AVL DVBSx
Software Development Kit (SDK)
User's Guide

www.availink.com

Availink Inc.
Scotia Centre
P. O. Box 268GT
George Town, Grand Cayman,
Cayman Islands

Availink (China) Inc.
Xueyuan International Tower,
Ste 1608
No. 1 Zhichun Road
Haidian District, Beijing,
100083
P. R. China

Availink (US) Inc.
20201 Century Blvd., STE 160
Germantown MD 20874 USA

Revision History

The User's Guide revision history for the past several revisions is listed in the table below. The version number consists of three fields separated by dots. In the order of appearance, they indicate different levels of significance in the revision. The Software Development Kit (SDK) consists of three fields as well. For a compatible user's guide, the first two fields must match those of the SDK. The third field in the version number of the User's Guide represents a revision that does not result from the revision of the SDK, such as typo fix, further clarification etc. Vice versa, the third field of the version number of the SDK represents a revision that does not require a revision of the User's Guide.

Date	Version	Revision
2010/2/26	1.0.0	Initial release.
2010/5/11	1.0.1	Corrected function descriptions, patch file name and grammatical mistakes.
2010/5/21	1.0.2	Changed patch file references added information about a dual channel AVL DVBSx device and added information about installing part specific firmware patch.

Availink AVL DVBSx SDK User's Guide
Copyright © 2010 by Availink Inc.

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

The AVL DVBSx Software Development Kit (SDK) provides an application programming interface (API) for the Availink® DVB-S2/DVB-S devices specified in the release notes that accompany this SDK. The API functions are used to initialize and to control the AVL DVBSx IC within a set-top box (STB) design environment. This user's guide provides step-by-step integration instructions and a reference for the API functions. It includes API function specifications, implementations, and examples to illustrate their usage.

Section 1.1 below provides an overview of the AVL DVBSx device. Section 1.2 is an overview of the SDK that includes a description of the software layers and the tuner implementation approach. Users who are familiar with these sections may proceed to Section 2, which describes the SDK installation instructions. Section 3 presents the recommended development flow with additional detail provided in Sections 4 and 5. The user should be able to perform the integration by following these steps and referencing other sections as needed during the development. Example code is provided in the SDK and is described in Section 8.

Section 6 provides general API design principles including naming conventions, control flow options, and resource sharing approaches. Section 7 provides a complete API reference, including all of the API definitions and data structures.

1.1 AVL DVBSx DEVICE OVERVIEW

1.1.1 SINGLE CHANNEL DEVICE:

Figure 1-1 illustrates a single channel AVL DVBSx device in a typical STB application.

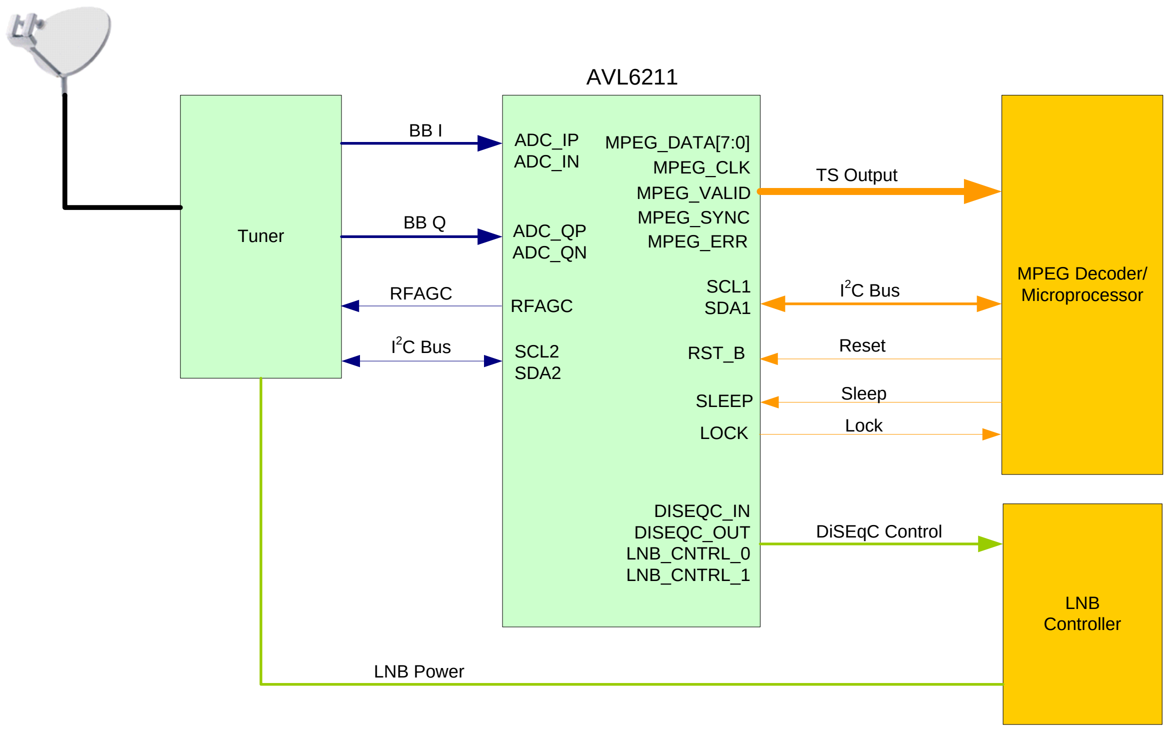

Figure 1-1: Typical Single Channel AVL DVBSx Device STB Application

The AVL DVBSx single channel device receives an analog IQ signal from a tuner, demodulates and decodes the DVB-S2 or DVB-S signal, and outputs a standard MPEG transport stream. It is configured and controlled through a standard two wire (I^2C) bus.

1.1.2 DUAL CHANNEL DEVICE:

Figure 1-2 illustrates a dual channel AVL DVBSx device in a typical STB application. The AVL DVBSx dual channel device consists of two channels, Channel A and Channel B. It receives two analog IQ signals from two tuners, demodulates and decodes them using its two channels and outputs dual standard MPEG transport stream to two MPEG decoders. The dual channel device could be configured and controlled through a single or dual two wire (I^2C) bus. Special care is required while using a single I^2C bus to ensure that the slave addresses of the two channels are different. However, the slave addresses could be same if two I^2C buses are used instead.

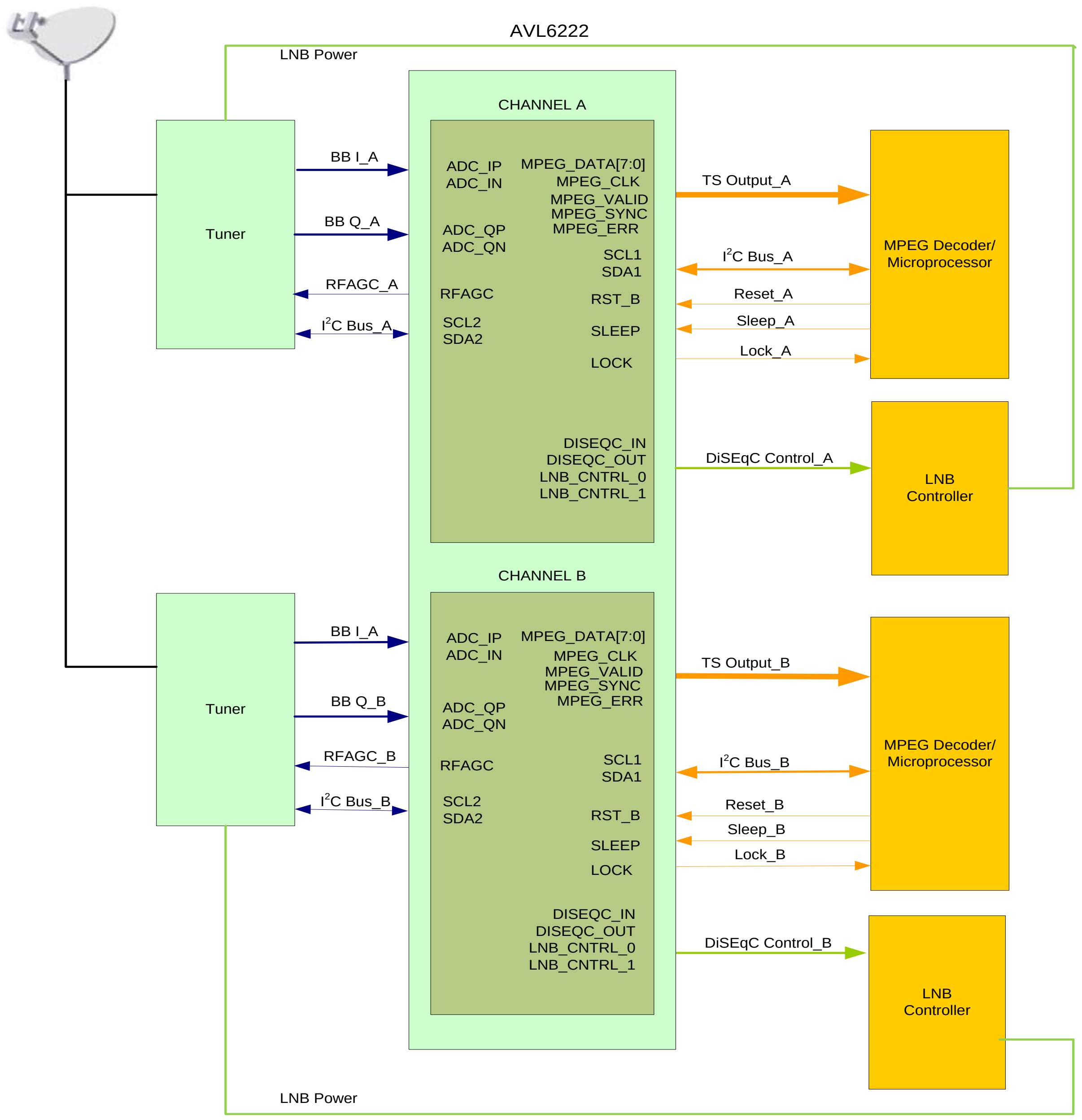

Figure 1-2: Typical Dual Channel AVL DVBSx Device STB Application

To simplify STB integration, a typical AVL DVBSx device also provides the following interface functions:

- I2C master (Repeater) to control an external tuner (Tuner Control Interface)
- Diseqc™ 2.0 port (Diseqc™ 2.0 Control Interface)
- Two Low Noise Block (LNB) control pins in case of a single channel device
- Four Low Noise Block (LNB) control pins in case of a dual channel device

For more details on the AVL DVBSx device, the user can consult Reference 1 and Reference 3.

1.2 SDK OVERVIEW

The AVL DVBSx SDK is designed to work in a variety of embedded environments. Figure 1-3 illustrates how the AVL DVBSx SDK interfaces with the STB application and subsequent layers. The STB application layer has the option of communicating with the AVL DVBSx SDK interfaces or it can directly interface to the STB hardware. The AVL DVBSx SDK interfaces communicate with the Board Support Package (BSP), that in turn communicates with the STB hardware layer.

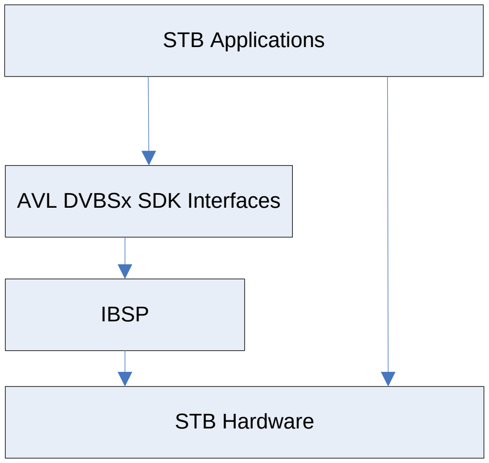

Figure 1-3: AVL DVBSx SDK User's Environment

The BSP layer provides a level of abstraction between the AVL DVBSx SDK interfaces and the STB Hardware. The BSP layer straddles the software – hardware boundary. The STB Hardware consists of the I2C controller, I2C bus, the AVL DVBSx ICs and the RF Tuner IC. Other STB hardware not related to the satellite signal processing is not relevant to this discussion.

Figure 1-4 illustrates the functional dependencies between the various software layers within the SDK. For example, the STB application calls functions in the IRx interface directly. The IRx interface layer calls functions in IBase and II2C, which calls functions in the IBSP layer that finally controls the I2C bus. As noted previously, the IBSP acts as a hardware abstraction layer to the API, and must be implemented by the User. An example and a blank BSP template is provided in the SDK.

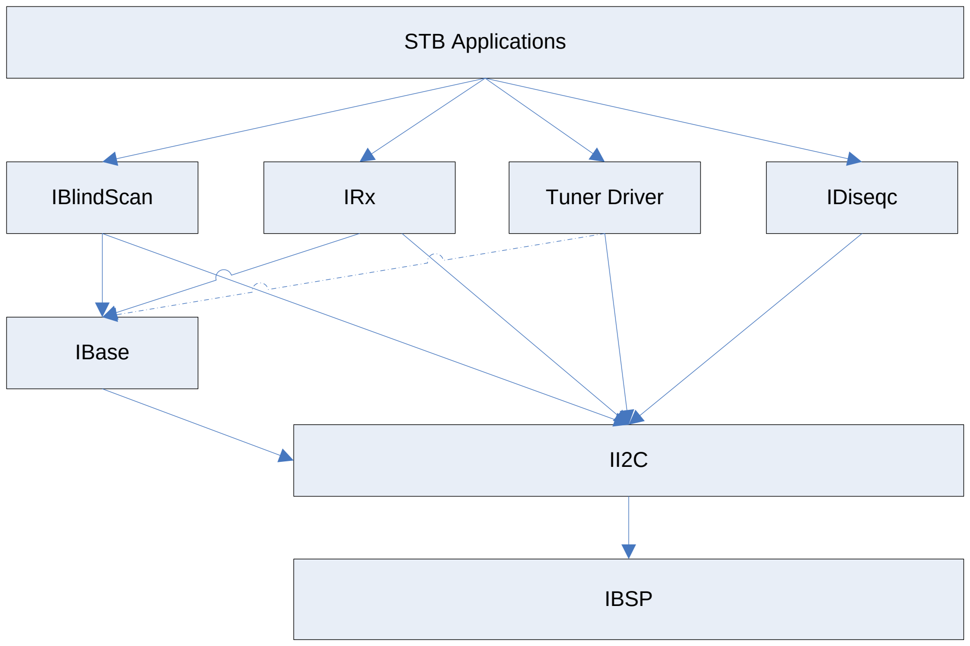

Figure 1-4: AVL DVBSx SDK Layer functional dependencies

A brief description of the software layers is provided here. A more detailed description is provided in later sections.

- IBase
  The Base software layer provides basic initialization, control, and information access capabilities for the AVL DVBSx device.

- IBlindScan
  The BlindScan software layer provides functions to control the blind scan process and to read the blind scan status and results.

- IDiseqc
  The Diseqc layer provides access to the DiSEqC™2.0 port on the AVL DVBSx device. The layer is responsible for LNB control and for managing the DiSEqC communication protocol.

- IRx
  The Receiver software layer provides access to the receiver portion of the AVL DVBSx device.

- II2C
  The I2C software layer is a layer internal to the SDK that enforces atomic access to the I2C device and implements the I2C protocol for the AVL DVBSx device. This layer manages endianess across different systems and allows programmers to work in their native endianess.

- Tuner Driver
  The Tuner Driver software layer implements the tuner control interface. It is comprised of a tuner driver that is implemented by code executed in the STB application. For user convenience, Availink provides tuner drivers for a variety of tuner IC vendors. The user is encouraged to contact Availink to verify whether a

tuner driver exists for your tuner IC. To facilitate tuner driver development, Availink provides a software interface called ITuner which contains tuner driver helper functions as well as a software interface called II2CRepeater which provides control of the I2C repeater hardware within the AVL DVBSx device.

- II2CRepeater
  The I2C Repeater software layer is used by the tuner driver layer to communicate with the I2C repeater hardware within the AVL DVBSx device.

- IBSP
  The board support package software layer abstracts the hardware from the SDK. It also provides locking mechanisms for resource sharing and multithread safety. A template is provided that contains function stubs that must be implemented by the user. An example implementation is also provided for reference.

1.3 AVL DVBSx Device Functional Modes

The AVL DVBSx device provides two functional modes, demodulator mode and blind scan mode. In demodulator mode, the AVL DVBSx device is capable of performing the demodulator signal processing needed to execute channel lock operations. In blind scan mode, the AVL DVBSx device is capable of performing the signal processing needed to execute blind scan operations.

After booting the AVL DVBSx chip, the user may place the device into either blind scan mode or demodulator mode by calling the SDK function IBase_SetFunctionalMode. The user may switch the device between these two modes through subsequent calls to the IBase_SetFunctionalMode function.

Many SDK functions can be called regardless of whether the AVL DVBSx device is in the demodulator functional mode or the blind scan functional mode. Some SDK functions are specific to a particular functional mode. Among these functions, the SDK restricts the user from calling those that may cause unpredictable behavior if the AVL DVBSx device is in the wrong functional mode. For these functions, the SDK returns the error code AVL_DVBSx_EC_GeneralFail if the AVL DVBSx device is not in the proper functional mode for the function call.

In general, the user is advised to call only those functions that are appropriate for the present functional mode of the AVL DVBSx device. Figure 1-5 summarizes the function calls that are appropriate for each functional mode of the AVL DVBSx device. If all of the functions in an SDK interface may be called from a given functional mode, then only the SDK interface name is shown. If some functions within an interface are appropriate for one functional mode while other functions in the interface are not appropriate for that mode, then the interface name and function name are shown in the figure.

For example, it is recommended that all IBlindScan functions only be called when the AVL DVBSx device is operating in the IBlindScan functional mode. Thus, the IBlindScan interface is shown inside the blind scan functional mode circle Similarly, all functions in the IBase, IBSP, IDISEQC, II2C, II2CRepeater, and ITuner interfaces may be called when the AVL DVBSx device is in either the blind scan or demodulator functional mode. Thus, these interfaces are shown in the intersection of the blind scan and demodulator functional mode circles. Some functions in the IRx interface may be called in either blind scan or demodulator mode. Other IRx functions may only be called when the AVL DVBSx device is in demodulator mode. Thus, in the figure, some of the IRx functions are shown in the intersection between the blindscan and demodulator functional mode circles, while the remainder of the IRx functions are shown only in the demodulator functional mode circle.

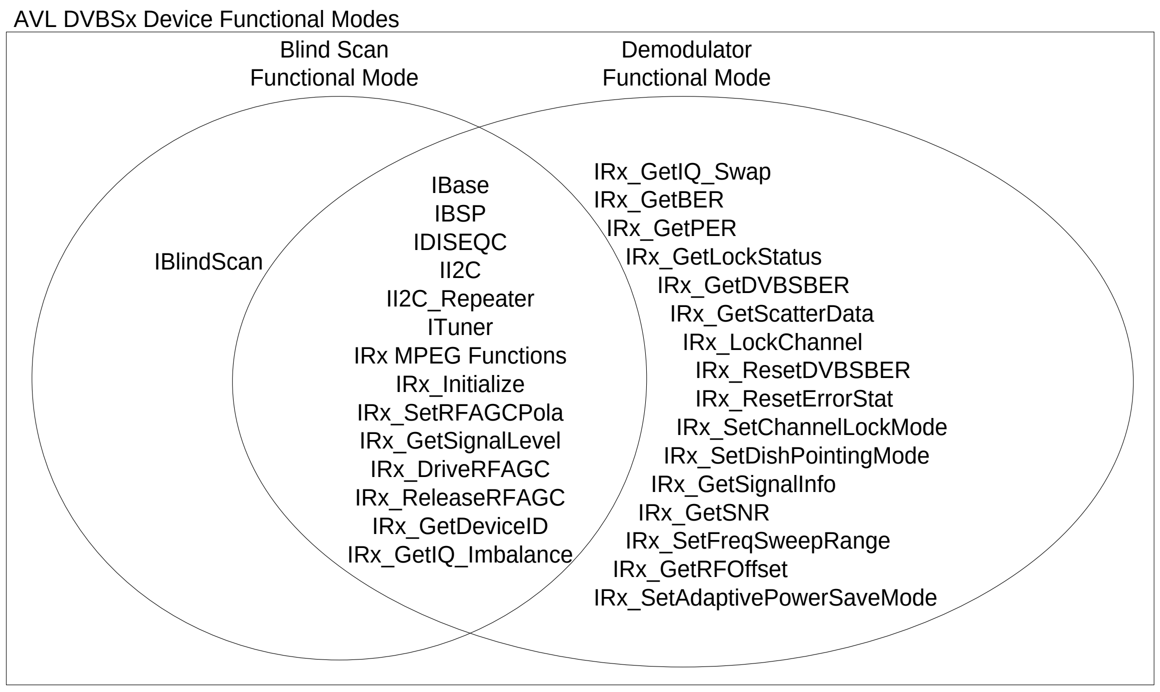

Figure 1-5: Required SDK Function Usage for each Functional Mode of AVL DVBSx Device

2 SDK INSTALLATION and ORGANIZATION

The SDK is delivered in ANSI C source code format, which provides maximum installation flexibility in a variety of operating environments. The user can install it to a Microsoft Windows system by running the provided setup program **AVL_DVBSx_SDK.exe**. For other operating systems, the user can extract the **AVL_DVBSx_SDK.zip** archive file to a desired location in the user's file system. In this guide, the SDK installation directory will be referred to as **installation_dir**.

