Advantiv® VSP Driver Specifications

For ADV800x family of products

Preliminary 0.40

.

June 2016

Revision History

Revision	Date	Remarks
0.1	June 21st, 2011	Initial revision
0.2	Oct 18th, 2011	to add new APIs
0.3	Jan 20th, 2012	to add section 6.3.14~17 & 6.4.6~7 for new APIs
0.31	Feb 8th, 2012	To modify section 6.3.1 ADIAPI_VspInputVideoChanged
0.32	March 5th, 2012	to add section 6.5.2 ADIAPI_VspSendVSInfoFrame
0.33	July 27th, 2012	add: 6.3.18 ADIAPI_VspEnableLowLatencyMode, and 6.4.8 ADIAPI_VspDetectTX4Kx2K
0.34	April 9th, 2013	added: 6.3.19 ADIAPI_VspPrimarySetMode 6.4.9 ADIAPI_VspDetectAllTX4Kx2K 6.4.10 ADIAPI_VspGetAviReceived 6.4.11 ADIAPI_VspGetMuteAtTx
0.35	July 24, 2013	added supported evaluation platforms replaced 8002/8003 by 800x where applicable
0.36	August 14, 2013	Added, copyright and product information sections, performed editorial corrections and updates
0.37	Sept 04, 2013	added 6.4.12 ADIAPI_VspInputover2K 6.5.3 ADIAPI_VspClearVSPacket
0.38	Nov 14, 2013	Added more formats to VSP_OUT_FORMAT added new structure: 7.4.5 VSP_HPS_PARAMS added new APIs: 6.3.20 ADIAPI_ PvspCapability 6.3.21 ADIAPI_ SvspCapability 6.3.22 ADIAPI_ VspAssignHps 6.4.13 ADIAPI_VspClockOverThreshold 6.4.14 ADIAPI_VspDetectVid4Kx2K changed APIs: 6.3.2 ATV_ERR ADIAPI_VspAutoAssignOutput 6.4.1 BOOL ADIAPI_VspOutputBypassed 6.4.12 BOOL ADIAPI_VspInputoverCap
0.39	Nov 4, 2014	Defines VSP_OUT_FMT_1680x720, 2560x1080 added, InVic Parameter added in ADIAPI_VspInputoverCap API, TX Aspect Ratio 64x27 & 256x135 added in TV_ERR ADIAPI_VspUpdateTX API
0.40	Jan 28, 2014	Added ADIAPI_VspCorrect4Kx2KVic(), ADIAPI_VspFindIntermVic(), and ADIAPI_VspSupportThisOutput()

Table of contents

PRODUCTINFORMATION

Product information can be obtained from the Analog Devices Web site and other Web sources.

Analog Devices Web Site

The Analog Devices Web site, www.analog.com, provides information about a broad range of productsanalog integrated circuits, amplifiers, converters, and digital signal processors. To access a complete technical library for each video product family, go to http://www.analog.com/en/audiovideo-products/products/index.html.
Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more. Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

Engineer Zone

Engineer Zone is a technical support forum from Analog Devices. It allows you direct access to Analog Devices technical support engineers. You can search FAQs and technical information to get quick answers to your questions about Analog Devices video products at http://ez.analog.com/community/video.

COPYRIGHT INFORMATION

DISCLAIMER

Analog Devices, Inc. (ADI) reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

The information contained in this document is proprietary of ADI. This document must not be made available to anybody other than the intended recipient without the written permission of ADI.
The content of this document is believed to be correct. If any errors are found within this document or if clarification is needed, contact the video community on EZ forum (http://ez.analog.com/community/video).

TRADEMARK AND SERVICE MARK NOTICE

The Analog Devices logo is a registered trademark of Analog Devices, Inc. The Advantiv® and Blackfin® are registered trademarks of Analog Devices Inc. All other brand and product names are trademarks or service marks of their respective owners. All other brand and product names are trademarks or service marks of their respective owners. Analog Devices' Trademarks and Service Marks may not be used without the express written consent of Analog Devices, such consent only to be provided in a separate written agreement signed by Analog Devices. Subject to the foregoing, such Trademarks and Service Marks must be used according to Analog Devices' Trademark Usage guidelines. Any licensee wishing to use Analog Devices' Trademarks and Service Marks must obtain and follow these guidelines for the specific marks at issue

1.0 Introduction

1.1 Purpose

This document describes the software architecture for ADI's ADV800xVideo Signal Processor product family, referred to in this document simply as the VSP, running on Analog Devices ADV800x reference platforms. The term ADV800x, used throughout this document, refers to VSP product family of ADV8002, ADV8003or future similar products.

Besides describing the software architecture, the document also serves as a programmer's reference for using and utilising various software modules running on the platform, with a complete list of Application Program Interfaces (APIs) and associated data types, macros and defines.

1.2 Scope

This document is intended to be used in conjunction with, and it largely depends on the following documents:

- ADV800x hardware manual
- Advantiv® Software Architecture Specifications
- Transmitter Library API Specifications for HDMI/DVI/ Interfaces (http://ez.analog.com/docs/DOC-8566)
- Receiver Library API Specifications for HDMI/DVI/Analog Interfaces(http://ez.analog.com/docs/DOC-8566)
- Advantiv® Repeater Driver Specifications
- Advantiv® Bitmap_OSD_Library_Specifications
- BLIMP user manual

It is strongly advised to go through the above documents, especially the Advantiv® software architecture specifications, before using this document to get an understanding of the overall VSP control flow and how the VSP fits into the Advantiv® software architecture.

1.3 Acronyms and Abbreviations

This is the list of common acronyms and abbreviations used in this document:

ADV800x	ADV8002, ADV8003, ADV8005 or future similar products
AVI	Auxiliary Video Information
CS	Color Space
CSC	Color Space Converter/Conversion
MIDWARE	Middleware layer
OSD	On-Screen Display
P2I	Progressive-To-Interlaced
PVSP	Primary VSP
RX	A product\device that acts as Audio\Video receiver interface
SVSP	Secondary VSP
TX	A product\device that acts as Audio\Video transmitter interface
VSP	Video Signal Processing or Video Signal Processor

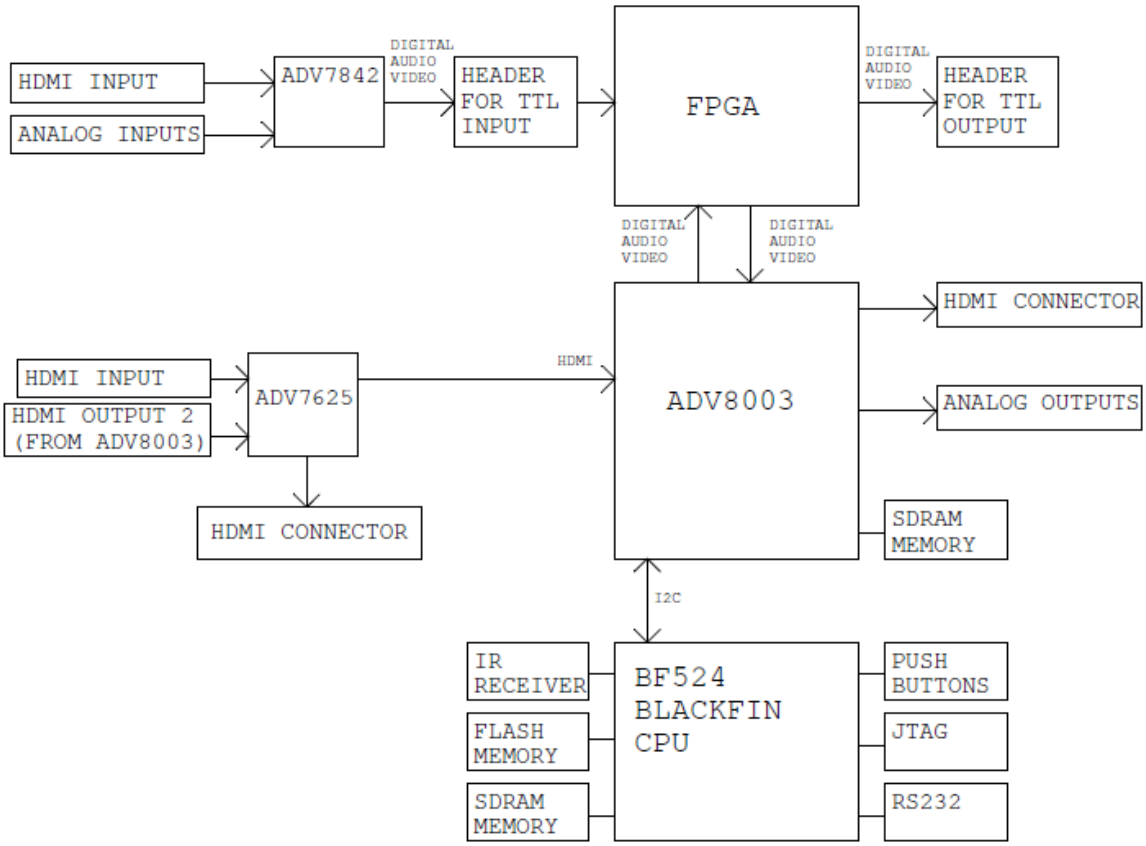
Overview

1.4 Hardware

The VSP software is designed to run on ADI ADV800xevaluation platform known as:

- EVAL ADV800x-SMZ

EVAL-ADV800x-SMZAdvantiv®evaluation boards are utilizing a Blackfin®processor on two evaluation platforms
1. EVAL-ADV8003-SMZ uses ADV8003 as TX device and ADV7842 and ADV7625 as RX devices
2. EVAL-ADV8005-SMZ uses ADV8005 as TX device and ADV7842 and ADV7625 as RX devices
In below diagram the ADV8005 evaluation platform is shown as an example. It utilizes ADV7842HDMI RX device and ADV7625 HDMI Rx device as front ends to the ADV8005. The ADV7625 provides TMDS RX to ADV8005, The ADV7842provides four HDMI inputs and programmable Analog inputs (Component, CVBS, S-Video, etc) and connects to the ADV8005using 36-bit pixel bus (TTL interface) to provide video data. Audio is supplied to theADV8005using I2S. The MCU used on the eval board is the Blackfin core BF524.All devices on the platform are controlled using a common I2C bus, with the MCU acting as the bus master.



EVAL-ADV800X- SMZEval platform

1.5 Software

The VSPsoftwareruns on the target platform's embedded microcontroller. All devices on the platform (e.g., ADV800x, ADV7842, and ADV7625) are controlled by system software via I2C. Communication between the OSD ROM and the MCU/ADV800x is done using SPI. Please refer to the OSD specifications document for details on the OSD.

The VSP software is built on top, and uses the existing ADI repeater driver to perform repeater functionality. This document will only focus on software features specific to the VSP. For information regarding repeater operation please refer to the Advantiv® repeater driver specifications and accompanying documents.

The VSP software consists of multiple modules organized in three layers:

- VSP application
- VSP middleware
- VSP low-level library

The various, VSP-specific, functions performed by the software are summarized below, along with the layer it belongs to.

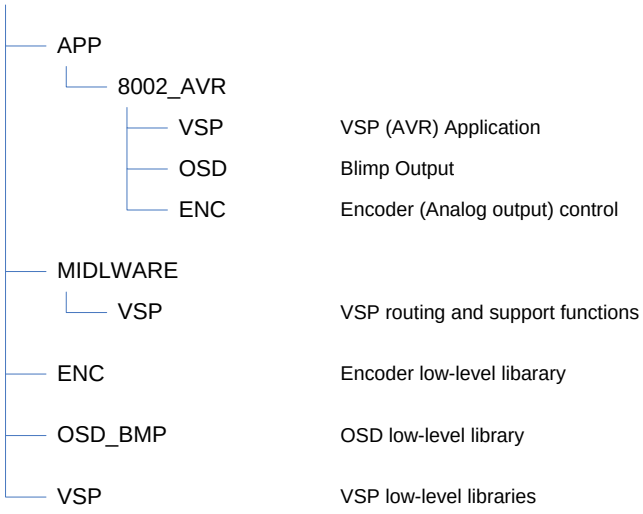
- Video scaling (Library layer)
- HDMI Splitter (Middleware layer)
- BKSV merging (Middleware layer)
- Video signal routing (Middleware layer)
- EDID merging (Application layer)
- Communication with the OSD software module (Application layer)

2.0 Software architecture

2.1 VSP Folder structure

The VSP module is supplied in source format. All source files are in standard ANSI C to simplify porting to any platform. The VSP module uses VSP hardware support libraries and the hardware abstraction layer to control and configure the system and functions.

The folder structure for the ADV800x VSP follows the general Advantiv software structure described in the Advantiv® Software Architecture Specifications document. The folder tree below describes the additional modules specific to the
ADV800x (VSP, OSD, and Encoder.) The HDMI TX portion is covered in the repeater specification document.



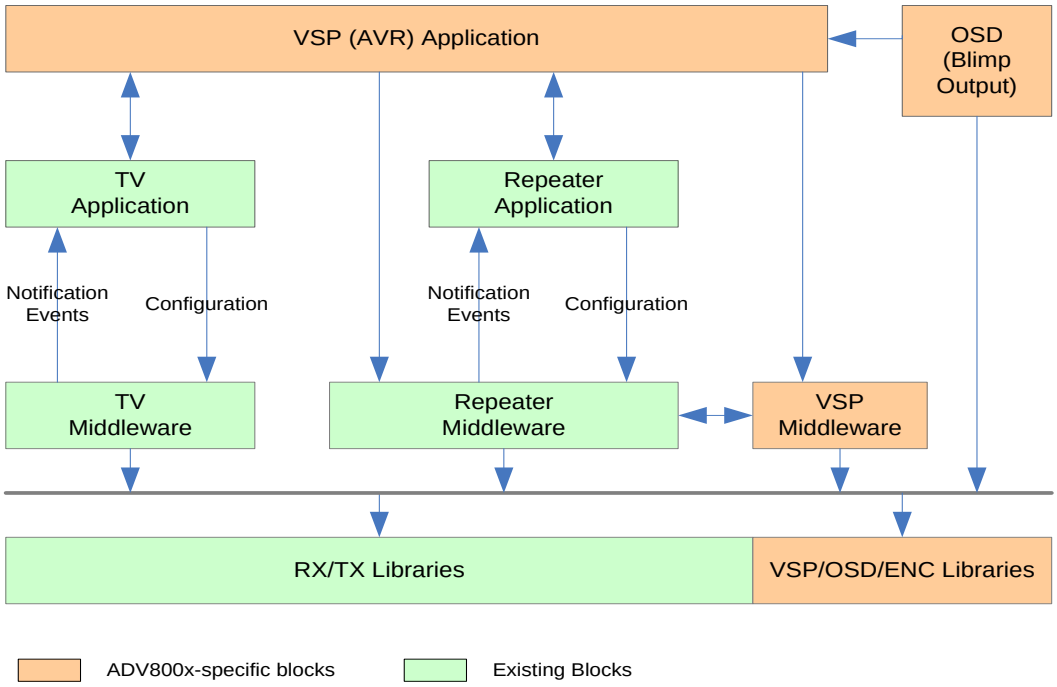
2.2 Software blocks interconnectivity

The VSP software is built on top of the existing repeater software. As far as HDMI repeater is concerned, the ADV800x is treated as an HDMI transmitter. All repeater functionality is performed in the repeater middleware and application.

For the VSP part, a VSP middleware and library modules were added to handle scaling, video signal routing and video enhancements.

For the OSD, the OSD library and application (Blimp output) were added to handle all OSD-related (both in terms of s/w and h/w) functionality.

The dual TX outputs (Splitter) are handled implicitly within the repeater middleware as will be explained later.



ADV800x Software Blocks Interconnectivity

The VSP (AVR) application handles both HDMI/Analog input to HDMI output routing combinations as well as some user-configurable functionalities related to the splitter.

3.0 VSP Software features

3.1 HDMI Splitter

The ADV800x support dual HDMI transmitters. Both of the HDMI outputs can be active at the same time and can output different video formats, however it must be the same video content on both outputs.

Splitting HDMI signal into two outputs involve three steps:

1- Routing the video signal and associated metadata (Info frames) to both outputs
2- Merging the EDIDs from both sinks into one EDID
3- Combining HDCP-related parameters (BKSVs, Bstatus) from both sinks

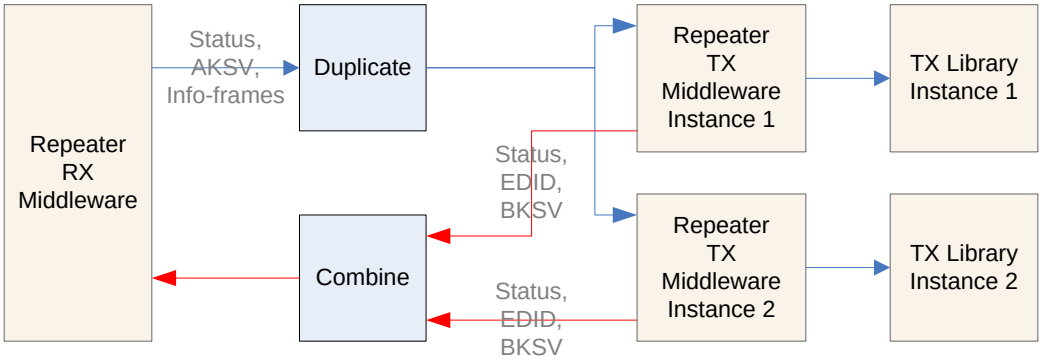
The following sections describe the details of each step.

