C6452 TSIP Driver

USER'S GUIDE

Document Revision History

Rev No	Author(s)	Revision History	Date	Approval(s)
1.1	Chandan Nath	Updated for adding compiler switches in build options	May 21, 2008	Updating
1.0	Jayaprakash		Nov 29, 2007	

TABLE OF CONTENTS

1 Introduction 4

- 1.1 Terms & Abbreviations 4
- 1.2 References 5
- 1.3 S/W Support 6
- 1.4 Driver Components 6
- 1.5 Default Driver Configuration 7
- 1.6 Driver Capabilities 7
- 1.7 Driver Limitations/Constraints 8
- 1.8 System Requirements 9

2 Installation Guide 9

- 2.1 Component Folder 9
- 2.2 Build 10
- 2.3 Build Options 10

3 DSP/BIOS TSIP DRIVER Structures 12

- *3.1.1 Initialization details 12*
- *3.1.2 TSIP device params (PSP_tsipDevParams) 13*
- *3.1.3 TSIP channel params (PSP_tsipChanParams) 16*
- *3.1.4 TSIP IO params (PSP_tsipIoParams) 17*

TSIP API’s 17

- 3.2 Structures Passed to GIO APIs 18
- *3.2.1 Structure passed to GIO_create 18*
- *3.2.2 Structure passed to GIO_submit 18*
- *3.2.3 Enumerations for IOCTL 18*
- 3.3 API Definition 20
- *3.3.1 GIO_ create 20*
- *3.3.2 GIO_delete 22*
- *3.3.3 GIO_control 22*
- *3.3.4 GIO_Submit 24*
- *3.3.5 GIO_flush/GIO_abort 26*

4 Example Applications 27

- *5.1 Writing Applications for TSIP 27*
- *5.1.1. File Inclusion 27*
- *5.2 Sample Applications 27*

TABLE OF FIGURES

Figure 1. C6452 TSIP driver architecture 6
Figure 2. TSIP Driver Directory Structure...9

1 Introduction

This document is the reference guide for the TSIP driver and it explains how to configure and use the driver.

DSP/BIOS applications use the driver typically through TSIP APIs to perform read/write operations on the TSIP peripheral. TSIP driver is implemented as a simple wrapper on top of the GIO class driver and provides an application-specific interface. For more information on the DSP/BIOS device driver model and the GIO class driver, refer to the References section of this document.

1.1 Terms & Abbreviations

Term	Description
⚐	This bullet indicates important information. Please read such text carefully.
❑	This bullet indicates additional information.
API	Application Programming Interface
CSLR	Chip Support Library for Registers
DDC	Device Driver Core
DMATCU	DMA Transfer Control Unit of TSIP
IOM	Input/Output Mini driver (Device Driver Adapter)
ISR	Interrupt Service Routine
OS	Operating System
ROM	Read Only Memory
SOC	System On Chip
SIU	Serial Interface Unit of TSIP
TDMU	Timeslot Data Management Unit of TSIP
TSIP	Telecom Serial Interface Port

1.2 References

1.	sprueu8_TSIP.pdf	TSIP functional specification document
2.	SPRU-404g.pdf	DSP/BIOS Driver Guide

1.3 S/W Support

TSIP device driver has been developed for the DSP/BIOS operating system using the TI supplied Chip Support Library. For more details on the version numbers refer to the release notes in the root of the installation.

1.4 Driver Components

The TSIP driver is constituted of following sub components:

TSIP IOM – Application interfacing, OS Specific Adaptation of TSIP Device Driver
TSIP DDC –OS Independent part of TSIP Driver Core
TSIP CSLR–The low-level TSIP h/w register overlay

System components:

PALOS – DSP/BIOS Abstraction

Below Figure shows C6452 TSIP driver architecture.

