TSIP BIOS Device Driver

TSIP Architecture/Design Document

Document Version	Author(s)	Date	Comments
0.1	Jayaprakash N	Nov 06, 2006	Initial Draft
0.2	Jayaprakash N	Nov 13, 2006	- Added dynamic IO mechanism support in the design. - Details added/enhanced as per MT-TI review on Nov08. - Added PALOS/PALSYS description(nov16) - Removed static IO mechanism (nov18) - Updated Error codes, IOCTLs (nov19) - Divided HW setup structure into three types (nov21)
0.3	Jayaprakash	Dec 12, 2006	Modified as per TI review comments - DDC objects, flow diagram segregated from IOM
0.4	Jayaprakash	Jan 16, 2008	Added an IOCTL to change flush semaphore-pend timeout value.

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

Table of Contents

List Of Figures

1 System Context

The purpose of this document is to explain the device driver design for TSIP peripheral used in C6452 SoC using DSP/BIOS operating system running on DSP 64+ joule. This driver is aimed at providing support for multiple TSIP instances.

Note: ***The usage of structure names and field names used throughout this design document is only for indicative purpose. These names shall not necessarily be matched with the names used in source code.***

1.1 Terms and Abbreviations

CBA	Common Bus Architecture
CSL	Chip Support Library/Layer
DDC	Device Driver Core
DMATCU	DMA Transfer Control Unit
FSM	Finite State Machine
GIO	General Input/Output
IOCTL	Input/Output Control
IOM	Input/Output Mini driver
IOP	Input/Output Packet
PSP	Platform Support Package
PV	Packet Voice
TDM	Time Division Multiplexing
TDMU	Timeslot Data Management Unit
TSIP	Telecom Serial Interface Port

1.2 References

1.	SPRU616	DSP/BIOS Driver Developer's Guide
2.	TSIP RDD	C6452_tsip_rdd
3.	TSIP Functional Specification	sprueu8_TSIP.pdf

1.3 Hardware

The TSIP device driver architecture presented in this document is situated in the context of C6452 SoC targeted at Packet Voice applications. The driver design is in the context of DSP/BIOS running on DSP 64x+ joule core. The following figure (Figure 1) shows C6452 Architecture shall be used for Packet Voice application.

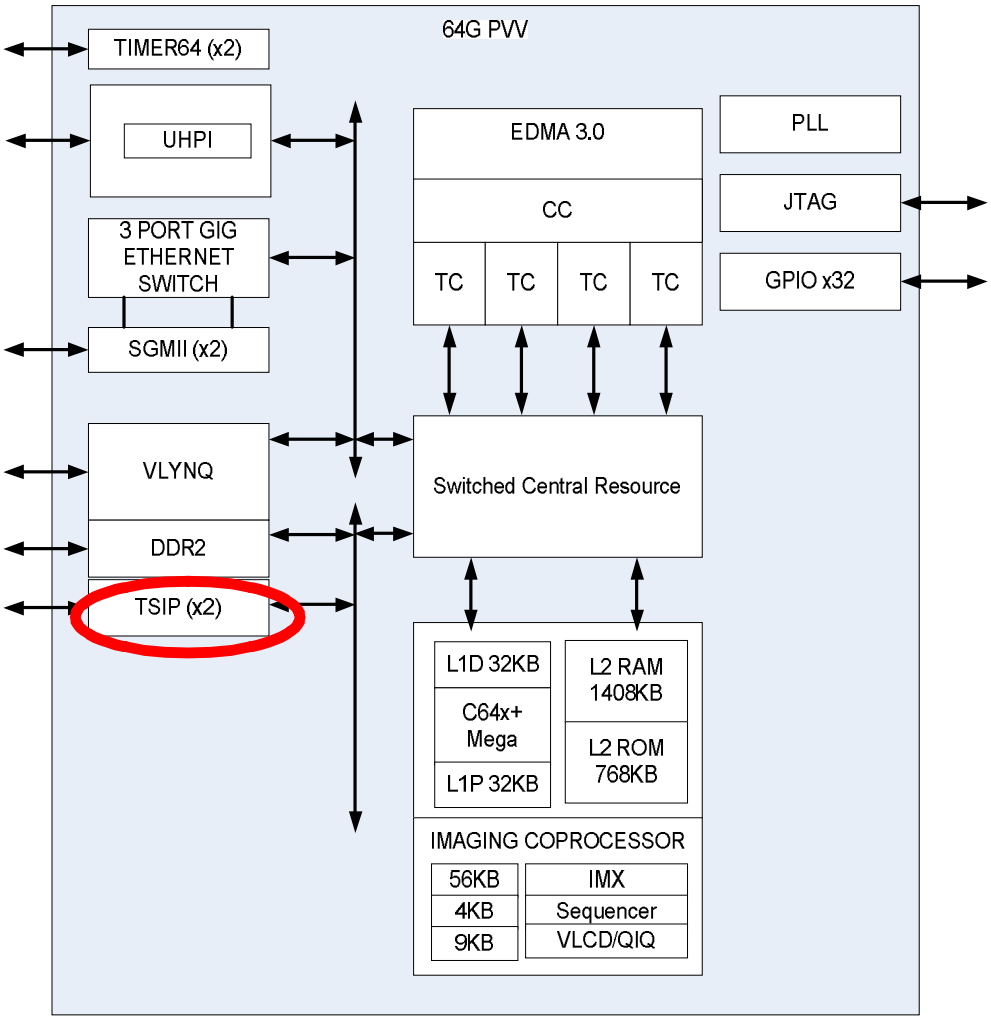

Figure 1 C6452 Block Diagram

The TSIP in C6452 is a multi-link serial interface consisting of a maximum of two transmit data signals (or links), two receive data signals, two frame sync input signals and two serial clock inputs. The module offers supports for a maximum of 256 timeslots for transmit and 256 timeslots for receive.

The following figure shows the TSIP peripheral of Tomahawk SoC.

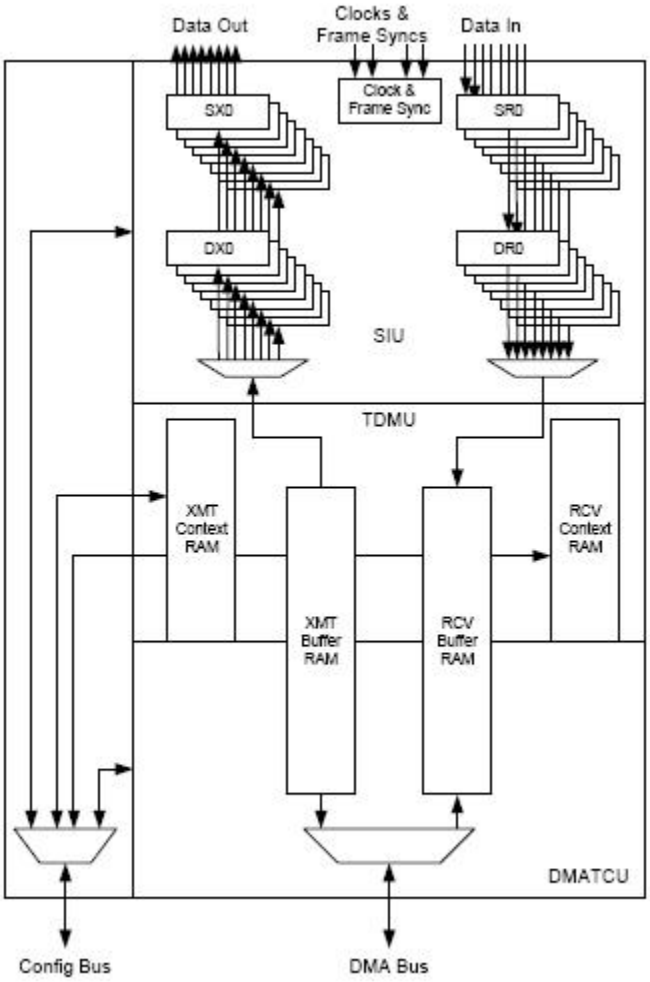

Figure 2 TSIP Block Diagram

The standard serial data rate for each TSIP transmit and receive data signal is 8.192 Mbps. The standard frame sync is a one or more bit wide pulse that occurs once every 125 µs or a minimum of one serial clock period every 1024 serial clocks. At the standard rate and default configuration there are 2-transmit and 2-receive links that are active. Each serial interface link supports up to 128 8-bit timeslots. This corresponds to an H-MVIP or H.110 serial data rate interface. The serial interface clock frequency may be either 16.384 MHz (default) or 8.192 MHz. Typical timeslot occupation is 96 timeslots (DS2) for each serial interface link.

