TEXAS INSTRUMENTS

PurePath^{TM} Console

Quick Start User's Guide

Version 140527

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products & application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

EVALUATION BOARD/KIT IMPORTANT NOTICE

Texas Instruments (TI) provides the enclosed product(s) under the following conditions:

This software is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. As such, the goods being provided are not intended to be complete in terms of required design-, marketing-, and/or manufacturing-related protective considerations, including product safety and environmental measures typically found in end products that incorporate such software.

Should this evaluation software not meet the specifications indicated in the User's Guide, the software may be returned within 30 days from the date of delivery for a full refund. THE FOREGOING WARRANTY IS THE EXCLUSIVE WARRANTY MADE BY SELLER TO BUYER AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies TI from all claims arising from the handling or use of the goods.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER PARTY SHALL BE LIABLE TO THE OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

TI currently deals with a variety of customers for products, and therefore our arrangement with the user **is not exclusive**.

TI assumes **no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein**.

Please read the User's Guide and, specifically, the Warnings and Restrictions notice in the User's Guide prior to handling the product. This notice contains important safety information about temperatures and voltages. For additional information on TI's environmental and/or safety programs, please contact the TI application engineer or visit www.ti.com/esh.

No license is granted under any patent right or other intellectual property right of TI covering or relating to any machine, process, or combination in which such TI products or services might be or are used.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

This manual describes the features and operation of the PurePath Console System Level Development Environment from Texas Instruments.

Related Documentation from Texas Instruments

The following table contains a list of data manuals that have detailed descriptions of the integrated circuits used in conjunction with the PurePath Console User's Guide. The data manuals can be obtained at the URL http://www.ti.com.

Table 1-1. Related Documentation from Texas Instruments

Part Number	Literature Number
TAS5721xx	SLOS739
TAS5731	SLOS726
TAS5548	SLES270

Contents

Figures

Tables

Chapter 1

PurePath Console Introduction

TI's PurePath Console Development Environment is a powerful, easy-to-use tool designed specifically to simplify system level tuning and integration.

1.1 PurePath Console Overview

PurePath Console is a highly integrated and easy-to-use audio development suite designed specifically to simplify the evaluation, configuration and debug process associated with the development of audio products.

PurePath Console's intuitive graphical interface makes audio design very straightforward; minimizing the need for advanced audio engineering expertise. Highly optimized audio performance, minimal power consumption and seam-less system integration are made possible with PurePath Console's many advanced control features implemented in an easy to use graphical interface targeted at reducing product development time.

1.2 PurePath Console Software Installation

PurePath Console Software is acquired by requesting and registering on the TI website. A request can be submitted from the following webpage: http://www.ti.com/tool/PurePathConsole or you can go to www.TI.com and search for PurePathConsole.

Once you have been registered – you can download the latest version of PurePath Console on the extranet web link provided to you via an email reply indicating the request has been approved.

To load the PurePath Console Software suite:

From Download – Save the installation file to a temporary directory. Go to the temporary directory and run setup_PurePathConsole_Main_vx.xx.exe. Follow the InstallShield Wizard instructions to compete the installation. When the installation is complete PurePath Console can be started via the start menu.

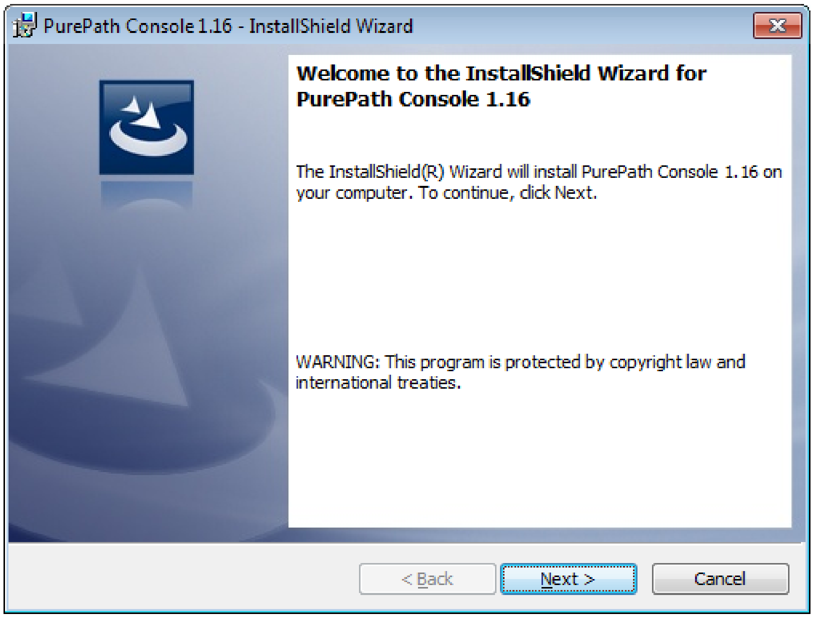


Figure 1-1. PurePath Console Install Shield

1.3 Starting PurePath Console

The "Choose Target" icon 

starts PurePath Console and opens the following windows offering options to support target devices.

It is important to run this application as an administrator

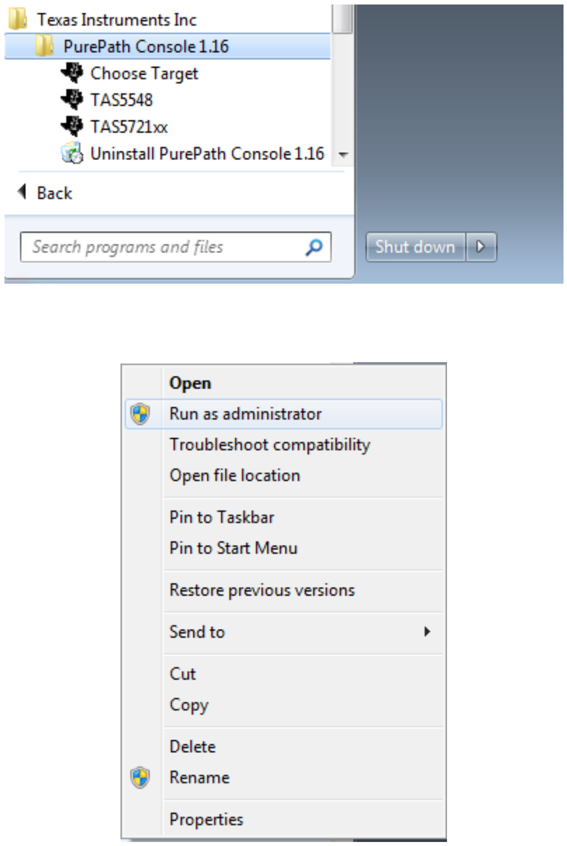


Figure 1-2. PurePath Console Startup

When PurePath Console is operational then - Choose the desired target or add optional targets.

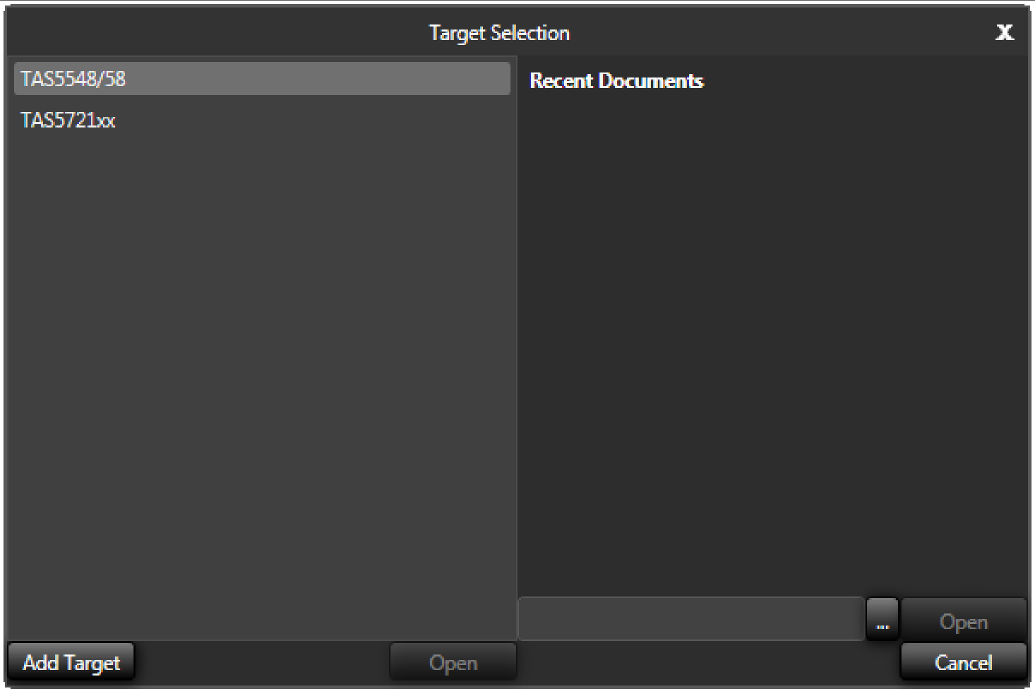


Figure 1-3. Open PurePath Console

Target Selection

A list of available target devices is displayed in the Target Selection window. A target device can be chosen by clicking on the device name and then "Open" or double clicking the device name. To add a new target device click on the "Add Target" button and chose the device plug-in via the browser.

