Video/Image Codec and Data Pipeline

FTF-CON-F0165

Jones He | System & Architecture

April 2, 2014
Agenda

• Video/Image/Graphics System in iMX6
• VPU Performance/Capability Overview
• Measured Performance in BSP4.1.0
• VPU Architecture Overview
• VPU Programmable Engine
• VPU Decoder
• VPU Encoder
• Tile Format Support
• JPEG Processing Unit (JPU)
• VPU Software Structure
• Stereo 3D
• VPU with Multimedia Framework
• VPU encode/decode with IPU pre-/post-process
• Use-case Demos
Video/Graphics System in i.MX6 Dual/Quad (D/Q)

• Outline
  – Video/Graphics Subsystem in i.MX6 D/Q
  – VPU-IPU-GPU Dataflow
Video/Graphics Subsystem in i.MX6 D/Q
VPU Performance/Capability Overview

• Outline
  – Performance & capability for decoder
  – Performance & capability for encoder and simultaneous encode/decode
  – Performance & capability for multi-streams (decodes, encodes)
  – Performance & capability for transcoding
  – i.MX6x VPU vs i.MX53 VPU
# Performance Overview—iMX6Q/D *(Decoder)*

<table>
<thead>
<tr>
<th>HW Decoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Performance (DDR=532MHz)</th>
<th>Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPEG-2</td>
<td>Main-High</td>
<td>1080i/p+720p@30fps, or</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264</td>
<td>BP/MP/HP-L4.1</td>
<td>2x 1080p@24fps, or</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>VC1</td>
<td>SP/MP/AP-L3</td>
<td>2x 1080p@30fps (VPU=352MHz)</td>
<td>45Mbps</td>
</tr>
<tr>
<td></td>
<td>MPEG4</td>
<td>SP/ASP</td>
<td>1080p+720p@30fps, or 2x 1080p@24fps, or 2x 1080p@30fps (VPU=352MHz)</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>DivX/XviD</td>
<td>3/4/5/6</td>
<td>1080p+720p@30fps, or 2x 1080p@24fps, or 2x 1080p@30fps (VPU=352MHz)</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>AVS</td>
<td>Jizhun</td>
<td>1080p+720p@30fps, or 2x 1080p@24fps, or 2x 1080p@30fps (VPU=352MHz)</td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>H.263</td>
<td>P0/P3</td>
<td>8k x 8k</td>
<td>120Mpel/s</td>
</tr>
<tr>
<td></td>
<td>RV10</td>
<td>8/9/10</td>
<td>1080p@30fps (VPU=352MHz)</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>Sorenson</td>
<td>--</td>
<td></td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>MJPEG</td>
<td>Baseline</td>
<td></td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>On2 VP8</td>
<td>--</td>
<td></td>
<td>40Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264-MVC for 3D (FW/HW)</td>
<td>H.264-MVC SHP</td>
<td>720p@30fps each view 1080i/p@24fps each view 1080p@30fps (VPU=352MHz)</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>Simulcast for 3D</td>
<td>Two independent streams</td>
<td>720p@60fps (30fps each view) 1080i/p@24fps each view 1080p@30fps (VPU=352MHz)</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>Frame-packing for 3D</td>
<td>Combine two frames into one</td>
<td>1080p@30fps decode → 1080p@60fps playback (30fps each view)</td>
<td>50Mbps</td>
</tr>
</tbody>
</table>

**HW Post-proc**
rotation, mirror, deblocking/deringing
## Performance Overview—iMX6DL/S (Decoder)

<table>
<thead>
<tr>
<th>HW Decoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Performance (DDR=400MHz) (VPU=266MHz if not specified)</th>
<th>Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPEG-2</td>
<td>Main-High</td>
<td>1080i/p+D1@30fps, 1080i/p+720p@30fps (content depend.) Dual 1080p@24fps, (Content depend.)</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264</td>
<td>BP/MP/HP-L4.1</td>
<td>50Mbps</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>VC1</td>
<td>SP/MP/AP-L3</td>
<td>45Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPEG4</td>
<td>SP/ASP</td>
<td>40Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DivX/XviD</td>
<td>3/4/5/6</td>
<td>40Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AVS</td>
<td>Jizhun</td>
<td>40Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H.263</td>
<td>P0/P3</td>
<td>20Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RV10</td>
<td>8/9/10</td>
<td>40Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sorenson</td>
<td>--</td>
<td>40Mbps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MJPEG</td>
<td>Baseline</td>
<td>8k x 8k</td>
<td>120Mpel/s</td>
</tr>
<tr>
<td></td>
<td>On2 VP8</td>
<td>--</td>
<td>1080p@30fps</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td>H.264-MVC for 3D (FW/HW)</td>
<td>H.264-MVC SHP</td>
<td>720p@30fps each view 1080i/p@24fps each view (content depend.)</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>Simulcast for 3D</td>
<td>Two independent streams</td>
<td>720p@60fps (30fps each view) 1080i/p@24fps each view (content depend.)</td>
<td>50Mbps</td>
</tr>
<tr>
<td></td>
<td>Frame-packing for 3D</td>
<td>Combine two frames into one</td>
<td>1080p@30fps decode → 1080p@60fps playback (30fps each view)</td>
<td>50Mbps</td>
</tr>
<tr>
<td>HW Post-proc</td>
<td></td>
<td></td>
<td>rotation, mirror, deblocking/deringing</td>
<td></td>
</tr>
</tbody>
</table>
## Performance Overview—iMX6Q/D (Encoder & Full-duplex)