Directly tied to the SIU of the TSIP is a TDMU that selectively packs and unpacks timeslot data for receive and transmit based on a channel timeslot definition. There are 1-transmit and 1-receive TDMU channels in C6452 SoC implementation. Each channel is capable of selecting and transferring all of the possible 256 timeslots in the respective direction.

The TSIP also includes a DMATCU. DMATCU has one to one correspondence with TDMU channels and shares transmit and receive channels of TDMU. There are 1-transmit and 1-receive DMATCU channels in C6452.

1.4 Software

The TSIP mini-driver discussed here is targeted at the C6452 device, running DSP/BIOS on the 64x+ DSP. However the TSIP driver can also be ported to any other OS, with minimal modifications in the OS specific section of the driver. More details can be found in the later part of this section.

1.4.1 Operating Environment and dependencies

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

1.4.2 System Architecture

The device driver described here is part of an IOM mini-driver. That is, it is implemented as the lower layer of a two layer device driver model and is a super set of all other driver layers. The upper layer is called the class driver and is the generic DSP/BIOS GIO module. 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 3 shows the overall DSP/BIOS device driver architecture. For more information about the IOM device driver model, see the *DSP/BIOS Device Driver Developer's Guide (SPRU616).*

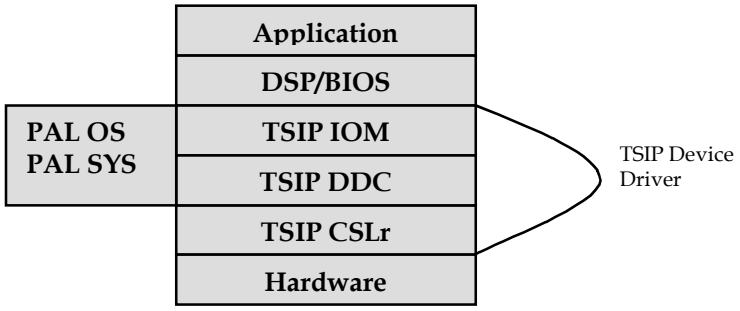

Figure 3 Device driver Layer

This device driver can be used as a general-purpose stand-alone mini-driver to interface with the TSIP peripheral on C6452.

Please refer PSP framework manual to get to know more details about the various device driver layers.

1.5 Component Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified. Refer to TSIP device driver API reference documentation for the complete details of the APIs.

1.5.1 IOM Interface

The IOM constitutes the Device Driver Manifest to Application. The user may not look into IOM interface, especially the upper-edge services exposed to the Application/OS. All other interfaces discussed later in this document are more of interest to people developing/maintaining the device driver.

The IOM can be modified to re-target Driver and/or customize to specific Apps framework by doctoring the upper-edge services.

The *tsip_mdBindDev ()* populates static settings in driver object creates the necessary interrupt handler, attaches the Driver Core interfaces. All these operations in effect, constitute the "loading" of TSIP Driver implementation. The tsip_mdUnbindDev () constitutes the "Un-loading" of the TSIP driver. The IOM mini-driver implements the following API interfaces to the class driver.

S.No	IOM Interfaces	Description
1	tsip_mdBindDev ()	Allocates and configures the TSIP port specified by devId
2	tsip_mdUnbindDev ()	Closes the TSIP device from use.
3	tsip_mdCreateChan ()	Creates a communication channel in specified mode to communicate data between the application and the TSIP device instance.
4	tsip_mdDeleteChan ()	Frees a channel and all its associated resources.
5	tsip_mdControlChan ()	Implements the IOCTLS for TSIP IOM mini divers.
6	tsip_mdSubmitChan ()	Submit an I/O packet to a channel for processing.

1.5.2 DDC Interface

DDC implements the core device driver layer and it provides standard abstract interfaces to the upper layers as per the PSP framework standards architecture.

The DDC layer APIs of TSIP driver can be called directly from the application or from the OS adaptation layer. So this can be ported to any OS without any modification.

The following basic interfaces are implemented and exposed to the IOM layer by the DDC layer of TSIP driver.

S.No	DDC Interfaces	Description
1	PSP_tsipCreate ()	Initialize/Setup the TSIP hardware with the given configuration parameters.
2	PSP_tsipDelete ()	Does the reverse of *PSP_tsipCreate.*
3	PSP_tsipOpen ()	Configure TSIP's TX/RX DMATCU channels.
4	PSP_tsipClose ()	Does the reverse of *PSP_tsipOpen.*
5	PSP_tsipIoctl ()	Perform input/output control on TSIP Hardware.
6	PSP_tsipSubmit()	Submits IOP requests to perform input/output
7	PSP_tsipAbort ()	Aborts the channel operation
8	PSP_tsipFlush ()	Flushes the channel operation

1.5.3 CSLR Interface

The CSL register interface (CSLr) provides register level implementations. CSLr is used by the DDC to configure TSIP registers. CSLR is implemented as a header file that has CSLR macros and register overlay structure.

1.6 Design Philosophy

This device driver is written in conformance to the DSP/BIOS IOM device driver model and handles communication to and from the TSIP hardware, and uses its internal DMA to transfer the data.

1.6.1 The Port and Channel Concept

The IOM model provides the concept of the *Port* and *Channel* for the realization of the device and its communication path as a part of the driver implementation. The *Port Object* maintains the state of the TSIP device or an instance. The *port* can also be called as *instance* or *device* and the names can be used interchangeably. C6452 contains two instances of TSIP, and the driver for this needs to maintain two port objects. This following figure shows the generic port-channel-hardware mapping for TSIP driver.