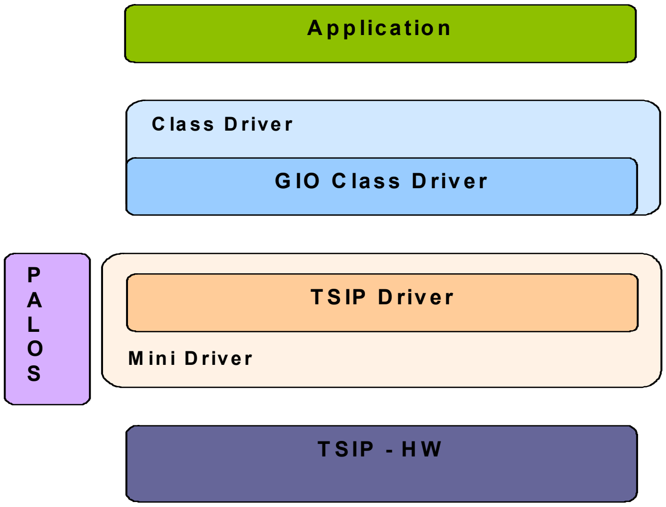

Figure 1. C6452 TSIP driver architecture

1.5 Default Driver Configuration

To configure the TSIP driver to default values, the application has to pass the following hardware setup parameters.

Clock select	Redundant
TX clock and FS source	CLK_A and FS_A
TX data delay	By 1 frame
TX output disabled state	High Impedance
TX output enable delay	0
TX frame sync polarity	Active low
TX frame clock polarity	Rising edge
TX data clock polarity	Rising edge
TX data rate	8.192 Mbps
TX clock mode	1X
TX frame count	3
TX frame size	127
RX clock and FS source	CLK_A and FS_A
RX data delay	By 1 frame
RX frame sync polarity	Active low
RX frame clock polarity	Rising edge
RX data clock polarity	Rising edge
RX data rate	8.192 Mbps
RX clock mode	1X
RX frame count	3
RX frame size	127
Loopback	None
testClk	EXTERNAL
Channel buffer Endian	Little
Transfer max priority	7
Transfer priority	7
TX frame interrupt selection type	Transfer acknowledge only
TX super frame interrupt selection type	Transfer acknowledge only
RX frame interrupt selection type	Transfer acknowledge only
RX super frame interrupt selection type	Transfer acknowledge only
TX frame interrupt delay	127
RX frame interrupt delay	127

1.6 Driver Capabilities

The significant driver features are:

- Isolates H/W and OS Accesses.
- Easy to maintain & re-target to new platforms.
- Supports multiple instances.
- Transmit and receive handled by different channels
- Cache is properly handled because TSIP is operating in DMA mode always using its internal DMATCU

1.7 Driver Limitations/Constraints

- TSIP driver shall only work with Interrupt mode of operation. Since TSIP embeds an inbuilt DMA unit as part its IP, it will not use EDMA to perform data movement.
- TSIP driver dynamically configures the DMA channel parameters with the given A and B context buffer configurations specified in the IOP request packet submitted by the application. In case no IOPs are pending, the driver stops the corresponding TX/RX DMATCU channel. However the channel will be enabled automatically upon the submission of a new IOP request.
- The number of super-frames in an IOP should be an even number. Because it is required to give the super-frame buffer configurations for both A and B contexts in a given IOP. See TSIP IO request submission structure (PSP_tsipIoParams) in the data structure section for more details.

1.8 System Requirements

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

2 Installation Guide

2.1 Component Folder

Upon installing the TSIP driver, the following directory structure is found in the driver's directory.

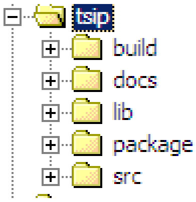

Figure 2. TSIP Driver Directory Structure

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

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

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

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

  Release Note gives the details about system requirements, steps to Install/Uninstall the package.This document list the known issues of the driver.

  User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.

  API reference document gives the details about the API's used in TSIP driver.
- **Lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **Package:** This folder contains files generated by XDC tool.
- **src:** This folder contains tsip driver source files. It also contains header files that are used by the driver.

2.2 Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed and how the same can be realized.

The component might be delivered to user in different formats:

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

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

2.3 Build Options

To compile driver for C6452, change build options as mentioned below:

The build folder contains a CCS project file that builds the driver into a library for debug and release mode.

Following compiler switches are used to compile for different options.

