SOFTWARE ARCHITECTURE TEMPLATE

McBSP

Driver Design Document

Rev No	Author(s)	Revision History	Date	Approval(s)
0.1	Imtiaz SMA	Created the document	30 April 09	

TABLE OF CONTENTS

1 Introduction 5
1.1 Purpose & Scope 5
1.2 Terms & Abbreviations 5
1.3 References 5
1.4 Overview 6
1.4.1 Hardware Overview 6
1.4.2 Software Overview 7
2 Requirements 8
2.1 Assumptions 10
2.2 Constraints 10
3 Design Description 10
3.1 Static View 10
3.2 Functional Partition 10
3.2.1 IOM Interface 10
3.3 Dynamic view 12
3.3.1 Driver Creation (Driver Initialization and Binding) 13
3.3.2 Channel Creation 17
3.3.3 IO submit 19
3.3.4 Control Commands 20
3.3.5 Channel deletion 24
3.3.6 Driver unbinding/deletion. 25
3.3.7 Asynchronous IO Mechanism 26
3.4 Constants & Enumerations 27
3.4.1 Mcbsp_TXEVENTQUE 27
3.4.2 Mcbsp_RXEVENTQUE 27
3.4.3 Mcbsp_OpMode 28
3.4.4 Mcbsp_OperatingMode 28
3.4.5 Mcbsp_BufferFormat 29
3.5 Data Structures 30
3.5.1 Driver Instance Object 30
3.5.2 Channel Object 32
3.5.3 Mcbsp_Params 36
3.5.4 Mcbsp_ChanParams 37
3.5.5 Mcbsp_srgConfig 39
3.6 API Definition 42
3.6.1 Mcbsp_init 42

4 Decision Analysis & Resolution 43
4.1 DAR Criteria 43
4.1.1 Alternative 1 43
4.1.2 Alternative 2 43
4.2 Decision 43
5 Revision History 43

TABLE OF FIGURES

Figure 1 McBSP Hardware Overview 6
Figure 2 McBSP Software Overview 7
Figure 3 Dynamic view of the McBSP driver 12
Figure 4 Driver Initialization 14
Figure 5 Driver Instance Binding 16
Figure 6 McBSP channel create command FlowIO Control 18
Figure 7 McBSP Control command Flow 23
Figure 8 Driver deletion 25

1 Introduction

This document describes the McBSP DSP/BIOS device driver. The McBSP driver conforms to the IOM driver model specified by the DSP/BIOS operating system. This document explains the design of the McBSP driver. Also the data types, data structures and application programming interfaces provided by the Mcbsp driver are explained in detail.

1.1 Purpose & Scope

This document explains the McBSP driver design in the context of the DSP/BIOS operating system. It explains the various programming interfaces provided by the driver. Please note that the McBSP driver design discussed here is applicable to the C6748/OMAPL138 SoCs. This document does not explain how to use the Mcbsp device driver, for usage instructions please refer to the BIOSPSP user guide that is available along with the driver.

1.2 Terms & Abbreviations

Term	Description
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

1.3 References

- EDMA3 User guide – SPRFUFL1.pdf
- McBSP user guide
- DSP/BIOS reference Documents.

1.4 Overview

The DSP/BIOS McBSP device driver presented in this document is situated in the context of DSP/BIOS Operating System running on the OMAPL138.The following sub sections explains in detail the hardware and the software context of the McBSP driver.

1.4.1 Hardware Overview

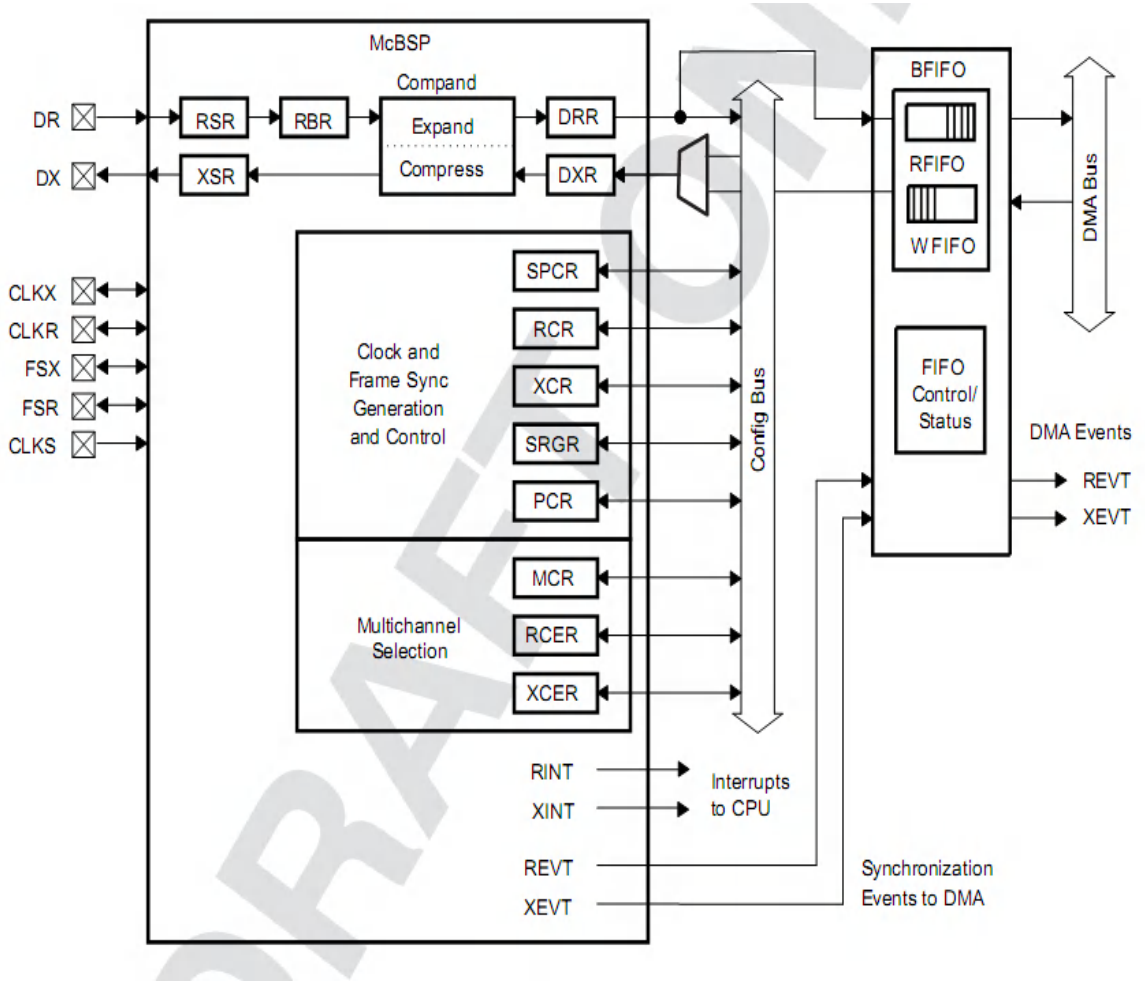

Figure 1 McBSP Hardware Overview

The Figure 1 McBSP Hardware Overview above shows the hardware overview of the McBSP controller. The McBSP contains the McBSP main Controller, a FIFO interface and also the EDMA controller interface.

The McBSP controller provides the hardware registers that allows the McBSP to be configured for the serial data transfer.

The McBSP Buffer FIFO (BFIFO) provides additional data buffering for the McBSP. The time it takes the CPU or DMA controller to respond to DMA requests from the McBSP may vary. The additional buffering provided by the BFIFO allows greater tolerance to such variations.

The EDMA controller interface allows the EDMA to be programmed to move the serial data between the Mcbsp and the DSP. There are dedicated EDMA channels available for the McBSP to transfer and receive data. (The software also uses two additional spare PARAM sets for PING PONG operation for providing additional buffering required especially when transferring audio data as the tolerance to delays is very less during the audio data transfer).

1.4.2 Software Overview

This section describes in detail the McBSP device driver architecture. The McBSP drive described here conforms to the DSP/BIOS IOM driver model.

