Texas Instruments

DSP/BIOS Audio Driver

User's Manual

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 and 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

Read This First

About This Manual

This User's Manual serves as a software programmer's handbook for working with the **DSP/BIOS Audio driver.** This manual provides necessary information regarding how to effectively install, build and use **DSP/BIOS Audio driver** in user systems and applications.

This manual provides details regarding how the **DSP/BIOS Audio driver** is architected, its composition, its functionality, the requirements it places on the hardware and software environment where it can be deployed, how to customize/ configure it to specific requirements, how to leverage the supported run-time interfaces in user's own application etc.,

This manual also provides supplementary information regarding steps to be followed for proper installation/ un-installation of the DSP/BIOS Audio device driver.

Terms and Abbreviations

Term/Abbreviation	Description
API	Application Programmers Interface
IP	Intellectual Property
EDMA	Enhanced Direct Memory Access Controller
IOM	I/O Mini Driver Model
McBSP	Multi-channel Buffered Serial Port
McASP	Multi-channel Audio Serial Port
DM6437/C6424	TI's digital media processor based on C64+ core

Related Documentation

Internal

- None

External

- None

Trademarks

Revision History

Date	Author	Revision History	Version
25th December 2006	Pratik Joshi	First draft	1.0
03rd January 2007	Pratik Joshi	Closed document review comments for BFT 2 release 0.4.0	1.1
16th January 2007	Pratik Joshi	Closed document review comments	1.2
2nd February 2007	Pratik Joshi	Release 0.5.0	1.3
22nd March, 2007	Pratik Joshi	Updated for release 0.6.0	1.4
23rd April, 2007	Pratik Joshi	Updated for release 0.7.0	1.5
7th May, 2007	Pratik Joshi	Detailed explanation of control commands added	1.6
22 June 2007	Anuj Aggarwal	Updated for GA patch release 1.00.01	1.7
2nd July, 2007	Pratik Joshi	Updated for GA release 1.00.03.00	1.8

Contents

Tables

Figures

Chapter 1

DSP/BIOS Audio Driver Introduction

This chapter introduces the **DSP/BIOS Audio driver** to the user by providing a brief overview of the purpose and construction of the **DSP/BIOS Audio driver** along with hardware and software environment specifics in the context of **DSP/BIOS Audio driver** deployment.

1.1 Overview

This section describes the functional scope of the DSP/BIOS Audio driver and its feature set. The section also details the various deployment environments, hardware and software, that the DSP/BIOS Audio driver is presently supported on. The chapter introduces the system architecture of the DSP/BIOS Audio driver to the user along with the functional decomposition and run-time specifics regarding deployment of DSP/BIOS Audio driver in user's application.

1.1.1 Supported Services and features

The DSP/BIOS Audio driver provides the following functional services and features:

- Multi-instantiable and re-entrant safe driver.
- Designed for (but not limited to) use with codec drivers.
- Features supported by the Audio driver, which can be directly used by the Audio codec driver ares:
    - Supports run-time Start/Stop of the Audio Play and Record operation.
    - Supports Pause-Resume feature for Audio Playback operation when integrated with the audio codec specific driver.
    - Supports Mute ON/OFF feature for Audio Playback operation when integrated with the audio codec specific driver.
    - Supports sampling rate change via IOCTL
    - Supports codec register read/write via IOCTLs
    - Supports selection of external clock source input to the codec

1.1.2 System Requirements

The DSP/BIOS Audio Driver is supported on platforms characterized by the following requirements

Hardware:

Target Board: DM6437/C6424 EVM

Emulation Setup: XDS 510 USB emulator

Cabling: Standard Serial Connector

Software:

Code generation tools: CCS 3.3.38.2
Code generation tool version is 6.0.8

Operating System: DSP/BIOS 5.31.06

Other S/W Components Used:

DSP/BIOS PSP package

Chapter 2

Installation Guide

The DSP/BIOS Audio driver is available as a driver package. User needs it be integrated with the pspdriver package. For installation follow the instructions provided along with the package.

2.1 Component Folder

Upon installing the DSP/BIOS Audio driver the following directory structure is found in the system's directory.

