Advantiv®
Advanced Television Solutions
by Analog Devices

# Advantiv® Software Architecture Specifications

**Preliminary 2.2**

**June 2016**

**Revision History**

| Revision | Date | Remarks |
|---|---|---|
| 1.0 | October 22nd, 2008 | Initial revision. |
| 1.1 | May 12, 2009 | Added Middleware into section 3 |
| 1.2 | June 11, 2009 | Updated Links and References |
| 1.3 | July 16th, 2009 | Adjusted for ATV EVAL board |
| 1.4 | October 16th, 2009 | Updated the Folder structure for U0.5<br>Changed to Rev 1.4 |
| 1.5 | January 8th, 2010 | Corrected terminology for application/driver<br>Added middleware block to architecture diagram |
| 1.6 | February 23rd, 2010 | Added application sub-module in the Platform module of the architecture diagram |
| 1.7 | March 30, 2010 | Removed reference to default HDMI Rx interrupt from HAL_INT1Pending and HAL_Int2Pending descriptions. |
| 2.0 | June 28, 2012 | Updated overall document to reflect changes in DVP evaluation hardware platforms. |
| 2.1 | Sep 02 ,2013 | Added, copyright and product information sections, performed editorial corrections and updates |
| 2.2 | Dec 10,2013 | Folder structure updated for cross point application |

# Table of contents

# Table of figures

# PRODUCT INFORMATION

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 products—analog 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.

### *EngineerZone*

EngineerZone 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*



### *TRADEMARK AND SERVICE MARK NOTICE*

# 1.0 Introduction

## *1.1 Purpose*

This document describes the general structure of the Advantiv software running on ADI Advantiv Video Evaluation platforms.

The Advantiv software stack consists of several software modules, including receiver, transmitter and platform libraries and the various applications that make use of those libraries such as a TV, a DVD player or an AVR system.

Although some applications may not require the use of all modules in the system, this document still serves as the *only* design document for any parts of the system that are common to all modules, whether they are used by a particular application or not. The user can choose to ignore the modules which are irrelevant to his/her particular needs. For a description of design particulars of any module not covered in this document, please refer to each module design document and API reference.

Besides the software architecture, this document also describes the guidelines that are observed when designing various parts of the system software so that integration and porting efforts to different platforms are minimized or greatly reduced.

# 2.0 Overview

## *2.1 Hardware*

The hardware platforms targeted by the Advantiv software example application are the various Advantiv product reference platforms from the various Consumer and Professional groups for the Advantiv products. These include but not limited to

