DSP/BIOS™ LINK

LINK DRIVER

LNK 012 DES

Version 1.11

This page has been intentionally left blank.

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

This page has been intentionally left blank.

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

1.1 Purpose and Scope

This document describes the design of the Link Driver component of DSP/BIOS™ LINK. It defines the main functions and data structures used in the implementation of the Link Driver component. It also describes how each function is implemented.

This document is intended for developers implementing the Link Driver component of DSP/BIOS™ LINK. Developers implementing new link driver(s) can also use it as a reference.

1.2 Terms and Abbreviations

Term	Definition or Explanation
ARM	Advanced RISC Machines (ARM Ltd's RISC Processor)
CHIRP	Channel IO Request Packets
DSP/BIOS™	Built In OS for DSP (TI's proprietary OS)
GPP	General Purpose Processor
Link Driver/LDRV	Link Driver component of DSP/BIOS™ Link.
PMGR	Processor Manager component of DSP/BIOS™ Link.
SHM	Shared Memory Driver

1.3 References

1.	LNK 002 ARC	DSP/BIOS™ LINK High Level Architecture Version 1.02 dated JUL 15, 2003

1.4 Overview

Link Driver consists of the following subcomponents:

1. LDRV_PROC

   This subcomponent provides APIs to access and control the target DSP(s). It also maintains the current state of the target DSP(s).

2. LDRV_CHNL

   This subcomponent provides APIs to transfer data between the GPP and the DSP. It allocates and de-allocates the user buffers while opening and closing the channel. During the data transfer, it is responsible for moving the buffers between FREE, REQUESTED and COMPLETED lists.

3. LDRV_IO

   This subcomponent acts as glue between the LDRV_CHNL subcomponent and the physical link driver. It uses the function pointer interface exported by the link driver to communicate with the Link Driver. The map between the channel id and the underlying link id is maintained by this subcomponent.

4. DSP

   This subcomponent encapsulates physical hardware access to communicate with the target DSP. Services of this subcomponent are exported by a function pointer interface. This allows other subcomponents in LDRV to interact with multiple DSPs. The integration of a DSP into the system is also simple.

   This subcomponent is designed to be independent of the rest of the sub-system. Applications that do not need the PROC and CHNL abstractions provided by Processor Manager (PMGR provides) can directly use only the DSP subcomponent.

5. The Physical Link Driver

   This subcomponent encapsulates low-level data communication between the GPP and the DSP over a physical link driver. Services of this subcomponent are exported by a function pointer interface. This allows LDRV_IO subcomponent to interact with multiple physical link drivers. The integration of a new link driver into the system is also simple.

   Figure 1 illustrates the subcomponents listed above and their interaction.

   Figure 2 illustrates an application using the DSP subcomponent directly to communicate with the target DSP.

   Figure 3 is a sequence diagram illustrating a typical interaction between the Processor Manager & Link Driver.