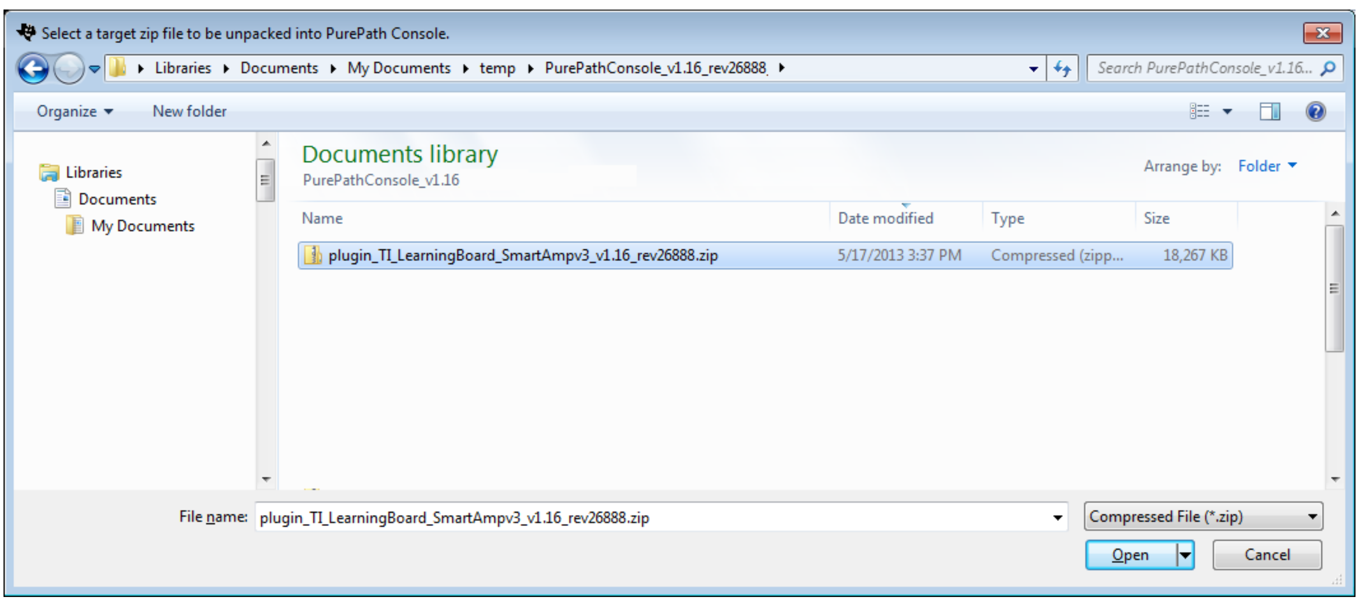


Figure 1-4. Loading Target Option

1.4 PurePath Console Operation

1.4.1 Main Application

The main application displays a modern looking toolbar and ribbon bar interface. The following is an image of the TAS5721 application GUI. The main menu is accessed by clicking on the Start Button (TI Logo)

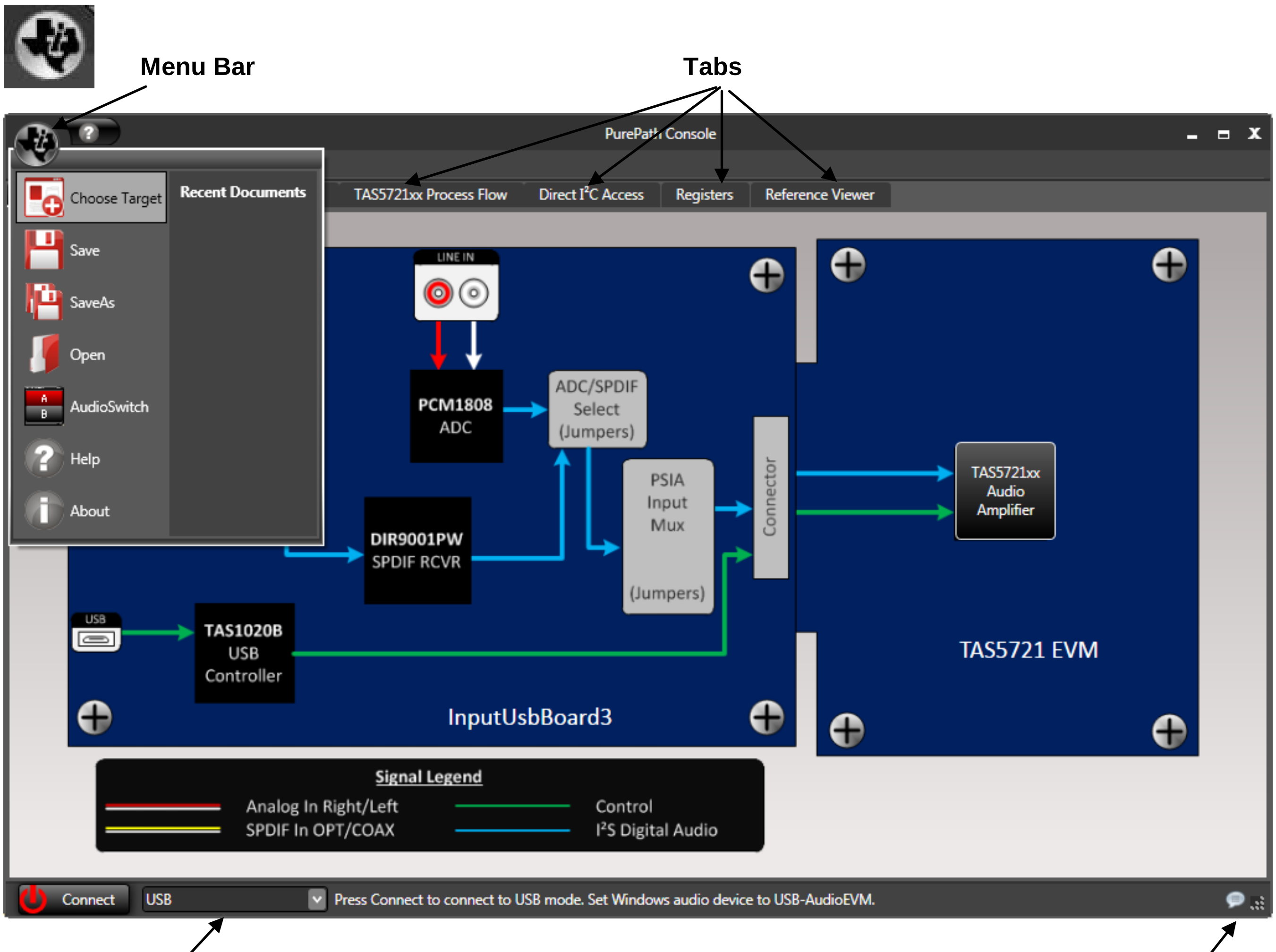


Figure 1-5. Locating the Main Menu

Start Button

The Start Button shows the main application drop-down menu ribbon and the Show/Hide Tabs display for managing open tabs. Selecting the Start Button opens a drop down men for choosing a target, displaying this help or the about box, A new target device can also be chosen from this drop-down.

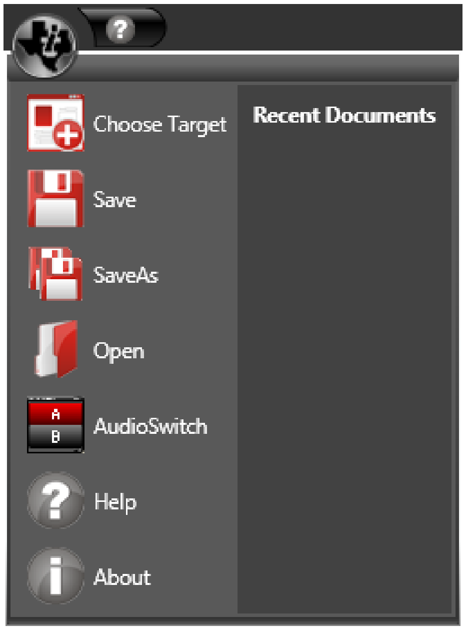


Figure 1-6. Main Menu

Choose Target

The **Choose Target** menu item displays the Target Selection Dialog for choosing a target. Targets that are marked for beta release only will display as such in the dialog. A target may be chosen by selecting the target then clicking 'Open', or double-clicking the target.

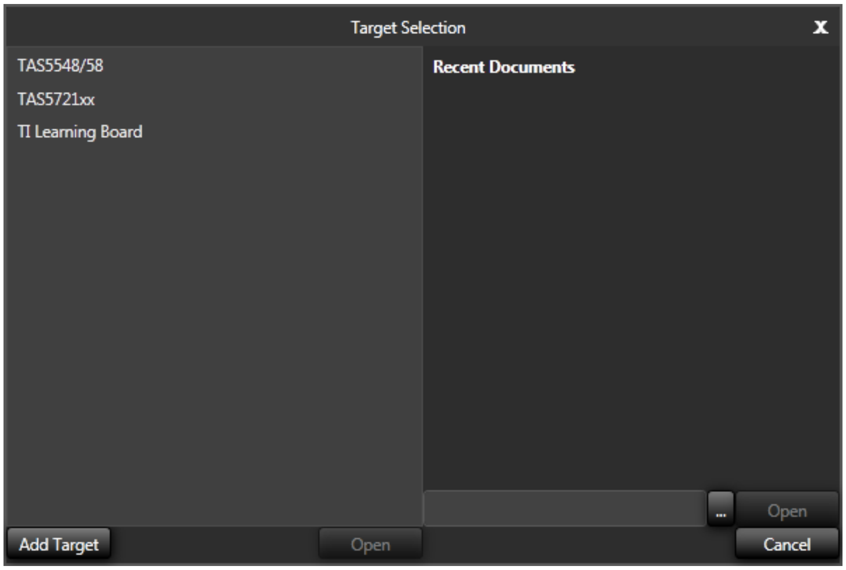


Figure 1-7. Target Selection Dialog Window

If the desired target is not shown select "Add Target" and go to the installation temporary directory.