After installation, the directory tree looks like this:

```
installation_dir
│
│  ReleaseNotes_SDK.txt
│  AVL_DVBSx_SDKUsersGuide.pdf
│
├─addon
│     ExtSharpBS2S7HZ6306.c
│     ExtSharpBS2S7HZ6306.h
│     SharpBS2S7HZ6306.h
│
├─AVL_DVBSx
│  ├─include
│ │     avl_dvbsx.h
│ │     avl_dvbsx_globals.h
│ │     avl.h
│ │     IBase.h
│ │     IBlindScan.h
│ │     IDiseqc.h
│ │     IRx.h
│ │     II2C.h
│ │     II2CRepeater.h
│ │     ITuner.h
│ │
│ └─src
│ │     avl_dvbsx.c
│ │     avl.c
│ │     IBase.c
│ │     IBlindScan.c
│ │     IDiseqc.c
│ │     IRx.c
│ │     II2C.c
│ │     II2CRepeater.c
│ │     ITuner.c
│
├─BSP
│ └─aardvark.net
│ │     bspdatadef.h
│ │     IBSP.c
│ │     IBSP.h
│ │
│ └─template
│ │     bspdatadef.h
│ │     IBSP.c
│ │     IBSP.h
│
└─examples (See installation for full list)
│ └─bin
│ └─blind_scan
│ └─Diseqc
│ └─LockSignal
```

In order for the installed SDK to be completely functional, a part specific firmware patch is required and supplied with each AVL DVBSx product. It is typically installed in the AVL_DVBSx_SDK directory using the part specific firmware patch installer provided with the SDK. In the following instructions the patch file is referred to as PARTNUMBER_patch.dat where PARTNUMBER would be AVL6211 or other DVBSx family device part number.

To uninstall the AVL DVBSx SDK and the patch completely from the user's system, it is required to uninstall the firmware patch first followed by uninstalling the SDK.

Table 2-1 summarizes the contents of the SDK.

Table 2-1: SDK Contents Summary

Folder/File	**Description**
addon/	Files under this directory are provided as helpers. For example, the tuner driver for the Sharp BS2S7HZ6306 tuner is provided. The user may use it directly or as an example of how to implement a tuner driver.
AVL_DVBSx/	Files under this directory implement the core API functions of the AVL DVBSx SDK. The file names are designed to indicate their contents. For example, IBase.c implements the interface `IBase`. Users in general do

Folder/File	**Description**
	not need to modify files in this directory when porting the SDK to their platform.
BSP/	The files under this directory are BSP template files which users use to create customized versions of the BSP layer tailored to their specific I2C hardware.
AVL_DVBSx_SDKUsersGuide.pdf	This user's guide.
examples	The examples directory contains example code and applications and is frequently being expanded. Please review the installation area for a complete list of the examples.

3 DEVELOPMENT FLOW

To integrate the AVL DVBSx SDK into an STB application the user must complete three tasks. First, the Board Support Package (BSP) layer named IBSP must be implemented based on the user's environment. Second, a tuner driver for the specific tuner must be prepared. Third, function calls to the Application Programming Interface (API) must be inserted into the STB application. To accomplish these tasks, Availink® recommends the following development flow:

1. Set up the hardware and the software development tool chain.

2. Install the AVL DVBSx SDK as described in the previous section.

3. Install the firmware patch data for the specific AVL DVBSx device as described in the previous section.

4. Configure the tool chain path variable or include directory to include the **installation_dir/AVL_DVBSx/include** directory, **installation_dir/addon** directory and the **installation_dir/BSP** directory (**IBSP.h**).

5. Implement the Board Support Package (BSP) layer. Specifically, define user-specific data types in **bspdatadef.h** and macros in the **IBSP.h** file and implement the IBSP interface in the **IBSP.c** file. Place the created files in the

installation_dir/BSP directory. Blank templates for these files are found in the **installation_dir/BSP/templates** directory. An example BSP used in the EVK GUI application for an AVL DVBSx device is provided in the **installation_dir/aardvark.net** directory. Section 4 has detailed instructions on how to implement a BSP.

6. Implement a call to the function AVL_DVBSx_IBase_GetVersion to check basic operation of the BSP and the I2C interface.

7. Incorporate the contents of the initialization file for the specific AVL DVBSx device **installation_dir/PARTNUMBER_patch.dat** into the STB application. For instance, the user can write the contents of the **PARTNUMBER_patch.dat** file to a flash memory. Then, whenever the user calls AVL_DVBSx_IBase_Initialize function, the user can pass the starting address of the file contents to the function. The source file, **PARTNUMBER_patch.dat.c** gets installed by the part specific patch installer in the SDK directory to show how to define global arrays to hold the contents of **PARTNUMBER_patch.dat** .

8. Initialize the AVL DVBSx device by calling the function AVL_DVBSx_IBase_Initialize and passing the contents of **PARTNUMBER_patch.dat** as an input parameter to the function in order to download the firmware patch to the AVL DVBSx device for demod and blind scan operations..

9. Insert calls to the desired API functions in the application to initialize and control the AVL DVBSx device and its components. There is example source code provided in the **installation_dir/examples** directory that covers most common tasks. In order to put the AVL DVBSx chip in a desired functional mode or change its functional mode from demodulator mode to blind scan mode or vice versa, call the function AVL_DVBSx_IBase_SetFunctionalMode, passing the appropriate functional mode variable defined as an enum in the SDK header file avl_dvbsx.h.

10. Run the integrated application.

4 IMPLEMENTING THE BSP LAYER

This section describes the implementation of the BSP layer, a critical part of STB application integration. To perform this task, the user must implement the `IBSP` functions in accordance with the STB hardware and its native compiler.

A set of template files and a reference BSP implementation are included in the BSP directory of the SDK. The files in the aardvark.net directory are an example of an implementation used in an evaluation kit such as AVL6211 EVK which performs I2C operations through an Aardvark USB to I2C adapter. The files in the template directory are empty templates for the BSP. The bspdatadef.h file is where the basic data types are defined.

In bspdatadef.h, all of the basic data type definitions must be modified according to the user's compiler. Uncomment the line #define AVL_CPLUSPLUS if the user is using a C++ compiler. Every data type has two definitions: one is a value definition and another is a reference definition. Depending on the development environment, the reference definition could be a pointer type or a reference type.

In IBSP.h, the macros listed in Table 4-1 must be modified according to the hardware and compiler.

Table 4-1: User Defined IBSP.h Macros

Macro Name	Description
MAX_II2C_READ_SIZE	This parameter sets the maximum number of bytes that the STB's I2C bus can read in one transaction.
MAX_II2C_Write_SIZE	This parameter sets the maximum number of bytes that the STB's I2C bus can write in one transaction. It **must** be greater than 5.

The data types defined in bspdatadef.h must be set to match the sizes used by the user's compiler. In a multi-threaded environment, AVL_semaphore and AVL_psemaphore must be defined with the correct data type. For example, the AVL_sempahore can be defined as a mutex or as a semaphore data type. In a single thread environment, these semaphores are not in use, so the user can use any default data type. For example, the AVL_semaphore can be defined as an unsigned char data type.

When modifying the IBSP.c file to implement the BSP functions, consider the following important points:

- The I2C address that the API passes to the BSP is a 7 bit address. In `AVL_DVBSx_IBSP_I2CRead` and `AVL_DVBSx_IBSP_I2CWrite` the address must be shifted one bit to the left and have the read/write bit set in the LSB. For example, 0x0C becomes 0x18 for a write and 0x19 for a read.
- Reading data from the AVL DVBSx device actually requires a write followed by a read (set address then read), the II2C API layer

handles this behavior. Thus, AVL_DVBSx_IBSP_I2CRead must perform ONLY an I2C read.

- The implementation of BSP I2C write/read functions must follow the protocol denoted by Figure 4-1. In other words, Figure 4-1 defines what should appear on the I2C bus when BSP I2C write/read functions are called.

KEY

☐	From Master to Slave
☐	From Slave to Master

SYMBOLS:

A – Acknowledge (SDA LOW)
N – Not Acknowledge (SDA HIGH)
Sr – Repeated Start Condition
S – START Condition
P – STOP Condition
R – R/W bit, 1=Read

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_I2CWrite(  AVL_uchar ucSlaveAddr,
AVL_puchar pucBuff,  AVL_puint16  puiSize )
```

S	ucSlaveAddr	R/ Wb	A	pucBuff[0]	A	pucBuff[1]	A	• • •	pucBuff[*puiSize-1]	P
	7 bits	0	1-bit	8 bits	1-bit	8 bits	1-bit		8 bits	

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_I2CRead(  AVL_uchar ucSlaveAddr,  AVL_puchar
pucBuff,  AVL_puint16  puiSize )
```

S	ucSlaveAddr	R/ Wb	A	pucBuff[0]	A	pucBuff[1]	A	• • •	pucBuff[*puiSize-1]	N	P
	7 bits	1	1-bit	8 bits	1-bit	8 bits	1-bit		8 bits		

Figure 4-1: I2C communication protocol

- API functions assume that the I2C operations in the BSP are atomic and thread-safe. I2C functions should not return until the I2C transaction has completed. Furthermore, in a multi-threaded environment, an interlock (mutex) must exist between the AVL DVBSx SDK and other host application functions. This interlock mechanism is required to prevent the AVL DVBSx SDK and the host application software from corrupting each other's I2C transactions.

- `AVL_DVBSx_IBSP_WaitSemaphore` and `AVL_DVBSx_IBSP_ReleaseSemaphore` must be implemented atomically. i.e., they must be uninterruptible, even across threads.

In a multi-thread environment, the following issues require the user's attention:

- There are two types of shared resources: one is the physical I2C bus and the other is hardware components in the AVL DVBSx device. The I2C bus can be shared by different devices. If this is the case, the user should add additional semaphores to the IBSP.c file. The second shared resource is hardware components in the AVL DVBSx device. There are three components inside the device: demodulator, tuner control and Diseqc. The SDK automatically manages semaphores to protect these resources. The IBSP interface will operate correctly provided the user implements the three BSP semaphore functions: InitSemaphore, WaitSemaphore and ReleaseSemaphore.
- In most cases, the token is a semaphore object. In bspdatadef.h, there are two data type prototypes: AVL_semaphore and AVL_psemaphore. The SDK uses both semaphore types. The user needs to redefine these two data types according to their system environment. The SDK defines token objects as AVL_semaphore type and passes the references (address pointer in most systems) to the IBSP

semaphore functions. Example BSP implementations are provided in the SDK.

- The SDK has limitations when using multiple threads:
  1. It is recommended that the user place all Diseqc operations in a single thread. Otherwise, the user will have to verify (protect) that the acknowledged data (data read back) corresponds to the requested data (send data command). In other words, the SDK does not provide protection between the request and response data. For example, if one thread issues a transmit request and it is suspended by a second thread before it reads the Diseqc response back, then the second thread will erroneously read back the response data of the first thread. Users can avoid this situation by using a semaphore to protect the entire transmit/response sequence.

  2. Although not a requirement, when programming the demodulator, Availink recommends putting all control code in one thread. If users want, they can put status query code in another thread. A problem can arise when a programmer uses two threads to control the demodulator. For example one thread can request the demodulator to lock a channel at a symbol rate of 20 Msps. The second thread can request the demodulator to lock a channel at a symbol rate of 30 Msps. Both threads complete, however the programmer is uncertain as to the locked channel symbol rate. The same recommendation applies when programming the tuner control component and Diseqc component. In summary the SDK allows the user one or more threads to target each

component. The programmer should be aware that using more than one thread on each component can produce uncertain results.

5 ADD-ON LAYER

The Add-on layer contains the software components that provide capabilities for tuner control and debugging . The modular components included are listed below:

1. Sharp Tuner: This component implements the tuner driver for the Sharp BS2S7HZ6306 tuner.

2. IAuxiliary interface: This component provides access routines to aid debugging by Availink support staff. It is rarely used by the STB application developers.

5.1 Sharp Tuner

The Sharp Tuner component is a complete implementation of a driver for the BS2S7HZ6306 direct down conversion tuner device from Sharp Microelectronics. This component can also serve as an example for implementing a driver for other tuner components. The examples in the SDK use the Sharp tuner component as the tuner driver.

5.2 IAuxiliary

The IAuxiliary component is not supported in the current version of the SDK.

6 SDK DESIGN PRINCIPLES

This section describes AVL DVBSx SDK design principles that are important for properly understanding and using the API and implementing the BSP.

6.1 NAMING CONVENTIONS

API function names are composed of three parts: ProductID_Interface_Function. For example:

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_LockChannel(
    struct AVL_DVBSx_Channel* psChannel,
    struct AVL_DVBSx_Chip* pAVLChip
);
```

- **AVL_DVBSx** is the Product ID. All the functions in the AVL DVBSx device API use AVL_DVBSx as the Product identifier.
- **IRx** is the Interface. Interface names always begin with an upper case "I". In this example, IRx is the Receiver interface.
- **LockChannel** is the Function name.

Some structure members include a suffix to specify the units. For example, in the following structure declaration:

```
struct AVL_DVBSx_Diseqc_Para(
    AVL_uint16 m_ToneFrequency_kHz,
    …
);
```

m_ToneFrequency_kHz indicates that the member is a 16 bit unsigned integer with an LSB value of kilohertz (kHz).

Some parameters also include a suffix to specify the units with a multiplier. For example, in the following function declaration:

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetRFOffset(
    AVL_pint16 piRFOffset_100kHz,
    const struct AVL_DVBSx_Chip* pAVL_Chip
);
```

piRFOffset_100kHz indicates that the parameter is a pointer to a 16 bit integer with an LSB value of 100 kilohertz (kHz).

6.2 PLATFORM INDEPENDENCE

The SDK implementation achieves platform independence by providing solutions to the following common issues:

- Target Endianness
  The SDK automatically handles systems with varying endianness that is transparent to the user.

- Compiler data type differences
  The API uses data types with the prefix "AVL_" defined in the file bspdatadef.h to avoid inconsistencies among user compilers which may define data types differently (e.g. int, short, etc.). The AVL data types listed in Table 7-1 must be defined by the user according to their compiler.

- Hardware abstraction layer
  The upper API layers rely on a uniform interface to hardware that is independent of the hardware itself. Specifically, this interface consists of I2C read and write functions, software semaphores, and delay capabilities and is provided by the `IBSP` layer. The user is responsible for implementing this BSP abstraction layer.

6.3 BLOCKING AND NONBLOCKING BEHAVIOR

The AVL DVBSx device API contains blocking and non-blocking functions. For the non-blocking functions, the API functions return immediately after sending any/all commands to the AVL DVBSx device. *They do not wait for the commands to be completed.* The host application may call a function later to check the status of the requested operation and is not required to wait in the API for it to finish. Blocking/non-blocking information for each API function can be found in Section 7.3.

6.4 RESOURCE SHARING

A shared resource is protected by a token. Whenever a function needs to access a shared resource, it must obtain the token first. Tokens are maintained by three BSP functions:
`AVL_DVBSx_IBSP_InitSemaphore`,
`AVL_DVBSx_IBSP_WaitSemaphore` and
`AVL_DVBSx_IBSP_ReleaseSemaphore`.

A token should be released after the operation requested by an SDK function call is completed. However, since the AVL DVBSx device

cannot interrupt the API routine, the resource status is updated through polling. The API must poll the AVL DVBSx device through the I2C bus by using functions such as `GetRxOPStatus` in the IRx implementation. Similar functions are used in each component. The status function will check the resource and return EC_OK if the resource is ready for further access. It will return `EC_Running` if the resource is still in use.
The status function is automatically called when necessary by all API functions that must access a shared resource. In this manner, locking and unlocking of shared resources is done without the intervention of the user.

7 DATA STRUCTURES AND FUNCTIONS

In this section we address SDK data structures and functions. Section 7.1 describes primitive data types used in the SDK. Section 7.2 discusses SDK data structures such as enumerations and structures. Section 7.3 is dedicated to the description of the SDK functions.

7.1 DATA TYPES

The SDK data structures and functions use their own set of primitive data types. The default data type definitions are listed in Table 7-1. The user needs to redefine these data types according to their development environment as described in the BSP in Section 4.

Table 7-1: Default SDK Data Type Definitions

SDK Data Type	**Corresponding C Type (Bit Width)**
AVL_char	**signed char** (8 bit)
AVL_uchar	**unsigned char** (8 bit)
AVL_int16	**signed short** (16 bit)
AVL_uint16	**unsigned short** (16 bit)
AVL_int32	**signed integer** (32 bit)
AVL_uint32	**unsigned integer** (32 bit)
AVL_pchar	**signed char** (8 bit) pointer
AVL_puchar	**unsigned char** (8 bit) pointer
AVL_pint16	**signed short** (16 bit) pointer
AVL_puint16	**unsigned short** (16 bit) pointer

SDK Data Type	Corresponding C Type (Bit Width)
AVL_pint32	**signed integer** (32 bit) pointer
AVL_puint32	**unsigned integer** (32 bit) pointer
AVL_semaphore	**platform dependent data structure**
AVL_psemaphore	**AVL_semaphore pointer**

Note: Some platforms provide different data types to protect resources such as a mutex or a semaphore. The AVL_semaphore and AVL_psemaphore must meet the following requirements:

1. There is only one owner at any time.
2. Cannot be reacquired even in the same thread or process.

7.2 DATA STRUCTURES

This section elaborates on the enumerations and structures used in the SDK. An overview will be given here and details of each enumeration and structure will be presented in the remaining sections.

Table 7-2 lists the data structures defined in the SDK.

Table 7-2: SDK Data Structures

Name	Type	Description
AVL_DVBSx_BlindScanPara	Struct	Parameters for a blind scan operation. Defined in IBlindScan.h.
AVL_DVBSx_BlindScanInfo	Struct	Results information from a blind scan operation. Defined in

Name	Type	Description
		IBlindScan.h.
AVL_DVBSx_BlindscanSpectrum	Enum	Blind scan spectrum inversion setting. Defined in avl_dvbsx.h.
AVL_DVBSx_Channel	Struct	Stores the parameters for locking to a channel. Defined in avl_dvbsx.h.
AVL_DVBSx_Chip	Struct	Stores the Availink chip parameters. Defined in avl_dvbsx.h
AVL_DVBSx_Diseqc_Para	Struct	Parameters to configure Diseqc operations. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_RxStatus	Struct	Diseqc Receive status information. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_RxTime	Enum	Time-out period for Diseqc reception. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_TxGap	Enum	Gap time for Diseqc transmission. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_TxMode	Enum	Diseqc transmission mode selection. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_TxStatus	Struct	Diseqc Transmit status information. Defined in IDiseqc.h.
AVL_DVBSx_Diseqc_WaveFormMode	Enum	Used to select normal or envelope DISEQC_OUT output

Name	Type	Description
		mode. Defined in IDiseqc.h.
AVL_DVBSx_ErrorCode	Type	Return value of most API functions to indicate their status. Defined in avl_dvbsx.h.
AVL_DVBSx_FecRate	Enum	FEC code rate. Defined in avl_dvbsx.h.
AVL_DVBSx_FunctionalMode	Enum	Defines the Availink device functional modes. Defined in avl_dvbsx.h.
AVL_DVBSx_LockMode	Enum	Channel lock mode. Defined in avl_dvbsx.h.
AVL_DVBSx_ModulationMode	Enum	Signal modulation mode. Defined in avl_dvbsx.h.
	Enum	MPEG bit order.