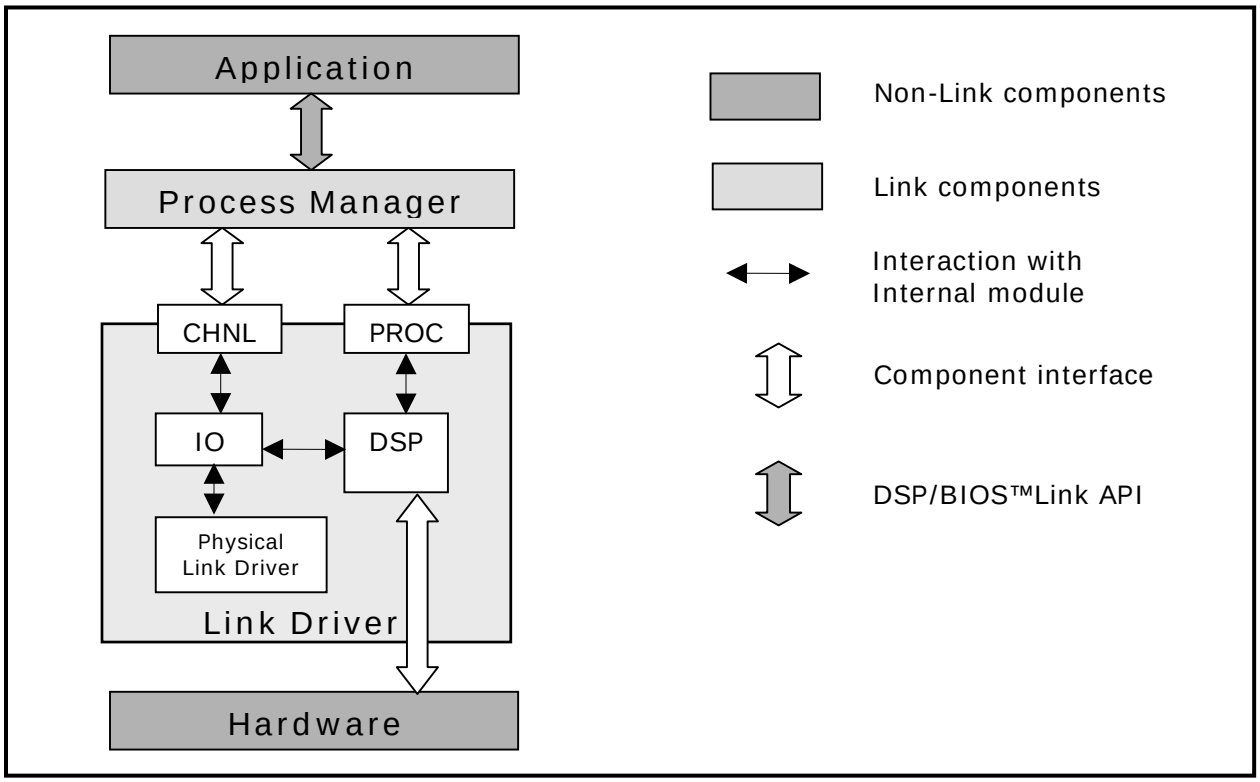

Figure 1. Organization and Typical Interactions of Link Driver Modules

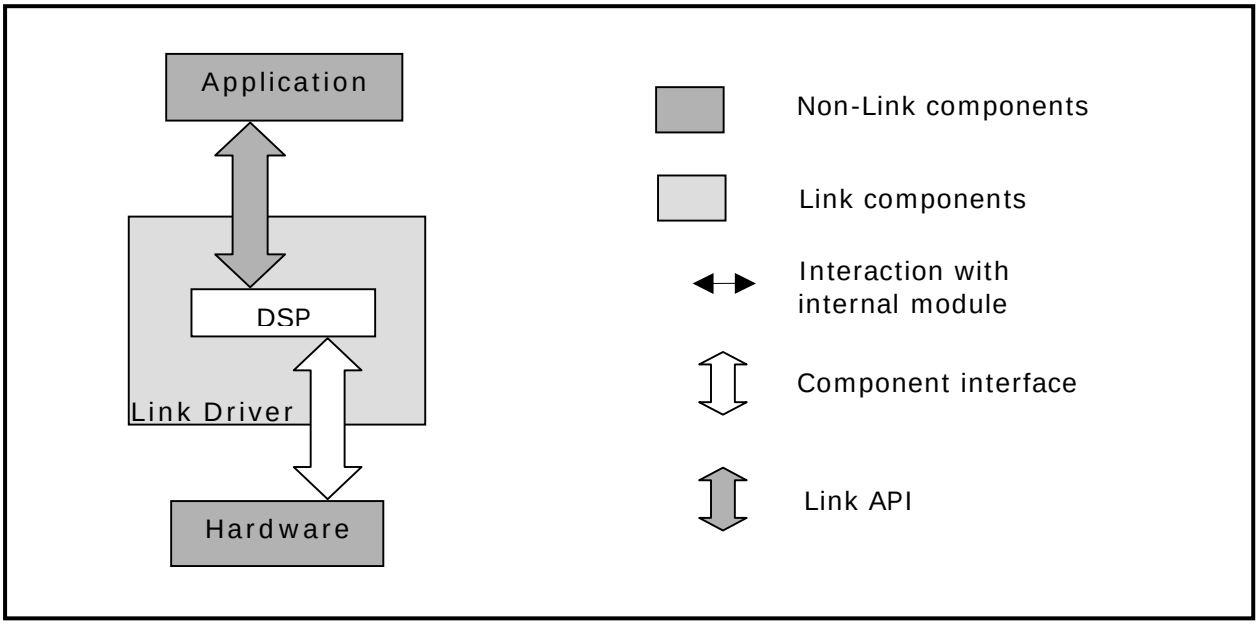

Figure 2. Scaled-down Version of Link Driver With DSP Component Only

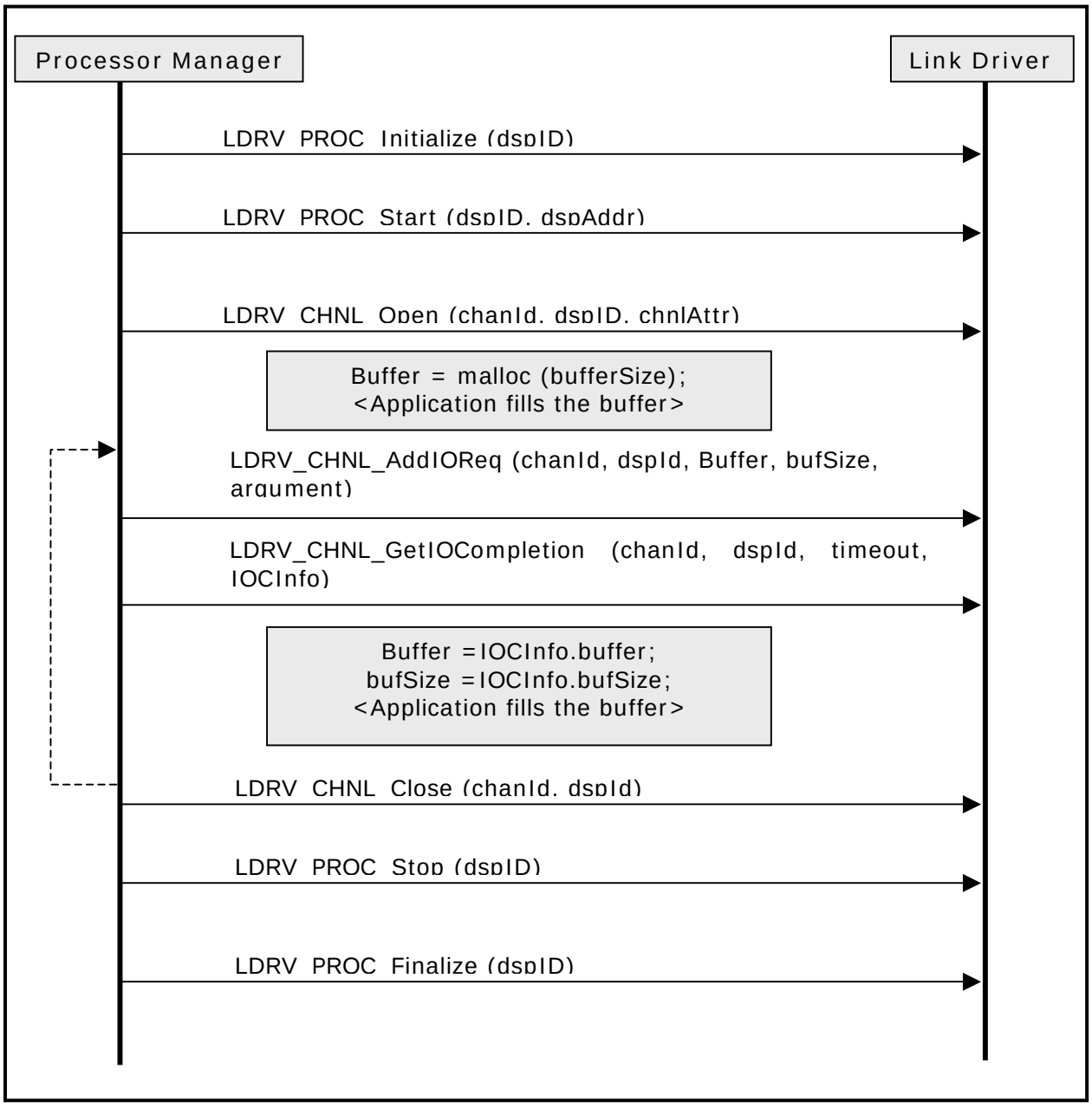

Figure 3. Typical Usage of Link Driver Components by Processor Manager

2 Requirements

The main requirements of Link Driver are that:

1. It must provide a buffering mechanism to upper layers.
2. It must provide abstraction from the physical link to PMGR.
3. It must be scalable.
4. It must provide easy mechanism to add/ replace a new link driver.
5. It must provide easy mechanism to add/replace a new DSP.