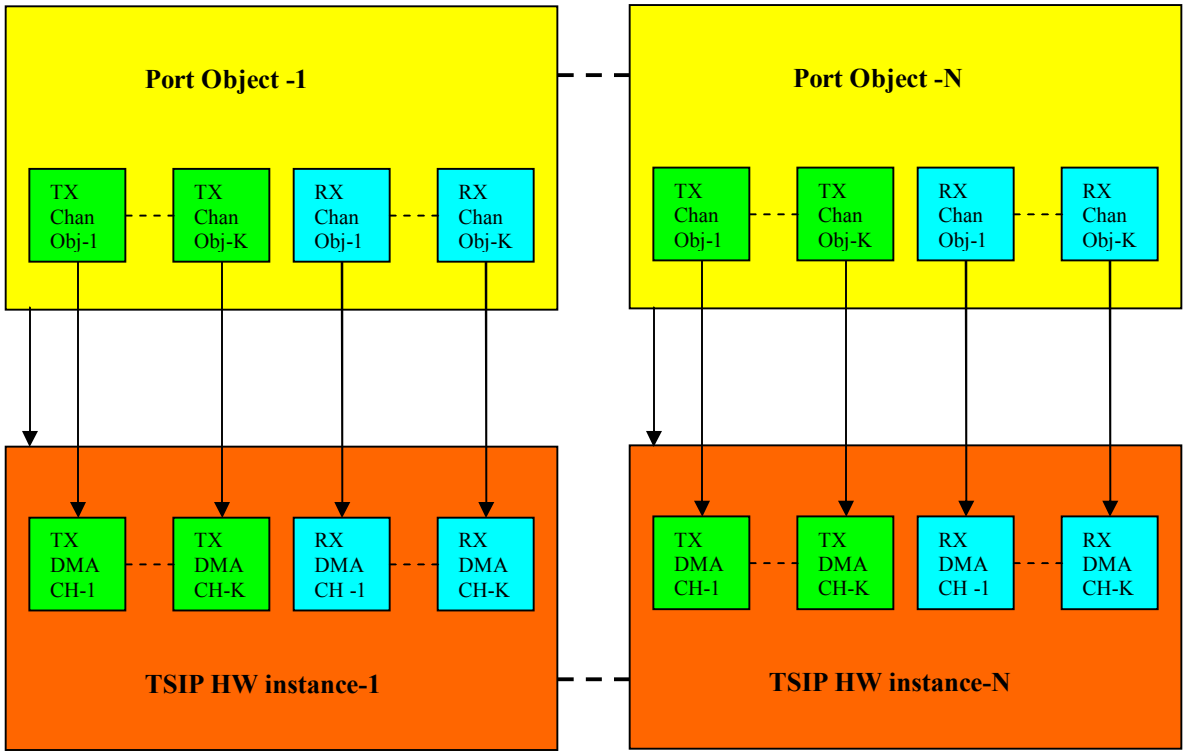

Figure 4 IOM Port-Channel to TSIP HW mapping

Channel represents the communication path between the application and the device driver. A channel can be operating in input or output mode or both. The *Channel Object* maintains the state of data communication path of the device. The TSIP DMATCU hardware has DMA channels to provide movement of data to and from the Hardware. Each TX and RX DMA channels can be realized as separate communication path and hence a separate IOM channel. So the number of IOM channel objects in TSIP depends on the number of TX/RX DMA channels that the TSIP has. In C6452, implementation there is one TX and one RX DMA channel.

The port object contains placeholders for all channel objects for TX and RX. The lifetime of the channel is between its creation using *tsip_mdCreateChan* and its deletion using *tsip_mdDeleteChan*. The application can delete a channel after it has finished with that operation. This channel can then be re-allocated.

1.7 Design Constraints

TSIP mini-driver imposes the following constraint(s).

- 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.
- In general frame buffers are non-cacheable. It is expected that the application must allocate super-frame buffers in a non-cacheable memory region. However, the application wants to provide super-frame buffer from cacheable region, TSIP driver needs to be built with PSP_TSIP_CACHEABLE_FRAMEBUFFER preprocessor macro enabled which has confined cache invalidate/flush functionalities to maintain data cache coherency.
- 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.

2 TSIP Driver Software Architecture

This section details the data structures used in the TSIP mini-driver and the interface it presents to the GIO layer. A diagrammatic representation of the mini driver functions is presented and then the usage scenario is discussed in some more details.

2.1 Static View

The driver is designed keeping a device, also called port, and channel concept in mind. The two instances of TSIP are treated as two devices, which each can have a maximum of two channels for C6452 SoC. In the two channel mode - one as receive and one as transmit channel, each channel can be used by a task independent of the other.

This driver uses two internal data structures, a port object and a channel object, to maintain its state during execution. The TSIP peripheral needs the port instance to maintain its state. The channel object holds the IOM channel state during execution. These are explained in greater detail in the following *Data Structures* sub-section. The following figure shows the static view of C6452 TSIP driver for one instance.

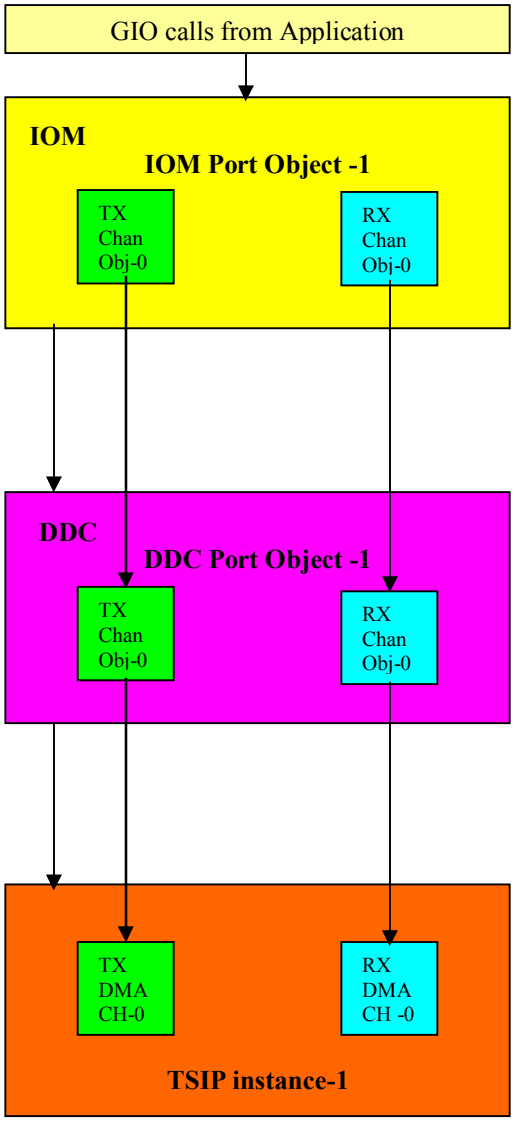

Figure 5 C6452 TSIP driver static view

2.1.1 Data Structures

The mini-driver employs the PortObj and ChannelObj structures to maintain state of the port and channel respectively.

In addition, the driver has two other structures defined – the device params and channel params. The device params structure is used to pass on data to initialize the driver during DSP-BIOS initialization. The channel params structure is used to specify required characteristics while creating a channel. The IO params structure is used to specify memory buffers to do IO transfers.

The following sections provide major data structures maintained by IOM, DDC and PSP interface. For more details about IOM and DDC data structures and their usage can be found in the API reference guide.

2.1.1.1 The Device Params – PSP interface

S.No	Structure Elements (PSP_tsipHwSetup)	Description
1	*siu*	SIU configurations.
2	*tdmu*	TDMU-DMATCU configurations.

2.1.1.2 The Channel Params – PSP interface

The channel parameter structure is passed when creating a channel. This channel parameter contains the bitmap configuration for A and B contexts, the default frame buffer configuration for A and B contexts for the TSIP DMATCU channel that is going to be created. If the user passes NULL to DMA configurations, then the driver will stop the TX/RX engine, when there is no IOP pending. See about IOP operation in section 2.2.2 Input/Output using TSIP driver.