- ❑ **_DEBUG**
  This is used as a flag to compiler whether to include the debug statements inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.
- ❑ **CHIP_XXXX**
  The CSL layer is written in a common file for all the variants of a SOC. This flag differentiates the variant we are compiling for, for e.g. - CHIP_DM648, and the CSL definitions for that variant appropriately gets defined for register base addresses, num of ports of a peripheral etc.
- ❑ **TSIP_INSTRUMENTATION_ENABLED**
  This option is passed to the compiler to include the instrumentation code parts into the final image/lib of the program. This helps build the iRelease/iDebug versions of the image/lib with a common code base.
- ❑ **PSP_TSIP_USE_FRAME_ISR**
  If this macro is enabled, the Frame interrupt will be enabled and ISR gets control at the end of each frame completion. Generally this is not recommended because of this ISR overhead the driver may miss to program the DMA properly.

- **PSP_TSIP_DEBUG**

  They are the macros used to print debug messages. These expand a valid print statement.

3 DSP/BIOS TSIP DRIVER Structures

This section discusses about the initialization details and initialization structures used in the TSIP driver. Please note that for some structure member information/details of the C6452 TSIP peripheral API reference document might need to be referred.

3.1.1 Initialization details

To use TSIP device driver, a device entry must be added and configured in the DSP/BIOS configuration tool.

To have TSIP device driver included in the application, corresponding TCI file have to be included in BIOS TCF i.e. *"c6452_tsip0.tci"* for TSIP0 and *"c6452_tsip1.tci"* for TSIP1 must be included in BIOS TCF file of the application for using TSIP instance of the driver.

The following are the device configuration settings required to use the TSIP driver.
Note: This has to be done for all of the required driver instances. The corresponding IOM function tsip_mdBindDev will be called from BIOS context to initialize the TSIP.

TCI Configuration Parameters	Description
initFxn - Init Function	Pointer to the application function used to prepare the C6452 TSIP module and initialize init time configuration data structures. This function typically does enabling TSIP module in LPSC, PIN muxing and setting TSIP driver configurations.
fxnTable - Function Table Pointer	*TSIP_Fxns*. This is a global variable, which points to the TSIP driver IOM APIs.
fxnTableType - Function Table Type	*IOM_Fxns*
deviceId - Device Id	Specify which TSIP to use. For example to use TSIP0 this should be given as 0 and to use TSIP1 this should be given as 1.
params – Pointer to Port parameter	A pointer to an object of type *tsipDevParams* as defined in the header file *psp_tsip.h*. This pointer will point to a device parameter structure. In BIOS TCI files, this structure object is passed as an argument. Application should declare and initialize the structure object properly.
Device Global Data Pointer	N/A, not used by this driver

Final tci file should contain the following details which were explained above:

```
bios.UDEV.create("TSIP0");
bios.UDEV.instance("TSIP0").fxnTableType = "IOM_Fxns";
bios.UDEV.instance("TSIP0").initFxn = prog.extern("tsip0_dev_init");
bios.UDEV.instance("TSIP0").params = prog.extern("tsip0DevParams");
bios.UDEV.instance("TSIP0").fxnTable = prog.extern("tsipMiniDrvFxns");
```

3.1.2 TSIP device params (*PSP_tsipDevParams*)

"psp_tsip.h" file contains *PSP_tsipDevParams* data structure that is passed while mdBindDev call which is defined with UDEV TSIP parameters in *.tcf file of application. The members of this structure are explained below:

Structure Members	Description
PSP_tsipSiuConf	siu
PSP_tsipTdmuConf	tdmu

3.1.2.1 PSP_tsipSiuConf

Following table describes the parameters contained in PSP_tsipSiuConf data structure available in "psp_tsip.h" file.

Structure Members	Description
PSP_TsipClock	clkSel
PSP_tsipTxSiuConf	tx
PSP_tsipRxSiuConf	rx
PSP_TsipLoopBack	loopBack
PSP_TsipTestClock	testClk

3.1.2.2 PSP_tsipTdmuConf

Following table describes the parameters contained in PSP_tsipTdmuConf and detail of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TsipEndian	endian
Uint8	maxPrio
Uint8	prio
PSP_TsipIntSelect	txFint
PSP_TsipIntSelect	txSint
PSP_TsipIntSelect	rxFint
PSP_TsipIntSelect	rxSint
Uint8	txFintDly
Uint8	rxFintDly