3 Assumptions

The following are assumed in the design:

1. Though the current implementation does not support multiple processors, the design assumes support for multiple DSPs in near future.
2. The function pointer interface provides a reasonable degree of plug-in capability, necessary for scalability of DSP/BIOS™ LINK.
3. The initial implementation shall be tested with only one link driver. The actual test of scalability and plug-in capability may not be feasible until more physical link drivers are implemented.

4 LDRV

This module provides a central place to initialize resources that the Link Driver uses.

4.1 Dependencies

4.1.1 Subordinates

CFG database

4.1.2 Preconditions

None.

4.2 Description

This subcomponent fetches the configuration data from the CFG database and maintains the information in a global object accessible to all its constituents - LDRV_Obj. The LDRV_Obj also contains the run time information required by LDRV component. This data includes:

1. Number of DSPs configured in the system.
2. Number of physical link tables configured in the system. A link table may be shared between multiple DSPs.
3. Number of MMU tables configured in the system. A MMU table may be shared between multiple DSPs.
4. An array of all link tables used in the system. If a link table is configured, but not used, it is not available at run-time.
5. An array of all MMU tables used in the system. If a MMU table is configured, but not used, it is not available at run-time.
6. Array of DSP objects containing run-time information for all target DSPs.

4.3 Typedefs and Data Structures

4.3.1 LDRVObject

This structure defines the channel object maintained for every channel opened on a per DSP basis.

Definition

```
typedef struct LDRV_Object_tag {
    Uint32          numDsps       ;
    Uint32          numLinkTables ;
    Uint32          numMmuTables  ;
    DspObject *     dspObjects    ;
    LinkAttrs **    linkTables    ;
    DspMmuEntry **  mmuTables     ;
} LDRV_Object ;
```

Fields

numDsps	Number of DSPs connected to the GPP.
numLinkTables	Number of link tables specified in configuration database.
numMmuTables	Number of MMU tables specified in configuration database.
dspObjects	Array of DSP objects.
linkTables	Array of pointers to link tables.
mmuTables	Array of pointers to MMU tables.

Comments

None.

See Also

DspObject
LinkAttrs
DspMmuEntry

4.4 API Definition

4.4.1 LDRV_Initialize

This function fetches the configuration data from the CFG database and makes it available for access at run time. It also allocates and initializes the global runtime objects required within LDRV context.

Syntax

```
DSP_STATUS LDRV_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Generic failure while allocating memory.
DSP_EFAIL	General error returned from GPP OS

Comments

None.

Constraints

None.

See Also

LDRV_Finalize

4.4.2 LDRV_Finalize

This function releases all the resources that were allocated earlier by a call to function LDRV_Initialize ().

Syntax

```
DSP_STATUS LDRV_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Generic failure while freeing memory
DSP_EFAIL	General error returned from GPP OS

Comments

None.

Constraints

None.

See Also

LDRV_Initialize

5 LDRV_PROC

This subcomponent provides services to control the DSP processor. The generic control function may be – reset, start, stop, read, write, send interrupt, clear interrupt, etc.

5.1 Dependencies

5.1.1 Subordinates

DSP subcomponent

5.1.2 Preconditions

The PROC subcomponent in PMGR must validate all data before passing it to LDRV_PROC. LDRV_PROC does not perform a runtime check on the function arguments and assumes runtime validation of arguments by calling the functions.

5.2 Description

It provides APIs to read from and write into the DSP memory space, allowing the PROC subcomponent (in PMGR) to load a DSP executable onto the target DSP. This subcomponent uses the services of a DSP module to perform its tasks.

This subcomponent also implements a state machine to encapsulate the current state of the DSP. Figure 4 shows the state transition diagram for LDRV_PROC.