3.1.1 Metadata Routing

The repeater application and middleware automatically handles metadata routing from HDMI RX device to HDMI TX device. To support dual HDMI TX devices, the repeater TX middleware was changed to support multiple instances of the same TX device.

The TX library already supports multiple identical devices. By adding multi-device support to the repeater TX middleware, two independent TX repeaters can be operational in a single system.

Additional handling of Status, Info-frames, EDID and BKSV from both sides of the repeater is required to achieve splitter functionality. This is demonstrated in the diagram below.



The functionality performed in the duplicate and combine blocks in the above diagram is divided between the repeater middleware and the application. For example, combining of status and BKSVs

is done in the repeater middleware as it is not customizable, while combining of EDIDs is done in the application as it largely depends on customer requirement.

Duplication of status and info-frames is done exclusively in the repeater middleware. It should be noted, however, that any calls made to the TX (middleware or library) from the application must take into account which TX device it needs to access and possibly duplicate the action on both devices if required.

3.1.2 EDID Merging

Since two sinks exit in the system, each with its own EDID, the EDID presented to the source device must reflect features that exist in both sinks and remove features that apply to one sink but not the other.
For example, if one of the sinks supports 12-bit deep color and the other sink does not, the EDID presented to the source must support maximum of 8-bit color depth, otherwise the secondary sink may not be able to display the signal.

EDID merging is complicated however by the existence of VSP or DSP in the signal path. If the audio or video signals received from the source can be manipulated by the system VSP or DSP, then the EDID presented to the source can reflect features not necessarily supported by both sinks (e.g., HBR audio) as the system will take care of converting non-supported formats into a format that can be used by all sinks in the system.

As EDID merging is essentially customer-specific algorithm, it takes place in the VSP application folder in the ATV tree.

The way EDID merging is current done is as follows:

For each EDID received from the sink device, the repeater application notification function (upon the reception of REP_EVENT_TX_EDID_READY event) calls the VSP application function "VspApp_SplitterMergeEdids" to merge the newly received EDID with the one currently in use by the RX device.

VspApp_SplitterMergeEdids() will check if either of the two VSPs is in the signal path either sink (i.e, if the signal output to the sink is scaled/processed by the VSP) and will alter the merging algorithm as follows:

For Video features merging:

- If both TX outputs are pass-through (no VSP,)the merged EDID will have the best common video and 3D features of both EDIDs
- If either of the two TX outputs is pass-through, the merged EDID will contain the video modes of the EDID associated with the pass-through outputplusthe best common 3D features of both EDIDs
- If both TX outputs are utilizing the VSP, the merged EDID will be based on the main TX EDID plus best common 3D features of both TX EDIDs. In this case, it doesn't really matter which EDID is used to provide supported formats as the output will be scaled for both TXs anyway.

For Audio features merging:

- If Audio Merge is enabled, the merged EDID will contain the best common audio features of both EDIDs
- If Audio Merge is disabled, default (fixed) audio format will be used

Audio and video merging can be individually controlled at compile-time by local macros in the VSP application.

VspApp_SplitterMergeEdids() return value indicate if the merged EDID contain the same features as the EDID currently used by the RX device so that to avoid unnecessarily hot-plugging the source device if the EDID did not change.

3.1.3 BKSV Merging

For HDCP splitter, HDCP authentication is performed between the input of ADV7625 and both outputs of ADV800x. From the source point of view, the attached system appears as a repeater with two BKSVs (the two attached sinks). For each sink, the ADV800x appears as any normal HDCP source.

Whenever a sink is attached to any one of the two HDMI outputs of ADV800x while encryption is enabled, the BKSV list and Bstatus of the attached sink will be combined with the BKSV list and Bstatus of the other sink (if one is attached) and the combined parameters will be sent to the HDMI RX device to complete the authentication. This combining of BKSVs/Bstatus is performed automatically in the repeater TX middleware and cannot be controlled by the application.
When a sink device is removed from the system, its BKSV list and Bstatus are not removed from the combined list so that the HDCP connection with the source device is not disrupted.

When the RX device receives a new Bstatus/BKSV list, it compares it with the current parameters in use. If the two parameters exactly match, no action will be taken and the HDCP link is maintained. If the parameters do not match, the source HPD will be toggled to indicate a re-authentication request from the repeater.

3.2 Video signal routing

The ADV800x has two independent VSP modules: Primary VSP and secondary VSP.
The primary VSP (referred to as VSP1 in this document) can accept wide variety of input formats and can upscale/downscale to a wide selection of output formats, with frame rate conversion. VSP1 can only output progressive formats, and hence a P2I (progressive-to-interlaced) module is used to convert VSP1 output to interlaced format if required.
The secondary VSP on the other hand (VSP2) can accept only progressive formats and although it can upscale the input signal and perform frame-rate conversion, it is not designed to do so. Furthermore, although VSP2 has an integrated P2I converter, it cannot output interlaced 1080 formats (1080i50/1080i60)

The intended use of VSP2 is to only downscale from 1080p/720p to 720p/480/576 formats ***without*** frame-rate conversion (i.e., 1080p50/720p50 to 720p50/576p50/576i50 and 1080p60/720p60 to 720p60/480p60/480i60)

To accommodate the limitation of VSP2, the system menu (OSD) should be designed so that it only allows two types of output formats: Main format and Auxiliary format.

1- The Main output format should give the user the option to select any format (CEA or VESA) that can be output on the user-selected interface. Only one interface can be designated as the main output.
2- The Auxiliary output format should be a slave to the main format. If the main format is 1080p/720p then the user should be able to select 720/576/480 formats (with the same frame rate as the main format) on the user-selected auxiliary interfaces. Any output interface other than the main output can be used as an auxiliary interface.

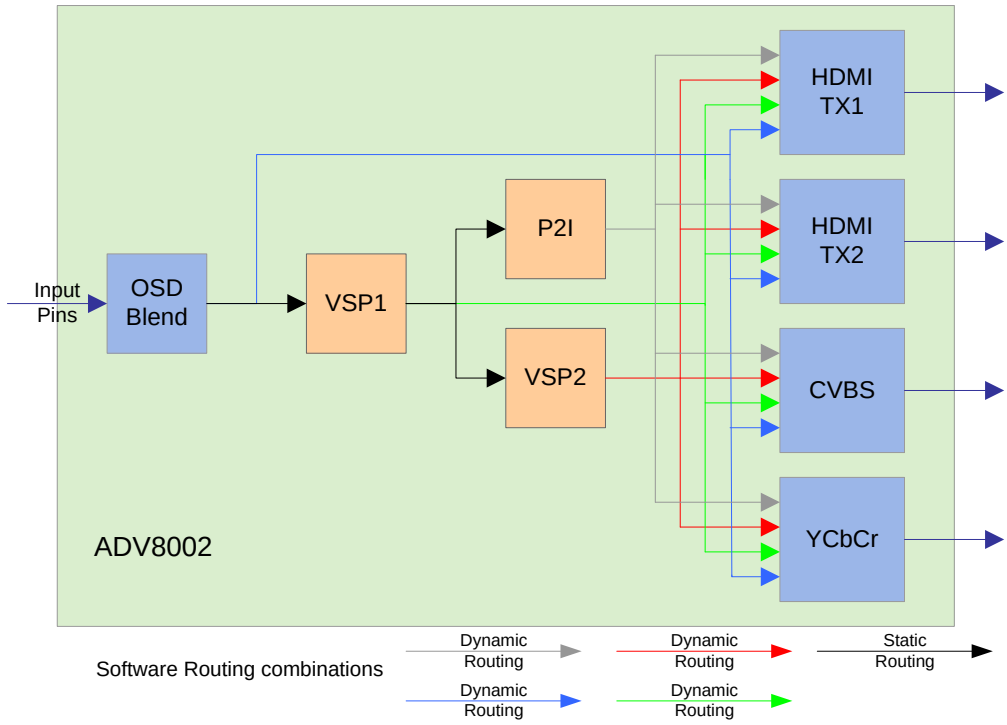
The VSP middleware is built with the assumption that the OSD blend module input will be always connected to theADV800x input pins. This was done to provide OSD on all ADV800x outputs at the same time. Any internal signal routing done by the software uses the OSD blend module as the signal source; the input pins are never used as signal source except for the OSD blend module.