- the Advantiv® Evaluation platforms

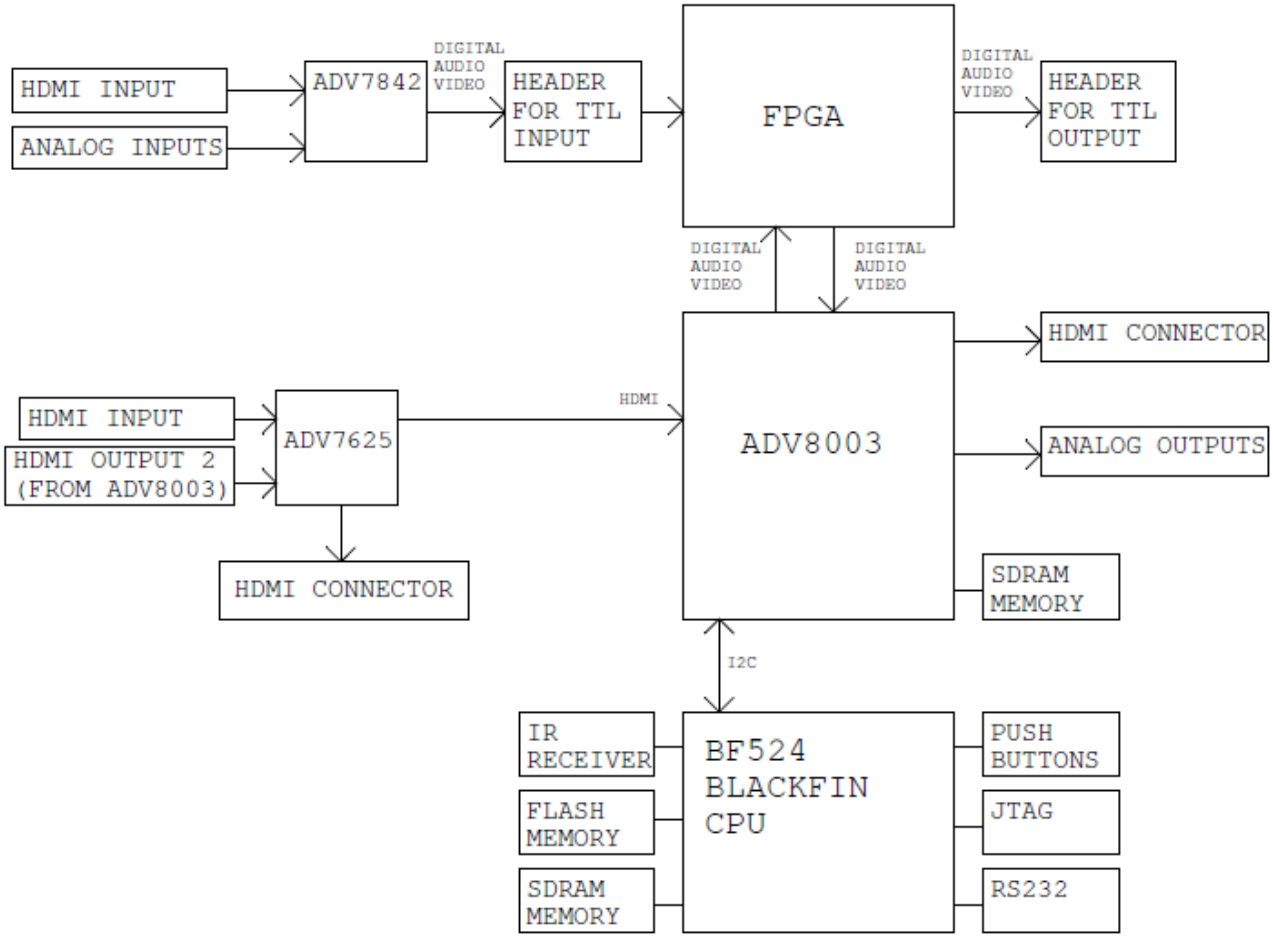



**Figure 1 – Advantiv® EVAL-ADV800X-SMZ Platform Components**

All platforms have a microprocessor to run the example software applications. . The Advantiv standalone platforms have Blackfin® microprocessor to run system software.

All platforms have a mini USB connector to control various components externally (using I2C) and a DB9 UART connector for system debugging. In addition the platforms may have a VGA connector to output video data to a VGA monitor, Analog audio I/O connectors and optical audio I/O connectors to output SPDIF or Analog audio to external speakers, audio headers for external connections. Please refer to the relevant evaluation platform schematic and user guide for full details on the hardware connections the software application will reflect the capability of the target hardware.

Each platform will include at least discrete ADI HDMI Rx and HDMI Tx pair (ADV7842 & ADV7511, ADV7619& ADV7511, ADV7611 & ADV7513) as or an ADI Transceiver device (ADV7623 or ADV7850). As ADI has a wide portfolio of video products depending on the platform other devices such as analog decoders, encoders and HDMI Muxes are also supported by the Advantiv software and may be included in an example software application.

The hardware configuration for a specific application, however, varies depending on the requirements of the application. A TV or receiver application, for instance, will typically use the RX board only to receive A/V data and output the received audio and/or video through the motherboard's VGA and Analog audio out. Alternatively, an HDMI TX board could also be used to output the received A/V data through HDMI interface.

## *2.2 Software*

All system software (application, libraries and otherwise) are developed and tested on the targets hardware micro-controller, be it BlackFin®. However the intent of the software is to provide an abstract API interface that will remain consistent across platforms, silicon revisions, and where possible, across chip families. The goal is to minimize porting effort to any target platform during the initial phases of integration, and maximize re-use of the existing application when moving to the future chip generations.

As such, the APIs exported by the Advantiv software stack are designed carefully so that it accommodates (as much as possible) variations in hardware characteristics and allow (within reason) for future expansion.

Since the Advantiv software stack is designed to run on any platform. All hardware interfaces are abstracted in the source code such that porting to any other platform can be done with ease. Hardware initialization is also abstracted in the source and no direct reference to any hardware component is made outside the hardware abstraction layer.

# 3.0 Software architecture

The Advantiv software stack is divided, internally, into four layers. The intention is that the application (the top layer) only "sees" the next layer (The RX, TX and platform libraries APIs) while the reset of the system remains essentially hidden.

The four layers, from top to bottom are:

1- The application
2- The RX, TX and Platform library APIs
3- The RX and TX Hardware Abstraction Layer
4- The Platform Hardware Abstraction Layer

The Advantiv RX and TX libraries are a software layer that sits between the application and the rest of the system. The libraries are intended to serve two purposes:

1- Provide the application with a consistent set of APIs that can be used to configure RX and TX hardware without the need to use low-level register access. This makes the application portable across different revisions of the silicon or even across different hardware modules.

