Texas Instruments

DSP/BIOS McBSP Device 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 McBSP device driver.** This manual provides necessary information regarding how to effectively install, build and use **DSP/BIOS McBSP device driver** in user systems and applications.

This manual provides details regarding how the **DSP/BIOS McBSP device 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 McBSP device driver.

Terms and Abbreviations

Term/Abbreviation	Description
3PDMAC	Third Party Direct Memory Access Controller.
API	Application Programmers Interface
IP	Intellectual Property
CSLR	TI Chip Support Library Register Layer– primitive h/w abstraction
DDK	Device Driver Development Kit
LLC	Lower Level Controller
EDMA	Enhanced Direct Memory Access Controller
ISR	Interrupt Service Routine
IOM	I/O Mini Driver Model
McBSP	Multi-channel Buffered Serial Port
PSP	Platform Support Package

Related Documentation

Internal

This is a list of documents that are TI Proprietary and Strictly Private. Exposure to audience outside TI will need due considerations and approvals from TI Legal authorities.

This is a list of documents that are TI Proprietary and Strictly Private. Exposure to audience outside TI will need due considerations and approvals from TI Legal authorities.

- McBSP512 Hardware Module Specifications.
- EDMA Hardware Module Specifications.
- spru943a.pdf

External

- None

Trademarks

Revision History

Date	Author	Revision History	Version
8th August 2006	Pratik Joshi	Created the Document	1.0
16th August 2006	Pratik Joshi	Modified to include details of Loopjob API support	1.1
28th August 2006	Pratik Joshi	Driver architecture changed to comply PSP architecture	1.2
21st September 2006	Pratik Joshi	BIOS version changed to 5.31	13
28th September 2006	Ankur Verma	Change the name of the IOCTLS and modified the APIs	1.4
December 1, 2006	Pratik Joshi	Porting description added and ready for BFT 1 release	1.5
January 16, 2007	Pratik Joshi	BIOS version changed 5.31.02	1.6
January 27, 2007	Pratik Joshi	CCS version modified	1.7
February 20, 2007	Pratik Joshi	Release 0.6.0	1.8
March 05. 2007	Pratik Joshi	Modified for the changes in driver	1.9
March 22, 2007	Pratik Joshi	Updated for release 0.6.0	1.10
March 08, 2007	Pratik Joshi	Updated for GA release 1.0.0	1.11
June 22, 2007	Anuj Aggarwal	Updated for GA release 1.00.01	1.12
June 29, 2007	Amit Chatterjee	Modified Release Version	1.13
July 18, 2007	Pratik Joshi	Modified Release Version	1.14
May 21, 2008	Chandan Nath	Updated for adding compiler switches in build options	1.15

Contents

Tables

Figures

Chapter 1

DSP/BIOS McBSP Driver Introduction

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

1.1 Overview

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

A brief definition of the component is provided at this point – its main characteristics and purpose.

1.1.1 Supported Services and features

The DSP/BIOS McBSP 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.
- Keeps External Frame Sync.
- Provides Asynchronous I/O mechanism.
- Operates in DMA Interrupt Mode.
- Built-in EDMA ping-pong buffering for continuous data streaming for input and output channels.
- User configurable word size for McBSP(8,12,16,20,24 and 32bit)
- By default configures the McBSP and EDMA channels to match the DSP data format mode of the audio codec.
- Features supported by the McBSP driver, which can be directly used by the Audio codec driver

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.

Configurable to generate the Frame synchronization and clock signals internally.

Run-time configuration of the data and frame length for McBSP.

1.1.2 System Requirements

Details about the tools and the BIOS version that the driver is compatible with can be found in the system Release Notes.

Chapter 2

Installation Guide

The DSP/BIOS McBSP 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 McBSP driver the following directory structure is found in the driver's directory.

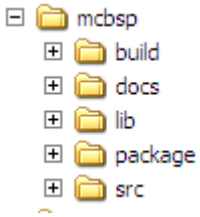


Figure 1: McBSP driver structure

This top level mcbsp folder contains mcbsp driver psp header file and XDC package files (package.bld, package.xdc and package.xs)

- **build:** This folder contains mcbsp driver library project file. The generated driver library shall be included in the application where MCBSP driver have to be used.
- **docs:** This folder contains architecture document, datasheet, release notes and user guide.