Save and Save As

The **Save and Save As** menu items will save all of the settings that have been entered into any of the PurePath Console tabs. The saved information can be loaded by the GUI to the same target for which the save was performed. In this case, the image that is saved is for the learning board. Later we will describe how the measured and tuning data can be saved and opened at a later time by any of the Smart Amp GUIs.

AudioSwitch

The AudioSwitch menu item displays the AudioSwitch dialog. The AudioSwitch dialog is used to switch between two soundcards. The soundcards can be configured by click the settings icon .

Figure 1-8. Audio Device Display

Help

The **Help** option will open the help file in the Reference Viewer tab.

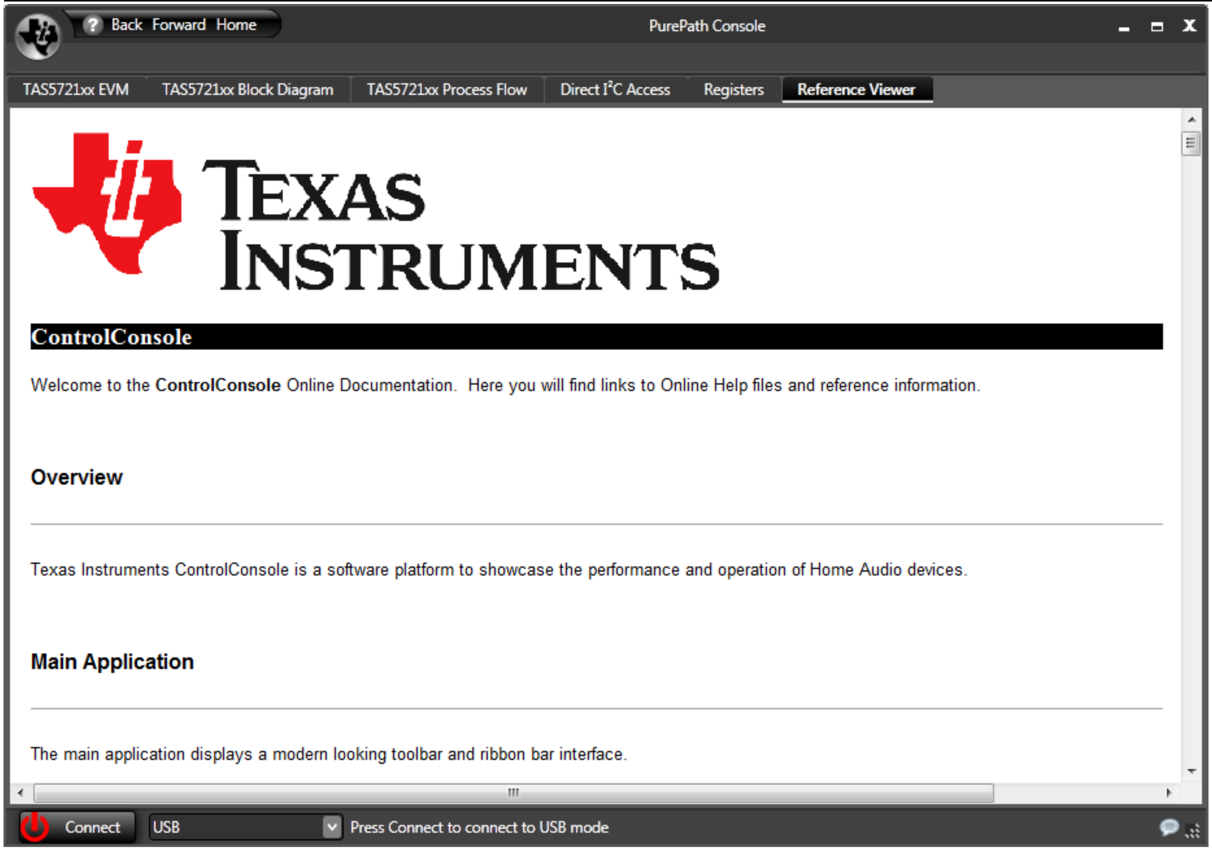


Figure 1-9. Help Display

About

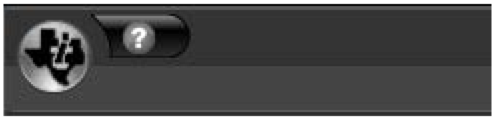
This will display the PurePath Console version and revision number.

Figure 1-10. Main Menu / About

1.4.2 Menu Bar

The menu bar for the application is located in the upper left corner and differs for each tab and may be slightly different for each target device.   Three of the most common menu displays are shown below.

Every tab displays the help icon: ?



The Registers tab displays a **Refresh** menu item.

The Reference Viewer displays **Back**, **Forward** and **Home** menu items.

1.4.3 Status Bar

The **Status Bar** is located at the bottom. It displays the connection status, the operating mode (USB) and notification status.

The connection status is indicated by the LED in the left corner. It can be red, yellow or green indicating the status of connection. Table 1-2 describes its operation. The user can define the mode of operation via the drop down mode box. Currently USB is the only operating mode supported.

PurePath Console will attempt to connect to a target via the USB port when the "Connect" button is clicked.

The notification icon in the lower right corner shows when new information is available in the Output Window. Clicking on the notification icon displays the Output window. Examples are shown in Table 1-2.

Table 1-2. Status Bar

Status Item	Description
Linked LED	 indicates no target is connected. Connecting a target board and clicking “Connect” will automatically cause PurePath Console to attempt to connect.  indicates that a target is connected, but not at the expected I^2C slave address. Check the I^2C slave address in the Direct I^2C tab.  indicates that a target is successfully connected and communicating.
Notification Icon	 indicates that no new messages are available. Clicking on the icon opens the Output Window. indicates that a new information is available in the Output Window. indicates that a new error message is available in the Output Window.

1.4.4 PurePath Console Tabs

EVM Tab

The EVM tab displays the EVM board, showing the overall layout and features of the EVM. Clicking on the processor will cause the processor Block Diagram tab to display.

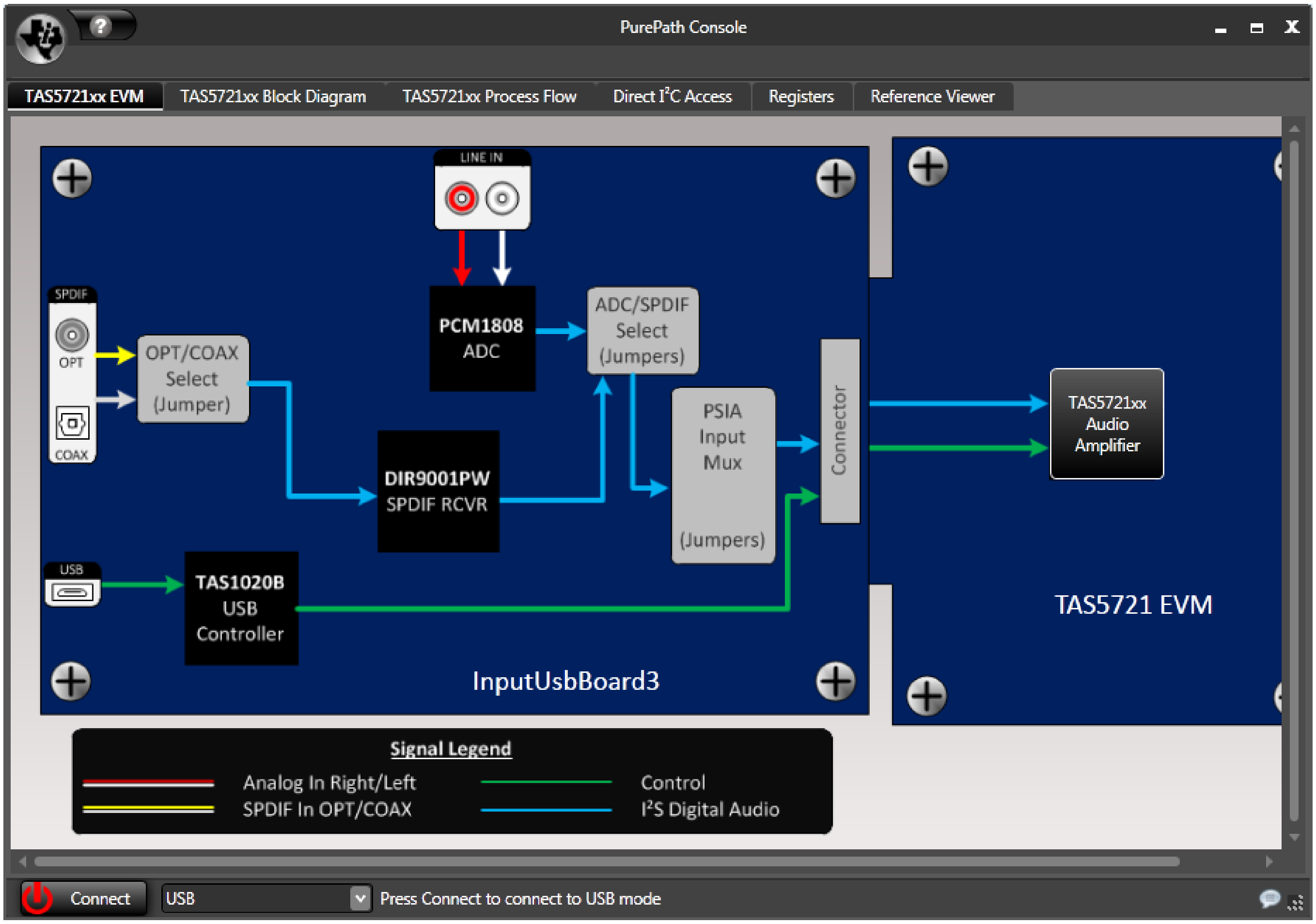


Figure 1-11. EVM Tab

Block Diagram Tab

The Block Diagram Tab displays the block diagram for the chosen processor as shown in Figure 1-12. Clicking on the DAP block will cause the Process Flow Tab to display. Some block diagrams such as the TAS5721, have selectable areas for configuring the input and output muxes. Clicking on those will display pop-up configuration as shown in Figure 1-12.

