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

FTF-CON-F0223

Daniele Dall’Acqua | Senior Field Application Engineer

A P R . 2 0 1 4
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

Video Element

Graphic Elements
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
    \[720 \times 1280 \times 30 \times 1.5 + 1920 \times 1080 \times 30 \times 1.5 = 128\text{Mbytes/sec}\]
  - Stage 2: CSC Conversion
    \[1920 \times 1080 \times 30 \times 1.5 + 1920 \times 1080 \times 30 \times 4 = 326\text{Mbytes/sec}\]
  - Stage 3: Combining (Worst Case)
    \[1920 \times 1080 \times 30 \times 4 \times 3 = 711\text{Mbytes}\]
- 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 lose 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
Display-centric devices are the fastest growing segment of the market

- **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

Freescale Target Markets

- Over 150 million processors shipped
- Top 3 AP silicon vendor
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

**System Control**
- Secure JTAG
- PLL, Osc
- Clock & Reset
- Smart DMA
- IOMUX
- Timer x3
- PWM x4
- Watch Dog x2

**Power Mgmt**
- Power Supplies
- Temp Monitor

**Internal Memory**
- ROM
- RAM

**Security**
- RNG
- TrustZone
- Ciphers
- Security Ctrl
- Secure RTC
- eFuses

**Connectivity**
- MMC 4.4 / SD 3.0 x3
- MMC 4.4 / SDXC
- UART x5, 5Mbps
- FC x3, SPI x5
- ESAI, FS/SSI x3
- 3.3V GPIO
- Keypad
- S-ATA & PHY 3Gbps
- USB2 OTG & PHY
- USB2 Host & PHY
- USB2 HSIC Host x2
- MIPI HSI
- S/PDIF Tx/Rx
- PCIe 2.0 (1-lane)
- FlexCAN x2
- MLB150 + DTCP
- 1Gb Ethernet + IEEE1588
- NAND Ctrl (BCH40)
- LP-DDR2, DDR3 / LV-DDR3 x32/64, 533 MHz

**Updated from i.MX53**
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

- Video Sources
- Optional Integrated I/F Bridges
- Camera Interface
- Processing (Including Image Enhancements and Conversions)
- Memory Interface
- Display Interface
- Sync & Control
- ARM Core
- GPU
- Memory
- Displays
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)

Optional Integrated I/F Bridges

Video Sources

Camera Interface

Display Interface

Processing

Including Image Enhancements And Conversions

Memory Interface

Sync & Control

ARM Core

GPU

Memory

Graphic RGB

Video YUV
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
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

YUV > RGB
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
    - $1280 \times 720 \times 30 \times 1.5 = 40 \text{Mbytes/sec VS 1Gbytes/sec}$+
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.
## NXP 6 Series VPU: **Decoder**

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

<table>
<thead>
<tr>
<th>HW Encoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Performance (2D or 3D)</th>
<th>Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H.264</td>
<td>BP</td>
<td>1080p@30fps, 720p@60fps</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>MJPEG</td>
<td>Baseline</td>
<td>8k x 8k</td>
<td>160Mpel/s</td>
</tr>
<tr>
<td></td>
<td>MPEG4</td>
<td>Simple</td>
<td>720p@30fps</td>
<td>15Mbps</td>
</tr>
<tr>
<td></td>
<td>H.263</td>
<td>P0/P3</td>
<td>720p@30fps</td>
<td>15Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264-MVC for 3D</td>
<td>Stereo HP (no interview prediction)</td>
<td>720p@30fps each view, 1080p@24fps each view (TBD)</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>Simulcast for 3D</td>
<td>Any VPU encoder supported profiles</td>
<td>720p@30fps each view, 1080p@24fps each view (TBD)</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>Frame-packing</td>
<td>Any VPU encoder supported profiles</td>
<td>1080p@30fps encoding → 1080p@30fps each view capture</td>
<td>20Mbps</td>
</tr>
</tbody>
</table>
# i.MX 6 Series VPU: Multi-streams

<table>
<thead>
<tr>
<th></th>
<th>Standard</th>
<th>Profile</th>
<th>Max # Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>D1@ 30fps</td>
</tr>
<tr>
<td><strong>HW Decoder</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H.264</td>
<td>BP/MP/HP</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>On2 VP8</td>
<td>--</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>VC1</td>
<td>SP/MP/AP</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>MPEG4</td>
<td>SP/ASP</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>H.263</td>
<td>P0/P3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td><strong>HW Encoder</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H.264</td>
<td>BP</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>MPEG4-SP/H.263</td>
<td>MPEG4-SP</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
In the Frame decoding flow we have a sketch at the beginning, the goal of the VPU is decoding the compressed stream in the YUV frame.
• 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

/dev/fb0: Foreground
/dev/fb1: Background
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.

If we set a colorkey = 0x0 (BLACK) for the foreground, each black pixel will show background image

/dev/fb1: Background
6 Display Composition API for Linux: Colorkey how-to

- Example

```c
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:

\[
\text{OP} = \text{IGP} \times \alpha + \text{IVP} \times (1 - \alpha) \\
\alpha = (A + \text{floor}(A/128))/256 \\
A = [0..255]
\]
6 Display Composition API for Linux: Global Alpha how-to

- Example

```c
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).

/dev/fb0: IGP
/dev/fb1: IVP
## 6 Display Composition API for Linux: Local Alpha how-to

- Example

```c
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 maintained 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;
}
```
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);
        }
        ....
    }
}
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.