Texas Instruments

DSP/BIOS McBSP Device Driver

Architecture Specifications

December 2007

Platform Software Group

Document Version 1.11

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

About This Document

The purpose of this document is to aid the implementation of the McBSP driver for DSP/BIOS Operating System, which will run on DM6437 SoC.

This document mainly illustrates the design of the API's for the IOM layer of the McBSP driver, which has been ported from the DSP/BIOS operating system.

The target audience includes device driver developers from TI as well as consumers of the driver.

Trademarks

Related Documents

- McBSP hardware specifications
- EDMA 3.0 hardware specifications
- spru943a.pdf

Notations

None

Terms and Abbreviations

Term	Description
3PDMAC	Third Party Direct Memory Access Controller.
API	Application Programming Interface.
CSL	Chip Support Layer.
DDK	Device Driver Development Kit.
EDMA	Enhanced Direct Memory Access Controller.
IOM	IO Mini Driver Model.
INTC	Interrupt Controller
IP	Intellectual Property
ISR	Interrupt Service Routine

McBSP	Multi-channel Buffered Serial Port

Revision History

Date	Author	Comments	Version
7th August, 2006	Pratik Joshi	Created	1.0
16th August 2006	Pratik Joshi	Modified to include details of Loopjob API support	1.1
28th August 2006	Pratik Joshi	Driver architecture is changed to comply PSP architecture	1.2
28th September 2006	Ankur Verma	Change the name of the IOCTLS.	1.3
30th November 2006	Pratik Joshi	Modified for the release 0.3.0	1.4
16th January 2007	Pratik Joshi	Modified for the release 0.4.1	1.5
27th January, 2007	Pratik Joshi	Modified for the release 0.5.0	1.6
20th February, 2007	Pratik Joshi	Modified for the release 0.6.0	1.7
23rd April, 2007	Pratik Joshi	Modified for the release 0.7.0	1.8
June 22, 2007	Anuj Aggarwal	Modified for the GA patch release 1.00.01	1.9
June 29, 2007	Amit Chatterjee	Modified Release Version	1.10
July 18, 2007	Maulik Desai	Modified Release Version	1.11

Table of Contents

List of Figures

List of Tables

1 System Context

The DSP/BIOS McBSP device driver presented in this document is situated in the context of DSP/BIOSv5.31.01 Operating System running on the DM6437.