<table>
<thead>
<tr>
<th>HW Encoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Performance (VPU=266MHz if not specified)</th>
<th>Bitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264</td>
<td>BP</td>
<td>1080p@30fps</td>
<td></td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>720p@60fps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MJPEG</td>
<td>Baseline</td>
<td>8k x 8k</td>
<td></td>
<td>160Mpel/s</td>
</tr>
<tr>
<td>MPEG4</td>
<td>Simple</td>
<td>720p@30fps (1080p@30fps is doable)</td>
<td></td>
<td>15Mbps</td>
</tr>
<tr>
<td>H.263</td>
<td>P0/P3</td>
<td>720p@30fps (1080p@30fps is doable)</td>
<td></td>
<td>15Mbps</td>
</tr>
<tr>
<td>H.264-MVC</td>
<td>Stereo HP (no interview prediction)</td>
<td>720p@60fps</td>
<td></td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1080p@48fps (24fps/view, VPU=352MHz)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulcast for 3D</td>
<td>All VPU encoder supported profiles</td>
<td>720p@60fps</td>
<td></td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1080p@48fps (24fps/view, VPU=352MHz)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame-packing</td>
<td>All VPU encoder supported profiles</td>
<td>1080p@30fps encoding → 1080p@60fps capture (30fps for each view)</td>
<td></td>
<td>20Mbps</td>
</tr>
<tr>
<td>Full-duplex HW Codec</td>
<td>H.264</td>
<td>BP</td>
<td>720p@30fps</td>
<td>20Mbps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1080p@24fps</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1080p@30fps (VPU = 352MHz)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPEG4</td>
<td>Simple</td>
<td>720p@30fps</td>
<td></td>
<td>15Mbps</td>
</tr>
<tr>
<td>H.263</td>
<td>P0/P3</td>
<td>720p@30fps</td>
<td></td>
<td>15Mbps</td>
</tr>
</tbody>
</table>
# Performance Overview—i.MX6Q/D (*Multi-streams*)

<table>
<thead>
<tr>
<th>HW Decoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Max # Streams @ 30fps</th>
<th>D1@30fps</th>
<th>720p@30fps</th>
<th>1080p@24fps</th>
<th>1080p@30fps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H.264</td>
<td>BP/MP/HP</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2 (VPU &gt;300MHz)</td>
</tr>
<tr>
<td></td>
<td>On2 VP8</td>
<td>--</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>VC1</td>
<td>SP/MP/AP</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2 (VPU=352MHz)</td>
</tr>
<tr>
<td></td>
<td>MPEG4</td>
<td>SP/ASP</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2 (VPU=352MHz)</td>
</tr>
<tr>
<td></td>
<td>H.263</td>
<td>P0/P3</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2 (VPU=352MHz)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HW Encoder</th>
<th>Standard</th>
<th>Profile</th>
<th>Max # Streams @ 30fps</th>
<th>D1@30fps</th>
<th>720p@30fps</th>
<th>1080p@30fps</th>
<th>1080p@24fps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H.264</td>
<td>BP</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td></td>
<td>2 (VPU=352MHz)</td>
</tr>
<tr>
<td></td>
<td>MPEG4-SH263</td>
<td>MPEG4-SP</td>
<td>6</td>
<td>2</td>
<td>1*</td>
<td></td>
<td>2* (VPU=352MHz)</td>
</tr>
<tr>
<td></td>
<td>H.263-P0/P3</td>
<td>H.263-P0/P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*: MPEG4/H263 108 0fps is doable in HW but may not enabled in SW
## Performance Overview—iMX6Q/D (Transcoding)

<table>
<thead>
<tr>
<th>Source Resolution (decoded streaming)</th>
<th>Max # Streams @ 30fps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Target Resolution (encoded streaming)</td>
</tr>
<tr>
<td></td>
<td>SD (720x480)</td>
</tr>
<tr>
<td>SD</td>
<td>4</td>
</tr>
<tr>
<td>HD720p</td>
<td>2</td>
</tr>
<tr>
<td>HD1080p</td>
<td>1</td>
</tr>
</tbody>
</table>
## i.MX6x VPU vs i.MX53 VPU

<table>
<thead>
<tr>
<th>Enhancements</th>
<th>i.MX53</th>
<th>i.MX6x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock rate</td>
<td>200MHz</td>
<td>266MHz (Will change the VPU spec to 350MHz)</td>
</tr>
<tr>
<td>Video Decoder Perf.</td>
<td>1080i/p@30fps, No 3D support</td>
<td>1080i/p@60fps, 3D support at 30fps per view</td>
</tr>
<tr>
<td>Video Encoder Perf.</td>
<td>720p@30fps</td>
<td>1080p@30fps</td>
</tr>
<tr>
<td>VP8 decoder</td>
<td>No</td>
<td>Supported</td>
</tr>
<tr>
<td>AVS decoder</td>
<td>No</td>
<td>Supported</td>
</tr>
<tr>
<td>Theora decoder</td>
<td>No</td>
<td>Partial HW support (no plan to enable it yet)</td>
</tr>
<tr>
<td>JPEG Decoder Performance</td>
<td>40Mbps/sec at YUV444 format (400, 420, 422, 444)</td>
<td>120Mbps/sec at YUV444 format (400, 420, 422, 444)</td>
</tr>
<tr>
<td>JPEG Encoder Performance</td>
<td>80Mbps/sec at YUV422 format (400, 420, 422)</td>
<td>160Mbps/sec at YUV444 format (400, 420, 422, 444)</td>
</tr>
<tr>
<td>Decoding of H.264-MVC S3D and other S3D streams</td>
<td>No</td>
<td>Supported at 1080i/p@30fps for each view</td>
</tr>
<tr>
<td>Encoding of H.264 MVC S3D video</td>
<td>No</td>
<td>Support ed at 720p@30fps for each view (no interview prediction)</td>
</tr>
<tr>
<td>Tiled format</td>
<td>No</td>
<td>Supported (for bandwidth reduction)</td>
</tr>
<tr>
<td>2D cache</td>
<td>No</td>
<td>Cache used for bandwidth reduction for encoder (motion estimation) and decoder (motion compensation)</td>
</tr>
</tbody>
</table>
Measured Performance

• Outline
  – Measured performance for video playback
  – Measured performance for video encoder
  – Measured performance for transcoding
# Measured Performance (Decoder)

