Use i.MX 6 Series Processors for Video Surveillance Application

FTF-IND-F0208

Presented by: Daniele Dall'Acqua
Produced by: Mark Talbot | UK Field Application Engineer

APR. 2014
At the end of this presentation

The aim of this presentation is to provide a grounding in the hardware blocks on i.MX 6 series applications processors and to take you from some simple examples of video handling to full streaming. At the end of today you should understand:

- The multimedia data flows through the i.MX 6 series processors
- Basics of controlling a video stream in software
- How to stream video from the command line
- How to get the best performance from your i.MX 6 series processors
Agenda

• **What is a typical surveillance system?**
• Review of i.MX 6 series family of processors
  – Differences between the family members when considering video performance
  – Detailed look at the video processing
    ▪ IP Blocks
    ▪ Video Flows
• Controlling Video with software
  – Gstreamer Introduction
  – Demonstration
• Hints & Tips
• Summary
Typical Surveillance System

- Video Analytics
- Digital Video Recorder
- HDMI or LVDS
- Ethernet
- WiFi, LTE, etc
- Internet

Freescale Product Content
Agenda

• What is a typical surveillance system?

• Review of i.MX 6 series family of processors
  – Differences between the family members when considering video performance
  – Detailed look at the video processing
    ▪ IP Blocks
    ▪ Video Flows

• Controlling Video with software
  – Gstreamer Introduction
  – Demonstration

• Hints & Tips

• Summary
i.MX 6 Series Processor Overview

- **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

- Multi-core 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 HW
- OpenGL/ES 2.x 3D accelerator with OpenCL EP support and OpenVG 1.1 acceleration
- Drives high resolution video in HW
- 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

Updated from i.MX53
i.MX 6 Series Processor Family & Surveillance

i.MX6Q

- Encode 1080p30
- Decode 1080p60
- 4 ch MIPI-CSI2
- Parallel Camera
- DISPLAY 4XGA
- 533 MHz
- 64 bit DDR3 OR 2x32 bit LPDDR2
- PCIe 2.0 WiFi
- SATA 3.0
- GbE interface
- Composition GPU 600MP/s
- 3D GPU 176M Tri/s
- 2D GPU 300MP/s
- 1M L2
- Cortex-A9
- Cortex-A9
- Cortex-A9
- Cortex-A9
- Cortex-A9
i.MX6 Family & Surveillance

i.MX6DL

- Encode 1080p30
- Decode 1080p30+D1
- 3D GPU 53M Tri/s
- Composition GPU 600MP/s
- Cortex-A9
- 512K L2
- 400 MHz
- 64 bit DDR3 OR 2x32 bit LPDDR2
- PCIe 2.0 WiFi
- GbE interface

Parallel Camera
- MIPI-CSI2 2 ch

DISPLAY WXGA

External Use | 11
i.MX6 Family & Surveillance

**i.MX6S**

- **Parallel Camera**
  - MIPI-CSI2 2 ch
  - OR
  - DISPLAY WXGA

- **Encode 1080p30**
  - Cortex-A9
  - 512K L2

- **Decode 1080p30+D1**
  - 3D GPU 53M Tri/s

- **Composition GPU 600MP/s**

- **GbE interface**

- **32 bit DDR3 OR 32 bit LPDDR2**

- **400 MHz**

- **PCIe 2.0 WiFi**
Intelligent Integration of Multi-Media

**i.MX 6Dual/6Quad VPU**
- H.264 MVC 1080p60 decode
- H.264 MVC 720p60 encode
- 350mW power consumption for single video!