1.1 Hardware

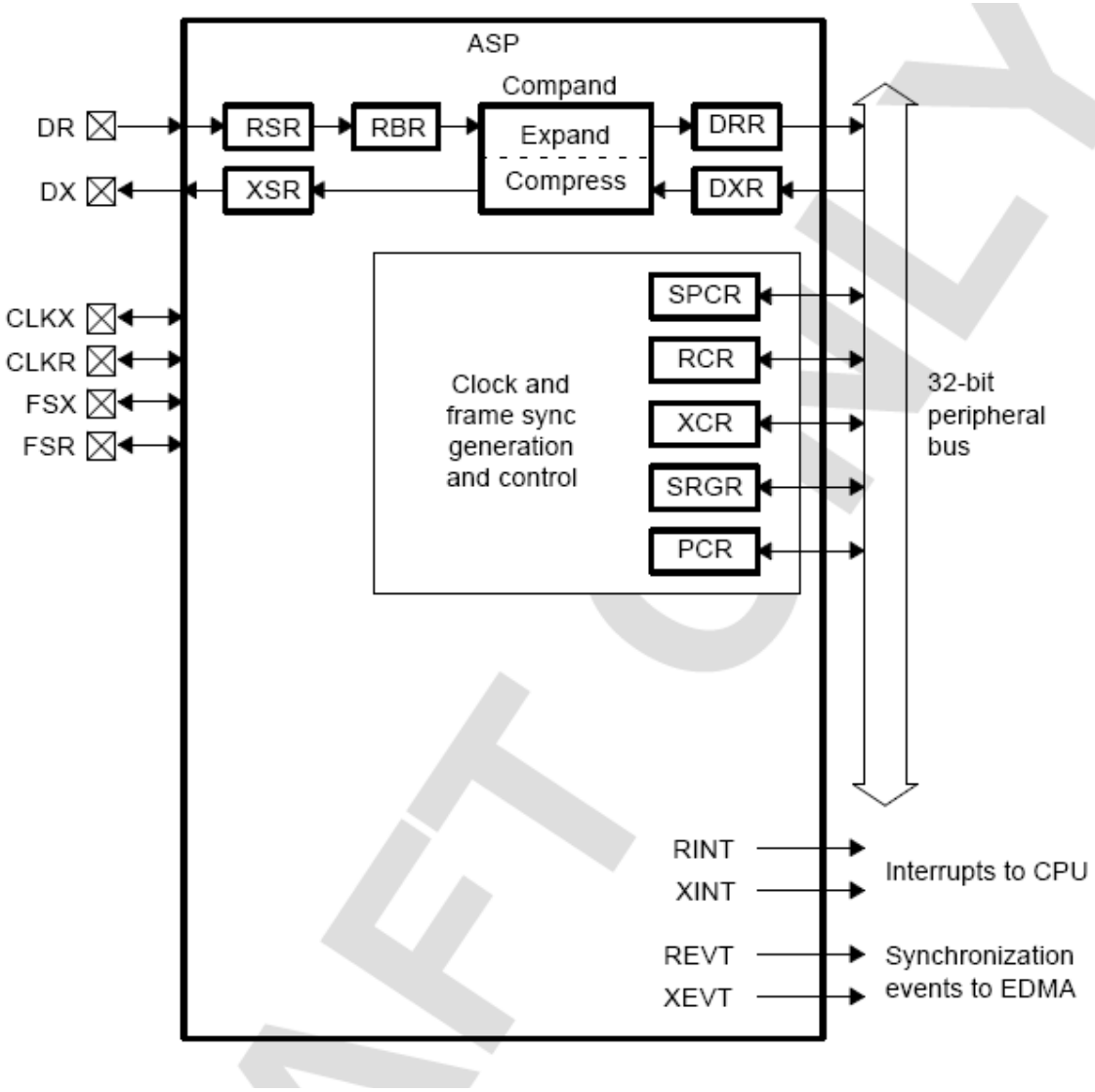

Figure 1: McBSP Internal block diagram

Multi-channel buffered serial ports are configurable, high-speed, full-duplex serial ports that allow direct interface to external communications devices like Audio Codec etc. McBSP has double buffered transmitter section and triple buffered receiver section in addition to the independent framing and clocking for transmitter and receiver sections.

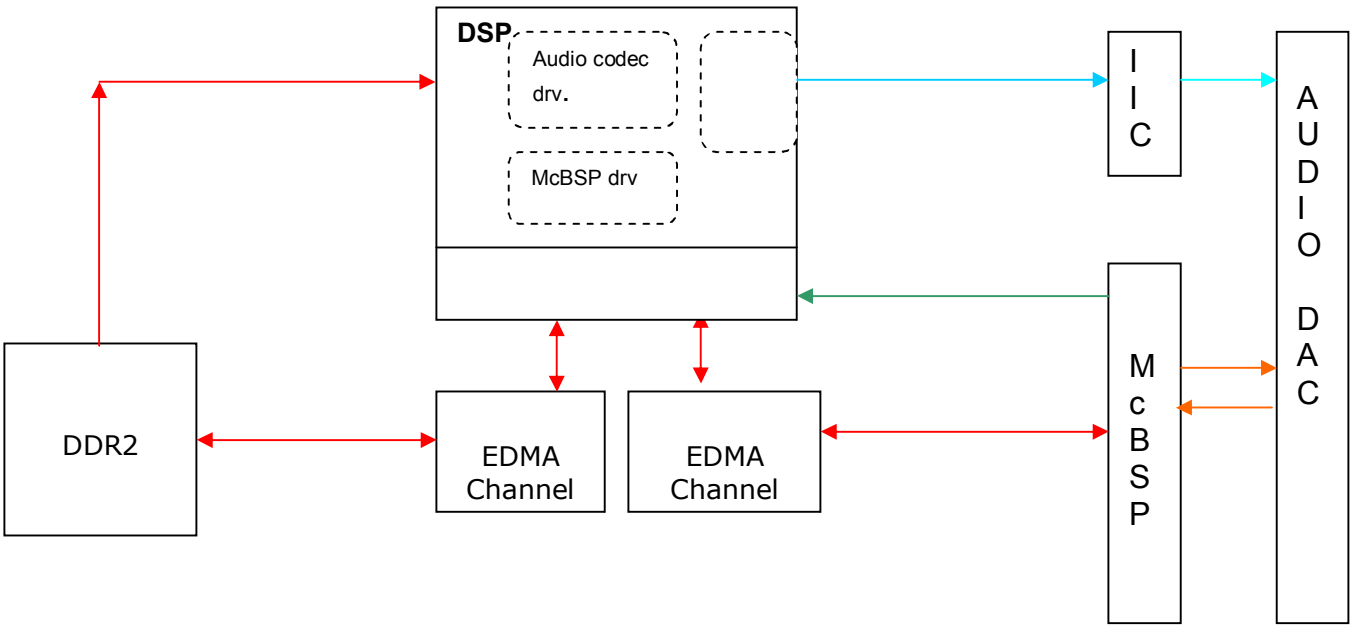

Figure 2: Components involved in the data transfer.

