720p30 on 1080p30 Screen
  - Stage 1: Resizing from 720p to 1080p
    720x1280x30x1.5 + 1920x1080x30x1.5 = **128Mbytes/sec**
  - Stage 2: CSC Conversion
    1920x1080x30x1.5 + 1920x1080x30x4 = **326Mbytes/sec**
  - Stage 3: Combining (Worst Case)
    1920x1080x30x4x3 =**711Mbytes**
- In this case, the CPU has to handle more than **1Gbytes/Sec**!
- If the data flow requires an extra step like rotation and/or video deinterlacing the data rate will increase.

Real Time

- Video Playback is a real time operation.
- When a frame can't be processed within the deadline, as per stream frame rate needs, it must be dropped. Otherwise the stream will loose the audio/video sync.
- Example: With a 720p30 stream, the full frame processing should not take more than 33ms, from decoding to rendering.



Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

i.MX Application Processors: The Unbound User Experience

Display-centric devices are the fastest growing segment of the market

Freescale Target Markets

Over **150 million** processors shipped

Top 3 AP silicon vendor

- Power & Performance Leadership
  - 1st Quad A9 + 64b memory, 1st Cortex CPU + integrated EPD
  - Multicore CPU, multicore graphics, multi-stream video, console class 3D, rich interfaces
- Smooth Scalability
  - Quad, Dual and Single core CPU offerings
  - Best in class flexibility/integration: Consumer, Auto, Industrial IO's + qualifications + BGA/POP offerings
    - GbE, PCIe, SATA, MIPI, USB, HDMI, LVDS, EPD, CAN, MLB, etc
    - Full PMIC integration (lower complexity & BOM)
    - LP-DDR2 / DDR3 (cost versus power options)
  - Single SW investment, multiple devices in market
  - Pin compatibility
- Trusted Technology
  - Up to 15 year life cycle support
  - Auto & Industrial grade quality design practices
  - Fully featured, market targeted reference designs
  - Android, Microsoft, QNX, Linux, Ubuntu optimizations

i.MX 6 Series Processors Overview

Scalable series of five ARM Cortex A9-based SoC families

i.MX 6SoloLite

- 1x 1GHz
- x32 400MHz DDR3
- No HW video accel.
- 2D graphics (2 GPUs)
- LCD, EPD

i.MX 6Solo

- 1x 1GHz
- x32 400MHz DDR3
- **HD1080p video**
- 2D+**3D** (2 GPUs), 53Mtri/s
- LCD, EPD

i.MX 6DualLite

- **2x** 1GHz
- **x64** 400MHz DDR3
- HD1080p video
- 2D+3D (2 GPUs), 53Mtri/s
- LCD, EPD

i.MX 6Dual

- 2x 1/**1.2GHz**
- x64 **533MHz** DDR3
- **Dual** HD1080p video
- 2D+3D (3 GPUs), **176 Mtri/s**
- LCD

i.MX 6Quad

- **4x** 1/1.2GHz
- x64 533MHz DDR3
- Dual HD1080p video
- 2D+3D (3 GPUs), 176 Mtri/s
- LCD

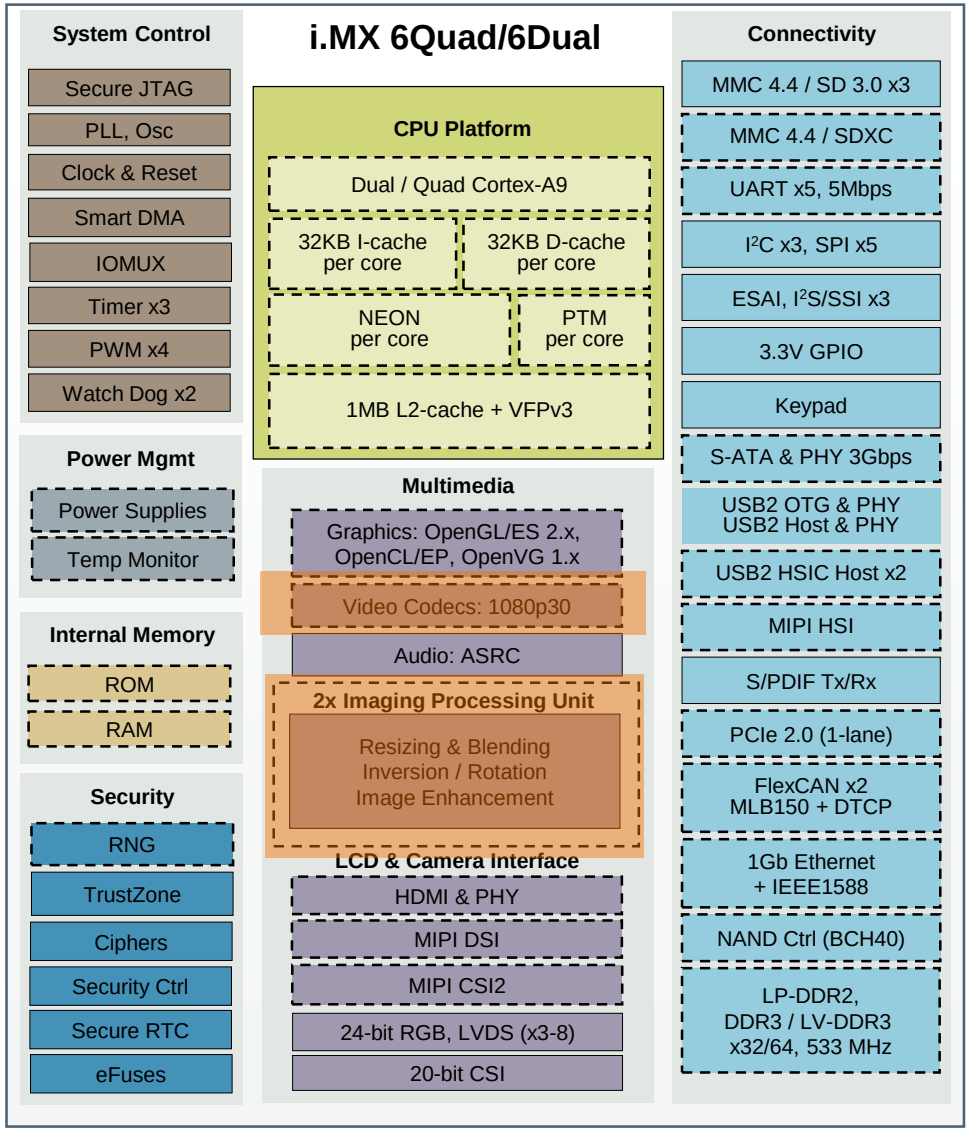
Pin-to-pin Compatible

Software Compatible

i.MX 6Quad/6Dual Applications Processor

- **Specifications**
  - CPU: i.MX 6Quad 4x Cortex-A9 @1.2 GHz, 12000 DMIPS
  - i.MX 6Dual 2x Cortex-A9 @1.2 GHz, 6000 DMIPS
    - Process: 40nm
    - Core Voltage: 1.1V
    - Package: 21x21 0.8mm Flip-chip BGA