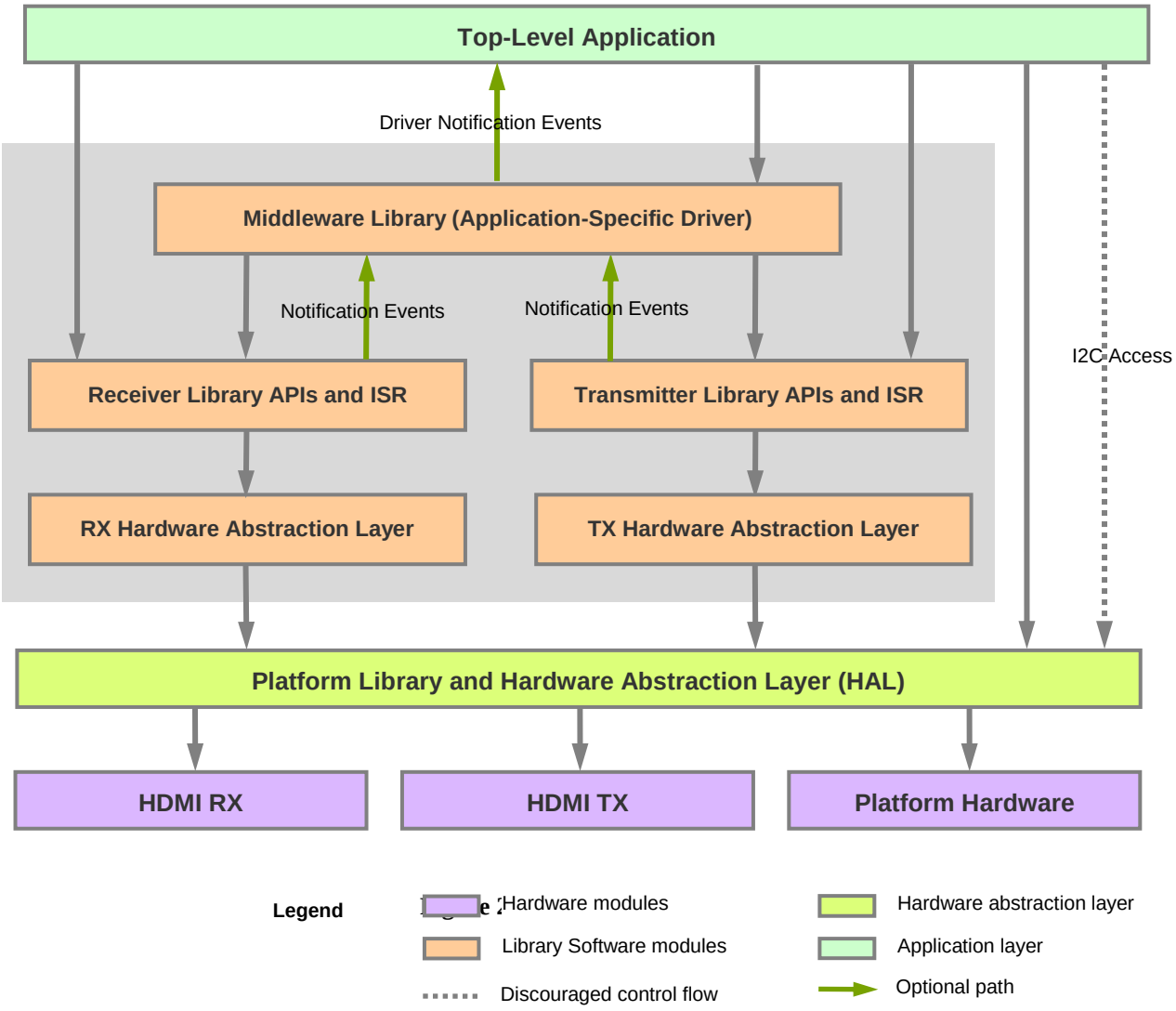
2- Provide basic services to aid the application in controlling the RX and TX modules, such as interrupt service routine, HDCP high-level control and status information.

The libraries do not, in any shape or form, alter the configuration or state of the RX or TX hardware on its own. It is the responsibility of the application, for example, to poll for status, read and parse info-frames, read and parse EDID, send EDID to the source, check for changes in operating conditions and configure the hardware accordingly. The libraries acts only as an abstraction layer between the application and the hardware.

Following is a diagram of the overall system architecture. The modules in the grey rectangle represent the software supplied by ADI that is portable across platforms. The application and Platform HAL are supplied by ADI as an example for the reference platforms.