Name	Type	Description
7.2.1 AVL_DVBSx_MpegBitOrder This enumeration defines the output bit order of the MPEG data bytes. As shown in the table below, the user can set the bit order to either normal or invert. The meaning differs depending on whether the MPEG interface is configured to output data in serial mode or parallel mode. In serial mode a setting of normal configures the device to output the MSB of each byte first; a setting of invert configures the device to output the LSB of each byte first. In parallel mode a setting of normal configures the device to output the MSB of each byte on pin MPEG_DATA_7 and the	Enum	Rising/Falling edge MPEG clock mode. Defined in avl_dvbsx.h.

Name	Type	Description
LSB of each byte on pin MPEG_DATA_0. A setting of invert configures the device to output the LSB of each byte on pin MPEG_DATA_7 and the MSB of each byte on pin MPEG_DATA_0.		

	MPEG Output Bit
_Norma	Normal output bit o
_Invert	Inverted output bit

AVL_DVBSx_MpegClockPolarity		
AVL_DVBSx_MpegErrorPolarity	Enum	The polarity of the MPEG error signal. Defined in avl_dvbsx.h.

Name	Type	Description

7.2.2 AVL_DVBSx_MpegErrorPolarity

This enumeration defines the polarity of the MPEG error signal.

	MPEG Error Polari
_Norma	The MPEG error sig the payload of a pa uncorrectable error
_Invert	The MPEG error sig payload of a packet uncorrectable error

Name	Type	Description
AVL_DVBSx_MpegFormat	Enum	MPEG output format options. Defined in avl_dvbsx.h.
AVL_DVBSx_MpegInfo	Struct	Structure to hold MPEG output configuration. Defined in avl_dvbsx.h.
AVL_DVBSx_MpegMode	Enum	Serial/Parallel MPEG output mode. Defined in avl_dvbsx.h.
AVL_DVBSx_MpegPulldown	Enum	Enable/disable internal MPEG interface pull-down resistors. Defined in avl_dvbsx.h
AVL_DVBSx_MpegSerialP	Enum	The pin on which to output the

Name	**Type**	**Description**
in		MPEG data in serial mode. Defined in avl_dvbsx.h.
AVL_DVBSx_MpegValidPolarity	Enum	The polarity of the MPEG valid signal. Defined in avl_dvbsx.h.

Name	Type	Description
7.2.3 AVL_DVBSx_ MpegSerialPin This enumeration defines the pin on which the Availink device outputs the MPEG data when the MPEG interface has been configured to operate in serial mode.	Enum	Signal pilot mode. Defined in avl_dvbsx.h

	Output Pin
_DATA0	Serial data is outpu MPEG_DATA_0.
_DATA7	Serial data is outpu MPEG_DATA_7.

7.2.4 AVL_DVBSx_ MpegValidPola rity

This enumeration defines the polarity of the MPEG data valid signal when the MPEG interface is configured to operate in TSP mode.

Name	Type	Description
MPEG Valid Signa		
_Normal The MPEG data va during the payload the parity bytes.		
_Invert The MPEG data va during the payload the parity bytes.		
AVL_DVBSx_Pilot		
AVL_DVBSx_PllConf	Struct	Parameters for configuring the PLL inside the Availink device. Defined in avl_dvbsx.h
AVL_DVBSx_RfagcPola	Enum	Polarity options for AVL_DVBSx_device RFAGC PWM output. Defined in avl_dvbsx.h.
AVL_DVBSx_RollOff	Enum	Signal roll-off factor. Defined in avl_dvbsx.h.
AVL_DVBSx_SignalInfo	Struct	Structure to store the properties which characterize the received signal. Defined in avl_dvbsx.h.
AVL_DVBSx_VerInfo	Struct	Stores the version numbers relevant to the Availink device hardware and software. Defined in avl_dvbsx.h.
AVL_Tuner	Struct	Structure to hold parameters of a tuner and its control functions.

Name	Type	Description
		Defined in ITuner.h.
AVL_VerInfo	Struct	Components of a version number. Defined in avl.h.

7.2.5 AVL_DVBSx_BlindScanPara

This structure defines the blind scan configuration parameters.

Table 7-3: AVL_DVBSx_BlindScanPara Structure Description

Structure Member Variables	Data Type	Description
m_uiStartFreq_100kHz	AVL_uint16	Defines the start frequency of the blind scan operation in units of 100 kHz.
m_uiStopFreq_100kHz	AVL_uint16	Defines the stop frequency of the blind scan operation in units of 100 kHz.
m_uiMinSymRate_kHz	AVL_uint16	Defines the minimum symbol rate to be scanned in units of kHz.
m_uiMaxSymRate_kHz	AVL_uint16	Defines the maximum symbol rate to be scanned in units of kHz.

7.2.6 AVL_DVBSx_BlindScanInfo

This structure stores the blind scan status information.

Table 7-4: AVL_DVBSx_BlindScanInfo Structure Description

Structure Member Variables	Data Type	Description
m_uiProgress	AVL_uint16	The percentage completion for the current blind scan process. A value of 100 indicates that the blind scan process is complete.
m_uiChannelCount	AVL_uint16	The number of channels that have been detected by the blind scan process thus far.
m_uiNextStartFreq_100kHz	AVL_uint16	After each blind scan process, this value stores the start frequency for the next blind scan process in units of 100kHz.
m_uiResultCode	AVL_uint16	The result of the most recently completed blind scan process. 0 – Completed successfully. 1 – Too many channels were found. The Availink device

Structure Member Variables	Data Type	Description
		can store 120 channels per scan.

7.2.7 AVL_DVBSx_BlindscanSpectrum

This enumeration defines the blind scan spectrum inversion setting. Before a blind scan operation is conducted, the user must configure the Availink device to indicate whether the received signal spectrum is inverted. Please see the description of the function IBlindScan_SetSpectrumInversion for more details regarding this enumeration.

Value	Description
AVL_DVBSx_BS_NORMAL	Received signal spectrum is not inverted.
AVL_DVBSx_BS_INVERT	Received signal spectrum is inverted.

7.2.8 AVL_DVBSx_Channel

This structure stores carrier channel information. The m_Flags member of this structure is an unsigned 32-bit integer comprised of bitmapped fields which store channel configuration information. The individual fields of the m_Flags member are described in Table 7-6.

Table 7-5: AVL_DVBSx_Channel Structure Description

Structure Member Variables	**Data Type**	**Description**
m_uiFrequency_kHz	AVL_uint32	The carrier frequency of the channel in units of kHz. This member can be used as the tuner center frequency.
m_uiSymbolRate_Hz	AVL_uint32	The symbol rate of the channel in units of Hz.
m_Flags	AVL_uint32	Contains bitmapped fields which store additional channel configuration information. For more details, see the description below this table as well as the CI_FLAG_XXXX macros defined in avl_dvbsx.h

As mentioned in Table 7-5, the individual bit fields of the m_Flags member of the AVL_DVBSx_Channel structure store additional channel configuration information. These bit fields are used during channel lock operations to configure the Availink demodulator. These bit fields and their meaning are described in Table 7-6. Please note that bit index 0 corresponds to the least significant bit, while bit index 31 corresponds to the most significant bit.

Table 7-6: AVL_DVBSx Channel m_Flags Member Bit Fields

Bit Index	**Field Name**	**Description**

Bit Inde x	Field Name	Description
31-8	Reserved .	Not used.
7	Adaptive Power Save Mode	This field controls whether adaptive power save mode is enabled. If this field is set to 0, adaptive power save mode is disabled. If this field is set to 1, adaptive power save mode is enabled. This bit field of the m_Flags member can be set by calling the SDK function SetAdaptivePowerSaveMode. See the description of that function for more details regarding adaptive power save mode.

6	Channel Lock Mode	**7.2.8.1This field controls the channel lock mode. If this field is set to 0, the channel lock mode is set to fixed. If this field is set to 1, the channel lock mode is set to adaptive. This bit field of the m_Flags member can be set by calling the SDK function SetAdaptivePowerSaveMode**
		
		 `AVL_DVBSx_ErrorCode`
		 `AVL_DVBSx_IRx_SetAdaptivePowerSaveMod`
		 `e (`
		 `struct AVL_DVBSx_Channel *psChannel,`
		 `AVL_uint16 uiEnable`
		 `)`
		
		 This function configures whether the adaptive power save mode is enabled or disabled according

Bit Index	Field Name	Description
		to the input parameter uiEnable. When adaptive power save mode is enabled, the Availink device conserves power by reducing the frequency of its internal clock signals in a manner that is proportionate to the symbol rate of the incoming signal. The adaptive power save mode provides reduced power consumption for symbol rates 15 MHz and lower. For symbol rates greater than 15 MHz, the adaptive power save mode provides no additional reduction in power consumption. The function SetAdaptivePowerSaveMode updates the m_Flags member of the psChannel input parameter to reflect the adaptive power save mode setting. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new adaptive power save mode setting is used. Please note that the new adaptive power save mode only applies to *subsequent* calls to the LockChannel function. If the adaptive power save mode is changed after the channel has been locked, the change has no effect. When using adaptive power save mode or when toggling the adaptive power save mode setting, the user *must* halt the Availink device before calling the LockChannel function. This can be

Bit Index	Field Name	Description
		accomplished by calling the IBase_Halt function. Halting the Availink device ensures that the Availink internal clock signals can be configured correctly. If the Availink device has not been halted and the internal clock signals cannot be configured properly, the LockChannel function returns the AVL_DVBSx_EC_GeneralFail error code. For more details, please see the function descriptions for IBase_Halt and IRx_LockChannel. **Parameters:** psChannel; A pointer to the channel object for which the adaptive power save mode is being configured. uiEnable; Controls whether adaptive power save mode is enabled or disabled (0 – Disable, 1 – Enable). **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the adaptive power save mode is set. **Remarks:** This function has not been implemented yet. However, it will be incorporated in the future release of AVL DVBSx SDK.

Bit Index	**Field Name**	**Description**
		SetChannelLockMode. See the description of that function for more details.
5	Automatic IQ Swap	This field controls whether the automatic IQ swap feature is enabled. If this field is set to 0, the user must manually configure whether the I and Q channels are swapped within the Availink demodulator. If this field is set to 1, the Availink demodulator automatically determines the spectral sense of the received signal and automatically configures whether the I and Q channels are swapped inside the demodulator.
4-2	Signal Standard	This field specifies the signal standard of the received signal. If the user does not know the signal standard, they may specify that the standard is unknown, causing the Availink demodulator to automatically detect the standard. This field may be configured with the following values: 0 – DVB-S 1 – DVB-S2 2 – Reserved. Do not use. 3 – Reserved. Do not use. 4 – Standard is unknown. Configures the Availink demodulator to automatically detect the standard.
1	Reserved	Not used. Set to 0.
0	IQ Swap	This field is only meaningful if automatic IQ swap is disabled (i.e. if bit 5 of m_Flags is set to 0). If automatic IQ swap is disabled and this field is set

Bit Index	Field Name	Description
		to 0, then the Availink device does not swap the I and Q signals. If automatic IQ swap is disabled and this field is set to 1, then the Availink device swaps the I and Q signals.

7.2.9 AVL_DVBSx_Chip

This structure stores data and flags related to the Availink device. The SDK supports up to two Availink devices, each of which has its own private data and global flags. There is no need for the user to access this structure directly, and the user is advised to refrain from doing so.

Table 7-7: AVL_DVBSx_Chip Structure Description

Structure Member Variables	Data Type	Description
m_SlaveAddr	AVL_uint16	Seven bit I2C slave address of the Availink device. Defined in avl_dvbsx.h.
m_uiBusId	AVL_uint16	I2C bus identifier.
m_StdBuffIndex	AVL_uint16	A device level global variable which stores the read position of the standard output.

Structure Member Variables	**Data Type**	**Description**
m_DemodFrequency_10kHz	AVL_uint16	The demodulator internal clock frequency in units of 10 kHz. Set during PLL setup in IBase.c and should not be changed.
m_FecFrequency_10kHz	AVL_uint16	The FEC internal clock frequency in units of 10 kHz. Set during PLL setup in IBase.c and should not be changed.
m_MpegFrequency_10kHz	AVL_uint16	The MPEG reference clock frequency in units of 10 kHz. Set during PLL setup in IBase.c and should not be changed.
m_semRx	AVL_semaphore	A semaphore used to protect the receiver command channel.
m_semI2CRepeater	AVL_semaphore	A semaphore used to protect the I2C repeater channel.
m_semI2CRepeater_r	AVL_semaphore	A semaphore used to protect the I2C repeater read channel.

Structure Member Variables	Data Type	Description
m_semDiseqc	AVL_semaphore	A semaphore used to protect the Diseqc command channel.

7.2.10 AVL_DVBSx_Diseqc_Para

This structure defines the Diseqc configuration.

Table 7-8: AVL_DVBSx_Diseqc_Para Structure Description

Structure Member Variables	Data Type	Description
m_ToneFrequency_kHz	AVL_uint16	The Diseqc clock frequency in units of kHz. A value of 22 kHz is typical.
m_TXGap	enum AVL_DVBSx_ Diseqc_TxGap	The gap between two output bytes.
m_TxWaveForm	enum AVL_DVBSx_ Diseqc_WaveF	The output waveform format, either normal or envelope.

Structure Member Variables	Data Type	Description
	ormMode	
m_RxTimeout	enum AVL_DVBSx_Diseqc_RxTime	The receive time window. At the end of each Diseqc transmission, the Availink device (Diseqc master) opens a receive window. If the Diseqc slave device exceeds the RxTimeout value, the received data is discarded.
m_RxWaveForm	enum AVL_DVBSx_Diseqc_WaveFormMode	The input data waveform format, either normal or envelope.

7.2.11 AVL_DVBSx_Diseqc_RxStatus

This structure stores the Diseqc receive status information.

Table 7-9: AVL_DVBSx_Diseqc_RxStatus Structure Description

Structure Member Variables	Data Type	Description
m_RxFifoCount	AVL_uchar	The number of bytes available in the receive FIFO.

Structure Member Variables	**Data Type**	**Description**
m_RxFifoParChk	AVL_uchar	The parity check result byte of the received data. This is a bit-mapped field in which each bit represents the parity check result for each byte in the receive FIFO. The upper bits without corresponding data are undefined. If a bit is 1, the corresponding byte in the FIFO has good parity. For example, if three bytes are in the FIFO, and the parity check value is 0x03 (value of bit 2 is zero), then the first and the second bytes in the receive FIFO are good. The third byte is in error.
m_RxDone	AVL_uchar	Indicates whether the receive window has closed. 0 – The receive window is still open. 1 – The receive window has closed.

7.2.12 AVL_DVBSx_Diseqc_RxTime

This enumeration defines the length of the time-out period for Diseqc reception after transmission ends. Data that is received outside of this time-out period is abandoned.

Value	**Diseqc Receiving Interval**
AVL_DVBSx_DRT_150ms	150 ms
AVL_DVBSx_DRT_170ms	170 ms
AVL_DVBSx_DRT_190ms	190 ms
AVL_DVBSx_DRT_210ms	210 ms

7.2.13 AVL_DVBSx_Diseqc_TxGap

This enumeration defines the interval of time between sequential Diseqc transmissions when the Diseqc interface is in tone burst mode (transmits tone0 or tone1). The gap specified by this enumeration is inserted between each tone.

Value	**Diseqc Transmit Gap**
AVL_DVBSx_DTXG_15ms	15 ms
AVL_DVBSx _DTXG_20ms	20 ms
AVL_DVBSx _DTXG_25ms	25 ms
AVL_DVBSx _DTXG_30ms	30 ms

7.2.14 AVL_DVBSx_Diseqc_TxMode

This enumeration defines the Diseqc transmitter output modes. The Diseqc interface supports several different transmission modes.

Value	**Transmit Modulation Mode**
AVL_DVBSx_DTM_Modulation	Modulates the output data according to supplied data.
AVL_DVBSx _DTM_Tone0	Defines a 0 tone signal.
AVL_DVBSx _DTM_Tone1	Defines a 1 tone signal.
AVL_DVBSx _DTM_Continuous	Transmits a continuous tone.

7.2.15 AVL_DVBSx_Diseqc_TxStatus

This structure holds the transmission status information.

Table 7-10: AVL_DVBSx_Diseqc_TxStatus Structure Description

Structure Member Variables	**Data Type**	**Description**
m_TxDone	AVL_uchar	Indicates if the transmission is complete (0 – Not complete, 1 – Complete).
m_TxFifoCount	AVL_uchar	The number of bytes remaining in the transmit FIFO.

7.2.16 AVL_DVBSx_Diseqc_WaveFormMode

This enumeration defines the Diseqc output signal waveform present on the DISEQC_OUT pin of the Availink device.

Value	Diseqc Waveform Mode
AVL_DVBSx _DWM_Modulation	Normal modulation mode.
AVL_DVBSx _DWM_Envelope	Envelope modulation mode.

7.2.17 AVL_DVBSx_ErrorCode

These definitions are used to report the result of Availink device operations. When contacting Availink technical support, it is helpful to report any error codes that have been observed.

Value	Description
AVL_DVBSx_EC_OK	The respective SDK function has returned with no errors.
AVL_DVBSx _EC_GeneralFail	A general failure has occurred. Example conditions under which this error is reported include device boot failure and BSP initialization failure. This error code may be used by the tuner driver developer to report conditions such as invalid tuner

	parameters.
AVL_DVBSx _EC_I2CFail	An I2C operation failed during communication with the Availink device through the BSP.
AVL_DVBSx _EC_Running	The Availink device is busy processing a previous command.
AVL_DVBSx _EC_MemoryRunout	There is insufficient memory to complete the requested operation. This type of error may occur during a blind scan process.
AVL_DVBSx _EC_BSP_ERROR1	A user defined error code used to report BSP errors.
AVL_DVBSx _EC_BSP_ERROR2	A user defined error code used to report BSP errors.

7.2.18 AVL_DVBSx_FecRate

This enumeration defines the signal standard and the FEC code rate for the received signal.

Value	**FEC Code Rate**
RX_DVBS_1_2	DVB-S, Rate 1/2
RX_DVBS_2_3	DVB-S, Rate 2/3
RX_DVBS_3_4	DVB-S, Rate 3/4
RX_DVBS_5_6	DVB-S, Rate 5/6
RX_DVBS_6_7	DVB-S, Rate 6/7
RX_DVBS_7_8	DVB-S, Rate 7/8
RX_DVBS2_1_4	DVB-S2, Rate 1/4
RX_DVBS2_1_3	DVB-S2, Rate 1/3

RX_DVBS2_2_5	DVB-S2, Rate 2/5
RX_DVBS2_1_2	DVB-S2, Rate 1/2
RX_DVBS2_3_5	DVB-S2, Rate 3/5
RX_DVBS2_2_3	DVB-S2, Rate 2/3
RX_DVBS2_3_4	DVB-S2, Rate 3/4
RX_DVBS2_4_5	DVB-S2, Rate 4/5
RX_DVBS2_5_6	DVB-S2, Rate 5/6
RX_DVBS2_8_9	DVB-S2, Rate 8/9
RX_DVBS2_9_10	DVB-S2, Rate 9/10

7.2.19 AVL_DVBSx_FunctionalMode

This enumeration defines the functional modes of the Availink device. The Availink device can operate in either the demodulator functional mode or the blind scan functional mode. In demodulator mode, the device is capable of performing the demodulator signal processing needed to lock to a channel. In blind scan mode, the device is capable of performing blind scan operations. The user can change functional modes by calling the SDK function IBase_SetFunctionalMode. Please see the description of the SetFunctionalMode function for more details.