Figure 2. depicts the various components involved in the transfer of audio data when the MCBSP driver runs on the DSP core of DM6437 processor. Audio data is stored in the SDRAM first by the DSP after decoding the Audio data. The main function of the McBSP driver is to program the EDMA channels to move the audio data from SDRAM to the MCBSP interface on every transfer event from the McBSP.

1.2 Software

The document provides an overall understanding of the TI McBSP device driver architecture.

1.2.1 Operating Environment and dependencies

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

1.3 Design Philosophy

Please refer section 1.3 of DM6437_BIOS_PSP_User_Guide.doc for and DM6437 and section 1.3 of C6424_BIOS_PSP_User_Guide.doc for C6424 for Design Philosophy.

2 McBSP Driver Software Architecture

This chapter deals with the overall architecture of DSP/BIOS McBSP device driver, including the device driver partitioning as well as deployment considerations. We'll first examine the system decomposition into functional units and the interfaces presented by these units. Following this, we'll discuss the deployed driver or the dynamic view of the driver where the driver operational scenarios are presented.

2.1 Static View

2.1.1 Functional Partition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities already discussed in section 0. In the following sub-sections, each of these functional sub-components of the device driver is further elaborated.

As per the design philosophy, the McBSP driver shall be split into three layers in order to increase the reusability of the driver. The upper layer called the Device Independent layer responsible for buffer management and application synchronization. The middle layer is called the IOM layer and is specific to the McBSP and EDMA devices, which exposes standard interfaces to the device independent layer. The lower level layer is called Lower level controller constitutes a set of well-defined API that abstracts low-level details of the underlying SoC device so that user can configure, control (start/stop etc.) and have read/write access to peripherals without having to worry about register bit field details.

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.1 of C6424_BIOS_PSP_User_Guide.doc for C6424 for Diagrammatic explanation.

2.1.1.1 LLC Layer

The LLC forms the lower most, h/w specific under-pinning of the TI device driver. The LLC Layer used in the McBSP driver conforms to TI PSP architecture. It consists of two parts:

- **CSL Register Layer**: Please refer section 1.2.2 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.2 of C6424_BIOS_PSP_User_Guide.doc for C6424 for CSLr Layer description.
- **LLC Layer**: Please refer section 1.2.2 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.2 of C6424_BIOS_PSP_User_Guide.doc for C6424 for LLC Layer description.

2.1.1.2 Device Driver Core functionality (DDC)

Please refer section 1.2.3 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.3 of C6424_BIOS_PSP_User_Guide.doc for C6424 for DDC Layer explanation.

2.1.1.3 McBSP driver's IO Mini Layer

The McBSP IO Mini layer exposes a particular set of functions in IOM function table to the upper layer. The MCBSP_IOM_init () function of the McBSP IO mini driver is responsible for returning the pointer to the IOM function table to the upper layer. The functions exported by the McBSP IO Mini layer through the IOM_Fxns table "McBSP_IOM_FXNS" are listed below.

- **mdBindDev**: Called by the application to bind the device to the IO mini layer.
- **mdCreateChan**: Used to create I/O channels.
- **mdDeleteChan**: Used to delete I/O channels.
- **mdSubmitChan**: Used by the upper layer for submitting I/O packets containing the information needed by the mini driver to program the EDMA channels for data transfer.
- **mdContrlChan**:Used to perform device specific control operations.
- **mdUnBindDev**: Used to unbind the IO mini driver from the device.

Please refer section 1.2.4 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.4 of C6424_BIOS_PSP_User_Guide.doc for C6424 for more explanation on IOM Layer.

The following table outlines the basic interfaces published by IOM layer.