**i.MX 6Dual/6Quad IPU**
- Four Display support (2x MIPI-DSI, Parallel, HDMI v1.4a)
- Stereoscopic camera input
- Color adjustments and gamut mapping
- Gamma correction and contrast stretching
- Compensation for low-light conditions & backlight reduction

**i.MX 6Dual/6Quad Triple-Play Graphics**
- 3 engines: 3D, OpenVG and BLT
- 200 MT/s, 4 shaders, 3 separate engines
- High quality 3D games optimized for mobile
- Augmented reality views (real world + 3D objects)
- Advanced 3D video formats (source/depth format)

**Movie Content**

**Recording Video**

**Game Content**

**3D LCD**

**3D Television**

**Publish**

**freescale**
Video/Graphics Subsystem in i.MX 6D/Q
Multimedia Processing Chain - Implementation

- Comprehensive HW support:
- Video/graphics fully handled by IPU, VPU and GPU.
  - The CPU does not have to touch pixels

HW Accelerated
Image Processing Unit (IPU)
Image Processing Unit (IPU)

The IPU provides hardware acceleration to support the flow of video/image data through i.MX 6 series processors, from video sources (e.g., cameras) to display devices, thus reducing the loading on the CPUs.

- **Image Capture**
  Supports parallel and MIPI cameras and other video sources: BT.656 & BT.1120

- **Displays**
  Supports various display interfaces: HDMI, LVDS, MIPI DSI, 24-bit parallel

- **Image Manipulation**
  Colour Space Conversion, Resizing, Rotation

- **Image Combining**
  Alpha Blending of images, per frame or per pixel
Image Processing Unit (IPU)

Cameras

CSI (Camera Sensor I/F)

DI (Display I/F)

DC (Display Contr.)

CM (Control Module)

IRT (Image Rotator)

SMFC (Sensor Multi FIFO Ctrl.)

VDI (Video De-Interlacer)

DMFC (Display Multi FIFO Ctrl.)

IDMAC (Image DMA Controller)

Combining Frame

Graphic Overlay

Frame

Displays

IPUv3M

32-bit AHB

MCU

64-bit AXI

Memory

64-bit AXI

Image Capture

Resize

Alpha Blend & Display

Graphic Overlay
Video Processing Unit (VPU)
Video Processing Unit (VPU)

The VPU is a multi-standard video codec (encoder/decoder) capable of handling multiple streams simultaneously by time multiplexing...

**Encoding**
Supported Formats: MPEG-4 SP, H.263, H.264 BP, MJPEG Baseline

**Decoding**
Supported Formats: MPEG-2, VC-1, MPEG-4 H.263, H.264, H.264-MVC, DivX, Real Video, MJPEG Baseline, On2 VP8, AVS Jizhun

**Colour Format**
All codecs support only 4:2:0, except MJPEG codec which support 4:2:0, 4:2:2, 2:2:4, 4:4:4 and 4:0:0

**Multiple Encoding and Decoding Simultaneously**
No hardware physical limit but total throughput limit
# VPU: Decoder Specification

<table>
<thead>
<tr>
<th>HW Decoder</th>
<th>Standard/Profile</th>
<th>Performance (2D or 3D)</th>
<th>Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPEG-2 Main-High</td>
<td>1080i/p+720p@30fps, Dual 720p @30fps, Dual 1080i/p @30fps</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264 BP/MP/HP-L4.1</td>
<td>1080p+720p@30fps, Dual 720p @30fps, Dual 1080p @30fps</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>VC1 SP/MP/AP-L3</td>
<td>1080p+720p@30fps, Dual 720p @30fps, Dual 1080p @30fps</td>
<td>45Mbps</td>
</tr>
<tr>
<td></td>
<td>MPEG4 SP/ASP</td>
<td>1080p+720p@30fps, Dual 720p @30fps, Dual 1080p @30fps</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>DivX/XviD 3/4/5/6</td>
<td>1080p+720p@30fps, Dual 720p @30fps, Dual 1080p @30fps</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>AVS Jizhun</td>
<td>1080p+720p@30fps, Dual 720p @30fps, Dual 1080p @30fps</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>H.263 P0/P3</td>
<td>8k x 8k</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>MJPEG Baseline</td>
<td>720p@30fps, 1080p@30fps for iMX6 Q/D, 1080p@30fps for iMX6 D/S</td>
<td>120Mpel/s</td>
</tr>
<tr>
<td></td>
<td>On2 VP8 --</td>
<td>720p@30fps, 1080p@30fps each view</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264-MVC for 3D (FW/HW) H.264-MVC SHP</td>
<td>720p@30fps, 1080p@30fps each view</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>Simulcast for 3D Two independent streams</td>
<td>720p@30fps, 1080i/p@24fps each view</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>Frame-packing for 3D Combine two frames into one</td>
<td>1080p@30fps decode → 1080p@30fps each view playback</td>
<td>50Mbps</td>
</tr>
</tbody>
</table>