  Architecture document contains the driver details which can be helpful for the developers as well as consumers to understand the driver design.

  Datasheet gives the idea about the memory consumption by the driver and description of the top level APIs.

  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.

- **Lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **Package:** This folder contains files generated by XDC tool.
- **Src** : This folder contains mcbsp driver source files. It also contains header files that are used by the driver.

2.2 Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed 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.
Following compiler switches are used to compile for different options.

- **_DEBUG**
  This is used as a flag to compiler whether to include the debug statements inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.
- **CHIP_XXXX**
  The CSL layer is written in a common file for all the variants of a SOC. This flag differentiates the variant we are compiling for, for e.g. - CHIP_DM648, and the CSL definitions for that variant appropriately gets defined for register base addresses, num of ports of a peripheral etc.

- **MCBSP_INSTRUMENTATION_ENABLED**
  This flag is passed to the compiler to include the instrumentation code parts into the final image/lib of the program. This helps build the iRelease/iDebug versions of the image/lib with a common code base.

- **GBL_INT**
  This compiler option is used to Enable/Disable Global error Interrupt Handling for MCBSP module.

Chapter 3

Run-Time Interfaces/Integration Guide

This chapter discusses the DSP/BIOS McBSP 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. DSP/BIOS McBSP Driver Configuration defines.

Defines	Description
PSP_MCBSP_NUM_INSTANCES	Number of McBSP instances on the SoC
PSP_MCBSP_NUM_CHANS	Number of channels supported by the McBSP driver
PSP_MCBSP_RX	Input channel Index
PSP_MCBSP_TX	Output channel Index
MAXLINKCNT	Maximum no. of EDMA link param tables that a driver will use
PSP_MCBSP_MAX_REQ	Queue depth. This defines the maximum requests that a driver can queue for each input and output channel. This is also limited by the queue depth of SIO/DIO or GIO class driver. In this case whichever is less will define the queue depth of the driver
PSP_MCBSP_FLUSHED	Request status returned in case request flush command is issued

	from application
PSP_MCBSP_ABORTED	Request status returned in case request abort command is issued from application
PSP_MCBSP_DMA_ERROR	Error returned when driver fails to either configure the edma channel or edma driver fails to complete the transfer successfully
PSP_MCBSP_INVALID	In case request queue is empty this status is returned
PSP_MCBSP_VALID	In case request queue is having pending requests this status is returned

Table 3-2. Supported enumerated data types

Enum	Description
PSP_mcbspMode	Data type to define the mode of channel, input mode or output mode
PSP_McbspChannelStatus	Created channel status enum type. The status of created channel should be "PSP_MCBSP_ALLOCATED"
PSP_mcbspFrmPhase	McBSP data stream contains whether single phase or dual phase
PSP_McbspDataDelay	Delay for the first data bit from the start of the data frame
PSP_McbspBclkMode	Specifies whether the bit clock for the McBSP transmitter or receiver is generated internally using SRGR or externally fed
PSP_McbspFrSynchMode	Specifies whether the frame sync signal for the McBSP transmitter or receiver is generated internally using FSGR or externally fed
PSP_McbspBclkPol	Specifies whether the bit clock signal for the McBSP transmitter or receiver is of rising edge or falling edge polarity
PSP_McbspFsPol	Specifies whether the frame sync signal for the McBSP transmitter or

	receiver is of active low or active high polarity
PSP_McbspSrgClkPol	Set the polarity of the CLKS pin used to generate Frame Sync and bit clock
PSP_McbspSrgClk	Source for McBSP Sample Rate Generator to generate the bit clock and frame sync signals in master mode
PSP_mcbspLoopback	Whether the McBSP should be enabled in digital loopback mode or not
PSP_McbspSrgFsgm	Whether Transmit frame sync signal (FSX) due to DXR(1/2)-to-XSR(1/2) copy or Transmit frame sync signal driven by the sample rate generator frame sync signal, FSG
PSP_McbspIoctlCmd	Enum for IOCTL command supported by McBSP driver

3.2 Data Structures

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

The file **"psp_mcbsp.h"** has data structures to configure the McBSP device driver instance and the I/O channel used in the communication.

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

Table 3-3. DSP/BIOS McBSP IO Mini Driver's Device Configuration parameters.