S.No	Structure Elements (*PSP_tsipChanParams)*	Description
1	bmA	Bitmap configuration for A context. (*PSP_tsipBitmap*)
2	bmB	Bitmap configuration for B context.(*PSP_tsipBitmap)*
3	aCID	Channel Identfier (CID) for "A" bitmap/context. lsb should be 0
4	bCID	Channel Identfier (CID) for "B" bitmap/context. lsb should be 1

5	errCb	Pointer to error callback function. (PSP_tsipErrCallback)
6	frmCb	Frame interrupt callback function pointer. If registered, it will be called every frame interval (from frame interrupt). It can be used for housekeeping purpose by the application. The assertion type of frame interrupt depends on the txFint/rxFint (PSP_TsipIntSelect) param.
7	sfrmCb	Super frame interrupt callback function pointer. If registered, it will be called every super frame interval. It can be used for housekeeping purpose by the application. The assertion type of super frame interrupt depends on the txSint/rxSint (PSP_TsipIntSelect) param.
8	fCbArg	Frame interrupt callback argument
9	sCbArg	Super Frame interrupt callback argument

2.1.1.3 PSP_tsipSiuConf – PSP interface

S.No	Structure Elements (PSP_tsipSiuConf)	Description
1	*clkSel*	Dual clock or Redundant clock switch. In dual clock mode TX and RX operates on independent clock.
2	*tx*	TX SIU configuration. (PSP_tsipTxSiuConf).
3	*rx*	RX SIU configuration. (PSP_tsipRxSiuConf).
4	*loopBack*	Loopback mode selection

2.1.1.4 PSP_tsipTdmuConf – PSP interface

S.No	Structure Elements (PSP_tsipTdmuConf)	Description
1	*endian*	DMU: Endianness for data packed/unpacked in the chan buffer
2	*maxPrio*	DMATCU: Max transfer priority[0(highest)-7(lowest)]
3	*prio*	DMATCU: Transfer priority [0(highest)-7(lowest)].
4	*txFint*	DMATCU: TX frame interrupt selection
5	*rxFint*	DMATCU: RX frame interrupt selection
6	*txSint*	DMATCU: TX super frame interrupt selection
7	*rxSint*	DMATCU: RX super frame interrupt selection
8	*txFintDly*	DMATCU: Tx transfer frame interrupt delay
9	*rxFintDly*	DMATCU: Rx transfer frame interrupt delay

2.1.1.5 Timeslot Bitmap Configuration – PSP interface

TSIP driver exposes a data structure to the application to enable/disable timeslots of all possible 256 timeslots provided by the TSIP SoC per direction. This structure can be passed while creating a channel (tsip_mdCreateChan) and through IOCTL in order for the application to change the bitmap after channel creation.

S.No	Structure Elements (PSP_tsipBitmap)	Description
1	*pcmType*	Type of PCM data format to select disabled, linear, A-law, U-law options.

2.1.1.6 TSIP DMA channel buffer structure – PSP interface

TSIP driver exposes a data structure to the application to configure the frame buffer configuration for both A and B contexts for transmit and receive separately. C6452 is having one transmit, one receive channel buffers, and each has ping-pong buffers. This configuration is passed from application while creating a channel and submitting an I/O request. An IOCTL command also can be provided in order for the application to change the frame buffer configuration after channel creation.

S.No	Structure Elements (*PSP_tsipDmaConfig)*	Description
1	*baseAddr*	Timeslot number
2	*frameAlloc*	Frame buffer allocation
3	*frameSize*	Size of each frame
3	*frameCnt*	Number of frames in the buffer

2.1.1.7 TSIP IO params structure – PSP interface

The IO parameters structure is used to pass the TSIP DMA configuration for both A and B context in a single GIO_submit call from the application. The application can specify the number of super frames it is needed from the driver in a single request using the size argument. The minimum size is 1 (which is 2 super frames A and B contexts). The application has to allocate N size of PSP_tsipIoParams if it sets the size as N.

S.No	Structure Elements (*PSP_tsipIoParams)*	Description
1	dmaA	DMA channel configuration for A context. *(PSP_tsipDmaConfig)*
2	dmaB	DMA channel configuration for B context. *(PSP_tsipDmaConfig)*

2.1.1.8 The Device (Port) Object - IOM

S.No	Structure Elements (*TSIP_portObj)*	Description
1	*portNumber*	Preserve port or instance number of TSIP
2	*params*	*PSP_tsipHwSetup* handle for initial port configuration
3	*txChan[]*	Holds all transmit channel objects for this port of type *TSIP_chanObj*
4	*rxChan[]*	Holds all receive channel objects for this port of type *TSIP_chanObj*

2.1.1.9 The Channel Object - IOM

S.No	Structure Elements (*TSIP_chanObj)*	Description
1	*chanId*	Preserve the channel number.
2	*inUse*	Keeps track of whether channel is already in use or not.
3	*mode*	Channel mode of operation: Input or Output.
4	*port*	Pointer to device port *TSIP_portObj* configuration structure.
5	*cbFxn*	IOM callback function.
6	*cbArg*	IOM callback argument.

2.1.1.10 The Device (Port) Object - DDC

S.No	Structure Elements (*DDC_tsipDevObj)*	Description
1	*devId*	Device or instance ID of TSIP
2	*txChan[]*	Holds all transmit channel objects for this port of type *TSIP_chanObj*
3	*rxChan[]*	Holds all receive channel objects for this port of type *TSIP_chanObj*
4	*isErrRegd[]*	Flag to say whether tx-rx[n] errInt is registered or not
5	*pReg*	Register overlay pointer for this instance
6	*appHandle*	Application Handle

2.1.1.11 The Channel Object - DDC

S.No	Structure Elements (*DDC_tsipChanObj)*	Description
1	*status*	Channel status whether free or allocated
2	*chanId*	Preserve the channel number.
3	*mode*	Channel mode of operation: Input or Output.
4	*drvCb*	Driver callback function for this channel
5	*cbArg*	Driver callback argument
6	*errCb*	Error callback function for this channel
7	*pDev*	Pointer to Device Obj
8	*queueFreeList*	List containing the Free IO Packets

9	*queueReqList*	List containing the IOP pending requests, received from the application
10	*iop[]*	IOP array of maximum supported requests
11	*currIop*	Current IOP packet being serviced
12	*iopSubmitCnt*	Number of submitted pending IOPs
13	*isABcfgReqd*	Flag to tell both A and B context config. Reqd
14	*isDisExtIoctl*	Flag to tell DISABLE CHANNEL EXT ioctl is invoked
15	*isEnabled*	Flag to tell whether channel is enabled/stopped
16	*errStats*	Error statistics counters
17	*pTCELreg*	TDMU channel error log registers overlay
18	*pDCreg*	DMATCU channel registers overlay
19	*pTCbm*	TDMU channel bitmaps overlay
20	*pTCbuff*	TDMU channel buffers overlay

2.2 Dynamic View

2.2.1 The Execution Threads

The TSIP device driver operation involves following execution threads:

BIOS thread: Function to load and un-load TSIP driver will be under BIOS OS initialization.

Application thread: Creation of channel, Control of channel, deletion of channel and processing of TSIP frame data will be under application thread.

Interrupt context: Processing TX/RX super-frame completion interrupts, and Error interrupts and notifies to application through Call back function.

2.2.2 Input / Output using TSIP driver

In TSIP the TDM data comes at every 125 microseconds as a frame. The application can configure number of frames in a super frame during TSIP initialization. The application can configure the timeslots configuration (bitmaps) for two different A and B super-frame contexts. The bitmap details for A and B context is configured during channel creation.

TSIP driver buffers super-frame data into different memory buffers provided by the application using IOP submit request.

The application can perform IO operation using GIO_submit () calls (corresponding IOM function is tsip_mdSubmitChan ()) to receive number of super-frames for A and B channels in a single request. The configuration for memory buffer address and size of number of requested super-frames should be passed as an argument to the GIO_submit call.(see PSP_tsipIoParams).