<table>
<thead>
<tr>
<th>Video clips</th>
<th>Video content complexity</th>
<th>Measured perf. at 264MHz (linear format)</th>
<th>Measured perf. at 264MHz (tiled format)</th>
<th>Measured perf. at 352MHz (tiled format)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Actual Display: 40fps</td>
<td>Actual Display: 56fps</td>
<td>Actual Display: 60fps</td>
</tr>
<tr>
<td>A Blu-ray clip</td>
<td>H264-HP, 37Mbps, 1080p</td>
<td>Effective Dec: 56fps</td>
<td>Effective Dec: 59fps</td>
<td>Effective Dec: 74ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual Display: 55fps</td>
<td>Actual Display: 59fps</td>
<td>Actual Display: 60fps</td>
</tr>
<tr>
<td>Avatar (Youtube)</td>
<td>H264-HP, 3.5Mbps, 1080p</td>
<td>Effective Dec: 77fps</td>
<td>Effective Dec: 80fps</td>
<td>Effective Dec: 100fps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual Display: 60fps</td>
<td>Actual Display: 60fps</td>
<td>Actual Display: 60fps</td>
</tr>
<tr>
<td>Sherlock (A movie Trailer)</td>
<td>H264-HP, 11Mbps, 1080p</td>
<td>Effective Dec: 64fps</td>
<td>Effective Dec: 70fps</td>
<td>Effective Dec: 86fps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual Display: 59fps</td>
<td>Actual Display: 60fps</td>
<td>Actual Display: 60fps</td>
</tr>
<tr>
<td>A Freescale demo clip</td>
<td>H264-HP, 10Mbps, 1080p</td>
<td>Effective Dec: 64fps</td>
<td>Effective Dec: 73fps</td>
<td>Effective Dec: 89fps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual Display: 59fps</td>
<td>Actual Display: 60fps</td>
<td>Actual Display: 60fps</td>
</tr>
<tr>
<td>Parkrun (A 1080i test clip)</td>
<td>H264-HP, 20Mbps, 1080i</td>
<td>Effective Dec: 38fps</td>
<td>Effective Dec: 52fps</td>
<td>Effective Dec:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual Display: 37fps</td>
<td>Actual Display: 45fps</td>
<td>Actual Display:</td>
</tr>
</tbody>
</table>

**Note:**
- Performance measured in VPU unit test (without multimedia framework).
- SabreSD board
- Tile format used
- VPU = 264 MHz, 352MHz
- DDR = 532 MHz
- Performance for Interlaced video not measured yet
### Measured Busload (Decoder)

<table>
<thead>
<tr>
<th>Video clips</th>
<th>Video content complexity</th>
<th>Measured bandwidth at 264MHz (linear format) at 30fps display rate</th>
<th>Measured bandwidth at 264MHz (tiled format) at 30fps display rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunflower (self-generated Blu-ray quality clip)</td>
<td>H264-HP, 40Mbps, 1080p</td>
<td>Total bus load: ~77% (53 dec fps) Bus utilization efficiency: ~17.5%</td>
<td>Total bus load: ~67% (59 dec fps) Bus utilization efficiency: ~19%</td>
</tr>
<tr>
<td>A Blu-ray quality clip</td>
<td>H264-HP, 37Mbps, 1080p</td>
<td>Total bus load: ~66% (56 dec fps) Bus utilization efficiency: ~17%</td>
<td>Total bus load: ~65% (60 dec fps) Bus utilization efficiency: 19%</td>
</tr>
<tr>
<td>Avatar (Youtube)</td>
<td>H264-HP, 3.5Mbps, 1080p</td>
<td>Total bus load: ~55% (76 dec fps) Bus utilization efficiency: ~16%</td>
<td>Total bus load: ~62% (80 dec fps) Bus utilization efficiency: 18%</td>
</tr>
<tr>
<td>Sherlock (A movie Trailer)</td>
<td>H264-HP, 11Mbps, 1080p</td>
<td>Total bus load: ~60% (60 dec fps) Bus utilization efficiency: ~16%</td>
<td>Total bus load: ~62% (69 dec fps) Bus utilization efficiency: 18%</td>
</tr>
<tr>
<td>A Freescale demo clip</td>
<td>H264-HP, 10Mbps, 1080p</td>
<td>Total bus load: ~58% (66 dec fps) Bus utilization efficiency: ~16%</td>
<td>Total bus load: ~63% (74 dec fps) Bus utilization efficiency: 18%</td>
</tr>
<tr>
<td>Parkrun (A 1080i test clip)</td>
<td>H264-HP, 20Mbps, 1080i</td>
<td>Total bus load: ~83% (57 dec fps) Bus utilization efficiency: ~17%</td>
<td>Total bus load: ~80% (57 dec fps) Bus utilization efficiency: 17%</td>
</tr>
</tbody>
</table>

**Note:**
- Performance measured in VPU unit test (without multimedia framework)
- SabreSD board
- Tile format used
- VPU = 264 MHz
- DDR = 532 MHz
- Performance for Interlaced video not measured yet
Measured Performance (Encoder)

<table>
<thead>
<tr>
<th>Video clips (250 frames)</th>
<th>Bitrate</th>
<th>Maximum measured perf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riverbed</td>
<td>H264-BP, 32Mbps, 1080p@30fps</td>
<td>40fps (VPU=264MHz), 50fps (VPU=352MHz)</td>
</tr>
<tr>
<td>Riverbed</td>
<td>H264-BP, 20Mbps, 1080p@30fps</td>
<td>41fps (VPU=264MHz), 51fps (VPU=352MHz)</td>
</tr>
<tr>
<td>Riverbed</td>
<td>H264-BP, 10Mbps, 1080p@30fps</td>
<td>42fps (VPU=264MHz), 52fps (VPU=352MHz)</td>
</tr>
</tbody>
</table>

