Accelerated Video Usage Models

Case Study in Linux* Running IEGD and Intel® EMGD on Platforms Featuring the Intel® Atom™ E6XX processor and Intel® System Controller Hub US15W

May 2011
Executive Summary

Video Applications range from HD Video playback inside a movie player, a media center, or a web browser, with or without visual effects. Platforms featuring the Intel® Atom™ E6XX processor and the Intel® System Controller Hub US15W Chipset can run IEGD or Intel EMGD and make it possible to decode a HD video in the hardware, and ready the video for presentation in the various forms desired by the Linux application. Intel® Embedded Graphics driver(s), customized to the hardware, present this opportunity to the application by making efficient use of hardware resources such as the CPU, GPU, and the display controller. These driver(s) consume less power and conserve CPU/GPU bandwidth when possible.

Intel Embedded Graphics driver(s), customized to the hardware, present an opportunity to make efficient use of hardware resources, including the CPU, GPU, and the display controller. These driver(s) consume less power and conserve CPU/GPU bandwidth when possible.

This paper will present examples of innovative hardware accelerated video usage models that can be enabled using platforms featuring the Intel® Atom™ E6XX processor and the Intel® System Controller Hub US15W Chipset.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by-step guidance, application reference solutions, training, Intel’s tool loaner program, and connect with an e-help desk and the

www.intel.com/embedded/edc. §
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Challenge</td>
<td>5</td>
</tr>
<tr>
<td>Intel® Embedded Graphics Driver(s) Architecture</td>
<td>5</td>
</tr>
<tr>
<td>Accelerated Video Usage Models</td>
<td>6</td>
</tr>
<tr>
<td>Decode Step</td>
<td>6</td>
</tr>
<tr>
<td>Presentation Step</td>
<td>7</td>
</tr>
<tr>
<td>Simple Accelerated Decode and Display</td>
<td>8</td>
</tr>
<tr>
<td>Accelerated Decode and Display Using Hardware Overlay</td>
<td>9</td>
</tr>
<tr>
<td>Accelerated Decode to a Pixmap</td>
<td>11</td>
</tr>
<tr>
<td>Frame Buffer Blend with Video Overlay Plane</td>
<td>14</td>
</tr>
<tr>
<td>Accelerated Video Merged with Sub-Pictures in Display Controller</td>
<td>15</td>
</tr>
<tr>
<td>Accelerated Video Merged with Sub-Pictures in GPU</td>
<td>18</td>
</tr>
<tr>
<td>Results</td>
<td>20</td>
</tr>
<tr>
<td>Conclusion</td>
<td>20</td>
</tr>
<tr>
<td>References</td>
<td>20</td>
</tr>
</tbody>
</table>
Business Challenge

The Intel® Atom™ E6XX processor and the US15W based hardware are used in various market segments. The needs and usage models vary significantly from the Gaming, In-Vehicle Infotainment, Media Phone, Cable, Digital Signage, and Video Surveillance markets. Video playback is achieved in numerous forms, including standalone players, media center applications, web browser plug-ins, etc. Some usage models require video compositing and alpha blending; others might require 3D rendering and video post processing effects. This paper is aimed at discussing the kind of resources the Intel hardware and graphics driver(s), provides to multimedia applications in order to make the most use of system resources, and achieves the usage model the applications desire.

Intel® Embedded Graphics Driver(s) Architecture

The Intel Embedded Graphics driver(s), IEGD [1] and Intel EMGD [2], provides the applications with access to the hardware video decode engine, graphics hardware, and the display controller. The access to these capabilities is OS-dependent. In Linux, applications would write to 3D API(s), like OGL ES 2.0, to use GPU 3D rendering capability. In order to access the video decode engine, the application would write to VAAPI [3] (video acceleration API). For 2D rendering and display, the X11 [4] API(s) are used. Intel Embedded Graphics driver(s) provide backend implementation to these API(s).
Accelerated Video Usage Models

In order to play with the accelerated video, two important steps need to be followed by the application: a) Decode and b) Presentation.