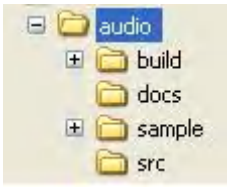

Figure 2-1 Audio driver structure

The folder structure of the pspdriver package is given in the following figure

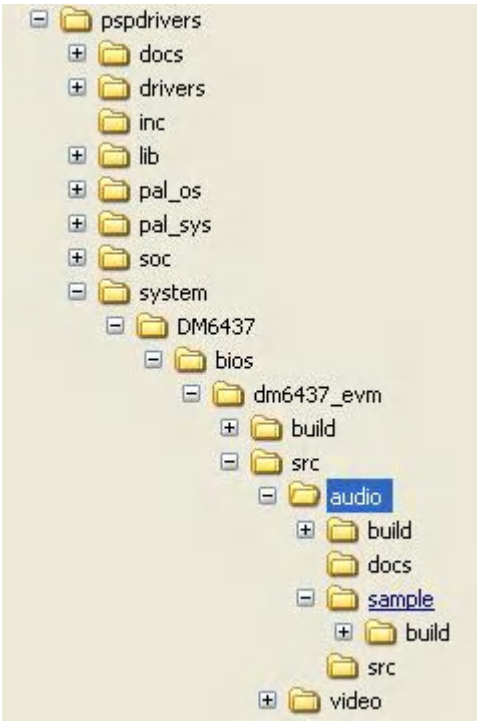

Figure 2-2 pspdrivers folder structure

In order to integrate the DSP/BIOS Audio driver with the pspdrivers package for DM6437/C6424 SoC, the user shall create a folder named "audio" under the **// pspdrivers\system\DM6437\bios\evmDM6437\src** for DM6437 and under the **// pspdrivers\system\C6424\bios\evm6424\src** for C6424. Inside the audio folder, build, docs, src and sample folder from the provided package.

- **build :** This folder contains the driver library project and driver library. The driver library shall be included in the application where Audio driver have to be used.
- **docs :** This folder contains Release notes document and User Guide Document.
  - Release Note gives the details about system requirements, steps to Install /Uninstall the Package.This document list the known issues of the driver.
  - User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.
- **sample :** This folder contains the sample applications that demonstrates the use of the driver. This sample applications demonstrates basic features of the driver. User can use this sample application as reference to make their applications. In addition to that this folder also contains the .tci files for McASP and McBSP instances.
  - **build :** This subfolder in sample folder contains sample application library as well as stand alone sample application project. User can run specific sample application by adding sample applications library in executable project.

2.2 Build

This section describes for each supported target environment, the applicable build options and supported configurations In addition to that it also describes how selected and featured capabilities can be enabled, how to install allowed user customizations and how the same can be realized.

The component might be delivered to user in different formats:

- Source-less ie., binary executables and object libraries only
- Source-inclusive ie., The entire source code used to implement the driver is included in the delivered product
- Source-selective ie., Only a part of the overall source is included. This delivery mechanism might be required either because; certain parts of the driver require source-level extensions and/or customization at the user's end or because, specific parts of the driver is exposed to user at the source-level to insure user's software development.

When source is included as part of the product delivery, the CCS project file is provided as part of the package. When object format is distributed, the driver header files are part of the "inc" folder and the driver library is provided in /drivers/lib folder.

2.2.1 Build Options

- Optimization level should be configured for –o2 for all drivers.
- PLL_INPUT_CLK compiler flag should be used to enable external clock source available on the EVM.

Chapter 3

Run-Time Interfaces/Integration Guide

This chapter discusses the DSP/BIOS Audio driver run-time interfaces that comprise the API classification & usage scenarios and the API specification itself in association with its data types and structure definitions.

3.1 Symbolic Constants and Enumerated Data types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. Described alongside the macro or enumeration is the semantics or interpretation of the same in terms of what value it stands for and what it means.

It is typical to classify the data types into logical groups and list them in alphabetical order for ease of use.

Table 3-1 Audio Driver Configuration enumerations

Enumeration	Description
ConfigCommand	Command issued an under lying layer to perform a particular task
PSP_audioAicMode	Audio codec will work in a master more or a slave mode
PSP_audioInputMode	Source of the audio input. From Mic or Line in input jack
PSP_audioOutputMode	Audio should be played back from speaker out or line out.

3.2 Data Structures

This section summarizes the entire user visible data structure elements pertaining to the DSP/BIOS Audio driver configuration interfaces.

The file **"psp_audio.h"** has a data structure to configure the Audio driver instance and the I/O channel used in the communication.

The "**PSP_audio_cfg**" contains the parameters used to configure the Audio IO mini driver instance. The device parameters are explained in the Table 3-2.