Note:
- Performance measured in VPU unit test (without multimedia framework).
- SabreSD board
- Tile format used
- VPU = 264 MHz, 352MHz
- DDR = 532 MHz
- YUV source data in file in SD card (file reading time not count in performance)
## Measured Performance and Busload (Transcoding)

<table>
<thead>
<tr>
<th>Transcoding</th>
<th>Transcoding description</th>
<th>Maximum measured performance</th>
<th>Observed bus-load</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2→H264</td>
<td>Tiled format; Decode 1080p MPEG2 video, and display it via HDMI without resizing, and meanwhile re-encode to H264-BP, 1080p.</td>
<td>27fps (VPU=266MHz) 35fps (VPU=352MHz) Tiled format;</td>
<td>~72% (with ~20% for GUI and others) ~78% (with ~20% for GUI and others)</td>
</tr>
<tr>
<td>MPEG2→H264</td>
<td>Linear format; Decode 1080p MPEG2 video, and display it via HDMI with resizing to 720p, and meanwhile re-encode to H264-BP, 720p.</td>
<td>22fps (VPU=266MHz) 24fps (VPU=352MHz) Linear format</td>
<td>~82% (with ~20% for GUI and others) ~84% (with ~20% for GUI and others)</td>
</tr>
<tr>
<td>MPEG4→H264 VC1→H264</td>
<td>Decode 720p MPEG4/VC1, etc, and display it via HDMI without resizing, and meanwhile re-encode to H264-BP, 720p, in &lt;20Mbps</td>
<td>&gt;=50fps(VPU=266MHz) &gt;= 60fps(VPU=352MHz)</td>
<td>-Not too much improvement for VPU=352MHz</td>
</tr>
</tbody>
</table>

### Note:
- Performance measured in VPU unit test (without multimedia framework).
- SabreSD board
- Linear format used (tile format not ready for transcoding)
- VPU = 264 MHz, 352MHz
- DDR = 532 MHz
- Measured in tiled format in BSP 4.1.0
VPU Architecture Overview

• Outline
  – Video coding standard algorithm and process
  – Architecture in top-level view
  – Architecture in software view
  – VPU-IPU interface
Top view of video encoding/decoding algorithm
(An H.264 example)
VPU Architecture Overview (Top-level view)

- **Flexible and optimized area-power accelerator architecture**
  - Embedded DSP core providing a certain level of flexibility & programmability (e.g., be able to support H.264-MVC-3D by only firmware change and currently being done in i.MX6x VPU)
  - Shared logic & SRAM for encoder and decoder for optimizing area and minimizing power with clock gating (vs separate enc and dec HW in other vendors)
  - On-chip RAM with secondary AXI option for reducing memory bandwidth (makes it competitive in performance).

- Freescale drives this IP development in terms of roadmap, API’s and Firmware.
- This is our 6th instantiation of this IP from our vendor (mature technology).
- Very flexible solution that allows us to customize the feature set of each product to the market requirements without compromising power or die size.
- Freescale works closely with our VPU vendor to optimally integrate VPU into all i.MX devices.
VPU Architecture Overview (SW view)

- **Host Application**
  - API Calls with Args
  - Return Codes with Output Info

- **VPU API’s**
  - Interrupt
  - CMD/RSP

- **Firmware on VPU**
  - VPU Codec Library
  - VPU System Manager

- **SDRAM**
  - Shared Buffer (Bit-stream Buffers, Frame Buffers, etc.)
  - VPU Buffer (Work Buffer, u-Code Buffers, Parameter Buffers, etc.)
VPU Programmable Engine

• Outline
  - Features of the embedded DSP
  - Examples of programmability & capability
    ▪ Programmability is limited to video slice level or above
    ▪ Not encouraged down to macroblock-level for programmability except for some particular reasons.

Note:
- Normally, Freescale does not provide VPU firmware source code to customers.
- Freescale can work with VPU vendor for implementing the customer’s needs if necessary.
VPU Programmable Engine & Features

- A highly optimized DSP processor for handling bit streams in various video codecs
- Supports special instructions for bitstream packing/unpacking with variable length code and Exp-Golomb code
- Supports program memory up to 128KB address space (20KB in i.MX6x)
- Supports data memory up to 128KB address space (6KB in i.MX6x)
VPU Programmable Engine Capability & Example

• **Capability of VPU Programmable Engine**
  - In general, programmable on slice level and above (e.g., MVC codec implementation)
  - Specifically, programmability can be extended to macroblock-level for some codecs, e.g., VP8, macroblock-level encoder rate control, etc.

• **Examples implemented by only firmware change:**
  - Implemented MVC-Stereo High Profile for both encoder and decoder
  - Enhanced encoder rate control for achieving better visual quality
  - Enhanced error handling capability for robust streaming and video playback
VPU Decoder

• Outline
  – Decoder pipeline
  – Decoder API and process flow
  – Decoding operation steps
  – Major differences between i.MX6x VPU and i.MX5x VPU
Video Decoder Process Pipeline
(example of H264/RV/AVS/VP8 decoder)

- Parsing bitstream for macroblocks
- Loading reference data for inter-prediction
- Inverse transform/quantization
- Generating inter-predicted macroblock
- Intra-prediction & Reconstructing macroblocks
- Deblocking filtering
- Writing decoded macroblocks
Example VPU Decode Flow for a single instance

1. vpu_init()
2. vpu_DecOpen()
3. vpu_DecUpdateStreamBuffer()
4. vpu_DecSetEscSeqInit(handle, 1)
5. vpu_DecGetInitialInfo()
6. vpu_DecRegisterFrameBuffer()
7. vpu_DecStartOneFrame()
8. Vpu_IsBusy() -> No
9. vpu_DecGetOutputInfo() -> No
10. Process output picture
11. vpu_DecClrDisp()
12. Exit
13. vpu_DecClose()
14. vpu_UnInit

vpu_Init()
vpu_DeInit()
vpu_IsBusy(), jpu_IsBusy()
vpu_SleepWake()
vpu_GetVersionInfo();
vpu_WaitForInt()
vpu_SWReset()