3.1.2.3 PSP_ TsipClock

Following table describes the parameters contained in *PSP_* TsipClock (stopbits) enum, which is a parameter of SIU configuration structure and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_CLK_REDUNDANT	Redundant clock
PSP_TSIP_CLK_DUAL	Dual clock

3.1.2.4 PSP_tsipTxSiuConf

Following table describes the parameters contained in *PSP_tsipTxSiuConf,* which is a parameter of SIU configuration and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TsipClkFsSrc	clkSrc
Uint16	dDly
PSP_TsipTxDisState	disState

PSP_TsipTxEnaDly	enaDly
PSP_TsipPolarity	fSyncP
PSP_TsipPolarity	fClkP
PSP_TsipPolarity	dClkP
PSP_TsipDataRate	dRate
PSP_TsipClkMode	clkMode
Uint8	fCnt
Uint8	fSize

3.1.2.5 PSP_tsipRxSiuConf

Following table describes the parameters contained in *PSP_tsipRxSiuConf,* which is a parameter of SIU configuration and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TsipClkFsSrc	clkSrc
Uint16	dDly
PSP_TsipPolarity	fSyncP
PSP_TsipPolarity	fClkP
PSP_TsipPolarity	dClkP
PSP_TsipDataRate	dRate
PSP_TsipClkMode	clkMode
Uint8	fCnt
Uint8	fSize

3.1.2.6 PSP_TsipLoopBack

Following table describes the parameters contained in *PSP_TsipLoopBack* enum, which is a parameter SIU configuration and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_LOOP_BACK_NONE	Normal mode
PSP_TSIP_LOOP_BACK_DLB	Data loop back (Internal)
PSP_TSIP_LOOP_BACK_LLB	Link loop back (External)

3.1.2.7 PSP_TsipTestClock

Following table describes the parameters contained in *PSP_TsipLoopBack* enum, which is a parameter SIU configuration and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_TEST_CLK_EXTERNAL	Clock and Frame Sync are given from external source

3.1.2.8 PSP_TsipClkFsSrc

Following table describes the parameters contained in *PSP_TsipClkFsSrc* enum, which is a parameter TX and RX SIU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_CLK_FS_SRC_A	CLK_A and FS_A
PSP_TSIP_CLK_FS_SRC_b	CLK_B and FS_B

3.1.2.9 PSP_TsipTxDisState

Following table describes the parameters contained in *PSP_TsipTxDisState* enum, which is a parameter of TX SIU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_TXDIS_STATE_HIGHZ	TX lanes o/p disabled state is high impedance
PSP_TSIP_TXDIS_STATE_LOW	Driven low
PSP_TSIP_TXDIS_STATE_HIGH	Driver high

3.1.2.10 PSP_TsipTxEnaDly

Following table describes the parameters contained in *PSP_TsipTxEnaDly* enum, which is a parameter TX SIU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_TXENA_DLY_ZERO	No added delay
PSP_TSIP_TXENA_DLY_ONEHALF	TX output enable delayed by one and half serial clock periods.

3.1.2.11 PSP_TsipPolarity

Following table describes the parameters contained in *PSP_TsipPolarity* enum, which is a parameter of TX/RX SIU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_POLARITY_ACTIVE_LOW PSP_TSIP_POLARITY_RISING_EDGE	Active low / Rising edge
PSP_TSIP_POLARITY_ACTIVE_HIGH PSP_TSIP_POLARITY_FALLING_EDGE	Active High / Falling edge

3.1.2.12 PSP_TsipClkMode

Following table describes the parameters contained in *PSP_TsipClkMode* enum, which is a parameter TX/RX SIU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_CLK_MODE_2X	2X mode
PSP_TSIP_CLK_MODE_1X	1X mode

3.1.2.13 PSP_TsipEndian

Following table describes the parameters contained in *PSP_TsipEndian* enum, which is a parameter TDMU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_LITTLE_ENDIAN	Little endian
PSP_TSIP_BIG_ENDIAN	Big endian

3.1.2.14 PSP_TsipIntSelect

Following table describes the parameters contained in *PSP_TsipIntSelect* enum, which is a parameter DMATCU configurations and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_INT_XFER_ACK_ONLY	Interrupt asserted on transfer acknowledge only
PSP_TSIP_INT_DELAY_CNT_ONLY	Interrupt asserted on delay count only
PSP_TSIP_INT_EITHER	Interrupt asserted on either transfer acknowledge or delay count
PSP_TSIP_INT_BOTH	Interrupt asserted on both transfer acknowledge and delay count.