Parameters	Description
enablecache	Set to "TRUE" if the submitted buffer resides in external memory.
enableSrgr	Set to "TRUE" to the enable the McBSP sample rate generator to generate the Bit Clock signals.
enableFsg	Set to "TRUE" to enable the McBSP Frame Synchronization generator to generate frame synchronization signals.
mcbspCfgPtr	Pointer to the CSL configuration structure to be used for the McBSP device.

The "**PSP_mcbspChanParams**" contains the parameters used to configure the McBSP IOM device driver channel object instance. The channel parameters are explained in the Table 3-4:

Table 3-4. Channel setup parameters for DSP/BIOS McBSP IO mini driver

Parameters	Description
"noOfTdmChans"	This parameter should be set to the number of TDM channels the McBSP is using for this IOM channel (e.g., 1 for mono, 2 for stereo etc.). This value will be used by the driver to maintain the frame sync.
"intrNum"	GEM interrupt number dedicated for a particular input and output channel.
"wordWidth"	Word width that driver should expect from audio codec
"userLoopJobBuffer"	"userLoopJobBuffer" shall be the buffer to be transferred when the loop job is running. If the buffer is null then driver uses the local buffer.
"userLoopJobLength"	Number of frames to be transferred before calling the periodic callback. Application shall specify the interval of periodic callback through the "userLoopJobLength"
"edmaHandle"	Handle to the EDMA Driver
"gblCbk"	callback required when global error occurs must be callable directly from the ISR context

The "**PSP_mcbspDataConfig**" specifies the configuration for the McBSP data stream including the whether it is single phase or dual phase, number of frames, the word length in each phase and data delay.

Table 3-5 McBSP data stream configuration structure

Parameters	Description
phaseNum	Phase of the McBSP data stream
wrdLen1	Length of the data word in first phase. This has to be in terms of bits
wrdLen2	Length of the data word in second when dual phase is selected. This has to be in terms of bits
frmLen1	Length of the data frame in first phase in terms of number of words

frmLen2	Length of the data frame in second phase in terms of number of words
dataDly	data delay – delay in the data stream after frame sync is received/generated

The "**PSP_srgConfig**" Configure the Sample rate generator to generate the BCLK and Frame Sync signals at the specified rate in McBSP master mode

Table 3-6 Sample Rate Generator Configuration structure

Parameters	Description
srgInputClkMode	Source for the Sample rate generator
srgFsgmMode	Sample rate generator transmit sync mode selection
bclkRate	Set Bit clock rate - Divide down value of SRGR input clock
srgFrmPulseWidth	Set the Frame Sync Pulse width in terms of clock ticks
srgFrmPeriod	Set the frame sync signal period in terms of clock ticks
srgClkPolarity	Set Frame sync and Bit clock polarity

"**PSP_mcbspReq_t**" required to be filled by application while making a request to McBSP Driver

Table 3-7 McBSP Request structure

Parameters	Description
addr	Pointer to the application Buffer
size	Size of the application Buffer
actualSize	Acutal bytes transmitted by the driver
cbFxn	McBSP Application call back per Request. This call back is called in the Interrupt Context
appPtr	Application Data. This data can be filled by application for request identification after completion
status	Status of the transfer. This data is filled by the driver
cmd	packet command, read, write, flush or abort
misc	McBSP Application Data

3.3 DSP/BIOS McBSP Driver API Classification

3.3.1 DSP/BIOS McBSP Driver Initialization

DSP/BIOS call initialization routine which internally invokes "**MCBSP_IOM_init ()**" API of the McBSP IO mini driver, which in turn initializes the global data used by the McBSP driver.

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.

3.3.2 Driver Binding

The binding function for the McBSP IOM driver “mdBindDev ()” API is called by the DSP/BIOS initialization routine after MCBSP_IOM_init () function is called.

The binding function should typically perform the following actions.

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

3.3.3 IOM Channel Creation

After the successful completion of McBSP driver initialization and binding, the user can create an abstraction of the communication path between the application and McBSP IOM driver. The channel instances are created through a call to mdCreateChan () function, which runs as a result of a call from GIO_create () API. The McBSP driver allows user to create two channels (in input and output mode) for a given McBSP device instance on the SoC. McBSP IOM driver returns error status if the application attempts to create a channel in unsupported channel mode by the driver.

