A DSP/BIOS Generic DMA McBSP Device Driver for TMS320C5000 DSPs

Software Development Systems

ABSTRACT

This document describes the usage and design of the generic TMS320C5000 DMA McBSP device driver for both the C54xx and C55xx DSPs. This device driver is written in conformance to the DSP/BIOS IOM device driver model and handles communication to and from the multichannel buffered serial port (McBSP), and uses the DMA to transfer the data. It can be used as a general purpose stand alone mini-driver to access a serial port, or with a codec specific device driver. For an example of how to implement a codec specific device driver that uses this generic mini-driver for data transportation, refer to *DSP/BIOS PCM3002 Codec Device Driver for the TMS320C5416 DSK* (SPRA855) or the *DSP/BIOS AIC23 Codec Device Driver for the TMS320C5510 DSK* (SPRA856).

Features:

- Multi instance (handles multiple serial ports simultaneously).
- Designed for (but not limited to) use with codec drivers.

Contents

List of Figures

List of Tables

1 Usage

The device driver described here is part of an IOM mini-driver. That is, it is implemented as the lower layer of a 2-layer device driver model. The upper layer is called the class driver and can be either the DSP/BIOS GIO, SIO/DIO, or PIP/PIO modules. The class driver provides an independent and generic set of APIs and services for a wide variety of mini-drivers and allows the application to use a common interface for I/O requests. Figure 1 shows the overall DSP/BIOS device driver architecture. For more information about the IOM device driver model as well as the GIO, SIO/DIO, and PIP/PIO modules, see the *DSP/BIOS Device Driver Developer's Guide* (SPRU616).