Table 1 : Interfaces exposed by the McBSP IOM layer to the Device Independent layer

Mini driver Implemented I/F for Device independent layer	DESCRIPTION
typedef Int (*IOM_TmdBindDev)(Ptr *devp, Int devid, Ptr devParams)	The mdBindDev function is called by the DSP/BIOS after the bios initialization The mdBindDev should typically perform the following actions. ❖ Acquire the Handles for the specified instance of the McBSP on the SoC. ❖ Configure the McBSP device with the specified parameters or default parameters, if there is no external configuration. The default parameters shall match the DSP data format mode of the audio codec.
typedef Int (*IOM_TmdUnBindDev)(Ptr devp)	The mdUnBindDev () is called by the GIO_delete(). The responsibility of the mdUnBindDev () is to free the resources (eg. McBSP, EDMA shadow region) allocated by the mdBindDev () function.
typedef Int (*IOM_TmdControlChan)(Ptr chanp, Uns cmd, Ptr args);	The GIO_control() call of the Class driver Layer invokes the mdControlChan () of the IOM layer. The mdControlChan () calls underlying DDC layer function that implements some of the device control commands (eg Start/Stop, Pause/Resume, Mute-ON/OFF, Reset) useful for the application designers.
typedef Int (*IOM_TmdCreateChan)(Ptr *chanp, Ptr devp, String name, Int mode, Ptr chanParams, IOM_TiomCallback cbFxn, Ptr cbArg);	The mdCreateChan () function is executed in response to the GIO_create() API call by the application. Application has to specify the mode in which the channel has to be created through the “mode” parameter. The McBSP driver supports only two modes of channel creation (input and output) mode for every device instance.

	The mdCreateChan () call underlying DDC layer function that acquires the necessary EDMA channels required for the transfer and it configures the EDMA channels with the default or user specified parameters.
typedef Int (*IOM_TmdDeleteChan)(Ptr chanp);	The mdDeleteChan () is invoked in response to the GIO_delete () API call by the application. It frees all the resources allocated by the mdCreateChan () function.
typedef Int (*IOM_TmdSubmitChan)(Ptr chanp, IOM_Packet *packet);	The mdSubmitChan () is invoked in response to the GIO_read(), GIO_write(), GIO_Flush () or GIO_Abort () API calls with the appropriate channel handle and IOM packet containing the operation to be performed and required parameters needed for programming the EDMA channels.

2.1.1.4 McBSP driver's Device Independent layer

The McBSP driver's Device Independent Layer shall be responsible for Application synchronization and converting I/O request to I/O packets, containing information in an IOM packet structure and buffer management.

The API's exposed by the McBSP driver to the application and their description are listed in the tab.

Table 2: APIs exposed by the Device Independent Layer

Device Independent Layer implemented I/F for application	DESCRIPTION
GIO_Handle GIO_create(String name, Int mode, Int *status, Ptr optArgs, GIO_Attrs *attrs);	Create McBSP driver channel for input and output : This service opens the given instance of the McBSP driver for operation. It internally initializes i/o channels of McBSP.Application has to pass the required device parameters through PSP_mcbspChanParams structure, which is explained in the section. This function is non reentrant.
Int GIO_submit(GIO_Handle gioChan, Uns	Read / Write buffer data from input/output

cmd, Ptr bufp, size_t *psize, GIO_AppCallback *appCallback)	channel: API is invoked for both read and write request. It takes channel handle, buffer pointer, buffer size, command (input/output)_and address of callback function.
Int GIO_read(gioChan, bufp, psize)	Read a buffer of data from input channel: Application has to call GIO_read () API with the handle to the input channel, pointer to the buffer to where the data shall be stored, size of data to read. The size of the frame is configurable from 1 byte to 64K bytes. Internally this function calls GIO_submit with command parameter as IOM_Read
Int GIO_write(gioChan, bufp, psize)	Write a buffer of data to output channel: Application has to call GIO_write() API with the handle to the output channel, pointer to the buffer to where the data is stored, size of data to write. The size of the frame is configurable from 1 byte to 64K bytes. Internally this function calls GIO_submit with command parameter as IOM_Write .
Int GIO_control(GIO_Handle gioChan, Uns cmd, Ptr args);	Device specific control operation: Application can directly use the some of the device specific controls exposed by the appropriate control command as argument. This API shall internally invoke the mdControlChan () function of the IOM layer.
Int GIO_Abort(GIO_Handle gioChan)	Abort the data operation of the specified channel: The GIO_Submit() API is called to abort the operation of the I/O channel. When an Abort packet is submitted to the I/O channel, the driver discards all the pending I/O packets regardless of the mode of the channel (for both input and output). Application is not allowed to submit the transfer requests to the driver when the aborting operation is active. Once the abortion of the channel is