3.3.4 Channel Control

McBSP IO Mini driver implements device specific control functionality which may useful for application designers. Application may invoke the control functionality through calls to GIO_control ().The control functions supported by the McBSP driver shall be useful when the McBSP IOM is used by the any Audio Codec specific IOM driver. McBSP IOM driver supports the following control functionality.

3.3.4.1 PSP_CTRL_McBSP_STOP

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);
- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_STOP
  - **args** - NULL
- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  This command puts receive/transmit section in reset. Record/Playback stops as soon as this command is issued. To resume the record/playback process, user needs to issue PSP_CTRL_McBSP_START IOCTL.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.2 PSP_CTRL_McBSP_START

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_START
  - **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED– if no errors
  - IOM_EBADARGS – if parameters are invalid

- **DESCRIPTION**

  This command takes receive/transmit section out of reset. Record/playback is possible.

- **LIMITATIONS/CONSTRAINTS**

  Issuing this command will not have any impact if PSP_CTRL_McBSP_STOP is not issued before calling this IOCTL.

3.3.4.3 PSP_CTRL_McBSP_LOOPBACK

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_LOOPBACK
  - **args** – address of mdIoctl variable

- **Example**

  PSP_mcbspLoopback mdIoctl;

  mdIoctl = PSP_MCBSP_LOOPBACK_ON;

status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_LOOPBACK, (Ptr) &mdIoCtl);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS – if Parameters are not valid

- **DESCRIPTION**

  It enables or disables the McBSP digital loopback mode.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.4 PSP_CTRL_McBSP_PAUSE

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_PAUSE
  - **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS – if Parameters are not valid

- **DESCRIPTION**

  Pause will take maximum of two packets time to take an effect. When driver is in PAUSE state requests will be queued in driver's request queue. Instead of transferring actual recorded/played back data, driver will transfer a buffer filled with 0s to ensure no frame sync loss takes place.

- **LIMITATIONS/CONSTRAINTS**

  As driver will accept the requests from application in PAUSE state, it is possible that driver request queue may get full and stop taking any further requests.

3.3.4.5 PSP_CTRL_McBSP_RESUME

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_RESUME
  - **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS – if Parameters are not valid

- **DESCRIPTION**

  Resumes record/playback operation from where it was PAUSED. If the requests are already queued, driver will take the first request in the queue and start the operation. If the there are not pending requests buffer filled with 0 is given for EDMA transfer until application submits a requests.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.6 PSP_CTRL_McBSP_MUTE_ON:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_MUTE_ON
  - **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS – if Parameters are not valid
  - IOM_ENOTIMPL - if IOCTL call for Input channel

- **DESCRIPTION**

  The request buffer is replaced with a mute buffer filled with 0 at the time of playback. When driver is in MUTE state, it will accept the requests from the application. Driver will playback the audio normally with only difference being the request buffer address is replaced with mute buffer. User will not hear any playback in the MUTE state.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.7 PSP_CTRL_McBSP_MUTE_OFF

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**

- **gioChan** – Channel handle
- **cmd** - PSP_CTRL_McBSP_MUTE_OFF
- **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS – if Parameters are not valid
  - IOM_ENOTIMPL - if IOCTL call for Input channel

- **DESCRIPTION**

  The mute buffer is replaced with request buffer at the time of playback. This command will take driver back into a normal playback state.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.8 PSP_CTRL_McBSP_CHAN_RESET

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_CHAN_RESET
  - **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  Put channel in reset and re-configure the channel with the default configuration. All the requests in the queue are aborted.

- **LIMITATIONS/CONSTRAINTS**

Record/playback will not start after this command is issued. The steps to reconfigure device is provided in document BIOS_AUDIO_Driver_UserGuide.doc, section 3.3.4.1

3.3.4.9 PSP_CTRL_McBSP_DEVICE_RESET

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_DEVICE_RESET

- **args** - NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  Driver puts receive and transmit section in reset state, disables frame sync and sample rate generator. EDMA transfer is also disabled. All the requests in both receive and transmit queues are aborted and peripheral is configured with the default values.

