Application Report

Sharing IVA-HD Between VISION SDK and PSDKLA On Jacinto6 SoC

Fredy Zhang

ABSTRACT

Jacinto6 hardware supports image and video accelerator high definition (IVA - HD) to handle complex video codecs with guaranteed power and performance.

The IVA-HD accelerator is controlled from the image processing unit (IPU) to ensure real-time data processing. Codec algorithm runs on IVA-HD and operates on the frame boundary.

There are two different versions of framework-components and codec in both VISION SDK and PSDKLA. There is a conflict when you enable both frameworks to process the multimedia streams simultaneously. This application report provides a solution for sharing the IVAHD between VISION SDK and PSDKLA.

Project collateral and source code discussed in this docoument can be dowloaded from the follwing URL: https://www.ti.com/lit/zip/spracu0.

Table of Contents

List of Figures

List of Tables

Trademarks

1 IVA-HD Share Problem in Current Use Cases

More customers are considering a system that supports infotainment features plus RVC or Animation in one single Jacinto/TDA. Recently, the Ethernet based RVC has been introduced to the system which require decoding the H264 stream by real time. Meanwhile some customers require playing the video logo with startup animation.

Since only one Jacinto/TDA has one instance of IVA-HD hardware, there are two different software framework and codec drivers to controller the IVA-HD. This will encounter resource conflicts.

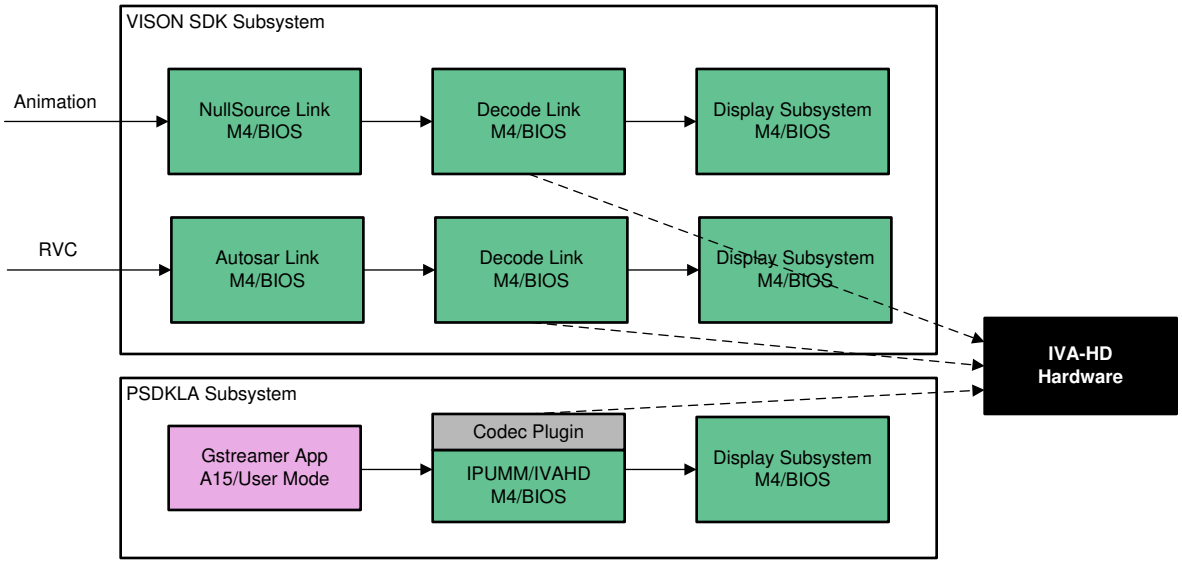

Figure 1-1. Sharing IVA-HD Problem Between VISION SDK and PSDKLA

Consider a system that supports these features as shown in Figure 1-1:

- Use case A: Animation or RVC

  For fast boot consideration requirement, most customers choose VISION SDK framework for developing RVC or Animation. VISION SDK designs with the Link and Chains concept are used to implement any use case.

  When the video data (typically, h264) reads as a video file from the boot partition by the NullSource Link or extracted as raw stream from the Ethernet frames, the NullSource Link or Autosar Link sends the streams to the decode Link for decoding. As shown in Figure 1-2, the decode link calls the driver based on the **VISION SDK codec**, which is located in the M4 core to configure the IVA-HD hardware module to implement the decoding, then sends the decoding frames to next link.