Connections between OSD blend module, VSP1, VSP2, P2I and various output interfaces are determined dynamically at run-time by the VSP middleware based on the required formats on the main and auxiliary output interfaces and the type of input signal. This routing is not configurable by the application.

The VSP middleware will automatically configure VSP1 and VSP2 to provide the formats required on at least two (Main and auxiliary) of the four output interfaces. The assignment of VSP1 or VSP2 to a specific interface is determined by the VSP middleware and is not reported to the application. Typically, the higher resolution format will be output from VSP1/P2I modules and the lower resolution format will be output from VSP2.

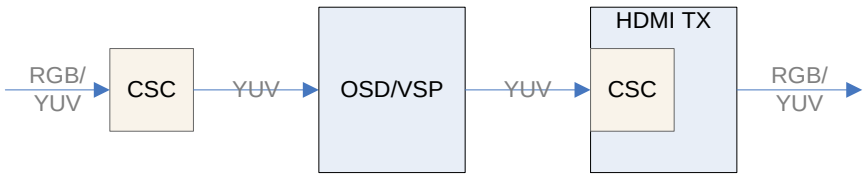
As can be seen from the diagram below, four possible signals can be output on any of the four output interfaces:

1- OSD blend output. This is essentially the same as the input signal, blended with the OSD.
2- VSP1 output, a scaled progressive format with OSD
3- P2I output, an interlaced version of the VSP1 output
4- VSP2 output, a scaled format with OSD



VSP Software routing combinations

It is worth noting that the OSD blend and VSP modules can only work in YUV color space. For this reason, the software will use the ADV800x built in CSC (Color Space Converter) to automatically convert the input signal to YUV color space first before it is sent to the OSD blend module. The output signal sent to HDMI TX modules will be converted back to the original color space (using the HDMI TX CSC module) only if the output is in pass-through mode or the sink only supports RGB. The signal will remain in YUV color space otherwise.



The following table summarize the ADV800x input/output color space combinations supported by the software.

ADV800x Input Color Space	HDMI TX Output Color Space		
	Sink is RGB only	Pass-through	Output is scaled

		mode	
YUV	YUV	YUV	YUV
RGB	RGB	RGB	YUV

As mentioned in previous sections, the routing of info-frames between HDMI RX input and HDMI TX output is done by the repeater middleware. However, to adjust for format and color space change done by the VSP middleware, the AVI info-frame is sent from the repeater middleware to the VSP middleware for update before it is sent to the HDMI TX.

3.3 Bandwidth Adjustment

The VSP uses external DDR memory to store video frames (up to 7 frames at a time) for scaling and other video enhancements features such as noise reduction. DDR memory is also used by the OSD to store menu data.
As DDR memory has a finite bandwidth, exceeding this bandwidth by the utilization of VSP1, VSP2 and the OSD will cause video artifacts due to the inability to read or write the required information from/to DDR memory by any of the modules.

When the VSP reads or writes pixel data to/from external memory, it can do so using 32, 24, 16 or 8 bits per pixel. This is independent of the actual color depth of the input signal; the VSP will automatically convert each pixel to fit within the required number of bits before writing it to memory. Obviously, using 32 bpp for pixel storage produce the best color results while using 8 bpp produce the lowest color quality. A desired side effect of using smaller pixel size though is lowering the bandwidth consumed by the VSP.

The VSP software automatically adjusts the performance of VSP1 and VSP2 to accommodate the maximum bandwidth for the DDR used in the target system. The software uses a combination of pixel width adjustment, random-noise reduction and low-latency mode.

Whenever the VSP middleware configures either one of the two VSPs, it calculates the bandwidth consumed by the current configuration (VSP1+VSP2+OSD) to see if it exceeds the maximum allowed bandwidth for the target system (defined by the macro MEM_MAX_BANDWIDTH in atv_preprocessor.h)

The software will always configure both VSPs for 32 bit pixel width at the start of each configuration request. If the calculated bandwidth exceeds the maximum allowed bandwidth for the target system, the software will try to reduce the bandwidth by performing each of the following steps in the listed order until the bandwidth fits within the maximum limit:

1- Disable random-noise reduction on VSP1 (if it is enabled). Random-noise-reduction operations consume significant memory bandwidth.
2- Reduce pixel size to 24 bpp on VSP2
3- Reduce pixel size to 24 bpp on VSP1
4- Reduce pixel size to 16 bpp on VSP2

5- Reduce pixel size to 16 bpp on VSP1
6- Reduce pixel size to 8 bpp on VSP2
7- Reduce pixel size to 8 bpp on VSP1
8- Put VSP2 in low-latency mode. Low-latency mode does not consume DDR bandwidth but has limitation on scaling options

The above order is currently fixed in the VSP middleware. An API will be added in future revisions of the software to allow the application to select the order if required.

The bandwidth consumed by the OSD varies depending on the OSD design and cannot be calculated by the application at run-time. The maximum bandwidth consumed by the OSD is calculated by Blimp (see OSD integration section) during the OSD design phase and is provided as a "#define OSD_MAX_BANDWIDTH" in the Blimp generated code.

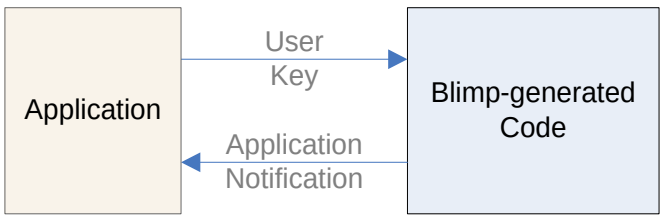
3.4 OSD Integration

The bitmap OSD feature of ADV800x is covered in the OSD and Blimp user manuals. This section will focus on the integration of Blimp output with the rest of the system.

Blimp is PC-based tool developed by ADI to allow the user to create custom menu system without the need to delve into the internals of ADV800x OSD programming. The output of Blimp is C source files that, when compiled with the rest of the ADV800x system software, will program ADV800x OSD module to produce menus exactly as designed (and seen) on Blimp.

Blimp output code completely handles the GUI part of the menu system, including user key press and menu navigation. The code expect the application to provide the user-pressed key via the function **ADIAPI_OSDEgPostKeyMsg(UserKey)**and will handle all required navigation from there.

Whenever the user navigates or select any item on the menu, the OSD code will call the application indicating the menu item that was selected or navigated to. This notification from the Blimp output code to the application is specified during the menu design phase.



One of the files generated by Blimp is called "blimp_external_api.c". This file contains all the functions that will be called from the OSD on any user-initiated menu event. All functions are empty by default, so no action will be taken when a user for example select any particular item from

the menu. Part of the OSD integration process is to fill all of the function is this file with code that will perform the required action selected by the user.

As an example, if a user selects the output format on HDMI TX1 to be 1080p60, the Blimp code will call the corresponding function in blimp_external_api.c, indicating that the user selected HDMI TX1 output to be 1080p60. This function is initially empty, and must be filled with a call to the VSP middleware to actually switch HDMI TX1 output to 1080p.

Blimp output should be placed in the OSD folder under the APP/8002_AVR folder. It internally communicates with the OSD library in the OSD_BMP folder.

The OSD is typically generated by Blimp for a specific resolution (720p in the case of the OSD provided by ADI.) The OSD blend module thus needs to first scale the OSD to the correct input resolution before it can be blended with input video. The OSD blend hardware has an internal scalar that is intended for this purpose and must be set by the application to scale the OSD whenever the input format changes. The OSD library provides an API to scale the OSD, which is called by the application whenever input format to ADV800x changes.

4.0 APIs

The VSP middleware utilizes the VSP hardware through the VSP library's exported APIs. It is thus essential that the application refrain from directly configuring the VSP module using the library APIs and only use the APIs exported by the middleware to avoid conflict with the VSP driver operation.

All VSP APIs are prefixed with "ADIAPI_Vsp" and all APIs return a value of type ATV_ERR. The complete list of APIs is defined in the VSP Middleware API Listsection of this document.

All data types and macros required for the VSP middleware APIs are defined in the file "MIDLWARE/VSP/vsp_mw.h". All data types and macros required for the VSP library are defined in the file "VSP/800x/vsp_lib.h"
A complete list of enumerations used by the VSP middleware is defined in the VSP Enumerations and structures Listsection.