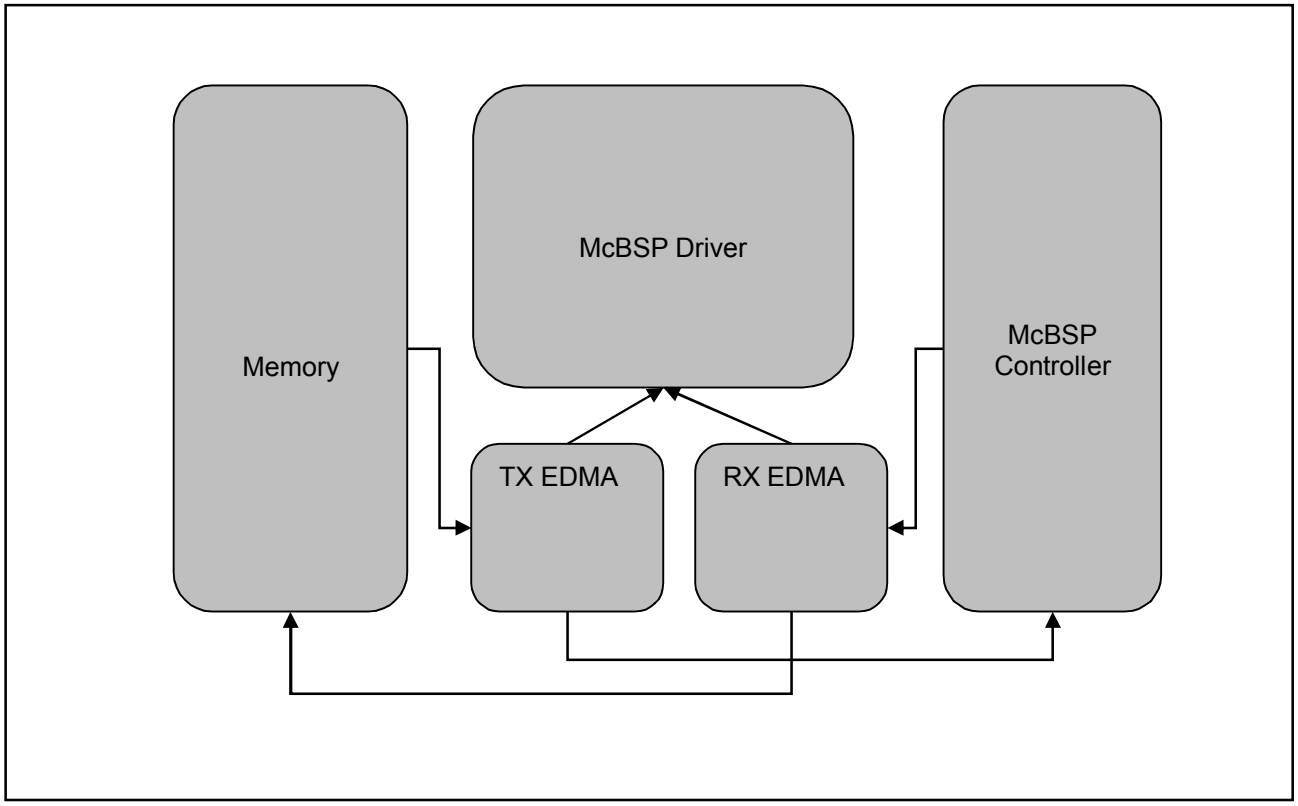

Figure 2 McBSP Software Overview

Figure 2 McBSP Software Overview depicts the various components involved in the transfer of audio data when the MCBSP driver runs on the DSP core of OMAPL138 processor. Serial Data is stored in the memory 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.

2 Requirements

This section in brief lists the most important functional requirements of the McBSP driver.

SR595 Sample and slot size Support

Driver shall support sample sizes of 8, 16 & 32bits and slot sizes of 8, 12, 16, 20, 24, 28, and 32bits.

SR576 Configuration at driver initialization

Following items shall be configurable at initialization: Setting clock speed for transmission and reception, Configure to use internal or external clock. Configure polarity of serial bit clock and frame sync signal Configure Left/right Alignment of Word in a slot Configure Bit order LSB/MSB first of the slot Enable/disable digital loopback mode

SR577 Polled mode of operation

The driver shall not support the polled mode of operation.

SR578 Mode of Operation

Driver shall support only DMA mode of operation.

SR579 Multi Instance Support

MCBSP driver shall handle multiple instances of MCBSP peripheral simultaneously.

SR580 TDM support

The driver would support TDM (and I2S)

SR596 Cancel IO

The driver shall support canceling of all pending IO operations

SR583 Cache coherency

If the data buffers that are submitted to the driver are in cacheable memory, the driver shall take care of cleaning and flushing the cache accordingly. The buffers provided to the driver, shall be properly aligned (to 128 byte boundary). In addition the buffers should also be of a size multiple of the DMA used (typically a multiple of 128bytes).

SR584 Runtime control commands for Audio mode

Start / Stop Mute on/off Pause Resume

SR585 Async mode of operation

This driver shall operate async mode of operation between application and driver, to enable streaming data transfer

SR586 Dummy transmit support

When there is no I/O packet available for transmission, the driver shall transmit a default pattern either provided by the application buffer or a driver provided buffer with zero data (size of this buffer and source of the buffer shall be configurable during init time) repeatedly. Once the driver receives new I/O packet to transmit, it shall start transmitting with newly received data. This feature shall be enabled/disabled during init time.

SR587 Hardware event callback

The error conditions like overrun/under run are indicated to the application, through registered (by application) callback functions.

SR594 No infinite loop during clock initialization

In the driver create path, for clock initialization an appropriate time out should be provided instead of a infinite loop.

SR582 Dynamic start/stop of serial port

The driver shall support dynamically starting and stopping of the serial port (both in RX and TX directions independently for normal usage scenario. The driver should provide an IOCTL to start and stop the serial port.

SR588 Mute (On/ Off)

Data transfer to McBSP is working normally but 0 value should be transferred instead of real data when mute is on (The intention is to send a buffer of known values. This buffer is different from the buffer given by the application and is owned by the McBSP Driver.)

SR592 Audio driver sample application

At this point of time it is not clear about EVM and connectivity and only AIC3106 sample application is assumed.

SR593 Edma transfer channel

Application should be able to configure (through channel params configuration) provide EDMA TC/queue for each channel (receive and transmit)

SR590 Error handling

The driver shall support error notification for the following errors to the application Receive buffer overrun Transmit buffer under-run

2.1 Assumptions

<TBD>

2.2 Constraints

<TBD>

3 Design Description

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.

3.1 Static View

3.2 Functional Partition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities it is expected to perform. 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 a single layer driver and coupled tightly with the DSP/BIOS operating system and the DSP/BIOS EDMA3 driver. The driver is fully compliant with the IOM driver model of the DSP/BIOS operating system.

3.2.1 IOM Interface

A driver which conforms to the IOM driver model exposes a well define set of interfaces

- Driver initialization function.
- IOM Function pointer table.

Hence the McBSP driver (because of the virtue of its IOM driver model compliance) exposes the following interfaces.

- Mcbsp_init()
- Mcbsp_IOMFXNS

The Mcbsp_init () is a startup function that needs to called by the user (application) to initialize all the data structures of the Mcbsp driver. This function also initializes all the instance specific information for the McBSP instance like the Base address, interrupt number etc.

Note: The working of the Mcbsp driver will be affected if this function is not called by the application prior to accessing the McBSP driver APIs.

The McBSP driver exposes a IOM function pointer table which contains the various APIs provided by the McBSP driver. The functions that need to be supported by an IOM driver are explained below.

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

Function	Description
mcbspMdBindDev	The mdBindDev function is called by the DSP/BIOS after the bios initialization The mdBindDev should typically perform the following actions. ❖ Acquire the device Handle 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 be specified to match the DSP data format mode of the audio codec).
mcbspMdUnBindDev	The mdUnBindDev function is called to destroy an instance of the Mcbsp driver. ❖ It will unroll all the changes done during the bind operation and free all the resources allocated to the McBSP.
mcbspMdControlChan	The mdControlChan function is used to issue a control command to the McBSP driver. Please refer to the list of control commands supported by the McBSP driver. ❖ Typical commands supported are PAUSE, RESUME, STOP, START etc.
mcbspMdCreateChan	The mdCreateChan () function is executed in response to the SIO_create() or 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 required EDMA channel and spare PARAM sets are acquired and configured. ❖ The required TX or RX sections (clocks, SRGR, frame sync etc.) are setup.
mcbspMdDeleteChan	The mdDeleteChan () is invoked in response to the GIO_delete () or SIO_delete API call by the application. ❖ It frees all the resources allocated during the creation of the channel.
mcbspMdSubmitChan	The mdSubmitChan () is invoked in response to the GIO or SIO layer provided read write 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.

3.3 Dynamic view

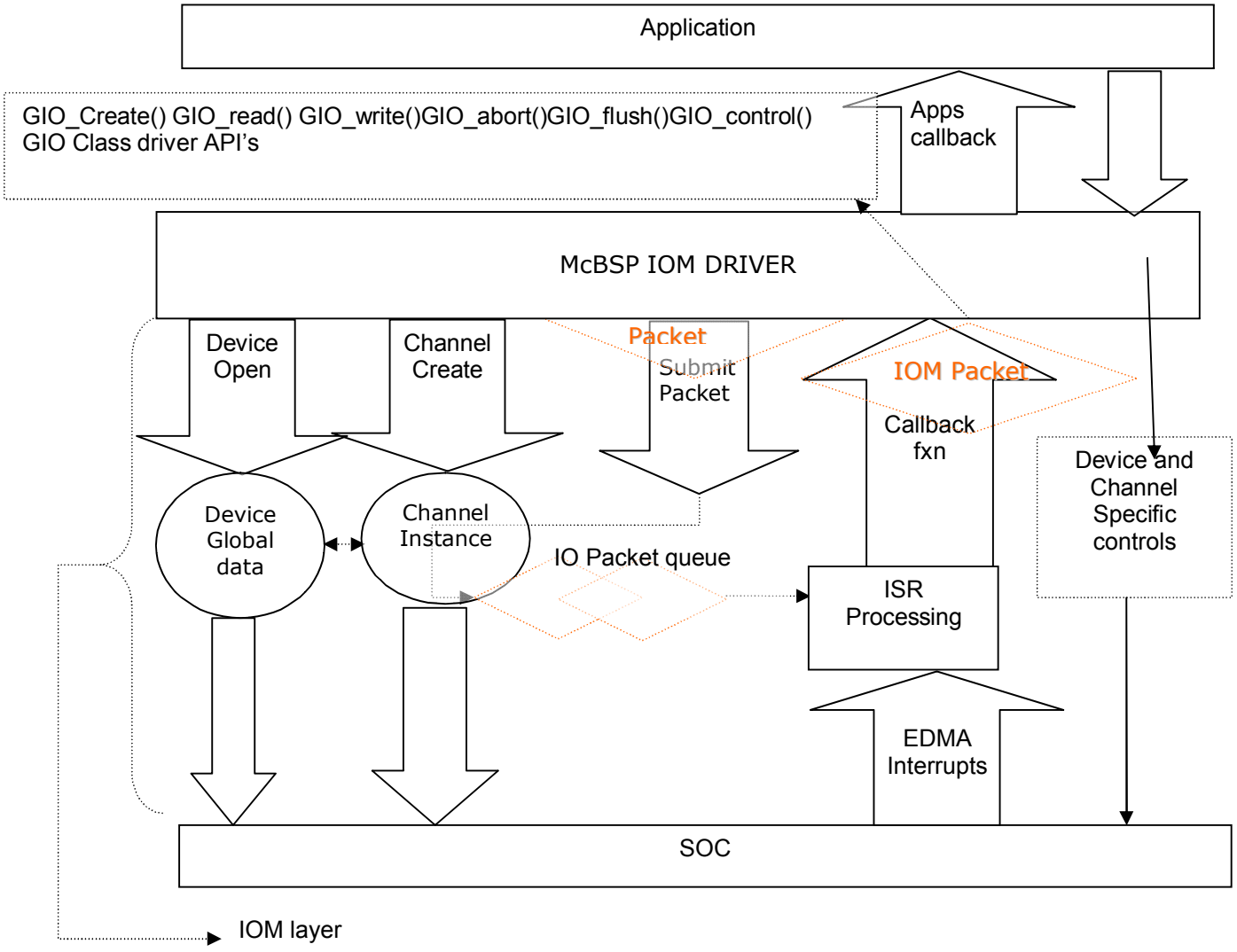

Figure 3 Dynamic view of the McBSP driver

The Mcbsp_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. After that the driver is created by the mcbspMdBindDev () of mini driver of McBSP driver.