- **LIMITATIONS/CONSTRAINTS**

  Once this command is issue successfully record and playback will not happen. The steps to reconfigure device is provided in document BIOS_AUDIO_Driver_UserGuide.doc, section 3.3.4.1

3.3.4.10 PSP_CTRL_McBSP_CHG_ADDR

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_CHG_ADDR
  - **args** – value of new address

- **EXAMPLE**

  Uint16 destBuffer[512 + 32] = {0};

  Uint16 pdestBuffer = (Uint16)(((Uint32)((destBuffer) + 31) >> 5) << 5);

  Status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_CHG_ADDR, (Ptr)pdestBuffer);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  This command changes the address for of source or destination for EDMA transfer. In case the channel is input, destination address is modified and in case the channel is output, source address is modified.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.11 PSP_CTRL_McBSP_CONFIG_DATA

- **SYNOPSIS**

- Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_CONFIG_DATA
  - **args** – address of PSP_mcbspDataConfig structure's instance, Refer section 3.2 of BIOS_McBSP_Driver_UserGuide.doc for PSP_mcbspDataConfig structure.

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS - if Invalid Parameters

- **DESCRIPTION**

  This command reconfigures the McBSP receive/transmit section depending on the channel.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command at the time when record/playback is going on may result in noise. It is not suggested that this command is called while record/playback is going on.

3.3.4.12 PSP_CTRL_McBSP_SRGR_START:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SRGR_START
  - **args** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It starts McBSP sample rate generator.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.13 PSP_CTRL_McBSP_SRGR_STOP:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SRGR_STOP
  - **args** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It stops McBSP sample rate generator.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.14 PSP_CTRL_McBSP_FSGR_START:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **RGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_FSGR_START
  - **args** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It starts McBSP frame sync generator.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.15 PSP_CTRL_McBSP_FSGR_STOP:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_FSGR_STOP
  - **args** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED - if no errors

- **DESCRIPTION**

  It stops McBSP frame sync generator.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.16 PSP_CTRL_McBSP_SET_CLKMODE:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SET_CLKMODE
  - **args** – address of mdIoctl variable

- **EXAMPLE**

  PSP_McbspBclkMode mdIoCtl;

  mdIoCtl = (PSP_MCBSP_BCLKCLKMODE_INTERNAL) ;

  Status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_SET_CLKMODE, (Ptr) &mdIoCtl);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It sets the bit-clock mode (internal or external) for the McBSP transmitter or receiver sections depending on the mode of the channel.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command while record/playback is going on may result in frame sync loss and record/playback may not be performed properly

3.3.4.17 PSP_CTRL_McBSP_SET_FRMSYNCMODE:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SET_FRMSYNCMODE
  - **args** – address of mdIoctl variable

- **EXAMPLE**

  PSP_McbspBclkMode mdIoCtl;

  mdIoCtl = (PSP_MCBSP_ FRSYNCHMODE_INTERNAL) ;

  Status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_SET_FRMSYNCMODE, (Ptr) &mdIoCtl);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It sets the frame sync clock mode (internal or external) for the McBSP transmitter or receiver sections depending on the mode of the channel.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command while record/playback is going on may result in frame sync loss and record/playback may not be performed properly

3.3.4.18 PSP_CTRL_McBSP_CONFIG_SRGR:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_CONFIG_SRGR
  - **args** – address of PSP_srgConfig structure's instance, Refer section 3.2 of BIOS_McBSP_Driver_UserGuide.doc for PSP_srgConfig structure.

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_EBADARGS - if Invalid Parameters

- **DESCRIPTION**

  Configure the sample rate generator to generate the bit clock and frame synchronization signals at the specified rate.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command while record/playback is going on may result in frame sync loss and noise

3.3.4.19 PSP_CTRL_McBSP_SET_BCLK_POL:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SET_BCLK_POL
  - **args** – address of mdIoctl variable

- **EXAMPLE**

  PSP_McbspBclkPol mdIoctl;

  mdIoctl = PSP_MCBSP_BCLK_FALLING_EDGE;

  Status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_SET_BCLK_POL,& mdIoctl);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It sets the polarity of the bit clock to rising or falling edge.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command while record/playback is going on may result in frame sync loss and noise

3.3.4.20 PSP_CTRL_McBSP_SET_FRMSYNC_POL:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_SET_FRMSYNC_POL
  - **args** – address of mdIoctl variable

- **EXAMPLE**

  PSP_McbspFsPol mdIoctl;

  mdIoctl = PSP_MCBSP_FSPOL_ACTIVE_LOW;

  Status = GIO_control((Ptr)ipChanHandle, PSP_CTRL_McBSP_SET_FRMSYNC_POL,& mdIoctl);

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It sets the frame synchronization polarity (Active high or active low) when the frame synch signal is generated by the McBSP sample rate generator.