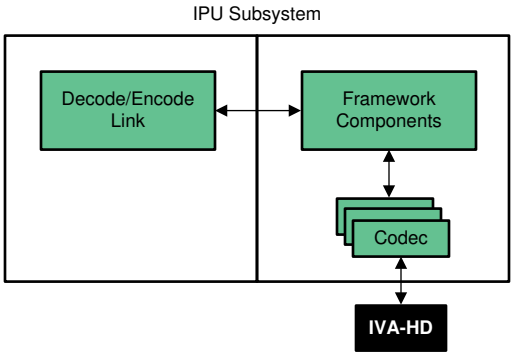

Figure 1-2. VISION SDK Codec

- Use case B: Multimedia player

  By default, TI PSDKLA uses open source gstreamer framework for any multimedia player use case. TI provides gstreamer plugins: ducatiH264 and ducaitiMJPEG. These plugins can work in gstreamer pipelines to process the multimedia file. As shown in Figure 1-3, the gstreamer calls the ducatiH264 plugin to configure IPUMM(codec) on the M4 core to communicate with IVAHD to decode the H.264 frames. The decoding plugin calls the TI IPUMM driver, which is located on the M4 core, to control the IVA-HD hardware module to implement the decoding/encoding. Then, it sends frames to the next plugin.

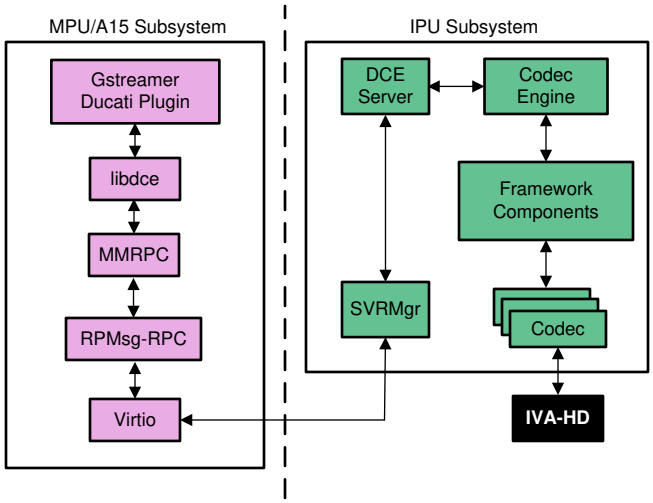

Figure 1-3. PSDKLA Codec

If the A and B use cases occurs simultaneously, one case will fail because one single IVA-HD hardware instance cannot be controlled by decode link and IPUMM on M4 at the same time. Actually, any use case that needs both the decode link on M4 and IPUMM support for decoding on A15 will fail.

2 IVA-HD Sharing Design

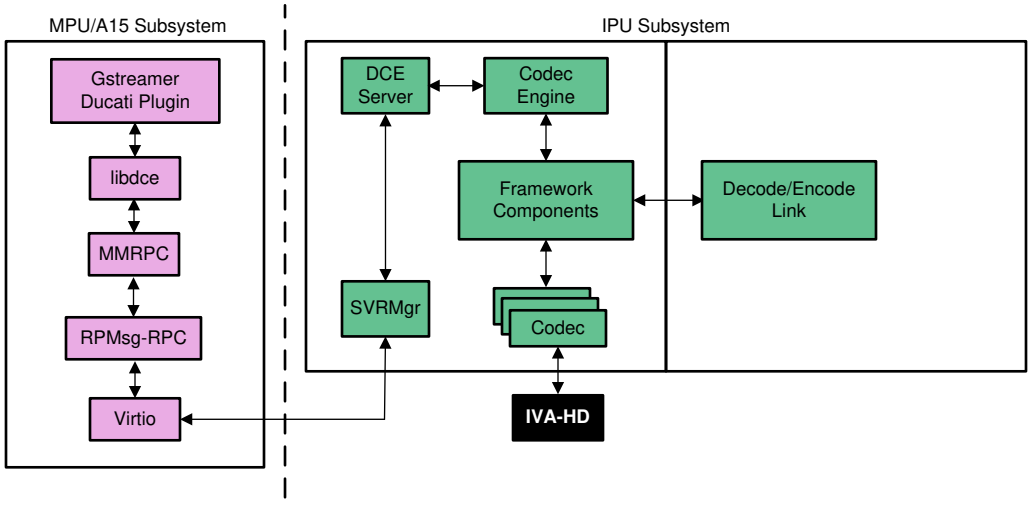

Figure 2-1. New IVA-HD Sharing Concept

Sharing the IVA-HD hardware requires that the driver can only be called in M4. Both of PSDKLA and VSION SDK have their own Framework-components and codec. They are not compatible. The better way is to use one framework and codec, and make this framework and codec compatible with another.