| HW Post-proc | Rotation, mirror, deblocking/deringing |

* Limit of performance – other factors (eg system loading) can affect performance.
## VPU: Encoder Specification

<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>H.264</td>
<td>BP</td>
<td>2D</td>
<td>1080p@30fps</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>720p@60fps</td>
<td></td>
</tr>
<tr>
<td>MJPEG</td>
<td>Baseline</td>
<td></td>
<td>8k x 8k</td>
<td>160Mpel/s</td>
</tr>
<tr>
<td>MPEG4</td>
<td>Simple</td>
<td>2D</td>
<td>720p@30fps</td>
<td>15Mbps</td>
</tr>
<tr>
<td>H.263</td>
<td>P0/P3</td>
<td>2D</td>
<td>720p@30fps</td>
<td>15Mbps</td>
</tr>
<tr>
<td>H.264-MVC for 3D</td>
<td>Stereo HP (no interview prediction)</td>
<td>2D</td>
<td>720p@30fps each view</td>
<td>20Mbps</td>
</tr>
<tr>
<td>Simulcast for 3D</td>
<td>Any VPU encoder supported profiles</td>
<td>2D</td>
<td>1080p@24fps each view</td>
<td>20Mbps</td>
</tr>
<tr>
<td>Frame-packing</td>
<td>Any VPU encoder supported profiles</td>
<td>3D</td>
<td>1080p@30fps encoding</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1080p@30fps each view capture</td>
<td></td>
</tr>
</tbody>
</table>

* Limit of performance – other factors (e.g., system loading) can affect performance
## VPU: Multiple Streams

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Standard</th>
<th>Profile</th>
<th>Max # Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D1@ 30fps</td>
<td>720p@ 30fps</td>
</tr>
<tr>
<td><strong>HW Decoder</strong></td>
<td>H.264</td>
<td>BP/MP/HP</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>On2 VP8</td>
<td>--</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>VC1</td>
<td>SP/MP/AP</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>MPEG4</td>
<td>SP/ASP</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>H.263</td>
<td>P0/P3</td>
<td>8</td>
</tr>
<tr>
<td><strong>HW Encoder</strong></td>
<td>H.264</td>
<td>BP</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>MPEG4-SP/H.263</td>
<td>MPEG4-SP/H.263</td>
<td>6</td>
</tr>
</tbody>
</table>

* Limit of performance – other factors (eg system loading) can affect performance
Graphics Processing Units (GPUs)
Graphics Processing Units (GPUs)

There are three independent GPUs on the i.MX 6D/Q, and two on the i.MX6 DL/S. They can be used to accelerate graphic overlays and for windowing systems eg: X11

- **High Performance OpenGL/OpenCL GPU Core**
  Vivante GC2000 @ 528MHz (600 MHz shader)

- **High Performance Composition Engine 2D Core**
  Up to 633M pixels / sec raw performance

- **High Performance Vector Graphics Core**
  264M Pixels / sec raw performance
Agenda

• What is a typical surveillance system?
• i.MX 6 Series Processor Family & Surveillance
  – Differences between the family members when considering video performance
  – Detailed look at the video processing
    ▪ IP Blocks
    ▪ Video Flows
• Controlling Video with software
  – Gstreamer Introduction
  – Demonstration
Hints & Tips
• Summary
Controlling Video

The VPU does have its own API. This can be used to control video encoding and decoding. The alternative, preferred route, is to use multi-media framework such as GStreamer.