IOGetPhyMem(), IOFreePhyMem()
IOGetVirtMem(), IOFreeVirtMem()
IOGetLramBase()

vpu_DecOpen();
vpu_DecClose();
vpu_DecSetEscSeqInit();
vpu_DecGetInitialInfo();
vpu_DecRegisterFrameBuffer();
vpu_DecGetBitstreamBuffer();
vpu_DecUpdateBitstreamBuffer();
vpu_DecStartOneFrame();
vpu_DecGetOutputInfo();
vpu_DecBitBufferFlush();
vpu_DecGiveCommand();
vpu_DecClrDispFlag();
Decoder Operation Steps

1. Call `vpu_Init()` to initialize the VPU
2. Open a decoder instance using `vpu_DecOpen()`
3. To provide the proper amount of bitstream, get the bitstream buffer address using `vpu_DecGetBitstreamBuffer()`
4. After transferring the decoder input stream, inform the amount of bits transferred into the bitstream buffer using `vpu_DecUpdateBitstreamBuffer()`
5. Before starting a picture decoder operation, get the crucial parameters for decoder operations such as picture size, frame rate, required frame buffer size using `vpu_DecGetInitialInfo()`
6. Using the returned frame buffer requirement, allocate the proper size of the frame buffers and convey this data to the i.MX 6Dual/Quad VPU using `vpu_DecRegisterFrameBuffer()`
7. Start a picture decoder operation picture-by-picture using `vpu_DecStartOneFrame()`
8. Wait for the completion of the picture decoder operation interrupt event
9. Check the results of the decoder operation using `vpu_DecGetOutputInfo()`
10. After displaying nth frame buffer, clear the buffer display flag using `vpu_DecClrDispFlag()`
11. If there is more bitstream to decode, go to Step 7, otherwise go to the next step
12. Terminate the sequence operation by closing the instance using `vpu_DecClose()`
13. Call `vpu_UnInit()` to release the system resources
Major API differences from iMX5x VPU

- "Streaming mode with prescan" in i.MX5x VPU is replaced by "rollback" mode in i.MX6x VPU. Reason:
  - Simplify the firmware
  - Improve the performance

- Add "MvcPicInfo" for S3D in DecOutputInfo
  ```
  typedef struct {
      int viewIdxDisplay;  //view index of display frame buffer
      int viewIdxDecoded;  //view index of decoded frame buffer
  } MvcPicInfo
  ```

- Add "AvcFpaSei" for frame-packing for S3D in DecOutputInfo.
  ```
  typedef struct {
      ..........  
      unsigned frame_packing_arrangement_id;
      unsigned frame_packing_arrangement_type
      unsigned frame_packing_arrangement_repetition_period;
      ..........  
  } AvcFpaSei
  ```
VPU Encoder

• Outline
  - Encoder pipeline
  - Encoder API and process flow
  - Encoder operation steps
  - Encoder visual quality
  - Encoder rate control concept
  - Encoder configuration example
Video Process Flow (example pipeline of H264 encoder)

- Loading a source MB to encode
- Loading reference data
- Motion estimation (1st step)
- Motion estimation (2nd step)
- Motion estimation (3rd step)
- Motion Compensation
- Intra/Inter-prediction for inv. Q/T
- Intra/Inter-prediction for Q/T
- Entropy Coding (bitstream packing)
- Write-back unfiltered data
- Write-back data
- Deblock filtering
- Intra mode decision (1st step)
- Intra mode decision (2nd step)
- Loading reference data (luma only)
- Loading reference data (Chroma only)
VPU driver API (encoder)

Example VPU Encode Flow for a single instance

vpu_init()

vpu_EncOpen()

vpu_EncGetInitialInfo()  

vpu_DecRegisterFrameBuffer()  

vpu_EncGiveCommand()  

Lack of source data?

No

Yes

Copy source data to source frame buffer

vpu_EncStartOneFrame()  

vpu_IsBusy()?

No

Yes

vpu_EncGetOutputInfo()  

Process output data

Exit

Yes

No

vpu_Init()

vpu_Deinit()

vpu_IsBusy(), jpu_IsBusy()  

vpu_SleepWake()

vpu_GetVersionInfo()  

vpu_WaitForInt()  

vpu_WaitForInt()  

vpu_SWReset()  

IOGetPhyMem(), IOFreePhyMem()  

IOGetVirtMem(), IOFreeVirtMem()  

IOGetIramBase()  

vpu_EncOpen();

vpu_EncClose();

vpu_EncGetInitialInfo();

vpu_EncRegisterFrameBuffer();

vpu_EncGetBitstreamBuffer();

vpu_EncUpdateBitstreamBuffer();

vpu_EncStartOneFrame();

vpu_EncGetOutputInfo();

vpu_EncGiveCommand();
Encoder Operation Steps

1. Call `vpu_Init()` to initialize the VPU
2. Open an encoder instance using `vpu_EncOpen()`
3. Before starting a picture encoder operation, get crucial parameters for encoder operations such as required frame buffer size using `vpu_EncGetInitialInfo()`
4. Using the returned frame buffer requirement, allocate size of frame buffers and convey this information to the VPU using `vpu_EncRegisterFrameBuffer()`
5. Generate high-level header syntaxes using `vpu_EncGiveCommand()`
6. Start picture encoder operation picture-by-picture using `vpu_EncStartOneFrame()`
7. Wait the completion of picture encoder operation interrupt event
8. After encoding a frame is complete, check the results of encoder operation using `vpu_EncGetOutputInfo()`
9. If there are more frames to encode, go to Step 4, otherwise go to the next step
10. Terminate the sequence operation by closing the instance using `vpu_EncClose()`
11. Call `vpu_UnInit()` to release the system resources
i.MX6x VPU encoder—Visual quality

- At the same frame rate, bitrate, and resolution, the visual quality of i.MX6x VPU has similar visual quality to the i.MX5x VPU.