Value	**AVL_DVBSx Device Functional Mode**
AVL_DVBSx_FunctMode_Demod	Demodulator functional mode. The Availink device performs demodulator signal processing and is capable of locking to a channel.
AVL_DVBSx _FunctMode_BlindScan	Blind scan functional mode. The Availink device performs blind

	scan signal processing and is capable of performing blind scan operations.

7.2.20 AVL_DVBSx_LockMode

This enumeration defines the channel lock mode.

Value	Lock Mode
AVL_DVBSx_LOCK_MODE_FIXED	Fixed lock mode
AVL_DVBSx_LOCK_MODE_ADAPTIVE	Adaptive lock mode

7.2.21 AVL_DVBSx_ModulationMode

This enumeration defines the modulation type of the received signal.

Value	Modulation Type
AVL_DVBSx_MM_QPSK	QPSK
AVL_DVBSx _MM_8PSK	8PSK
AVL_DVBSx _MM_16APSK	16APSK
AVL_DVBSx _MM_32APSK	32APSK

7.2.22 AVL_DVBSx_MpegBitOrder

This enumeration defines the output bit order of the MPEG data bytes. As shown in the table below, the user can set the bit order to either normal or invert. The meaning differs depending on whether the MPEG interface is configured to output data in serial mode or parallel mode.

In serial mode a setting of normal configures the device to output the MSB of each byte first; a setting of invert configures the device to output the LSB of each byte first. In parallel mode a setting of normal configures the device to output the MSB of each byte on pin MPEG_DATA_7 and the LSB of each byte on pin MPEG_DATA_0. A setting of invert configures the device to output the LSB of each byte on pin MPEG_DATA_7 and the MSB of each byte on pin MPEG_DATA_0.

Value	**MPEG Output Bit Order**
AVL_DVBSx_MPBO_Normal	Normal output bit order.
AVL_DVBSx_MPBO_Invert	Inverted output bit order.

7.2.23 AVL_DVBSx_MpegClockPolarity

This enumeration defines the polarity of the MPEG clock signal. The MPEG clock polarity must be configured to match the desired capture edge for the MPEG decoder device receiving the Availink device MPEG output signals.

Value	**MPEG Clock Polarity**
AVL_DVBSx_MPCP_Falling	The MPEG data is valid on the falling edge of the MPEG_CLK.
AVL_DVBSx_MPCP_Rising	The MPEG data is valid on the rising edge of the MPEG_CLK.

7.2.24 AVL_DVBSx_MpegErrorPolarity

This enumeration defines the polarity of the MPEG error signal.

Value	MPEG Error Polarity
AVL_DVBSx_MPEP_Normal	The MPEG error signal is high during the payload of a packet which contains uncorrectable error(s).
AVL_DVBSx_MPEP_Invert	The MPEG error signal is low during the payload of a packet which contains uncorrectable error(s).

7.2.25 AVL_DVBSx_MpegFormat

This enumeration defines the MPEG output format.

Value	MPEG Output Format
AVL_DVBSx_MPF_TS	Transport Stream (TS) Mode. MPEG output data bytes are a normal sequence of 188 byte packets (sync byte plus 187 data bytes).
AVL_DVBSx_MPF_TSP	Transport Stream plus Parity (TSP) Mode. MPEG output data bytes are a sequence of 204 byte packets (sync byte + 187 data bytes + 16 parity bytes). MPEG_VALID is not asserted during the parity byte output.

7.2.26 AVL_DVBSx_MpegInfo

This structure contains the MPEG interface configuration parameters that typically need to be configured for the user's application.

Structure Member Variables	Data Type	Description
m_MpegFormat	enum AVL_DVBSx_MpegFormat	Refer to AVL_DVBSx_MpegFormat
m_MpegMode	enum AVL_DVBSx_MpegMode	Refer to AVL_DVBSx_MpegMode
m_MpegClockPolarity	enum AVL_DVBSx_MpegClockPolarity	Refer to AVL_DVBSx_MpegClockPolarity

7.2.27 AVL_DVBSx_MpegMode

This enumeration defines the MPEG output mode.

Value	MPEG Output Mode
AVL_DVBSx_MPM_Parallel	Output parallel data.
AVL_DVBSx_MPM_Serial	Output serial data.

7.2.28 AVL_DVBSx_MpegPulldown

This enumeration defines the state of the internal MPEG interface pull-down resistors.

Value	MPEG Output Mode
AVL_DVBSx_MPPD_Disable	Disable internal pull-down resistors.
AVL_DVBSx_MPPD_Enable	Enable internal pull-down resistors.

7.2.29 AVL_DVBSx_MpegSerialPin

This enumeration defines the pin on which the Availink device outputs the MPEG data when the MPEG interface has been configured to operate in serial mode.

Value	Output Pin
AVL_DVBSx_MPSP_DATA0	Serial data is output on pin MPEG_DATA_0.
AVL_DVBSx_MPSP_DATA7	Serial data is output on pin MPEG_DATA_7.

7.2.30 AVL_DVBSx_MpegValidPolarity

This enumeration defines the polarity of the MPEG data valid signal when the MPEG interface is configured to operate in TSP mode.

Value	MPEG Valid Signal Polarity
AVL_DVBSx_MPVP_Normal	The MPEG data valid signal is high during the payload data and low during the parity bytes.
AVL_DVBSx_MPVP_Invert	The MPEG data valid signal is low during the payload data and high during the parity bytes.

7.2.31 AVL_DVBSx_Pilot

This enumeration defines the signal pilot mode.

Value	MPEG Output Mode
RX_Pilot_off	Pilot off
RX_Pilot_on	Pilot on

7.2.32 AVL_DVBSx_PllConf

This structure contains the PLL configuration parameters. The SDK defines an internal const object array for this data structure in the file IBSP.c. The user may comment out unused elements from that array to reduce the SDK footprint size.

Structure Member Variables	Data Type	Description
m_uiClkf, m_uiClkr, m_uiPllod, m_uiPllod2, m_uiPllod3	AVL_uint16	Members which store the PLL clock divider values.
m_RefFrequency_kHz	AVL_uint16	Reference clock frequency in units of kHz.
m_DmodFrequency_10kHz	AVL_uint16	Internal demodulator clock frequency in units of 10kHz.
m_FecFrequency_1	AVL_uint16	Internal FEC clock frequency

0kHz		in units of 10 kHz.
m_MpegFrequency_10kHz	AVL_uint16	Internal MPEG clock frequency in units of 10 kHz.

7.2.33 AVL_DVBSx_RfagcPola

This enumeration defines the polarity of the RF AGC control signal. The polarity of the RF AGC control signal must be configured to match that required by the tuner.

Value	**RFAGC polarization Mode**
AVL_DVBSx_RA_Normal	Normal polarization. This setting is used for a tuner whose gain increases with increased AGC voltage.
AVL_DVBSx_RA_Invert	Inverted polarization. The default value. Most tuners fall into this category. This setting is used for a tuner whose gain decreases with increased AGC voltage.

7.2.34 AVL_DVBSx_RollOff

This enumeration defines the roll-off factor, or excess bandwidth, of the received signal.

Value	**Roll-Off Factor**
AVL_DVBSx_RO_20	0.20
AVL_DVBSx_RO_25	0.25
AVL_DVBSx_RO_35	0.35

7.2.35 AVL_DVBSx_SignalInfo

This structure stores the properties which characterize the received signal. After the Availink device locks to a received signal, it decodes the signal's properties and reports these values to the host processor.

Structure Member Variables	**Data Type**	**Description**
m_pilot	enum AVL_DVBSx_Pilot	Refer to enum AVL_DVBSx_Pilot
m_coderate	enum AVL_DVBSx_FecRate	Refer to enum AVL_DVBSx _FecRate
m_modulation	enum AVL_DVBSx_ModulationMode	Refer to enum AVL_DVBSx _ModulationMode
m_rolloff	enum AVL_DVBSx_RollOff	Refer to enum AVL_DVBSx_RollOff

7.2.36 AVL_DVBSx_VerInfo

This structure defines the version number for the Availink device, the API and the patch. In any system involving an Availink DVB-S/DVB-S2 demodulator, there are three versioned components: the Availink device, the API, and the patch content. For each component, there are three parts of version information: Major, Minor and build, as defined in the data structure AVL_VerInfo.

Structure Member Variables	Data Type	Description
m_Chip	AVL_VerInfo	Chip version
m_API	AVL_VerInfo	API version
m_Patch	AVL_VerInfo	Patch version

7.2.37 AVL_Tuner

This structure defines a tuner control interface. The STB application can use this interface to control the tuner. The tuner driver developer should implement this interface for the specific tuner being used.

Structure Member Variables	**Data Type**	**Description**
m_uiSlaveAddress	AVL_uint16	The 8 bit I2C write address of the tuner.
m_uiI2CBusClock_kHz	AVL_uint16	The I2C bus clock speed in units of kHz.
m_uiFrequency_100kHz	AVL_uint16	The carrier frequency of the channel in units of 100 kHz. The tuner locks to this frequency.
m_uiLPF_100kHz	AVL_uint16	The lowpass filter bandwidth setting of the tuner in units of 100 kHz.
m_pParameters	void *	User defined custom tuner parameters such as baseband gain, etc.
m_pAVLChip	struct AVL_DVBSx_Chip *	A pointer to the chip object to which the tuner is connected.
m_pInitializeFunc	AVL_DVBSx_ErrorCode (struct AVL_Tuner *)	A pointer to a function which initializes the tuner.
m_pGetLockStatusF	AVL_DVBSx_Err	A pointer to a function which

unc	orCode (struct AVL_Tuner *)	queries the lock status of the tuner.
m_pDumpDataFunc	AVL_DVBSx_ErrorCode (AVL_puchar ucpData, AVL_puchar ucpSize, struct AVL_Tuner *)	A pointer to a function which dumps data for debugging purposes. Use of this pointer is optional. The customer may assign a NULL value to this pointer.
m_pLockFunc	AVL_DVBSx_ErrorCode (struct AVL_Tuner *)	A pointer to a function which locks the tuner to a particular frequency.

7.2.38 AVL_VerInfo

The struct AVL_VerInfo defines three components of a version number: major, minor, and build.

Structure Member Variables	**Data Type**	**Description**
m_Major	AVL_uchar	The major version number.
m_Minor	AVL_uchar	The minor version number.
m_Build	AVL_uint16	The build version number.

7.3 FUNCTIONS

The functions are separated into different categories (interfaces). The second part of a function name indicates the interface it belongs to. Functions in the same interface control the same component of the chip or do similar jobs.

All the functions in one category are listed in the tables at the beginning of each subsection. The last column in the function tables shows whether the function is blocking (B) or non-blocking (NB).

7.3.1 IBase

This interface provides the initialization, control, and information access capabilities for the Availink device from a host processor.

Table 7-11: IBase Interface Functions

Name	**Summary**	**Behavior**
DownloadFirmware	Downloads a firmware patch to the Availink device.	B

7.3.1.1DownloadFirmware `AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_DownloadFirmware ( AVL_puchar pFirmwareData, const struct AVL_DVBSx_Chip * pAVLChip )` This function downloads firmware patch data to the Availink device. This function is an internal SDK function. It is recommended that the user refrain from directly calling this function. This function is inherently called by the SDK function IBase_Initialize. The user should call that function instead of	Reads the voltage level of the GPIO pins.	B

calling the IBase_DownloadFirmware function. **Parameters:** pFirmwareData; Pointer to the firmware patch to be downloaded to the Availink device. pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the firmware is being downloaded. **Returns:** AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the firmware has been downloaded to the Availink device; Return AVL_DVBSx_EC_I2C Fail if there is an I2C communication problem. GetGPIOVal		

GetFunctionalMode	Gets the current device functional mode.	B
GetRxOPStatus	Checks if the device has finished processing the last operational command sent to it.	B
GetStatus	Checks whether device initialization is complete.	B
GetVersion	Retrieves the device version information.	B
Halt	Halts the Availink device.	NB

7.3.1.2Halt `AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_Halt ( struct AVL_DVBSx_Chip * pAVLChip )` This function commands the Availink device to halt normal channel processing. When the Availink device is halted, all internal hardware blocks are powered on and the device is ready to initiate channel processing upon command. On boot-up, the Availink device behaves as if it has been sent the halt command. **Parameters:**	Initializes the Availink device.	NB

pAVLChip; Pointer to the AVL_DVBSx_Chip object being halted. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the halt command has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_Running if the halt command could not be sent because the device is busy processing a previous command; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. Initialize		
SendRxOP	Sends an operational command to the Availink device.	NB
SetFunctionalMode	Sets the device functional mode.	B
SetGPIODir	Sets the direction of the GPIO pins.	B

SetGPIOVal	Sets the voltage level on those GPIO pins which have been configured as outputs.	B
Sleep	Puts the Availink device into low power sleep mode.	NB
Wake	Wakes the Availink device from sleep.	NB

7.3.1.3DownloadFirmware

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_DownloadFirmware
(
 AVL_puchar  pFirmwareData,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function downloads firmware patch data to the Availink device. This function is an internal SDK function. It is recommended that the user refrain from directly calling this function. This function is inherently called by the SDK function IBase_Initialize. The user should call that function instead of calling the IBase_DownloadFirmware function.

Parameters:

pFirmwareData; Pointer to the firmware patch to be downloaded to the Availink device.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the firmware is being downloaded.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the firmware has been downloaded to the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.4GetGPIOVal

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_GetGPIOVal  (
AVL_puchar  pucVal,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function reads the voltage level of the GPIO pins. The Availink device provides three GPIO pins: CS_0, LNB_CNTRL_1, and GPIO_CLK.

Parameters:

pucVal; Pointer to a variable in which to store the GPIO pin voltage levels. This is a bitmapped field in which each of the three LSBs will contain the voltage level of a particular GPIO pin. A value bit of zero means that the corresponding pin is low, while a value bit of one, means that the corresponding pin is high. Bit 0 (the LSB) corresponds to pin CS_0, bit 1 corresponds to pin LNB_CNTRL_1, and bit 2 corresponds to pin GPIO_CLK. The GPIO pin voltage level can be read regardless of whether the pin has been configured as an input or output.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the GPIO pin voltage levels are being read.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the GPIO pin voltages have been read; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.5GetFunctionalMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IBase_GetFunctionalMode  (
AVL_DVBSx_FunctionalMode *pFunctionalMode
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function determines the present functional mode of the Availink device. The device can be in one of two functional modes, demodulator mode or blind scan mode. In demodulator mode the Availink device performs demodulator signal processing and is capable of locking to a channel. In blind scan mode, the device is capable of performing blind scan operations.

Parameters:

pFunctionalMode; A pointer to an object to which the function writes the device functional mode.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the functional mode is being retrieved.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the functional mode has been retrieved; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.6GetRxOPStatus

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_GetRxOPStatus  (
const struct AVL_DVBSx_Chip *   pAVLChip
)
```

This function checks whether the Availink device has finished processing the last operational command that was sent to it.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip for which the operational command status is being checked.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the device has finished processing the last command; Return AVL_DVBSx_EC_Running if the device is still processing a previous command; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:
This function is an 'internal' SDK function. Availink recommends that the user refrain from directly calling this function.

7.3.1.7GetStatus

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_GetStatus  (
struct AVL_DVBSx_Chip *   pAVLChip
)
```

This function checks whether the Availink device has completed its initialization procedure. The Availink device implements an initialization procedure when the function AVL_DVBSx_IBase_Initialize is called. The function AVL_DVBSx_IBase_GetStatus should be called following any calls to AVL_DVBSx_IBase_Initialize to ensure that device initialization is complete.

Parameters:
 pAVLChip; A pointer to the AVL_DVBSx_Chip for which the initialization status is being checked.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if Availink device initialization is complete; Returns AVL_DVBSx_EC_GeneralFail if the Availink device has not been successfully initialized; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.8GetVersion

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_GetVersion  (
struct AVL_DVBSx_VerInfo *  pVerInfo,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function retrieves the Availink device version information.

Parameters:
 pVerInfo; Pointer to an object in which to store the retrieved version information. Refer to AVL_DVBSx_Ver_Info for details.
 pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the version information is being retrieved.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the version information has been retrived; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.9Halt

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_Halt  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function commands the Availink device to halt normal channel processing. When the Availink device is halted, all internal hardware blocks are powered on and the device is ready to initiate channel processing upon command. On boot-up, the Availink device behaves as if it has been sent the halt command.

Parameters:
pAVLChip; Pointer to the AVL_DVBSx_Chip object being halted.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the halt command has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_Running if the halt command could not be sent because the device is busy processing a previous command; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.10 Initialize

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_Initialize  (
const struct AVL_DVBSx_PllConf *  pPLLConf,
AVL_puchar  pInitialData,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function initializes the Availink device. The function downloads the firmware patch file to the device and boots the device. It is recommended that the user call the function AVL_DVBSx_IBase_GetStatus after calling AVL_DVBSx_IBase_Initialize to determine whether the initialization process is complete.

Parameters:
pPLLConf; Pointer to the AVL_DVBSx_PLLConf object which contains the PLL configuration to be used to initialize the PLL.
pInitialData; Pointer to the buffer which holds the firmware patch data.
pAVLChip; Pointer to the AVL_DVBSx_Chip object being initialized.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the initialization data has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.11 SendRxOP

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_SendRxOP  (
AVL_uchar  ucOpCmd,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function sends an operational command to the Availink device. The available operational commands are defined in IBase.h and begin with “OP_RX_”.

Parameters:
ucOpCmd; The operational command code as defined in IBase.h.
pAVLChip; A pointer to the AVL_DVBSx_Chip object to which an operational commmand is being sent.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the operational command is sent to the device; Return AVL_DVBSx_EC_Running if the command could not be send because the Availink device is still processing a previous command; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:
This function is an 'internal' SDK function. Availink recommends that the user refrain from directly calling this function.