The figure above shows the flow of data from the application to the driver to the underlying physical device. The IO packet shown in the figure 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.

Before data communication between an application and a device can begin, a channel instance handle must be obtained by 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 communicate 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 () 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 registered 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.

3.3.1 Driver Creation (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 initialization of the following instance specific information

- Base address for the instance
- FIFO address for the instance
- TX and RX CPU event numbers
- TX and RX EDMA event numbers
- Module clock value

The function also sets the "inUse" field of the Mcbsp instance module object to FALSE so that the instance can be used by an application which will create it. It will also initialize all the module level global variables.

LOOP JOB ENABLED MODE

In loop job enabled mode the LOOP JOB buffers and the mute buffers are initialized.

NON LOOP JOB ENBALED MODE

Non loop job mode does not have any LOOP Job buffers hence only the mute Buffers are initialized.

Please refer to the figure below for the typical control flow during the initialization of the driver. Refer to the section 3.6 for the API reference for the initialization function.

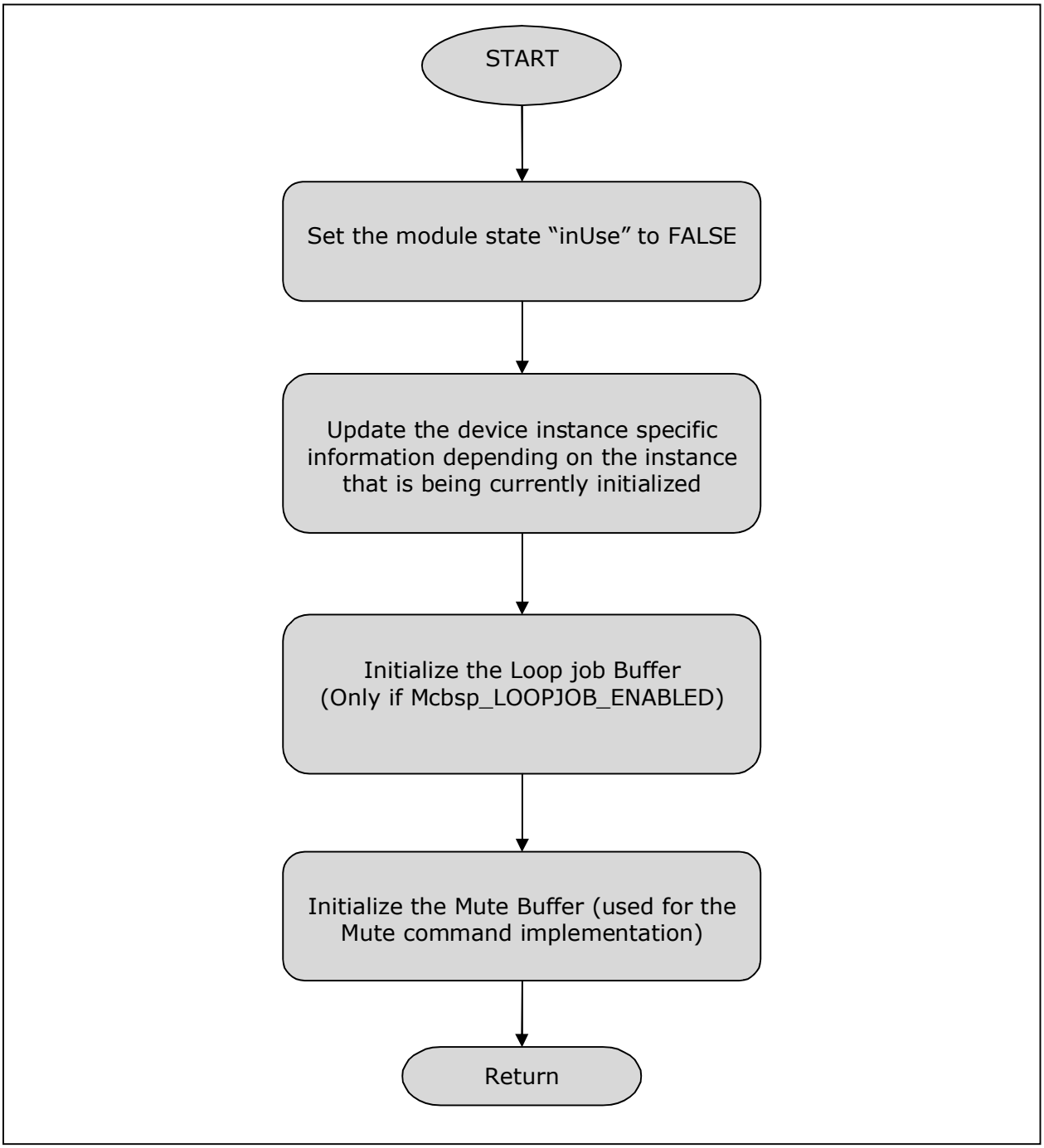

Figure 4 Driver Initialization

The binding function (mcbspMdBindDev) of the McBSP IOM mini-driver is called by in case of a static or dynamic creation of the driver. In case of dynamic creation application will call DEV_createDevice () API to create the device instance otherwise the instance could be created statically through a tcf file. Each driver instance corresponds to one hardware instance of the McBSP. This function shall typically perform the following actions:

- Check if the instance being created is already in use by checking the Module variable "isUse".
- Update the instance object with the use supplied parameters.
- Initialize all the channel objects with default information.
- Initialize the queues used to hold the pending packets and currently executing packet(floating queue).
- Configure the McBSP hardware with the user supplied "raw" parameters or default parameters (if user has not supplied the same).
- Configure the McBSP to receive the Frame Sync and bit clock either externally or internally for both receiver and transmitter depending on the user supplied parameters.
- Return the device handle.

NON LOOP JOB MODE

In case that the McBSP driver is compiled for non loop job mode the following additional actions are performed by the driver

- The "SWI" required for the handling the completion of the last packet in the driver is created.

Note: The Driver binding operation expects the following parameters

1. Pointer to hold the device handle.
2. Instance number of the instance being created.
3. Pointer to the user provided device parameter structure required for the creation of the device instance.

The user provided device parameter structure will be of type "**Mcbsp_Params**". Refer to Driver Instance Object

This structure is the Mcbsp driver's internal data structure. This data structure is used by the driver to hold the information specific to the instance. There will be one unique instance object for every instance of the McBSP controller supported by the driver.

Definition

```
typedef struct Mcbsp_Object_t
{
    Int32                       instNum;
    Mcbsp_DriverState           devState;
    Mcbsp_OperatingMode         mode;
    Mcbsp_OpMode                opMode;
```

```
    Bool                    enablecache;
    Mcbsp_HwInfo            hwInfo;
    Uint32                  mcbspSpiFreq;
    Bool                    stopSmFsXmt;
    Bool                    stopSmFsRcv;
    Mcbsp_ChannelObj        xmtObj;
    Mcbsp_ChannelObj        rcvObj;
    Mcbsp_srgConfig         srgrConfig;
    SWI_Handle              swiHandle;
    Bool                    txSrgEnable;
    Bool                    rxSrgEnable;
    Bool                    srgConfigured;
    volatile Bool           srgEnabled;
    Bool                    txFsgEnable;
    Bool                    rxFsgEnable;
    Bool                    fsgConfigured;
    volatile Bool           fsgEnabled;
    Uint32                  retryCount;
} Mcbsp_Object;
```

Fields

instNum	Instance number of the McBSP.
devState	Current state of the driver (Created/Deleted).
Mode	Operating mode of the McBSP (Mcbsp, SPI master Mode, SPI slave mode).
opMode	Mode of operation of the driver(POLLED/INTERRUPT/DMA)
enableCache	Whether the driver should take care of cache cleaning operations for the buffers submitted by the application
hwInfo	Structure holding the hardware information related to the instance (e.g. interrupt numbers, base address etc).
mcbspSpiFreq	Frequency of operation of the Mcbsp in the SPI mode.
stopSmFsXmt	State of transmit state machine. (TRUE = stopped, FALSE = running).
stopSmFsRcv	State of receive state machine. (TRUE = stopped, FALSE = running).
xmtObj	Transmit channel object
rcvObj	Receive channel object

srgrConfig	Sample rate generator configurations supplied by the user.
txSrgEnable	Variable to indicate if the sample rate generator is required by the TX section.
rxSrgEnable	Variable to indicate if the sample rate generator is required by the RX section.
srgConfigured	Variable to indicate if the sample rate generator is configured or not.
srgEnabled	Variable to indicate if the sample rate generator is running.
txFsgEnable	Variable to indicate if the frame sync generator is required by the TX section.
rxSrgEnable	Variable to indicate if the frame sync generator is required by the RX section.
fsgEnabled	Variable to indicate if the frame sync generator is running.
retryCount	Retry count to be used by the driver when waiting in indefinite loops. (e.g. waiting for the TX to get empty etc).