The TSIP channel transfer is enabled upon submission of the IO request. Once the IOP is submitted, the driver configures the DMA with appropriate configuration from the IOP for A and B and enables the channel. Once the requested numbers of super-frames have been received or transmitted, the driver will notify the IOP completion to the application through callback function from super-frame interrupt. Then the driver configures DMA channel with the next pending IOP buffer details and starts working on that. The application should ensure that enough number of IO requests is pending at anytime, to get the continuous super-frame data from the driver. In case no IOP is pending, the driver stops the channel after completion of the current IOP.

The following shows the IO receive operation. IO transmit is operation same as IO receive operation.

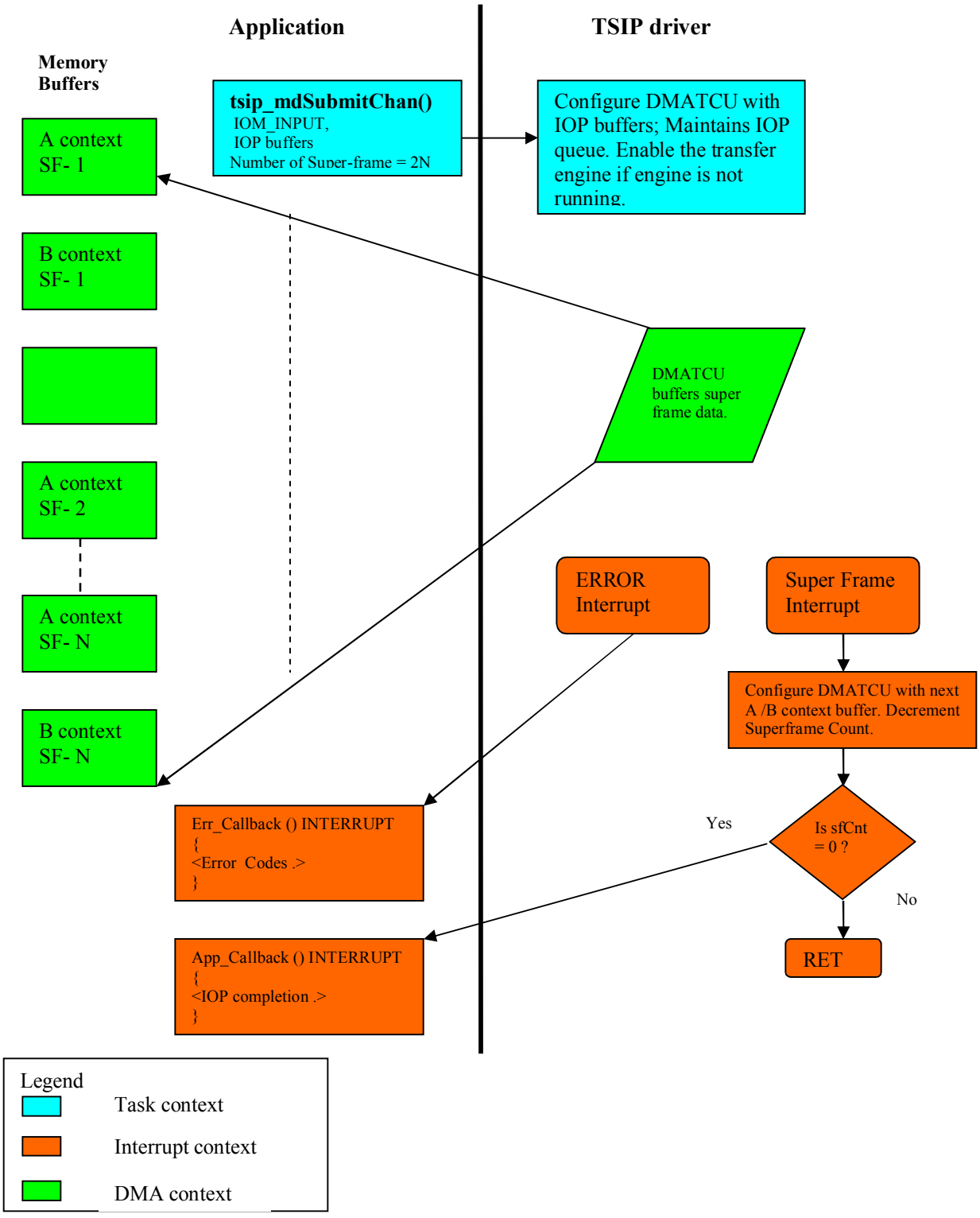

Figure 6 TSIP driver dynamic IO receive operation

2.2.3 Functional Decomposition

2.2.3.1 tsip_mdBindDev

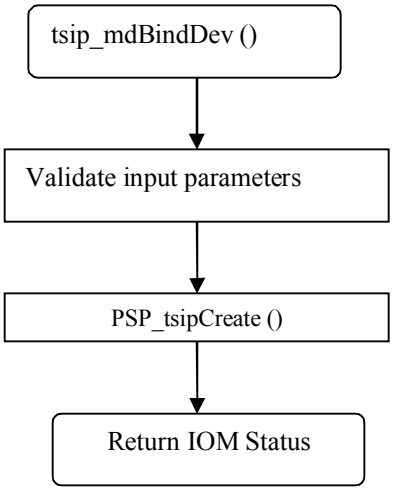

Figure 7 tsip_mdBindDev () flow diagram

What is required from the application?

- Valid structure to setup the TSIP
- Valid device parameters structure

During *tsip_mdBindDev*, the mini-driver has access only to pointers of device params. Memory for these structures is to be allocated outside the driver by the application.

2.2.3.2 tsip_mdUnBindDev

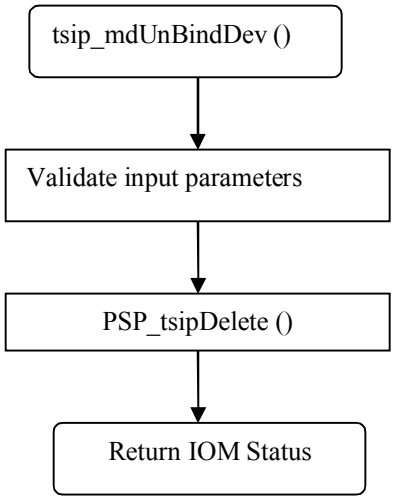

Figure 8 tsip_mdUnBindDev () flow diagram

The *tsip_mdBindDev* and *tsip_mdUnbindDev* functions are called by the DSP-BIOS if driver is created using TCF configuration. Otherwise these functions are called by device driver BIOS APIs like *DEV_createDevice ()* etc. Refer BIOS GIO/IOM model device driver guide for more details. These functions will not be used by the application directly to interface with the TSIP driver.

2.2.3.3 tsip_mdCreateChan

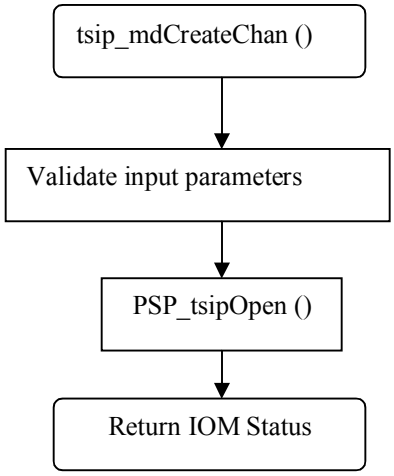

Figure 9 tsip_mdCreateChan () flow diagram

The application has to pass the valid information about configuring the frame buffer and bitmaps details. The memory allocation for frame buffer and bitmap should be done outside the TSIP driver by the application.