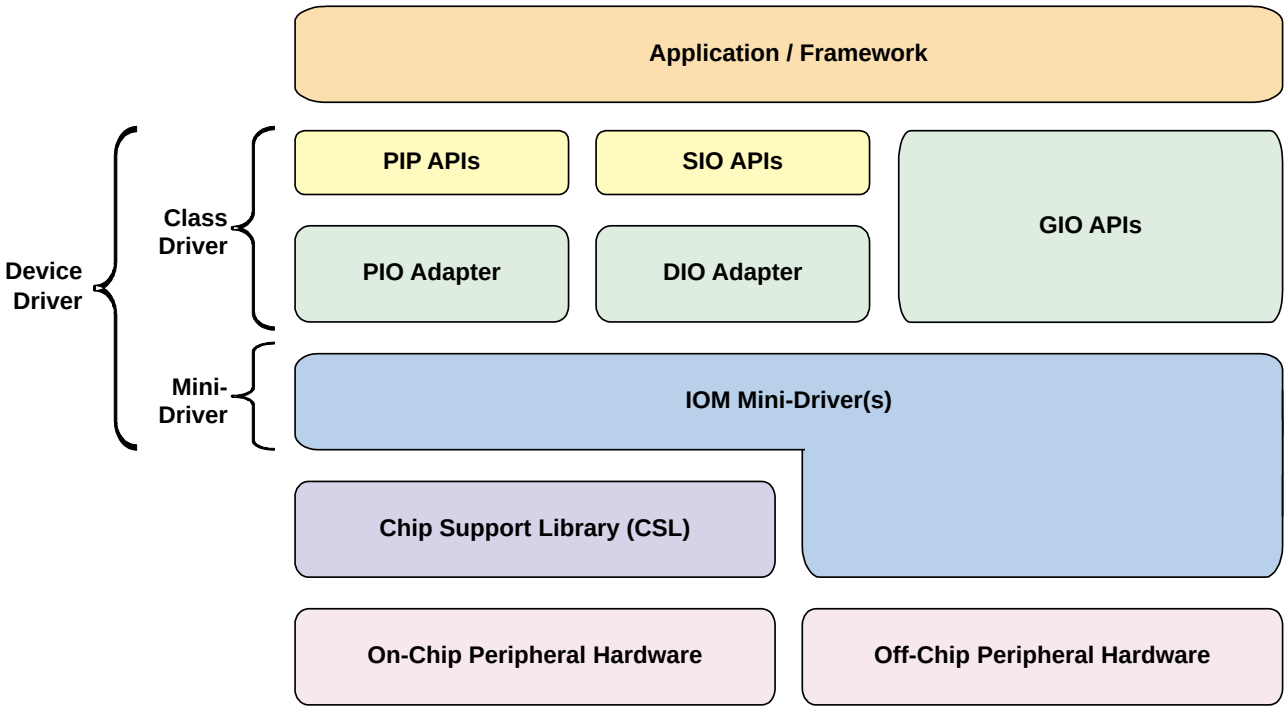

Figure 1. DSP/BIOS IOM Device Driver Model

This device driver can be used as a general-purpose stand-alone mini-driver to interface with the McBSP on TMS320C54xx and TMS320C55xx chips using the DMA. However, this device driver is mainly used in conjunction with the codec-specific portion of the mini-driver to handle its data processing. In that case, the codec-specific part only has to set up the codec and pass the required parameters to this generic part of the mini-driver. Figure 2 shows the data flow between the components in a system in which the mini-driver is split into a generic part and a codec-specific part.

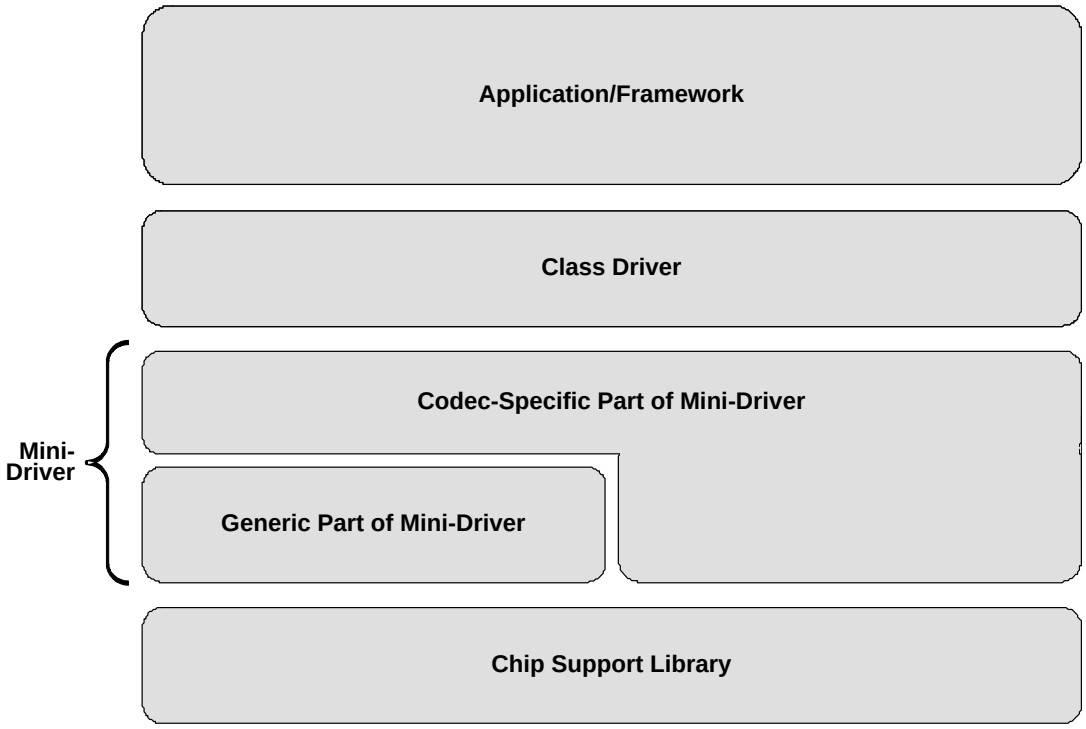

Figure 2. Codec Device Driver Partitioning

1.1 Configuration

To use this driver with a codec-specific portion of the mini-driver, the properties should be set up for the codec device driver. In that case, the codec-specific part will then set up this generic device driver. Refer to the individual codec-specific device driver documentation on how to set up the properties in this case.