- **Key Features and Advantages**
  - Multicore architecture for high performance, 1MB L2 cache
  - 64-bit LP-DDR2, DDR3 and raw / managed NAND
  - S-ATA 3Gbps interface (SSD / HDD)
  - Delivers rich graphics and UI in hardware
    - OpenGL/ES 2.x 3D accelerator with OpenCL EP support and OpenVG 1.1 acceleration
  - Drives high resolution video in hardware
    - Multi-format HD1080 video decode and encode
    - 1080p60 decode, 720p60 encode
    - High quality video processing (resizing, de-interlacing, etc.)
  - Flexible display support
    - Four simultaneous: 2x Parallel, 2x LVDS, MIPI-DSI, or HDMI
    - Dual display up to WUXGA (1920x1200) and HD1080
  - MIPI-CSI2 and HSI
  - Increased analog integration simplifies system design and reduces BOM
    - DC-DC converters and linear regulators supply cores and all internal logic
    - Temperature monitor for smart performance control
  - Expansion port support via PCIe 2.0
  - Car network: 2xCAN, MLB150 with DTCP, 1Gb Ethernet with IEEE1588



i.MX 6 Series Overview

Scalable series of five ARM Cortex A9-based SoC families

i.MX 6SoloLite

- 1x 1GHz
- x32 400MHz DDR3
- No HW video accel.
- 2D graphics (2 GPUs)
- LCD, EPD

i.MX 6Solo

- 1x 1GHz
- x32 400MHz DDR3
- **HD1080p video**
- 2D+**3D** (2 GPUs), 53Mtri/s
- LCD, EPD

i.MX 6DualLite

- **2x** 1GHz
- **x64** 400MHz DDR3
- HD1080p video
- 2D+3D (2 GPUs), 53Mtri/s
- LCD, EPD

i.MX 6Dual

- 2x 1/**1.2GHz**
- x64 **533MHz** DDR3
- **Dual** HD1080p video
- 2D+3D (3 GPUs), **176 Mtri/s**
- LCD

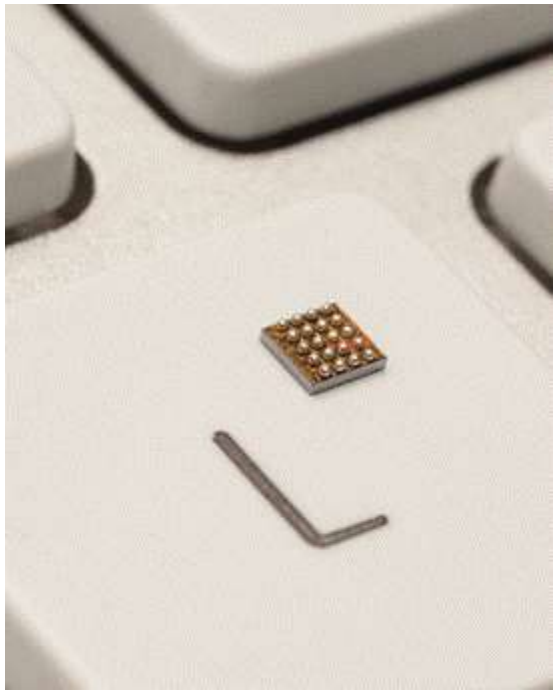
i.MX 6Quad

- **4x** 1/1.2GHz
- x64 533MHz DDR3
- Dual HD1080p video
- 2D+3D (3 GPUs), 176 Mtri/s
- LCD

Pin-to-pin Compatible

Software Compatible

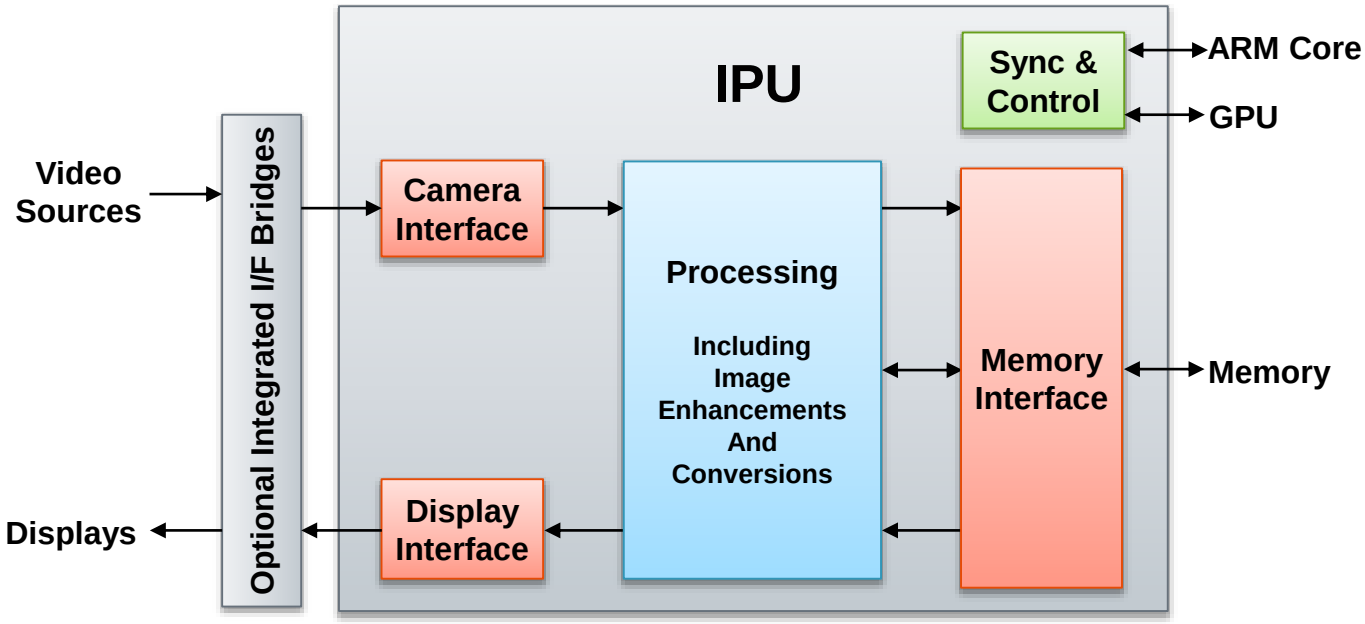
The Image Processing Unit (IPU)



Functions: comprehensive support for the flow of data from an image sensor and/or to a display device.

- Connectivity to relevant devices
- Image processing and manipulation
- Synchronization and control capabilities

The Image Processing Unit



IPUv3 (in 6Q, 6D, 6DL, 6S)

- **De-interlacing**

  A motion adaptive filter; using three input fields

  - For slow motion – retains the full resolution (of both top and bottom fields), by using temporal interpolation
  - For fast motion – prevents motion artifacts, by using vertical interpolation
- **Resizing**
  - Bi-linear interpolation, with flexible resizing ratio
  - For radical downsizing: first x2 or x4 downsizing by averaging
- **Video/Graphics Overlays**
- **Display Enhancement**
  - Color adjustment – general linear transformation; followed by a hue-preserving gamut mapping
  - Gamma adjustment – configurable piecewise linear transformation.
  - May be used to compensate for low-light conditions and to allow backlight reduction
- **Other**
  - Rotation: 90, 180, 270 degrees
  - Inversion: horizontal and vertical
  - Color Space Conversion