- **Abstracts you from the hardware**
  Freescale has provides plug-ins which use the i.MX 6 series processor IP blocks: IPU, VPU, GPU

- **Generating code to encode or decode video is simpler**
  If you can generate code to decode video on a Linux PC, it is easier to migrate the code to an embedded platform

- **Custom plugins (eg: CODEC, containers) can be added**
  Custom plugins, which then can be used in a Gstreamer pipeline
GStreamer

- Gstreamer is an Open Source Multimedia Framework
  - Gstreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple Ogg/Vorbis playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing.
  - Applications can take advantage of advances in codec and filter technology transparently. Developers can add new codecs and filters by writing a simple plugin with a clean, generic interface.
  - Gstreamer is released under the LGPL. The 1.x series is API and ABI stable and supersedes the previous stable 0.10 series. Both can be installed in parallel.
Gstreamer Basics

An “element” (eg: vpudec) will either have a source or sink, or both. The sink and source pads will have a specified format for their input or output respectively.

A “pipeline” is a linked set of “elements” as shown below

A “bin” is a set of linked “elements” and can be treated as if it were an “element”
Elements have 4 states:
- GST_STATE_NULL
- GST_STATE_READY
- GST_STATE_PAUSED
- GST_STATE_PLAYING

Buffers are objects for streaming data between elements. They always flow downstream.

Events are objects sent between elements or from an application to and element. They flow upstream and downstream and can be async or sync.

Messages are objects posted by elements to be collected by applications.
GStreamer Tools

• There are a number of command-line tools useful when trying to use Gstreamer and develop your own pipelines.

  - gst-inspect provides:
    ▪ A list of available plugins on your platform
    ▪ Information on individual plugins and elements to help you construct a pipeline
    ▪ We will use this later on.

  - gst-launch
    ▪ Builds and runs basic GStreamer pipelines from the command prompt using a PIPELINE-DESCRIPTION
    ▪ In simple form, a PIPELINE-DESCRIPTION is a list of elements separated by exclamation marks (!). Properties may be appended to elements, in the form property=value.
Constructing a Very Simple Pipeline

Let's consider a pipeline example: taking a camera input and displaying it on the LVDS screen.

```
gst-launch -v mfw_v4lsrc capture_mode=5 device=/dev/video1 ! mfw_v4lsink
```

On the command line:
```
gst-launch -v mfw_v4lsrc capture_mode=5 device=/dev/video1 ! mfw_v4lsink
```

Verbose o/p

1080p
Source Code Example

/*
   ===========================================================================
   Name        : Freescale Camera Viewer
   Commandline Equiv: gst-launch mfw_v4lsrc device=/dev/video1 capture-mode=5 ! mfw_v4lsink
   Author      : Mark Talbot:
   Description : Gstreamer Example - no error handling
   ===========================================================================
*/

#include <gst/gst.h>
#include <glib.h>
#include <stdio.h>

int main (int argc,
          char *argv[])
{
    GstElement *pipeline, *source, *sink;

    /* Initialisation */
    gst_init (&argc, &argv);

    /* Check input arguments */
    if (argc != 2) {
        g_printerr ("Usage: %s <input dev: eg: /dev/video0>\n", argv[0]);
        return -1;
    }

    g_print ("Device to be used: %s \n", argv[1]);
/* Create gstreamer elements */
pipeline = gst_pipeline_new("video-player");
source  = gst_element_factory_make("mfw_v4lsrc", "video-source");
sink    = gst_element_factory_make("mfw_v4lsink", "fs1-v4lsink");

if (!pipeline || !source || !sink) {
    g_printerr("One element could not be created. Exiting.\n");
    return -1;
}