5.0 VSP Middleware API List

The VSP middleware provides a set of APIs to control, configure and provide status on all aspects of the VSP module. All APIs are available for the application software to use at any time.

5.1 Initialization

5.1.1 ADIAPI_VspMwInit

Description

Initialize the VSP middleware module. This API must be called before any other call to the VSP middleware.

Synopsis

#include "vsp_mw.h"

ATV_ERRADIAPI_VspMwInit(DEF_VSP_PARAMS *VspParams)

Parameters

VspParams

Pointer to a structure holding default output formats and port assignments to be used during initialization. This is typically the required formats on the four output interfaces and the primary/secondary channel assignments. See VSP Enumerations and structures List for details.

Return value

ATVERR_OK

Remarks

This function must be called once by the application during start-up to initialize the VSP middleware.

5.2 EDID

5.2.1 ADIAPI_VspNewEdidReceived

Description

Inform the VSP of the EDID received from the sink device (Either sink connected to TX1 or TX2). This is used by the VSP middleware to find the appropriate output format for the particular sink in "auto" output mode. This function is automatically called from the repeater middleware whenever a new EDID is received and does not need to be called by the application.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspNewEdidReceived(UINT16 SegNum, UCHAR *EdidSeg)

Parameters

SegNum *EDID segment number, 0 or 1*

EdidSeg *Pointer to the EDID segment*

Return value

ATVERR_OK

Remarks

This function is called from the repeater middleware whenever a new EDID is received from either TX. If the repeater middleware is not used, this function must be explicitly called from the application.

5.3 VSP Configuration

5.3.1 ADIAPI_VspInputVideoChanged

Description

Re-configure the VSP to adjust to the change in input video timing.

Synopsis

#include "vsp_mw.h"

UINT16ADIAPI_VspInputVideoChanged(BOOL VidDet, VSP_TIMING_FORMAT *InTiming)

Parameters

VidDet *TRUE if input video timing (to the ADV800x) is known*
FALSE if input video timing is not known

InTiming *Pointer to the Timing details of input video. Only used if VidDet is TRUE.*

The details for VSP_TIMING_FORMAT structure is as follows:

```
typedefstruct
{
UINT16  HFrontPorch;        /* Horizontal front porch (pixels) */
   UINT16 HSyncDur;                 /* Horizontal sync pulse width (pixels) */
   UINT16 HBackPorch;       /* Horizontal back porch (pixels) */
UINT16  VFrontPorch;        /* Vertical front porch (lines) */
   UINT16 VSyncDur;                 /* Vertical sync pulse width (lines) */
   UINT16 VBackPorch;       /* Vertical back porch (lines) */
   UINT16 HActive;          /* Horizontal active pixels */
   UINT16 VActive;          /* Vertical active lines */
   BOOL   IsIntl;           /* TRUE if interlaced, FALSE if progressive */
   UCHAR   FrameRate;               /* Frame rate */
   UCHAR   VideoID;                 /* VIC from AVI info frame or 0 if not known */
   UCHAR   HPol;            /* Horizontal sync polarity. 0 if negative, 1 if positive */
   UCHAR   VPol;            /* Vertical sync polarity. 0 if negative, 1 if positive */
UINT32  VfmPeriod;          /* Not used */
} VSP_TIMING_FORMAT;
```

Return value

Input Video Frequency in MHz

Remarks

This function must be called from the application whenever the input video to the ADV800x is lost or changed.

5.3.2 ADIAPI_VspAutoAssignOutput

Description

This API is used to designate any of the output interfaces as the primary or secondary output. The VSP middleware can produce a maximum of two different output resolutions at any given time by utilizing the primary and secondary VSPs. This API instructs the middleware on which two output interfaces (and hence which two output resolutions) should be used. The assignment of primary and secondary output interfaces by the caller does not guarantee that primary VSP will be used for the primary output (or secondary VSP for the secondary output). The selection of which VSP will produce which signal is internal to the VSP middleware and cannot be changed by the caller.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspAutoAssignOutput(UCHAR OutChannel, VSP_OUTPUT_IDOutType)

Parameters

OutChannel *Designates OutType as primary/Secondary output*
Set to 0 to designate OutType as the Primary output
Set to 1 to designate OutType as the Secondary output

OutType *Output interface to be designated as primary/Secondary. Can be any of the following:*
OUTPUT_SEL_TYPE_OUT_HDMI1
OUTPUT_SEL_TYPE_OUT_HDMI2
OUTPUT_SEL_TYPE_OUT_COMP
OUTPUT_SEL_TYPE_OUT_SVIDEO
OUTPUT_SEL_TYPE_OUT_CVBS

Return value

ATVERR_OK

Remarks

None

5.3.3 ADIAPI_VspSelectTx1OutputMode

Description

Select HDMI TX1 output format

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSelectTx1OutputMode(VSP_OUT_FORMATNewOutStd)

Parameters

NewOutStd	*Select the required HDMI TX1 output format.* *The value VSP_OUT_FMT_AUTO will cause the VSP middleware to automatically select the best format that can be displayed on HDMI TX1 based on the EDID of the attached sink.* *The Value VSP_OUT_FMT_PASSTHRU will pass the input signal to HDMI TX1 output without scaling or video enhancements.* *All other values will produce the required format on HDMI TX1 output*

Return value

ATVERR_OK

Remarks

This function is used to set TX1 output format.

5.3.4 ADIAPI_VspSelectTx2OutputMode

Description

Select HDMI TX2 output format

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSelectTx2OutputMode(VSP_OUT_FORMATNewOutStd)

Parameters

NewOutStd	*Select the required HDMI TX2 output format.* *The value VSP_OUT_FMT_AUTO will cause the VSP middleware to automatically select the best format that can be displayed on HDMI TX2 based on the EDID of the attached sink.* *The Value VSP_OUT_FMT_PASSTHRU will pass the input signal to HDMI TX2 output without scaling or video enhancements.* *All other values will produce the required format on HDMI TX2 output*

Return value

ATVERR_OK

Remarks

This function is used to set TX2 output format.

5.3.5 ADIAPI_VspEnableGameMode

Description

Enable/disable game mode on primary VSP.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspEnableGameMode(BOOL Enabled)

Parameters

Enabled = TRUE to enable game mode
= FALSE to disable game mode

Return value

ATVERR_OK = Game Mode is Enabled for Progressive Input;
ATVERR_TRUE = Game Mode is Enabled for Interlaced Input;
Application should configure RX chip to double its clock;
ATVERR_FAILED = Game Mode Disable is selected. Hardware reset is required.

Remarks

Game mode saves DDR memory bandwidth and provides low latency between input and output video, desired for many gaming applications. However, many VSP advanced features, like sharpness, RNR, etc., will not be supported in game mode.

5.3.6 ADIAPI_VspMuteTxOutput

Description

Mute/unmute VSP output when the current TX instance is utilizing the VSP. This function is used internally by the repeater middleware to softly mute the TX output to avoid breaking the repeater HDCP link. This function should not be called directly from the application.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspMuteTxOutput(UCHAR Enable)

Parameters

Enable *= 1 tomute*
= 0 to unmute

Return value

ATVERR_OK

If the TX output is connected to one of the VSPs. Both VSP outputs will be muted/unmuted

ATVERR_FAILED

If the TX output is not connected to either VSP.VSP Mute state will not be changed.

Remarks

This function is used internally by the repeater middleware and should not be called by the application.

5.3.7 ADIAPI_VspSetMutingTime

Description

This function is used to set VSP blackout time.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSetMutingTime(UINT16 BlackoutTm)

Parameters

BlackoutTm *mute time in ms when using blackout*

Return value

ATVERR_OK

Remarks

This function is used to change blackout time.

5.3.8 ADIAPI_VspPassthruMutingTime

Description

This function is used to set TMDS, Blackout and HDCP time.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspPassthruMutingTime(UINT16 TmdsTm, UINT16 BlackoutTm, UINT16 HdcpTm)

Parameters

TmdsTm	*mute time in ms when using TMDS*
BlackoutTm	*mute time in ms when using blackout*
HdcpTm	*mute time in ms before to enable HDCP*

Return value

ATVERR_OK

Remarks

This function is used to change mute times.

5.3.9 ADIAPI_VspPrimarySetInputCrop

Description

This function is used to set the crop information for primary VSP.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspPrimarySetInputCrop(VSP_FRAME_CROP *InputCrop)

Parameters

InputCrop *Crop Information like Enable, TopLeftX, TopLeftY, Width, Height, etc.*

Return value

ATVERR_OK

Remarks

This function only saves the crop information. Function to configure Primary VSP should be run for crop to take effect.