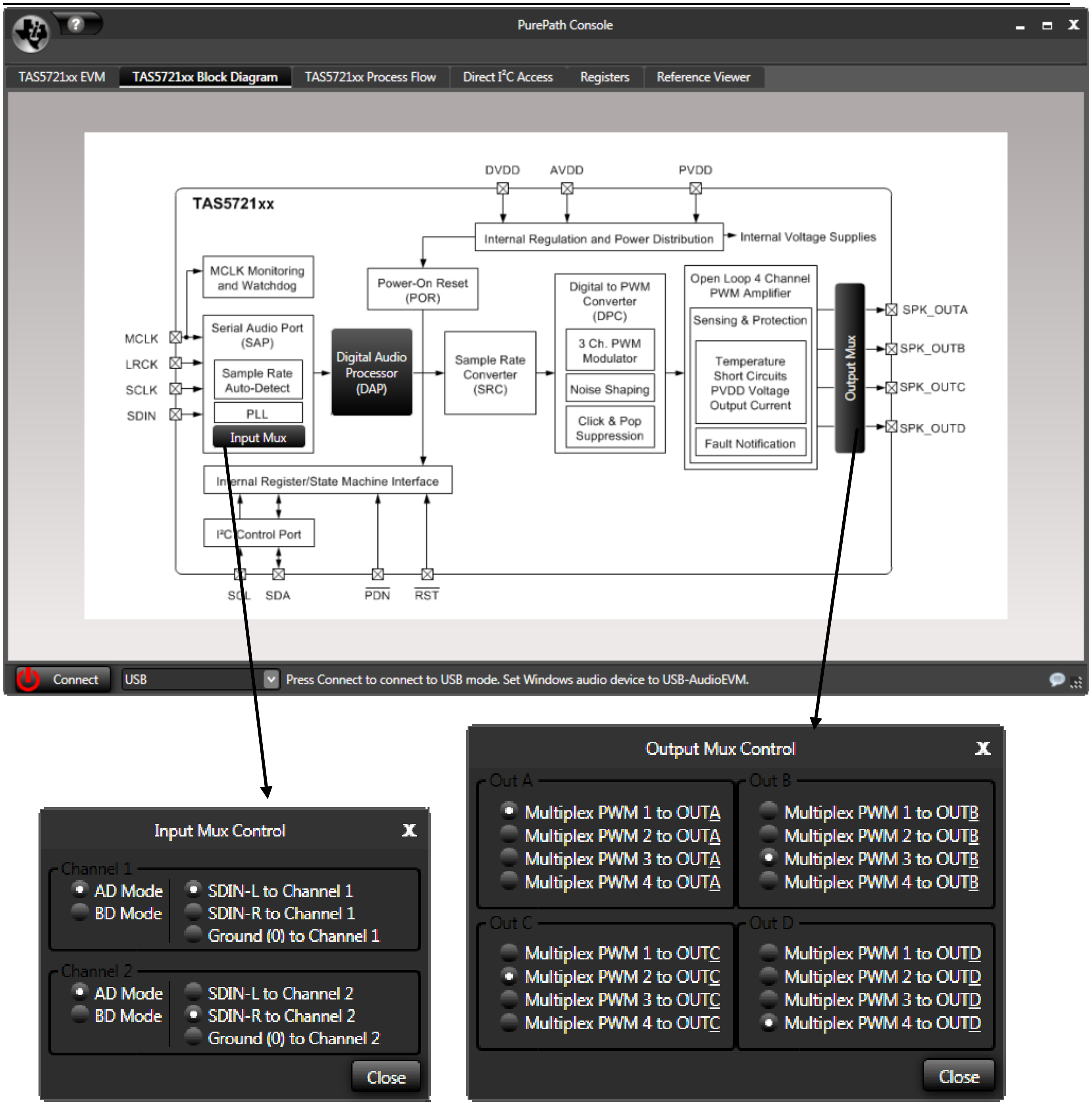


Figure 1-12. *Block Diagram Tab and Input/Output Windows*

Process Flow Tab

The Process Flow tab will display the process flow diagram for the chosen processor. Controls on the diagram may be directly manipulated to control the process flow.

The active audio path, based on mux and mixer settings is highlighted in a darker color.

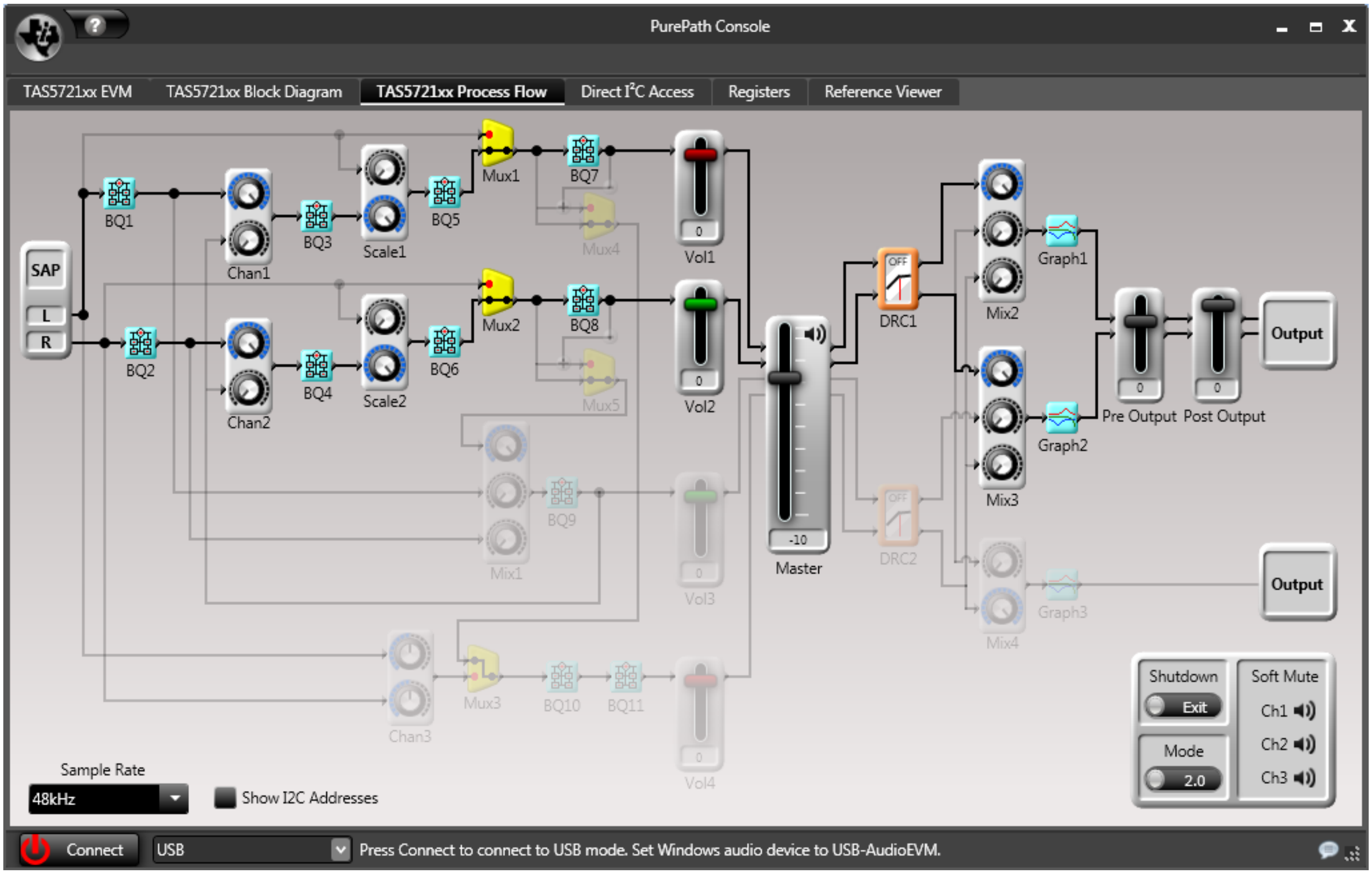


Figure 1-13. Process Flow Tab

The sliders may be adjusted by either:

- Clicking on them and dragging the mouse up or down
- Hovering the mouse over the knob or slider and spinning the mouse wheel
- Hovering the mouse over the knob or slider and pressing the up or down arrow
- Enter the desired value directly in the edit box below the slider being adjusted

The multiplexer (mux) controls may be switched by simply clicking on them.

Biquad, DRC, and AGL components have popup GUIs that are available by right-clicking on the control and selecting "Activate...".