7.3.1.12 SetFunctionalMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IBase_SetFunctionalMode  (
const struct AVL_DVBSx_Chip * pAVLChip,
enum AVL_DVBSx_FunctionalMode enumFunctionalMode
)
```

This function sets the Availink device functional mode. The Availink device can be placed into one of two functional modes, demodulator mode or blind scan mode.

In demodulator mode, the device performs demodulator signal processing. The demodulator mode must be used for channel lock operations. In the blind scan mode, the device performs blind scan signal processing. The blind scan mode must be used for all blind scan operations.

For normal channel lock operations, the device must be placed into the demodulator functional mode prior to calling the function AVL_DVBSx_IRx_LockChannel. For blind scan operations, the device must be placed into the blind scan functional mode prior to calling the function AVL_DVBSx_IBlindScan_Scan.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the functional mode is being set.
enumFunctionalMode; The functional mode into which the device is being placed.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the functional mode has been set; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.13 SetGPIODir

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_SetGPIODir  (
AVL_uchar  ucDir,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the direction of the GPIO pins. The Availink device provides three GPIO pins: CS_0, LNB_CNTRL_1, and GPIO_CLK.

Parameters:

ucDir; This is a bitmapped field in which each of the three LSBs controls the direction of a particular GPIO pin. Setting a direction bit to a one configures the respective pin as an input, and setting the bit to a zero configures the pin as an output. Bit 0 (the LSB) corresponds to pin CS_0, bit 1 corresponds to pin LNB_CNTRL_1, and bit 2 corresponds to pin GPIO_CLK..

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the GPIO pin directions are being configured.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if GPIO pin directions have been configured; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:

GPIO pins which are not used should not be left floating and should be pulled up or down through a resistor. If an external device drives the GPIO pin, be sure to configure the pin as an input so that a conflict does not occur.

7.3.1.14 SetGPIOVal

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_SetGPIOVal  (
AVL_uchar  ucVal,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function sets the voltage level of any GPIO pins that have been configured as outputs. The Availink device provides three GPIO pins: CS_0, LNB_CNTRL_1, and GPIO_CLK.

Parameters:

ucVal This is a bitmapped field in which each of the three LSBs controls the voltage level of a particular GPIO pin that has been configured as an output. Setting a value bit to a zero drives the corresponding pin low, while setting a value bit to a one, drives the corresponding pin high. Bit 0 (the LSB) corresponds to pin CS_0, bit 1 corresponds to pin LNB_CNTRL_1, and bit 2 corresponds to pin GPIO_CLK.

pAVLChip A pointer to the AVL_DVBSx_Chip object for which the GPIO pin output voltages are being configured.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the GPIO pin output voltages have been configured; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:

This function only controls the voltage of those GPIO pins that have been configured as outputs. If a GPIO pin has been configured as an input, the function has no effect for that pin.

7.3.1.15 Sleep

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_Sleep  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function commands the Availink device to enter low power sleep mode. When the Availink device is in sleep mode, some internal hardware blocks are held in an idle state to reduce power consumption. To bring the device out of sleep mode, the user must call the function IBase_Wake.

Parameters:
pAVLChip; Pointer to the AVL_DVBSx_Chip object being commanded to enter low power sleep mode.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the sleep command has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_Running if the sleep command could not be sent because the device is busy processing a previous command; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.1.16 Wake

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBase_Wake  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function commands the Availink device to wake from low power sleep mode. When the Availink device wakes up, all internal hardware blocks are powered on and the device is ready to initiate channel processing upon command.

Parameters:
pAVLChip; Pointer to the AVL_DVBSx_Chip object being commanded to wake up from low power sleep mode.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the wake command has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_Running if the wake command could not be sent because the device is busy processing a previous command; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.2 IBlindScan

The IBlindScan interface provides a simple approach for performing a blind scan operation. Before calling the functions in this interface, the user must place the Availink device into the blind scan functional mode by calling the function AVL_DVBSx_IBase_SetFunctionalMode. This function places the Availink device into blind scan mode.

Once the user has placed the device into the blind scan functional mode, the user must execute the blind scan procedure which is depicted in Figure 7-1. In particular, the user must first call the function IBlindScan_SetSpectrumInversion to configure the device to indicate whether the received signal spectrum is inverted. Please see the function description IBlindScan_SetSpectrumInversion for details regarding how to use this function properly.

Once the blind scan spectrum inversion setting has been configured, the user must call the function IBlindScan_Reset to ensure that the device resets its internal blind scan statistics. Following this, the user may begin to perform the blind scan procedure.

As shown in Figure 7-1, the blind scan procedure is comprised of a loop in which the user configures the blind scan parameters, locks the tuner, initiates a scan operation, and monitors the scan progress. The user is advised to configure the tuner lowpass filter bandwidth to the maximum bandwidth over which the tuner frequency response is flat.

Once the blind scan operation is complete, the list of detected channels are returned to the user in an array. The Availink device can hold a maximum of 120 carrier channels within its internal memory. If the Availink device detects that there are more than 120 channels in the scan range, a special code is returned to the user. Please see Section 7.3.2.2. GetScanStatus and Section 7.2.6 AVL_DVBSx_BlindScanInfo for more details regarding this.

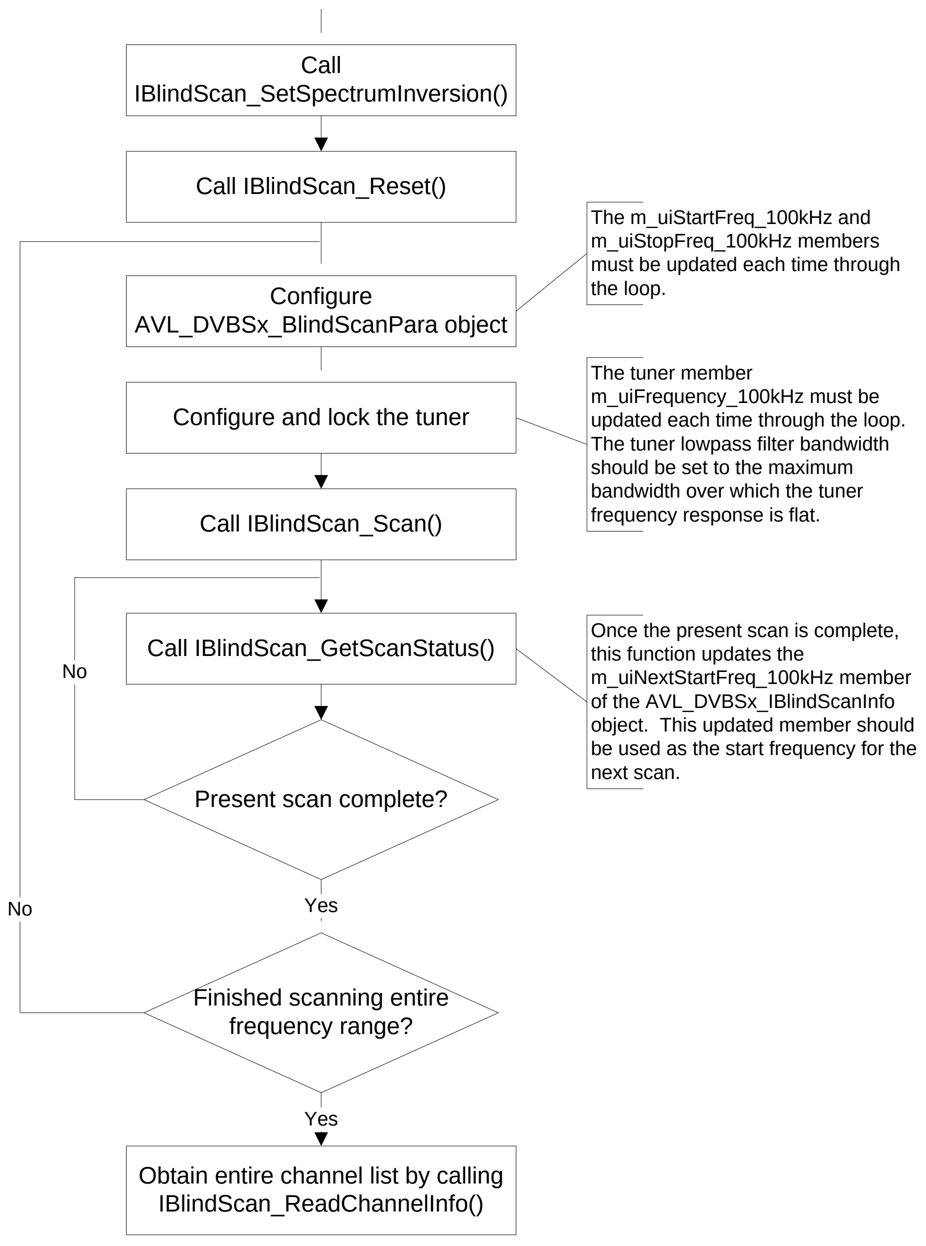
Call IBlindScan_SetSpectrumInversion()
Call IBlindScan_Reset()
Configure AVL_DVBSx_BlindScanPara object
The m_uiStartFreq_100kHz and m_uiStopFreq_100kHz members must be updated each time through the loop.
Configure and lock the tuner
The tuner member m_uiFrequency_100kHz must be updated each time through the loop. The tuner lowpass filter bandwidth should be set to the maximum bandwidth over which the tuner frequency response is flat.
Call IBlindScan_Scan()
Call IBlindScan_GetScanStatus()
Once the present scan is complete, this function updates the m_uiNextStartFreq_100kHz member of the AVL_DVBSx_IBlindScanInfo object. This updated member should be used as the start frequency for the next scan.
No
Present scan complete?
Yes
No
Finished scanning entire frequency range?
Yes
Obtain entire channel list by calling IBlindScan_ReadChannelInfo()

Figure 7-1: Overview of Blind Scan Process

While Figure 7-1 provides a high level overview of the procedure for performing a blind scan operation, the user is advised to refer to the blind scan example provided with the SDK for more details.

The IBlindScan interface provides the following functions:

Table 7-12: IBlindScan Interface Functions

Name	Summary	Behavior
Cancel	Cancels the current scan procedure	NB
GetScanStatus	Checks the internal status of the blind scan procedure	B
ReadChannelInfo	Retrieves the blind scan results	B
Reset	Resets the device's internal blind scan state variables	B
Scan	Performs the blind scan	B
SetSpectrumInversion	Configures the blind scan spectrum inversion setting	B

7.3.2.1Cancel

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBlindScan_Cancel  (
struct AVL_DVBSx_Chip * pAVLChip
)
```

This function cancels a previously requested blind scan process. Prior to calling this function the user must place the Availink device into the blind scan functional mode by calling the function AVL_DVBSx_IBase_SetFunctionalMode. If the Availink device has not been placed in the blind scan functional mode, then the AVL_DVBSx_IBlindScan_Cancel function returns the AVL_DVBSx_EC_GeneralFail error code.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which a blind scan operation is being canceled.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the command to cancel the blind scan operation has been successfully sent to the Availink device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if the cancel command could not be sent to the Availink device because the device is still processing a previous command; Returns AVL_DVBSx_EC_GeneralFail if the Availink device is not in the blind scan functional mode.

Remarks:

This function sets a cancel flag for the blind scan process. The scan does not actually stop until the current scan section is done. After cancelling, it is recommended that the user continue to call the AVL_DVBSx_IBase_GetScanStatus function until the AVL_DVBSx_BlindScanInfo progress field shows 100%. Any results found up to that point will be valid and the scan can be restarted using the start frequency returned.

7.3.2.2GetScanStatus

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IBlindScan_GetScanStatus  (
struct AVL_DVBSx_BlindScanInfo *  pBSInfo,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function queries the internal blind scan procedure for status information. The function updates all members of the AVL_DVBSx_BlindScanInfo structure to which the pBSInfo input parameter points. Please refer to Section 7.2.6 AVL_DVBSx_BlindScanInfo for details regarding individual members of the AVL_DVBSx_BlindScanInfo structure.

Parameters:

pBSInfo; Pointer to the object in which the blind scan status is to be stored.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the blind scan status is being queried.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the blind scan status has been retrieved; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.2.3ReadChannelInfo

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IBlindScan_ReadChannelInfo  (
AVL_uint16  uiStartIndex,
AVL_puint16  pChannelCount,
struct AVL_DVBSx_Channel *  pChannel,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function retrieves the blind scan results.

Parameters:

uiStartIndex; The blind scan results are stored in an array that is internal to the Availink device. This parameter indicates the array index at which to retrieve the results.
pChannelCount; Indicates the number of channel results to retrieve. The function updates this value with the actual number of channel results that are reported.
pChannel; Pointer to an object in which the blind scan results are to be stored.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the blind scan results are being retrieved.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the blind scan results have been retrieved; returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.2.4Reset

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBlindScan_Reset  (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function resets the Availink device's internal blind scan state variables. In a typical blind scan operation, the function AVL_DVBSx_IBlindScan_Scan is called repeatedly inside a loop. The user must call the function AVL_DVBSx_IBlindScan_Reset one time just outside the start of the loop.

Please refer to Figure 7-1 as well as the blind scan example provided with the SDK for more details regarding the blind scan procedure.

Parameters:

pAVLChip; A pointer to the chip for which the internal blind scan state variables are being reset.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device's internal blind scan state variables are reset; returns AVL_DVBSx_EC_I2CFail if there is an I2C communication error.

7.3.2.5Scan

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBlindScan_Scan  (
struct AVL_DVBSx_BlindScanPara *  pBSPara,
AVL_uint16 uiTunerLPF_100kHz,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function performs a blind scan operation.   Prior to calling this function the user must place the Availink device into the blind scan functional mode by calling the function AVL_DVBSx_IBase_SetFunctionalMode. If the device has not been placed in the blind scan functional mode, then the AVL_DVBSx_IBlindScan_Scan function returns the AVL_DVBSx_EC_GeneralFail error code.

Once the AVL_DVBSx_IBlindScan_Scan function is successfully called, the user may call the function AVL_DVBSx_IBlindScan_GetScanStatus to retrieve information regarding the current blind scan status.

Please note that the tuner LPF bandwidth setting must be passed as an input parameter to the IBlindScan_Scan function. This ensures that the Availink device computes the tuner step size properly.   The user is advised to configure the tuner LPF bandwidth to the maximum bandwidth over which the tuner frequency response is flat.

Parameters:

pBSPara; A pointer to the blind scan configuration parameters.

uiTunerLPF_100kHz; The tuner LPF bandwidth setting in units of 100 kHz.
pAVLChip; A pointer to the AVL_DVBSx_Chip object on which the blind scan operation is being performed.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the blind scan parameters and the blind scan command are successfully sent to the Availink device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if the blind scan command could not be sent to the device because the device is still processing a previous command; Returns AVL_DVBSx_EC_GeneralFail if the device is not in the blind scan functional mode or if the configured blind scan start frequency (m_uiStartFreq_100kHz member of the pBSPara object) is larger than the configured blind scan stop frequency (m_uiStopFreq_100kHz member of the pBSPara object).

Remarks:

Before performing a blind scan operation, the user must properly configure the blind scan spectrum inversion setting by calling the function IBlindScan_SetSpectrumInversion. Please see the description of the function IBlindScan_SetSpectrumInversion for more details regarding this.

The scan function is used to find carrier channels within the L-band signal into the tuner. It reports the L-band RF frequency, signal standard, and the symbol rate of each carrier frequency it detects. After it has found all of the channels, the STB application needs to lock each channel to extract the program channels within each carrier channel. Blind scan does not report the IQ setting (normal/invert) for each carrier channel so the user has two choices:

1. Try to lock with each IQ setting and see which works.
2. Enable automatic IQ Swap mode when calling the AVL_DVBSx_IRx_LockChannel function.

7.3.2.6SetSpectrumInversion

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IBlindScan_SetSpectrumInversion  (
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_IBlindscanSpectrum
enumSpectrumInversion
)
```

This function configures the Availink device in blind scan mode to indicate whether the received signal spectrum is inverted. The factors which can cause spectrum inversion are the specific tuner device and the connection method between the tuner and the Availink device. By default the Availink device assumes that the received signal spectrum is not inverted. The device spectrum inversion setting must be set to inverted if one and only one of the following is true:

1. The spectrum of the tuner baseband output is reversed from the spectrum of the tuner RF input signal
2. The user swaps the I and Q inputs between the tuner and the Availink device

If both are true, the spectrum inversion setting must be configured to normal (not inverted).

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the blind scan spectrum inversion setting is being configured.
enumSpectrumInversion; Configures the Availink device to indicate whether the received signal spectrum is inverted.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the blind scan spectrum inversion setting is configured successfully;
Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.3 IBSP

This interface is the board support package (BSP). It predefines a group of functions used by the entire SDK. The BSP must be implemented by the user in accordance with their platform specification.

Table 7-13: IBSP Interface Functions

Name	**Summary**	**Behavior**
Delay	Implements a delay.	B
Dispose	Frees all BSP resources.	B
I2CRead	Performs an I2C read operation.	B
I2CWrite	Performs an I2C write operation.	B
Initialize	Performs any initialization which is needed before other BSP functions can be called.	B
InitSemaphore	Initializes a semaphore.	B
ReleaseSemaphore	Releases a semaphore resource.	B
WaitSemaphore	Queries and waits for a semaphore.	B

7.3.3.1 Delay

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_Delay  (
AVL_uint32  uiMS
)
```

This function implements a delay in units of milliseconds.

Parameters:

uiMS; The delay period in units of milliseconds.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the delay has been initiated.

7.3.3.2Dispose

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_Dispose  (
void
)
```

This function destroys all resources allocated by the AVL_DVBSx_IBSP_Initialize function as well as by other BSP operations.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the allocated resources have been destroyed.

Remarks:
This function should never be called inside the SDK. The user can re-declare this function to any prototype.

7.3.3.3I2CRead

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_I2CRead  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_puchar  pucBuff,
AVL_puint16  puiSize
)
```

This function performs an I2C read operation.

Parameters:
 pAVLChip; Pointer to the Availink device for which the read operation is being performed.
 pucBuff; Pointer to a buffer in which to place the read data.
 puiSize; The number of bytes to be read. The function updates this value with the number of bytes actually read. If there is an error, the function sets this value to zero.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the read operation is successful; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:
This function should perform a direct I2C read operation without first writing the device internal address. The Availink SDK automatically handles writing the device internal address prior to performing the read operation.

7.3.3.4I2CWrite

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_I2CWrite  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_puchar  pucBuff,
AVL_puint16  puiSize
)
```

This function performs an I2C write operation.

Parameters:

pAVLChip; Pointer to the Availink device for which the write operation is being performed.
pucBuff; Pointer to a buffer which contains the data to be written.
puiSize; The number of bytes to be written. The function updates this buffer with the number of bytes actually written. If there is an error, the function sets this value to zero.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the write operation is successful; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.3.5Initialize

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_Initialize  (
void
)
```

This function performs any initialization that is needed before other BSP functions can be called.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the initialization is successful.

Remarks:

This function should never be called inside the SDK. The user can re-declare this function to any prototype.

7.3.3.6InitSemaphore

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_InitSemaphore  (
AVL_psemaphore  pSemaphore
)
```

This function initializes a semaphore object.

Parameters:

pSemaphore; A pointer to the AVL_semaphore object to be initialized.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the initialization is successful.

Remarks:

All of the semaphore objects should be initialized with 1 as the maximum count and the initialized state should be signaled. In particular, after initialization, the first query should succeed.

7.3.3.7ReleaseSemaphore

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_ReleaseSemaphore
(
AVL_psemaphore  pSemaphore
)
```

This function releases the semaphore.

Parameters:
 pSemaphore; A pointer to the AVL_semaphore object which is being released.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the semaphore has been released.

7.3.3.8WaitSemaphore

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IBSP_WaitSemaphore  (
AVL_psemaphore  pSemaphore
)
```

This function queries the semaphore. If the semaphore is held by others, then the function should be blocked until the semaphore is available.

Parameters:
 pSemaphore; A pointer to the AVL_semaphore object that is being queried.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the wait operation was successful.

7.3.4 IDiseqc

This interface controls the Diseqc interface of the Availink device. The Availink device's Diseqc interface is fully compatible with the Diseqc 2.0 specification. The IDiseqc interface also includes LNB control functionality.

Table 7-14: IDiseqc Interface Functions

Name	Summary	Behavior
GetLNBOut	Reads the output voltage level setting of the LNB_CNTRL_0 pin	B
GetRxStatus	Queries the Diseqc receive status in modulation mode	B
GetTxStatus	Queries the Diseqc transmit status	B
Initialize	Initializes the Diseqc component with a set of specified configuration parameters	B
ReadModulationData	Reads data from the Diseqc input FIFO	B
SendModulationData	Sends data in modulation mode	NB
SendTone	Sends data in tone mode	NB
SetLNBOut	Sets the voltage level of the LNB_CNTRL_0 output pin	B
StartContinuous	Start transmitting continuous	B

	waveform	
StopContinuous	Stop transmitting continuous waveform	B

7.3.4.1GetLNBOut

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_GetLNBOut  (
AVL_puchar  puiOut,
struct AVL_DVBSx_Chip *  pAVLChip
 )
```

This function reads the output voltage setting of the LNB_CNTRL_0 pin. This pin is typically used to control circuitry which powers the LNB with either a +13 V or a +18 V signal.

Parameters:

puiOut; A pointer to a variable in which to store the pin output voltage level (0 – Output level is low, 1 – Output level is high).
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the voltage level of pin LNB_CNTRL_0 is being read.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the voltage level for pin LNB_CNTRL_0 has been read; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.4.2GetRxStatus

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_GetRxStatus  (
struct AVL_DVBSx_Diseqc_RxStatus *  pRxStatus,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function queries the status of the Diseqc receiver. The status information includes an indication of whether the Diseqc receiver timeout window has expired as well as the received data, if any.