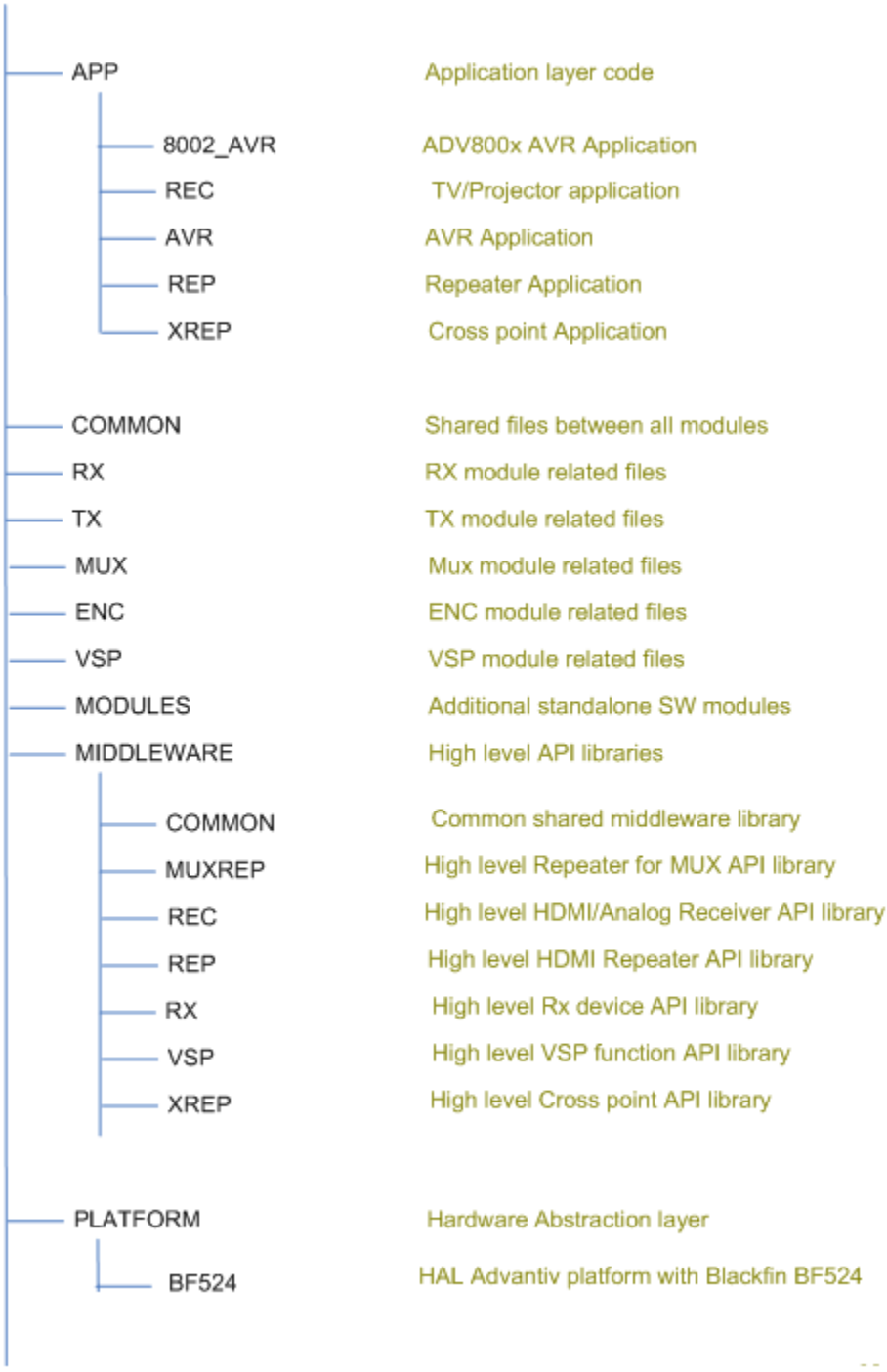
Although the application can gain direct access to the hardware using the RX/TX HAL or platform HAL I2C Read/Write functions (The dotted lines in the diagram), this practice is strongly discouraged by ADI as it breaks the layered structure of the software and reduce the application portability across silicon revisions or platforms.

Depending on application preference, the libraries and ISR can provide notification events to the application (via a pre-defined call-back function) to indicate status changes or reception of certain data packets. This notification can be enabled or disabled at compile time as described in each module specific design and API documents.


Top-Level Application
Driver Notification Events
Middleware Library (Application-Specific Driver)
Notification Events
Notification Events
I2C Access
Receiver Library APIs and ISR
Transmitter Library APIs and ISR
RX Hardware Abstraction Layer
TX Hardware Abstraction Layer
Platform Library and Hardware Abstraction Layer (HAL)
HDMI RX
HDMI TX
Platform Hardware
Legend
Hardware modules
Library Software modules
Discouraged control flow
Hardware abstraction layer
Application layer
Optional path

### *3.1 Folder structure*

The Advantiv software stack is supplied in source format. All source files are in standard ANSI C to simplify porting to any platform. `Figure 3 depicts the folder structure of the software stack.




**`Figure 3 – Folder Structure**

**/**

The root folder should be empty

**/ APP**

This folder contains different types of software applications that demonstrate the functionality of varies types of ADI devices. Each type of application is located in a sub-folder under this folder. ADI provides sample applications for TVs, DVD Players, AVRs, etc. with the option to customize application behaviour according to customer requirements. All customizations are done in the application–specific folder.

**/ APP / <Application>**

This is where each application resides under the APP folder. No part of the applications exists anywhere outside this folder. The application has no external dependencies except on RX, TX, Middleware, or Platform libraries. The application can make use of all APIs exported by the RX and TX libraries, the Middleware library, the platform library and modules under the MODULES folder. The only restriction on application usage of APIs is direct I2C access to hardware modules, for portability reasons. The structure of the <application> folder and its sub-folders is application specific. The documentation on each application structure and usage resides in a DOC folder under this folder.

It should be noted that the software structure is designed to support two types of application architectures:

1- A high-level application that make use of customizable drivers (e.g., repeater or TV driver) provided by ADI. This is the simplest and fastest way to create fully-functional user applications as demonstrated by the control flow in the system architecture diagram above.

2- A user-designed application that uses device libraries (RX and TX libraries) to build specific functionality (e.g., repeater or TV). This obviously will take more time and effort than the first approach. The user is advised to consult API libraries documents for information on interfacing with the required device APIs.

**/ COMMON**

This folder contains all files that are not architecture or platform dependant and are shared between all modules in the system. Files in this folder are typically header files with defines or data structures that are shared globally. This folder should not be viewable by the application. Any files in this folder that may contain declarations required by the application are typically included in the libraries' exported header file(s.) One of the files that reside in this folder is "atv_common.h" which contains some declarations that are shared between RX and TX modules as defined in the Library APIs section.