	complete, the application can freshly start submitting the I/O requests.
Int GIO_Flush (GIO_Handle gioChan)	Flush the data in the specified channel: The McBSP driver treats the data flush of the input channel is same as the abort operation and it discards all the pending read requests. In case of output channel, when flush is issued, driver performs all the pending write requests and the application is not allowed to submit the write requests to the driver when the flush is under progress.
Int GIO_delete(GIO_Handle gioChan);	Close the channel from operation: When there is no further operation to perform with the I/O channel, the application can close the channel and there by relinquishing all the resources held by the channel. This function shall internally call mdDeleteChan () function of the IOM layer.

2.2 Dynamic view of the DSP/BIOS McBSP driver

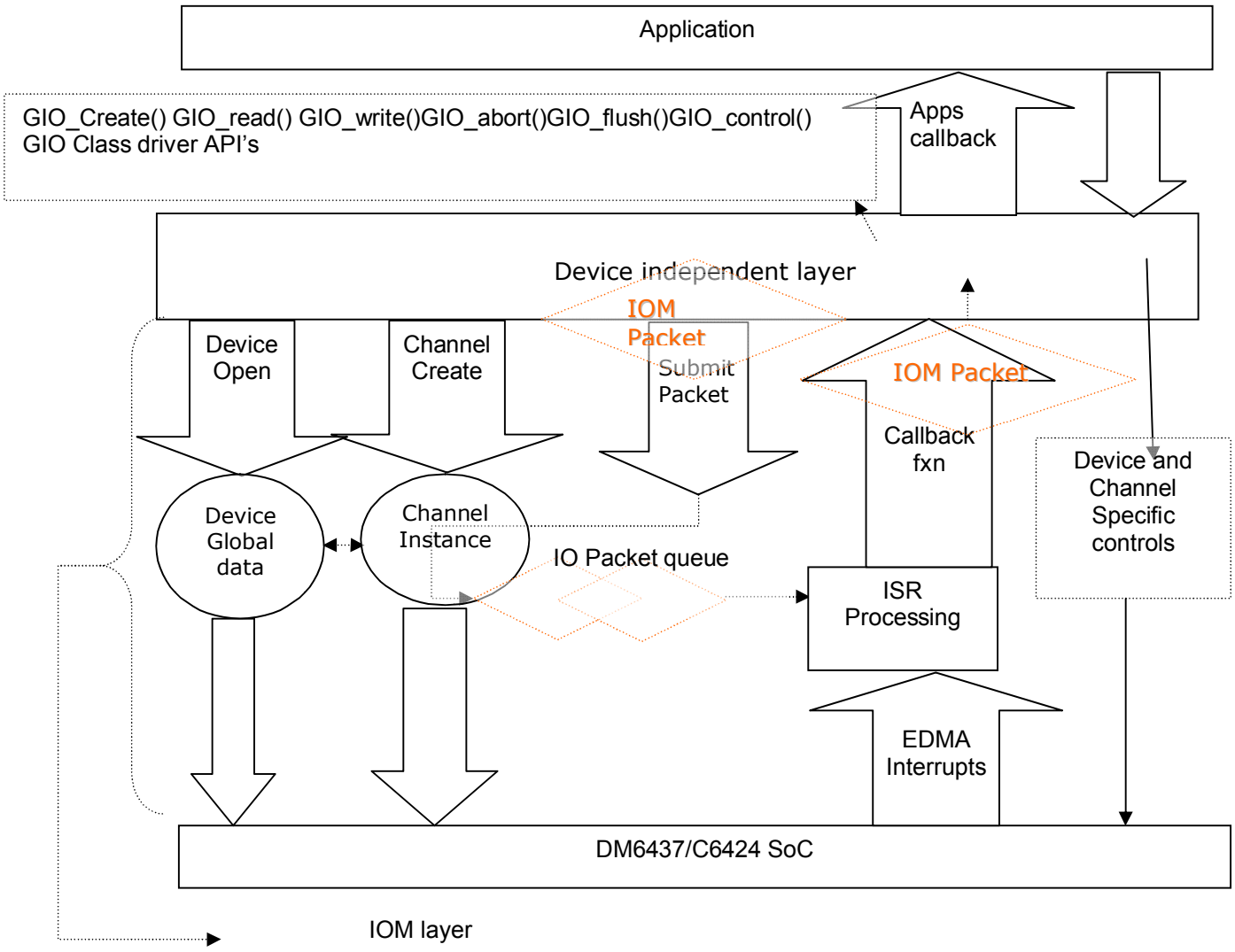

Figure 3: Dynamic view of the McBSP driver

When the bios calls mdBindDev() of mini driver of McBSP driver, the MCBSP_IOM_init () function of the IOM layer is invoked first and is responsible for initializing the device object and channel object structure of the McBSP IOM driver.