Table 3-2 PSP_audio_cfg Configuration parameters

Parameters	Description
mode	Audio codec mode of operation. Master or slave PSP_AUDIO_AIC_MASTER – AIC33 is in master mode PSP_AUDIO_AIC_SLAVE – AIC33 is in slave mode
inputFreq	Input frequency to McASP or McBSP module
sampleFreq	Output sampling rate
inputGain	Input gain control. The value should be in range of 1 and 99
outputGain	Output gain controls. The value should be in range of 1 and 99
inputMode	Pointer to the Input mode structure to select input source, MIC_IN or LINE_IN. In case of MIC_IN user can configure mic bias volatage.
outputMode	Audio output from system through Speaker out or LINE OUT

3.3 DSP/BIOS Audio Driver API Classification

User will have to select device id 0 if one wants to use McBSP peripheral in conjunction with Audio driver or select device id 1 if one wants to use McASP peripheral in conjunction with Audio driver.

3.3.1 DSP/BIOS Audio Driver Initialization

- **Device Id – 0 (McBSP Driver)**

  DSP/BIOS call initialization routine which internally invokes "**AUDIO_AIC33_init0 ()**" API of the Audio driver, which in turn initializes the global data used by the McBSP. Pinmux settings of McBSP driver also performed here.

  The initialization function sets the "inUse" field of both the McBSP port object instance and the channel object instance to "FALSE" to make sure that the driver is not being used by any applications.

- **Device Id – 1 (McASP Driver)**

DSP/BIOS call initialization routine which internally invokes "**AUDIO_AIC33_init1 ()**" API of the Audio driver, which in turn initializes the global data used by the McASP.

The initialization function sets the "inUse" field of both the McASP port object instance and the channel object instance to "FALSE" to make sure that the driver is not being used by any applications.

Final tci file should contain the following details which were explained above(Given is example for McBSP):

For uDevCode:

```
var udevCodec0 = bios.UDEV.create("udevCodec0");

udevCodec0.fxnTable = prog.extern("AUDIO_AIC33_FXNS");

udevCodec0.initFxn = prog.extern("AUDIO_AIC33_init0");

udevCodec0.fxnTableType = "IOM_Fxns";

udevCodec0.params = null;

udevCodec0.deviceId = 0;

udevCodec0.deviceGlobalDataPtr = null;
```

For dioCodec:

```
var dioCodec = bios.DIO.create("dioCodec");

dioCodec.comment = "DIO Adapter for IOM Codec driver" ;

dioCodec.deviceName = prog.get("udevCodec0");

dioCodec.useCallBackFxn = false;

dioCodec.chanParams = prog.extern("audio_cfg");
```

3.3.2 Driver Binding

The binding function for the Audio driver "audio_mdBindDev ()" API is called by the DSP/BIOS initialization routine is called.

The binding function should typically perform the following actions.

- Acquire the resources needed by the driver such as memory for port and channel object instances.
- Configure the device to match the DSP data format mode of Audio Codec.

3.3.3 IOM Channel Creation

After the successful completion of driver initialization and binding, the user can create an abstraction of the communication path between the application and Audio driver. The channel instances are created through a call to mdCreateChan () function, which runs as a result of a call from application call to channel creation API. The Audio driver allows user to create two channels (in input and output mode) for a given McASP/McBSP device instance on the SoC. The device IOM driver returns error status if the application attempts to create a channel in mode other than either input or output. For a given McASP/McBSP instance only one input and one output channels can be created.

3.3.4 Control Commands

The device IO Mini driver implements device specific control functionality which may useful for application designers. The device IOM driver supports the following control functionality.

3.3.4.1 AUDIO_STOP

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_STOP
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_STOP,NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Stops record/playback operation depending of the i/o stream passed.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.2 AUDIO_START

- **SYNOPSIS**

- Int SIO_ctrl(stream, cmd, arg)

- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_START
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_START, NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Starts record/playback operation depending of the i/o stream passed.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.3 AUDIO_PAUSE

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_PAUSE
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_PAUSE, NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

"Pause" record/playback operation of passed IO stream. No audio record/playback will take place when driver is in pause state.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.4 AUDIO_RESUME

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_RESUME
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_RESUME, NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

"Resume" record/playback operation of passed i/o stream. Call this command to continue with record/playback operation after "AUDIO_PAUSE" command is issued.