Direct I2C Access Tab

The Direct I^2C Access tab provides direct access to I^2C registers and script execution support. This tab supports the following capabilities:

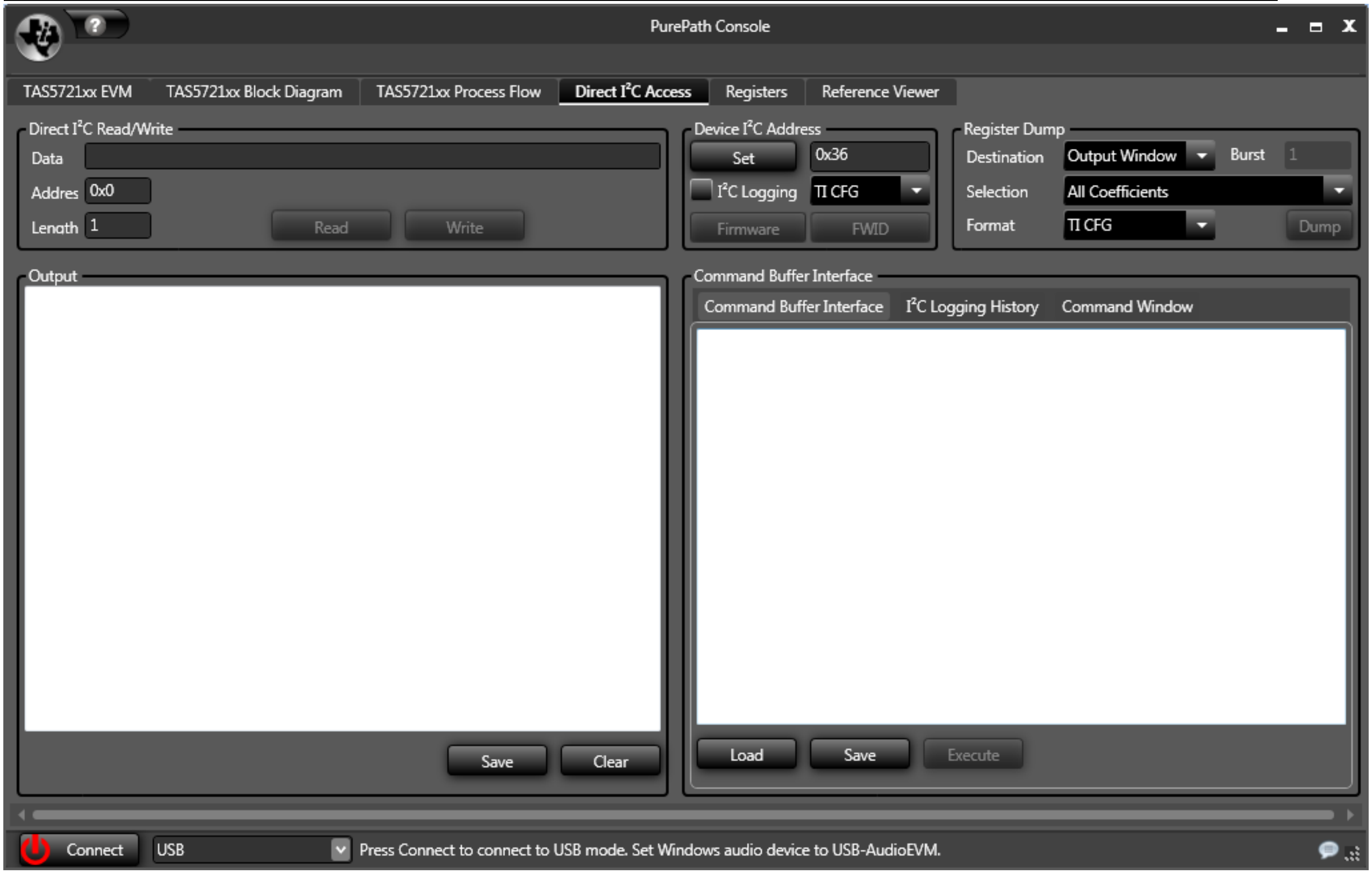


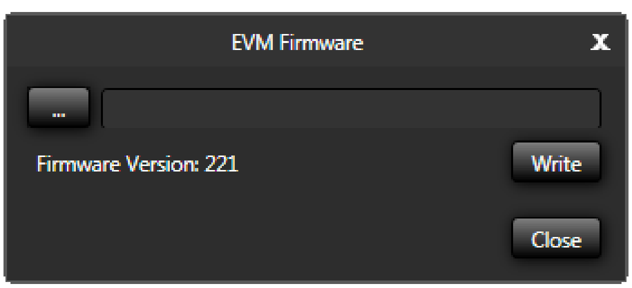
Figure 1-14. Direct I²C Tab

Direct I²C Read/Write

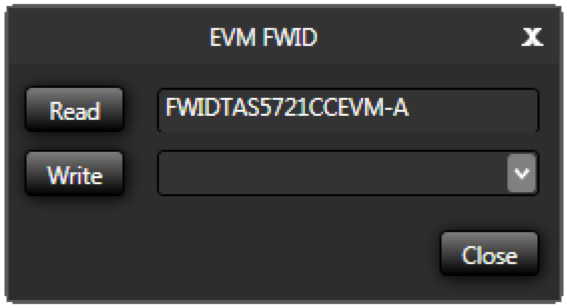
- Direct read and write of a I²C register
- The Length field is initialized based on the known length for the specified I²C register, but can be modified

Device I²C Address

- The I²C slave address for the device. Set to the default for the device EVM, but may be changed.
- The Firmware button will be active after successfully connecting to the target EVM. The Firmware button will activate the Firmware GUI where the firmware version can be read from the target EVM and firmware can be written to the target EVM.

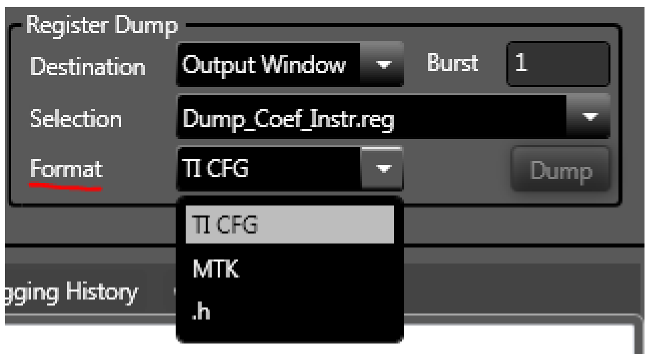


- The FWID button in Figure 1-14 will be active after successfully connecting to the target EVM. The FWID button will activate the EVM FWID GUI where the FWID can be read from the target EVM. FWID can also be written to the target EVM.



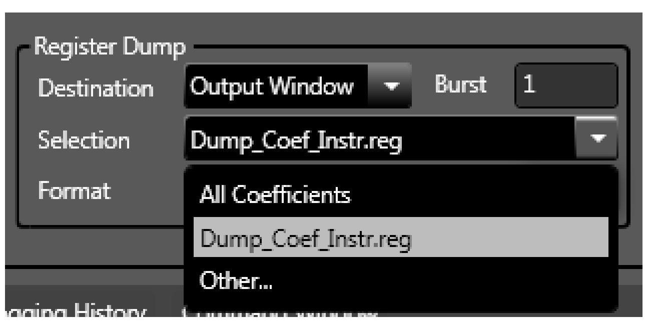
Register Dump

- **Memory Dump** outputs all the I^{2}C register values to the Output window or to a file. Dump Button is activated after successfully connecting () to the target EVM.



The **Destination** choice determines if the output goes to the output window or a file. If a file is chosen, a file chooser dialog is displayed when [Dump] is pressed. The default extension is .cfg, .h or .mtk based on the **Format** choice.

The **Burst Siz**e defines the maximum size of burst output that is to be written by the register dump. The Burst Size textbox is only visible when TI CFG or .h format is chosen. MTK format does not support burst writes. Burst Size is an integer and must be in the range 1 to 255.



The **Selection** box determines if all I^{2}C registers on the processor are dumped, or only certain registers are dumped. The output can be formated by using .reg file that is stored in the target TargetLibrary of the device. See the following sections for description and formating of the .reg files.

The **Format** choice determines if the output format is .cfg, .h or MTK. The format of the .cfg output (miniDSP or DAP) is determined by the device under test.

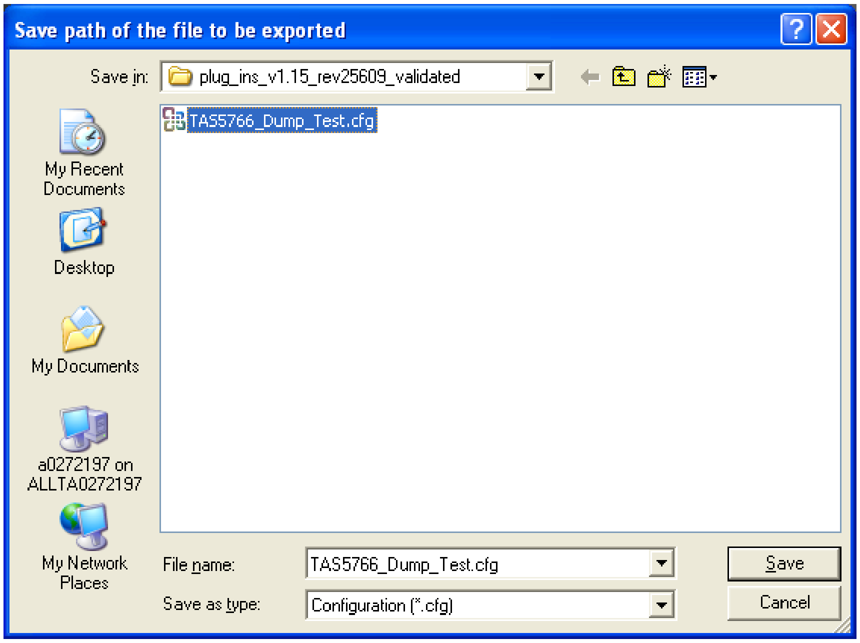


Figure 1-15. Dump File Window

When dumping registers, the following additional information is used:

In miniDSP devices (PCM5151,TAS5766, :

- Dumping all registers includes dumping all registers on all pages. All registers are all considered to be 1 byte in length

In DAP devices (TAS5729, TAS5731 :

- Dumping all registers dumps all the registers listed in the .addr file, with the sizes listed in the .addr file. Dumping selected registers with a .reg file dumps the selected registers using register sizes listed in the .addr file for the registers. Information on .addr file format is contained in Appendix A.