- **LIMITATIONS/CONSTRAINTS**

  Calling this command while record/playback is going on may result in frame sync loss and noise

3.3.4.21 PSP_CTRL_McBSP_MODIFY_LOOPJOB:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_MODIFY_LOOPJOB
  - **args** – address of PSP_mcbspChanParams structure's instance, Refer section 3.2 of BIOS_McBSP_Driver_UserGuide.doc for PSP_mcbspChanParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors
  - IOM_ENOTIMPL - if argument is NULL

- **DESCRIPTION**

  User defined loopjob buffer or internal loopjob will be transferred when the loop job is running. Loopjob will come in picture only when driver is starved for request from application. There are two types of loopjob, one is user provided loopjob where user provides the loopjob buffer and the other is internal loopjob. In internal loopjob, driver uses an internal buffer of zero.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.22 PSP_CTRL_McBSP_RECEIVE_SYNCERR_INT_ENABLE:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - gioChan – Channel handle
  - cmd-PSP_CTRL_McBSP_RECEIVE_SYNCERR_INT_ENABLE
  - args – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – If no errors

- **DESCRIPTION**

  It enables receive Interrupt.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.4.23 PSP_CTRL_McBSP_XMIT_SYNCERR_INT_ENABLE:

- **SYNOPSIS**
  - Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);

- **ARGUMENTS**
  - **gioChan** – Channel handle
  - **cmd** - PSP_CTRL_McBSP_XMIT_SYNCERR_INT_ENABLE
  - **args** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – if no errors

- **DESCRIPTION**

  It enables transmit Interrupt.

- **LIMITATIONS/CONSTRAINTS**

  None

3.3.5 Transfer Requests Submission

Application can submit a transfer request using McBSP class driver API GIO_submit () which in turn creates an IOM packet containing the all the transfer parameters needed by the IOM driver to program the underlying hardware for data transfer. The mdSubmitChan function of the McBSP IOM driver must handle command code passed to it as part of the IOM_Packet structure. Depending on the command code, it either handles the code or returns the IOM_ENOTIMPL (not implemented) error code.
The currently supported McBSP IOM mini-driver command codes are: IOM_READ, IOM_WRITE, IOM_ABORT, and IOM_FLUSH.

Note: Maximum size of single request by an application is 16K samples.

- **IOM_READ.** Drivers that support input channels must implement IOM_READ.
- **IOM_WRITE.** Drivers that support output channels must implement IOM_WRITE.
- **IOM_ABORT and IOM_FLUSH.** To abort or flush I/O requests already submitted, all I/O requests pending in the mini-driver must be completed and returned to the class driver. The mdSubmitChan function should dequeue each of the I/O requests from the mini driver's channel queue. It should then set the size and status fields in the IOM_Packet. Finally, it should call the cbFxn for the channel.
  - ❖ While aborting, all input and output requests are discarded.
  - ❖ While flushing, all pending output requests and processed normally and all pending input requests are discarded. This

requires the processing of each IOM_Packet in the original order they were queued up to the channel. Until all the requests are flushed, the calling task i.e. the task from where IOM_FLUSH is called will wait till all the requests are flushed. Driver will not accept any packet requests until flush is complete.

3.3.6 ISR processing for transfer completion interrupt

The McBSP IOM driver handles the EDMA transfer completion interrupt, which is raised by the EDMA after a transfer completion by the EDMA Transfer Controller. McBSP IOM driver uses two McBSP synchronized EDMA channels, one in input mode(McBSP data reception) and one in output mode(McBSP data transmission) for data transfer. When transfer EDMA completion interrupt occurs, the driver fetches the completed packet from the head of the IO packet queue for the channel and submits back the packet to the upper layer which intern calls the corresponding application callback.

3.3.7 IOM Channel Deletion

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

3.3.8 McBSP IO Mini driver unbinding

The “mdUnBindDev ()” shall free resources allocated by the “mdBindDev ()” function.

3.4 McBSP IOM driver’s API Usage Scenarios/Integration Example

The McBSP IOM driver is a generic one that may work across (not restricted) many codec drivers. The audio codec specific driver has to supply the device and channel specific configuration parameters to use the services of the McBSP IOM driver.