2.2.3.4 tsip_mdDeleteChan

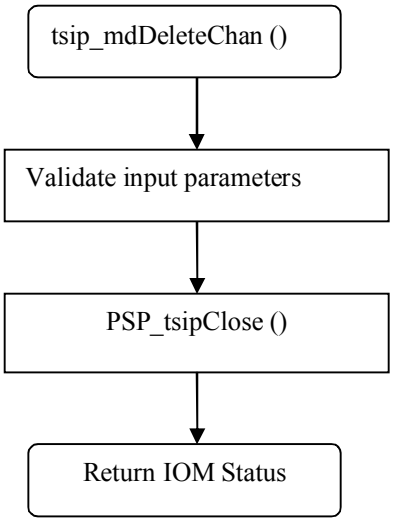

Figure 10 tsip_mdDeleteChan () flow diagram

2.2.3.5 tsip_mdControlChan

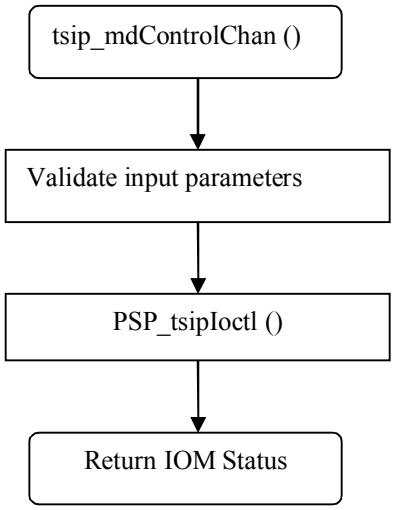

Figure 11 tsip_mdControlChan () flow diagram

2.2.3.6 tsip_mdSubmitChan

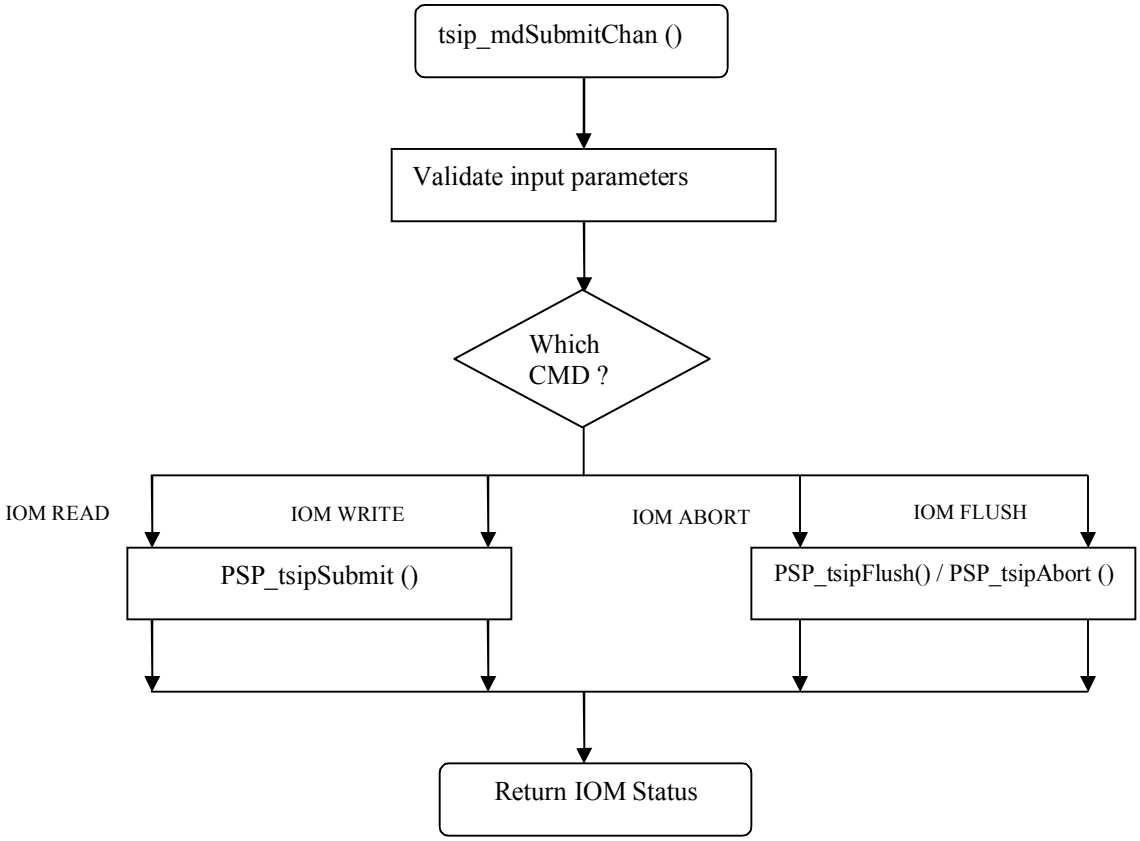

Figure 12 tsip_mdSubmitChan () flow diagram

2.2.3.7 PSP_tsipCreate

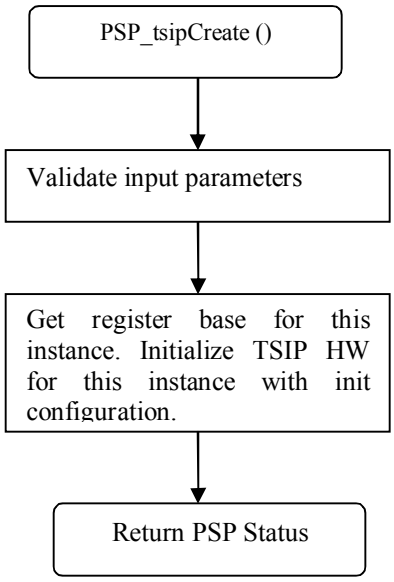

Figure 13 PSP_tsipCreate () flow diagram

2.2.3.8 PSP_tsipDelete

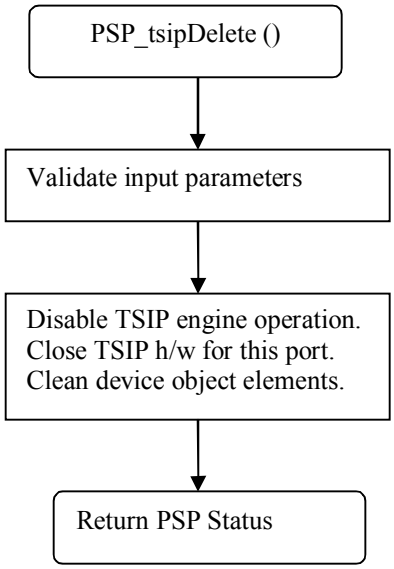

Figure 14 PSP_tsipDelete () flow diagram

2.2.3.9 PSP_tsipOpen

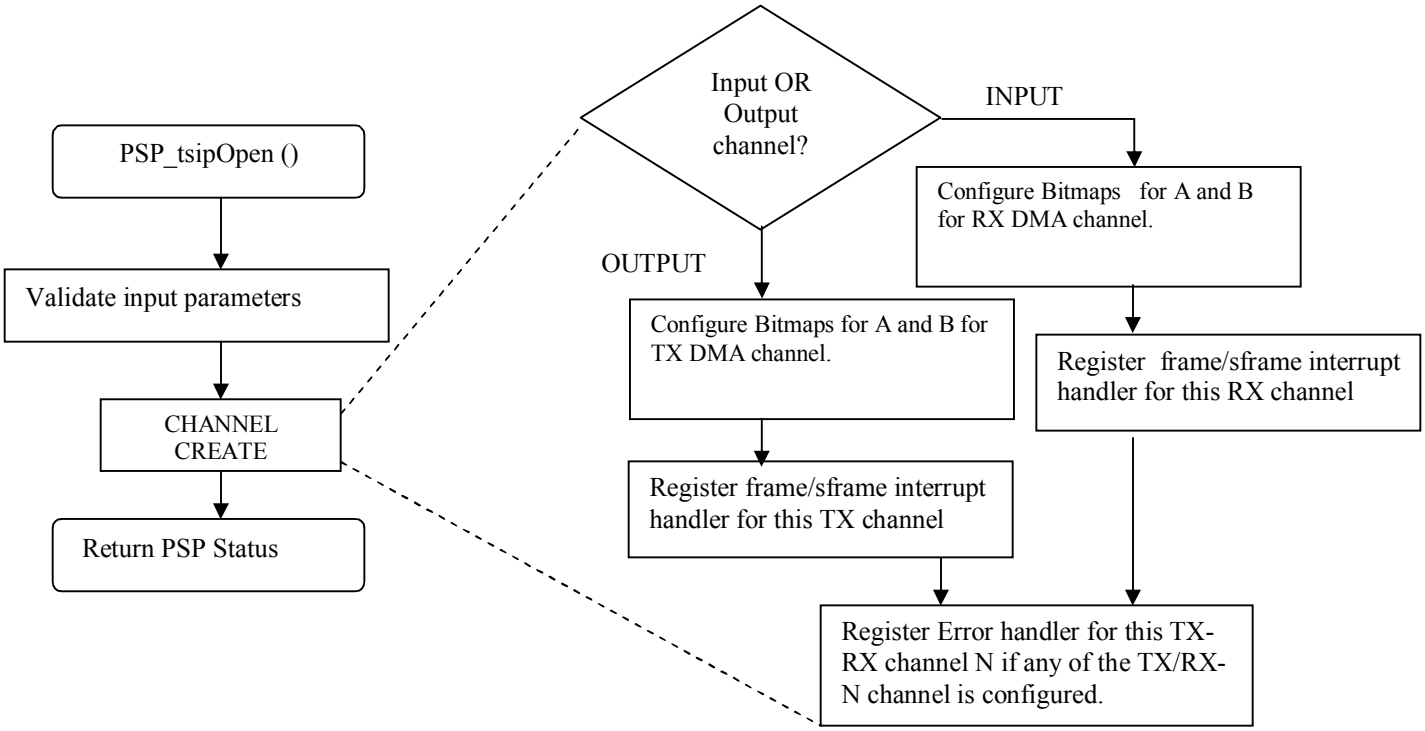

Figure 15 PSP_tsipOpen () flow diagram

2.2.3.10 PSP_tsipClose

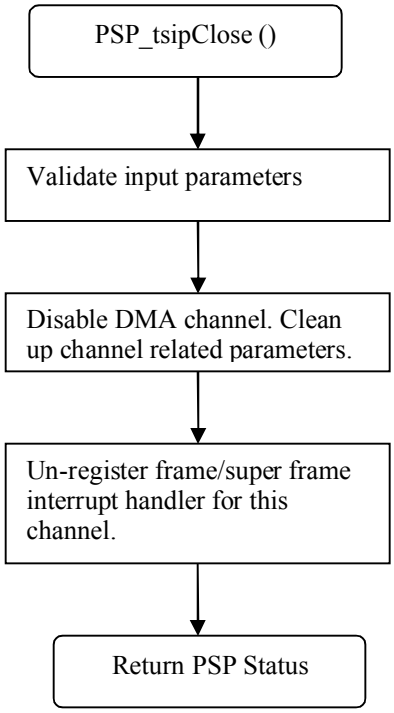

Figure 16 PSP_tsipClose () flow diagram

2.2.3.11 PSP_tsiploctl

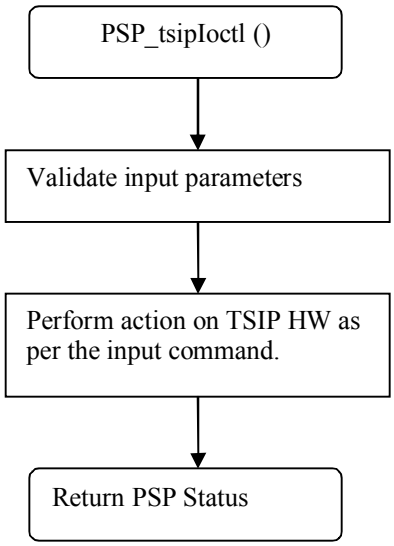

Figure 17 PSP_tsiploctl () flow diagram

2.2.3.12 PSP_tsipSubmit

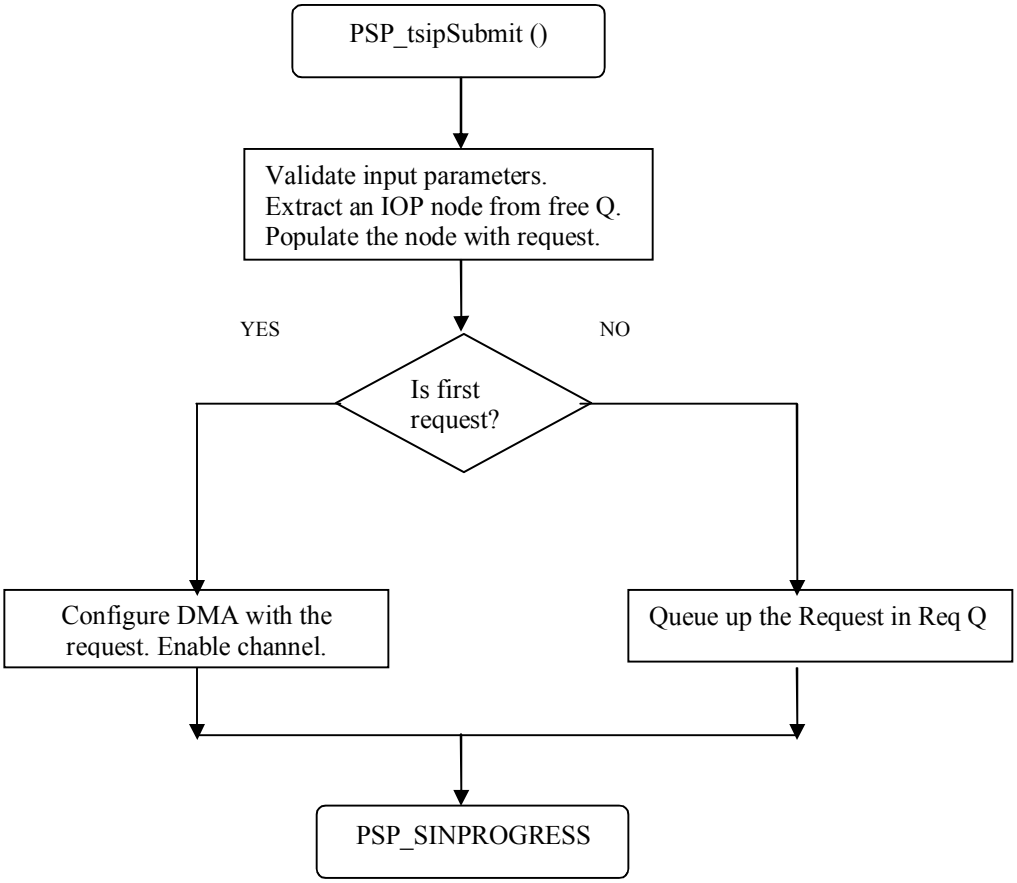

Figure 18 PSP_tsipSubmit () flow diagram

2.2.3.13 PSP_tsipAbort

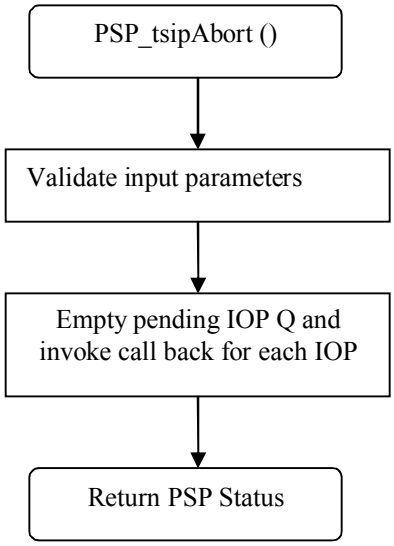

Figure 19 PSP_tsipAbort () flow diagram

2.2.3.14 PSP_tsipFlush

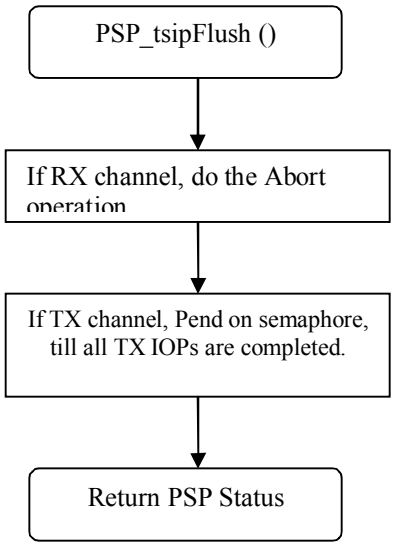

Figure 20 PSP_tsipFlush () flow diagram