3.1.3 TSIP channel params (PSP_tsipChanParams)

The file psp_tsip.h has the **PSP_tsipChanParams** data structure that is passed to GIO_create. The parameters are explained below:

Structure Members	Description
PSP_tsipBitmap	bmA
PSP_tsipBitmap	bmB
Uint8	aCID
Uint8	bCID
PSP_tsipErrCallback	errCb
PSP_tsipFrmCallback	frmCb
PSP_tsipFrmCallback	sfrmCb
Ptr	fCbArg
Ptr	sCbArg

3.1.3.1 PSP_tsipBitmap

Following table describes the parameters contained in *PSP_tsipBitmap structure which contains array of PSP_TsipPcmType[256] enum type*, which is a parameter used to configure bitmaps and details of the same is available in "psp_tsip.h" file:

Structure Members	Description
PSP_TSIP_PCM_DISABLED	Timeslot disabled
PSP_TSIP_PCM_8_BIT_LINEAR	Timeslot is 8-bit linear
PSP_TSIP_PCM_U_LAW	Timeslot is u-law encoded
PSP_TSIP_PCM_A_LAW	Timeslot is a-law encoded

3.1.4 TSIP IO params (PSP_tsipIoParams)

The file psp_tsip.h has the **PSP_tsipIoParams** data structure that is passed to GIO_submit. The parameters are explained below:

Structure Members	Description
PSP_tsipDmaConfig	dmaA
PSP_tsipDmaConfig	dmaB

3.1.4.1 PSP_tsipDmaConfig

Following table describes the parameters contained in *PSP_tsipDmaConfig structure, which is a parameter,* used to configure DMA for A and B contexts and details for the same is available in "psp_tsip.h" file:

Structure Members	Description
Uint32	baseAddr
Uint32	frameAlloc
Uint32	frameSize
Uint32	frameCnt

TSIP API's

This chapter describes the functions, data structures, enumerations and macros for the TSIP driver module.

The following API functions are defined by the GIO module:

GIO_create	Allocate and initialize a TSIP channel object. It will invoke the corresponding IOM function tsip_mdCreateChan.
GIO_delete	De-allocate a TSIP channel object. It will invoke the It will invoke the corresponding IOM function tsip_mdDeleteChan.
GIO_control	Send a control command to the mini-driver. It will invoke the corresponding IOM function tsip_mdControlChan.
GIO_submit	API used to transfer the data with slaves. It will invoke the corresponding IOM function tsip_mdSubmitChan.

3.2 Structures Passed to GIO APIs

3.2.1 Structure passed to GIO_create

The file psp_tsip.h has the **PSP_tsipChanParams** data structure that is passed to GIO_create.

3.2.2 Structure passed to GIO_submit

The file psp_tsip.h has the **PSP_tsipIoParams** data structure that is passed to GIO_submit.

3.2.3 Enumerations for IOCTL

Following are the enumerations passed as command argument while GIO_control call.