To use this as a general purpose C54xx DMA McBSP or C55xx DMA McBSP device driver, a device entry for every McBSP port instance has to be added in the configuration tool. Refer to *DSP/BIOS Device Driver Developer's Guide* (SPRU616) for further configuration details. Below is a description of the properties of this device entry when using this device driver stand-alone.

- **Init function:** Type _C54XX_DMA_MCBSP_init *or* _C55XX_DMA_MCBSP_init
- **Function table ptr:** Type _C54XX_DMA_MCBSP_FXNS *or*_C55XX_DMA_MCBSP_FXNS
- **Function table type:** Select IOM_Fxns.
- **Device ID:** Type 0 for McBSP0, type 1 for McBSP1 or type 2 for McBSP2 if applicable. This device driver only supports McBSP0, McBSP1 or McBSP2 without modifications to the source.

- **Device params ptr:** A pointer to your instance of the device parameter structure (described below). This device driver has no default parameters and should be left to 0x0.
- **Device global data ptr:** This property must be set to 0x0.

The channel parameters (described below) are passed to the device driver during run-time when creating the device communications channel. For further details and on how to pass channel parameters when using this device driver with SIO, PIP or IOM refer to the *DSP/BIOS Device Driver Developer's Guide* (SPRU616).

This device driver uses the supplied McBSP and DMA CSL configuration structures (see the parameter sections below) to set up the McBSP and DMA respectively. This means that the device driver is generally (see "Constraints" below) only limited in operation by how you set up these peripheral configuration structures.

1.2 Device Parameters (same for C55xx)

```
typedef struct C54XX_DMA_MCBSP_DevParams {
      Int versionId;
      Uns rxDmaId;
      Uns txDmaId;
      MCBSP_Config *mcbspcfg;
      Uns rxIerMask[2];
      Uns txIerMask[2];
} C54XX_DMA_MCBSP_DevParams;
```

- **versionId:** Version number of the driver.
- **rxDmaId:** Receive DMA channel.
- **txDmaId:** Transmit DMA channel.
- **mcbspCfg:** This parameter is a pointer to a CSL configuration structure that will be passed to MCBSP_config() for the McBSP port.
- **rxIerMask:** Interrupt enable register mask, set in receiver ISR.
- **txIerMask:** Interrupt enable register mask, set in transmitter ISR.

1.3 Channel Parameters (same for C55xx)

```
typedef struct C54XX_DMA_MCBSP_ChanParams {
      DMA_Config *dmaCfg;
} C54XX_DMA_MCBSP_ChanParams;
```

- **dmaCfg:** A pointer to a CSL configuration structure to be passed to DMA_config() for the DMA channel used by this IOM channel.

1.4 Control Commands

This device driver has no run-time control commands.

2 Architecture

This section describes the design and implementation of the device driver. The driver uses various DSP/BIOS and CSL modules (see Appendix A). Refer to *TMS320C5000 DSP/BIOS Application Programming Interface* (SPRU404), *TMS320C54x Chip Support Library API Reference Guide* (SPRU420), and *TMS320C55x Chip Support Library API Reference Guide* (SPRU433). The technical details of the McBSP and DMA are available from *TMS320C54x Enhanced Peripherals Reference Set* (SPRU302) and *TMS320C55x Peripherals Reference Guide* (SPRU317).

2.1 Data Structures

This driver uses two key internal data structures: a port object and a channel object to maintain its state during execution. This device driver is multi instance, which means it can handle several McBSP ports running simultaneously. Every McBSP used needs a port object instance associated with it to keep its state. In turn, every port has two associated channel object instances (one for input and one for output) that hold the IOM channel states during execution. The contents of these structures are described below.