2.2.3.15 ddc_ISR

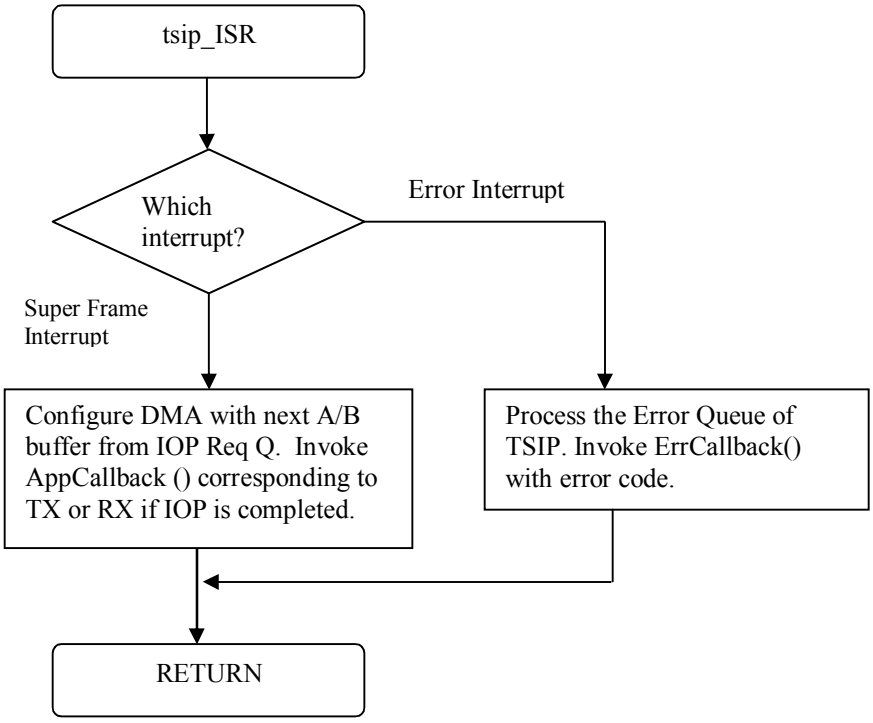

Figure 21 tsip_ISR () flow diagram

3 APPENDIX A – IOCTL commands

The application can perform the following IOCTL on the channel. All commands shall be applicable for both TX and RX channels.

S.No	IOCTL Command	Description
1	PSP_TSIP_IOCTL_GET_PID	Gets the Peripheral Version ID of the TSIP
2	PSP_TSIP_IOCTL_START_CHANNEL	Starts the TDMU-DMATCU channel operation
3	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.
4	PSP_TSIP_IOCTL_STOP_CHANNEL_EXT	Stops the TDMU-DMATCU operation only after completion of the current super frame.
5	PSP_TSIP_IOCTL_MODIFY_BITMAP_A	Modifies Bitmap configuration-A
6	PSP_TSIP_IOCTL_MODIFY_BITMAP_B	Modifies Bitmap configuration-B
7	PSP_TSIP_IOCTL_GET_ERROR_STATS	Gets the driver maintained error statistics counters
8	PSP_TSIP_IOCTL_CLEAR_ERROR_STATS	Clears error statistics counters.
9	PSP_TSIP_IOCTL_GET_ACTIVE_CONTEXT	Returns the active context (A or B or None)
10	PSP_TSIP_IOCTL_GET_ACTIVE_BITMAP	Returns the active bitmap(A or B or None)
11	PSP_TSIP_IOCTL_SET_ACTIVE_BITMAP	Set the active Bitmap(channel config) to A or B
12	PSP_TSIP_IOCTL_SETUP_SIU	Setup/Configure SIU module