Parameters:
pRxStatus; A pointer to an object in which to store the Diseqc receiver status information.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the Diseqc receiver status is being queried.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Diseqc receiver status has been retrieved; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_GeneralFail if the Diseqc component is not in modulation mode.

7.3.4.3GetTxStatus

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_GetTxStatus  (
struct AVL_DVBSx_Diseqc_TxStatus *  pTxStatus,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function queries the status of the Diseqc transmitter. The status information includes an indication of whether the Diseqc transmit data FIFO is empty.

Parameters:
pTxStatus; A pointer to an object in which to store the Diseqc transmitter status information.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the Diseqc transmitter status is being queried.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Diseqc transmitter status has been retrieved; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.4.4Initialize

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_Initialize  (
const struct AVL_DVBSx_Diseqc_Para *  pDiseqcPara,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function initializes the Diseqc component using the configuration parameters in pDiseqcPara.

Parameters:
pDiseqcPara; A pointer to the Diseqc configuration parameters.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the Diseqc interface is being initialized.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Diseqc interface has been initialized; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.4.5ReadModulationData

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IDiseqc_ReadModulationData  (
AVL_puchar  pucBuff,
AVL_puchar  pucSize,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function reads data from the Diseqc input FIFO.

Parameters:
pucBuff; A pointer to a buffer in which to store the read data.
pucSize; The number of bytes to read from the FIFO. The maximum size is 8. The function updates this parameter to indicate the number of bytes that have actually been read.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the Diseqc input FIFO is being read.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Diseqc input FIFO has been read; Returns AVL_DVBSx_I2C_Fail if there is an I2C communication problem; Returns

AVL_DVBSx_EC_GeneralFail if the Diseqc interface is not in modulation mode or if the Diseqc interface is still receiving the data.

Remarks:
Availink recommends the user call the function IDiseqc_GetRxStatus before calling IDiseqc_ReadModulationData to ensure that the Diseqc receive operation is complete.

7.3.4.6SendModulationData

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IDiseqc_SendModulationData  (
const AVL_puchar  pucBuff,
AVL_uchar  ucSize,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function sends data to the Diseqc bus.

Parameters:
pucBuff; Pointer to an array which contains the data to be sent to the Diseqc bus.
ucSize; The number of bytes to be sent. The maximum is 8.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which data is to be sent to the Diseqc bus.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the data has been sent; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if it is not safe to switch the Diseqc mode because the last transmission is not

yet complete; Returns AVL_DVBSx_EC_MemoryRunout if ucSize is larger than 8.

7.3.4.7SendTone

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_SendTone  (
AVL_uchar  ucTone,
AVL_uchar  ucCount,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to transmit the specified Diseqc tone.

Parameters:
ucTone Specifies the tone to be transmitted (0 - Tone_0, 1 - Tone_1).
ucCount The number of tones to be transmitted. The maximum is 8.
pAVLChip A pointer to the AVL_DVBSx_Chip object for which the Diseqc tones are to be transmitted.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to transmit the Diseqc tones; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_MemoryRunout if ucCount is larger than 8; Returns AVL_DVBSx_EC_Running if it is not safe to switch the Diseqc mode because the last transmission is not complete yet.

7.3.4.8SetLNBOut

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_SetLNBOut  (
AVL_uchar  uiOut,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function sets the output level of the LNB_CNTRL_0 pin.

Parameters:
uiOut; Controls the level of pin LNB_CNTRL_0 (0 – Set output level low, 1 – Set output level high).
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the output level of pin LNB_CNTRL_0 is being set.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the pin level has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.4.9StartContinuous

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IDiseqc_StartContinuous  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

Configures the Availink device to output a continuous 22 kHz Diseqc waveform.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to output the waveform.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to transmit the waveform; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if it is not safe to switch the Diseqc mode because the last transmit is not complete yet.

7.3.4.10 StopContinuous

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IDiseqc_StopContinuous
(
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to stop outputting a continuous 22 kHz Diseqc waveform.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to stop outputting the waveform.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to stop transmitting the waveform; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5 IRx

This interface provides access to the receiver portion of the Availink device. The functions are used to configure receiver components, to lock channels, and to retrieve the status of the receiver.

Many of the functions in this interface may be called regardless of whether the device is in the demodulator functional mode or the blind scan functional mode. Four of the functions in this interface can only be called when the device is in the demodulator functional mode. These include the functions LockChannel, ResetErrorStat, ResetDVBSBER, and SetDishPointingMode. If any of these functions is called while the device is in the blind scan functional mode, the respective function returns the error code AVL_DVBSx_EC_GeneralFail. For more information regarding how to set the functional mode of the Availink device, please see the description for the function SetFunctionalMode.

Table 7-15: IRx Interface Functions

Name	Summary	Behavior
ReleaseMpegOutput	Configures the device to release the MPEG output interface.	B
DisableMpegPersistentClockMode	Disables MPEG persistent clock mode.	B
DriveMpegOutput	Configures the device to drive the MPEG output interface.	B
DriveRFAGC	Configures the device to drive the RF AGC output.	B
DriveRFAGC `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_DriveRFAGC ( const struct AVL_DVBSx_Chip * pAVLChip )` This function configures the Availink device to drive the RF AGC output pin. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to drive the	Enables MPEG persistent clock mode.	B

RF AGC output pin. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to drive the RF AGC output pin; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. EnableMpegPersistentClockMode		
GetBER	Reads back the BER	B
GetDeviceID	Reads the Availink device ID.	B

7.3.5.1GetDeviceID `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetDeviceID ( const struct AVL_DVBSx_Chip * pAVLChip, AVL_puint32 puiDeviceID )` This function reads the Availink device ID specific to the customer. Availink can customize the device ID to meet the needs of individual customers. Please contact your Availink sales representative for more details regarding custom device IDs. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the custom device ID is	Reads back the DVBS BER	B

being read. puiDeviceID; A pointer to a variable in which to store the device ID. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device ID has been read; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. GetDVBSBER		
GetIQ_Imbalance	Retrieves the IQ imbalance of the received signal.	B
GetIQ_Swap	Retrieves the IQ swap status of the received signal.	B

7.3.5.2GetIQ_Imbalance `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetIQ_Imbalance ( const struct AVL_DVBSx_Chip * pAVLChip, AVL_pint16 piAmplitude, AVL_pint16 piPhase )` This function reads the IQ imbalance of the received signal. The function returns the amplitude imbalance and the phase imbalance. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the IQ imbalance of the received signal is being	Checks the channel lock status. i.e., locked or not locked	B

read. piAmplitude; A pointer to a variable in which to store the amplitude imbalance. The amplitude imbalance is returned in units of dB, but is scaled by a factor of 100. For example, a value of 300 means an amplitude imbalance of 3 dB. piPhase; A pointer to a variable in which to store the phase imbalance. The phase imbalance is returned in units of degrees, but is scaled by a factor of 100. For example, a value of 200 means a phase imbalance of 2 degrees. **Returns:** AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the IQ imbalance has been retrieved from the Availink device; Return		

AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.3GetIQ_Swap

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_GetIQ_Swap (
const struct
AVL_DVBSx_Chip *
pAVLChip,
AVL_puint16
puiIQ_Swap
)
```

This function retrieves the IQ swap status of the received signal. This function is useful if the user enabled automatic IQ swap when locking to the channel. After channel lock has been achieved, the user may call GetIQ_Swap to determine whether the

Availink device had to internally swap the I and Q channels of the received signal. The user is advised to refrain from calling this function before channel lock because the reported IQ swap status is unreliable. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the IQ swap status is being retrieved. puiIQ_Swap; A pointer to a variable in which to store the IQ swap status (0 – Not swapped, 1 – Swapped). **Returns:** AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the IQ swap status has been retrieved; Return		

AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. GetLockStatus		
GetPER	Reads back the PER	B
GetRFOffset	Get the RF frequency offset	B
GetScatterData	Read back scatter data.	B
GetSignalInfo	Get the general information of the input signal	
GetSignalLevel	Reads back the input RF signal level	B
GetSNR	Gets the signal SNR	B
Initialize	Initializes the receiver components	B
LockChannel	Locks to a channel	NB
ReleaseRFAGC	Configures the device to release the RF AGC output pin	B

7.3.5.4ReleaseRFAGC `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_ReleaseRFAGC ( const struct AVL_DVBSx_Chip * pAVLChip )` This function configures the Availink device to release the RF AGC output pin, placing it in a high impedance state. This allows the RF AGC signal to be driven by other devices which may be sharing the same tuner with the Availink device. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to release the RF AGC output pin.	Resets the DVBS BER counters associated with the GetDVBSBER function	NB

Returns: AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device has been configured to release the RF AGC pin; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. ResetDVBSBER		
ResetErrorStat	Resets the BER and PER counters associated with the GetBER and GetPER functions	NB
SetAdaptivePowerSaveMode	Sets the adaptive power save mode	B

7.3.5.5SetAdaptivePowerSaveMode `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetAdaptivePowerSaveMode ( struct AVL_DVBSx_Channel *psChannel, AVL_uint16 uiEnable )` This function configures whether the adaptive power save mode is enabled or disabled according to the input parameter uiEnable. When adaptive power save mode is enabled, the Availink device conserves power by reducing the frequency of its internal clock signals in a manner that is proportionate to	Sets the channel lock mode	B

the symbol rate of the incoming signal. The adaptive power save mode provides reduced power consumption for symbol rates 15 MHz and lower. For symbol rates greater than 15 MHz, the adaptive power save mode provides no additional reduction in power consumption. The function SetAdaptivePowerSaveMode updates the m_Flags member of the psChannel input parameter to reflect the adaptive power save mode setting. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new adaptive power save mode setting is used. Please		

note that the new adaptive power save mode only applies to *subsequent* calls to the LockChannel function. If the adaptive power save mode is changed after the channel has been locked, the change has no effect. When using adaptive power save mode or when toggling the adaptive power save mode setting, the user *must* halt the Availink device before calling the LockChannel function. This can be accomplished by calling the IBase_Halt function. Halting the Availink device ensures that the Availink internal clock signals can be configured correctly. If the Availink device has not been halted and the internal clock signals		

cannot be configured properly, the LockChannel function returns the AVL_DVBSx_EC_GeneralFail error code. For more details, please see the function descriptions for IBase_Halt and IRx_LockChannel. **Parameters:** psChannel; A pointer to the channel object for which the adaptive power save mode is being configured. uiEnable; Controls whether adaptive power save mode is enabled or disabled (0 – Disable, 1 – Enable). **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the adaptive power save mode is		

set. **Remarks:** This function has not been implemented yet. However, it will be incorporated in the future release of AVL DVBSx SDK. SetChannelLockMode		
SetFreqSweepRange	Set the carrier frequency sweep range	B
SetMpegBitOrder	Configures the output bit order of the MPEG data bytes.	B
SetMpegErrorPolarity	Configures the polarity of the MPEG error signal.	B

7.3.5.6SetMpegBitOrder `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegBitOrder ( const struct AVL_DVBSx_Chip * pAVLChip, enum AVL_DVBSx_MpegMode enumMpegMode, enum AVL_DVBSx_MpegBitOrder enumMpegBitOrder )` This function configures the output bit order of the MPEG data. The user can set the bit order to either normal or invert. The meaning differs depending on whether the MPEG interface is configured to output data in serial mode or parallel mode. Please	Set up the MPEG output mode	B

refer to the description of AVL_DVBSx_MpegBit Order for more details. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the MPEG bit order is being configured. enumMpegMode; Indicates whether the MPEG interface is being operated in parallel mode or serial mode. enumMpegBitOrder; Specifies the output bit order of the MPEG data bytes. Please refer to the description of AVL_DVBSx_MpegBit Order for more details. **Returns:** AVL_DVBSx_ErrorCod e, Return AVL_DVBSx_EC_OK if		

the MPEG output bit order has been configured; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. **7.3.5.7 SetMpegErrorPolarity** `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegErrorPolarity ( const struct AVL_DVBSx_Chip * pAVLChip, enum AVL_DVBSx_MpegErrorPolarity enumErrorLockPolarity, enum AVL_DVBSx_MpegErrorPolarity enumErrorUnlockPolarity )` This function		

configures the polarity of the MPEG error signal. This function allows the user to configure the Availink device with the error signal polarity that should be used when the demod is not locked to the channel as well as the error signal polarity that should be used when the demod is locked to the channel. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the polarity of the MPEG error signal is being configured. enumErrorLockPolarity; The MPEG error signal polarity to be used during periods in which the Availink device is locked to a channel. Please refer to the description of		

AVL_DVBSx_MpegErrorPolarity. enumErrorUnlockPolarity; The MPEG error signal polarity to be used during periods in which the Availink device is not locked to a channel. Please refer to the description of AVL_DVBSx_MpegErrorPolarity. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the polarity of the MPEG error signal has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. SetMpegMode		
SetMpegPulldown	Configures whether the internal MPEG interface pull-down resistors are enabled.	B

SetMpegSerialPin	Selects the pin on which MPEG data is output in serial mode.	B
SetMpegValidPolarity	Sets the polarity of the MPEG valid signal in TSP mode.	B

7.3.5.8 SetMpegPulldown `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegPulldown ( const struct AVL_DVBSx_Chip * pAVLChip, enum AVL_DVBSx_MpegPulldown enumPulldownState )` The Availink device provides internal pull-down resistors on the MPEG interface signals. The pull-down resistors may be enabled or disabled by calling the SetMpegPulldown function. When the Availink device is reset, the pull-down resistors are enabled by default. **Parameters:**	Sets the polarization of the RFAGC control	B

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the internal pull-down resistors are being configured. enumPulldownState; Indicates whether the internal pull-down resistors should be enabled or disabled. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the I2C transaction is successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. **7.3.5.9SetMpegSerialPin** `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegSerialPin ( const struct`		

`AVL_DVBSx_Chip * pAVLChip,`
`enum AVL_DVBSx_MpegSerialPin enumSerialPin`
`)`
 	
This function selects the pin on which MPEG data is output in serial mode. The MPEG data can be output on pin MPEG_DATA_7 or pin MPEG_DATA_0.
 	
Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the output data pin is being selected.
enumSerialPin; Indicates the pin on which to output the MPEG data in serial mode.
 	
Returns:
AVL_DVBSx_ErrorCode, Returns		

AVL_DVBSx_EC_OK if the MPEG output pin has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. **7.3.5.10 SetMpegValidPolarity** `AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegValidPolarity ( const struct AVL_DVBSx_Chip * pAVLChip, enum AVL_DVBSx_MpegValidPolarity enumValidPolarity )` This function configures the polarity of the MPEG valid signal when the MPEG interface is configured		

to operate in TSP mode. Calling this function has no effect if the MPEG interface is configured to operate in TS mode. **Parameters:** pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the polarity of the MPEG valid signal is being configured. enumValidPolarity; The polarity of the MPEG valid signal in TSP mode. Please see the description of AVL_DVBSx_MpegValidPolarity for more details. **Returns:** AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the polarity of the MPEG valid signal has been configured;		

Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem. SetRFAGCPola		
SetDishPointingMode	Sets the dish pointing mode to either enabled or disabled	B

7.3.5.11 ReleaseMpegOutput

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_ReleaseMpegOutput
(
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to release the MPEG output interface, placing it in a high impedance state. This allows the MPEG bus to be driven by other devices which may be sharing the MPEG bus with the Availink device.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to release the MPEG output interface.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device register write operation was successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.12 DisableMpegPersistentClockMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_DisableMpegPersistentClockMode (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

7.3.5.13 This function disables MPEG persistent clock mode. For more details regarding MPEG persistent clock mode, please see the description of the function DriveRFAGC

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_DriveRFAGC (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to drive the RF AGC output pin.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to drive the RF AGC output pin.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to drive the RF AGC output pin; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

EnableMpegPersistentClockMode.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which MPEG persistent clock mode is being disabled.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device register write operation was successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.14 DriveMpegOutput

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_DriveMpegOutput (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to drive the MPEG output interface.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to drive the MPEG output interface.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device register write operation was successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.15 DriveRFAGC

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_DriveRFAGC (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to drive the RF AGC output pin.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to drive the RF AGC output pin.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device has been configured to drive the RF AGC output pin; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.16 EnableMpegPersistentClockMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_EnableMpegPersistentClockMode (
AVL_uint16 uiMpegDataClkFreq_10kHz,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function enables MPEG persistent clock mode.

MPEG persistent clock mode enhances Availink device compatibility with those conditional access modules which require a persistent MPEG clock signal. When MPEG persistent clock mode is enabled, the Availink device begins to output an MPEG clock signal during signal acquisition. The MPEG clock frequency is set to the value specified by the uiMpegDataClkFreq_10kHz input parameter to the EnableMpegPersistentClockMode function call. The MPEG data and valid signals remain low during signal acquisition. Once channel lock is achieved, the MPEG data and valid signals are no longer held low, and the Availink device automatically adjusts the MPEG clock frequency in accordance with the actual MPEG data rate of the received signal.

Parameters:

uiMpegDataClkFreq_10kHz; The desired MPEG clock frequency during signal acquisition in units of 10 kHz.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which MPEG persistent clock mode is being enabled.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device register write operation was successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.17   GetBER

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetBER (
AVL_puint32  puiBER,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function is provided for BER laboratory testing. The function requires the use of a transmitter that can generate either an LFSR15 or LFSR23 data stream. The function reports an accurate BER calculation result when the input signal data is comprised of either an LFSR15 or LFSR23 data stream.

Parameters:
 puiBER carries back the current BER value. The value is scaled by 1e-9. For example, reading back decimal 123456 means the BER value is 0.000123456.
 pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if everything is OK; Returns AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
It may take a few seconds for the chip to calculate a stable BER value after FEC lock. This function reports a value of 0 before a stable BER is calculated.

7.3.5.18 GetDeviceID

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetDeviceID (
const struct AVL_DVBSx_Chip *  pAVLChip,
AVL_puint32 puiDeviceID
)
```

This function reads the Availink device ID specific to the customer. Availink can customize the device ID to meet the needs of individual customers.   Please contact your Availink sales representative for more details regarding custom device IDs.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the custom device ID is being read.
puiDeviceID; A pointer to a variable in which to store the device ID.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the device ID has been read; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.19 GetDVBSBER

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetDVBSBER (
AVL_puint32  puiBER,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function reports an estimated BER when a DVBS input signal is applied to the receiver. The DVBS bit error rate estimate is performed after Viterbi decoding but before Reed Solomon decoding. The function does not require the input of a known test pattern to the receiver, and thus can be used to estimate BER for on-air DVBS signals.

Parameters:

puiBER carries back the current BER value. The value is scaled by 1e-9. For example, reading back decimal 123456 means the BER value is 0.000123456.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if everything is OK; Returns AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
If the receiver is not locked to a DVBS signal, the function reports back a BER value of 0xFFFFFFFF.

7.3.5.20 GetIQ_Imbalance

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetIQ_Imbalance (
const struct AVL_DVBSx_Chip *  pAVLChip,
AVL_pint16 piAmplitude,
AVL_pint16 piPhase
)
```

This function reads the IQ imbalance of the received signal. The function returns the amplitude imbalance and the phase imbalance.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the IQ imbalance of the received signal is being read.
piAmplitude; A pointer to a variable in which to store the amplitude imbalance. The amplitude imbalance is returned in units of dB, but is

scaled by a factor of 100. For example, a value of 300 means an amplitude imbalance of 3 dB.
piPhase; A pointer to a variable in which to store the phase imbalance. The phase imbalance is returned in units of degrees, but is scaled by a factor of 100. For example, a value of 200 means a phase imbalance of 2 degrees.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the IQ imbalance has been retrieved from the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.21 GetIQ_Swap

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetIQ_Swap (
const struct AVL_DVBSx_Chip *  pAVLChip,
AVL_puint16 puiIQ_Swap
)
```

This function retrieves the IQ swap status of the received signal. This function is useful if the user enabled automatic IQ swap when locking to the channel. After channel lock has been achieved, the user may call GetIQ_Swap to determine whether the Availink device had to internally swap the I and Q channels of the received signal. The user is advised to refrain from calling this function before channel lock because the reported IQ swap status is unreliable.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the IQ swap status is being retrieved.

puiIQ_Swap; A pointer to a variable in which to store the IQ swap status (0 – Not swapped, 1 – Swapped).