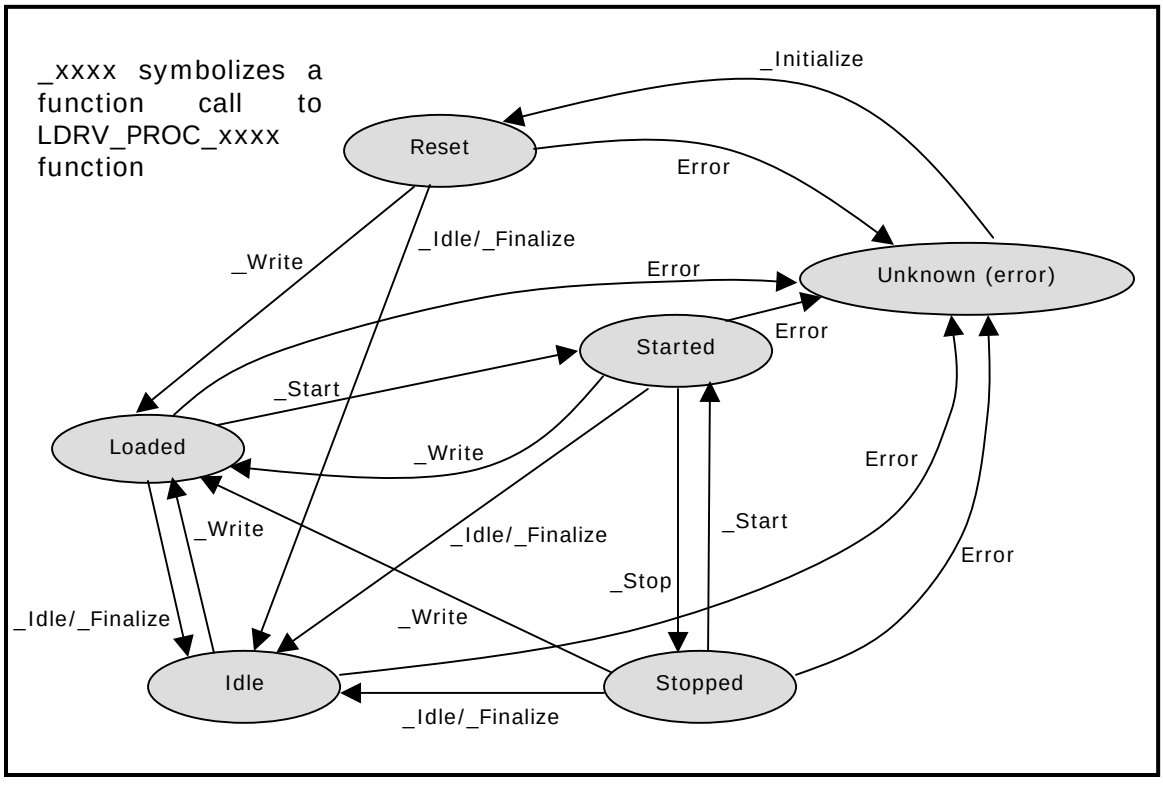

Figure 4. LDRV_PROC State Transition Diagram

5.3 API Definition

5.3.1 LDRV_PROC_Initialize

This function sets up the peripherals required to make the target DSP reachable from the GPP. This function also calls the initialize function exported by the corresponding the DSP subcomponent. The target DSP is in the RESET state after successful completion of this function.

Syntax

```
DSP_STATUS LDRV_PROC_Initialize (ProcessorId dspId)
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP to initialize

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before calling this function.

See Also

LDRV_PROC_Finalize

5.3.2 LDRV_PROC_Finalize

This function releases the communication between the GPP and the target DSP. This design ensures that the DSP is in RESET state after successful completion of this function. This behavior may be customized depending upon the application needs.

Syntax

```
DSP_STATUS LDRV_PROC_Finalize (ProcessorId dspId);
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP to finalize

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS

Comments

None.

Constraints

`LDRV_Initialize ()` must be called before calling this function.

The DSP must not be in the `Error` state.

See Also

`LDRV_PROC_Initialize`

5.3.3 LDRV_PROC_Start

This function starts the DSP run from the specified address. The target DSP is in the `STARTED` state after successful completion of this function.

Communication between the GPP and the target DSP may require handshake over certain physical links before any data transfer can happen. This function initiates the handshake process.

Syntax

```
DSP_STATUS LDRV_PROC_Start (ProcessorId dspId, Uint32 dspAddr)
```

Arguments

`IN`	`ProcessorId`	`dspId`
	Identifier for the DSP to start	
`IN`	`Uint32`	`dspAddr`
	Address to start execution on the DSP	

Return Values

`DSP_SOK`	Operation completed successfully
`DSP_EFAIL`	General error from the GPP OS
`DSP_EWRONGSTATE`	Operation performed in wrong state.

Comments

None.

Constraints

`LDRV_Initialize ()` must be called before calling this function.

The DSP must be either in the `Loaded` or in the `Stopped` state.

See Also

`LDRV_PROC_Stop`

5.3.4 LDRV_PROC_Stop

This function stops the DSP execution. The target DSP is in the `STOPPED` state after successful completion of this function.

Syntax

```
DSP_STATUS LDRV_PROC_Stop (ProcessorId dspId)
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP to stop

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS
DSP_EWRONGSTATE	Operation performed in wrong state.

Comments

None.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must be either in the Started or in the Stopped state.

See Also

LDRV_PROC_Start

5.3.5 LDRV_PROC_Idle

This function puts the DSP in idle mode. On successful execution of this function, the DSP is running the IDLE code.

Syntax

```
DSP_STATUS LDRV_PROC_Idle (ProcessorId dspId)
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP to idle

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS
DSP_EWRONGSTATE	Operation performed in wrong state.

Comments

None.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must not be in the Error state.

See Also

LDRV_PROC_Initialize
LDRV_PROC_Finalize

5.3.6 LDRV_PROC_Read

This function reads specified number of bytes from the DSP memory space in a given buffer.

Syntax

```
DSP_STATUS LDRV_PROC_Read (ProcessorId   dspId,
                           Uint32        dspAddr,
                           Endianism     endianInfo,
                           Uint32 *      numBytes,
                           Uint8 *       buffer) ;
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	Uint32	dspAddr

Address from where to read

IN	Endianism	endianInfo

Specifies endianism attribute of the target memory

IN	Uint32 *	numBytes

Number of bytes to read

OUT	Uint8 *	buffer

Buffer to store the read data

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS
DSP_EWRONGSTATE	Operation performed in wrong state

Comments

None.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must not be in the Error state.

See Also

LDRV_PROC_Initialize
LDRV_PROC_Write

5.3.7 LDRV_PROC_Write

This function writes specified number of bytes to the DSP memory space from a given buffer.

Syntax

```
DSP_STATUS LDRV_PROC_Write (ProcessorId    dspId,
                            Uint32         dspAddr,
                            Endianism      endianInfo,
                            Uint32         numBytes,
                            Uint8 *        buffer) ;
```

Arguments

IN Uint32 dspAddr

Address to which we need to write

IN Endianism endianInfo

Specifies endianism attribute of the target memory

IN Uint32 numBytes

Number of bytes to write

IN Uint 8 * buffer

Buffer containing data to write

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS
DSP_EWRONGSTATE	Operation performed in wrong state

Comments

None.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must not be in the Error state.

See Also

LDRV_PROC_Initialize
LDRV_PROC_Read

5.3.8 LDRV_PROC_GetState

This function gets the current state of the DSP.

Syntax

```
DSP_STATUS LDRV_PROC_GetState (ProcessorId  dspId,
                               ProcState *  procStatus) ;
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

OUT	ProcState *	procStatus

OUT argument to return the current state of the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS

Comments

The state of the DSP is maintained locally by this subcomponent.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must not be in the Error state.

See Also

LDRV_PROC_Initialize
LDRV_PROC_Finalize
LDRV_PROC_Idle
LDRV_PROC_Start
LDRV_PROC_Stop

5.3.9 LDRV_PROC_SetState

This function sets the current state of the DSP.