.cfg Dump format in outputting coefficient values for miniDSP devices

The .cfg dump format creates an output that can be later loaded to set the coefficient memory locations.

Format: **W**rite   **98** Device address   **01** Register address   **xx** Register value

Set page address to page 0
w 98 00 00

#Read register values
w 98 01 00
w 98 02 10
w 98 03 00
w 98 04 01
w 98 05 01
w 98 06 00
w 98 07 00
w 98 08 24
w 98 09 00

```
w 98 0A 00
w 98 0B 01
w 98 0C 7C
w 98 0D 00
```

.cfg Dump format in outputting coefficient values for DAP devices

The .cfg dump format creates an output that can be later loaded to set the coefficient memory locations.

```
X00 00
X01 C1
X02 10
X03 A0
X04 05
X05 00
X06 00
X07 01 10
X08 00 C0
X09 00 C0
X0A 00 C0
...
...
X1F 00
X20 00 01 77 72
X21 00 00 43 03
...
...
X26 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X27 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X28 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
...
```

.MTK Dump format n outputting coefficient values

The MTK dump format is a format that can be read by Matlab programs . I

The following is an example:

```
# Book 0, Page 0x00
aud.io.adac.aw 98 00 0x00
aud.io.adac.aw 98 01 0x00
aud.io.adac.aw 98 02 0x10
aud.io.adac.aw 98 03 0x00
aud.io.adac.aw 98 04 0x01
aud.io.adac.aw 98 05 0x01
aud.io.adac.aw 98 06 0x00
aud.io.adac.aw 98 07 0x00
aud.io.adac.aw 98 08 0x24
aud.io.adac.aw 98 09 0x00
aud.io.adac.aw 98 0A 0x00
aud.io.adac.aw 98 0B 0x01
aud.io.adac.aw 98 0C 0x7C
aud.io.adac.aw 98 0D 0x00
aud.io.adac.aw 98 0E 0x00
aud.io.adac.aw 98 0F 0x00
aud.io.adac.aw 98 10 0x00
aud.io.adac.aw 98 11 0x00
aud.io.adac.aw 98 12 0x00

...
```

...

.h Dump Format in outputting coefficient values

The .h format can be used to create system controller device configuration files.

Additional information describing the .h dump format is contained in Appendix B.

The following is an example:

```
typedef unsigned char cfg_u8;
typedef union {
    struct {
        cfg_u8 offset;
        cfg_u8 value;
    };
    struct {
        cfg_u8 command;
        cfg_u8 param;
    };
} cfg_reg;
#define CFG_META_SWITCH (255)
#define CFG_META_DELAY  (254)
#define CFG_META_BURST  (253)
cfg_reg registers[] = {
// Book 0, Page 0x00
    { 0x00, 0x00 },
    { 0x01, 0x00 },
    { 0x02, 0x10 },
    { 0x03, 0x00 },
    { 0x04, 0x01 },
    { 0x05, 0x01 },
    { 0x06, 0x00 },
    { 0x07, 0x00 },
    { 0x08, 0x24 },
    { 0x09, 0x00 },
    { 0x0A, 0x00 },
    { 0x0B, 0x01 },
    { 0x0C, 0x7C },
    { 0x0D, 0x00 },
    { 0x0E, 0x00 },
    { 0x0F, 0x00 },
    { 0x10, 0x00 },
    { 0x11, 0x00 },
    { 0x12, 0x00 },

...
...
```

Formatted dump output

The output format of the Dump can be specified by a file Dump_Coef_Instr.reg (in this example). This is stored in the TargetLibrary\ directory of the PurePath Console. This capability can be used to output the current contents of the registers, in a specified order, interspersed with comments, or to have specific register value placed at specific locations in the output sequence. The specific register values could for example invoke a mute or unmute of the audio.

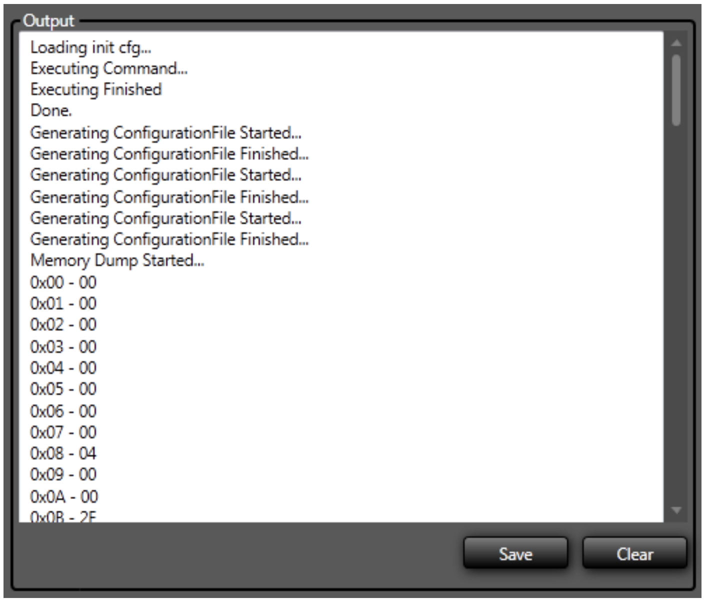
Additional detail of the .reg file format and features is provided in the Appendix C.

The following is an example:

```
# Dump of SmartAmp v3 process flow
# reg[1][3] = 0x04
w 98 00 01
w 98 03 04
# reg[0][0x3] = 0x11
w 98 00 00
w 98 03 11
# Coefficient Memory A
# page 44 (0x2C)
w 98 00 2C
w 98 08 14
w 98 09 3D
w 98 0A 13
w 98 0B 00
w 98 0C FF
w 98 0D FF
w 98 0E FF
w 98 0F 00
w 98 10 80
w 98 11 00
w 98 12 00
…
…
w 98 7C 00
w 98 7D 00
w 98 7E 00
w 98 7F 00
# Post-Initialization  Page zero is selected
w 98 00 00
#                              # Go to operational mode
w 98 02 00
#                              # unmute left and right channels
w 98 03 00
```

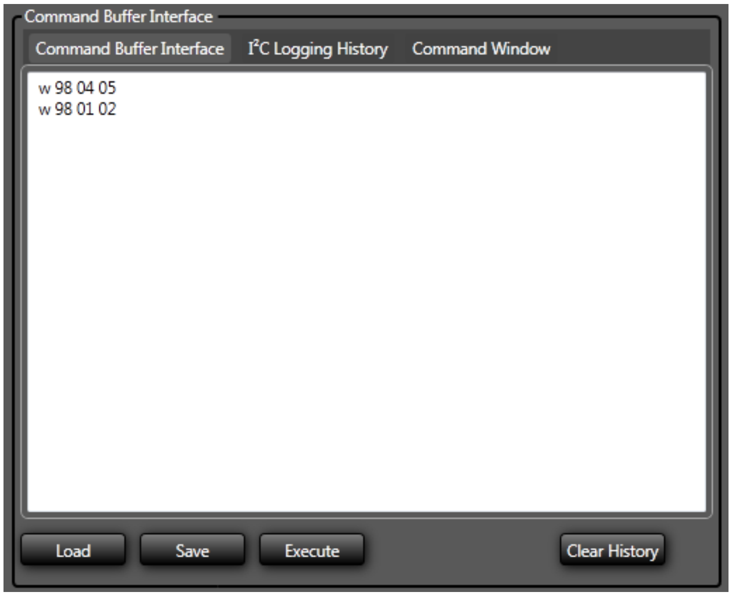
Output window

The Output window of the Direct I²C tab displays the output from .cfg file execution and memory dump. The output may be saved to a file with the Save button. The output window may be cleared with the Clear button.



Command Buffer Interface

The Command Buffer Interface is an editable area for typing and executing .cfg file commands on the target.



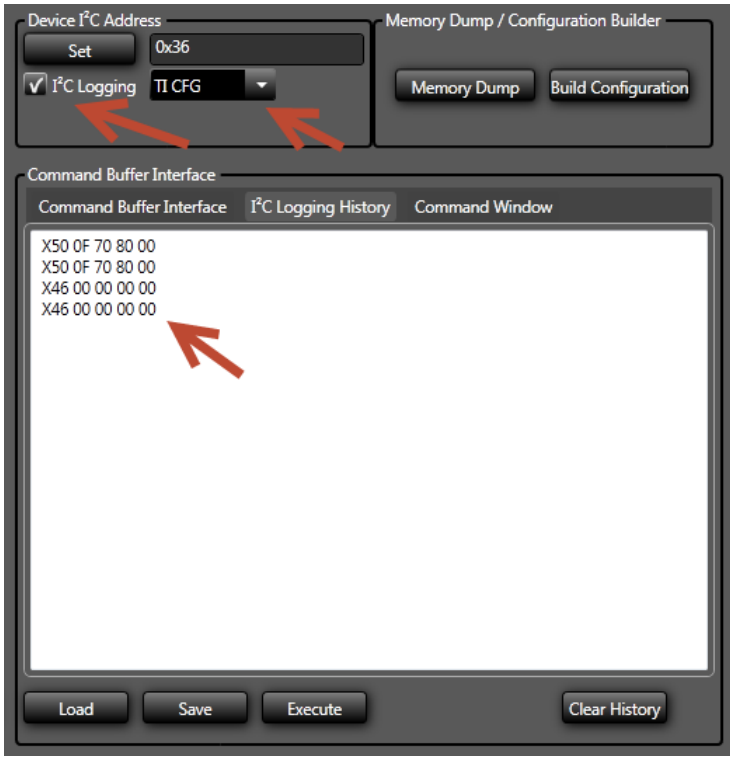
The user may type command directly into the window, and execute the contents of the window with the **Execute** button. The results of the .cfg execution is displayed in the Output window.