**Decode Step**

During the Decode step, the application provides compressed video bit-stream and an empty VA surface to the driver through VA-API. The driver passes the frame worth of compressed bit-stream data to the decode engine. The control data and the resultant raw video frame is output into the NV12, format based, VA surface as shown in Figure 2.
Presentation Step

During the Presentation step, the application calls in the vaPutSurface() to call and pass on a drawable to the driver for rendering and/or displaying the video frame onto the screen. This step is responsible for enabling multiple usage models. A drawable can be anything that you can draw graphical data onto. In this case, we are dealing with an X11 Window or an X11 Pixmap as shown in Figure 3.
**Figure 3: Presentation Step**

```
NV12 Video Frame

vaPutSurface

X Drawable

Request to present Scaled, Color Converted, and Clipped data on surface
```

**Simple Accelerated Decode and Display**

At the Presentation step, if the drawable is a X11 window, then the driver will take the source NV12 video VA surface and render it to the X11 Window drawable. This will display it onto the screen using the 2D blit X11 API. A simple media player application, as demonstrated in Figure 4, can be created based on this usage model.
Accelerated Decode and Display Using Hardware Overlay

An exception to the previous usage model is where the driver can choose to render the video data into the hardware overlay plane, which is part of the display controller, based on the hardware availability. This causes the video to appear into the X11 window drawable, which uses color keying where the display controller will replace the color key rectangle on the X11 window drawable. This causes the video on the hardware overlay plane to peek from behind the main display plane. The use of the hardware overlay plane will
take the burden of scaling, color conversion, and blitting away from the GPU. The XVideo option should be turned on in the X-configuration file to enable the driver to use overlay. As shown in Figure 5, the use of a hardware overlay plane is not obvious. The final output will be exactly the same as that in Figure 4, but the end result is a lower GPU/CPU load that allows end users to perform 3D graphics operations more efficiently. Imagine watching HD video, without a hitch, while operating the graphical menu or changing the look and feel of the player application.
Figure 5: Accelerated Video Rendered to Hardware Overlay Plane

Accelerated Decode to a Pixmap

A “ pixmap” is an example of an off-screen drawable that graphical data can be rendered into. After drawing the data into a pixmap, the application can
choose to post process the video pixmap memory. This converts the pixmap to an OGL ES 2.0 texture using the “texture from pixmap” extension. It also adds 3D rendering, or animation and visual effects, using CPU or other GPU OGL ES 2.0 extensions. After the data has been manipulated, the video frame can be displayed back onto the screen using a X11 window. See Figure 6 for an example of an accelerated video rotated in 3D space.
Figure 6: Accelerated Video Rendered to a Pixmap, Rotated in 3D Space and Displayed

1. **Encoded Video Bitstream**
2. **Decode Video**
3. **NV 12 Video Frame**
4. **vaPutSurface**
5. **X Drawable**
6. **Texture From Pixmap**
7. **OpenGL Operation**

Accelerated Video Usage Models
Frame Buffer Blend with Video Overlay Plane

In the case where the hardware overlay plane is available, and the driver chooses to use it for video rendering/display, the application can use the display controller to alpha blend the frame buffer with the overlay data. This may result in eye-catching displays; e.g., a 3D control menu blended with video during playback. The driver provides this functionality when FBBBlendOvl and XVideo options are turned on in an X configuration file. Because of this function, the whole display plane is alpha blended with the hardware overlay plane. This feature may be useful when the accelerated video is being played back on a full screen with ARGB data. This function is specific to an instance when the ARGB data needs to be blended for the video to be on the display plane. This feature will not work well when the video is not in full screen mode. Many Linux libraries do not support maintaining the alpha channel in 2D API, leaving most of the desktop area to appear black in this mode. The 2D alpha override feature is available for some operating systems [5]. See Figure 7 for an example of a full screen video running on the first display, overlaid with a graphical control menu.
Accelerated Video Usage Models