Figure 3. shows the flow of data from the application to the driver to the underlying physical device. The IO packet shown in the Figure 3 is standard structure used to submit the I/O requests to the IOM layer of the McBSP driver. It contains pointer to the data buffer, size of the buffer and the status of the request. The mode of the IO channel with which the

packet was issued decides whether it is a read or a write command, not the IO packet command field.

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_close ().

Application shall call GIO_submit () API to submit read/write 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 device independent layer of the McBSP driver and the device independent layer of the 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 upper layer driver, 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.

2.2.1 Driver Open (Driver initialization and Binding)

The McBSP IOM driver initializes the global data used by the McBSP driver. The initialization function for the McBSP driver is not included in the IOM_Fxns table, which is exported by the McBSP driver; instead a separate extern is created for use by the DSP/BIOS. The initialization function is responsible for returning the “IOM” function table structure, which is needed by the device independent layer of the 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.

The binding function (mdBindDev) of the McBSP IOM mini-driver is called by application before using the driver. This function shall typically perform the following actions:

1. Set device defaults and perform setup based on the configured device parameters and optional global device data.

2. Acquire driver resources such as McBSP.
3. Configure the McBSP by default for the following operations.
    - Data output for audio playback
    - Data input for audio recording

- Configure the McBSP in DSP data format mode of the audio codec.
- Configure the McBSP to receive the Frame Sync and bit clock either externally or internally for both receiver and transmitter depending on the device parameter input.

The mdBindDev () of the McBSP IOM driver expects device setup parameters in the "PSP_mcbspDevParams" structure defined in the "psp_mcbsp.h" header file.

The parameters are explained in the following table

Table 3: Device setup parameters table

Device 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.
mcbspRawCfgPtr	Pointer to the LLC configuration structure to be used for the McBSP device.

2.2.2 Channel Creation

The application can create communication channels by calling GIO_create() API which in turn calls McBSP IO mini driver's mdCreateChan function. The application shall call mdCreateChan twice with change in mode parameter to create two logical channels one for input (for audio recording) and one for output (for audio playback). The mdCreateChan function should allocate a channel object and set the fields in the channel object to their initial values as needed. For each channel there will be a channel object and the mode field in the channel object specifies whether this is an input or output channel. McBSP driver acquires the necessary EDMA channels used to transfer of data. Application has to pass the channel parameters in the "**PSP_mcbspChanParams**" structure exposed by the driver.

Table 4: Channel setup parameters table.

Channel 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"	McBSP interrupt number

“wordWidth”	The parameter informs the driver what is the width word (not slot) and this help driver indirectly to decided no. of bytes to be transferred into each serialer for each slot- This is very important parameter - in case of invalid value default value driver will assume is 32
“userLoopJobBuffer”	Buffer to be transferred when the loop job is running it should be noted that this buffer size should be n*userLoopjobLength where n is the no of serialisers configured in the direction of the channel we are creating
“userLoopJobLength”	Number of bytes of the userloopjob buffer for each serialiser. Please note that this is no. of bytes and this should be pre-calcuated properly for word width of slot - Please refer the wordWidth of this structure
“edmaHandle”	Handle to the EDMA Driver
“gblCbk”	callback required when global error occurs - must be callable directly from the ISR context

2.2.3 IO Access

Application invokes GIO_read () and GIO_write () APIs for data transfer using McBSP. These APIs in turn creates and submits 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 command codes currently supported by the McBSP IOM mini-driver are: IOM_READ, IOM_WRITE, IOM_ABORT, and IOM_FLUSH.

- **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 device independent layer. 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 output requests are processed normally and all input requests are discarded. This requires the processing of each IOM_Packet in the original order they were queued up to the channel.