- **LIMITATIONS/CONSTRAINTS**

This command should be called after "AUDIO_PAUSE", if "AUDIO_ RESUME" command called before "AUDIO_PAUSE" then driver returns an error message.

3.3.4.5 AUDIO_MUTEON

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_MUTEON
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_MUTEON, NULL);
- **RETURN VALUE**

- if successful - SYS_OK
- non-zero device-dependent error value if unsuccessful

- **DESCRIPTION**

Mute the output stream. User will not hear any audio from LINE_OUT when driver is in “MUTEON” state

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.6 AUDIO_MUTEOFF

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_MUTEOFF
  - **args –** NULL
- **EXAMPLE**

  stat = SIO_ctrl(inStream,AUDIO_MUTEOFF, NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Take driver out of Mute state for the output stream.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.7 AUDIO_INPUT_SAMPLERATE

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_INPUT_SAMPLERATE
  - **args –** address of a new samplerate value

- **EXAMPLE**

  newSampleRate = 48000;

  stat = SIO_ctrl(inStream,AUDIO_INPUT_SAMPLERATE,&newSampleRate);

- **RETURN VALUE**

  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful

- **DESCRIPTION**

Modifies input stream's sample rate

- **LIMITATIONS/CONSTRAINTS**

Sample rate of input stream should not be changed while recording is going on. This may result in unexpected behaviour.

3.3.4.8 AUDIO_OUTPUT_SAMPLERATE

- **SYNOPSIS**

  - Int SIO_ctrl(stream, cmd, arg)

- **ARGUMENTS**

  - **stream –** stream handle
  - **cmd –** AUDIO_OUTPUT_SAMPLERATE
  - **args –** address of a new samplerate value

- **EXAMPLE**

  newSampleRate = 48000;

  stat = SIO_ctrl(outStream,AUDIO_OUTPUT_SAMPLERATE,&newSampleRate);

- **RETURN VALUE**

  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful

- **DESCRIPTION**

Modifies output stream's sample rate

- **LIMITATIONS/CONSTRAINTS**

Sample rate of output stream should not be changed while playback is going on. This may result in unexpected behaviour.

3.3.4.9 AUDIO_IN_SELECT

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_IN_SELECT
  - **args –** NULL
- **EXAMPLE**

  audio_cfg.inSelect = PSP_AUDIO_MIC_IN;

  stat = SIO_ctrl(inStream,AUDIO_IN_SELECT, NULL);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Changes the input/record source based on the structure configuration. “audio_cfg” structure is passed to reconfigure AIC33 completely. It implies that if user modifies any other parameter of the structure, AIC33 will be configured with that configuration. It will not retain any old configuration of AIC33.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.10 AUDIO_OUT_SELECT

- This command is not supported due to hardware limitation

3.3.4.11 AUDIO_IN_GAIN_CONTROL

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_IN_GAIN_CONTROL
  - **args –** address of a new gain value
- **EXAMPLE**

  newGain = 75;

stat = SIO_ctrl(inStream,AUDIO_IN_GAIN_CONTROL,(Arg)&newGain);

- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Changes gain of input channel according to new gain value passed.

- **LIMITATIONS/CONSTRAINTS**

Value of gain should be in the range of 1 and 99

3.3.4.12 AUDIO_OUT_GAIN_CONTROL

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**
  - **stream –** stream handle
  - **cmd –** AUDIO_OUT_GAIN_CONTROL
  - **args –** address of a new gain value
- **EXAMPLE**

  newGain = 75;

  stat = SIO_ctrl(outStream,AUDIO_OUT_GAIN_CONTROL,(Arg)&newGain);
- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Changes gain of output channel according to new gain value passed.

- **LIMITATIONS/CONSTRAINTS**

Value of gain should be in the range of 1 and 99

3.3.4.13 AUDIO_DEVICE_RESET

- **SYNOPSIS**
  - Int SIO_ctrl(stream, cmd, arg)
- **ARGUMENTS**

- **stream –** stream handle
- **cmd –** AUDIO_DEVICE_RESET
- **args –** NULL

- **EXAMPLE**

  stat = SIO_ctrl(outStream, AUDIO_DEVICE_RESET,NULL);

- **RETURN VALUE**

  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful

- **DESCRIPTION**

Resets the audio driver peripheral i.e. McASP or McBSP. After this command is issued successfully, audio record and playback will not continue. Please follow below mentioned steps for restarting record and playback again. The steps are same for both McASP as well as McBSP peripherals