5.3.10 ADIAPI_VspSecondarySetInputCrop

Description

This function is used to set the crop information for secondary VSP.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSecondarySetInputCrop(VSP_FRAME_CROP *InputCrop)

Parameters

InputCrop *Crop Information like Enable, TopLeftX, TopLeftY, Width, Height, etc.*

Return value

ATVERR_OK

Remarks

This function only saves the crop information. Function to configure Secondary VSP should be run for crop to take effect.

5.3.11 ADIAPI_VspSelectManualOutputMode

Description

This function is used to select primary/secondary VSP in manual output mode.

Synopsis

```
#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSelectManualOutputMode( VSP_OUT_FORMAT MainOutStd,
                                          VSP_OUT_FORMAT AuxOutStd)
```

Parameters

MainOutStd	*output format for main VSP*
AuxOutStd	*output format for auxiliary VSP*

Return value

ATVERR_OK

Remarks

This function is called from OSD to select primary and secondary VSP otuput mode by user.

5.3.12 ADIAPI_VspPrimarySetOutputAlbum

Description

This function is used to save the albume mode parameters.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspPrimarySetOutputAlbum(VSP_FRAME_CROP *OutputAlbum)

Parameters

OutputAlbum *album mode paramters,*
whenparamters are 0s, album mode should be disabled.

Return value

ATVERR_OK

Remarks

This function is called from console command to set album mode parameters. These paramters will take effect only when Primary VSP is configured.

5.3.13 ADIAPI_PriVspConfigure

Description

This function is used to configure Primary VSP manually without VSP routine selection

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_PriVspConfigure(VSP_CFG_MODE Mode, UCHAR InVic, UCHAR OutVic)

Parameters

Mode	*Primary VSP working mode*
InVic	*Primary VSP Input VIC*
OutVic	*Primary VSP Output VIC*

Return value

ATVERR_OK

Remarks

This function is called from console command to configure primary VSP without changing signal routine.

5.3.14 ADIAPI_SecVspConfigure

Description

This function is used to configure Secondary VSP manually without VSP routine selection

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_SecVspConfigure(VSP_CFG_MODE Mode, UCHAR InVic, UCHAR OutVic)

Parameters

Mode	*Secondary VSP working mode*
InVic	*Secondary VSP Input VIC*
OutVic	*Secondary VSP Output VIC*

Return value

ATVERR_OK

Remarks

This function is called from console command to configure secondary VSP without changing signal routine.

5.3.15 ADIAPI_VspUpdateTX

Description

This function is used to update TX variables

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspUpdateTX(VSP_OUT_FORMAT Tx1Std, VSP_OUT_FORMAT Tx2Std, UINT16 Tx1Aspect, UINT16 Tx2Aspect)

Parameters

Tx1Std	*TX1 video standard, like VSP_OUT_FMT_1080P60HZ, etc*
Tx2Std	*TX2 video standard, like VSP_OUT_FMT_1080P60HZ, etc*
Tx1Aspect	*TX1 video aspect, 4x3 or 16x9 or 64x27 or 256x135*
Tx2Aspect	*TX2 video aspect, 4x3 or 16x9 or 64x27 or 256x135*

Return value

ATVERR_OK

Remarks

This function is called from console command to work with ADIAPI_PriVspConfigure() or ADIAPI_SecVspConfigure() together.

5.3.16 ADIAPI_VspEnableLowLatencyMode

Description

This function is used to configure VSP in Low Latency Mode

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspEnableLowLatencyMode(BOOL Enabled)

Parameters

Enabled	*TRUE: VSP in low latency mode* *FALSE: VSP in auto mode*

Return value

ATVERR_OK

Remarks

This function is called from console command.

5.3.17 ADIAPI_VspPrimarySetMode

Description

This function is used to configure Primary VSP working mode

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspPrimaryMode(VSP_CFG_MODE PVspMode)

Parameters

PVspMode *Primary VSP working mode, like auto mode*

Return value

ATVERR_OK

Remarks

This function is used for PoP.

5.3.18 ADIAPI_PvspCapability

Description

This function is used to check if Primary VSP has capability to convert input format to output format

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_PvspCapability(UINT16 VicIn, UINT16 VicOut)

Parameters

VicIn VIC of Input timing
VicOut VIC of Output timing

Return value

ATVERR_OK: PVSP support this conversion
ATVERR_NOT_AVAILABLE: PVSP cannot support this output
ATVERR_FAILED: PVSP cannot support this input

5.3.19 ADIAPI_SvspCapability

Description

This function is used to check if Secondary VSP has capability to convert input format to output format

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_SvspCapability(UINT16 VicIn, UINT16 VicOut)

Parameters

VicIn VIC of Input timing
VicOut VIC of Output timing

Return value

ATVERR_OK: SVSP support this conversion
ATVERR_FAILED: SVSP cannot support this input
ATVERR_NOT_AVAILABLE: SVSP cannot support this output
ATVERR_TRUE: SVSP will do frame rate conversion
ATVERR_INV_PARM: SVSP will do upscaling

5.3.20 ADIAPI_VspAssignHps

Description

This function is used to check whether HPS is required
If yes, it is PVSP or SVSP to use HPS

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspAssignHps(UINT16 InVic, UINT16 Vsp1Vic, UINT16 Vsp2Vic)

Parameters

InVic VIC of Input timing
Vsp1Vic VIC of PVSP Output timing
Vps2Vic VIC of SVSP Output timing

Return value

0: HPS is not required
VSP1_IDX: PVSP uses HPS
VSP2_IDX: SVSP uses HPS

5.3.21 ADIAPI_VspCorrect4Kx2KVic

Description

This function is used to correct VICs for 3840x2160@24/25/30/50/60Hz

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspCorrect4Kx2KVic(UINT16 RxDetVic)

Parameters

RxDetVic *VIC detected by Rx*

Return value

VICs for aspect at 64:27 or 16:9

5.4 VSP Status

5.4.1 ADIAPI_VspOutputBypassed

Description

This API can be used to check if an output is processed by any one of the two VSPs (i.e., if the particular output is connected to one of the two VSP outputs AND the VSP operating mode is not bypass)

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspOutputBypassed(UCHAR OutputIdx)

Parameters

OutputIdx	*Output Index, = 0: HDMI TX1* *=1: HDMI TX2* *=2: Component* *=3: CVBS and S-Video* *=0xFF: to use the currently selected instance of HDMI TX (Refer to multi-device support for details)*

Return value

ATVERR_TRUE	If this output does not utilize the VSP
ATVERR_FALSE	If this output is utilizing one of the two VSPs

Remarks

None

5.4.2 ADIAPI_VspRnrAllowed

Description

This API can be used to check if RNR state can be changed by user. Random Noise Reduction can be automatically disabled by the VSP middleware to reduce DDR bandwidth (See Bandwidth Adjustment). If RNR is currently disabled by the middleware to reduce DDR bandwidth, this API can be used by the menu system to check if the RNR option should be greyed-out.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspRnrAllowed(BOOL *Allowed)

Parameters

Allowed *pointer of receive the return value*

Return value

The pointer "Allowed" will hold the return value.
It will be set to TRUE if RNR is not disabled by the middleware (and hence can be changed from the menus) and FALSE if RNR is disabled by the middleware

Remarks

This function is called from the OSD menus to check if RNR state change from the menus is allowed or not.

5.4.3 ADIAPI_VspPrimarySetRnr

Description

This API can be used to set primary VSP random noise reduction level.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspPrimarySetRnr(VSP_NOISE_RED NrLevel)

Parameters

NrLevel	*noise reduction level*

Return value

ATVERR_OK	RNR level is set by NrLevel;
ATVERR_FAILED	NrLevel is not a valid value, RNR level is not changed;

Remarks

This function is called from the OSD menus to set RNR value.

5.4.4 ADIAPI_VspGetInOutVic

Description

This API can be used to get VIC of input video and output video.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspGetInOutVic(UCHAR *InVic, UCHAR *OutVic, UCHAR Index)

Parameters

**InVic*	*address pointer to save the VIC of input video*
**OutVic*	*address pointer to save the VIC of output video*
Index	*Output video is from TX1 or TX2 or HD Encoder or SD Encoder or PVSP or SVSP*

Return value

ATVERR_OK

Remarks

This function is defined for future use. from the console command when to force VSP to output 60Hz or 59.94Hz.