Figure 7: Frame Buffer Overlay Blending

Accelerated Video Merged with Sub-Pictures in Display Controller

When one is optimizing for hardware acceleration, applications like Adobe Flash*, H264 based video, and graphical data (advertisements, progress bar, controls, etc.) need to be merged back together before or during display, once they are rendered out of the hardware acceleration system. The vaPutSurface() call, implemented by the Intel Embedded Graphics driver(s), provides the capability to present the decoded video frame and present a graphical ARGB based sub-picture. This sub-picture is associated with a particular video frame. This sub-picture can be bigger or smaller in resolution than the video frame. It can be created by the application using OGL ES 2.0 extensions or CPU calls. It is then passed on to the driver using VA-API sub-picture methods and is associated with a video frame. The current

---

* Adobe Flash is a trademark of Adobe Systems Incorporated.
implementation of IEGD and Intel EMGD will render and display the ARGB sub picture on the Sprite C display plane if it is available. This technique would allow for alpha blending of sub-pictures on the Sprite C plane. Depending on available hardware resources, allowing the video frame on the hardware overlay plane may be more efficient. In this case, the majority of decoding and presentation work is handled by the decode engine and the display controller; there is less of a burden on the CPU/GPU.

See Figure 8 for an example of the hardware accelerated flash video web browser plug-in, running on the first display.
Figure 8: Accelerated Video Rendered to Hardware Overlay Plane and Sub-Picture Rendered to Sprite-C Plane

- **Encoded Video Bitstream**
- **Decode Video**
- **NV12 Video Frame**
- **vaPutSurface**
- **ARGB Graphic**
- **vaAssociateSubpicture**
- **Render to Overlay Plane and draw ColorKey and associated Subpicture**
- **X Drawable**
- **Front Buffer**
- **Overlay Plane**
- **ARGB Sprite C Plane**
Accelerated Video Merged with Sub-Pictures in GPU

In the future, Intel EMGD may provide the capability to render the ARGB sub-picture and merge it with the NV12 video VA surface, then output the merged content straight to the drawable provided by the application. This will result in additional ways to blend HD video efficiently when the display controller hardware resources are not available. Figure 9 showcases how accelerated video will be blended with graphics when there are one or two displays.
Figure 9: Accelerated Video Merged with Sub-Picture Data in GPU, Rendered to a Pixmap, Rotated in 3D Space, and Displayed
Results

This paper looked at applications, codec(s), and Intel Embedded Graphics driver(s) that enable multiple usage models pertaining to the advantages of hardware accelerated decode methods. It presented video data in different forms, featuring how to use the underlying system resources efficiently.

Conclusion

The Intel® Atom™ E6XX processor and US15W based platforms provide hardware accelerated video decode. The Intel Embedded Graphics driver(s) provide the developers, ISV(s), and system integrators capabilities to present that video data in various ways. This enables multiple usage models, which include graphics and video compositing, alpha blending, video post processing, introducing 3D rendering, and visual effects on video. Along with the benefits of real time video decode and low power consumption, HD video graphics drivers intelligently use the system’s display controller resources when possible, to reduce CPU and GPU utilization. This keeps headroom for additional tasks.

Using the techniques mentioned above, Intel worked with third parties to enable Hardware Accelerated Adobe Flash Video and Hardware Accelerated Video inside the Clutter GUI Framework. Users may creatively deploy the benefits of these innovative usage models in their own environments.


References

Accelerated Video Usage Models


Authors

Abhishek Girotra is a Graphics SW TME at Intel Corporation.
Email: abhishek.girotra@intel.com

Anthony Pabon is a Sr. Software Engineer at Intel Corporation.
Email: anthony.pabon@intel.com

Acronyms

API       Application Programming Interface
ARGB     Alpha Red Green Blue
CPU      Central Processing Unit
EMGD     Embedded Media and Graphics Driver
GPU      Graphics Processing Unit
GUI      Graphical User Interface
HD       High-Definition
IEGD     Intel Embedded Graphics Driver
ISV      Independent Software Vendor
OGL ES   OpenGL for Embedded Systems
OS       Operating System
VA       Video Acceleration
VAAPI    Video Acceleration API