Comments

1. The Mcbsp Driver works only in the EDMA mode of operation.
2. SPI mode is supported only if the underlying hardware supports it.
3. One instance object represents one instance of the driver.

Constraints

None

See Also

Mcbsp_ChannelObj

3.3.2 Channel Object

This structure is the Mcbsp driver's internal data structure. This data structure is used by the driver to hold the information specific to the channel. There will be at most two channels supported per instance(one for TX and one for RX).it is used to maintain the information pertaining to the channel like the current channel state, callback function etc. This structure is initialized by mdCreateChan and a pointer to this is passed down to all other channel related functions. Lifetime of the data structure is from its creation by mdCreateChan till it is invalidated by mdDeleteChan.

Definition

```
typedef struct Mcbsp_ChannelObj_t
{
    Uint16                  mode;
    Mcbsp_DriverState       chanState;
    Ptr                     devHandle;
```

```
    IOM_TiomCallback        cbFxn;
    Arg                     cbArg;
    Ptr                     edmaHandle;
    Uint32                  edmaEventQue;
    EDMA3_RM_TccCallback    edmaCallback;
    Uint32                  xferChan;
    Uint32                  tcc;
    Uint32                  pramTbl[Mcbsp_MAXLINKCNT];
    Uint32                  pramTblAddr[Mcbsp_MAXLINKCNT];
    QUE_Obj                 queuePendingList;
    QUE_Obj                 queueFloatingList;
    IOM_Packet              *tempPacket;
    IOM_Packet              *dataPacket;
    Uint32                  submitCount;
    Mcbsp_BufferFormat      dataFormat;
    volatile Bool           nextFlag;
    volatile Bool           bMuteON;
    volatile Bool           paused;
    volatile Bool           flush;
    volatile Bool           isTempPacketValid;
    Bool                    enableHwFifo;
    Mcbsp_GblErrCallback    gblErrCbk;
    Uint32                  userDataBufferSize;
    Ptr                     loopJobBuffer;
    Uint16                  loopJobLength;
    Uint32                  nextLinkParamSetToBeUpdated;
    volatile Bool           loopjobUpdatedinParamset;
    Uint16                  roundedWordWidth;
    Uint16                  currentDataSize;
    Uint32                  rxBytesIndex;
    Uint32                  txBytesIndex;
    Mcbsp_DataConfig        chanConfig;
    Mcbsp_ClkSetup          clkSetup;
    Mcbsp_McrSetup          multiChanCtrl;
    Uint32                  chanEnableMask[4];
    Bool                    userLoopJob;
}Mcbsp_ChannelObj;
```

Fields

mode	Current operating mode of the channel (INPUT/OUTPUT).
chanState	Current state of the channel (opened/closed).
devHandle	Pointer to the instance object.
cbFxn	Callback function pointer
cbArg	Callback function argument
edmaHandle	Pointer to the EDMA handle given by the application.
edmaEventQue	EDMA event queue to be used by this channel.
edmaCallback	EDMA callback function pointer.
xferChan	The EDMA transfer channel to be used.
tcc	Transfer completion code to be used in case of EDMA mode.
pramTbl	Value of the two spare PARAM sets issued by the EDMA driver.
pramTblAddr	Address of the two spare paramsets.
queuePendingList	Queue for holding the pending packets.
queueFloatingList	Queue for Holding the currently executing packets.
tempPacket	Temporary place holder for the currently completed packet.
dataPacket	pointer to hold the IOM packet
submitCount	Total number of packets held in the driver for this channel
dataFormat	The Format in which the McBSP data is arranged in the buffer.
nextFlag	Flag used in stopping the McBSP state machines.
bMuteON	Flag to indicate if the mute is ON.
paused	Flag to indicate if the channel is paused.
flush	Flag to indicate if the flush command is Issues to the driver.
isTempPacketValid	Flag to indicate if the "tempPacket" is holding a valid packet.
enableHwFifo	Flag to indicate if the Hardware FIFO is to be enabled for this channel (RX/TX).

gblErrCbk	Application registered callback function to be called in case of an error.
userDataBufferSize	Size of the user supplied buffer.
loopJobBuffer	Loop job buffer to be used when the driver does not have any more packets for the IO
loopJobLength	Length of the loop job buffer.
userLoopJobLength	User specified loop job's length.
nextLinkParamSetToBeUpdated	Variable to indicate which of the spare paramset is to be updated next.
loopjobUpdatedinParamset	Variable to indicate if the loop job is loaded in to the paramset.
roundedWordWidth	The actual word width to be transferred per sync event.
currentDataSize	The size of the current data packet
rxBytesIndex	Number of RX bytes transferred (Only supported in SPI mode).
txBytesIndex	Number of TX bytes transferred (Only supported in SPI mode).
chanConfig	Channel configuration required for the configuring of the channel.
clkSetup	Clock setup to be used for this channel.
multiChanCtrl	Multiple channel selection settings.
chanEnableMask	Mask for the channels to be enabled
userLoopJob	Variable to indicate if the user loop job is used or internal driver loop job buffer.

Comments

1. Only 2 channels are supported per instance
2. SPI mode is supported only if the underlying hardware supports it.

Constraints

None

See Also

Mcbsp_Object

Mcbsp_Params section for more details.

Please refer to the Figure below for the control flow in the driver during the Bind operation.

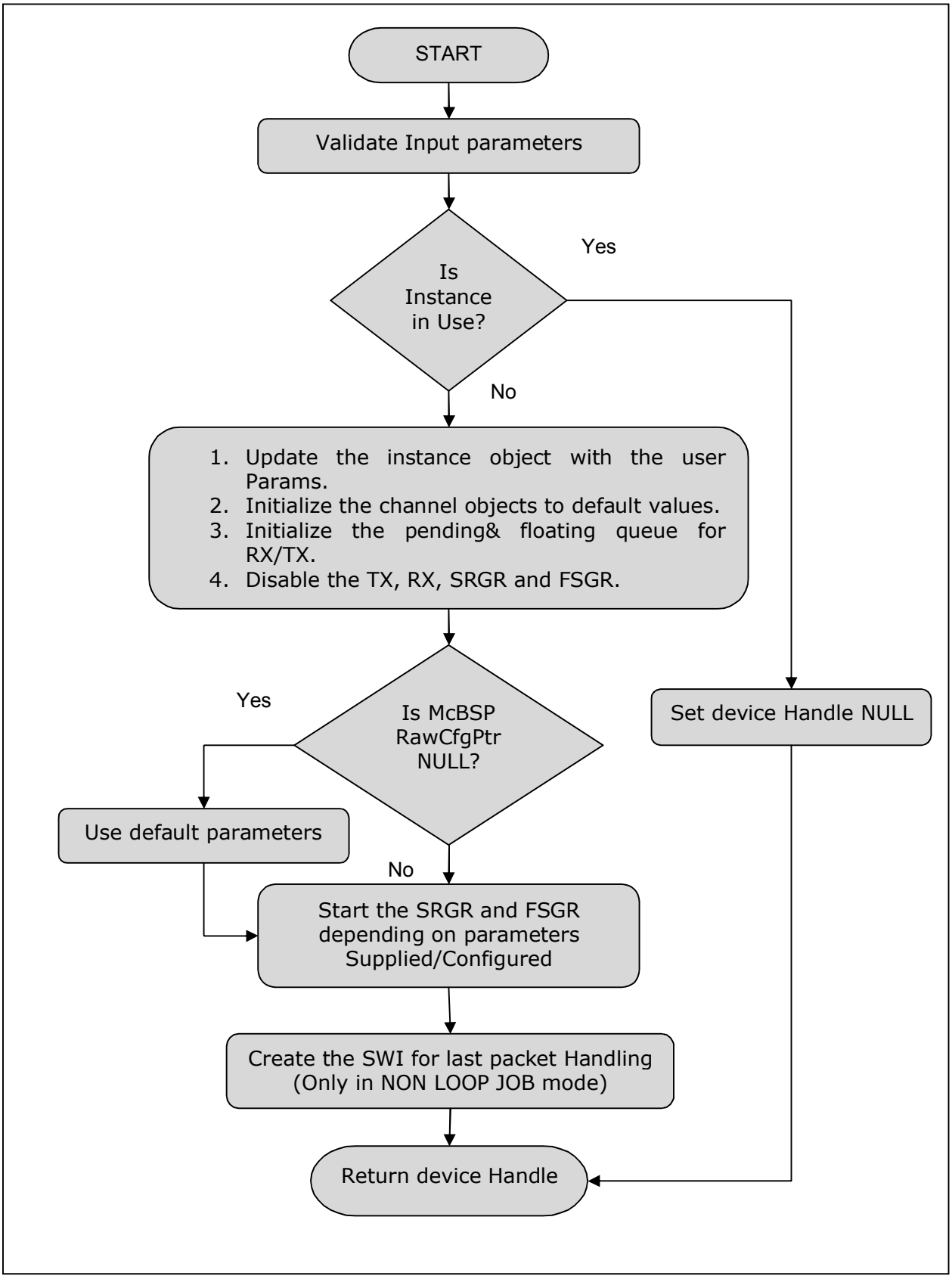

Figure 5 Driver Instance Binding

3.3.3 Channel Creation

The application once it has created the device instance, needs to create a communication channel for transactions with the underlying hardware. As such a channel is a logical communication interface between the driver and the application. An application can create as many channels as it requires, limited only by the MAX number of channels the Device Driver allows.