- Step 1 – Issue AUDIO_DEVICE_RESET command
- Step 2 – Free the allocated memory, if any
- Step 3 – Delete both the channels
- Step 4 – Create both the channels again
- Step 5 – (Optional) Only in case of McASP in master mode (i.e. AIC33 in slave mode) call IOCTL “PSP_CTRL_RCV_GPIO_INPUT”.
- Step 6 – Prime the driver

- **LIMITATIONS/CONSTRAINTS**

None

3.3.4.14 AUDIO_MODIFY_LOOPJOB

- **SYNOPSIS**

  - Int SIO_ctrl(stream, cmd, arg)

- **ARGUMENTS**

  - **stream –** stream handle
  - **cmd –** AUDIO_MODIFY_LOOPJOB
  - **args –** address McASP/McBSP channel param strucutre

- **EXAMPLE**

  PSP_mcaspChanParams chanPrams;

```
unsigned short loopBuf[512];

chanParams.userLoopJobBuffer = & loopBuf[0];

chanParams.userLoopJobLength = 512;

stat = SIO_ctrl(outStream,AUDIO_MODIFY_LOOPJOB,(Arg)&chanParams);
```

- **RETURN VALUE**
  - if successful - SYS_OK
  - non-zero device-dependent error value if unsuccessful
- **DESCRIPTION**

Enables application provided loopjob or internal loopjob depending on the input parameter passed. If no buffer is provided then driver will enable internal loopjob. Loopjob only takes effect when driver is starved of the requests.

- **LIMITATIONS/CONSTRAINTS**

None

3.3.5 IOM Channel Deletion

Application can free the resources held by the channel, if the channel is currently not in use, by calling SIO_delete () API. The corresponding “audio_mdDeleteChan ()” function of the device IOM driver shall run from the application context and should de-allocate the specified channel object.

3.3.6 Audio IO Mini driver unbinding

The resources allocated by the “audio_mdBindDev ()” function are freed by calling device “audio_mdUnBindDev ()”.

Chapter 4

Architecture

The device driver described here is part of an IOM mini-driver. That is, it is implemented as the lower layer of a 2-layer device driver model. The upper layer is called the class driver and can be either the DSP/BIOS GIO, SIO/DIO, or PIP/PIO modules. The class driver provides an independent and generic set of APIs and services for a wide variety of mini-drivers and allows the application to use a common interface for I/O requests. Figure 3-1 shows over all DSP/BIOS device driver architecture.