5.4.5 ADIAPI_VspGetInOutFrameRate

Description

This API can be used to get Frame Rate of input video and output video.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspGetInOutFrameRate(UCHAR *InFR, UCHAR *OutFR, UCHAR Index)

Parameters

**InVic*	*address pointer to save the Frame Rate of input video*
**OutVic*	*address pointer to save the Frame Rate of output video*
Index	*Output video is from TX1 or TX2 or HD Encoder or SD Encoder or PVSP or SVSP*

Return value

ATVERR_OK

Remarks

This function is called from the console command when to set VSP in Frame Track Mode.

5.4.6 ADIAPI_VspGetInnputTiming

Description

This API can be used to get input timing informaiton

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspGetInputTiming(VSP_TIMING_FORMAT *InTiming)

Parameters

**InTiming address pointer to save the input timing information*

Return value

ATVERR_OK

Remarks

This function is defined for future use.

5.4.7 ADIAPI_VspGetTimingFormat

Description

This API can be used to get VSP timing format

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspGetTimingFormat(UINT16 Index, VSP_TIMING_FORMAT *VspTiming)

Parameters

Index	*timing index in array TimingFormats[]*
**VspTiming*	*address pointer to the timing format*

Return value

ATVERR_OK

Remarks

This function is defined for future use.

5.4.8 ADIAPI_VspDetectTX4Kx2K

Description

This API can be used to check if TX is working for 4Kx2K video

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspDetectTX4Kx2K(UCHAR TxID, UINT16 HActLine)

Parameters

TxID	*=0:*	*TX1;*
	=1:	*TX2;*
	=0xFF:	*all TXs*
HActLine	*=0:*	*using TX source to detect video timing*
	others:	*horizontal active lines*

Return value

TRUE: TX in 4Kx2K mode
FALSE: non-4Kx2K

5.4.9 ADIAPI_VspDetectAllTX4Kx2K

Description

This API can be used to detect if each TX is in 4Kx2K

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspDetectAllTX4Kx2K(void)

Parameters

none

Return value

bit0 = HIGH: TX1 in 4Kx2K
bit1 = HIGH: TX2 in 4Kx2K

5.4.10 ADIAPI_VspGetAviReceived

Description

This API can be used to save AVI infoframe received from RX

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspGetAviReceived(UCHAR *AviPkt)

Parameters

AviPkt: *pointer to save AVI Infoframe*

Return value

ATVERR_OK: AVI is saved
ATVERR_FAILED: invalid AVI Infoframe

5.4.11 ADIAPI_VspGetMuteAtTx

Description

This API can be used to check if TX in MUTE or not

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspGetMuteAtTx(void)

Parameters

none

Return value

TRUE: TX in mute
FALSE: TX not in mute

5.4.12 ADIAPI_VspInputoverCap

Description

This API checks input Horizontal timing

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspInputoverCap(UINT16 InVic)

Parameters

InVic *input VIC*

Return value

TRUE: input active horizontal pixel is over VSP capability
FALSE: with VSP capability

5.4.13 ADIAPI_VspClockOverThreshold

Description

This API checks if VSP input or output timing is over threshold

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspClockOverThreshold(UINT16 VidVic)

Parameters

VidVic: VIC for timing

Return value

TRUE: over VSP threshold, special settings required for VSP to support this timing
FALSE: within threshold

5.4.14 ADIAPI_VspDetectVid4Kx2K

Description

This API checks if it is 4Kx2K timing

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspDetectVid4Kx2K(UINT16 VidVic)

Parameters

VidVic: VIC for timing

Return value

TRUE: it is 4Kx2K timing
FALSE: non-4Kx2K timing

5.4.15 ADIAPI_VspFindInterimVic

Description

This API searches for the interim VIC for 4Kx2K scaling

Synopsis

#include "vsp_mw.h"

UINT16 ADIAPI_VspFindInterimVic(UINT16 InVic, UINT16 OutVic)

Parameters

InVic: Input VIC to VSP
OutVic: Output VIC form VSP

Return value

VIC: used by output from SVSP and input to PVSP

5.4.16 ADIAPI_VspSupportThisOutput

Description

This API checks if VSP could support this output timing.

Synopsis

#include "vsp_mw.h"

BOOL ADIAPI_VspSupportThisOutput(UINT16 OutVic)

Parameters

OutVic: Output VIC form VSP

Return value

TRUE:	VSP could generate this output;
FALSE:	this output is not available

5.5 Info Frame

5.5.1 ADIAPI_VspSendAVInfoFrame

Description

To send AVI info frame to both sink devices. The AVI info-frame received from the source devices is automatically sent to both sinks attached to HDMI TX1/TX2 by the repeater middleware. This function can be used to send a different AVI info-frame to the sinks, if required by the application. However, this info-frame will be overwritten by any new AVI received from the source device.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSendAVInfoFrame(UCHAR *AviPkt, UCHAR Size)

Parameters

AviPkt	*Pointer to AVI packet*
Size	*Packet size*

Return value

ATVERR_OK with
AviPkt containing the AVI InforFrame which was sent out.

Remarks

This function is used by the AVR application to send a dummy AVI info-frame to both sinks when the active input to the system is analog (YCbCr or CVBS)

5.5.2 ADIAPI_VspSendVSPacket

Description

To send VS info frame to sink. The VS infoframe is used to carry 4Kx2K information. As 8002 does not support 4Kx2K, this function is valid only for 8003 and further version.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspSendVSPacket(UCHAR *VsiPkt, UCHAR Size)

Parameters

VsPkt	*Pointer to Vs packet*
Size	*Packet size*

Return value

ATVERR_OK with
VsPkt containing the VsInforFrame which was sent out for 8002.

Remarks

This function is used by the AVR application when to prepare 4Kx2K video to sinks or when to crop 3D to 2D.

5.5.3 ADIAPI_VspClearVSPacket

Description

To send VS info frame to sink without extended video features.

Synopsis

#include "vsp_mw.h"

ATV_ERR ADIAPI_VspClearVSPacket(UINT16 Id)

Parameters

Id	*= 0: to clear all video features*
	= 0x3D: to remove 3D feature
	= 0x4000: to remove 4Kx2K feature

Return value

ATVERR_OK

6.0 Configuration

There are several configuration options for the VSP application. This section focuses only the options that are specific to the VSP. Description of other configuration options can be found in the repeater specifications document.

6.1 Supporting VSP

#define VSP_DEVICE 800x

This macro is located in "atv_preprocessor.h". It defines the VSP devices used asADV800x. This macro must be defined for proper VSP operation.

#define ES_800x 2

This macro is located in "atv_preprocessor.h". It defines the ES revision of the ADV800x used in the target system.

#define TX_SPLITTER 1

This macro is located in "atv_preprocessor.h". It indicates that both HDMI TX outputs of ADV800x are used in the target system.

This macro should be set to 0 when only one TX (HDMI TX1) is being used.

6.2 Supporting OSD

#define TX_INCLUDE_OSD 1

This macro is located in "rep_config.h". It indicates that OSD support is included with the VSP application. If OSD support is not required, this macro should be set to 0.

6.3 System

#define MEM_MAX_BANDWIDTH 1862 / In MBytes/s */*

This macro is located in "atv_preprocessor.h" It defines the maximum DDR memory bandwidth in the target system (in Megabytes/s). This macro is used by the VSP middleware to adjust VSP configuration to fit within this value.

#define ENC_DEVICE 800x

This macro is located in "atv_preprocessor.h". It defines the analog encoder device used as ADV800x. This macro can be omitted if analog output is not required.

#define AVR_APPLICATION 1

This macro is located in “atv_preprocessor.h”. It indicates that the VSP application should function like an AVR (handle analog/HDMI input/output routing combinations)

VSP Enumerations and structures List

This section describes all the enumerationsand structures that are used by the VSP middleware APIs.