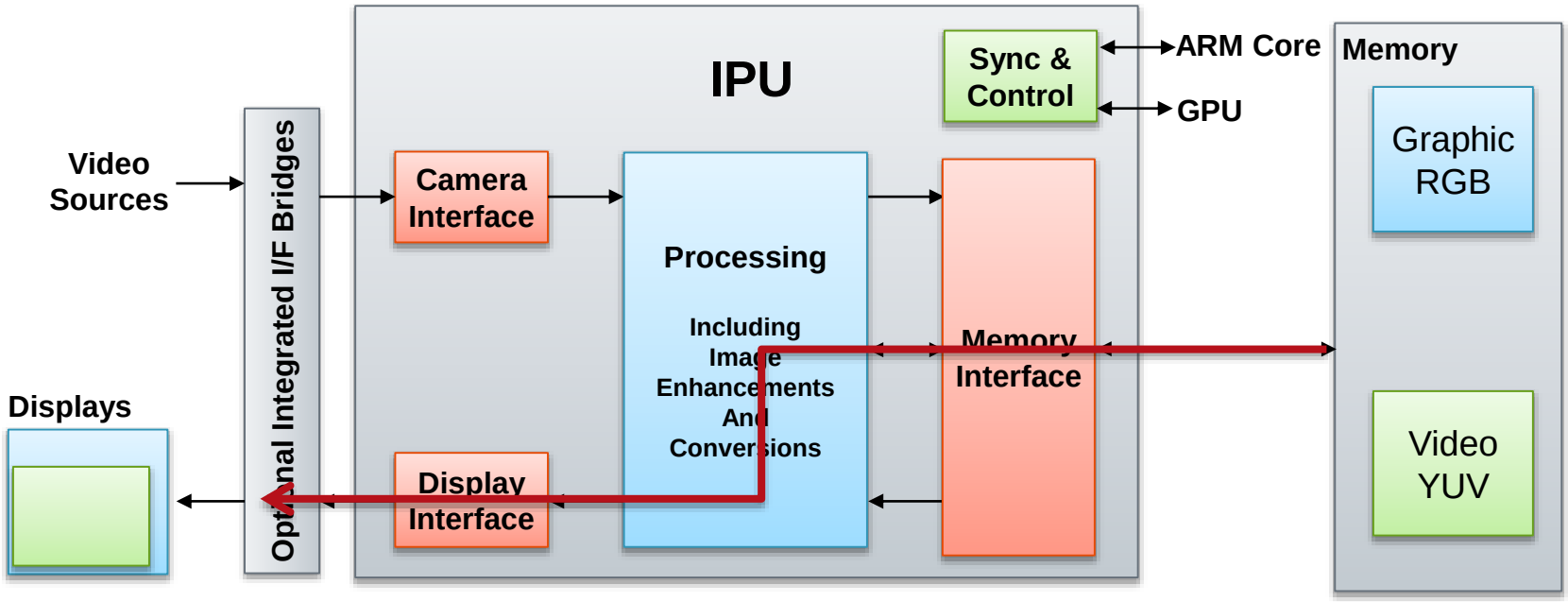
IPUv3 (in 6Q, 6D, 6DL, 6S)

- **De-interlacing**

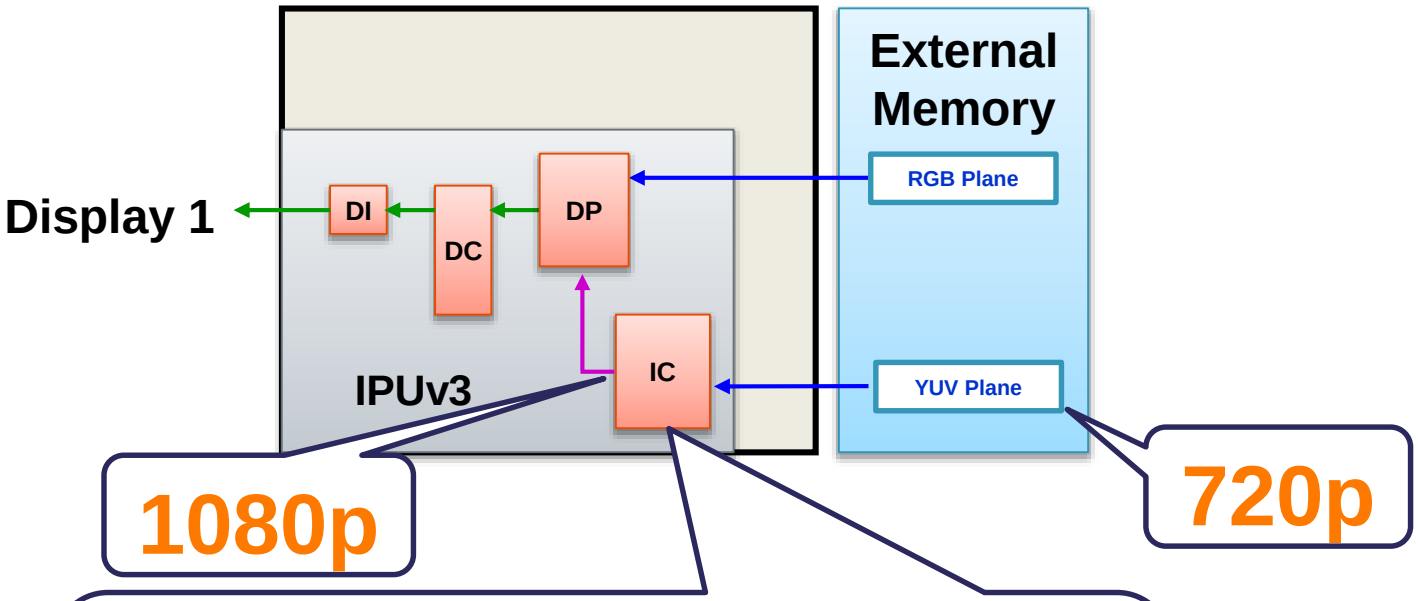
  A motion adaptive filter; using three input fields

  - For slow motion – retains the full resolution (of both top and bottom fields), by using temporal interpolation
  - For fast motion – prevents motion artifacts, by using vertical interpolation
- **Resizing**
  - Bi-linear interpolation, with flexible resizing ratio
  - For radical downsizing: first x2 or x4 downsizing by averaging
- **Video/Graphics Combining**
- **Display Enhancement**
  - Color adjustment – general linear transformation; followed by a hue-preserving gamut mapping
  - Gamma adjustment – configurable piecewise linear transformation.
  - May be used to compensate for low-light conditions and to allow backlight reduction
- **Other**
  - Rotation: 90, 180, 270 degrees
  - Inversion: horizontal and vertical
  - Color Space Conversion

The Image Processing Unit (Session Focus)



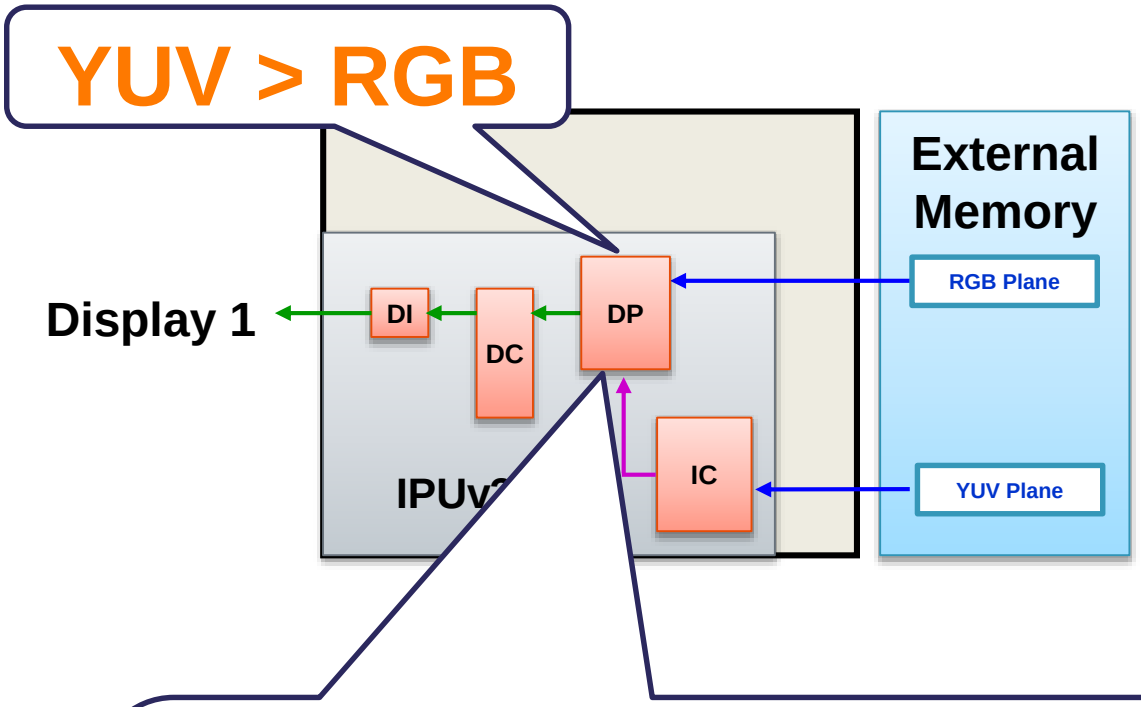
IPUv3 Resizing



Resizing in the Image Converter (IC)