The IPUMM and VISION SDK decode links run on the same core. As shown in Figure 2-1, since the frame-components support multi-instance, the IVA-HD can be controlled by the two entities, both gstreamer decode plugin and decode link can call the frame-components.

- This document uses PSDKLA framework components instead of VISION-SDK framework components
- The decode link uses the IPUMM codec to replace the VISION SDK codec

3 IVA-HD Sharing Implementation

3.1 Boot Flow

For PSDKLA + VISION-SDK architecture, early boot late-attach is often used. Figure 3-1 shows the boot flow.

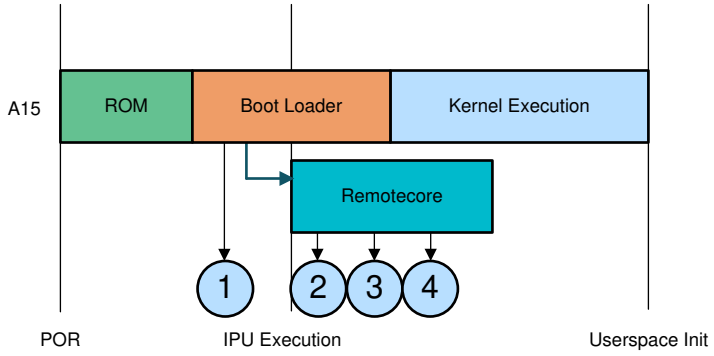

Figure 3-1. Boot Flow

For PSDKLA, IVA-HD's clock and power is controlled by kernel. For VISOIN-SDK, IVA-HD is configured by using IPU. Use the following boot flow to solve the resource conflicts.

1. IVA-HD DPLL configure in u-boot
2. Configure IPU to support IPUMM and decode link at the same time
3. IVA-HD configure
   a. Codec engine and IPUMM setup
   b. Framework components
   c. Codec
   d. IVA-HD boot parameter
4. RPMSG Driver configure

3.2 IVA-HD DPLL Configure in uboot

For generic use cases, the IVA-HD clock and power is configured in kernel. Normally in the solution, the IVAHD DPLL in u-boot is configured.

If the IVA-HD is configured in u-boot, you must remove the kernel IVA-HD configuration. If not, the kernel will reset the IVA-HD.

3.3 Configure IPU to Support IPUMM and Decode Link at the Same Time

Linux use the IPUMM to decode in PSDKLA. The decode/encode link is in VISION-SDK for RVC/Animation decoding. So, you must support IPUMM and the decode link in M4. This should be configured in cfg.mk

For example, in J6 EVM: sdk/vision_sdk/apps/configs/tda2xx_evm_linux_all/cfg.mk **IVAHD_INCLUDE** aims to support the decode/encode link. **IPUMM_INCLUDE** aims to support IPUMM. The reference code is shown below:

```
# Both IVAHD_INCLUDE & IPUMM_INCLUDE should not be set to "yes"
# Only one should be enabled to avoid IVA-HD resource conflict
IPUMM_INCLUDE=yes
IVAHD_INCLUDE=yes
```

3.4 IVA-HD Configure

Compare the driver between PSDKLA and VISION SDK as shown in Table 3-1. It shows that the framework components and code are incompatible.

Table 3-1. Components Compare

Type	PSDKLA	VISION SDK	Compatibility
Codec engine	YES	NO	/
Framework components	YES	YES	Incompatible
Codec	YES (IPUMM)	YES (1)	Incompatible

(1) VISION SDK use the codec as below:

```
ivahd_h264enc_02_00_09_01_production/
ivahd_h264vdec_02_00_17_01_production/
ivahd_hdvicp20api_01_00_00_23_production/
ivahd_jpegvdec_01_00_13_01_production/
ivahd_jpegvenc_01_00_16_01_production/
```

3.4.1 Codec Engine and IPUMM Setup

To setup the codec engine and IPUMM, see the VisionSDK_Linux_UserGuide.pdf: 2.4.2.2 Optional Components ipumm, Codec Engine and Framework components to set up the codec engine and IPUMM.

The doc is located in the below path: PROCESSOR_SDK_VISION_xx_xx_xx_xx/vision_sdk/docs/Linux/VisionSDK_Linux_UserGuide.pdf .