The user may load external .cfg files into the window with the **Load** button or save the current contents to a file with the **Save** button.

I²C Logging History

The I²C Logging History window displays the output of I²C logging, if it is enabled in the Device I²C Address section described above.

The I²C logging format displayed in the window is determined by the device mode (DAP or miniDSP) and the I²C logging format (CFG or MTK).



The Clear History button clears the window.

Command Window

Command Window is used for running IronPython scripts. Additional information is on this window is contained in the online help which is accessed by pressing F1.

I2C Register Tab

The Register Tab provides direct access to the processor registers. The values may be edited as decimal, hex, or each bit by simply clicking the bit display.

The color coding is as follows:

- Dark Grey - reserved, not able to be edited
- Light Grey / Blue - alternating to show where fields are in each register.

A pop-up describes each register field.

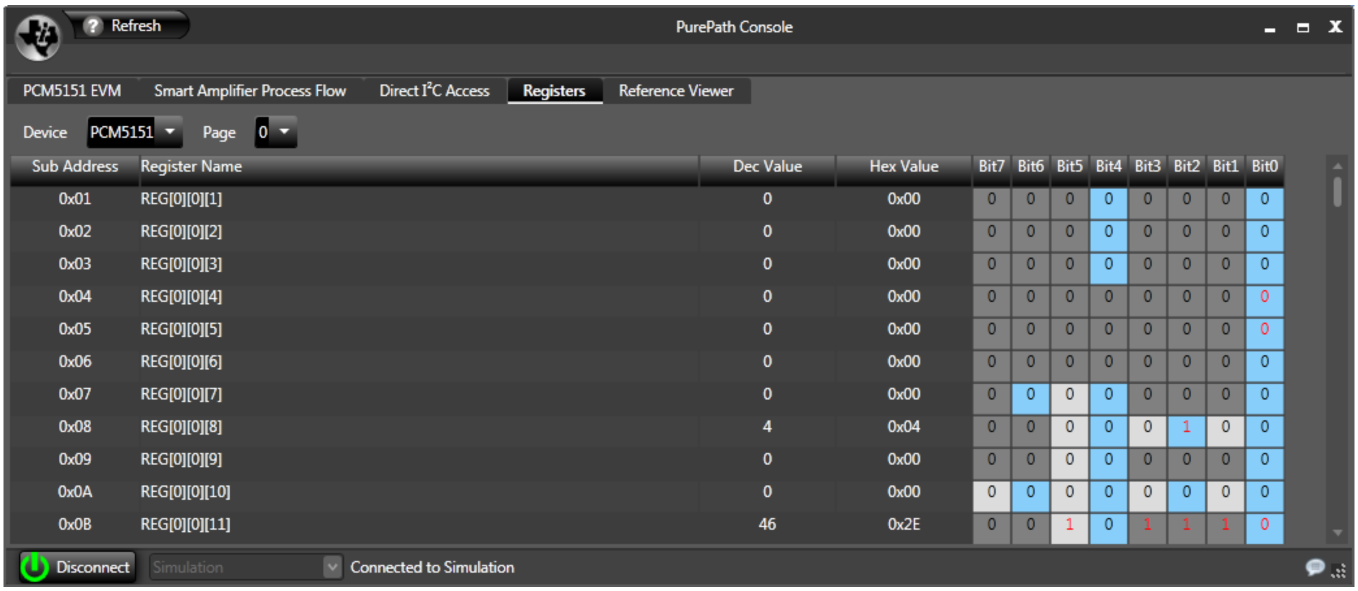


Figure 1-16. Register Tab

Reference Viewer Tab

The Reference Viewer is simply an HTML browser which could load target specific material from either www.ti.com or locally from the device support directory. If internet access is not available, the online help will be displayed.

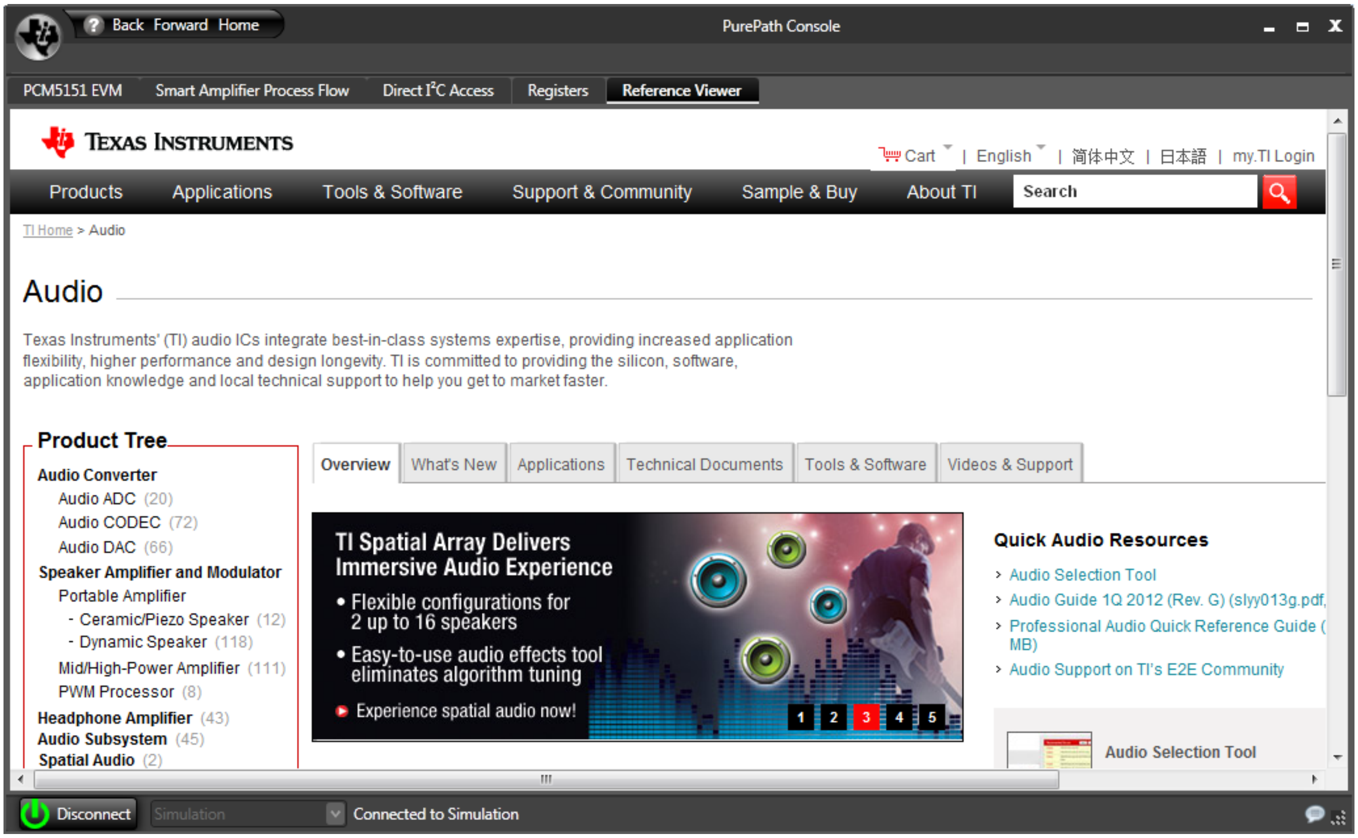


Figure 1-17. Reference Tab

The menu items for this tab are:

- **Back** – Navigate back to the referrer of a link - similar to any browser Back button
- **Forward** – Navigate forward after using Back - similar to any browser Forward button
- **Home** – Navigate back to the home page

Appendix A

.addr File Format

The .addr file format has been supported in multiple generations of TI DAP based device tools since 2003. The nature of the I^2C interface on the DAP allows multiple bytes to be read or written from an I^2C register. On the TAS devices, generally 4 to 20 bytes are supported. The .addr file is a configuration file used to inform the tools how many bytes are expected to be read or written from each I^2C register.

The .addr file is useful since it removes the need for the user to remember how many bytes each I2C register contains. This is particularly important in GDE-built process flows because the I^2C mapping is dynamically determined by the GDE. For this reason, the GDE is able to generate a .addr file from the I^2C Interface Overview dialog.

The .addr file has lines the following format. Any lines not meeting this format are ignored as comments.

    # N M !comment

where:

- N is the I2C address
- M is the number of bytes to expect at that address
- comment is an optional comment describing the line

An example .addr file for the TAS5731 follows:

If not .addr file is given, or for addresses not defined in a .addr file, the default size in the Direct I2C tab is 4 bytes.

Appendix B

Register Dump .h File Format

When .h file format is chosen, the .reg file given is used to define the order for output, just as in .cfg format. The output of .h format is defined below.