Syntax

```
DSP_STATUS LDRV_PROC_GetState (ProcessorId  dspId,
                               ProcState    procStatus) ;
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	ProcState	procStatus

The new state of the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS

Comments

The state of the DSP is maintained locally by this subcomponent.

Constraints

LDRV_Initialize () must be called before calling this function.

The DSP must not be in the Error state.

See Also

LDRV_PROC_Initialize
LDRV_PROC_Finalize
LDRV_PROC_Idle
LDRV_PROC_Start
LDRV_PROC_Stop

5.3.10 LDRV_PROC_Control

Provides a hook to perform device dependent control operations.

Syntax

```
DSP_STATUS LDRV_PROC_Control (ProcessorId dspId,
                              Int32       cmd,
                              Pvoid      arg) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	Int32	cmd
	Command identifier.	
IN	Pvoid	Arg
	Optional argument	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified

Comments

None.

Constraints

`LDRV_Initialize ()` must be called before calling this function.

The DSP must not be in the `Error` state.

See Also

`None.`

5.3.11 LDRV_PROC_Debug

This is a debug mode function. It prints the debug information of the specified DSP.

Syntax

```
Void LDRV_PROC_Debug (IN ProcessorId procId)
```

Arguments

`IN`	`ProcessorId`	`procId`

Identifier for the target DSP

Return Values

None.

Comments

None.

Constraints

None.

See Also

`None.`

5.3.12 LDRV_PROC_Instrument

This is a debug mode function. It returns the statistics information (instrumentation data) of the specified DSP.

Syntax

```
Void LDRV_PROC_Instrument (IN ProcessorId procId)
```

Arguments

`IN`	`ProcessorId`	`procId`

Identifier for the target DSP

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

6 LDRV_CHNL

This subcomponent provides buffer management services for all logical channels in DSP/BIOS™ LINK

6.1 Dependencies

6.1.1 Subordinates

LDRV_IO

6.1.2 Preconditions

The CHNL subcomponent in PMGR must validate all data before passing it to LDRV_CHNL. LDRV_CHNL does not perform a runtime check on the function arguments and assumes runtime validation of arguments by calling the functions.

6.2 Description

It creates three different queues to manage the data buffers. A queue of:

1. Free buffers
2. Buffers on which data transfer is requested, and,
3. Buffers on which data transfer has been completed or cancelled.

It also provides APIs for use by CHNL (of PMGR) subcomponent to affect the data transfer between the GPP and the DSP. These APIs work in conjunction with the LDRV_IO subcomponent.

6.3 Typedefs and Data Structures

6.3.1 LDRVChnlObject

This structure defines the channel object maintained for every channel opened on a per DSP basis.

Definition

```
typedef struct LDRVChnlObject_tag {
    Uint32          signature     ;
    ChannelState    chnlState     ;
    List *          freeList      ;
    List *          requestList   ;
    List *          completedList ;
    ChannelAttrs    attrs         ;
    SyncEvObject *  syncEvent     ;
    SyncEvObject *  chnlIdleSync  ;
} LDRVChnlObject ;
```

Fields

`signature`	Signature of object
`ChnlState`	State of the channel
`FreeList`	List for free channel IO request packets (CHIRP)
`requestList`	List for requested CHIRPs
`completedList`	List for completed CHIRPs
`Attrs`	Attributes of this CHIRPs
`SyncEvent`	Event to signal when some IO is completed or cancelled for this channel
`chnlIdleSync`	Event to signal when channel has no more pending IO requests.

Comments

None.

See Also

`ChannelAttrs`
`LDRVChnlIRP`
`LDRVChnlIOInfo`

6.3.2 LDRVChnlIRP

This structure encapsulates information associated with an IO buffer.

Definition

```
typedef struct LDRVChnlIRP_tag {
    ListElement        link      ;
    Uint8 *            buffer    ;
    Uint32             arg       ;
    Uint32             size      ;
    Uint32             iocStatus ;
} LDRVChnlIRP ;
```

Fields

`link`	List element header needed for this structure
`buffer`	Buffer to fill/empty
`arg`	Issue reclaim argument
`size`	Buffer length
`iocStatus`	Status of IO completion

Comments

None.

See Also

`LDRVChnlObject`

6.3.3 LDRVChnlIOInfo

This structure encapsulates information about a data transfer buffer.

Definition

```
typedef struct LDRVChnlIOInfo_tag {
    Pvoid    buffer           ;
    Uint32   size             ;
    Uint32   arg              ;
    IOState  completionStatus ;
} LDRVChnlIOInfo ;
```

Fields

`buffer`	Pointer to the data buffer
`size`	Size of the data buffer
`arg`	Argument to send or received together with the data buffer
`completionStatus`	Completion status of this IO request

Comments

None.

See Also

`LDRVChnlObject`

6.4 API Definition

6.4.1 LDRV_CHNL_Initialize

This function allocates and initializes the resources required by this module. It also initializes the data transfer mechanism of the physical link driver by calling LDRV_IO_Initialize ().

Syntax

```
DSP_STATUS  LDRV_CHNL_Initialize (ProcessorId procId)
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred
DSP_EFAIL	General error from the GPP OS

Comments

None.

Constraints

procId must be valid.

See Also

LDRV_CHNL_Finalize
LDRV_CHNL_Open

6.4.2 LDRV_CHNL_Finalize

This function closes all open channels (if any). It then closes the data transfer mechanism of the physical link by calling LDRV_IO_Finalize ().

Syntax

```
DSP_STATUS LDRV_CHNL_Finalize (ProcessorId procId)
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred

DSP_EFAIL	General failure

Comments

None.

Constraints

procId must be valid.

See Also

LDRV_CHNL_Initialize

6.4.3 LDRV_CHNL_Open

This function prepares the specified channel for data transfer. It creates the three required queues for buffer management on the channel. It also creates the SYNC objects required for waiting on a pending data transfer request.

Syntax

```
DSP_STATUS LDRV_CHNL_Open (ProcessorId      procId,
                           ChannelId        chnlId,
                           ChannelAttrs *   attrs) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel to open	
IN	ChannelAttrs *	attrs
	Channel attributes	

Return Value

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred
DSP_EFAIL	General error from the GPP OS
CHNL_E_BUSY	Channel already in use.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

attrs must be a valid pointer.

See Also

LDRV_CHNL_Initialize

6.4.4 LDRV_CHNL_Close

This function closes the specified channel. It frees all the resources allocated earlier in the call to LDRV_CHNL_Open ().

Once a channel is closed, no further IO can be performed on it, unless it is opened again.

Syntax

```
DSP_STATUS LDRV_CHNL_Close (ProcessorId  procId,
                            ChannelId    chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Channel to close	

Return Value

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred
DSP_EFAIL	General error from the GPP OS

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

LDRV_CHNL_Initialize
LDRV_CHNL_Open

6.4.5 LDRV_CHNL_AddIORequest

This function adds an IO request on a channel. An IO request may be a request for transferring a buffer from the GPP to DSP or from the DSP to GPP. The attributes specified while creating the channel determines the direction of the data transfer.