- Fully flexible resizing ratio Maximal downsizing ratio: 8:1. Subject to this limitation, any N->M resizing can be performed.
- Independent horizontal and vertical resizing ratios.
- Output rate 100Mpixels/sec (e.g. 1920x1080 @ 30 fps)
- One active flow at time to DP

IPUv3 Color Space Conversion

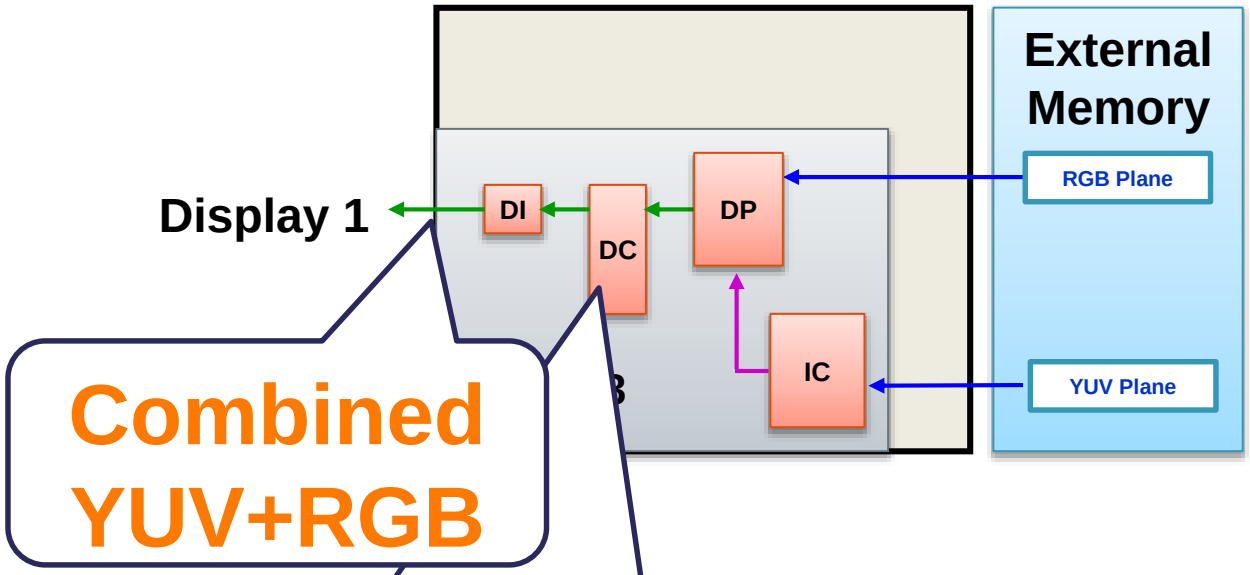


Color Space Conversion in the Display Processor (DP)

Color conversion/correction - linear (multiplicative & additive) programmable including:

- YUV <-> RGB, YUV<->YUV conversions where YUV stands for any one of the color formats defined in the MPEG-4 standard
- Adjustments: brightness, contrast, color saturation, etc.
- Special effects: gray-scale, color inversion, sephia, blue-tone, etc.
- Color-preserving clipping, for gamut mapping
- Hue-preserving gamut mapping - for minimal color distortion
- Applied to the output of combining or to one of the inputs

IPUv3 Video/Graphics Combining (Overlay Capabilities)



Combining in the Display Processor (DP)

Two planes

- One plane may have any size and location
- The other one must be “full-screen” (cover the full output area)
- Output 266 MP/sec

Combining methods

- Color keying and/or alpha blending
- Alpha: global or per-pixel; interleaved with the pixels (upper plane) or as a separate input

IPUv3 HW Flow Advantages

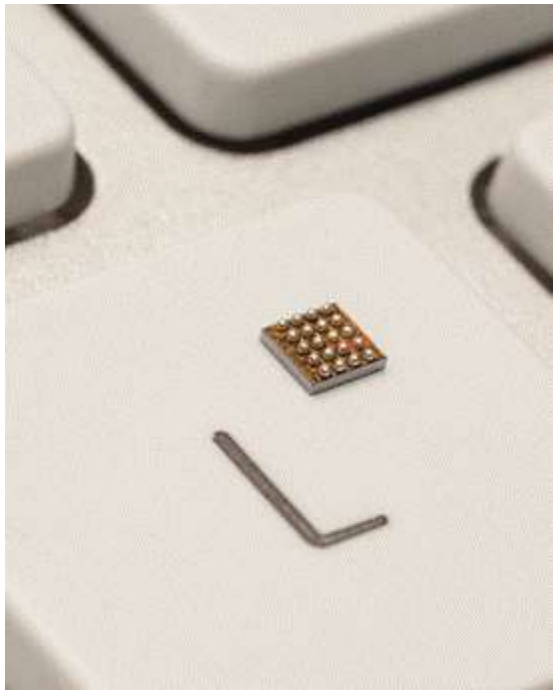
- No Impact over CPU

- Reduced DDR/RAM bandwidth overhead
  - Same use cases as shown in the introduction it will become
    - 1280x720x30x1.5 = 40Mbytes /sec VS 1Gbytes/sec+

Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

The Video Processing Unit



- Video Processing Unit (VPU) of i.MX 6 is a high performance multi-standard video codec
- VPU can support encode or decode of multiple video clips with multiple standards simultaneously.

i.MX 6 Series VPU: *Decoder*

	Standard	Profile	Performance (2D or 3D)		Bitrate
HW Decoder	MPEG-2	Main-High	2D	1080i/p+720p@30fps Dual 720p @30fps Dual 1080i/p @30ps **(TBD)**	50Mbps
	H.264	BP/MP/HP-L4.1			50Mbps
	VC1	SP/MP/AP-L3			45Mbps
	MPEG4	SP/ASP			40Mbps
	DivX/XviD	3/4/5/6			40Mbps
	AVS	Jizhun			40Mbps
	H.263	P0/P3		1080p+720p@30fps Dual 720p @30fps Dual 1080p @30fps **(TBD)**	20Mbps
	MJPEG	Baseline		8k x 8k	120Mpel/s
	On2 VP8	--		720p@30fps 1080p@30fps for i.MX 6Q/D (**TBD**) 1080p@30fps for i.MX 6DL/S	20Mbps
	H.264-MVC for 3D (FW/HW)	H.264-MVC SHP	3D	720p@30fps each view 1080p@24fps each view 1080p@30fps each view (**TBD**)	40Mbps
	Simulcast for 3D	Two independent streams		720p@30fps each view 1080i/p@24fps each view 1080i/p@30fps each view (**TBD**)	50Mbps
	Frame-packing for 3D	Combine two frames into one		1080p@30fps decode → 1080p@30fps each view playback	50Mbps
HW Post-proc	Rotation, mirror, deblocking/deringing				