The McBSP IOM driver allows at most two channels to be created. They are

1. A channel for Transmission.(IOM_OUTPUT)
2. A channel for Reception. (IOM_INPUT)

The application can create a communication channel by calling GIO_create ()/SIO_create () API which in turn calls McBSP IO mini driver's mcbspMdCreateChan () function. The application shall call mcbspMdCreateChan with the appropriate "mode" (IOM_INPUT or IOM_OUTPUT) parameter for the type of the channel to be created.

One logical channel (created with mode IOM_OUTPUT) will be used for transmission of data (e.g. audio playback or data transmission) whereas the second channel (created with mode IOM_INPUT) will be used for receiving data (e.g. audio recording or data reception). The user can supply the parameters which will characterize the features of the channel (e.g. No of slots, Slot width etc). The user can use the "Mcbsp_ChanParams" Structure to specify the parameters to configure the channel.

The mcbspMdCreateChan () function typically does the following.

- It validates the input parameters given by the application.
- It checks if the requested channel is already opened or not. If it is already opened the driver will flag an error to the application else the requested channel will be allocated.
- It updates the appropriate channel objects with the user supplied parameters.
- The McBSP is configured with the appropriate word width.
- The EDMA parameters for the requested channel are setup.
- If the global error callback function registration is enabled, the appropriate user supplied function is registered to be called in case of an error.
- If the LOOPJOB configuration is enabled then the respective section (TX or RX) is enabled and the EDMA transfer is enabled.
- The channel creation if it fails will perform a cleanup and also free all the resource allocated by it till now.
- If the complete process of channel creation is successful, then the application will be returned a unique Handle. This Handle should be used by the application for further transactions with the channel. This Handle will be used by the driver to identify the channel on which the transactions are being requested.

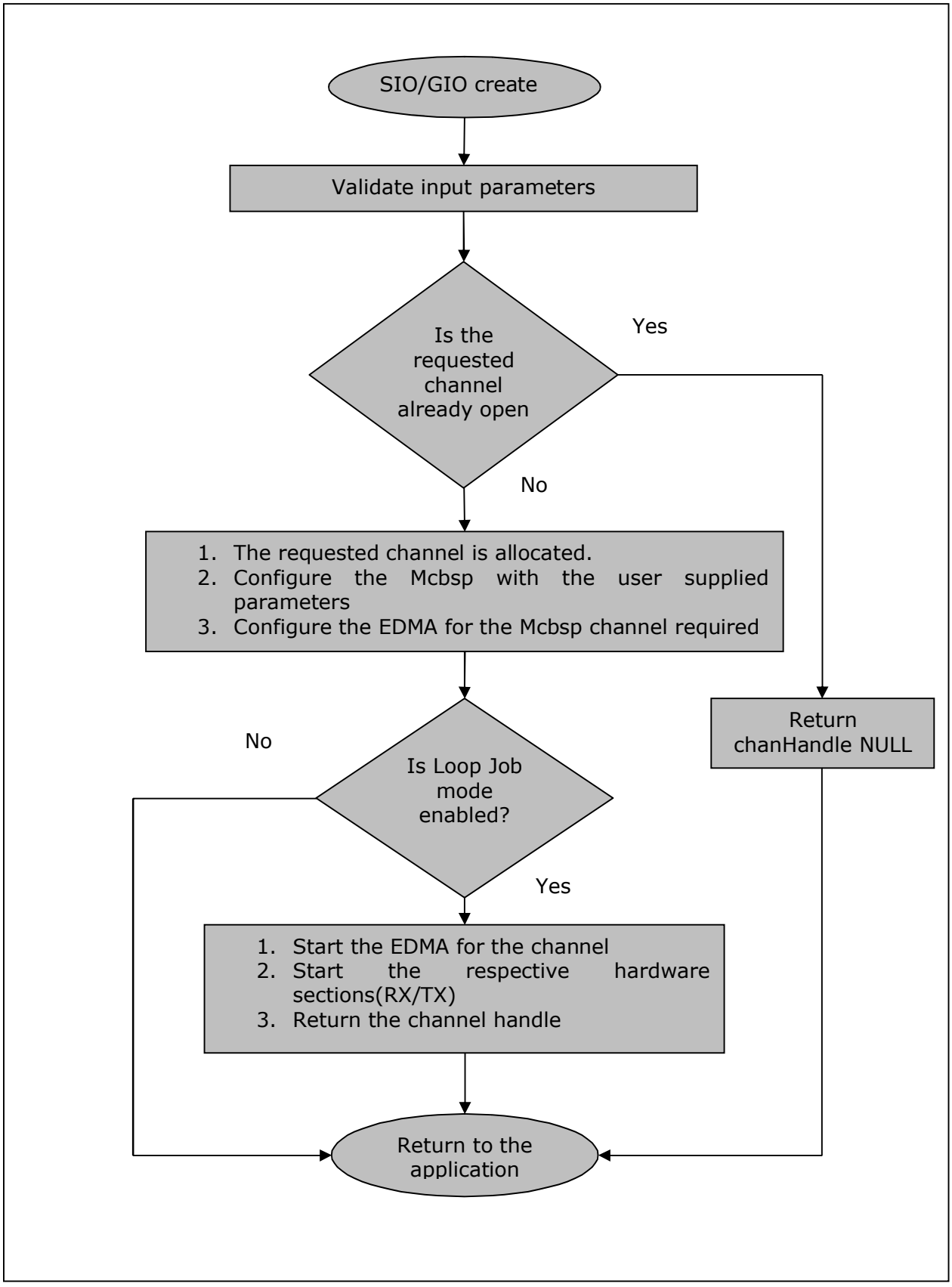

Figure 6 McBSP channel create command FlowIO Control

3.3.4 IO submit

McBSP IOM driver provides an interface to submit IO packets for the IO transactions to performed. 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.

The command codes to be supported by the McBSP IOM mini-driver are: IOM_READ, IOM_WRITE, IOM_ABORT, and IOM_FLUSH.

- **IOM_READ**. Drivers that support input channel must implement IOM_READ.
- **IOM_WRITE**.Drivers that support output channel 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 callback function registered for the channel for the channel.

  Note: The behavior of the driver will be same for both the ABORT and FLUSH i.e. all the packets will be aborted and returned back to the application.

The mdSubmitChan function of the McBSP driver typically performs the following activities.

1. The input packet is validated.
2. If the driver has sufficient packets then the current IO packet is loaded in to the pending queue.
3. Otherwise the IOP is programmed in to the link Params of the EDMA.

NON LOOP JOB MODE

In NON LOOP JOB mode, the first packet is always loaded in to the Main transfer channel. The subsequent two packets are loaded in to the spare param sets of the EDMA. Also if this is the first packet for the driver then also the clocks are started as per the requirement of the section.

Any other packets after this are loaded in to the pending queue. These packets will be loaded by the EDMA callback in to the appropriate param set of the EDMA.

3.3.5 Control Commands

McBSP IOM driver implements device specific control functionality which may be useful for any application, which uses the McBSP IOM driver. Application may invoke the control functionality through a call to GIO_control (). McBSP IOM driver supports the following control functionality.

The typical control flow for the McBSP control function is as given below.

- Validate the command sent by the application.
- Check if the appropriate arguments are provided by the application for the execution of the command.
- Process the command and return the status back to the application.

The below table lists the control commands supported by the McBSP driver

Command	Command Argument	Explanation
Mcbsp_Ioctl_McBSP_START	NULL	Starts the requested (TX or RX) section.
Mcbsp_Ioctl_McBSP_STOP	NULL	Stops the requested (TX or RX) section.
Mcbsp_Ioctl_McBSP_MUTE_ON[1]	NULL	Mutes the TX channel
Mcbsp_Ioctl_McBSP_MUTE_OFF[2]	NULL	Un-Mutes the TX channel
Mcbsp_Ioctl_McBSP_PAUSE	NULL	Pauses the selected section (channel)
Mcbsp_Ioctl_McBSP_RESUME	NULL	Resumes a previously paused channel.
Mcbsp_Ioctl_McBSP_CHAN_RESET	NULL	Resets the requested channel.
Mcbsp_Ioctl_McBSP_DEVICE_RESET	NULL	Resets the entire device by resetting

[1] These commands are applicable only for the TX section
[2]

		both the channels.
Mcbsp_Ioctl_McBSP_SRGR_START	NULL	starts the sample rate generator
Mcbsp_Ioctl_McBSP_SRGR_STOP	NULL	stops the sample rate generator
Mcbsp_Ioctl_McBSP_FSGR_START	NULL	starts the frame sync generator
Mcbsp_Ioctl_McBSP_FSGR_STOP	NULL	Stops the frame sync generator.
Mcbsp_Ioctl_McBSP_SET_CLKMODE,	Mcbsp_BclkMode *	command to configure the bit clock mode.
Mcbsp_Ioctl_McBSP_SET_FRMSYNCMODE,	Mcbsp_FrSyncMode *	command to configure the frame sync mode
Mcbsp_Ioctl_McBSP_CONFIG_SRGR,	Mcbsp_srgConfig *	command to configure the sample rate generator
Mcbsp_Ioctl_McBSP_SET_BCLK_POL	Mcbsp_BclkPol *	command to set the Bit clock polarity
Mcbsp_Ioctl_McBSP_SET_FRMSYNC_POL	Mcbsp_FsPol *	command to set the frame sync polarity
Mcbsp_Ioctl_McBSP_MODIFY_LOOPJOB	Mcbsp_ChanParams *	command to configure the user supplied loop job buffer.
Mcbsp_Ioctl_McBSP_RECEIVE_SYNCERR_INT_ENABLE	NULL	command to enable the SYNCERR for RX section
Mcbsp_Ioctl_McBSP_XMIT_SYNCERR_INT_ENABLE	NULL	command to enable the SYNCERR for TX section

Mcbsp_Ioctl_McBSP_LOOPBACK	Mcbsp_Loopback *	Command to enable/disable the loopback mode
Mcbsp_Ioctl_McBSP_SPI_CHAN_RESET	NULL	Resets the required channel
Mcbsp_Ioctl_McBSP_SPI_DEVICE_RESET	NULL	Resets both the TX and RX channels
Mcbsp_Ioctl_McBSP_SPI_SET_CS_POL	Mcbsp_FsPol *	Set the polarity for the Chip select
Mcbsp_Ioctl_McBSP_SPI_SET_CLKX_POL	Mcbsp_ClkPol *	Sets the clock polarity for the TX clock

The basic control flow for the handling of the control commands for the driver is shown below. Please not that the individual command handling is not detailed here.