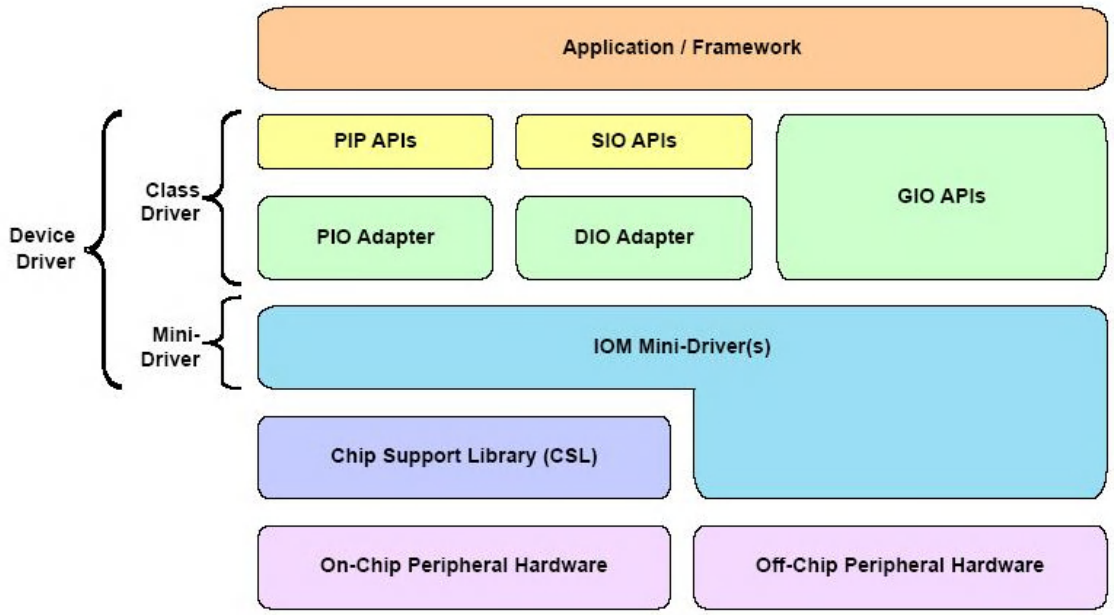

Figure 3-1DSP/BIOS Device Driver Model

Mini-driver implementation is split into a codec-specific portion and a generic portion that will work across many different codecs. Figure 3-2 shows the data flow between the components in a system in which the mini-driver is split into a generic part i.e. McASP/McBSP driver and audio codec-specific part.

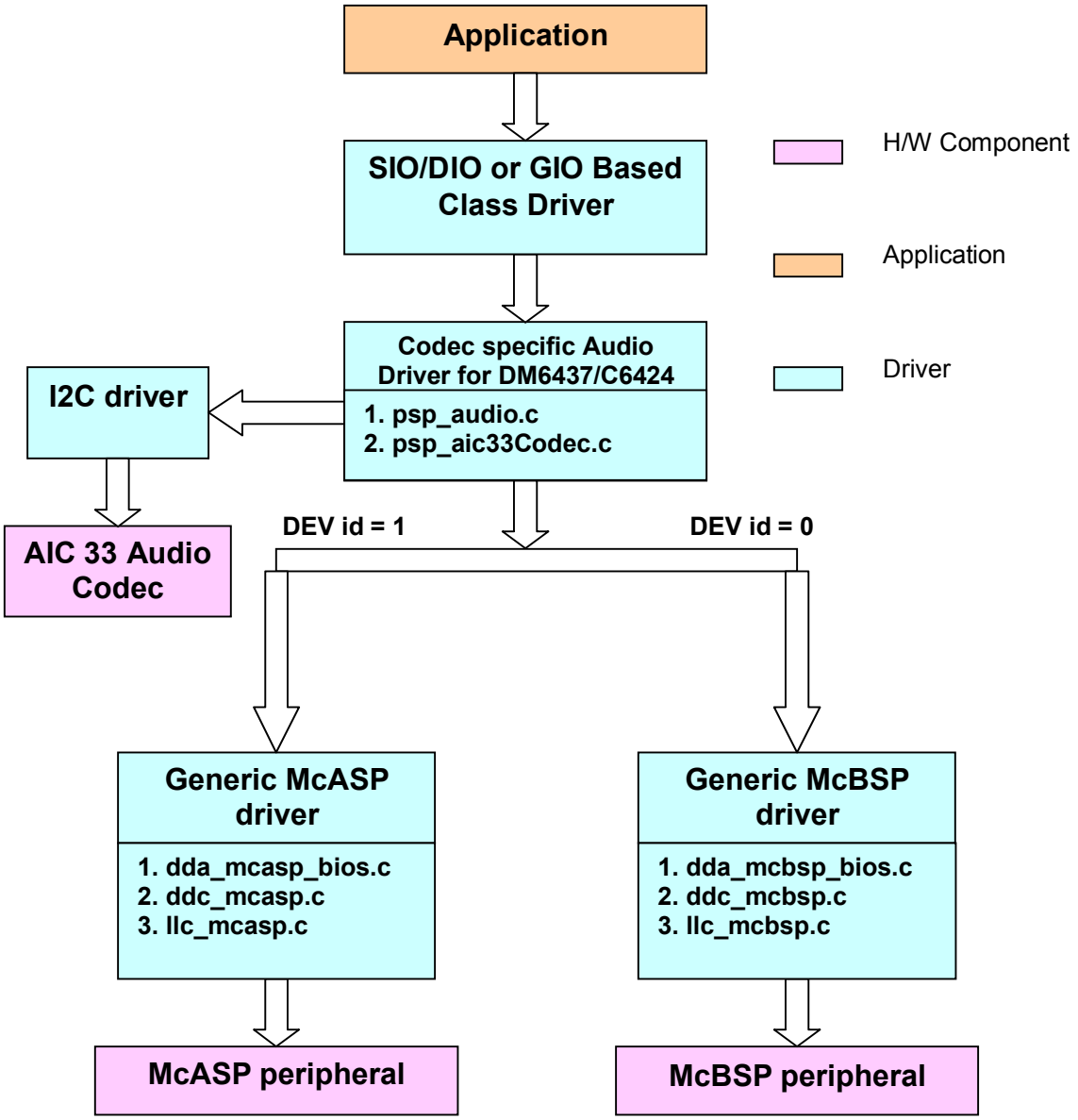

Figure 3-2 Codec Device Driver Partitioning

NOTE: Chip support library (CSL) here is register layer CSL only.