IOCTL Command	Description
PSP_TSIP_IOCTL_GET_PID	Get s the peripheral ID version of the TSIP. Command Argument : PID structure <PSP_tsipPID>
PSP_TSIP_IOCTL_START_CHANNEL	Starts the TDMU-DMATCU channel operation
PSP_TSIP_IOCTL_STOP_CHANNEL	Stops the TDMU-DMATCU channel operation immediately. However the TSIP HW will disable the channel only after completion of the current frame
PSP_TSIP_IOCTL_STOP_CHANNEL_EXT	Stops the channel after completion of the current s-frame.
PSP_TSIP_IOCTL_MODIFY_BITMAP_A	Modifies Bitmap configuration-A to select different set of timeslots for different PCM formats: 8-bit linear, A and U law. Note: Modifying bitmap-A when bitmap-A is active will return error (Invalid state). So the application can first check which bitmap is active by using GET_ACTIVE_BITMAP command before invoking this command. Command Argument : Bitmap Configuration <PSP_tsipBitmap>
PSP_TSIP_IOCTL_MODIFY_BITMAP_B	Modifies Bitmap configuration-B to select different set of timeslots for different PCM formats: 8-bit linear, A and U law. Note: Modifying bitmap-B when bitmap-B is active will return error (Invalid state). So the application can first check which bitmap is active by using GET_ACTIVE_BITMAP command before invoking this command. Command Argument : Bitmap Configuration <PSP_tsipBitmap>
PSP_TSIP_IOCTL_GET_ERROR_STATS	Gets the driver maintained error statistics counters value. Command Argument : Error statistics structure <PSP_tsipErrStats>
PSP_TSIP_IOCTL_CLEAR_ERROR_STATS	Clears error statistics counters.
PSP_TSIP_IOCTL_GET_ACTIVE_CONTEXT	Returns the active context (A or B or None). Command Argument: Active Context enum <PSP_TsipActiveContext>
PSP_TSIP_IOCTL_GET_ACTIVE_BITMAP	Returns the active bitmap(A or B or None). Command Argument: Active Bitmap enum <PSP_TsipActiveBitmap>
PSP_TSIP_IOCTL_SET_ACTIVE_BITMAP	Sets the active bitmap to A or B. Command Argument : CID structure for A or B <PSP_tsipCID>
PSP_TSIP_IOCTL_GET_HW_SETUP	Gets the hardware setup of the TSIP port. Command Argument : TSIP HW setup structure <PSP_tsipHwSetup>
PSP_TSIP_IOCTL_SETUP_SIU	Sets up/ Configures SIU module. Command Argument : SIU HW setup structure <PSP_tsipSiuConf>
PSP_TSIP_IOCTL_SETUP_TDMU	Sets up/Configures DMATCU and TDMU modules. Command Argument : SIU HW setup structure <PSP_tsipTdmuConf>
PSP_TSIP_IOCTL_RESET_TDMU	Resets (Software Reset) the DMATCU and TDMU. After TDMU/DMATCU reset, TDMU/DMATCU modules need to be reconfigured by the application by using SETUP TDMU IOCTL command. Enabling SIU/TDMU is done automatically by the driver upon next IOP request
PSP_TSIP_IOCTL_RESET_SIU	Resets (Software reset) the SIU. After SIU reset, SIU module needs to be reconfigured by the application by using SETUP SIU IOCTL command. Enabling SIU/TDMU is done automatically by the driver upon next IOP request.
PSP_TSIP_IOCTL_SET_SEM_TIMEOUT	Change the timeout value used in semaphore pending for flush operation.

3.3 API Definition

3.3.1 GIO_ create

Syntax

```
GIO_Handle GIO_create (
            String name,
            Int mode,
            Int *status,
            Ptr chanParams,
            GIO_Attrs *attrs
            );
```

Parameters

name

The name argument is the name specified for the device when it was created in the configuration or at runtime. It is used to find a matching name in the device table.
Note: strings are case sensitive.

For TSIP drivers the string contains one token '/' followed by the name.

- TSIP driver or port instance
  This identifies the TSIP driver or port instance and this will be typically "*/TSIP0", "/TSIP1"* and so on, where suffix to TSIP denotes instance ID. This string depends on the device registration string given in BIOS driver TCI file.

mode

The mode argument specifies the mode in which the device is to be opened. This will be either IOM_INPUT or IOM_OUTPUT.

status

The status argument is an output parameter that this function fills with a pointer to the status that was returned by the mini-driver.

chanParams

GIO_Create for TSIP requires PSP_tsipChanParams as its channel parameter and details of the same is explained in section 3.2.1.

attrs

The attrs parameter is a pointer to a structure of type *GIO_Attrs*. The number of maximum IO request can be passed using this structure.

Return Value

It returns the handle of type *GIO_Handle* on successful opening of a device. It returns NULL if the device could not be opened.

Description

An application calls *GIO_create* to create and initialize a TSIP driver channel to the driver.

Constraints

This function can only be called after the device has been loaded and initialized.