Syntax

```
DSP_STATUS LDRV_CHNL_AddIORequest (ProcessorId      procId,
                                   ChannelId        chnlId,
                                   LDRVChnlIOInfo * ioInfo)
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Channel to send/receive data	
IN	LDRVChnlIOInfo *	ioInfo
	The IOInfo structure containing information regarding the IO request	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred
DSP_EFAIL	General error from the GPP OS
CHNL_E_EOS	Channel is in EOS (End of Stream) state.
CHNL_E_NOIRPS	No more IO could be accepted because maximum limit of pending IO request has reached

Comments

None.

Constraints

procId must be valid.

ChnlId must be valid.

IoInfo must be a valid pointer.

See Also

LDRVChnlIOInfo
LDRV_CHNL_GetIOCompletion

6.4.6 LDRV_CHNL_GetIOCompletion

This function gets a buffer on which IO is complete. It waits for a specified amount of time, if required and specified, for an IO completion event on a channel. On successful completion, the function returns a buffer to the caller. The contents of the buffer depend on the direction of channel.

For an input channel, the buffer contains valid data as received from the DSP and for an output channel, the buffer is an empty buffer that was earlier used to send data to the DSP.

Syntax

```
DSP_STATUS LDRV_CHNL_GetIOCompletion (ProcessorId      procId,
                                      ChannelId        chnlId,
                                      Uint32           timeout,
                                      LDRVChnlIOInfo * ioInfo) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Channel on which to send/receive data	
IN	Uint32	timeout
	Timeout value	
OUT	LDRVChnlIOInfo *	ioInfo
	Structure containing the OUT buffer pointer and also any values associated with the buffer	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error occurred
DSP_EFAIL	General error from the GPP OS
DSP_ETIMEOUT	Timout occurred while performing the operation.
CHNL_E_NOIOC	Timeout parameter was "NO_WAIT", yet no I/O completions were queued.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

ioInfo must be a valid pointer.

See Also

LDRVChnlIOInfo
LDRV_CHNL_AddIORequest
LDRV_CHNL_AddIOCompletion

6.4.7 LDRV_CHNL_AddIOCompletion

This function performs the required operations for completing an IO operation on a CHIRP.

Syntax

```
DSP_STATUS LDRV_CHNL_AddIOCompletion (ProcessorId   procId,
                                      ChannelId     chnlId,
                                      LDRVChnlIRP * chirp) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	
IN	LDRVChnlIRP *	chirp
	The IO request packet on which IO is complete	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS

Comments

This function adds the specified CHIRP to the queue containing CHIRPs on which IO is complete.

Constraints

procId must be valid.

chnlId must be valid.

chirp must be a valid pointer.

See Also

None.

6.4.8 LDRV_CHNL_Idle

In case of input mode channel this function discards all pending input requests from the channel. In case of output mode channel, action of this function depends upon the flush parameter and is as follows:

- If flush is TRUE this function blocks till all output buffers are transferred to the DSP.
- If flush is FALSE this function discards all the output requests pending on this channel without blocking.

Syntax

```
DSP_STATUS  LDRV_CHNL_Idle (ProcessorId  procId,
                            ChannelId    chnlId,
                            Bool         flush) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId

Channel on which to cancel IO

IN Bool flush

This parameter tells whether to block or not on output mode channels.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General error from the GPP OS
DSP_EMEMORY	Memory error occurred

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

LDRV_CHNL_AddIORequest
LDRV_CHNL_GetIOCompletion

6.4.9 LDRV_CHNL_Control

Provides a hook to perform device dependent control operations on channels.

Syntax

```
DSP_STATUS LDRV_CHNL_Control (ProcessorId  procId,
                              ChannelId    chnlId,
                              Int32        cmd,
                              Pvoid        arg) ;
```

Arguments

IN ProcessorId procId

Processor Identifier

IN ChannelId chnlId

Channel Identifier

IN Int32 cmd

Command id.

IN Pvoid arg

Optional argument

Return Values

DSP_SOK	Operation completed successfully
DSP_ENOTIMPL	Functionality not implemented

Comments

This function provides a hook to perform the device dependent control operations on channels. Not implemented in current implementation

Constraints

None.

See Also

LDRV_CHNL_Initialize

6.4.10 LDRV_CHNL_GetChannelMode

This function gets the mode of the channel (Input or Output).

Syntax

```
ChannelMode LDRV_CHNL_GetChannelMode (ProcessorId   procId,
                                      ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Values

ChannelMode_Input	The channel is an input channel.
ChannelMode_Output	The channel is an output channel.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.11 LDRV_CHNL_GetChannelState

This function gets the current state of the channel.

Syntax

```
ChannelState LDRV_CHNL_GetChannelState (ProcessorId   procId,
                                        ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Value

The current state of the channel

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.12 LDRV_CHNL_SetChannelState

This function sets the channel's state.

Syntax

```
Void LDRV_CHNL_SetChannelState (ProcessorId   procId,
                                ChannelId     chnlId,
                                ChannelState  state) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	
IN	ChannelState	state
	New state of the channel.	

Return Value

None.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.13 LDRV_CHNL_GetChannelEndianism

This function gets the data endianism associated with a channel.

Syntax

```
Endianism LDRV_CHNL_GetChannelEndianism (ProcessorId   procId,
                                         ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

IN	ChannelId	chnlId

Identifier for the channel

Return Value

The endianism associated with the specified channel.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.14 LDRV_CHNL_ChannelHasMoreChirps

This function gets the current state of the requested queue in the channel.