6.3.1 VSP_OUT_FORMAT

Description

All supported middleware video output formats (on any output interface)

Synopsis

```
#include “vsp_mw.h”

typedefenum
{
   VSP_OUT_FMT_AUTO = 0,
   VSP_OUT_FMT_PASSTHRU,
   VSP_OUT_FMT_576I_50HZ,
   VSP_OUT_FMT_576P_50HZ,
   VSP_OUT_FMT_720P_50HZ,
   VSP_OUT_FMT_1080I_50HZ,
   VSP_OUT_FMT_1080P_50HZ,
   VSP_OUT_FMT_VGA_60HZ,
   VSP_OUT_FMT_480I_60HZ,
   VSP_OUT_FMT_480P_60HZ,
   VSP_OUT_FMT_720P_60HZ,
   VSP_OUT_FMT_1080I_60HZ,
   VSP_OUT_FMT_1080P_60HZ,
   VSP_OUT_FMT_1080P_24HZ,
   VSP_OUT_FMT_1080P_25HZ,
   VSP_OUT_FMT_1080P_30HZ,
   VSP_OUT_FMT_1920x1200_60HZ,
   VSP_OUT_FMT_1440x900_60HZ,
   VSP_OUT_FMT_1280x1024_60HZ,
   VSP_OUT_FMT_1360x768_60HZ,
   VSP_OUT_FMT_1280x800_60HZ,
   VSP_OUT_FMT_1280x768_60HZ,
   VSP_OUT_FMT_1024x768_75HZ,
   VSP_OUT_FMT_1024x768_72HZ,
```

VSP_OUT_FMT_1024x768_60HZ,
VSP_OUT_FMT_800x600_75HZ,
VSP_OUT_FMT_800x600_72HZ,
VSP_OUT_FMT_800x600_60HZ,
VSP_OUT_FMT_640x480_75HZ,
VSP_OUT_FMT_640x480_72HZ,
VSP_OUT_FMT_640x480_60HZ,
VSP_OUT_FMT_1280x720P_24HZ,
VSP_OUT_FMT_1280x720P_25HZ,
VSP_OUT_FMT_1280x720P_30HZ,
VSP_OUT_FMT_720x240P_60HZ,
VSP_OUT_FMT_2880x480I_60HZ,
VSP_OUT_FMT_2880x240P_60HZ,
VSP_OUT_FMT_1440x480P_60HZ,
VSP_OUT_FMT_720x288P_50HZ,
VSP_OUT_FMT_2880x576I_50HZ,
VSP_OUT_FMT_2880x288P_50HZ,
VSP_OUT_FMT_1440x576P_50HZ,
VSP_OUT_FMT_2880x480P_60HZ,
VSP_OUT_FMT_2880x576P_50HZ,
VSP_OUT_FMT_1920x1080IE_50HZ,
VSP_OUT_FMT_1920x1080I_100HZ,
VSP_OUT_FMT_1280x720P_100HZ,
VSP_OUT_FMT_720x576P_100HZ,
VSP_OUT_FMT_720x576I_100HZ,
VSP_OUT_FMT_1920x1080I_120HZ,
VSP_OUT_FMT_1280x720P_120HZ,
VSP_OUT_FMT_720x480P_120HZ,
VSP_OUT_FMT_720x480I_120HZ,
VSP_OUT_FMT_640x400_60HZ,
VSP_OUT_FMT_640x480_67HZ,
VSP_OUT_FMT_720x350_70HZ,
VSP_OUT_FMT_720x400_60HZ,
VSP_OUT_FMT_720x400_70HZ,
VSP_OUT_FMT_746x471_60HZ,
VSP_OUT_FMT_864x480_60HZ,
VSP_OUT_FMT_1152x870_75HZ,
VSP_OUT_FMT_1152x900_66HZ,
VSP_OUT_FMT_1280x1024_72HZ,
VSP_OUT_FMT_1280x1024_74HZ,
VSP_OUT_FMT_1280x1024_76HZ,
VSP_OUT_FMT_1920x1080P_100HZ,
VSP_OUT_FMT_1920x1080P_120HZ,
VSP_OUT_FMT_720x576P_200HZ,
VSP_OUT_FMT_720x576I_200HZ,
VSP_OUT_FMT_720x480P_240HZ,
VSP_OUT_FMT_720x480I_240HZ,
VSP_OUT_FMT_1600x1200_60HZ,

```
    VSP_OUT_FMT_848x480_60HZ,
    VSP_OUT_FMT_1280x960_60HZ,
    VSP_OUT_FMT_1366x768_60HZ,
    VSP_OUT_FMT_1400x1050_60HZ,
    VSP_OUT_FMT_1600x900_60HZ,
    VSP_OUT_FMT_1680x1050_60HZ,
    VSP_OUT_FMT_320x240_60HZ,
    VSP_OUT_FMT_640x350_85HZ,
    VSP_OUT_FMT_2048x1080_24HZ,
    VSP_OUT_FMT_2048x1080_50HZ,
    VSP_OUT_FMT_2048x1080_60HZ,
    VSP_OUT_FMT_2048x1152_60HZ,
    VSP_OUT_FMT_1792x1344_60HZ,
    VSP_OUT_FMT_1856x1392_60HZ,
    VSP_OUT_FMT_1920x1440_60HZ,
    VSP_OUT_FMT_2560x1600_60HZ,
    VSP_OUT_FMT_1680x720_24HZ,
    VSP_OUT_FMT_1680x720_25HZ,
    VSP_OUT_FMT_1680x720_30HZ,
    VSP_OUT_FMT_1680x720_50HZ,
    VSP_OUT_FMT_1680x720_60HZ,
    VSP_OUT_FMT_1680x720_100HZ,
    VSP_OUT_FMT_1680x720_120HZ,
    VSP_OUT_FMT_2560x1080_24HZ,
    VSP_OUT_FMT_2560x1080_25HZ,
    VSP_OUT_FMT_2560x1080_30HZ,
    VSP_OUT_FMT_2560x1080_50HZ,
    VSP_OUT_FMT_2560x1080_60HZ,
    VSP_OUT_FMT_2560x1080_100HZ,    /* not in use as pixel clock over 300MHz */
    VSP_OUT_FMT_2560x1080_120HZ,    /* not in use as pixel clock over 300MHz */
    VSP_OUT_FMT_3840x2160_30HZ,
    VSP_OUT_FMT_3840x2160_25HZ,
    VSP_OUT_FMT_3840x2160_24HZ,
    VSP_OUT_FMT_4096x2160_SMPTE24,
    VSP_OUT_FMT_4096x2160_SMPTE25,
    VSP_OUT_FMT_4096x2160_SMPTE30,
    VSP_OUT_FMT_OFF
} VSP_OUT_FORMAT;
```

Fields

VSP_OUT_FMT_AUTO and all VESA formats are only applicable for HDMI outputs.
VSP_OUT_FMT_PASSTHRU is only applicable for HDMI outputs.

6.3.2 OUTPUT_ID

Description

Define video output interfaces

Synopsis

```
#include "vsp_mw.h"

typedefenum
{
   OUT_HDMI1=0,
   OUT_HDMI2,
   OUT_COMPONENT,
   OUT_CVBS
}OUTPUT_ID;
```

Fields

OUT_HDMI1	HDMI TX1 output
OUT_HDMI2	HDMI TX2 output
OUT_COMPONENT	Component (YCbCr) output
OUT_CVBS	CVBS or S-Video output

6.3.3 VSP_NOISE_RED

Description

Noise reduction options

Synopsis

```
#include "vsp_mw.h"

typedefenum
{
   VSP_NR_DISABLED=0,
   VSP_NR_LOW,
   VSP_NR_MED,
   VSP_NR_HI
} VSP_NOISE_RED;
```

Fields

VSP_NR_DISABLED	Disable noise reduction
VSP_NR_LOW	Low level of noise reduction
VSP_NR_MED	Medium level of noise reduction
VSP_NR_HI	High level of noise reduction

6.3.4 DEF_VSP_PARAMS

Description

This structure defines the output formats and assignment that should be used during VSP middleware initialization

Synopsis

```
#include "vsp_mw.h"

typedefstruct {
   VSP_OUTPUT_ID          MainOutId;
   VSP_OUTPUT_ID          AuxOutId;
   VSP_OUT_FORMAT     Tx1Format;
   VSP_OUT_FORMAT     Tx2Format;
   VSP_OUT_FORMAT     CompFormat;
   VSP_OUT_FORMAT     CvbsFormat;
} DEF_VSP_PARAMS;
```

Fields

MainOutId	Define the output interface that will be used as the Main output
AuxOutId	Define the output interface that will be used as the Auxiliary output
Tx1Format	Define the video format required on HDMI TX 1 output
Tx2Format	Define the video format required on HDMI TX 2 output
CompFormat	Define the video format required on Component (YCbCr) output

6.3.5 VSP_HPS_PARAMS

Description

This structure defines the HPS (Horizontal Pre-Scaler) parameters

Synopsis

```
#include "vsp_mw.h"

typedefstruct {
   BOOL       PowerDown;
   BOOL       BypassFilter;
   BOOL       BypassDownsample;
   UCHAR      PhaseSel;
   UCHAR      FilterMode;
} VSP_HPS_PARAMS;
```

Fields

PowerDown	Define the power-down mode for HPS block
BypassFilter	Define the usage of filtering before downscaling
BypassDownsample	Define the usage of down-sampling
PhaseSel	Define whether the down-sampling should start by keeping or dropping the first pixel when in a 2-1 downscaling
FilterMode	Define the filer operating mode