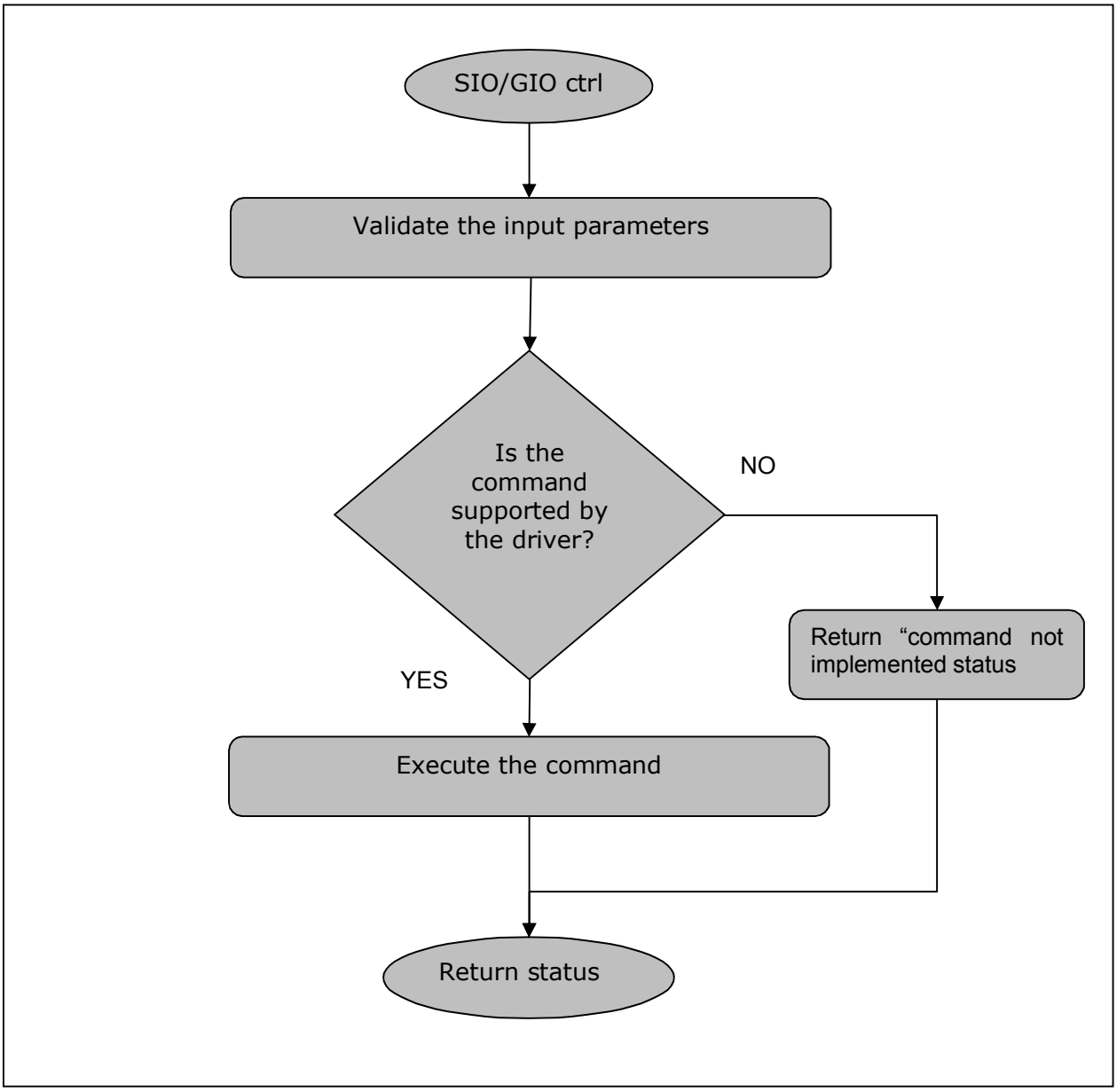

Figure 7 McBSP Control command Flow

3.3.6 Channel deletion

The channel once it has completed all the transaction can close the channel so that all the resources allocated to the channel are freed. The Mcbsp driver provides the "mcbspMdDeleteChan" API to delete a previously created McBSP channel.

The typical activites performed during the channel creation are as follows

1. The channel to be deleted is reset.
2. The reset operation aborts all the packets in the pending queue and also the packets in the current active queue.
3. The EDMA transfer for this channel is disabled.
4. The McBSP state machines are stopped.
5. The interrupt handlers are unregistered.
6. All the spare ParamSets of the EDMA are freed.
7. The status of the channel is updated to DELETED.

3.3.7 Driver unbinding/deletion.

The McBSP driver provides the interfaces for deleting the driver. The mcbspUnBindDev function de-allocates all the resources allocated to the driver during the driver binding operation. The typical operations performed by the unbind operation are as listed below.

- Check if both the TX and the RX channels are closed.
- Update the instance object.
- Set the status of the driver to "DELETED".
- Set the status of the module "inUse" to FALSE (so that it can be used again).

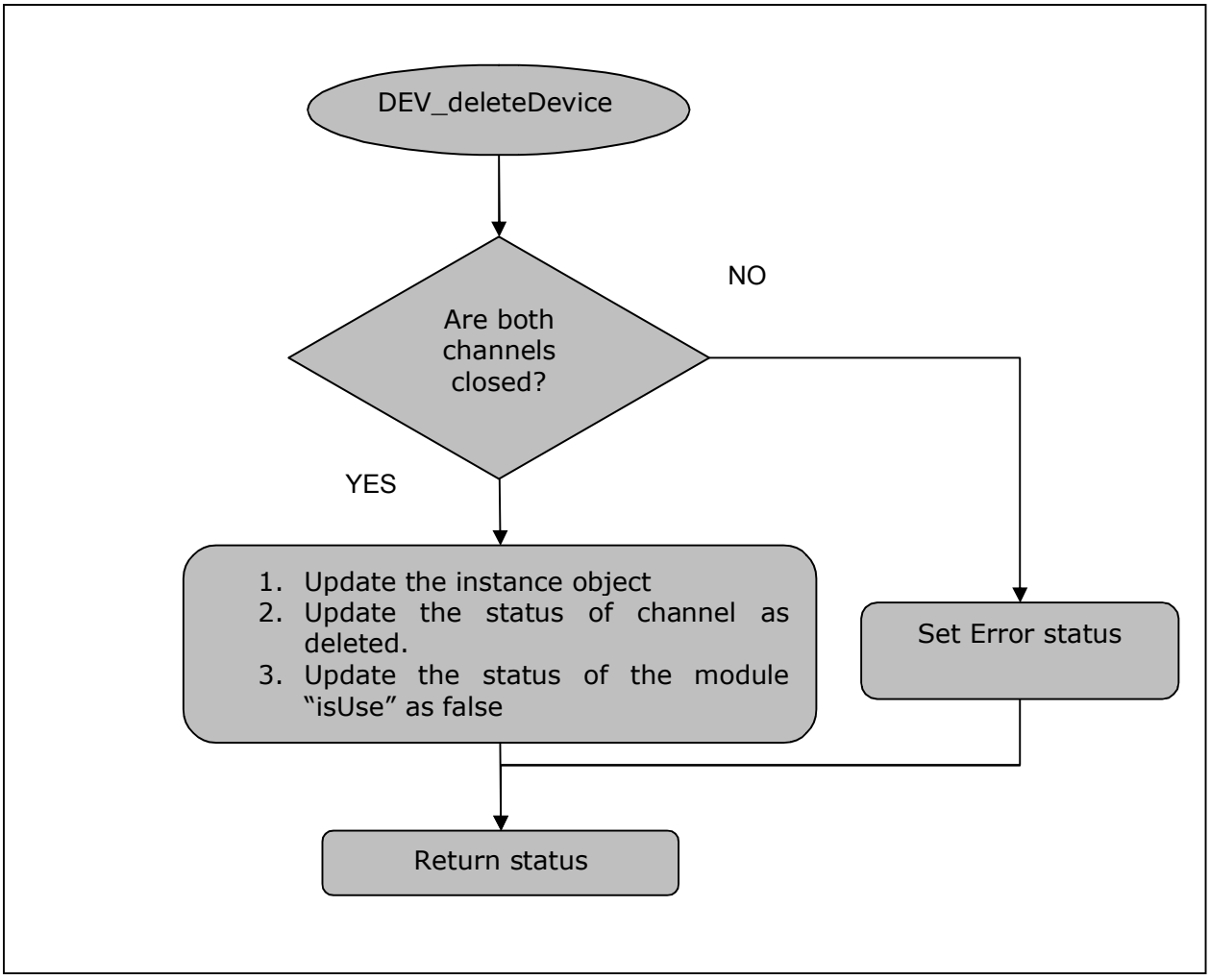

Figure 8 Driver deletion

3.3.8 Asynchronous IO Mechanism

The McBSP IOM driver supports asynchronous IO mechanism. In Asynchronous IO mechanism multiple IO requests can be submitted by the application thread without causing it to block while waiting for the IO request to complete. Application can submit multiple I/O requests using the "GIO_read ()" or "GIO_write ()" (also SIO) 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 internally will queue the IOM packets submitted to support the asynchronous I/O Functionality.