i.MX 6 Series VPU: *Encoder*

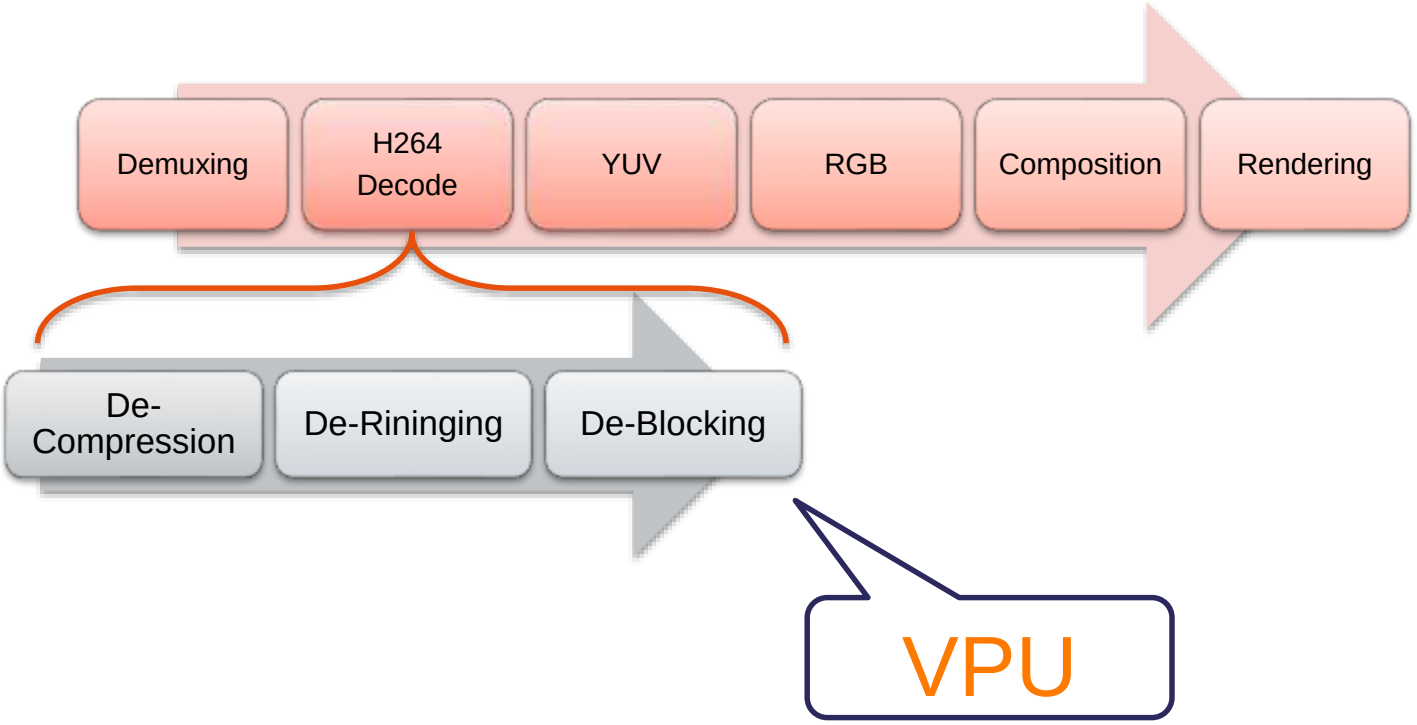
	Standard	Profile	Performance (2D or 3D)		Bitrate
HW Encoder	H.264	BP	2D	1080p@30fps 720p@60fps	20Mbps
	MJPEG	Baseline		8k x 8k	160Mpel/s
	MPEG4	Simple		720p@30fps	15Mbps
	H.263	P0/P3		720p@30fps	15Mbps
	H.264-MVC for 3D	Stereo HP (no interview prediction)	3D	720p@30fps each view 1080p@24fps each view (**TBD**)	20Mbps
	Simulcast for 3D	Any VPU encoder supported profiles		720p@30fps each view 1080p@24fps each view (**TBD**)	20Mbps
	Frame-packing	Any VPU encoder supported profiles		1080p@30fps encoding → 1080p@30fps each view capture	20Mbps

i.MX 6 Series VPU: *Multi-streams*

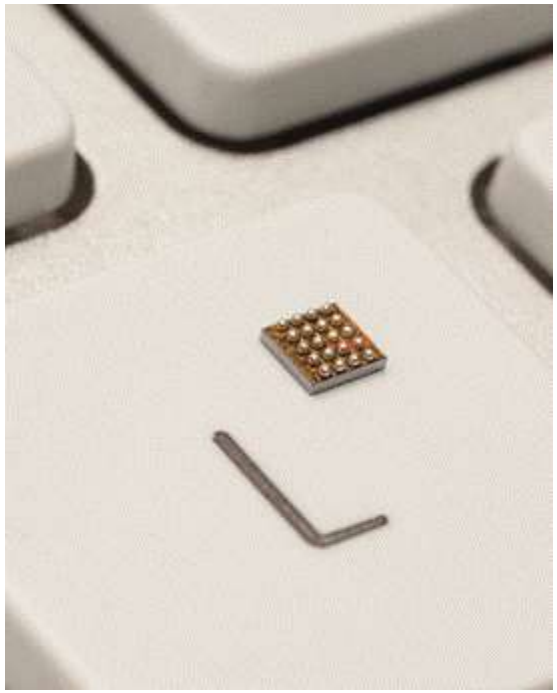
	Standard	Profile	Max # Streams			
			D1@ 30fps	720p@ 30fps	1080p@ 24fps	1080p@ 30fps
HW Decoder	H.264	BP/MP/HP	8	3	2	1
	On2 VP8	--	4	2	1	1 (iMX 6Q/D, **TBD**) 1 (iMX 6DL/S)
	VC1	SP/MP/AP	8	3	2	1
	MPEG4	SP/ASP	8	3	2	1
	H.263	P0/P3	8	3	2	1
HW Encoder	H.264	BP	6	2	2 (**TBD**)	1
	MPEG4-SP/H.263	MPEG4-SP H.263-P0/P3	6	2	--	--

i.MX 6 Series VPU

In the Frame decodecing flow we have a sketch at the beginning, the goal of the VPU is decoding the compressed stream in the YUV frame.



i.MX 6 IPU & VPU Software



- IPU & VPU functionalities can be accessed under Linux OS using specific libraries, API or even via the driver directly.
- On top of these libraries, Freescale provides Gstreamer Element to further simplify the integration into applications.

Checkpoint Summary

- ✓Composing video and graphical elements is a CPU/RAM bounded task

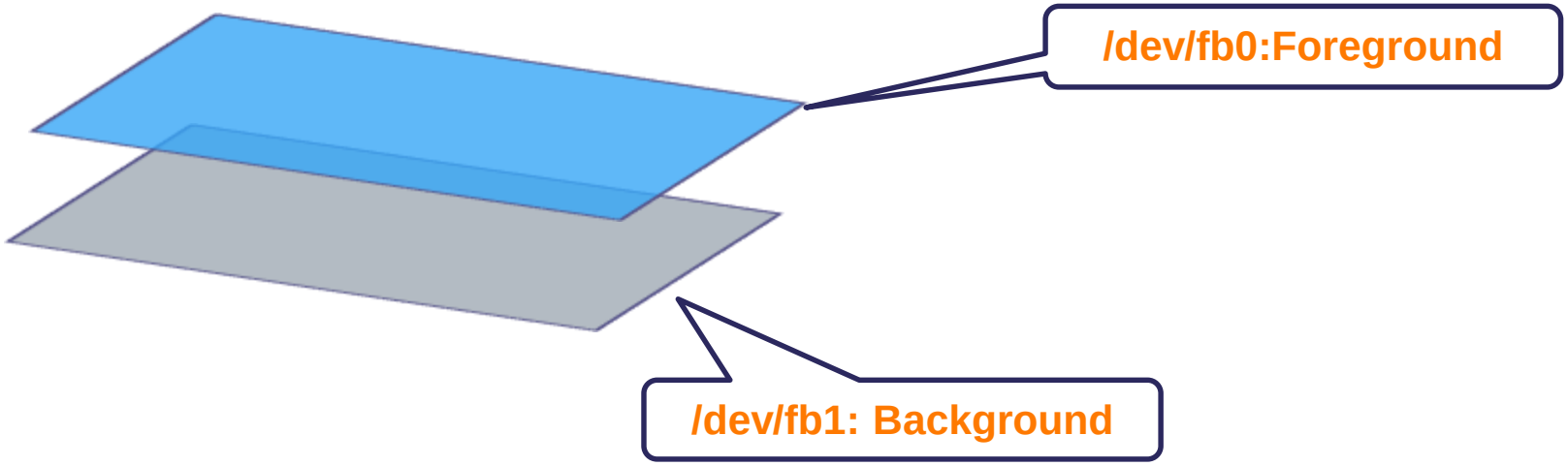
- ✓i.MX 6 Series has a hardware component that can minimize the RAM load and completely offloading the CPU load for this.

Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

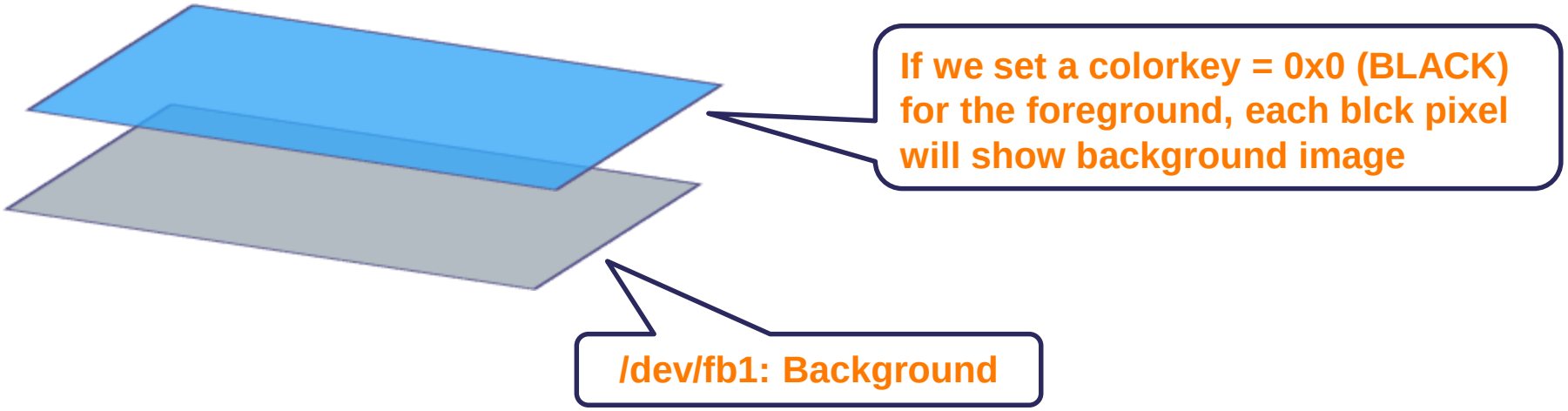
i.MX 6 Display Composition API for Linux

- Foreground is usually dedicated to Graphical Elements
- Background to Video Elements



i.MX 6 Display Composition API for Linux: Colorkey

- A color key applied to one of the two planes indicates a “COLOR” which becomes transparent when overlayed with the other plane.



i.MX 6 Display Composition API for Linux: Colorkey how-to

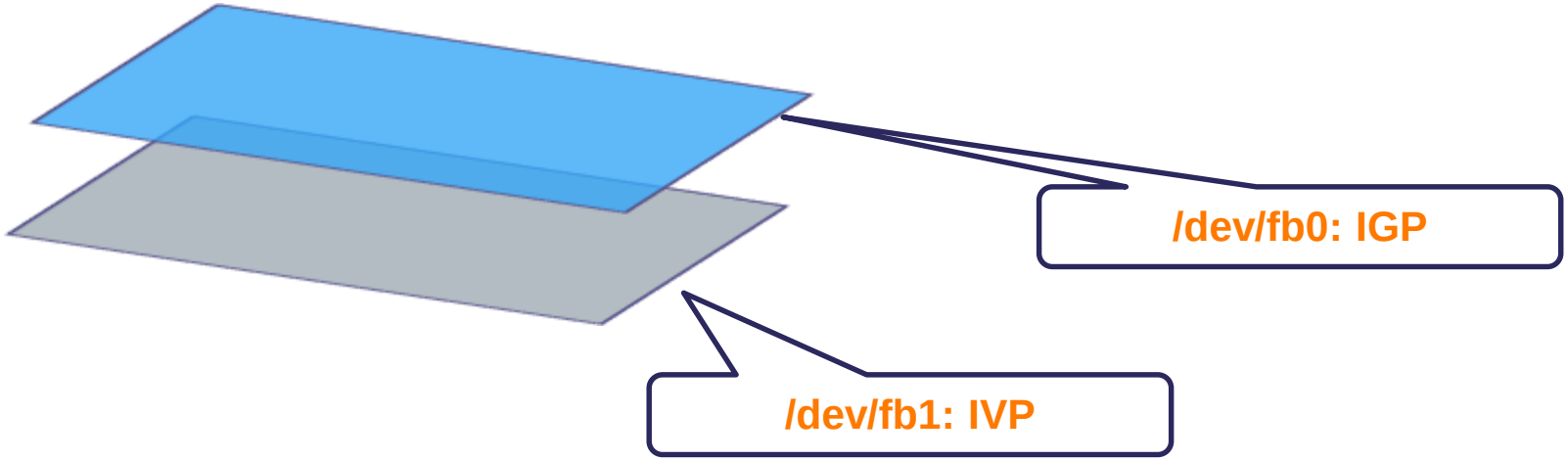
- Example

```
int main (int argc, char *argv[])
{
    struct mxcfb_color_key color_key;
    color_key.color_key = 0x0;
    color_key.enable = 1;

     if ((fd_fb = open("/dev/fb0", O_RDWR, 0)) < 0)
    {
              return 0;
     }
     if ( ioctl(fd_fb, MXCFB_SET_CLR_KEY, &color_key) < 0) {
              printf("Error in applying Color Key\n");
     }

     close (fd_fb);
     return 0;
}
```

i.MX 6 Display Composition API for Linux: Global Alpha

- A global alpha value applied to one of the two planes makes that plane more transparent versus the other plane. Following the equation:

$$OP = IGP \times \alpha + IVP \times (1 - \alpha)$$
$$\alpha = (A + floor(A/128))/256$$
$$A = [0..255]$$



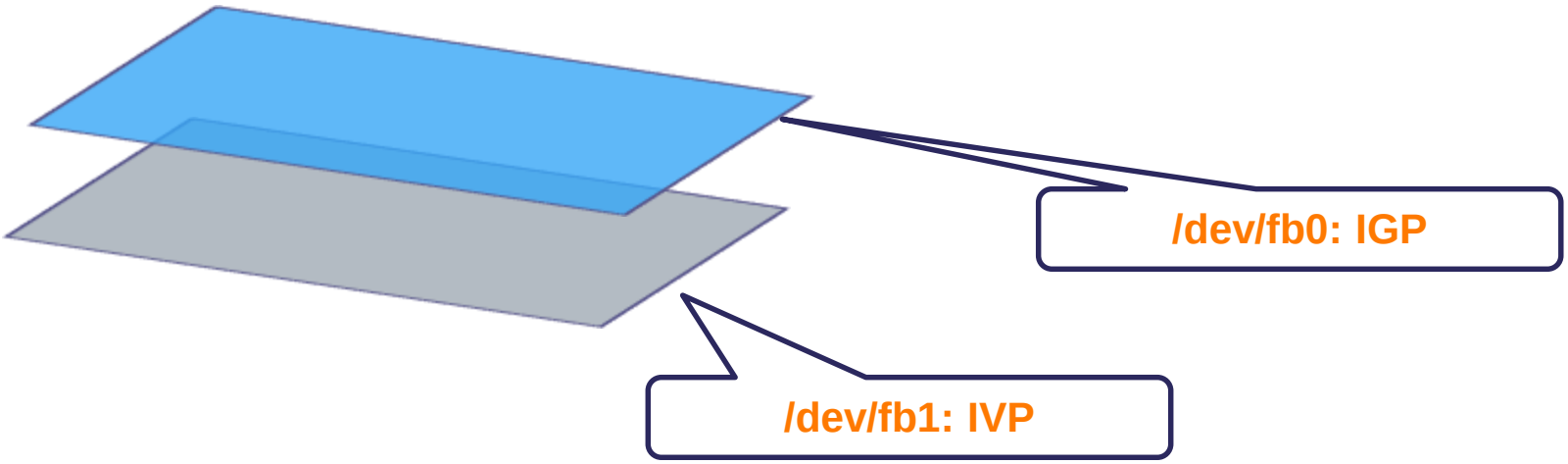
i.MX 6 Display Composition API for Linux: Global Alpha how-to