Before data communication between an application and a device can begin, a channel instance handle must be returned to the application by a call to GIO_create () API. The channel handle represents a unique communication path between the application and McBSP device driver. All subsequent operations that talk to the driver shall use this channel handle. A channel object typically maintains data fields related to a channel's mode, I/O request queues, and possibly driver state information. Application should relinquish channel resources by deleting all channel instances when they are no longer needed through a call to GIO_delete (). Application shall call GIO_submit () APIs to submit an I/O request to driver. The device independent layer shall construct an I/O packet and submits the packet to the IOM layer to do the I/O operation. When a mini-driver completes its processing, usually in an ISR context, it calls its associated callback function to pass the IO packet back to the upper layer and the

class driver in turn calls the application specified callback for that particular I/O request. The submit/callback function pair handles the passing of IO packets between the application and the McBSP IOM layer of the driver. Before an IO packet is passed back to the device independent layer, the mini-driver must set the completion status field and the data size field in the IO Packet. This status value and size are returned to the application call that initially made the I/O request.

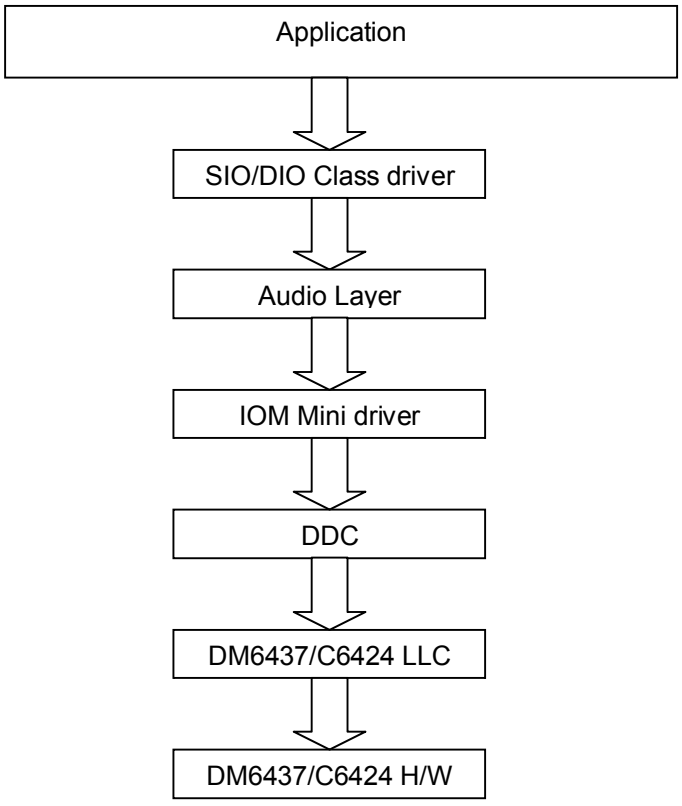
Chapter 4

Porting Guidline

This section describes porting of McBSP driver on different TI platforms.

4.1 Porting Description

The figure below shows MCBSP device driver architecture and changes those are required at the driver layers while porting MCBSP device driver to any other Platform.



Driver Architecture

There will not be any change required in the SIO/DIO Layer, IOM Layer, DDA Layer and DDC layer while porting MCBSP device driver on any TI platform. This Layer will be used as-is.

Audio Layer

Audio layer is designed in such a way that it can be used with both McBSP driver as well as McASP driver. The job of the audio layer is:

1. To perform the pinmux settings specific to the platform
2. To configure external audio codec to work with McBSP driver. Here AIC33 audio codec is used.

While porting of McBSP driver for other platform, above mentioned two things one needs to take care. One needs to perform the pinmux setting as per the new platform and configuration of audio codec based on the audio codec that is being used with McBSP audio driver.

LLC Layer

This layer provides the abstraction to the Driver core on different platforms. This layer is specific to a specific platform. Mainly this layer should be having register overlaying, macro definitions. If not register overlaying is used then this layer will have low-level APIs to communicate with the hardware

This layer should be having an as-is map of the peripheral device registers in the processor's memory map. Peripheral device registers map may differ from one platform to other. This change needs to be incorporated while porting driver code from once platform to another platform at LLC layer.

Chapter 5

McBSP Sample Application

This section describes the example application that is included in the package. This sample application can be run as is for quick demonstration, but the user will benefit most by using these samples as sample source code in developing new applications.