2.1.1 The Port Object (same for C55xx)

```
/* Number of IOM channels per port (must be 2, one for input and one for out-
put) */
#define NUMCHANS 2
/* Structure containing port specific variables */
typedef struct PortObj {
    Uns inUse;
    MCBSP_Handle hMcbsp;
    ChanObj chans[NUMCHANS];
} PortObj, *PortHandle;
```

- **inUse:** This variable is set when this port is configured, so that it can fail if another attempt to configure the port is made.
- **hMcbsp:** This variable holds the CSL handle returned by the CSL function MCBSP_open() for this port, and is used to access the McBSP during execution.
- **chans:** An array holding the channel objects associated with this port.

2.1.2 The Channel Object (same for C55xx)

```
/* Structure containing IOM channel specific variables */
typedef struct ChanObj {
    Uns inUse;
    Uns dmaId;
    void *devp;
    DMA_Handle hDma;
    IOM_Packet *flushPacket;
    IOM_Packet *dataPacket;
    QUE_Obj pendList;
    IOM_TiomCallback cbFxn;
    Ptr cbArg;
} ChanObj, *ChanHandle;
```

- **inUse:** This variable is set when this IOM channel is configured, so that it can fail if another attempt to configure the IOM channel is made.
- **dmaId:** DMA channel id.
- **devp:** When binding a device, this pointer is assigned to the port handle.
- **hDma:** When a channel is created, the DMA handle is assigned to the specific IOM channel that is made.
- **flushPacket:** Since this device driver uses an asynchronous flush command, this is where the flush packet sent to the IOM channel is stored when such a command has been issued.
- **dataPacket:** This is the current data packet sent to the DMA.
- **pendList:** A queue holding pending data packets that are to be submitted to the DMA.
- **cbFxn:** The callback function specified when creating the IOM channel is stored here. It is used to send an IOM packet to the upper layers.
- **cbArg:** The callback argument specified when creating the channel is stored here. It is used in conjunction with the callback function to send an IOM packet to the upper layers.

2.2 Data Flow

This section describes how a buffer is processed and passes through this device driver. When an IOM packet is issued to an IOM channel, it is first checked to see which command has been issued. The buffer is then submitted to the DMA or placed on a queue for later processing. Inside the interrupt, a buffer is obtained from the DMA and then calls the callback function inside the class driver for buffer consumption (refer to Figure 1 and the *DSP/BIOS Driver Developer's Guide* (SPRU616) for further details). Described below are the supported commands IOM_READ, IOM_WRITE, IOM_ABORT and IOM_FLUSH for this driver.

2.2.1 The IOM Read and Write Commands

The device driver handles read and write similarly. The mode of the IOM channel to which the packet was issued decides whether it's a read or a write command, not the IOM packet command field.

First there is a check to see if there is space available to place a new DMA packet in the corresponding DMA channel. If not, the packet gets put on a queue (pendList) until there is space available for the packet. If there is space available, the packet is submitted and programs the appropriate DMA channel to start the transfer.

Depending on whether this is an input or an output channel, the destination or the source field is set with the packet's address field respectively.

The driver then programs an appropriate DMA channel with the correct packet. The buffer's address and size are stored inside the IOM_Packet. Note that on the TMS32055xx DSPs the compiler assigns all data symbols as word address; however, the DMA expects all addresses to be byte address and we must shift the address by 2 in order to change the word address to a byte address for the DMA transfer.

When this setup is done, the DMA will start the new DMA job corresponding to the issued IOM packet (input or output depending on which IOM channel) when the currently executing job terminates and the next DMA job is set up and becomes the new job.

When a DMA job completes, a DMA interrupt will occur. The ISR fetches the completed input or output DMA jobs corresponding packet from the pendList in the channel object, marks it as completed and calls the callback function on this packet to send it to the upper layers. If not, the ISR will set up the next DMA job corresponding to the issued IOM packet (input or output depending on which IOM channel) and it becomes the new job.