/* Set up the pipeline */

/* we set up the input filename to the source element */
g_object_set(G_OBJECT(source), "device", argv[1], "capture-mode", 4, NULL);

/* For Debug */
/* g_object_set (G_OBJECT (source), "device", "/dev/video1", "capture-mode", 5, NULL);*/

/* we add all elements into the pipeline */
/* file-source | video-output */
gst_bin_add_many (GST_BIN (pipeline),
    source, sink, NULL);

/* we link the elements together */
/* file-source -> video-output */
gst_element_link_many (source, sink, NULL);

/* Set the pipeline to "playing" state*/
g_print("Now playing: %s\n", argv[1]);
gst_element_set_state (pipeline, GST_STATE_PLAYING);

/* Iterate */
g_print("Running...\n");
Source Code Example – cont.

/* When you press enter we will do a graceful exit */
printf("**************Press ENTER to stop"\n);  
   while (1)
      {
          char c=getchar();
          if (c=='\n' || c==EOF) break;
      }

/* Out of the main loop, clean up nicely */
g_print("Returned, stopping playback\n");
gst_element_set_state (pipeline, GST_STATE_NULL);

        g_print("Deleting pipeline\n");
gst_object_unref (GST_OBJECT (pipeline));

        return 0;
    }
}
Constructing a More Complicated Pipeline

Let’s make it a little more complicated, and take a stream from the camera and encode it in mpeg4 and in an mkv container

```
flexible_pipeline src vpuenc mux sink
```

The command line to record:
```
gst-launch -v mfw_v4lsrc capture_mode=5 device=/dev/video1 ! queue ! vpuenc codec=0 ! matroskamux ! filesink location=mycamera.mkv sync=false
```

And.....

To play the recording back
```
gst-launch -v filesrc location=mycamera.mkv typefind=true ! matroskademux ! queue max-size-time=0 ! vpudec ! mfw_v4lsink
```
Useful Command-Lines

• Wouldn’t it be nice not having to know the format of the file you are trying to play?
  - playbin2
    ▪ Playbin2 is self-constructed pipeline elements which will auto connect all necessary elements to decode a media file/resource, including source, parser, decoder and sink, etc.
    ▪ `gst-launch playbin2 uri=file:///media/test.avi`

• What about pausing, resuming, changing the volume, playing a list of files etc.
  - gplay
    ▪ A command line based player. It is based on Gstreamer playbin2 element and provides full functions of playback, including trick mode, video display setting etc.
    ▪ `gplay -r /home/user/test1.avi /home/user/test2.avi`
Video Streaming and Demo
HTTP Streaming

• Freescale multimedia framework supports HTTP protocol-based streaming.
• HTTP server with test content is required eg: Apache2
• To play content:
  – gst-launch playbin2 uri=http://SERVER/test.avi
Streaming Video Using RTP and RTCP

• Definitions
  - The **Real-time Transport Protocol (RTP)** defines a standardized packet format for delivering audio and video over IP networks. RTP is used extensively in communication and entertainment systems that involve streaming media, such as telephony, video teleconference applications, television services and web-based push-to-talk features.
  - RTP is used in conjunction with the **RTP Control Protocol (RTCP)**. While RTP carries the media streams (e.g., audio and video), RTCP is used to monitor transmission statistics and quality of service (QoS) and aids synchronization of multiple streams. RTP is originated and received on even port numbers and the associated RTCP communication uses the next higher odd port number.
Demo Setup

Camera Board
Will send video from local camera to other board

DVR Board
Will display video from local camera and from remote board
A New Element for Displaying Streams

- Throughout this presentation we have used: `mfw_v4lsink`. This element allows for one stream to be displayed on one screen.
- For a DVR (Digital Video Recorder) ideally we want to display multiple streams on a single display. To do this we need to use a new element: `mfw_isink`
- Definition of `mfw_isink`: plugin for Gstreamer is a IPU lib based sink element which provides multi-overlay support of video playback
- By default, `mfw_isink` uses `fb2` as a display framebuffer on LCD. Since `fb2` is invisible, by default. So, you need to run the following command in a terminal window to enable `mfw_isink` local alpha feature when using `mfw_isink` with `gst-launch`:
  ```bash
  export VSALPHA=1
  ```
- Let’s `gst-inspect mfw_isink`
What Video Formats Can mfw_isink Support?