- Example

```
int main (int argc, char *argv[])
{
     struct mxcfb_gbl_alpha alpha;
      alpha.alpha = 128;
      alpha.enable = 1;

     if ((fd_fb = open("/dev/fb0", O_RDWR, 0)) < 0)
     {
               return 0;
      }
       if ( ioctl(fd_fb, MXCFB_SET_GBL_ALPHA, &alpha) < 0) {
               printf("Error in applying Alpha\n");
      }
      close (fd_fb);
      return 0;
}
```

i.MX 6 Display Composition API for Linux: Local Alpha

- It acts like global alpha, except that the alpha value is embedded in the PIXEL. Thus, the Framebuffer format should be 32-bit (ARGB).



i.MX 6 Display Composition API for Linux: Local Alpha how-to

- Example

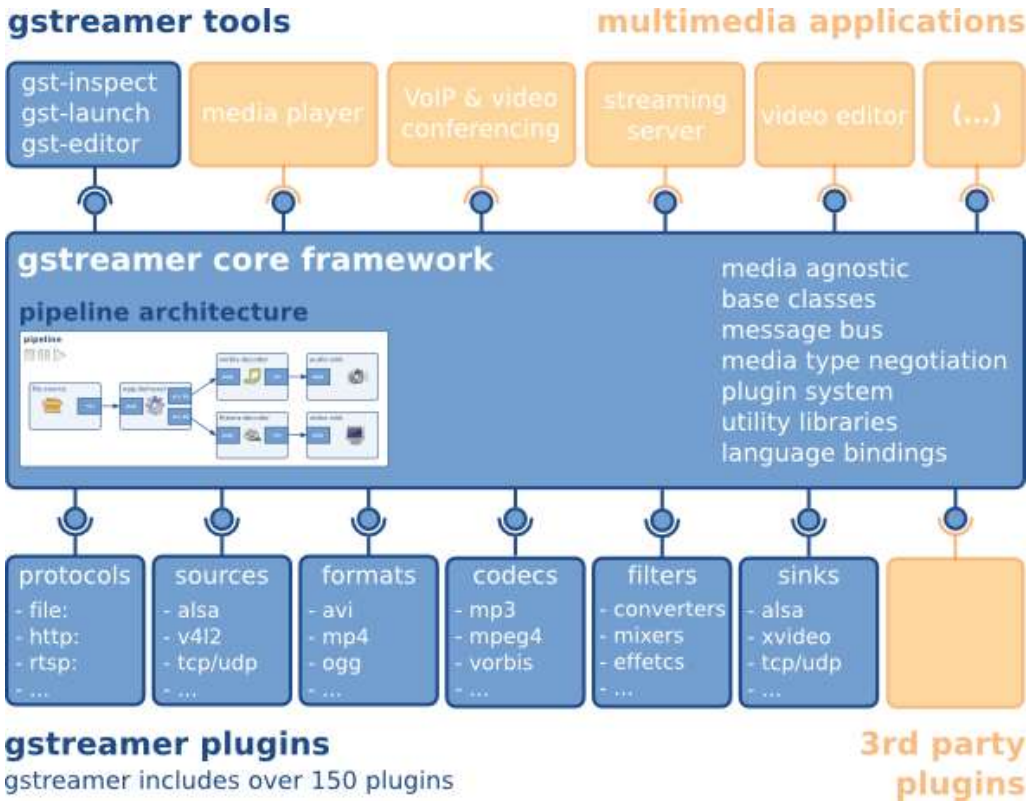
```
int main (int argc, char *argv[])
{
    struct mxcfb_loc_alpha alpha;
    alpha.alpha_in_pixel = 1;
    alpha.enable = 1;

    if ((fd_fb = open("/dev/fb0", O_RDWR, 0)) < 0)
    {
              return 0;
     }
      if ( ioctl(fd_fb, MXCFB_SET_LOC_ALPHA, &alpha) < 0) {
                        printf("Error in enabling Local Alpha\n");
     }
     close (fd_fb);
     return 0;
}
```

Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

Gstreamer Overview

- GStreamer is a framework for creating streaming media applications



Gstreamer Terminology

- Elements
  - An element is the most important class of objects in GStreamer. You will usually create a chain of elements linked together and let data flow through this chain of elements. An element has one specific function, which can be the reading of data from a file, decoding of this data or outputting this data to your sound card (or anything else).
  - By chaining together several such elements, you create a pipeline that can do a specific task, for example media playback or capture.
  - GStreamer ships with a large collection of elements by default, making the development of a large variety of media applications possible

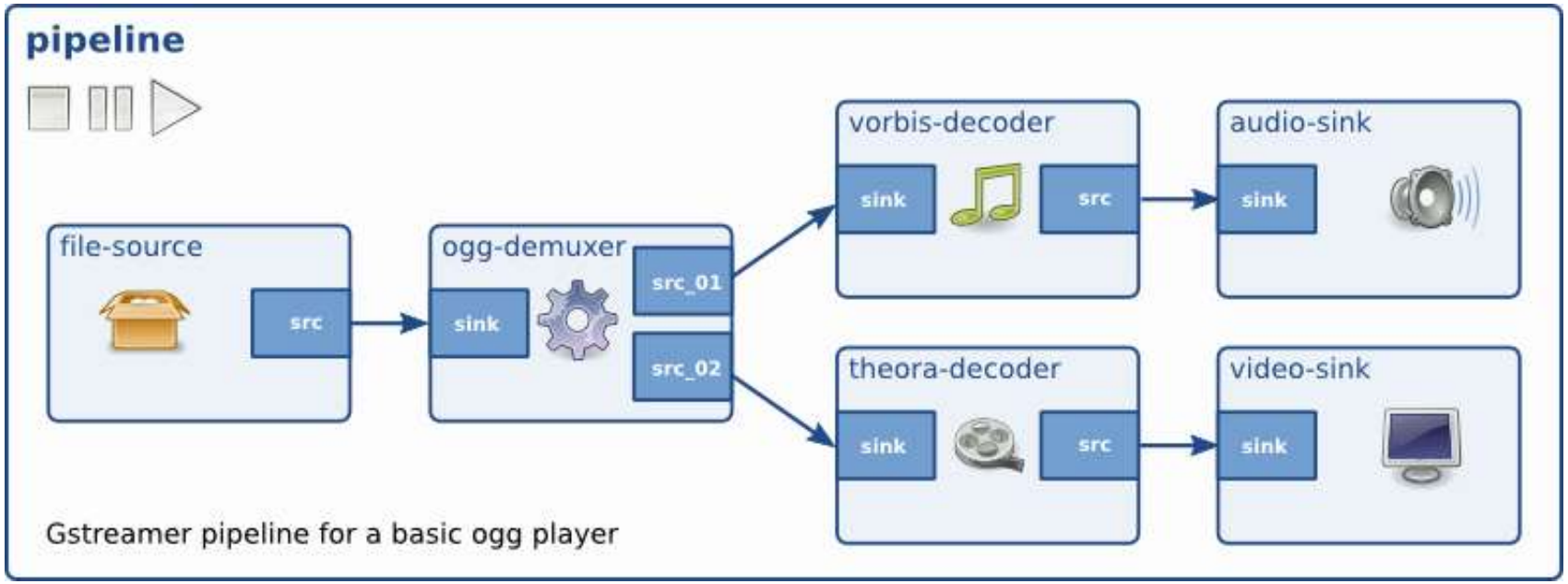
Gstreamer Terminology

- Pads
  - Pads are element's input and output, where you can connect other elements. They are used to negotiate links and data flow between elements in GStreamer.