3.4 Constants & Enumerations

3.4.1 Mcbsp_TXEVENTQUE

This constant defines the EDMA3 event queue to be used in case of Transmit channel operation.

Definition

```
#define Mcbsp_TXEVENTQUE (0u)
```

Comments

None

Constraints

Please check the available event queues in the EDMA3 before changing/modifying this.

See Also

None

3.4.2 Mcbsp_RXEVENTQUE

This constant defines the EDMA3 event queue to be used in case of Receive channel operation.

Definition

```
#define Mcbsp_RXEVENTQUE (1u)
```

Comments

None

Constraints

Please check the available event queues in the EDMA3 before changing/modifying this.

See Also

None

3.4.3 Mcbsp_OpMode

This Enumeration defines the operating mode of the McBSP driver.

Definition

```
typedef enum Mcbsp_OpMode_t
{
    Mcbsp_OpMode_POLLED = 0,
    Mcbsp_OpMode_INTERRUPT,
    Mcbsp_OpMode_DMAINTERRUPT
} Mcbsp_OpMode;
```

Comments

None

Constraints

Only EDMA mode of operation is supported by the McBSP driver.

See Also

None

3.4.4 Mcbsp_DevMode

This Enumeration is used to define the operational mode of the McBSP device like normal McBSP device or SPI device (master/slave) mode.

Definition

```
typedef enum Mcbsp_DevMode_t
{
    Mcbsp_DevMode_McBSP,
    Mcbsp_DevMode_SPIMASTER,
    Mcbsp_DevMode_SPISLAVE
} Mcbsp_DevMode;
```

Comments

None

Constraints

The SPI mode of operation is only supported if the underlying hardware supports the same.

See Also

None

3.4.5 Mcbsp_BufferFormat

This Enumeration is used to specify the different types of Buffer formats supported by the McBSP driver.

Definition

```
typedef enum Mcbsp_BufferFormat_t
{
    Mcbsp_BufferFormat_1SLOT,
    Mcbsp_BufferFormat_MULTISLOT_NON_INTERLEAVED,
    Mcbsp_BufferFormat_MULTISLOT_INTERLEAVED
} Mcbsp_BufferFormat;
```

Comments

None

Constraints

None

See Also

None

3.5 Data Structures

3.5.1 Driver Instance Object

This structure is the Mcbsp driver's internal data structure. This data structure is used by the driver to hold the information specific to the instance. There will be one unique instance object for every instance of the McBSP controller supported by the driver.

Definition

```
typedef struct Mcbsp_Object_t
{
    Int32                       instNum;
    Mcbsp_DriverState           devState;
    Mcbsp_OperatingMode         mode;
    Mcbsp_OpMode                opMode;
    Bool                        enablecache;
    Mcbsp_HwInfo                hwInfo;
    Uint32                      mcbspSpiFreq;
    Bool                        stopSmFsXmt;
    Bool                        stopSmFsRcv;
    Mcbsp_ChannelObj            xmtObj;
    Mcbsp_ChannelObj            rcvObj;
    Mcbsp_srgConfig             srgrConfig;
    SWI_Handle                  swiHandle;
    Bool                        txSrgEnable;
    Bool                        rxSrgEnable;
    Bool                        srgConfigured;
    volatile Bool               srgEnabled;
    Bool                        txFsgEnable;
    Bool                        rxFsgEnable;
    Bool                        fsgConfigured;
    volatile Bool               fsgEnabled;
    Uint32                      retryCount;
} Mcbsp_Object;
```

Fields

instNum	Instance number of the McBSP.
devState	Current state of the driver (Created/Deleted).

Mode	Operating mode of the McBSP (Mcbsp, SPI master Mode, SPI slave mode).
opMode	Mode of operation of the driver(POLLED/INTERRUPT/DMA)
enableCache	Whether the driver should take care of cache cleaning operations for the buffers submitted by the application
hwInfo	Structure holding the hardware information related to the instance (e.g. interrupt numbers, base address etc).
mcbspSpiFreq	Frequency of operation of the Mcbsp in the SPI mode.
stopSmFsXmt	State of transmit state machine. (TRUE = stopped, FALSE = running).
stopSmFsRcv	State of receive state machine. (TRUE = stopped, FALSE = running).
xmtObj	Transmit channel object
rcvObj	Receive channel object
srgrConfig	Sample rate generator configurations supplied by the user.
txSrgEnable	Variable to indicate if the sample rate generator is required by the TX section.
rxSrgEnable	Variable to indicate if the sample rate generator is required by the RX section.
srgConfigured	Variable to indicate if the sample rate generator is configured or not.
srgEnabled	Variable to indicate if the sample rate generator is running.
txFsgEnable	Variable to indicate if the frame sync generator is required by the TX section.
rxSrgEnable	Variable to indicate if the frame sync generator is required by the RX section.
fsgEnabled	Variable to indicate if the frame sync generator is running.
retryCount	Retry count to be used by the driver when waiting in indefinite loops. (e.g. waiting for the TX to get empty etc).

Comments

4. The Mcbsp Driver works only in the EDMA mode of operation.
5. SPI mode is supported only if the underlying hardware supports it.
6. One instance object represents one instance of the driver.

Constraints

None

See Also

Mcbsp_ChannelObj

3.5.2 Channel Object

This structure is the Mcbsp driver's internal data structure. This data structure is used by the driver to hold the information specific to the channel. There will be at most two channels supported per instance(one for TX and one for RX).it is used to maintain the information pertaining to the channel like the current channel state, callback function etc. This structure is initialized by mdCreateChan and a pointer to this is passed down to all other channel related functions. Lifetime of the data structure is from its creation by mdCreateChan till it is invalidated by mdDeleteChan.

Definition

```
typedef struct Mcbsp_ChannelObj_t
{
    Uint16                  mode;
    Mcbsp_DriverState       chanState;
    Ptr                     devHandle;
    IOM_TiomCallback        cbFxn;
    Arg                     cbArg;
    Ptr                     edmaHandle;
    Uint32                  edmaEventQue;
    EDMA3_RM_TccCallback    edmaCallback;
    Uint32                  xferChan;
    Uint32                  tcc;
    Uint32                  pramTbl[Mcbsp_MAXLINKCNT];
    Uint32                  pramTblAddr[Mcbsp_MAXLINKCNT];
    QUE_Obj                 queuePendingList;
    QUE_Obj                 queueFloatingList;
    IOM_Packet              *tempPacket;
    IOM_Packet              *dataPacket;
    Uint32                  submitCount;
    Mcbsp_BufferFormat      dataFormat;
    volatile Bool           nextFlag;
    volatile Bool           bMuteON;
    volatile Bool           paused;
    volatile Bool           flush;
    volatile Bool           isTempPacketValid;
    Bool                    enableHwFifo;
    Mcbsp_GblErrCallback    gblErrCbk;
    Uint32                  userDataBufferSize;
```

```
    Ptr                     loopJobBuffer;
    Uint16                  loopJobLength;
    Uint32                  nextLinkParamSetToBeUpdated;
    volatile Bool           loopjobUpdatedinParamset;
    Uint16                  roundedWordWidth;
    Uint16                  currentDataSize;
    Uint32                  rxBytesIndex;
    Uint32                  txBytesIndex;
    Mcbsp_DataConfig        chanConfig;
    Mcbsp_ClkSetup          clkSetup;
    Mcbsp_McrSetup          multiChanCtrl;
    Uint32                  chanEnableMask[4];
    Bool                    userLoopJob;
}Mcbsp_ChannelObj;
```

Fields

mode	Current operating mode of the channel (INPUT/OUTPUT).
chanState	Current state of the channel (opened/closed).
devHandle	Pointer to the instance object.
cbFxn	Callback function pointer
cbArg	Callback function argument
edmaHandle	Pointer to the EDMA handle given by the application.
edmaEventQue	EDMA event queue to be used by this channel.
edmaCallback	EDMA callback function pointer.
xferChan	The EDMA transfer channel to be used.
tcc	Transfer completion code to be used in case of EDMA mode.
pramTbl	Value of the two spare PARAM sets issued by the EDMA driver.
pramTblAddr	Address of the two spare paramsets.
queuePendingList	Queue for holding the pending packets.
queueFloatingList	Queue for Holding the currently executing packets.
tempPacket	Temporary place holder for the currently completed packet.
dataPacket	pointer to hold the IOM packet

submitCount	Total number of packets held in the driver for this channel
dataFormat	The Format in which the McBSP data is arranged in the buffer.
nextFlag	Flag used in stopping the McBSP state machines.
bMuteON	Flag to indicate if the mute is ON.
paused	Flag to indicate if the channel is paused.
flush	Flag to indicate if the flush command is Issues to the driver.
isTempPacketValid	Flag to indicate if the "tempPacket" is holding a valid packet.
enableHwFifo	Flag to indicate if the Hardware FIFO is to be enabled for this channel (RX/TX).
gblErrCbk	Application registered callback function to be called in case of an error.
userDataBufferSize	Size of the user supplied buffer.
loopJobBuffer	Loop job buffer to be used when the driver does not have any more packets for the IO
loopJobLength	Length of the loop job buffer.
userLoopJobLength	User specified loop job's length.
nextLinkParamSetToBeUpdated	Variable to indicate which of the spare paramset is to be updated next.
loopjobUpdatedinParamset	Variable to indicate if the loop job is loaded in to the paramset.
roundedWordWidth	The actual word width to be transferred per sync event.
currentDataSize	The size of the current data packet
rxBytesIndex	Number of RX bytes transferred (Only supported in SPI mode).
txBytesIndex	Number of TX bytes transferred (Only supported in SPI mode).
chanConfig	Channel configuration required for the configuring of the channel.
clkSetup	Clock setup to be used for this channel.
multiChanCtrl	Multiple channel selection settings.
chanEnableMask	Mask for the channels to be enabled

userLoopJob	Variable to indicate if the user loop job is used or internal driver loop job buffer.