5.1 Introduction

The sample application will perform audio record and playback in a continuous loop. It will record the audio from the line-in port and playback the audio through line-out/headphone-out port.

5.2 Building the Application

The sample application for DM6437 is located in the \pspdrivers\system\DM6437\bios\evmDM6437\audio\sample folder and the sample application for C6424 is located in the \pspdrivers\system\C6424\bios\evm6424\audio\sample. The sample can be rebuilt directly from its project file using Code Composer studio.

5.3 Loading the Application

The sample application is loaded and executed via Code composer studio. It is good idea to reset the board before loading Code Composer.

At the audio layer McBSP driver is configured with following values to perform the record and playback

```
static PSP_mcbspHwCfgRaw    mcbspRawHwCfg = {
                    /**< Serial Port Control Register */
                    0x01C00000u,
                    /**< Receive Control Register */
                    0x000001A0u,
                    /**< Transmit Control Register */
                    0x000001A0u,
                    /**< Sample Rate Generator Register */
                    0x00000000u,
                    /**< Pin Control Register */
                    0x00000000u
                    };
```

In addition to configuring McBSP for record and playback, few other parameters are also configured.

```
static PSP_mcbspDevInitParams mcbsp_deviceparams = {
                    /**< Caching enabled                */
                    TRUE,
                    /**< Internal clock is to be generated  */
```

```
                FALSE,
                 /**< Framesync is to be generated       */
               PSP_McBSP_McBSP_SPIMASTER;
                  /**< Mode in which mcbsp port needs to be created*/
               PSP_OPMODE_DMAINTERRUPT
               /**< Operation mode of driver i.e interrupt or EDMA      */
                 300000
                 /**<Output SPI Master bus frequency */
                 99000000,
               /**< Module Clock Frequency..Valid in Mcbsp SPI Master mode   */
                &mcbspRawHwCfg
                 /**< Pointer to configuration parameters to be used for the McBSP */

};  /* Device parameters structure */
```

5.4 Pragma directives used in the Application

The prdInputBuf and prdOutputBuf are used by the application and need to be cache aligned at 128 bytes.
Compiler Switch “PSP_MCBSPINSPI_ASYNC_MODE_SUPPORT” is used to enable ASYNC MODE of operation of the sample application.

5.5 Modes of Operation.

McBSP can be configured to run in 3 below mentioned modes.

1. McBSP in SPI MASTER mode. (
2. McBSP in SPI SLAVE MODE.
3. McBSP mode.

Configuration of the device in which mode we want to operate should be specified as part of device parameter structure, in the “mcbspRawCfgPtr” as shown below:

SPI Mode:

```
PSP_mcbspHwCfgRaw    mcbspRawHwCfg = {
                          /**< Serial Port Control Register */
                0x00001000u,    /** enable clock stop without delay   */
                          /**< Receive Control Register */
                0x00000000u,    /** Have a zero bit delay (SPI mode), */
                          /**< Transmit Control Register */
                0x00000000u,    /** Have a zero bit delay (SPI mode), */
                          /**< Sample Rate Generator Register */
                0x20080101u,    /** set clkdiv to 1,
                                    set internal clock,
                                    set 8 frame width            */
                          /**< Pin Control Register */
                0x0000000Au     /** set transmit frame polarity     */
                           };
```

McBSP Mode:

```
static PSP_mcbspHwCfgRaw    mcbspRawHwCfg = {
                            /**< Serial Port Control Register */
                            0x01C00000u,
                            /**< Receive Control Register */
                            0x000001A0u,
                            /**< Transmit Control Register */
                            0x000001A0u,
                            /**< Sample Rate Generator Register */
                            0x00000000u,
                            /**< Pin Control Register */
                            0x00000000u
                            };
```

NOTE:

To use the McBSP in SPI mode to communicate between two boards following steps should be followed:

1. Load the Sample application (with approparite Device parameters) using CCS. .
2. Run the Slave application first, which will be waiting for the data from the master.
3. Run the Master application.
4. Repeat the steps 2 and 3 for multiple transfer interations.

Appendix A

DSP/BIOS McBSP Driver References

References

[1] McBSP512 Module Hardware Specifications.

[2] EDMA 3.0 Module Hardware Specifications.

[3] SPRU943_McBSP.pdf - TMS320DM643x DMP Multichannel Buffered Serial Port (McBSP) Interface - User's Guide