13	PSP_TSIP_IOCTL_SETUP_TDMU	Setup/Configure TDMU-DMATCU module
14	PSP_TSIP_IOCTL_GET_HW_SETUP	Returns the TSIP entire hardware setup.
15	PSP_TSIP_IOCTL_RESET_SIU	Resets(Soft) SIU module.
16	PSP_TSIP_IOCTL_RESET_TDMU	Resets(Soft) TDMU and DMATCU modules.
18	PSP_TSIP_IOCTL_SET_SEM_TIMEOUT	Change the timeout value used in semaphore pending for flush operation.

4 APPENDIX B – Error Codes

S.No	Error Code	Description
1	PSP_TSIP_ERROR_RX_ORUN_BEGIN	Receive overrun at beginning of the frame
2	PSP_TSIP_ERROR_RX_ORUN	Receive overrun during the frame
3	PSP_TSIP_ERROR_RX_BITMAP	Receive overrun – Enabled bitmap greater than size.
4	PSP_TSIP_ERROR_RX_INVALID_SIZE	Receive size less than 8
5	PSP_TSIP_ERROR_TX_CONFIG_MISMATCH	Transmit configuration ID mismatch
6	PSP_TSIP_ERROR_TX_URUN_BEGIN	TX under run at beginning of the frame
7	PSP_TSIP_ERROR_TX_URUN	Transmit under run during the frame
8	PSP_TSIP_ERROR_TX_BITMAP	TX under run – Enabled bitmap greater than size.
9	PSP_TSIP_ERROR_TX_INVALID_SIZE	Transmit size less than 8
10	PSP_TSIP_ERROR_CBA_READ	CBA read status error
11	PSP_TSIP_ERROR_CBA_WRITE	CBA write status error
12	PSP_TSIP_ERROR_UNKNOWN	Unknown error.