Example

```
The example below shows creation of Channel for TSIP
GIO_Handle                xmtHandle;
GIO_Attrs                   gioAttrs;

 /*
 * Initialize channel attributes.
 */
gioAttrs.nPackets       = PSP_TSIP_MAX_IOP_REQ;

for (tsCnt=0; tsCnt < PSP_TSIP_NUM_TIMESLOTS; tsCnt++)
{
  chanParams.bmA.type[tsCnt] = PSP_TSIP_PCM_DISABLED;
  chanParams.bmB.type[tsCnt] = PSP_TSIP_PCM_DISABLED;
}
chanParams.errCb = &tsipErrCallback;
chanParams.aCID  = CID_A;
chanParams.bCID  = CID_B;

/* If user wants some function to be called from every frame/sframe period
 from frameISR, he/she has to register some valid function address*/
chanParams.frmCb  = NULL;
chanParams.fCbArg = NULL;
chanParams.sfrmCb = NULL;
chanParams.sCbArg = NULL;

/* Enable all A/B bitmaps to linear */
for(tsCnt=0; tsCnt < FRAME_SIZE; tsCnt++)
{
  chanParams.bmA.type[tsCnt] = PSP_TSIP_PCM_8_BIT_LINEAR;
  chanParams.bmB.type[tsCnt] = PSP_TSIP_PCM_8_BIT_LINEAR;
}

/* Create TX TSIP channel */
xmtHandle = GIO_create(
                    device,
                    IOM_OUTPUT,
                    &stat,
                    &chanParams,
                    &gioAttrs
                    );

if (NULL == xmtHandle)
{
   TSIP_DEBUG("TSIP-TX channel creation <Failed>");

}
else
{
   TSIP_DEBUG("TSIP-TX channel creation <Success>");
}
```

3.3.2 GIO_delete

Syntax

int GIO_delete(GIO_Handle gioChan);

Parameters

Handle of the TSIP driver channel that was created with a call to GIO_create.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

This function call will close the logical channel associated with GIO_create. It will also free the buffers allocated by driver.

Constraints

This function can only be called after the device has been loaded, initialized and created.

Example

The example below shows deletion of Channel for TSIP

```
GIO_Handle chanHandle;

/* chanHandle should be obtained by calling GIO_create here. */


/* Deleting the chanHandle */
Status = GIO_delete(chanHandle);
If (status != IOM_COMPLETED)
{
      printf(" Failed deleting channel \r\n");
}
```

3.3.3 GIO_control

Syntax

status = GIO_control (gioChan, cmd, args);

Parameters

gioChan

Handle of the TSIP driver channel that was created with a call to *GIO_create*.

cmd

The cmd argument specifies the control command

args

The args argument is a pointer to the argument or structure of arguments that are specific to the command being passed.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

An application calls *GIO_control* to send device-specific control commands to the mini-driver.

IOCTL commands available for TSIP driver are available in section 3.2.3

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle chanHandle;


/* channel creation should be done here  using GIO_create.*/

/* Invoking IOCTL using GIO_control */
status = GIO_control(chanHandle, PSP_TSIP_IOCTL_STOP_CHANNEL, NULL);

if (IOM_COMPLETED != status)
{
        printf(" Failed to perform stop channel IOCTL \r\n");
        return;
}
```

3.3.4 GIO_Submit

Syntax

```
status = GIO_submit (        GIO_Handle gioChan,
                             Uns cmd,
                             ptr bufP,
                             Uns* Psize,
                             GIO_AppCallBack* appCallback
                      )
```

Parameters

gioChan

Handle of the TSIP driver channel that was created with a call to *GIO_create*.

cmd

The cmd argument specifies the control command, which can be any of the following

- IOM_READ /* for read operation */
- IOM_WRITE /* for write operation */
- IOM_FLUSH /* for flushing the packets */
- IOM_ABORT /* for aborting the packets */

bufP

The bufP argument is a pointer to the buffer which need to be written or data to read, and this value should be used only when cmd is used as IOM_READ or IOM_WRITE. For other two commands it is NULL.

Psize

The Psize argument is a pointer to the argument of data type size_t, and this value represents the number of bytes of data to be written or to read.

appCallback

The Psize argument is a function pointer, which will be called once the submit operation is completed. Since TSIP works in synchronous mode of operation this parameter will be NULL.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

This function is called by the application to perform the read/write/flush/abort operation.

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle xmtHandle;
PSP_tsipIoParams    txIop1[2];
/* GIO application call back */
GIO_AppCallback     appCb;

 txIop1[0].dmaA.baseAddr = (Uint32)&aTxBuff1_s1[0];
 txIop1[0].dmaA.frameAlloc = FRAME_SIZE + FRAME_OVERHEAD;
 txIop1[0].dmaA.frameCnt   = NUM_FRAMES;
 txIop1[0].dmaA.frameSize  = FRAME_SIZE + FRAME_OVERHEAD;
 /* 2nd Sframe (B) */
 txIop1[0].dmaB.baseAddr = (Uint32)&bTxBuff1_s2[0];
 txIop1[0].dmaB.frameAlloc = FRAME_SIZE + FRAME_OVERHEAD;
 txIop1[0].dmaB.frameCnt   = NUM_FRAMES;
 txIop1[0].dmaB.frameSize  = FRAME_SIZE + FRAME_OVERHEAD;

 appCb.fxn = &tsip_AppCallback; /* App callback function */
 appCb.arg = &xmtHandle;
 sfrmSize = 2u;
 retCode = GIO_submit(xmtHandle,IOM_WRITE,&txIop1[0],&sfrmSize,&appCb);
```