The codec specific portion of the mini-driver uses two codec specific functions, mbBindDev() and mdCreateChan(), to configuration audio codec. These functions then call mbBindDev() and mdCreateChan() in the generic driver to complete generic portions of the driver initialization. The only thing the codec-specific part does is to set up the codec and leaves the transfers of samples to

the generic device driver. The fact that this device driver uses the generic device driver is hidden from the user in all aspects except that the generic device driver library has to be linked into the application. The function mdCreateDev() is responsible for configuring the codec It does minimal setup required and then calls a function called aic33Codec_config () function in psp_aic33Codec.c to perform most of the configuration work required to prepare audio codec for record and playback operation. The function mdCreateChan() generates the EDMA configuration used for the data transfers.

Chapter 5

Audio Codec API Description

This section details the APIs that configures Audio codec on DM6437/C6424 EVM.

5.1 Default Configuration

The default configuration of Audio driver at the creation time is as follows:

- AIC to operate in a master mode and McASP/McBSP in a slave mode.
- Input frequency to the McASP/McBSP peripheral – 99 MHz i.e. CPU frequency / 6.
- Default sampling rate – 8000 Hz
- Input Gain – 50
- Output Gain – 99
- Input mode – LINE IN
- Output Select – LINE OUT

5.2 API

5.2.1 aic33Codec_config () – Configure AIC33 codec

Int	**aic33Codec_config**	**(**	**Uint32**	**Id**
			PSP_audio_cfg*	**audioCfg**
		)		

Parameters:

Id	Audio codec id
audioCfg	Audio codec configuration structure

Returns:

On failure, it returns a -1, otherwise, it returns a valid handle to the instance so initialized.

5.2.2 aic33_closeCodec () – close the AIC33 codec instance

Void	**aic33_closeCodec**	**(** **)**	**aic33_CodecHandle**	**aic33handle**

Parameters:

aic33handle Handle of the opened instance

Returns:

Void

5.2.3 aic33_inGainControl () – Configure the gain of input channel

Int	**aic33_inGainControl**	**(**	**Uint8**	**newGain**
			PSP_audio_cfg*	**audioCfg**
		)		

Parameters:

newGain New gain setting for the input channel. New gain value should be in the range of 1 to 99 otherwise function will return an error

audioCfg Audio codec configuration structure

Returns:

On failure returns -1, on success returns 0.

5.2.4 aic33_outGainControl () – Configure the gain of input channel

Int	**aic33_outGainControl**	**(**	**Uint8**	**newGain**
			PSP_audio_cfg*	**audioCfg**
		)		

Parameters:

newGain New gain setting for the input channel. New gain value should be in the range of 1 to 99 otherwise function will return an error

audioCfg Audio codec configuration structure

Returns:

On failure returns -1, on success returns 0.

5.2.5 aic33_sampleRateControl () – Configure AIC33 for passed sample rate

Int	**aic33_sampleRateControl**	**(** **)**	**PSP_audio_cfg***	**audioCfg**

Parameters:

audioCfg Audio codec configuration structure

Returns:

On failure, audio codec is configured to default sampling rate i.e 44.1 KHz and, error (-1) is returned, on success 0 is returned.

5.3 Dependency of Sample application:

Following Components needs to be linked for successful build and functionality of the application.

- McASP
- McBSP
- I2C
- Audio
- PAL_OS
- SoC specific PAL_SYS
- EDMA3

Note:

The AIC33 codec driver uses I2C driver which is an asynchronous driver. However, the codec relies on the result of the operation which means, it needs a synchronous operation. Hence, it implements a semaphore to pend on completion of this operation. The semaphore is posted by from inside a callback registered to the I2C driver, by the codec driver, during open. However, the call back is called only in interrupt context. Thus the I2C driver must be configured to work in interrupt mode, by the sample application. If this not done the codec operations shall fail.

Appendix A

DSP/BIOS Audio Driver References

References

[1] AIC33 audio codec datasheet (07 Jul 2006). To download the datasheet please visit

http://focus.ti.com/docs/prod/folders/print/tlv320aic33.html