**/ RX**

This folder contains the RX module sub-folders tree and is the only folder in the RX tree that will be "seen" by the application. A single header file, which resides in this folder, is the only header file that needs to be included by the application to access all exported RX APIs and data structures.

For details on RX module APIs and data structures, please refer to the RX module design document and APIs specification

**/ TX**

This folder contains the TX module sub-folders tree and is the only folder in the TX tree that will be "seen" by the application. A single header file, which resides in this folder, is the only header file that needs to be included by the application to access all exported TX APIs and data structures.

For details on TX module APIs and data structures, please refer to the TX module design document and APIs specification.

/ RX

This folder contains the RX module sub-folders tree and is the only folder in the RX tree that will be "seen" by the application. A single header file, which resides in this folder, is the only header file that needs to be included by the application to access all exported RX APIs and data structures.

For details on RX module APIs and data structures, please refer to the RX module design document and APIs specification

/ ENC

This folder contains the ENC module sub-folders tree and is the only folder in the ENC tree that will be "seen" by the application. A single header file, which resides in this folder, is the only header file that needs to be included by the application to access all exported ENC APIs and data structures.

For details on ENC module APIs and data structures, please refer to the ENC module design document and APIs specification.

**/ MUX**

This folder contains the MUX module sub-folders tree and is the only folder in the MUX tree that will be "seen" by the application. A single header file, which resides in this folder, is the only header file that needs to be included by the application to access all exported MUX APIs and data structures.
For details on MUX module APIs and data structures, please refer to the MUX module design document and APIs specification.

**/ PLATFORM**

This folder contains all platform and hardware-specific code; including platform initialization, interrupt connection, timers, I2C, UART and other hardware specific routines. The Platform Hardware abstraction layer (HAL) resides in this folder. The application may use any of the Platform HAL APIs except for direct I2C access for portability reasons.

**/ PLATFORM / <Platform Name>**

The <Platform Name> folder contains the files that belong to the hardware platform in use the Advantiv Transceiver Evaluation board, other Advantiv evaluation boards or any customer-specific board.

**/ PLATFORM / <Platform Name>/ <Processor Family>**

Some ATV evaluation boards have the option to use different main processors. The <Processor Family> folder contains the files that are specific for the processor used on the evaluation platform.

**/ PLATFORM / <Platform Name> / APP**

This folder contains application related files that change depending on the platform for which a given application available in /APP is compiled. Each application which requires hardware specific files has a corresponding sub-folder under this folder.

**/ MODULES**

This folder contains software modules add-ons that can be optionally included with the main build (if the target hardware supports it). The CEC software module is an example of add-on modules, and if included, it will reside in the /MODULES/CEC folder.

**/ MIDLWARE**

This folder contains a collection of general, high-level application-layer APIs and data structures designed to provide higher level functionality for the top-level application. Part of the middleware library provide driver-like interface to the top-level application. For example, the "Midlware/REP" folder contains APIs and notification support functions designed to act as a standalone HDMI repeater driver. The "App/Rep" folder contains the top-level application that uses those APIs to configure repeater behaviour and act on notification messages generated by the driver.

## *3.2 Library APIs*

The device libraries (RX, TX, ENC, MUX, VSP) provide a comprehensive set of APIs to control, configure and provide status on all aspects of the ADI device hardware RX and TX are the most commonly used libraries and for clarity we will discuss these further in this section. All library APIs are available for the application software to use at any time.

All prototypes, macros and data structures needed to use a particular module's exported APIs are included in a header file in the module's root folder, called the module reference header file. The name of this file is specified in each module's design document and generally has the format "<module-id>_lib.h" where <module-id> is "Rx", "Tx", etc.

The module reference header file also includes any files in the /COMMON or /PLATFORM folders required to utilize the libraries. The module reference header file is the ONLY files the application needs to include to gain access to all module features and its lower level layers.

All ATV APIs are prefixed with "**ADIAPI_<module-id><API name>",** where <module-id> can be "Rx", "Tx", etc. For example:

**ADIAPI_RxGetChipId(…)**

**ADIAPI_TxGetChipId(…)**

All APIs return a value indicating its success or failure. This value is one of the ATV_ERR macro values, defined in /COMMON/"atv_common.h" and can be one of the following:

```
typedef enum
{
        ATVERR_OK,
        ATVERR_TRUE,
        ATVERR_FALSE,
        ATVERR_INV_PARM,
        ATVERR_NOT_AVAILABLE,
        ATVERR_FAILED
} ATV_ERR;
```

- **ATVERR_OK**
  Indicate that the function completed successfully.
- **ATVERR_TRUE**
  Indicate that the requested state is true
- **ATVERR_FALSE**
  Indicate that the requested state is false
- **ATVERR_INV_PARM**
  Indicate that one or more of the passed parameters are either invalid or do not apply to the current mode of operation.
- **ATVERR_NOT_AVAILABLE**
  Indicate that the requested operation can not be performed or the required information is not available.
- **ATVERR_FAILED**
  Indicate that the function failed to execute due to other reasons.

### *3.3 Driver APIs*

Drivers residing in the Middleware folder provide the same API interface as the one used for RX and TX libraries. The module ID field will indicate the driver ID.