- Bins and Pipelines
  - A bin is a container for a collection of elements. A pipeline is a special subtype of a bin that allows execution of all of its contained child elements. Since bins are subclasses of elements themselves, you can mostly control a bin as if it were an element, thereby abstracting away a lot of complexity for your application.

Gstreamer Terminology

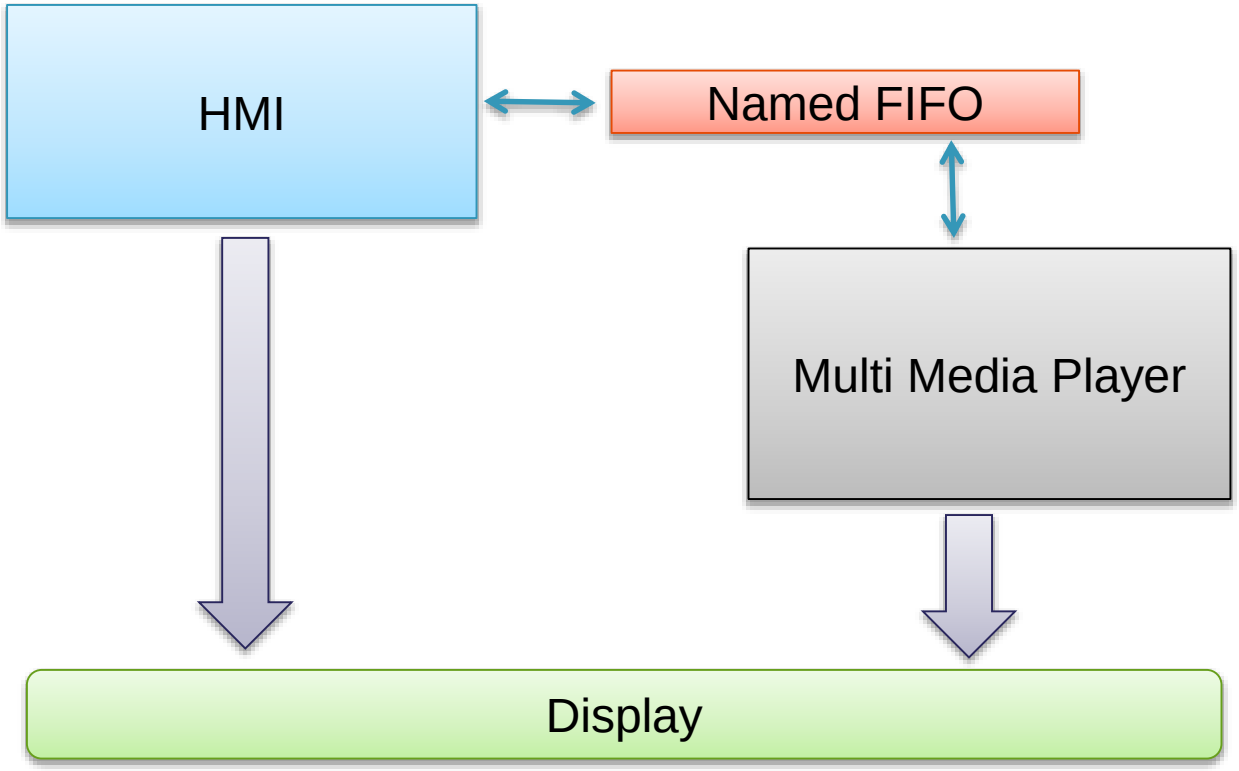
- Example of a Gstreamer pipeline



Agenda

- Introduction to Video & Graphic element composition problem.
- i.MX 6 Series graphical hardware components
- i.MX 6 Display composition API for Linux Framebuffer
- Gstreamer elements
- Multimedia Player Example

Multimedia Player Code Example (SW Architecture)



Multimedia Player Code Example

- Media server communication with Application it is implemented via Named Pipe (FIFO).
- Media Player state machine it is mainted inside the media server
- Playback it is managed via Gstreamer

Multimedia Player Code Example (Basic Example)

```c
#include <gst/gst.h>

int main(int argc, char *argv[]) {
  GstElement *pipeline;
  GstBus *bus;
  GstMessage *msg;

  /* Initialize GStreamer */
  gst_init (&argc, &argv);

  /* Build the pipeline */
  pipeline = gst_parse_launch ("playbin2 uri=http://docs.gstreamer.com/media/sintel_trailer-480p.webm", NULL);

  /* Start playing */
  gst_element_set_state (pipeline, GST_STATE_PLAYING);

  /* Wait until error or EOS */
  bus = gst_element_get_bus (pipeline);
  msg = gst_bus_timed_pop_filtered (bus, GST_CLOCK_TIME_NONE, GST_MESSAGE_ERROR | GST_MESSAGE_EOS);

  /* Free resources */
  if (msg != NULL)
    gst_message_unref (msg);
  gst_object_unref (bus);
  gst_element_set_state (pipeline, GST_STATE_NULL);
  gst_object_unref (pipeline);
  return 0;
}
```

Multimedia Player Code Example (State Machine Control)

```
int main (int argc, char *argv[])
{
   int pipe_in;
   int pipe_out;
   ....
   gst_init (0, NULL);
   memset(file_to_play,0,MAX_LINE*sizeof(char));

   mkfifo("/tmp/mmp_out_fifo", 0666);
   mkfifo("/tmp/mmp_in_fifo", 0666);

   pipe_in  = open("/tmp/mmp_in_fifo", O_RDWR);
   pipe_out = open("/tmp/mmp_out_fifo",O_RDWR);
   for(;;)
   {
        memset(cmd,0,MAX_LINE*sizeof(char));
        memset(pipe_cmd,0,MAX_LINE*sizeof(char))
        br = read(pipe_in, pipe_cmd, MAX_LINE*sizeof(char));
        sscanf(pipe_cmd,"%s\n", cmd);

        if (strcmp("play",cmd) == 0)
        {
                player_set_state(PLAY);
        }
     ....
 }
```

Multimedia Player Code Example (Player State Machine)

```
void player_set_state( int state)
{
         char gst_pipe_to_exec[MAX_LINE];
        if ((state == PLAY) && ((player_state == PAUSE) || (state == STOP)))
        {

              if ((pipeline != NULL) && (state == PAUSE))
              {
                     gst_element_set_state (pipeline, GST_STATE_PLAYING);
                     player_state = PLAY;

              } else {
                      FILE *file_test;
                      file_test = fopen(file_to_play,"r");
                      if (file_test != NULL)
                      {
                             fclose(file_test);
                             sprintf(gst_pipe_to_exec,"playbin2 video-sink=\"mfw_v4lsink disp-width=%d
                                          disp-height=%d axis-top=%d axis-left=%d\" uri=file://%s",
                                            player_width, player_height, player_top, player_left,
                                                      file_to_play);
                              pipeline = gst_parse_launch (gst_pipe_to_exec, NULL);
                              gst_element_set_state (pipeline, GST_STATE_PLAYING);
                              bus = gst_element_get_bus (pipeline);
                              player_state = PLAY;

                        }
                }
        }
        if ((player_state == PLAY) && (state == PAUSE))
        {
                gst_element_set_state(pipeline, GST_STATE_PAUSED);
                player_state = PAUSE;
        }
```

Session Closing

By now, you should be able to:

- Control the Video & Graphic composition via i.MX 6 Series HW components, using Linux Device Drivers API
- Create a simple Multimedia player based on Gstreamer Framework that can be re-used with any Graphical Framework.

www.Freescale.com