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the IQ swap status has been retrieved; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.22 GetLockStatus

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetLockStatus (
AVL_puint16  puiLockStatus,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Checks the current Availink device lock status.

Parameters:
puiLockStatus; Carries back the lock status. If the Availink device has locked to the signal, puiLockStatus will be 1, otherwise, it will be 0.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:

This function should be called after the AVL_DVBSx_IRx_LockChannel has been called.

7.3.5.23 GetPER

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetPER (
AVL_puint32  puiPER,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Reads back the current PER calculation result.

Parameters:

puiPER; Carries back the current PER value. The value is scaled by 1e-9. For example, reading back decimal 123456 means the PER value is 0.000123456.

pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:

It takes a few seconds for the chip to calculate a stable PER value after the FEC lock. Function will give 0 before a stable PER is calculated.

7.3.5.24 GetRFOffset

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetRFOffset (
AVL_pint16  piRFOffset_100kHz,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Get the RF frequency offset. This is typically used to adjust the tuner to center the signal in the tuner bandwidth for best performance.

Parameters:
piRFOffset_100kHz; Carries back the offset of the RF frequency in the unit of 100kHz. For example, if the signal RF frequency is 1000M, the tuner is at 998M, and the spectrum inversion is configured consistently with the hardware, this value will be -20 (20 x 100kHz = -2M). If the spectrum inversion is configured oppositely to the hardware, the offset will be 20. For the setup of the spectrum inversion register, please refer to the SDK function AVL_DVBSx_IBlindScan_Scan for more information.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
Call this function after the Availink device is locked to the input signal.

7.3.5.25 GetScatterData

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetScatterData (
AVL_puchar  ucpData,
AVL_puint16  puiSize,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Reads back the scatter data from the Availink device. It is useful to visualize the quality of the input signal.

Parameters:
ucpData; Carries back the scatter data. The buffer size MUST >= 2*(*puiSize) because there are two bytes per set of the IQ. The first (*pSize) bytes in the buffer are I values, and the following (*puiSize) bytes are Q values.
puiSize; As input tells the function how many sets of IQ the caller likes to read back; As output, tells the caller how many sets of IQ are read back.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
Normally, the size of the scatter data will be 132 sets of IQ. i.e. 264 bytes.

7.3.5.26 GetSignalInfo

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetSignalInfo (
struct AVL_DVBSx_SignalInfo *  pSignalInfo,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Get the current locked signal information.

Parameters:
pSignalInfo; Carries back the detected signal information. Refer to AVL_DVBSx_Signal_Info.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
Call this function after Availink device is locked to the input signal.

7.3.5.27 GetSignalLevel

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetSignalLevel  (
AVL_puint16  puiRFSignalLevel,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function gets the RF signal level.

Parameters:
puiRFSignalLevel; Carries back the signal level as a value ranging from 0 to 65535 with 0 corresponding to the weakest signal level and 65535 corresponding to the strongest signal level.

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the signal level is being read.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the signal level is successfully read; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:

This function returns a value in the range 0 to 65535, with 0 corresponding to the weakest signal and 65535 corresponding to the strongest signal. The respective signal level in units of dbm varies with the tuner configuration and board design. The user needs to obtain the relationship from the Tuner data sheet or through measurement. Figure 7-2 shows the typical relationship measured in a reference design using a Sharp BS2S7HZ6306 tuner.

This function is also useful during dish pointing as the returned signal level can be used as a reference to identify optimal dish positioning. When using this function for dish pointing, special care must be taken. First, it is necessary to enable dish pointing mode by calling the function SetDishPointingMode. Next, the GetSignalLevel function can be called as needed to guide the user during coarse dish pointing adjustments. Once the channel is locked, the developer is advised to repeatedly call the function GetSNR to guide fine adjustments to the dish position. Finally, once dish pointing is complete, it is necessary to disable dish pointing mode by again calling the function SetDishPointingMode.

Enabling the Availink dish pointing mode forces the in-chip RF AGC circuit to converge faster. It is important to disable dish pointing mode before returning to normal operation, as the faster RF AGC convergence can result in a slight performance degradation.

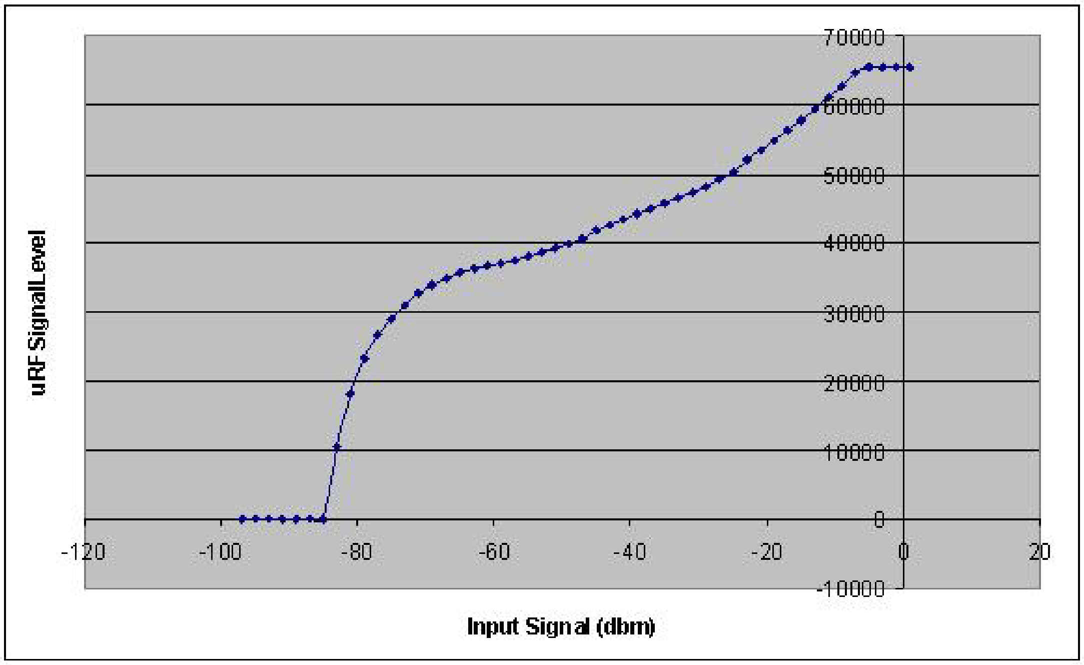

Figure 7-2: Signal Level for Ref. Design Using Sharp BS2S7HZ6306 Tuner

7.3.5.28 GetSNR

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_GetSNR  (
AVL_puint32  puiSNR_db,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Reads back the current SNR estimate.

Parameters:
puiSNR_db; Carries back the current estimated SNR value. The value is scaled by 100. For example, reading back decimal 2578 means the SNR estimate is 25.78 db.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the SNR is being read.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the SNR is successfully read; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:
It may take a few seconds for the chip to calculate a stable SNR value after FEC lock. This function returns an SNR value of 0 before a stable SNR value is calculated.

This function is also useful during dish pointing as the returned SNR can be used as a reference for fine adjustments to the dish position. When using this function for dish pointing, special care must be taken. First, it is necessary to enable dish pointing mode by calling the function SetDishPointingMode. Next, the GetSignalLevel function may be called repeatedly to guide the user during coarse dish pointing adjustment. Once the channel is locked, the GetSNR function can be called as needed to guide fine adjustments to the dish position. Finally, once dish pointing is complete, it is necessary to disable dish pointing mode by again calling the function SetDishPointingMode.

Enabling the Availink dish pointing mode forces the in-chip RF AGC circuit to converge faster. It is important to disable dish pointing mode before returning to normal operation, as the faster RF AGC convergence can result in a slight performance degradation.

7.3.5.29 Initialize

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_Initialize  (
const struct AVL_DVBSx_Chip *  pAVLChip   )
```

Initializes the demodulator.

Parameters:
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which Availink device it is working on.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

Remarks:
This function must be called first before all other functions declared in this interface.

7.3.5.30 LockChannel

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_LockChannel  (
const struct AVL_DVBSx_Channel *  psChannel,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function locks to a channel using the parameters specified in the psChannel input parameter. This function may only be called when the device is in the demodulator functional mode. If this function is called while the device is in the blind scan functional mode, the function returns the error code AVL_DVBSx_EC_GeneralFail. Please see the description for the function AVL_DVBSx_IBase_SetFunctionalMode for details regarding how to change the Availink device functional mode.

If adaptive power save mode is enabled, or if the adaptive power save mode setting has been toggled, the user must halt the Availink device before calling the LockChannel function. Please see the function descriptions of IBase_Halt and IRx_SetAdaptivePowerSaveMode for more details.

During a channel lock operation, the Availink device can automatically detect the signal standard (DVB-S or DVB-S2). The device can also automatically detect the spectral sense of the received signal, thus automatically configuring itself to swap the I and Q signals if needed. The user can control whether either or both of these options are enabled by setting the appropriate bits in the m_Flags member of the psChannel input parameter. Please see the description for the AVL_DVBSx_Channel structure for more details regarding how these features can be enabled.

7.3.5.31 Using the m_Flags member of the psChannel input parameter, the user can also configure the channel lock mode. The lock mode may be set to either fixed mode or adaptive mode. Please see the function SetAdaptivePowerSaveMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetAdaptivePowerSaveMode (
struct AVL_DVBSx_Channel *psChannel,
AVL_uint16 uiEnable
)
```

This function configures whether the adaptive power save mode is enabled or disabled according to the input parameter uiEnable. When adaptive power save mode is enabled, the Availink device conserves power by reducing the frequency of its internal clock signals in a manner that is proportionate to the symbol rate of the incoming signal. The adaptive power save mode provides reduced power consumption for symbol rates 15 MHz and lower. For symbol rates greater than 15 MHz, the adaptive power save mode provides no additional reduction in power consumption.

The function SetAdaptivePowerSaveMode updates the m_Flags member of the psChannel input parameter to reflect the adaptive power save mode setting. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new adaptive power save mode setting is used. Please note that the new adaptive power save mode only applies to *subsequent* calls to the LockChannel function. If the adaptive power save mode is changed after the channel has been locked, the change has no effect.

When using adaptive power save mode or when toggling the adaptive power save mode setting, the user *must* halt the Availink device before calling the LockChannel function. This can be accomplished by calling the IBase_Halt function. Halting the Availink device ensures that the Availink internal clock signals can be configured correctly. If the Availink device has not been halted and the internal clock signals cannot be configured properly, the LockChannel function returns the AVL_DVBSx_EC_GeneralFail error code. For more details, please see the function descriptions for IBase_Halt and IRx_LockChannel.

Parameters:

psChannel; A pointer to the channel object for which the adaptive power save mode is being configured.
uiEnable; Controls whether adaptive power save mode is enabled or disabled (0 – Disable, 1 – Enable).

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the adaptive power save mode is set.

Remarks:

This function has not been implemented yet. However, it will be incorporated in the future release of AVL DVBSx SDK.

SetChannelLockMode for more details regarding how to configure the channel object for either mode.

If the user has configured the psChannel object for fixed lock mode, then the LockChannel function locks to the channel using a fixed frequency sweep range and a fixed set of demodulator parameters regardless of symbol rate. By default the frequency sweep range is 5 MHz, but the user may first call the function SetFreqSweepRange to set it to another value if desired. Please see the description of the function SetFreqSweepRange for details.

If the user has configured the psChannel object for adaptive lock mode, then the LockChannel function automatically adapts the frequency sweep range and other demodulator parameters to provide a faster lock mechanism for signals having symbol rates lower than 3 MHz. For signals having symbol rates greater than or equal to 3 MHz, the LockChannel function sets the frequency sweep range to 5 MHz and behaves identically to the fixed mode.

The user is advised to use only one lock mode in their set top box application. Once a mode is selected, there is generally no need to switch between modes. If the user does switch from adaptive mode to fixed mode, extra care must be taken. In particular, the user must call the function SetFreqSweep Range to manually set the frequency sweep range back to either the default value of 5 MHz or to whichever value the user desires.

If the user prefers to manually adjust the carrier frequency sweep range, they must use the fixed channel lock mode and they must call the function SetFreqSweepRange prior to calling the LockChannel function. If they use the adaptive lock mode, their frequency sweep range setting will be automatically overwritten once the LockChannel function is called.

Parameters:

psChannel; Holds the channel related parameters needed by the Availink device to lock to the input signal.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the lock operation is being performed.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the lock parameters and command are successfully sent to the Availink device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if the lock command could not be sent to the device because the device is still processing a previous command; Returns AVL_DVBSx_EC_GeneralFail if the device is not in the demodulator functional mode or if the internal clocks could not be configured because the Availink device has not been halted.

Remarks:

Calling this function commands the Availink device to lock using the channel parameters pointed to by psChannel. Use the GetLockStatus function to determine if the device has successfully locked to the channel.

If the automatic IQ swap mode is enabled by the CI_FLAG_IQ_AUTO_BIT bit of the m_Flags member of psChannel, then the device automatically detects whether the frequency spectrum is inverted. If the device determines that the spectrum is inverted, it automatically swaps the I and Q input signals. In general, using automatic IQ swap detection increases lock time by anywhere from approximately 50 ms to 1 second depending on the symbol rate. In particular, for high symbol rates (e.g. ≥20 MHz), automatic IQ swap detection increases lock time by approximately 50 ms. For low symbol rates (e.g. ≤ 2 MHz), automatic IQ swap detection increases lock time by approximately 1 second. After the channel is locked, the correct spectral invert (IQ swap) setting for the channel may be read by calling the function GetIQ_Swap.

7.3.5.32 ReleaseRFAGC

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_ReleaseRFAGC  (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function configures the Availink device to release the RF AGC output pin, placing it in a high impedance state. This allows the RF AGC signal to be driven by other devices which may be sharing the same tuner with the Availink device.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object which is being configured to release the RF AGC output pin.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the Availink device has been configured to release the RF AGC pin; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.33 ResetDVBSBER

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_ResetDVBSBER  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function resets the device’s internal DVBS BER statistics accumulation associated with the GetDVBSBER function. This function does not reset the BER statistics accumulation that is associated with the GetBER and GetPER functions. This function may only be called while the device is in the demodulator functional mode. If the device is in the blind scan functional mode and this function is called, the function returns the AVL_DVBSx_EC_GeneralFail error code.

Parameters:
pAVLChip; Pointer to the AVL_DVBSx_Chip object for which the DVBS BER is being reset.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the DVBS BER is successfully reset; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_GeneralFail if the device is not in the demodulator functional mode.

7.3.5.34 ResetErrorStat

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_ResetErrorStat  (
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function resets the Availink device’s internal BER/PER statistics accumulation associated with the GetBER and GetPER functions.

This function does not reset the BER statistics accumulation that is associated with the GetDVBSBER function.

This function may only be called while the device is in the demodulator functional mode. If the device is in the blind scan functional mode, the function returns the error code AVL_DVBSx_EC_GeneralFail.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the error statistics are being reset.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the command to reset the error statistics is successfully sent to the device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if the command to reset the error statistics could not be sent to the device because the device was still processing a previous command; Returns AVL_DVBSx_EC_GeneralFail if this function is called while the device is in the blind scan functional mode.

Remarks:
In addition to this function the BER/PER statistics are cleared automatically whenever signal lock is lost.

7.3.5.35 SetAdaptivePowerSaveMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetAdaptivePowerSaveMode (
struct AVL_DVBSx_Channel *psChannel,
AVL_uint16 uiEnable
)
```

This function configures whether the adaptive power save mode is enabled or disabled according to the input parameter uiEnable. When adaptive power save mode is enabled, the Availink device conserves power by reducing the frequency of its internal clock signals in a manner that is proportionate to the symbol rate of the incoming signal. The adaptive power save mode provides reduced power consumption for symbol rates 15 MHz and lower. For symbol rates greater than 15 MHz, the adaptive power save mode provides no additional reduction in power consumption.

The function SetAdaptivePowerSaveMode updates the m_Flags member of the psChannel input parameter to reflect the adaptive power save mode setting. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new adaptive power save mode setting is used. Please note that the new adaptive power save mode only applies to *subsequent* calls to the LockChannel function. If the adaptive power save mode is changed after the channel has been locked, the change has no effect.

When using adaptive power save mode or when toggling the adaptive power save mode setting, the user *must* halt the Availink device before calling the LockChannel function. This can be accomplished by calling the IBase_Halt function. Halting the Availink device ensures that the Availink internal clock signals can be configured correctly. If the Availink device has not been halted and the internal clock signals cannot be configured properly, the LockChannel function returns the

AVL_DVBSx_EC_GeneralFail error code. For more details, please see the function descriptions for IBase_Halt and IRx_LockChannel.

Parameters:
psChannel; A pointer to the channel object for which the adaptive power save mode is being configured.
uiEnable; Controls whether adaptive power save mode is enabled or disabled (0 – Disable, 1 – Enable).

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the adaptive power save mode is set.

Remarks:
This function has not been implemented yet. However, it will be incorporated in the future release of AVL DVBSx SDK.

7.3.5.36 SetChannelLockMode

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetChannelLockMode
(
struct AVL_DVBSx_Channel *psChannel,
AVL_DVBSx_LockMode enumChannelLockMode
)
```

This function sets the channel lock mode according to the input parameter enumChannelLockMode. The function updates the m_Flags member of the psChannel input parameter to reflect the new lock mode. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new lock mode is used. Please note that the new lock mode only applies to *subsequent* calls to the LockChannel function. If the channel lock mode is changed after the channel has been locked, then the change has no effect.

The Availink device supports two channel lock modes, a fixed mode and an adaptive mode. The default mode of operation is fixed mode. In fixed mode the frequency sweep range and other demodulator parameters remain fixed regardless of symbol rate. In adaptive mode the frequency sweep range and other demodulator parameters are adapted to provide a faster lock mechanism for signals having symbol rates lower than 3 MHz. For signals having symbol rates greater than or equal to 3 MHz, the fixed and adaptive lock modes operate identically.

The user is advised to use only one lock mode in their set top box application. Once a mode is selected, there is generally no need to switch between modes. Please see the description for the function LockChannel for more details regarding the channel lock modes.

Parameters:

psChannel; A pointer to the channel object for which the lock mode is being set.
enumChannelLockMode; The lock mode for which the channel should be configured.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the lock mode is set.

7.3.5.37 SetFreqSweepRange

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetFreqSweepRange
(
AVL_uint16 uiFreqSweepRange_10kHz,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Sets the carrier frequency sweep range for subsequent LockChannel calls. The default value is 500, which corresponds to a sweep range of ±5 MHz. This function is typically used to narrow the sweep range in cases where there are multiple narrow bandwidth signals within the default range and there is the possibility that the wrong signal will be locked. If multiple signals are detected within the specified sweep range, the Availink device will lock to the strongest one. Note that this sweep range must be large enough to accommodate the RF path frequency uncertainty.