For example:

**ADIAPI_RepMain(…)** is the repeater driver main entry point.

All other aspects of API design remain the same. Please refer to the driver-specific documents for more information on driver usage.

# 4.0System Configuration

Both the RX and TX modules can be user-configured during start-up. The configuration is application dependant and as such is controlled by the application. Please refer to RX and TX module design document and API specifications for more information on RX and TX modules compile-time and run-time configuration. Globally, the Advantiv software stack is configured through compilation switches (symbols). The following switches are used to control the compiler output for all modules in the system:

**UART_DEBUG=  1**

This switch is used to enable or disable debug output to the console. If set to 0, debug output will be disabled and all debug messages will be removed from the code, thus reducing code size. If set to 1, debug messages from the software will be sent to the console (using the system's printf or equivalent function) via the debug macros defined in Platform-specific macros section.

# 5.0 Platform Hardware Abstraction Layer

The Platform Hardware Abstraction Layer (HAL) resides in the platform folder. The Hardware Abstraction Layer is platform-dependant and does not change if the RX or TX hardware modules change.

Each platform has a subfolder under the /PLATFORM folder in the main tree that contains platform-specific implementation of all the HAL APIs.

The platform used for Advantiv software stack is the Advantiv video evaluation board using the Blackfin processor. Further platforms can be defined in subfolders under the /PLATFORM folder.

The HAL is a collection of APIs, macros and defines designed to make the upper layers (Libraries and application) platform-independent as much as possible. The following is a description of HAL components.

## *5.1 Platform-specific macros*

### 5.1.1 Data types

Data types are platform and architecture dependant. The data types used by the software are defined in the "atv_types.h" header file located in the platform-specifc folder.

Should the platform use different data types, the macros defined in "atv_types.h" must be replaced with the appropriate macros specific to the application and/or platform. The following are the types defined in "atv_types.h" along with the associated size.

| **Type** | **Size** |
|---|---|
| | |
| UINT8, UCHAR | 8-bit unsigned integer |
| UINT16 | 16-bit unsigned integer |
| UINT32 | 32-bit unsigned integer |
| CHAR | 8-bit signed integer |
| INT16 | 16-bit signed integer |
| INT32 | 32-bit signed integer |
| BOOL | Platform dependant. Mostly 8-bit unsigned integer but this should not be assumed. |
| TRUE | Non-zero |
| FALSE | 0 |
| NULL | ((void *)0) |

### 5.1.2 Defines and Macros

Platform-dependant defines and macros are grouped together in several files in the platform-specific folder.

The file "platform_config.h" in each platform-specific folder contains any macros or defines that are platform-dependant. For example, functions that send debug messages to the console should be abstracted in this file since printf is a platform-dependant implementation.
The following macro should be used to send debug messages to the console:

**#define DBG_MSG** /* platform's printf equivalent function */

The compilation switch “UART_DEBUG” is used to enable/disable the UART debug output and is described in the System Configuration section.

The file “atv_tal.h” contains the tool-chain abstraction layer. For ease of portability, all compiler directives should be defined in this file and used by all software layers:

```
#define STATIC      static
#define INLINE      inline
#define CONSTANT    const
#define EXTERNAL    extern
```

The file “atv_osal.h” contains the OS abstraction layer. Normally, this file will include C library headers required to support functions used by the software such as memcpy and strlen. In case those functions are not supported by the compiler in use, the prototypes should be defined in this file and the implementation should be made in the platform-specific folder.

## *5.2 HAL APIs*

The HAL layer exports APIs to provide direct access to the hardware such as I2C access, interrupt detection, timers or any other hardware-specific implementation. All HAL functions are prefixed by “HAL_” and are listed in the header file “platform.h” in the /PLATFORM folder along with all HAL data structures and macros needed by upper-level layers. “platform.h” is also included in each module reference header file and does not need to be explicitly included by the application.

When porting to a different platform, as a minimum, the I2C read/write functions and the timing functions must be implemented as they are used by the library to access the hardware. All other HAL APIs are used by the application layer and are never called from the library and thus can be omitted if the application layer is not used.

Following is a list of the main HAL APIs.

### 5.2.1 HAL_I2CReadByte

***Description***

Read one I2C byte and return number of bytes read.

***Synopsis***

***#include "platform.h"***

UCHAR HAL_I2CReadByte (UCHAR Device, UCHAR Register, UCHAR *Data)

***Parameters***

*Device*
I2C device address

*Register*
Register offset within I2C device

*Data*
Pointer to receive the read byte

***Return value***

1 on success, 0 on failure

***Remarks***

None

### 5.2.2 HAL_I2CWriteByte

***Description***

Write one I2C byte and return number of bytes written.

***Synopsis***

***#include "platform.h"***

UCHAR HAL_I2CWriteByte (UCHAR Device, UCHAR Register, UCHAR Data)

***Parameters***

*Device*
I2C device address

*Register*
Register offset within I2C device

*Data*
Byte to write

***Return value***

1 on success, 0 on failure

***Remarks***

None

### 5.2.3 HAL_I2CReadBlock

***Description***

Read I2C bytes and return number of bytes read.