- Visual quality can be measured as:
  - Objective such as PSNR, SSIM, etc
  - Subjective

- Encoder visual quality is determined by:
  - Rate control algorithm for CBR
  - Prediction algorithms
  - Entropy coding methods
  - Encoder configurations

- Visual quality and rate control accuracy can be improved by fine-tuning the VPU encoder rate control algorithm and parameters in firmware.
6x VPU encoder—Rate control basic concept, leaking bucket

Buffer level → QP → bits → Visual quality

Pouring in variable rate (adjusted by QP)

Buffer level

Leaking in constant rate
6x VPU encoder—
Encoder configuration example

• //General setting
  • vpu_enc->width=704 // picture width
  • vpu_enc->height=480 // picture height
  • vpu_enc->tgt_framerate=30 // frame rate
  • avc_constrainedIntraPredFlag=0 // constrained_intra_pred_flag
  • encOP->gopSize = vpu_enc->gopsize // e.g., 30, GOP picture number (0 : only first I, 1 : all I, 3 : I,P,P,I,)

• //DEBLKING FILTER
  • avc_disableDeblk = 0 // disable_deblk (0 : enable, 1 : disable, 2 : disable at slice boundary)
  • avc_deblkFilterOffsetAlpha = 0 // deblk_filter_offset_alpha (-6 ~ 6)
  • avc_deblkFilterOffsetBeta = 0 // deblk_filter_offset_beta (-6 ~ 6)
  • avc_chromaQpOffset = 0 // chroma_qp_offset (-12 ~ 12)

• //SLICE STRUCTURE
  • slicemode.sliceMode = 0 // slice mode (0 : one slice, 1 : multiple slice)
  • slicemode.sliceSizeMode = 0 // slice size mode (0 : slice bit number, 1 : slice mb number)
  • slicemode.sliceSize = 0 // slice size number (bit count or mb number)

• //RATE CONTROL
  • vpu_enc->bitrate=1024 //bit rate in kbps (ignored if rate control disable)
  • vpu_enc->encOP->initialDelay=0 // delay in ms (initial decoder buffer delay) (0 : ignore)
  • vpu_enc->encOP->vbbBufferSize=0 // VBV buffer size in bits (0 : ignore)
  • vpu_enc->encOP->rcIntraQp=40 // rcIntraQp, qp value for constant intra frame QP function,
  • vpu_enc->max_qp=45 // userQpMax, maximum qp (13 ~ 51)
  • vpu_enc->min_qp =10 // userQpMin,
  • vpu_enc->gamma=0 // encOP->userGamma, gamma value in RC (0 ~ 0.99999) x 32768
  • vpu_enc->rc_interval_mode=0 // rate control interval mode (0 - default mode, 1 - frame based, 2 - slice based, 3 - MB interval)
  • Vpuc_enc->rc_mb_interval=100 // rate control interval, This value is only valid when mode is 3

• // ERROR RESILIENCE
  • vpu_enc->intraRefresh=0 // Intra MB Refresh (0 - None, 1 ~ MbNum-1)

• //Intra mode selection
  • vpu_enc->intra16x16_mode_only // For enabling or disaling Intra4x4 mode
Tile format support

• Outline
  – Tiled format concept and benefits
  – Tiled format handling in Video Data Order Adapter (VDOA)
VPU tiled format support

- Why tiled format:
  - Much more efficient DDR access
- i.MX6x VPU supports the following tile format:
  - Linear map (Type 0)
  - Tiled macroblock raster frame map (Type 1)
  - Tiled macroblock raster field map (Type 2)
Tiled format handling in Video Data Order Adapter (VDOA)

- Tiled format handling in Video Data Order Adapter (VDOA)
  - The decoded data from VPU is stored in system memory in tiled format (each tile is 16x16 macroblock)
  - VDOA converts the tiled data (16x16 tile) into raster-scan format
  - VDOA outputs the raster-scan format data to IPU for display
JPEG Processing Unit (JPU)

• Outline
  - JPU overview and facts
  - JPU process and pipeline
  - API consideration for JPEG/MJPEG codec
JPEG Processing Unit (JPU)

- An enhanced JPEG codec with higher performance compared to i.MX5x VPU
  - A separate JPU hardware module without firmware control

- Decoding up to 120Mpixels/sec at YUV444 format
  - Support YUV4:0:0, 4:2:0, 4:2:2, and 4:4:4 formats
  - Performance will be doubled for the input format of YUV4:2:0

- Encoding up to 160Mpixels/sec at YUV444 format
  - Support YUV4:0:0, 4:2:0, 4:2:2, and 4:4:4 formats
  - Performance will be doubled for the input format of YUV4:2:0

- JPEG API consideration
  - Linux libjpeg compatible API
  - i.MX5x VPU compatible API
JPEG Processing Unit (JPU)—encoder example

JPEG decoding process is the inverse of encoding process

(1) Read multiple MCUs for 4:2:0, 4:2:2 and 4:4:4 format
(2) Quantized Coefficient memory generate non-zero flag

Getting YUV image data

Pre-processing

Transform & Quantization

Transform & Quantization
JPEG API Consideration

• Considered to support Linux libjpeg compatible API for still image decoding
  – Adding a wrapper on top of the existing VPU API
  – Difficult to fully support libjpeg API

• i.MX5x compatible API for local MJPG file playback (file-play mode) and streaming mode
VPU Software Structure

The VPU software can be divided into two parts:

1) Kernel driver: takes responsibility for system control and reserving resources (memory/IRQ). It provides an IOCTL interface for the application layer in user-space as a path to access system resources.