Pad Templates:

SINK template: 'sink'
Availability: Always
Capabilities:

video/x-raw-yuv
  format: NV12
video/x-raw-yuv
  format: I420
video/x-raw-yuv
  format: YUY2
video/x-raw-yuv
  format: Y444
video/x-raw-yuv
  format: Y42B

video/x-raw-yuv
  format: 422P
video/x-raw-yuv
  format: UYVY
video/x-raw-yuv
  format: YUYV
video/x-raw-yuv
  format: Y800
video/x-raw-gray
  bpp: 8
video/x-raw-rgb
  bpp: 16
video/x-raw-rgb
  bpp: 24
video/x-raw-rgb
  bpp: 3
How Do We Change the Size of The Video?

display : get/set the device of the output
  flags: readable, writable
  String. Default: "master" Current: null

mode : get/set the device mode of the output
  flags: readable, writable
  Integer. Range: 0 - 3 Default: 0 Current: 0

rotation : get/set the rotation of the output
  flags: readable, writable
  Integer. Range: 0 - 7 Default: 0 Current: 0

axis-left : get/set the left of the output
  flags: readable, writable
  Integer. Range: -2147483648 - 2147483647 Default: 0 Current: 0

axis-top : get/set the right of the output
  flags: readable, writable
  Integer. Range: -2147483648 - 2147483647 Default: 0 Current: 0

disp-width : get/set the width of the device
  flags: readable, writable
  Integer. Range: 0 - 2147483647 Default: 0 Current: 0

disp-height : get/set the height of the device
  flags: readable, writable
  Integer. Range: 0 - 2147483647 Default: 0 Current: 0
Script for Receiving Video (DVR)

• First Script to Enable Cameras + Configure Display

```
load_cam_mod.sh
export VSALPHA=1           ## Ensure fb2 is visible
modprobe ov5640_camera_mipi    ## Load MIPI camera module
modprobe mxc_v4l2_capture              ## Load Capture module
echo 0 > /sys/class/graphics/fb0/blank         ##Make sure screen is on
echo -e -n "\033[9" > /dev/tty0  ##Make sure screen does not go blank during playback
```

• Receive Script

```
stream_receive.sh
## IMX1 - Receive Stream
VIDEO_CAPS=""application/x-rtp, media=(string)video, payload=(int)96, clock-rate=(int)90000, encoding-name=(string)H264"
VIDEO_DEC_SINK="rtph264depay ! vpudec ! mfw_isink axis-left=0 axis-top=0 disp-width=512 disp-height=384 "

## Set_up RTP
PLAYBACK_VIDEO="udpsrc caps=$VIDEO_CAPS port=5000 ! rtpbin.recv_rtpSink_0
rtpbin. ! $VIDEO_DEC_SINK\nudpsrc port=5001 ! rtpbin.recv_rtcp_sink_0 \nrtpbin.send_rtcp_src_0 ! udpsink port=5005 sync=false async=false"

PLAYBACK_AV="$PLAYBACK_VIDEO"

# Playback pipeline
gst-launch -v gstrtpbin name=rtpbin $PLAYBACK_AV
```

Setting output to quarter of LVDS
Script for Sending Video (Camera)

• First Script to Enable Cameras:
  ```
  load_camera_module.sh
  modprobe ov5640_camera_mipi
  modprobe ov5642_camera
  modprobe mxc_v4l2_capture
  ```