***Synopsis***

***#include "platform.h"***

UINT16 HAL_I2CReadBlock (UCHAR Device, UCHAR Register, UCHAR *Data,
UINT16 NofBytes)

***Parameters***

*Device*
I2C device address

*Register*
Starting register offset within I2C device

*Data*
Pointer to receive the read bytes

*NofBytes*
Number of bytes to read

***Return value***

Number of bytes read.

***Remarks***

None

### 5.2.4 HAL_I2CWriteBlock

***Description***

Write I2C bytes and return number of bytes written.

***Synopsis***

***#include "platform.h"***

UINT16 HAL_I2CWriteBlock (UCHAR Device, UCHAR Register, UCHAR *Data,
UINT16 NofBytes)

***Parameters***

*Device*
I2C device address

*Register*
Starting register offset within I2C device

*Data*
Pointer to the bytes to be written

*NofBytes*
Number of bytes to write

***Return value***

Number of bytes written.

***Remarks***

None

### 5.2.5 HAL_I2CReadAddr16Block8

***Description***

Read I2C bytes from 16-bit address and return number of bytes read. This function depends on the target device in use. It may not need to be implemented for some devices.

***Synopsis***

```
#include "platform.h"

UINT16 HAL_I2CReadAddr16Block8 (UCHAR Device, UINT16 Register, UCHAR *Data,
                                UINT16 NofBytes)
```

***Parameters***

*Device*
I2C device address

*Register*
16-bit register address

*Data*
Pointer to receive the read bytes

*NofBytes*
Number of bytes to read

***Return value***

Number of bytes read.

***Remarks***

None

### 5.2.6 HAL_I2CWriteAddr16Block8

***Description***

Write I2C bytes to a 16-bit address and return number of bytes written. This function depends on the target device in use. It may not need to be implemented for some devices.

***Synopsis***

***#include "platform.h"***

UINT16 HAL_I2CWriteAddr16Block8 (UCHAR Device, UINT16 Register, UCHAR *Data,
UINT16 NofBytes)

***Parameters***

*Device*
I2C device address

*Register*
16-bit register address

*Data*
Pointer to the bytes to be written

*NofBytes*
Number of bytes to write

***Return value***

Number of bytes written.

***Remarks***

None

### 5.2.7 HAL_I2CReadAddr16Block16

***Description***

Read I2C 16-bit words from 16-bit address and return number of words read. This function depends on the target device in use. It may not need to be implemented for some devices.

***Synopsis***

***#include "platform.h"***

UINT16 HAL_I2CReadAddr16Block8 (UCHAR Device, UINT16 Register, UINT16 *Data,
UINT16 NofWords)

***Parameters***

*Device*
I2C device address

*Register*
16-bit register address

*Data*
Pointer to receive the read words

*NofWords*
Number of words to read

***Return value***

Number of 16-bit words read

***Remarks***

None

### 5.2.8 HAL_I2CWriteAddr16Block16

***Description***

Write 16-bit I2C words to 16-bit I2C address and return number of words written. This function depends on the target device in use. It may not need to be implemented for some devices.

***Synopsis***

```
#include "platform.h"

UINT16 HAL_I2CWriteAddr16Block16 (UCHAR Device, UINT16 Register,
                                  UINT16 *Data, UINT16 NofWords)
```

***Parameters***

*Device*
I2C device address

*Register*
16-bit register address

*Data*
Pointer to the words to be written

*NofWords*
Number of 16-bit words to write

***Return value***

Number of words written

***Remarks***

None

### 5.2.9 HAL_I2CGenericAccess

***Description***

This function provides generic access to I2C controller to read/write any combination of register address length and register size. This function depends on the target device in use. It may not need to be implemented for some devices.

***Synopsis***

***#include "platform.h"***

UINT16 HAL_I2CGenericAccess (UCHAR Device, UINT16 WriteCount,
UCHAR *WriteData, UINT16 ReadCount, UCHAR *ReadData)

***Parameters***

*Device*
I2C device address

*WrtiteCount*
Number of bytes (in TxData) to write

*WriteData*
Pointer to the buffer of bytes to be written

*ReadCount*
Number of bytes to read into RxData

*ReadData*
Pointer to a buffer to store the read bytes

***Return value***

0 on success (All specified bytes were read and/or written)
1 on failure

***Remarks***

None

### 5.2.10 HAL_DelayMs

***Description***

Suspend execution by the specified number of milliseconds.

***Synopsis***

***#include "platform.h"***

**void** HAL_DelayMs (UINT16 Counter)

***Parameters***

*Counter*
Required delay in milliseconds.

***Return value***

None

***Remarks***

None

### 5.2.11 HAL_GetCurrentMsCount

***Description***

Get the current value of a free running milliseconds counter.

***Synopsis***

***#include "platform.h"***

UINT32 HAL_GetCurrentMsCount (**void**)

***Parameters***

*None*

***Return value***

The return value is the current count of a free-running milliseconds counter. If the counter reaches 0xffffffff, it wraps to 0.

***Remarks***

None

### 5.2.12 HAL_GetUartByte

***Description***

Read a character from the console.

***Synopsis***

***#include "platform.h"***

BOOL HAL_GetUartByte (UCHAR *UartCh)

***Parameters***

*UartCh*
Pointer to receive read byte from the console.