Syntax

```
Bool LDRV_CHNL_ChannelHasMoreChirps (ProcessorId   procId,
                                     ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

IN ChannelId chnlId

Identifier for the channel

Return Values

TRUE The channel has more request CHIRPs.

FALSE The requested queue in the channel is empty.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.15 LDRV_CHNL_GetRequestdChirp

This function gets a CHIRP from the request queue of a channel.

Syntax

```
LDRVChnlIRP * LDRV_CHNL_GetRequestChirp (ProcessorId   procId,
                                         ChannelId     chnlId) ;
```

Arguments

IN ProcessorId procId

Identifier for the DSP

IN ChannelId chnlId

Identifier for the channel

Return Values

NULL If the request list is empty

Non-NULL Pointer to a CHIRP from the request queue

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.16 LDRV_CHNL_Debug

This is a debug mode function. It prints the debug information of the specified channel.

Syntax

```
Void LDRV_CHNL_Debug (ProcessorId procId, ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Value

None.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

6.4.17 LDRV_CHNL_Instrument

This is a debug mode function. It returns the statistics information (instrumentation data) of the specified channel.

Syntax

```
Void LDRV_CHNL_Instrument (ProcessorId procId, ChannelId chnlId)
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Value

None.

Comments

None.

Constraints

procId must be valid.

chnlId must be valid.

See Also

None.

7 LDRV_IO

This subcomponent acts as glue between the LDRV_CHNL and the physical link driver(s).

7.1 Dependencies

7.1.1 Subordinates

The DSP subcomponent is used by this subcomponent for interacting with the DSP.

7.2 Description

This subcomponent provides the logical IO services to LDRV_CHNL. It passes all the requests to the actual physical link driver, using its function pointer interface exported by the link driver(s).

Usage of function pointer interface ensures that multiple link drivers can be easily plugged into the system.

To determine which physical link to be used for IO, it maintains a map between channel ID and the physical link ID.

7.3 API Definition

7.3.1 LDRV_IO_Initialize

This function initializes the resources required by this module. It also calls the function initialize from the function pointer interface exported by all link drivers attached to the specified DSP.

Syntax

```
DSP_STATUS LDRV_IO_Initialize (ProcessorId  procId)
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize
LDRV_IO_Finalize

7.3.2 LDRV_IO_Finalize

This function releases the resources required by this module. It also calls the function finalize from the function pointer interface exported by all link drivers attached to the specified DSP.

Syntax

```
DSP_STATUS LDRV_IO_Finalize (ProcessorId procId)
```

Arguments

IN	ProcessorId	procId

DSP ID of DSP for which the finalization must be performed

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory

DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize
LDRV_IO_Initialize

7.3.3 LDRV_IO_ OpenChannel

This function opens the physical channel corresponding to the specified logical channel by calling the function openChannel from corresponding link driver's function pointer interface.

Syntax

```
DSP_STATUS LDRV_IO_OpenChannel (ProcessorId procId, ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize
LDRV_CloseChannel

7.3.4 LDRV_IO_CloseChannel

This function closes the physical channel corresponding to the specified logical channel by calling the function closeChannel from corresponding link driver's function pointer interface.

Syntax

```
DSP_STATUS LDRV_IO_CloseChannel (ProcessorId procId,
                                 ChannelId   chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize
LDRV_OpenChannel

7.3.5 LDRV_IO_Request

1. This function sends an IO request on specified channel by calling the function ioRequest from corresponding link driver's function pointer interface.

Syntax

```
DSP_STATUS LDRV_IO_Request (ProcessorId dspId, ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory

DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize

7.3.6 LDRV_IO_ScheduleDpc

This function schedules the DPC to perform IO on specified channel by calling the function scheduleDpc from corresponding link driver's function pointer interface.

Syntax

```
DSP_STATUS LDRV_IO_Request (ProcessorId dspId, ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for the DSP	
IN	ChannelId	chnlId
	Identifier for the channel	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize

7.3.7 LDRV_IO_Handshake

This function performs the necessary handshake (if required) for all the links between the GPP and the target DSP by calling functions handshakeStart and handshakeStart from corresponding link driver's function pointer interface.

Syntax

```
DSP_STATUS LDRV_IO_Handshake (ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Initialize

7.3.8 LDRV_IO_Debug

This is a debug mode function. It prints the debug information for the link driver(s) towards specified target DSP.

Syntax

```
Void LDRV_IO_Debug (IN ProcessorId dspId)
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

None.

Comments

None.

Constraints

procId must be valid.

See Also

None.

8 DSP

This subcomponent provides interfaces to directly control and communicate with the target DSP.

8.1 Description

This subcomponent directly interacts with the hardware and provides access to the target DSP. It essentially abstracts the DSP from other subcomponents in acts as an abstraction for the DSP.

8.2 Typedefs and Structures

8.2.1 DspMmuEntry

This structure defines an MMU entry for the c55x DSP.

Definition

```
typedef struct DspMmuEntry_tag {
    Uint32  entry           ;
    Uint32  virtualAddress  ;
    Uint32  physicalAddress ;
    Uint32  size            ;
    Uint32  access          ;
    Uint32  preserve        ;
} DspMmuEntry ;
```

Fields

entry	Entry number for the MMU record
virtualAddress	Virtual address
physicalAddress	Physical address
Size	Indicates the size of MMU TLB entry
access	Access permission
preserve	Indicates if the entry is preserved

Comments

None.

See Also

DspObject

8.2.2 DspObject

This structure defines the context under which the DSP subcomponent works.

Definition

```
typedef struct DspObject_tag {
    Char8          dspName    [DSP_MAX_STRLEN] ;
    Char8          execName   [DSP_MAX_STRLEN] ;
    Char8          parserName [DSP_MAX_STRLEN] ;
    LinkAttrs *    linkTable     ;
    Uint32         numLinks      ;
    Uint32         autoStart     ;
    Uint32         resetVector   ;
    Uint32         wordSize      ;
    Uint32         endian        ;
    Bool           mmuFlag       ;
    DspMmuEntry *  mmuTable      ;
    Uint32         numMmuEntries ;
    DspInterface * interface     ;
} DspObject ;
```

Fields

dspName	Name of the DSP
execName	Name of default DSP executable
parserName	Name of the parser used
linkTable	Array of link attributes
numLinks	Number of links towards the DSP
autoStart	Auto start flag for the DSP.
WordSize	Word size of the DSP
endian	Endianism of the DSP
mmuFlag	Indicates if the MMU is enabled on the DSP
mmuTable	Table of MMU entries
numMmuEntries	Number of MMU entries
interface	The function pointer interface to access the services of the DSP subcomponent for this DSP

Comments

None.