2) User space library: the application in user-space calls related IOCTLs and codec library functions to implement a complex codec system.
   - VPU library (e.g., libvpu.so) is located in: /usr/lib/
   - VPU firmware binary (e.g., vpu_fw_imx6q.bin) is located in: /lib/firmware/vpu/
Source Code Structure (Kernel Driver)

The table below lists the kernel space source files available in the following directories:

- `<ltib_dir>/rpm/BUILD/linux/arch/arm/plat-mxc/include/mach/`
- `<ltib_dir>/rpm/BUILD/linux/drivers/mxc/vpu/`

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxc_vpu.h</td>
<td>Header file defining IOCTLs and memory structures</td>
</tr>
<tr>
<td>mxc_vpu.c</td>
<td>Device management and file operation interface implementation</td>
</tr>
</tbody>
</table>
Source Code Structure (User space)

The table below lists the user space library source files available in the following directory:
<ltib_dir>/rpm/BUILD/imx-lib-xxxx/vpu

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpu_io.c</td>
<td>Interfaces with the kernel driver for opening the VPU device and allocating memory</td>
</tr>
<tr>
<td>vpu_io.h</td>
<td>Header file for IOCTLS</td>
</tr>
<tr>
<td>vpu_lib.c</td>
<td>Core codec implementation in user space</td>
</tr>
<tr>
<td>vpu_lib.h</td>
<td>Header file of the codec</td>
</tr>
<tr>
<td>vpu_reg.h</td>
<td>Register definition of VPU</td>
</tr>
<tr>
<td>vpu_util.c</td>
<td>File implementing common utilities used by the codec</td>
</tr>
<tr>
<td>vpu_util.h</td>
<td>Header file</td>
</tr>
</tbody>
</table>
Stereo 3D

• Outline
  - S3D coding methods
    - Simulcast method
    - Combined Frame (Frame Packing, or Frame compatible)
    - H.264-MVC S3D
## Stereo 3D Coding Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Coding method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulcast Method</td>
<td>Left view and right view coded separately in a simulcast way</td>
</tr>
<tr>
<td>Frame-packing Method</td>
<td>Combination of two views into one frame in various frame packing methods</td>
</tr>
<tr>
<td></td>
<td>- <strong>MPEG-2 Multiview profile using</strong> temporal L/R interleaving for stereo video</td>
</tr>
<tr>
<td></td>
<td>- <strong>H.264 Stereo SEI message and Frame</strong> Packing Arrangement</td>
</tr>
<tr>
<td></td>
<td>SEI message allow various methods of L/R packing (Frame Compatible S3D)</td>
</tr>
<tr>
<td></td>
<td>- Temporal interleaving</td>
</tr>
<tr>
<td></td>
<td>- spatial row/column,</td>
</tr>
<tr>
<td></td>
<td>- spatial side-by-side,</td>
</tr>
<tr>
<td></td>
<td>- Spatial up-and-bottom,</td>
</tr>
<tr>
<td></td>
<td>- checkerboard (quincunx),</td>
</tr>
<tr>
<td>H.264-MVC S3D Method</td>
<td>Coded in H.264-MVC Stereo High Profile with base view and enhanced view, with the exploitation of interview prediction</td>
</tr>
</tbody>
</table>
Simulcast Method

Left view video input → Encode

Right view video input → Encode

Bitstream left and right view

Channel

Left view output

Right view output

Decode

Decode
Frame-Packing Method

- top-bottom
- side-by-side
- checkerboard
- row-interleaving
- column-interleaving
- Temporal interleaving

1\textsuperscript{st} view
2\textsuperscript{nd} view
H.264-MVC-S3D Method

S3D can be generated using only two views, S0 and S1
VPU with Multimedia Framework

• Outline
  – Multimedia Framework
  – Supported Multimedia Format
Video playback using VPU

Video players
(Totem, WMP)

Multimedia framework (Gstreamer, Dshow)

pipeline architecture

source plug-in ➔ parser plug-in ➔ audio decoder ➔ audio sink

video decoder ➔ video sink

Video players
(Samsung Muon, Android Gallery)

Other multimedia frameworks
(OpenCORE/stageFright, Flash10, etc)

OpenMax-IL

VPU driver

VPU HW
Supported Streaming Containers

• MP4:
  - Playback: MPEG4, H.264, H.263
  - Capture, MPEG4, H.264

• AVI:
  - Playback: MPEG4, Divx/Xvid, H.264, WMV/VC1
  - Capture: MPEG4, H.264

• MPEG2-TS: Playback: MPEG2, H.264, VC1

• MPEG2-PS: Playback: MPEG2, H.264, MPEG4, AVS

• FLV: Playback: H.264, Sorenson, VP6

• ASF: Playback: WMV/VC1

• WebM: Playback: VP8

• RMVB: Playback: RV8/9/10

• Matroska (MKV): Playback: MPEG4, Divx/Xvid, H.264, WMV/VC1

• 3GP: Playback: MPEG4, H.264

• Ogg: Playback: Theora
## Streaming Protocol Support

<table>
<thead>
<tr>
<th>Protocol</th>
<th>File format</th>
<th>Supported OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>.mp4/.3gp/.mov, .flv/.f4v, .avi, .wmv/.asf, .mpg/.vob/.ts, .mp3, .aac, .wma, .mkv</td>
<td>Android, Linux</td>
</tr>
<tr>
<td>RTSP</td>
<td>.mp4</td>
<td>Android</td>
</tr>
<tr>
<td>HTTPLive</td>
<td>.m3u8</td>
<td>Android</td>
</tr>
<tr>
<td>RTP</td>
<td>.ts</td>
<td>Android</td>
</tr>
<tr>
<td>UDP</td>
<td>.ts</td>
<td>Android</td>
</tr>
</tbody>
</table>
VPU encode/decode with IPU pre-/post process

- Outline
  - The Display Ports In i.MX6 D/Q
  - Max Display Port Resolutions
  - The Video Input Ports In i.MX6 D/Q
  - IPU Internal Structure and Process Flow
The Display Ports In i.MX6 D/Q

• 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, only 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
Max Display Port 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*
The Video Input Ports In i.MX6 D/Q

• **Three ports; up to six input channels**
  - Two parallel – connected directly to the IPU
  - One MIPI/CSI-2 – connected to the MIPI/DSI receiver, can transfer up to four concurrent channels

• **Four concurrent channels**
  - Each IPU has two input ports (CSI0 and CSI1), each can process an input channel from one of the external ports.
  - The MIPI/CSI-2 bridge sends all its channels to all the IPU input ports and each port can select for processing a different channel, identified by its DI (Data Identifier).
  - Additional channels can be transferred through a CSI transparently – as generic data – directly to the system memory.