Comments

3. Only 2 channels are supported per instance
4. SPI mode is supported only if the underlying hardware supports it.

Constraints

None

See Also

Mcbsp_Object

3.5.3 Mcbsp_Params

This structure is used to supply user parameters during the creation of the driver instance. During the creation of the driver using the static creation or dynamic creation the user needs to supply the above structure with the required parameters. The structure is as defined below

Definition

```
typedef struct Mcbsp_Params_t
{
    Mcbsp_DevMode           mode;
    Mcbsp_OpMode            opMode;
    Bool                    enablecache;
    Mcbsp_Loopback          dlbMode;
    Mcbsp_ClkStpMode        clkStpMode;
    Uint32                  mcbspSpiFreq;
    Mcbsp_srgConfig        *srgSetup;
} Mcbsp_Params;
```

Fields

mode	Operating mode of the Mcbsp (Mcbsp, SPI master Mode, SPI slave mode).***Default mode is McBSP mode****.*
opMode	Mode of operation of the controller. ***Default is EDMA mode.***
	Note: *Only EDMA mode is supported for the Mcbsp mode of operation*
enableCache	Whether the driver should take care of cache cleaning operations for the buffers submitted by the application.
dlbMode	Digital loop back mode selection.
clkStpMode	Clock mode selection.
mcbspSpiFreq	Frequency of operation in SPI mode.
srgSetup	Sample rate generator setup.

Comments

1. The Mcbsp Driver works only in the EDMA mode of operation.
2. SPI mode is supported only if the underlying hardware supports it.
3. *"mcbspSpiFreq"* is required only when configuring in SPI mode. Otherwise "0" can be specified.

Constraints

None

See Also

Mcbsp_srgConfig

3.5.4 Mcbsp_ChanParams

This structure is used to supply user parameters during the creation of the channel instance. During the creation of the channel, user needs to supply the above structure with the appropriate parameters as per his mode of operation. The structure is as defined below

Definition

```
typedef struct Mcbsp_ChanParams_t
{
    Uint32                  wordWidth;
    Ptr                     userLoopJobBuffer;
    Uint16                  userLoopJobLength;
    Mcbsp_GblErrCallback    gblCbk;
    Ptr                     edmaHandle;
    Uint32                  edmaEventQue;
    Uint32                  hwiNumber;
    Mcbsp_BufferFormat      dataFormat;
    Bool                    enableHwFifo;
    Mcbsp_DataConfig       *chanConfig;
    Mcbsp_ClkSetup         *clkSetup;
    Mcbsp_McrSetup         *multiChanCtrl;
    Uint32                  chanEnableMask[4];
}Mcbsp_ChanParams;
```

Fields

wordWidth	Word width per slot
userLoopJobBuffer	User supplied loop job buffer
userLoopJobLength	User supplied buffer length

gblCbk	Pointer to the function to handle the Error conditions.
edmaHandle	Handle to the EDMA driver.
edmaEventQue	Event queue of the EDMA to be used by this channel.
hwiNumber	HWI number for the ECM group in which the event is configured
dataFormat	Buffer format to be used by the application
enableHwFifo	Flag to indicate whether hardware FIFO's are to be enabled.
chanConfig	Channel configuration settings.
clkSetup	Clock configuration settings.
MultiChanCtrl	Multi channel control settings.
chanEnableMask	Multiple channel selection mask

Comments

1. The user can decide to give his loop Job buffer if required. Otherwise the "userLoopJobBuffer" and "userLoopJobLength" can be NULL and 0 respectively. In case that the user has not specified the buffer then the driver will use its internal buffer.

   Note: This is applicable only if the driver is in loop job mode.

2. Please refer to the user guide for the various buffer formats supported by the McBSP driver.
3. "gblCbk" function will be called in the ISR context hence appropriate care should be taken that the function confirms to the ISR coding guidelines.
4. "hwiNumber" needs to specified according to the ECM event group that the channel being configured falls in to.

Constraints

See above.

See Also

SIO_create (), GIO_Create (),*Mcbsp_DataConfig*

3.5.5 Mcbsp_srgConfig

This is the McBSP sample rate generator configuration structure.The application needs to configure the Sample rate generator to generate the BCLK and Frame Sync signals at the specified rate in McBSP master mode.

Definition

```
typedef struct Mcbsp_srgConfig_t
{
    Bool            gSync;
    Mcbsp_ClkSPol   clksPolarity;
    Mcbsp_SrgClk    srgInputClkMode;
    Uint32          srgrInputFreq;
    Uint32          srgFrmPulseWidth;
}Mcbsp_srgConfig;
```

Fields

gSync	Sample rate generator synchronization bit
clksPolarity	CLKS polarity used to drive the CLKG and FSG clocks.
srgInputClkMode	Source for the sample rate generator.
srgrInputFreq	Input frequency for the Sample rate generator
srgFrmPulseWidth	Frame sync width

Comments

1. This structure will be required to specify the sample rate generator settings if it is required.
2. The driver will device internally if the sample rate generator need to be enabled or not depending on the TX or RX section clock requirements

Constraints

None

See Also

Mcbsp_Params

3.5.6 Mcbsp_DataConfig

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

Definition

```
typedef struct Mcbsp_DataConfig_t
{
    Mcbsp_Phase              phaseNum;
    Mcbsp_WordLength         wrdLen1;
    Mcbsp_WordLength         wrdLen2;
    Uint32                   frmLen1;
    Uint32                   frmLen2;
    Mcbsp_FrmSync            frmSyncIgn;
    Mcbsp_DataDelay          dataDelay;
    Mcbsp_Compand            compandSel;
    Mcbsp_BitReversal        bitReversal;
    Mcbsp_IntMode            intMode;
    Mcbsp_Rjust              rjust;
    Mcbsp_DxEna              dxState;
}Mcbsp_DataConfig;
```

Fields

phaseNum	Option to choose single phase or dual phase frame.
wrdLen1	Word length for the first frame.
wrdLen2	Word length for the second frame. *Will be used only in case of a dual phase frame.*
frmLen1	Length of the first frame.
frmLen2	Length of the second frame. *To be specified only in case of dual frame.*
frmSyncIgn	Option to select the action to be taken in case if an unexpected frame sync.
dataDelay	Data delay from the frame sync
comapandSel	Companding law selection
bitReversal	Option to select the bit reversal of data (MSB first or LSB first).

intMode	Event which should generate an CPU interrupt
rjust	Receive data justification settings
dxState	DX pin high impedance state enable disable option.

Comments

1. This frmLen2 and wrdLen2 options should be used only in case of an dual phase frame.
2. dxState option is applicable only while creating a channel for transmission.
3. rjust option is applicable only in case of creating a channel for reception.

Constraints

None

See Also

Mcbsp_Params

3.5.7 Mcbsp_ClkSetup

This structure is used to configure the clock settings for the Mcbsp channel.

Definition

```
typedef struct Mcbsp_ClkSetup_t
{
    Mcbsp_FsClkMode        frmSyncMode;
    Uint32                 samplingRate;
    Mcbsp_TxRxClkMode      clkMode;
    Mcbsp_FsPol            frmSyncPolarity;
    Mcbsp_ClkPol           clkPolarity;
}Mcbsp_ClkSetup;
```

Fields

frmSyncMode	Frame sync generator mode (Internal/external).
samplingRate	Frame sync frequency.
clkMode	Bit clock mode (internal/external)
frmSyncPolarity	Frame sync polarity (active high/active low)
clkPolarity	Bit clock polarity

3.6 API Definition

3.6.1 Mcbsp_init

Syntax

```
Void Mcbsp_init(Void);
```

Arguments

IN	None
	No input arguments

Return Value

None	No return value

Comments

Function to initialize the Mcbsp driver data structures. This function initializes the hardware information per instance like CPU event numbers, EDMA numbers etc.

It also initializes the Mute buffers and aligns it to the required boundary.

LOOP JOB MODE

In loop Job mode the source loop job buffer and transmit loop job buffer are initialized and aligned to the required cache line boundary.

Constraints

- This function should be called by the application before creating any Mcbsp driver instance.
- This function should be called only once in the life time of the Application.

See *Also*

None

Note: Please refer to the section 3.2.1 for the other Interfaces provided by the McBSP driver.

4 Decision Analysis & Resolution

4.1 DAR Criteria

1. State machine disable handling during the FIFO mode for the last packet.

4.1.1 Alternative 1

- Wait in the EDMA callback for the FIFO to empty before the state machine is disabled.

4.1.2 Alternative 2

- Post a SWI to handle the last packet completion condition.

4.2 Decision

It has been decided to go ahead with the alternative 2. It is not advisable to wait for the FIFO to be emptied in the ISR context. Hence it is decided that the IO packet completion will be deferred to an SWI which will wait for the FIFO to empty before stopping the state machine.

5 Revision History

Version #	Date	Author Name	Revision History
0.1	30 April 2009	Imtiaz SMA	Created Newly for the OMPL138 Mcbsp Driver.
0.2	07 August 2009	Imtiaz SMA	Modified for the interface changes in the driver

«««« § »»»»