• Stream script
  ```
  stream_send.sh
  ## IMX Send Stream
  IP=192.168.0.4 # IP address of the playback machine
  VIDEO_SRC="mfw_v4lsrc capture_mode=3 device=/dev/video1"
  VIDEO_ENC="vpuenc codec=6 ! rtph264pay"

  ## Set up rtp
  STREAM_VIDEO="$VIDEO_SRC ! $VIDEO_ENC ! rtpbin.send_rtp_sink_0 !
  rtpbin.send_rtp_src_0 ! queue ! udpsink host=$IP port=5000 !
  rtpbin.send_rtcp_src_0 ! udpsink host=$IP port=5001 sync=false async=false!
  udpsrc port=5005 ! rtpbin.recv_rtcp_sink_0"

  STREAM_AV="$STREAM_VIDEO"

  # Stream pipeline

  gst-launch -v gstrtpbin name=rtpbin $STREAM_AV
  ```
The Video Streaming Demo

- On the DVR (receiver board) we are running an executable (FTFgst) that runs two instances of the previous script and does two local play backs.
- The FTFgst also shows a very simple overlay to give some idea how a graphics overlay could be implemented.
- Demo starting order
  - Receiver board
    - Run: load_cam_mod.sh
    - Run: FTFgst
  - Sending board
    - Run: load_camera_module.sh
    - Run first script in bg: stream_send.sh &
    - Run next script either:
      - 2nd_stream_send.sh (Test source)
      - 3rd_stream_send.sh (Second SDP camera)
Demo screen

Very Simple Graphic Overlays

Fake local Video Source
Local Camera Video Source
1st Remote Camera Video Source
2nd Remote Camera Video Source
Hints & Tips
• What is a typical surveillance system?
• i.MX 6 Series Processor Family & Surveillance
  – Differences between the family members when considering video performance
  – Detailed look at the video processing
    ▪ IP Blocks
    ▪ Video Flows
• Controlling Video with software
  – Gstreamer Introduction
  – Demonstration
• Hints & Tips
• Summary
Hints & Tips for Handling Multiple Video Streams

• By default the clock to the VPU is setup to 266MHz, if you want to do 1080p60 you will need to change this to 352MHz
  - In kernel configuration select:
    ▪ Device Drivers → MXC support drivers → MXC VPU → MX6 VPU 352M

• Disable the DVFS
  - echo performance >
    /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

• Unblank a screen
  - echo 0 > /sys/class/graphics/fb<x>/blank
    ▪ Where <x> = framebuffer number
Hints & Tips for Handling Multiple Video Streams – cont.

• Playing more than 4 video files at the same time
  – The BSP has a hard coded limit lefts in from the i.MX53 processor, which had a hardware limit of 4 videos. This can be changed to what you want
    ▪ In file: mfw_gst_vss_common.c the limit is set to 4.
      • vd->vsmax = 4
    ▪ Also change in vss/mfw_gst_vss_common.h which limits number of streams to 8
      • #define VS_MAX 8

• As you increase the number of videos being decoded you will run out of frame buffers
  – frame-plus=1 option in vpudec to reduce number of buffers