7.3.5.38 The user should be aware the LockChannel function overwrites the frequency sweep range if the user has used the function SetChannelLockMode to change the lock mode from fixed to adaptive. If the user wishes to have manual control over the sweep range, they should always use the fixed channel lock mode. Please see the function SetAdaptivePowerSaveMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetAdaptivePowerSaveMode (
struct AVL_DVBSx_Channel *psChannel,
AVL_uint16 uiEnable
```

)

This function configures whether the adaptive power save mode is enabled or disabled according to the input parameter uiEnable. When adaptive power save mode is enabled, the Availink device conserves power by reducing the frequency of its internal clock signals in a manner that is proportionate to the symbol rate of the incoming signal. The adaptive power save mode provides reduced power consumption for symbol rates 15 MHz and lower. For symbol rates greater than 15 MHz, the adaptive power save mode provides no additional reduction in power consumption.

The function SetAdaptivePowerSaveMode updates the m_Flags member of the psChannel input parameter to reflect the adaptive power save mode setting. Thus, when a pointer to this channel object is passed as an input parameter to any subsequent calls to the LockChannel function, the new adaptive power save mode setting is used. Please note that the new adaptive power save mode only applies to *subsequent* calls to the LockChannel function. If the adaptive power save mode is changed after the channel has been locked, the change has no effect.

When using adaptive power save mode or when toggling the adaptive power save mode setting, the user *must* halt the Availink device before calling the LockChannel function. This can be accomplished by calling the IBase_Halt function. Halting the Availink device ensures that the Availink internal clock signals can be configured correctly. If the Availink device has not been halted and the internal clock signals cannot be configured properly, the LockChannel function returns the AVL_DVBSx_EC_GeneralFail error code. For more details, please see the function descriptions for IBase_Halt and IRx_LockChannel.

Parameters:
psChannel; A pointer to the channel object for which the adaptive power save mode is being configured.
uiEnable; Controls whether adaptive power save mode is enabled or disabled (0 – Disable, 1 – Enable).

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK once the adaptive power save mode is set.

Remarks:
This function has not been implemented yet. However, it will be incorporated in the future release of AVL DVBSx SDK.

SetChannelLockMode for more details.

Parameters:
uiFreqSweepRange_10kHz; Half of the frequency sweep range in units of 10kHz. The maximum value is 500 (sweep from -5MHz to +5MHz).
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the frequency sweep range is being set.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the frequency sweep range is successfully upated; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:

The input parameter uiFreqSweepRange_10kHz specifies half of the actual sweep range. The carrier frequency sweep range will be [-uiFreqSweepRange_10kHz, +uiFreqSweepRange_10kHz].

7.3.5.39 SetMpegBitOrder

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegBitOrder  (
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_MpegMode enumMpegMode,
enum AVL_DVBSx_MpegBitOrder enumMpegBitOrder
)
```

This function configures the output bit order of the MPEG data. The user can set the bit order to either normal or invert. The meaning differs depending on whether the MPEG interface is configured to output data in serial mode or parallel mode. Please refer to the description of AVL_DVBSx_MpegBitOrder for more details.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the MPEG bit order is being configured.
enumMpegMode; Indicates whether the MPEG interface is being operated in parallel mode or serial mode.
enumMpegBitOrder; Specifies the output bit order of the MPEG data bytes. Please refer to the description of AVL_DVBSx_MpegBitOrder for more details.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the MPEG output bit order has been configured; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.40 SetMpegErrorPolarity

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetMpegErrorPolarity  (
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_MpegErrorPolarity
enumErrorLockPolarity,
enum AVL_DVBSx_MpegErrorPolarity
enumErrorUnlockPolarity
)
```

This function configures the polarity of the MPEG error signal. This function allows the user to configure the Availink device with the error signal polarity that should be used when the demod is not locked to the channel as well as the error signal polarity that should be used when the demod is locked to the channel.

Parameters:

pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the polarity of the MPEG error signal is being configured.
enumErrorLockPolarity; The MPEG error signal polarity to be used during periods in which the Availink device is locked to a channel. Please refer to the description of AVL_DVBSx_MpegErrorPolarity.
enumErrorUnlockPolarity; The MPEG error signal polarity to be used during periods in which the Availink device is not locked to a channel. Please refer to the description of AVL_DVBSx_MpegErrorPolarity.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the polarity of the MPEG error signal has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.41 SetMpegMode

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegMode  (
const struct AVL_DVBSx_MpegInfo *  pMpegMode,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Set up the MPEG output mode according to the parameters in the pMpegMode structure. This is typically only required once during initialization based on the desired interface timing characteristics.

Parameters:

pMpegMode; Refer to AVL_DVBSx_MPEG_Info. Provides the MPEG output mode information. This parameter must be initialized before it is passed to the function.
pAVLChip; A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which device it is working on.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is an I2C problem.

7.3.5.42 SetMpegPulldown

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegPulldown  (
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_MpegPulldown enumPulldownState
)
```

The Availink device provides internal pull-down resistors on the MPEG interface signals. The pull-down resistors may be enabled or disabled by calling the SetMpegPulldown function. When the Availink device is reset, the pull-down resistors are enabled by default.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the internal pull-down resistors are being configured.
enumPulldownState; Indicates whether the internal pull-down resistors should be enabled or disabled.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the I2C transaction is successful; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.43 SetMpegSerialPin

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetMpegSerialPin
(
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_MpegSerialPin enumSerialPin
)
```

This function selects the pin on which MPEG data is output in serial mode. The MPEG data can be output on pin MPEG_DATA_7 or pin MPEG_DATA_0.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the output data pin is being selected.

enumSerialPin; Indicates the pin on which to output the MPEG data in serial mode.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the MPEG output pin has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.44 SetMpegValidPolarity

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetMpegValidPolarity  (
const struct AVL_DVBSx_Chip *  pAVLChip,
enum AVL_DVBSx_MpegValidPolarity enumValidPolarity
)
```

This function configures the polarity of the MPEG valid signal when the MPEG interface is configured to operate in TSP mode. Calling this function has no effect if the MPEG interface is configured to operate in TS mode.

Parameters:
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which the polarity of the MPEG valid signal is being configured.
enumValidPolarity; The polarity of the MPEG valid signal in TSP mode. Please see the description of AVL_DVBSx_MpegValidPolarity for more details.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the polarity of the MPEG valid signal has been configured; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.5.45 SetRFAGCPola

```
AVL_DVBSx_ErrorCode AVL_DVBSx_IRx_SetRFAGCPola  (
enum AVL_DVBSx_RfagcPola  enumAGCPola,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

Set the polarity of the RF AGC. This value should be set according to the specific tuner voltage/gain characteristics.

Parameters:

enumAGCPola The polarity of the RF AGC. Refer to AVL_DVBSx_RFAGCPola.
pAVLChip A pointer that points to a AVL_DVBSx_Chip object which is used to tell the function which device it is working on.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if everything is OK; Return AVL_DVBSx_EC_I2CFail if there is a I2C problem.

7.3.5.46 SetDishPointingMode

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_IRx_SetDishPointingMode  (
AVL_uchar ucMode,
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function controls whether dish pointing mode is enabled or disabled. This function may only be called while the Availink device is in the demodulator functional mode. If the device is in the blind scan functional mode, the function returns the error code AVL_DVBSx_EC_GeneralFail.

When dish pointing mode is enabled, the developer can use the functions GetSignalLevel and GetSNR to guide the dish pointing process. In particular, it is recommended that the developer use repeated calls to the GetSignalLevel function to guide the user during coarse dish pointing adjustments. After the signal is locked, the developer may use repeated calls to the GetSNR function to guide the user during fine dish pointing adjustments. Once dish pointing is complete, the developer must disable dish pointing mode by again calling the SetDishPointingMode function. Please see the descriptions for the functions GetSignalLevel and GetSNR for additional details.

Enabling the Availink dish pointing mode forces the in-chip RF AGC circuit to converge faster. It is important to disable dish pointing mode before returning to normal operation, as the faster RF AGC convergence can result in a slight performance degradation.

Parameters:

ucMode; Controls whether dish pointing mode is enabled or disabled. If ucMode is set to 1, dish pointing mode is enabled. If ucMode is set to 0, dish pointing mode is disabled.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which dish pointing mode is being configured.

Returns:
AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the dish pointing mode is set successfully; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_GeneralFail if the device is not in the demodulator functional mode.

7.3.6 II2C

The II2C interface is used to read and write data from and to the Availink device.

In general the user should always use the functions provided in the II2C interface for I2C communication with the Availink device. This is because these functions provide multi-thread protection and they eliminate the difference between big endian and little endian systems. The functions also automatically break potentially large I2C transactions into smaller I2C transactions to meet the hardware limitations defined in the BSP.

For example, if the user wishes to perform an I2C read operation of a length that is greater than MAX_II2C_READ_SIZE bytes, the respective SDK function automatically performs multiple I2C read operations, each having a length less than or equal to

MAX_II2C_READ_SIZE bytes. The results of all of these read transactions are stored in concatenated form in the user specified buffer. Similar behavior is provided for I2C write operations.

Please note that these I2C functions can only be used to perform I2C operations for the Availink device. When reading or writing data from or to the Availink device, all of the data is treated as being an unsigned data type. The user needs to cast data properly to get desired values.

Table 7-16: II2C Interface Functions

Name	**Summary**	**Behavior**
Read	Reads an AVL_uchar array from a particular address.	B
Read16	Reads an AVL_uint16 value from a particular address.	B
Read32	Reads an AVL_uint32 value from a particular address.	B
ReadDirect	Reads AVL_uchar array from the current address.	B
Write	Writes an AVL_uchar array to a particular address.	B
Write16	Writes an AVL_uint16 value to a particular address.	B
Write32	Writes an AVL_uint32 value to a particular address.	B

7.3.6.1Read

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Read  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_uint32  uiOffset,
AVL_puchar  pucBuff,
AVL_uint16  uiSize
)
```

This function reads one or more bytes from the Availink device at an internal address specified by the user.

Parameters:
pAVLChip; A pointer to the Availink device for which the read operation is being performed.
uiOffset; The Availink device internal address from where data is to be read.
pucBuff; A pointer to a buffer in which to store the read data.
uiSize; The number of bytes to read.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

Remarks:
The function first performs an I2C write operation to send the read address to the Availink device.

7.3.6.2Read16

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Read16  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_uint32  uiAddr,
AVL_puint16  puiData
)
```

This functions reads a 16-bit unsigned integer from the Availink device.

Parameters:
pAVLChip A pointer to the Availink device for which the read operation is being performed.
uiAddr The Availink device internal address from where the data is to be read.
puiData A pointer to a variable in which to store the read data.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.6.3Read32

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Read32  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_uint32  uiAddr,
AVL_puint32  puiData
)
```

This function reads a 32-bit unsigned integer from the Availink device.

Parameters:
pAVLChip A pointer to the Availink device for which the read operation is being performed.
uiAddr The Availink device internal address from where the data is to be read.

puiData A pointer to a variable in which to store the read data.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.6.4ReadDirect

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_ReadDirect  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_puchar  pucBuff,
AVL_uint16  uiSize
)
```

This function reads one or more bytes from the Availink device at the present address.

Parameters:
pAVLChip A pointer to the Availink device for which the read operation is being performed.
pucBuff A pointer to a buffer in which to store the read data.
uiSize The number of bytes to read.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.6.5Write

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Write  (
Const struct AVL_DVBSx_Chip *pAVLChip,
AVL_puchar  pucBuff,
AVL_uint16  uiSize
)
```

This function writes one or more bytes to the Availink device at an internal address specified by the user.

Parameters:
pAVLChip A pointer to the Availink device to which the data is to be written.
pucBuff A pointer to a buffer which contains the data to be written.
uiSize The number of bytes to be written.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been written to the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.6.6Write16

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Write16  (
Const struct AVL_DVBSx_Chip *pAVLChip,
AVL_uint32  uiAddr,
AVL_uint16  uiData
)
```

This function writes a 16-bit unsigned integer to the Availink device.

Parameters:
pAVLChip A pointer to the Availink device to which the data is to be written.
uiAddr The Availink device internal address to which the data is to be written.
uiData The data to be written to the Availink device.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been written to the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.6.7Write32

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2C_Write32  (
const struct AVL_DVBSx_Chip *pAVLChip,
AVL_uint32  uiAddr,
AVL_uint32  uiData
)
```

This function writes a 32-bit unsigned integer to the Availink device.

Parameters:
pAVLChip A pointer to the Availink device to which the data is to be written.
uiAddr The Availink device internal address to which the data is to be written.
uiData The data to be written to the Availink device.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been written to the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem.

7.3.7 II2CRepeater

The II2CRepeater interface controls the in chip I2C repeater component. This repeater provides a dedicated I2C bus for tuner control. Availink strongly recommends that users refrain from directly calling the functions in this interface in their STB application. Instead, users should use this interface to implement a tuner driver for the tuner device being used. The tuner driver functions should then be called from the STB application. For customer convenience, Availink provides tested tuner drivers for a variety of tuner devices.

Table 7-17: II2CRepeater Interface Functions

Name	**Summary**	**Behavior**
GetOPStatus	Checks the status of the last I2C repeater command.	B
Initialize	Initializes the I2C repeater hardware component in the Availink device.	NB
ReadData	Reads data from a tuner which does not require a repeated start condition during its read cycle.	B
ReadData_Multi	Reads data from a tuner which requires a repeated start condition during its read cycle.	B
SendData	Sends data to the I2C device connected to the I2C repeater.	NB

7.3.7.1GetOPStatus

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_II2CRepeater_GetOPStatus  (
const struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function checks if the last I2C repeater operation is finished.

Parameters:
pAVLChip A pointer to the AVL_DVBSx_Chip object for which the status of the last I2C repeater operation is being queried.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if there is no pending I2C operation; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Return AVL_DVBSx_EC_Running if the Availink device is still processing the last I2C repeater operation.

7.3.7.2Initialize

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_II2CRepeater_Initialize  (
AVL_uint16  I2CBusClock_kHz,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function Initializes the I2C repeater.

Parameters:

I2CBusClock_kHz The clock speed of the I2C bus between the tuner and the Availink device in units of kHz.
pAVLChip A pointer to the AVL_DVBSx_Chip object for which the I2C repeater is being initialized.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the initialize command has been sent to the Availink device; Returns AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Returns AVL_DVBSx_EC_Running if the initialize command could not be sent to the Availink device because the device is still processing a previous command.

Remarks:

This function must be called before any other function in this interface. This is a non-blocking function. The user is advised to call II2CRepeater_GetOPStatus to determine when I2C repeater initialization is complete.

7.3.7.3ReadData

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2CRepeater_ReadData
(
AVL_uchar  ucSlaveAddr,
AVL_puchar  pucBuff,
AVL_uint16  uiSize,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function reads data from the tuner via the I2C repeater. This function is used to read data from a tuner which does not require a repeated start condition during its read cycle.

Parameters:

ucSlaveAddr The slave address of the tuner device. Please note that the Availink device only supports a 7 bit slave address.
pucBuff Pointer to a buffer in which to store the data read from the tuner.
uiSize The number of bytes to read from the tuner. The maximum permissible value is 20.
pAVLChip A pointer to the AVL_DVBSx_Chip object for which data is being read from the tuner.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the tuner; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Return AVL_DVBSx_EC_Running if read command could not be sent to the Availink device because the device is still processing a previous command; Return AVL_DVBSx_EC_GeneralFail if uiSize is larger than 20.

Remarks:

This function triggers an I2C read operation. The read position (or device internal address) can be determined by calling the function II2CRepeater_SendData.

7.3.7.4ReadData_Multi

```
AVL_DVBSx_ErrorCode
AVL_DVBSx_II2CRepeater_ReadData_Multi  (
AVL_uchar  ucSlaveAddr,
AVL_puchar  pucBuff,
AVL_uchar ucRegAddr,
AVL_uint16  uiSize,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function reads data from the tuner via the I2C repeater. This function is used to read data from a tuner which requires a repeated start condition during its read cycle.

Parameters:

ucSlaveAddr The slave address of the tuner device. Please note that the Availink device only supports a 7 bit slave address.
pucBuff Pointer to a buffer in which to store the data read from the tuner.
ucRegAddr The address of the tuner register being read.
uiSize The number of bytes to read from the tuner. The maximum permissible value is 20.
pAVLChip A pointer to the AVL_DVBSx_Chip object for which data is being read from the tuner.

Returns:

AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the data has been read from the tuner; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem; Return AVL_DVBSx_EC_Running if the read command could not be sent to the Availink device because the device is still processing a previous command; Return AVL_DVBS_EC_GeneralFail if uiSize is larger than 20.

Remarks:
This function triggers an I2C read operation.

7.3.7.5SendData

```
AVL_DVBSx_ErrorCode AVL_DVBSx_II2CRepeater_SendData
(
AVL_uchar  ucSlaveAddr,
const AVL_puchar  ucBuff,
AVL_uint16  uiSize,
struct AVL_DVBSx_Chip *  pAVLChip
)
```

This function sends data to the tuner via the I2C repeater.

Parameters:
ucSlaveAddr; The slave address of the tuner device. Please note that the Availink device only supports a 7 bit slave address.
ucBuff; Pointer to the buffer which contains the data to be sent to the tuner.
uiSize; The number of bytes to be sent to the tuner. The maximum permissible value is 17.
pAVLChip; A pointer to the AVL_DVBSx_Chip object for which data is being sent to the tuner.

Returns:
AVL_DVBSx_ErrorCode, Return AVL_DVBSx_EC_OK if the send command has been sent to the Availink device; Return AVL_DVBSx_EC_I2CFail if there is an I2C communication problem;

Return AVL_DVBSx_EC_Running if the command could not be sent to the Availink device because the device is still processing a previous command; Return AVL_DVBSx_EC_GeneralFail if uiSize is larger than 17.

Remarks:

The tuner register address to which data is being written must be included in the input buffer ucBuff. For many tuners, the register address is specified as the first byte to be transmitted, and thus should be placed in the first byte entry of ucBuff. However, the user must refer to the datasheet for the specific tuner being used to ensure that they place the register address in the correct location within the input buffer.

7.3.8 ITuner

This interface provides helper functions for tuner control. It works together with the AVL_Tuner structure to facilitate tuner control. Please note that the ITuner interface is not the tuner driver interface. The tuner driver interface is actually AVL_Tuner.

Table 7-18: ITuner Interface Functions

Name	Summary	Behavior
CalculateLPF	Calculates the tuner lowpass filter bandwidth based on the symbol rate.	B

7.3.8.1CalculateLPF

```
AVL_DVBSx_ErrorCode AVL_DVBSx_ITuner_CalculateLPF  (
AVL_uint16  uiSymbolRate_10kHz,
struct AVL_Tuner *  pTuner
)
```

This function is provided as a helper function. It calculates a tuner LPF bandwidth that is proportional to symbol rate. The user may either use this function or implement their own technique for computing the tuner lowpass filter bandwidth.

Parameters:

uiSymbolRate_10kHz; The symbol rate in units of 10kHz.
pTuner; Pointer to the AVL_Tuner object for which to compute the lowpass filter bandwidth value. The function updates the value of the member m_uiLPF_100kHz of the tuner object.

Returns:

AVL_DVBSx_ErrorCode, Returns AVL_DVBSx_EC_OK if the tuner object is updated. This function is provided as a helper function. The user may choose to ignore this implementation and instead use their own calculation.

8 EXAMPLES

Example files are included with the SDK to demonstrate how to use the SDK to control the Availink device. Users may review these examples as a starting point to understanding the SDK or to create their own simple STB application.

There are three examples in the examples directory following installation of the Availink SDK:

1. A blind scan example which demonstrates how to perform a blind scan operation.
2. A Diseqc example which demonstrates how to control the Diseqc hardware component in the Availink device.
3. A LockSignal example which demonstrates basic operations such as:
   a) Initializing the Availink device
   b) Loading the firmware and booting up the chip
   c) Setting up parameters to make the Availink device lock to a particular signal.

All of these examples use the aardvark.net (USB I2C adapter) IBSP implementation to perform I2C operations and were tested using the Availink evaluation kit hardware. The development environment is Microsoft's VC++ 2008 express version. Users can get this free development environment from the Microsoft website.

The Object Oriented Programming (OOP) design of the SDK should allow the user to easily make these examples work with their own platform and board support package (BSP).

9 REFERENCE DOCUMENTS

The following list of documents can be used to supplement the AVL DVBSx SDK User's Guide.

1. AVL6211 DVB-S2/DVB-S Channel Receiver Datasheet, Availink document QS03-602-00001-1.2.
2. AVL6211 Low Level Programming Guide, Availink document QS03-602-00007-1.0.
3. AVL6222 DVB-S2/DVB-S Dual Channel Receiver Datasheet, Availink document QSO3-602-00004-1.0.