2.2.2 The IOM Abort and Flush Commands

The driver handles IOM_ABORT and IOM_FLUSH commands in the same way for both input and output mode. For IOM_FLUSH, this driver does not wait for pending output to complete, but simply stops the DMA and returns the outstanding packets to the application. The driver calls the abortio function when either of these commands are received by mdSubmitChan.

The abortio function stops the DMA and calls the callback function for all of the packets in the driver, in the order that they were submitted. The status field for each of these packets is set to IOM_ABORTED, since the DMA is stopped before the I/O could complete.

After all of the submitted packets have been returned to the client, the mdSubmit function returns IOM_COMPLETED. Unlike some other drivers, this driver handles IOM_ABORT and IOM_FLUSH commands synchronously, since mdSubmit always returns IOM_COMPLETED, and never waits for pending I/O to complete before returning.

3 Constraints

- This device driver currently only supports three McBSP ports (McBSP0, McBSP1 and McBSP2).
- The driver is not currently reentrant, which means two different threads cannot issue IOM packets to the same IOM channel in a safe way. Use a semaphore to protect the issuing of IOM packets if necessary.

4 References

All these documents are available from http://www.ti.com.

1. *A DSP/BIOS AD50 Codec Device Driver for the TMS320C5402 DSK* (SPRA854).
2. *A DSP/BIOS PCM3002 Codec Device Driver for the TMS320C5416 DSK* (SPRA855).
3. *A DSP/BIOS AIC23 Codec Device Driver for the TMS320C5509 EVM* (SPRA857).
4. *DSP/BIOS AIC23 Codec Device Driver for the TMS320C5510 DSK* (SPRA856).
5. *DSP/BIOS Device Driver Developer's Guide* (SPRU616).
6. *TMS320C5000 DSP/BIOS Application Programming Interface* (SPRU404).
7. *TMS320C54x Chip Support Library API Reference Guide* (SPRU420).
8. *TMS320C55x Chip Support Library API Reference Guide* (SPRU433).
9. *TMS320C54x Enhanced Peripherals Reference Set* (SPRU302)
10. *TMS320C55x Peripherals Reference Guide* (SPRU317).

Appendix A Device Driver Data Sheet

A.1 Device Driver Library Name

c5402_dma_mcbsp.l54 (near mode) and c5402_dma_mcbsp.l54f (far mode) for TMS320C5402 DSPs.

c5416_dma_mcbsp.l54 (near mode) and c5416_dma_mcbsp.l54f (far mode) for TMS320C5416 DSPs.

c5509_dma_mcbsp.l55 (small model) and c5509_dma_mcbsp.l55l (large model) for TMS320C5509 DSPs.

c5510_dma_mcbsp.l55 (small model) and c5510_dma_mcbsp.l55l (large model) for TMS320C5510 DSPs.

A.2 DSP/BIOS Modules Used

- HWI – Hardware Interrupt Manager
- QUE – Queue Manager
- IOM – I/O Manager
- ATM – Atomic Manager

A.3 DSP/BIOS Objects Used

- QUE_Obj

A.4 CSL Modules Used

- McBSP module
- DMA module
- IRQ module

A.5 CPU Interrupts Used

- Two DMA interrupts

A.6 Peripherals Used

- McBSP
- DMA

A.7 Interrupt Disable Time

Maximum time that hardware interrupts can be disabled by the driver:

- 209 cycles – c54xx device drivers
- 221 cycles – c55xx device drivers

These measurements are taken using the compiler option –O3.

A.8 Memory Usage

Table A–1. c5402_dma_mcbsp Device Driver Memory Usage

	Uninitialized memory	Initialized memory
CODE	—	843 (16-bit words)
DATA	50 (16-bit words)	54 (16-bit words)

Table A–2. c5510_dma_mcbsp Device Driver Memory Usage

	Uninitialized memory	Initialized memory
CODE	—	912 (8-bit bytes)
DATA	180 (16-bit words)	190 (16-bit words)

NOTE: This data was gathered using the sectti command utility.
Uninitialized data: .bss
Initialized data: .cinit + .const
Initialized code: .text + .text:init

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265