• You will also need to increase the DMA zone memory
Summary

- Reviewed the IP blocks with the i.MX 6 series processors, which make it ideal for video surveillance applications
- Understood how the streams flow through the device
- Looked Linux software tools available to control video from the command line and reviewed a source code example
- Constructed a RTP/RTCP script and seen a demo of streaming video and displaying multiple streams on a single display
- Discussed changes to the BSP which can be made to optimize performance
Back-up Slide: Source Code for Very_Simple_Video_Player

```c
/*
 * Name: Freescale Very Simple Video Player
 * Author: Mark Talbot
 * Copyright: Freescale example
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static gboolean
bus_call (GDBus *bus, gchar *msg, gpointer data)
{
    Mainloop *loop = (Mainloop *) data;
    switch (msg_message_type (msg))
    {
        case DBUS_MESSAGE_METHOD_CALL:
            g_print ("%s", msg_message_name (msg));
            g_message_loop_quit (loop);
            return TRUE;
        case DBUS_MESSAGE_ERROR:
            g_message_error_handler (msg, error, debug);
            g_message_free (msg);
            g_printerr ("%s", error_message (error));
            g_message_free (error);
            g_message_loop_quit (loop);
            break;
        default:
            break;
    }
    return TRUE;
}

static void
on_pad_added (GstElement *element, gchar *pad, gpointer data)
{
    sinkpad = get_element_pad (element, "sink");
    sinkpad = get_element_pad (element, "sink");
    g_message_error_handler (msg, error, debug);
    g_message_free (msg);
    g_printerr ("%s", error_message (error));
    g_message_free (error);
    g_message_loop_quit (loop);
    break;
}
```
Display Processor (DP)

- DP has following features:
  - Support input format YUVA/RGBA
  - Combining 2 video/graphics planes
  - Color conversion (YUV <-> RGB, YUV<->YUV) & Correction (gamut-mapping)
  - Gamma correction and Contrast stretching
  - Support output format YUV/RGB
  - Dynamic task switching between async and sync flows
Display Interfaces for i.MX 6D/Q Processors

- **Six ports**
  - Two parallel - driven directly by the IPU
  - Two LVDS channels - driven by the LVDS bridge
  - One HDMI – driven by the HDMI transmitter
  - One MIPI-DSI – driven by the MIPI-DSI transmitter

- **Four simultaneous outputs**
  - Each IPU has two display ports (DI0 and DI1)
  - Therefore, up to four external ports can be active at any given time.
  - Additional asynchronous data flows can be sent through the parallel ports and the MIPI-DSI port

- **Display Content Integrity Check (DCIC)**
  - For parallel interfaces: probes the I/O loopback (essentially equivalent to probing the external wires)
  - For other integrated interfaces (e.g. LVDS): probes the IPU output (essentially equivalent to the inputs to the serializers)
Display Interfaces for i.MX 6DL/Solo Processors

- **Five ports**
  - Two parallel - driven directly by the IPU or EPDC (only one port)
  - One LVDS channels - driven by the LVDS bridge
  - One HDMI – driven by the HDMI transmitter
  - One MIPI-DSI – driven by the MIPI-DSI transmitter

- **Up to 3 simultaneous outputs with EPDC**
  - EPDC with two IPU driven outputs
  - Only two simultaneous outputs driven by the IPU (Parallel (x2), LVDS, HDMI, MIPI-DSI)

- **Display Content Integrity Check (DCIC)**
  - For parallel interfaces: probes the I/O loopback (essentially equivalent to probing the external wires)
  - For other integrated interfaces (e.g. LVDS): probes the IPU output (essentially equivalent to the inputs to the serializers)
Camera Interface

Video Sources

Parallel 0
MIPI/CSI-2
Parallel 1

i.MX 6Dual/6Quad

MIPI/CSI-2 Receiver
Bridge/gasket

mux
mux

CSI0
IPU #0
CSI1

CSI1
IPU #1
CSI0
Maximum Display Resolutions

- MIPI DSI, 2 lanes
  - WXGA (1366 x 768) or 720p (1280 x 720)

- RGB
  - Port 1 – 4XGA (2048 x 1536)
  - Port 2 – 4XGA (2048 x 1536)

- LVDS
  - Single channel – WXGA (1366 x 768) or 720p (1280 x 720)
  - Dual channel – UXGA (1600 x 1200) or 1080p (1920 x 1080)

- HDMI
  - 1080p (1920 x 1080) or 4XGA (2048 x 1536)

Note: Assuming 30% blanking intervals overhead, 24bpp, 60fps