• **Formats supported:**
  - BT.656
  - BT.1120
  - BT 1358 (not validated)
  - YUV422, RGB888, YUV444 = over an 8 bit bus
  - RAW format up to 16bpp which will be translated to 8 bit using companding
  - Generic data up to 20bit
IPUv3H – Internal Structure and Process Flow

Cameras

Displays

CSI (Camera Sensor I/F) → SMFC (Sensor Multi FIFO Ctrl.)

VDI (Video De-Interlacer)

IC (Image Converter)

IDMAC (Image DMA Controller)

DMFC (Display Multi FIFO Ctrl.)

DMFC

CM (Control Module) → IRT (Image Rotator)

DI (Display I/F) → DMFC

DC (Display Contr.)

DP (Display Processor)

32-bit AHB

MCU

64-bit AXI

Memory

External Use | 60
Use-case Demos

• Outline
  – Demo for unit test (linear format vs tile format)
    ▪ Single-stream playback
    ▪ MVC-3D playback (not show real 3D, but in temporal interleaving format)
    ▪ Transcoding
  – Demo for Gstreamer (linear format vs tile format)
    ▪ Single-stream playback
    ▪ Dual-stream playback
    ▪ Transcoding
    ▪ 3D demo (with 3D TV and glasses)
#####Demo Case 1, Unit Test, Playback, VPU=264MHz#############################
#####Unit test with tiled format and display to 1080p hdmi display############
cp /home/linaro/FAE/sunflower_2B_2ref_WP_40Mbps.264 /dev/shm/tmp_video
/unit_tests/mxc_vpu_test.out -D "-i /dev/shm/tmp_video -f 2 -t 1 -a 60 -y 1"
/unit_tests/mxc_vpu_test.out -D "-i /dev/shm/tmp_video -f 2 -t 1 -a 60 -y 0"
/unit_tests/mxc_vpu_test.out -D "-i balloons_view01_3d.264 -f 2 -l 2"

#####Demo Case 2, Unit Test, Transcoding, VPU=264MHz#############################
#####Unit test with linear format and display to 264p or 1080p hdmi display#######
#####Unit test with tile format is not ready yet##################################
/unit_tests/mxc_vpu_test.out -T "-i /home/linaro/FAE/Coral_Reef_Adventure_720_video.wmv3 -f 3
-t 1 -x 0 -y 0 -a 60 -w 1280 -h 720 -o /dev/shm/transcode.264 -q 25 -g 30"
cp /home/linaro/FAE/mpeg2_1080p25_video1.mpv /dev/shm/tmp_video
/unit_tests/mxc_vpu_test.out -T "-i /dev/shm/tmp_video -f 4 -t 1 -x 0 -y 0 -a 60 -w 1920 -h 1088 -o
/dev/shm/transcode.264 -q 25 -g 30"
### Demo Case 3: Gstreamer, Single stream decoding, ~60fps VPU=264MHz
### Decoding H264 1080p@10Mbps, tiled format and display to 1080p hdmi display

```bash
sudo cp /home/linaro/FAE/Container_clips/Avatar_1920x1080_30fpsH264_2x44100AAC_3.6Mbps_246sec.mp4 /dev/shm/tmp_video
time gst-launch filesrc location=/dev/shm/tmp_video typefind=true ! aiurdemux ! vpudec output-format=4 framedrop=false ! queue max-size-buffers=2 ! mfw_v4lsink sync=false
```

### Demo Case 4: Gstreamer, Single stream decoding, <60fps VPU=264MHz
### Decoding H264 1080p@37Mbps, tiled format and display to 1080p hdmi display

```bash
sudo cp /home/linaro/FAE/Container_clips/Sherlock_1920x1080_24fpsH264_2x48000AAC_9.6Mbps_140sec.mp4 /dev/shm/tmp_video
time gst-launch filesrc location=/dev/shm/tmp_video typefind=true ! aiurdemux ! vpudec output-format=4 framedrop=false ! queue max-size-buffers=2 ! mfw_v4lsink sync=false
```

### Demo Case 5: Gstreamer, Dual-display with dual streams, 30fps VPU=264MHz
### Decoding two H264 1080p@10Mbps, linear format, 1080p and XGA display

```bash
gst-launch playbin2
uri=file:///home/linaro/FAE/Container_clips/FTF20033_1920x1080_30fpsH264_2x44100AAC_9.7mbps_137sec.mp4 flags=0x57 video-sink="mfw_v4lsink" &
gst-launch playbin2
uri=file:///home/linaro/FAE/Container_clips/Mosaic_1920x1080_H.264_10mbps_video1_repeat.mp4 flags=0x57 video-sink="mfw_v4lsink device=/dev/video18"
```
### Demo Case 6, Gstreamer, Transcoding, 27fps for VPU=264MHz  
#### 1080p-MPEG2→1080p-H264 with tiled format and display to 1080p hdmi display####

```
```

### Demo Case 7, Gstreamer, Transcoding, 43fps for VPU=264MHz  
#### 1080p-MPEG2→VGA-H264 with linear format and display to VGA hdmi display####

```
```

### Demo Case 7, 3D demo 30fps for each view, VPU=264MHz  

```
gplay FLIGHT_3D_sideBySide.mkv.mp4
```