2.2.4 IO Control

McASP IO Mini driver implements device specific control functionality which may useful for any audio codec driver, which internally uses the MABSP IOM driver. Application may invoke the control functionality through a call to GIO_control (). McASP IOM driver supports the following control functionality.

- PSP_CTRL_McBSP_STOP: Stop the data transfer operation.
- PSP_CTRL_McBSP_START: Re-start the data transfer operation.
- PSP_CTRL_McBSP_LOOPBACK: Enable or disable the loopback mode for the McBSP.
- PSP_CTRL_McBSP_SRGR_START: Start the McBSP sample rate generator to generate the bit clock.
- PSP_CTRL_McBSP_SRGR_STOP: Stop the McBSP sample rate generator.
- PSP_CTRL_McBSP_FSGR_START: Start the McBSP frame sync generator to generate the word clock.
- PSP_CTRL_McBSP_FSGR_STOP: Stop the McBSP frame sync generator.
- PSP_CTRL_McBSP_PAUSE: Pause the playback operation.
- PSP_CTRL_McBSP_RESUME: Resumes the playback operation from the paused state.
- PSP_CTRL_McBSP_MUTE_ON: Mute the playback operation.
- PSP_CTRL_McBSP_MUTE_OFF: Take out the playback operation from the muted state.
- PSP_CTRL_McBSP_CHAN_RESET: Resets the I/O channel and re-configure the channel with the default configuration.
- PSP_CTRL_McBSP_DEVICE_RESET: Reset the entire the device and re-configure the device.
- PSP_CTRL_McBSP_CNG_ADDR: Change the data or output address of the EDMA channel.

- PSP_CTRL_McBSP_CONFIG_DATA: Configure the data for the McBSP transmitter and receiver sections.
- PSP_CTRL_McBSP_SET_CLKMODE: Sets the bit-clock mode (internal or external) for the McBSP transmitter or receiver sections depending the mode of the channel.
- PSP_CTRL_McBSP_SET_FRMSYNCMODE: Sets the frame synch mode (external or internal) for the McBSP transmitter or receiver sections.
- PSP_CTRL_McBSP_CONFIG_SRGR: Configure the sample rate generator to generate the bit clock and frame synchronization signals at the specified rate. The configuration structure used to configure the McBSP Sample Rate Generator is defines as follows.

```
Typedef struct PSP_srgConfig {
        Uint16 srgInputClkMode; /* Source for the
                                   Sample rate generator */
        Uint16 bclkRate;    /* Set Bit clock rate - Divide
                               down value of SRGR input clock */
        Uint16 srgFrmPulseWidth; /* Set the Frame Sync
                                    Pulse width in terms of clock ticks */
        Uint16   srgFrmPeriod;    /* Set the frame sync
                                     signal period in terms of clock ticks */
        Uint16 srgClkPolarity; /* Set Frame sync and Bit
                                  clock polarity */
} PSP_srgConfig;
```

- PSP_CTRL_McBSP_SET_BCLK_POL: Sets the polarity of the bit clock (Rising or falling edge) when the bit is generated by the McBSP sample rate generator
- PSP_CTRL_McBSP_SET_FRMSYNC_POL: Sets the frame synchronization polarity (Active high or active low) when the frame synch signal is generated by the McBSP sample rate generator.
- PSP_CTRL_McBSP_RECEIVE_SYNCERR_INT_ENABLE: Enable the interrupt for receive

- PSP_CTRL_McBSP_XMIT_SYNCERR_INT_ENABLE: Enable the interrupt for transmit PSP_CTRL_McBSP_MODIFY_LOOPJOB: Issues a loopjob operation. When issued with non NULL prdCallback driver will enter in to loop job operation and when issued with prdCallback equal to NULL driver stops loop job and links loop parameter to zero data buffer.

2.2.5 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.

2.2.6 Driver Close

The "mdUnBindDev ()" shall free resources allocated by the "mdBindDev ()" function. If successful, "mdUnBindDev ()" function should return IOM_COMPLETED. If unsuccessful, this function should return a negative error code.

2.2.7 Asynchronous IO Mechanism

The McBSP IOM driver supports asynchronous IO mechanism. In Asynchronous IO mechanism multiple IO requests can be submitted in one shot without causing the thread to block while waiting for resources. Application can submit multiple I/O requests using the GIO_read() or GIO_write () APIs and then callback function that was specified during the transfer request submission shall be called as a result of transfer completion by the driver for every transfer. The driver queues the IOM packets submitted internally to support the asynchronous I/O.