First, the header will always contain these definitions:

```
typedef unsigned char cfg_u8;
typedef union {
    struct {
        cfg_u8 offset;
        cfg_u8 value;
    };
    struct {
        cfg_u8 command;
        cfg_u8 param;
    };
} cfg_reg;

#define CFG_META_SWITCH (255)
#define CFG_META_DELAY  (254)
#define CFG_META_BURST  (253)
```

Then, the header will contain the following definition for registers[]

```
cfg_reg registers[] = {

… registers here

};
```

The definitions of individual register writes are tuples of {register, value}.

Burst writes are a header with a meta command and lenght {CFG_META_BURST, length}, followed by tuples with 2 values in each.   Odd number of bytes are padded with an extra byte.

The following full example shows the output:

```
cfg_reg registers[] = {
    { 0x00, 0x00 },
    { 0x7f, 0x00 },
    { 0x06, 0x21 },
    { 0x07, 0x05 },
    { 0x08, 0x04 },
    { 0x09, 0xb0 },
    { 0x0a, 0x01 },
    { 0x0b, 0x02 },
    { 0x0c, 0x08 },
    { 0x0d, 0x00 },
    { 0x0e, 0x80 },
    { 0x12, 0x02 },
    { 0x13, 0x08 },
    { 0x14, 0x80 },
    { 0x00, 0x00 },
    { 0x7f, 0x50 },
    { 0x00, 0x01 },
    { CFG_META_BURST, 9 },
```

```
        { 0x1c, 0x40 },
        { 0x00, 0x00 },
        { 0x00, 0x7f },
        { 0xff, 0xff },
        { 0x00, 0x00 },
};
```

PurePath™ Console is able to generate its output, a compiled process flow, in the form of a 2-dimensional C array suitable for inclusion into a program running on the application processor or microcontroller. Such arrays specify a list of individual 8-bit register writes to the amplifier. Additionally, the format allows one to specify delay (meaning, the application needs to pause for a specified time at that point) and also includes a mechanism to cause the application switch between such arrays.
The purpose of this Appendix to is to specify this data structure, extend it in a backwards-compatible manner for burst writes (i.e., register writes with a payload of greater than one byte) and provide some example application to code to illustrate its use.

Data Structure Development

The data structure used to represent register writes consists of a series of 8-bit 2-tuples.

```
unsigned char registers[][2] = {
    { 0x00, 0x6c }, // Write 0x6c at offset 0
    { 0x01, 0x03 }, // Write 0x03 at offset 1
    { 0x02, 0x00 }, // Write 0x00 at offset 2
    { 0x03, 0xa0 }, // Write 0xa0 at offset 3
    { 0x04, 0x53 }, // Write 0x53 at offset 4
    { 0x05, 0xe0 } // Write 0xe0 at offset 5
};
```

Each ordered pair represents a single write as indicated in the comments above.
To make our intentions a little clearer we use a C data structure to define the 2-tuple.

```
typedef unsigned char cfg_u8;
typedef struct {
    cfg_u8 offset;
    cfg_u8 value;
} cfg_reg;
cfg_reg registers[] = {
    { 0x00, 0x6c }, // Write 0x6c at offset 0
    { 0x01, 0x03 }, // Write 0x03 at offset 1
    ...
};
```

To support commands other than single 8-bit writes, we reserve offset values greater than 127. This is not as great an inconvenience as it may seem because most devices do not use registers with offset higher than 127. (For a few that do, there is an alternate method to write to represent offsets that are 128 or higher.)
We'll call such reserved offsets "meta commands" – every 2-tuple in the array is a write command by default, except when the offset is greater than 127 in which case it is a meta command. The byte following the meta command serves as a parameter to it. PurePath GDE supports two different meta commands. We shall define one more and leave the other 124 for future use.

1. The **delay meta command** uses an offset of 254 and serves as an indication to the host processor to wait for a certain number of milliseconds, as specified by its parameter, before proceeding to the next command. It is typically used after turning on the PLL or after software reset of the device.

```
cfg_reg registers[] = {
    ...
    { 254, 100 }, // Wait for hundred milliseconds at this point
    { 0x01, 0x03 }, // Write 0x03 at offset 1
    ...
};
```

2. The **switch meta command** uses an offset of 255 and tells the host processor to stop processing this array and switch to another array of commands, as indicated by its parameter. After finishing that array of commands, the host processor switches back to the original array of commands at the point where it left off. This mechanism is used to decouple system configuration from miniDSP instructions/coefficients and

serves to minimize the size of stored arrays by allowing re-use of common sub-sequences.

```
cfg_reg registers[] = {
    ...
    { 255, 1 }, // Switch to array#1
    { 0x01, 0x03 }, // Restart here after finishing array#1
   0020...
};
```

3. The **burst write meta command** shall indicate that what follows is a burst write. The parameter specifies the burst length, including the register offset. Unlike the case for other commands, what follows this command is not another command, but the burst payload itself. After executing the burst write, the host processor must calculate the index of the next valid command by incrementing the current index by one two plus the length of the payload, where the payload length is an even number, or the next higher even number where the payload length is odd.

```
cfg_reg registers[] = {
    ...
    { 253, 5 }, // A burst payload of length 5 follows
    { 0xc0, 0x01 }, // Burst is written to register at offset 0xc0, and
    { 0x02, 0x03 }, // the data being written is {1, 2, 3 4}.
    // Notice that we use an offset greater than 127 with no
    // possibility of it being confused for a meta command.
    { 0x04, 0x00 }, // Notice that the last byte is wasted since we require the next
    // command to start at an even position
    { 0x01, 0x03 }, // Next command position
    ...
};
```

Addressing Offsets Greater than 127

Older TAS family devices use register offsets greater than 127. Normally, the application processor is necessarily required to write a burst of several bytes at such offsets. This may be accomplished without any problems: Offsets built into burst write payloads are not constrained in any way as the example that introduced burst writes shows.
However, in the cases that a single byte write is required at an offset greater than 127, we encode it as a burst of length 2.

```
cfg_reg registers[] = {
    ...
    { 253, 2 }, // A burst payload of length 2 follows
    { 0xc0, 0x01}, // Write 0x01 at offset 0xc0
```

Microcontroller Code

To illustrate how such an array may be used in application code the following example is provided. We shall assume the existence of an external library that provides us with an API to do I2C writes, to introduce a delay and to obtain a pointer to other register command arrays.

```
// Externally implemented function that can write n-bytes to the device
extern int i2c_write(unsigned char *data, int n);
// Externally implemented function that delays execution by n milliseconds
extern int delay(int n);
// Externally implemented function to obtain other register command arrays
extern void *switch_array(int array_id, int *array_size);
```

The implementation of these functions is not important, nor their exact prototypes. The application code below may easily be adapted for such changes.

```
void transmit_registers(cfg_reg *r, int n)
{
    int i = 0, sz
    cfg_reg *sw_r;
    while (i < n) {
        switch (r[i].command) {
```

```
        case CFG_META_SWITCH:
            sw_r = switch_arra(r[i].param, &sz);
            transmit_registers(sw_r, sz);
            break;
        case CFG_META_DELAY:
            delay(r[i].param);
            break;
        case CFG_META_BURST:
            i2c_write((unsigned char *)&r[i+1], r[i].param);
            i += (r[i].param + 1)/2;
            break;
        default:
            i2c_write((unsigned char *)&r[i], 2);
            break;
        }
        i++;
    }
}
```

Appendix C

Register dump .reg definition files.

A .reg file can be used in the Register Dump section to customize the register dump format process, to do any of the following:

- Include only a portion of the registers in the memory dump
- Order the memory dump in a certain order
- Insert comments or specific register writes (e.g. for mute and unmute) into the resulting memory dump

There are 3 types of commands in the .reg file:

1. Register Dump Commands
   Register Dump commands dump the register contents at the specified page and register. They have the following two formats supported:
   {page},{register} = `xx`
   {book},{page},{register} = `xx`

   The {book}, {page} and {register} may be represented in decimal or hex.
   The {register} may be a single register value or a range of registers.

   Example: Dump page 44, register 10 from memory.
   `44, 0xA = xx`

   Example: Dump page 44, registers 0 through 10 from memory.
   `44, 0x00-0xA = xx`

   Example: Dump book 120, page 44, register 10 from memory.
   `120, 44, 0xA = xx`

   Example: Dump book 120, page 44, registers 0 through 10 from memory.
   `120, 44,0x00-0xA = xx`

2. Register Write Commands
   Register Write commands look just like Register Dump commands, but specify a value after the '=' instead of specifying 'xx'. They cause a specific register write to be inserted at the given location. Memory is not read for this line.

   Example:
   ```
   # Change page to 44
   0x00 = 44
   ```

3. All other lines are comments and are echoed directly to the output.

An example .reg file is given below.

```
# First, write 0xFE to register 0x7 of page 0
0x00 = 0x00
0x07 = 0xFE
# Change page to 44
0x00 = 44
```

```
# Then, dump registers 0x00 to 0x0B
0x2C,0x00 = xx
44,0x01 = xx
44,0x02 = xx
44,0x03-0x0B = xx
# Lastly, write 0xF5 to register 0x7
44,0x07 = 0xF5
```

The result of dumping memory with this .reg file on a miniDSP and outputting .cfg format is as follows:

```
# First, write 0xFE to register 0x7 of page 0
w 00 00
w 07 FE
# Change page to 44
w 00 2C
# Then, dump registers 0x00 to 0x0B
w 00 2C
w 01 04
w 02 00
w 03 00
w 04 00
w 05 00
w 06 00
w 07 00
w 08 00
w 09 00
w 0A 00
w 0B 00
# Lastly, write 0xF5 to register 0x7
w 07 F5
```