3.4.2 Framework Components

From the VisionSDK_Linux_UserGuide.pdf (located in the following path: PROCESSOR_SDK_VISION_xx_xx_xx_xx/vision_sdk/docs/Linux/VisionSDK_Linux_UserGuide.pdf), it is known that the framework components version packaged along with VSDK is a patched version for IVA-HD profiling and that patched version will not work with IPUMM.

The solutions use the PSDKLA framework components. This framework components will not work with VISION SDK codec. It only works with IPUMM. So, change the codec to IPUMM in the VISION SDK.

To install new framework components of PSDKLA and replace the VISION SDK version, see the VisionSDK_Linux_UserGuide located at the following path: PROCESSOR_SDK_VISION_xx_xx_xx_xx/vision_sdk/docs/Linux/VisionSDK_Linux_UserGuide.pdf..

3.4.3 Codec

PSDKLA use the IPUMM codec. But the VISION SDK use the below codec:

- ivahd_h264enc_02_00_09_01_production
- ivahd_h264vdec_02_00_17_01_production
- ivahd_hdvicp20api_01_00_00_23_production
- ivahd_jpegvdec_01_00_13_01_production
- ivahd_jpegvenc_01_00_16_01_production

In VISION SDK, IPUMM codec is used to replace the VISION SDK code. Main modification as below：

```
------------------------------ build/tools_path.mk ---------------------------

- hdvicplib_PATH                ?= $(TI_SW_ROOT)/codecs/ivahd_hdvicp20api_01_00_00_23_production
- jpegvenc_PATH                 ?= $(TI_SW_ROOT)/codecs/ivahd_jpegvenc_01_00_16_01_production
- jpegvdec_PATH                 ?= $(TI_SW_ROOT)/codecs/ivahd_jpegvdec_01_00_13_01_production
- h264venc_PATH                 ?= $(TI_SW_ROOT)/codecs/ivahd_h264enc_02_00_09_01_production
- h264vdec_PATH                 ?= $(TI_SW_ROOT)/codecs/ivahd_h264vdec_02_00_17_01_production
+hdvicplib_PATH                ?= $(mm_PATH)/extrel/ti/ivahd_codecs
+jpegvenc_PATH                 ?= $(mm_PATH)/extrel/ti/ivahd_codecs
+jpegvdec_PATH                 ?= $(mm_PATH)/extrel/ti/ivahd_codecs
+h264venc_PATH                 ?= $(mm_PATH)/extrel/ti/ivahd_codecs
+h264vdec_PATH                 ?= $(mm_PATH)/extrel/ti/ivahd_codecs
```

3.4.4 IVA-HD Boot Parameter

Based on the use case, the decode link needs to start first. So, IVA-HD was configured in M4:

```
sdk/links_fw/src/rtos/links_ipu/iva/codec_utils/src/hdvicp2_config.c:icont_boot[]
```

removing the configure from IPUMM:

```
sdk/ti_components/codecs/ipumm
--- a/src/ti/framework/dce/ivahd.c

void ivahd_init(uint32_t chipset_id)
...
-    ivahd_boot();
     DEBUG("RMAN_register() for HDVICP is successful")
```

3.5 RPMSG Startup

The two subsystems communicate with each other through RPMSG. The rpmsg driver startup from *System*ipcConnectToHLOSThread(void) *; Remove* `IpcMgr_rpmsgStartup();` *from IPUMM start.*

```
------ links_fw/src/rtos/bios_app_common/tda2ex/ipu2/src/ipu_primary.c ------static void
ipumm_startup_task(uint32_t arg0, uint32_t arg1)
...
-    IpcMgr_rpmsgStartup();
     IPUMM_Main(0, NULL);
 }
```

4 Early Decoding Demo

A demo was provided to test this use case. VISION SDK reads an h264 video data from u-boot, then sends the decode link for decoding. The use case is shown in Figure 4-1. After running Linux, you can play the video using the gstreamer.

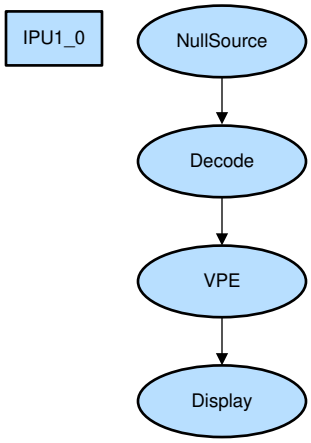

Figure 4-1. Decoding Example

PSDKLA can run gstreamer, command as below:

```
gst-launch-1.0 playbin uri=file:///home/root/test.mp4 video-sink=waylandsink
```

5 References

- Early Boot and Late Attach in Linux wiki
- Multimedia

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265