3.3.5 GIO_flush/GIO_abort

Syntax

Status = GIO_flush/GIO_abort (GIO_Handle gioChan)

Parameters

gioChan

Handle of the TSIP driver channel that was created with a call to *GIO_create*.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code.

Description

Similar to GIO_submit with cmd either IOM_FLUSH or IOM_ABORT with all other arguments are being NULL.

GIO_abort aborts all the pending IO requests and returns call back for all IO requests for both READ and WRITE channels with the call back status as IOM_ABORTED.

GIO_flush on READ channel is same as GIO_abort and returned call back status for all read IO requests is IOM_FLUSHED. But GIO_flush on WRITE channel completes all pending write IO requests normally and returns call back for all write requests with call back status as IOM_COMPLETED since it is a normal IO completion.

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle gioChan;

/* channel creation and allocBuffer should be done here */

        status = GIO_flush (gioChan);

        status = GIO_abort(gioChan);
```

4 Example Applications

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

5.1 Writing Applications for TSIP

This section provides guidance to user for writing their own application for TSIP drivers.

5.1.1. File Inclusion

To write sample application user has to include following header files in the application:

1. **std.h**

   This file contains standard data types, macros and structures.

2. **gio.h**

   This file contains GIO layer macros and structures. These macros are wrapper macros to form a wrapper above GIO.

3. **tsk.h**

   This file contains all task module details.

4. **psp_tsip.h**

   This file contains TSIP parameters which are passed to driver at the time of TSIP driver registration with BIOS.

5.2 Sample Applications

5.2.1. Introduction

The sample application is a representative test program. Initialization of TSIP driver is done by calling initialization function from BIOS.

The sample application performs read/write operation to Terminal connected to COM port.

5.2.2. Building the application

Please follow below steps to build sample application:

- Open CCS 3.3 setup. Import proper CCS configuration file. Set the proper CCS Gel file (Refer C6452_BIOS_PSP_Release_Notes.doc for details). Click on "Save & Quit" button and exit the setup.
- Open sample application as mentioned in *"<root>\pspdrivers\system\c6452\bios\evm6452\tsip\build\ c6452_evm_tsip_st_sample.pjt"*.

- Compile this project using Project->Build
- Note: Following Components needs to be linked for successful build and functionality of the application.
  - TSIP
  - PAL_OS
  - SoC specific PAL_SYS
  - EDMA3

5.2.3. *Loading and running the application*

The sample application is loaded and executed via Code composer studio. It is good idea to reset the board before loading Code Composer. The application will print out the status messages and type of functionality the driver performs on the message log.

5.2.4. *Sample TSIP application*

The sample application file "psp_bios_tsip_sample.c" is available in "*<root>\pspdrivers\system\c6452\bios\evm6452\tsip*\src" folder for the TSIP driver package. It demonstrates the sample transmit/receive of multiple IO submit in loopback mode. Please see the file for details. Refer section **"3.1.1 Initialization details"** for configuring TCI file.

5.2.5. *Pragma directives used in the sample application*

The RX and TX buffer are used by the application and need to be cache aligned at 128 bytes.