***Return value***

The return value is TRUE if a byte was available and FALSE if no byte is available. If a new byte is available, it will be removed from the console buffer and returned in UartCh.

***Remarks***

This is a non-blocking function. It returns immediately regardless if a byte is available at the console or not.

### 5.2.13 HAL_SendUartByte

***Description***

Write a character to the console.

***Synopsis***

***#include "platform.h"***

BOOL HAL_SendUartByte (UCHAR UartCh)

***Parameters***

*UartCh*
Byte to be sent to the console.

***Return value***

The return value is TRUE if the byte was sent and FALSE if the byte could not be sent.

***Remarks***

This is a non-blocking function. It returns immediately regardless if the byte was sent or not.

### 5.2.14 HAL_RxInt1Pending

***Description***

Read the state of HDMI RX interrupt 1 line. HDMI RX has two interrupt lines, INT1 and INT2. Depending on which interrupt line is used as the RX interrupt, this function may not need to be implemented.

***Synopsis***

***#include "platform.h"***

BOOL HAL_RxInt1Pending (**void**)

***Parameters***

*None*

***Return value***

The return value is TRUE if HDMI RX Interrupt 1 line is asserted (interrupt is pending) and FALSE if HDMI TX interrupt line is not asserted.

***Remarks***

None

### 5.2.15 HAL_RxInt2Pending

***Description***

Read the state of HDMI RX interrupt 2 line. HDMI RX has two interrupt lines, INT1 and INT2. Depending on which interrupt line is used as the RX interrupt, this function may not need to be implemented.

***Synopsis***

***#include "platform.h"***

BOOL HAL_RxInt2Pending (**void**)

***Parameters***

*None*

***Return value***

The return value is TRUE if HDMI RX Interrupt 2 line is asserted (interrupt is pending) and FALSE if HDMI TX interrupt line is not asserted.

***Remarks***

None

### 5.2.16 HAL_TxIntPending

***Description***

Read the state of HDMI TX interrupt line.

***Synopsis***

***#include "platform.h"***

BOOL HAL_TxIntPending (**void**)

***Parameters***

*None*

***Return value***

The return value is TRUE if HDMI TX interrupt line is asserted (interrupt is pending) and FALSE if HDMI TX interrupt line is not asserted.

***Remarks***

None

### 5.2.17 HAL_PlatformInit

***Description***

Initialize the hardware platform and bring it to a known state.

***Synopsis***

***#include "platform.h"***

**void** HAL_PlatformInit (**void**)

***Parameters***

*None*

***Return value***

None

***Remarks***

This function is normally called once during application initialization.

### 5.2.18 HAL_AssertRxHPD

***Description***

Set Hot Plug Detect state for selected HDMI RX port. This function uses the hardware platform to control HPD signal for RX devices that can not control the HPD signal internally. For RX devices with built-in HPD control, this function can be ignored.

***Synopsis***

```
#include "platform.h"

void HAL_AssertHPD (UCHAR PortIndex, BOOL Assert)
```

***Parameters***

*PorIndext*

Specify the HDMI RX port index (0-3) to change the state of its Hot-Plug-Detect signal. The value ALL_PORTS indicate that the required change in HPD state applies to all RX ports.

*Assert*

This value indicates the required state of Hot Plug Detect for the selected port.
Set to TRUE to turn HPD on, and FALSE to turn HPD off.

***Return value***

None

***Remarks***

### 5.2.19 HAL_IsRxHPDOn

***Description***

Return the current state of HDMI RX port Hot Plug Detect signal. This function uses the hardware platform to return the status of RX HPD signal for RX devices that can not control the HPD signal internally. For RX devices with built-in HPD control, this function can be ignored.

***Synopsis***

***#include "platform.h"***

BOOL HAL_IsRxHPDOn (UCHAR PortIndex)

***Parameters***

*PortIndex*
Specify the HDMI RX port index (0-3) to return the state of its Hot-Plug-Detect signal.

***Return value***

The state of Hot Plug Detect signal for the selected port. Returns TRUE if HPD is asserted and FALSE if HPD is not asserted.

***Remarks***

### 5.2.20 HAL_Rx5VDetected

***Description***

Return the current state of HDMI RX port +5 volts signal. This function uses the hardware platform to return the status of RX cable-detect (+5 volts) signal for RX devices that can not sense the cable-detect signal internally. For RX devices with built-in cable-detect sensing, this function can be ignored.

***Synopsis***

***#include "platform.h"***

BOOL HAL_Rx5VDetected (UCHAR PortIdx)

***Parameters***

*Port*

The HDMI RX port index (0-3) to return the state of it's +5 Volts signal.

***Return value***

The state of the +5 volts signal for the selected HDMI port. Returns TRUE if +5 volts is asserted (high) and FALSE if +5 volts is not asserted (low).

***Remarks***

### 5.2.21 HAL_GetPlatformRevision

***Description***

Return the revision numbers of the hardware platform.

***Synopsis***

***#include "platform.h"***

**void** HAL_GetHwRevisions (UINT16 *HwRev)

***Parameters***

*HwRev*
This pointer will receive the revision number of the hardware platform.

***Return value***

None

***Remarks***