See Also

DspMmuEntry

8.3 API Definition

8.3.1 DSP_Setup

This function performs necessary operations to make the DSP reachable from the GPP.

Syntax

```
DSP_STATUS DSP_Setup (ProcessorId dspId, DspObject *  dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	General failure returned from GPP OS

Comments

This function initializes the necessary hardware abstraction layer. It sets up the ARM port interface and the DSP boot configuration.

Constraints

None.

See Also

DspObject
DSP_Initialize

8.3.2 DSP_Initialize

This function resets the DSP and initializes peripherals required by the DSP (for example, MMU entries).

Syntax

```
DSP_STATUS DSP_Initialize (ProcessorId dspId, DspObject *  dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj

Pointer to object containing context information for DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

This function:

1. Resets the DSP
2. Sets up the MMU table
3. Sets up the clock divisors.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Setup
DSP_Finalize

8.3.3 DSP_Finalize

This function idles the DSP.

Syntax

```
DSP_STATUS DSP_Finalize (ProcessorId dspId, DspObject *  dspObj)
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	DspObject *	dspObj

Pointer to object containing context information for DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Setup
DSP_Initialize

8.3.4 DSP_Start

This function starts the DSP run from the specified address.

Syntax

```
DSP_STATUS DSP_Start (ProcessorId  dspId,
                      DspObject *  dspObj,
                      Uint32       dspAddr)
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	DspObject *	dspObj

Pointer to object containing context information for DSP

IN	Uint32	dspAddr

Location to start the execution on the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Initialize
DSP_Stop

8.3.5 DSP_Stop

This function stops execution on the DSP.

Syntax

```
DSP_STATUS DSP_Stop (ProcessorId dspId, DspObject *  dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

This function configures the ARM port interface to put the DSP into a self loop and then puts the DSP into a self loop.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Initialize
DSP_Start

8.3.6 DSP_Idle

This function idles the DSP.

Syntax

```
DSP_STATUS DSP_Idle (ProcessorId dspId, DspObject *  dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified

DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

This function writes the idle code onto the DSP and starts its execution.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Setup
DSP_Stop

8.3.7 DSP_EnableInterrupt

This function enables the specified interrupt for communication with DSP.

Syntax

```
DSP_STATUS DSP_EnableInterrupt (ProcessorId        dspId,
                                DspObject *        dspObj,
                                InterruptObject *  intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP	
IN	InterruptObject *	intInfo
	Pointer to an object containing interrupt information	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject

InterruptObject
DSP_DisableInterrupt
DSP_Interrupt
DSP_ClearInterrupt

8.3.8 DSP_DisableInterrupt

This function disables the specified interrupt for communication with DSP.

Syntax

```
DSP_STATUS DSP_EnableInterrupt (ProcessorId        dspId,
                                DspObject *        dspObj,
                                InterruptObject *  intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP	
IN	InterruptObject *	intInfo
	Pointer to an object containing interrupt information	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
InterruptObject
DSP_EnableInterrupt
DSP_Interrupt
DSP_ClearInterrupt

8.3.9 DSP_Interrupt

This function sends the specified interrupt to the DSP.

Syntax

```
DSP_STATUS DSP_Interrupt (ProcessorId        dspId,
                          DspObject *        dspObj,
```

```
                                    InterruptObject *  intObj) ;
```

Arguments

IN	ProcessorId	dspId	
	Identifier for the DSP		
IN	DspObject *	dspObj	
	Pointer to object containing context information for DSP		
IN	InterruptObject *	intInfo	
	Pointer to an interrupt object containing the information regarding the interrupt to be sent to the DSP		

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
InterruptObject
DSP_EnableInterrupt
DSP_DisableInterrupt
DSP_ClearInterrupt

8.3.10 DSP_ClearInterrupt

This function clears an interrupt received from the DSP side on to the GPP side.

Syntax

```
DSP_STATUS DSP_ClearInterrupt (ProcessorId        dspId,
                               DspObject *        dspObj,
                               InterruptObject *  intObj) ;
                               Uint16 *           retVal) ;
```

Arguments

IN	ProcessorId	dspId	
	Identifier for the DSP		
IN	DspObject *	dspObj	

Pointer to object containing context information for DSP

IN	InterruptObject *	intInfo

Pointer to an interrupt object containing the information regarding the interrupt to be sent to the DSP

OUT	Uint16 *	retVal

Interrupt value present before clearing the interrupt

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

None.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
InterruptObject
DSP_EnableInterrupt
DSP_DisableInterrupt
DSP_Interrupt

8.3.11 DSP_Read

This function reads data from the DSP memory space.

Syntax

```
DSP_STATUS DSP_Read (ProcessorId  dspId,
                     DspObject *  dspObj,
                     Uint32       dspAddr,
                     Endianism    endianInfo,
                     Uint32 *     numBytes,
                     Uint8 *      buffer) ;
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	DspObject *	dspObj

Pointer to object containing context information for DSP

IN	Uint32	dspAddr

Address to read

IN	Endianism	endianInfo

Specifies the memory endianism of the target memory

OUT	Uint32 *	numBytes

IN/OUT argument to specify the number of bytes to read and upon return contain the actual number of bytes read

OUT	Uint8 *	buffer

Buffer to hold the read data

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

This function performs the endianism conversion required

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Write

8.3.12 DSP_Write

This function writes data into the DSP memory space.

Syntax

```
DSP_STATUS DSP_Write (ProcessorId dspId,
                      DspObject * dspObj,
                      Uint32      dspAddr,
                      Endianism   endianInfo,
                      Uint32      numBytes,
                      Uint8 *     buffer) ;
```

Arguments

IN	ProcessorId	dspId

Identifier for the DSP

IN	DspObject *	dspObj

Pointer to object containing context information for DSP

IN	Uint32	dspAddr
	Address to write the data	
IN	Endianism	endianInfo
	Specifies the memory endianism of the target memory	
IN	Uint32	numBytes
	Number of bytes to write	
IN	Uint8 *	buffer
	Buffer containing the data to write	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified
DSP_EFAIL	DSP_Setup () was not called before calling this function

Comments

This function performs the necessary endianism conversion on the data before writing it to the target memory.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Write

8.3.13 DSP_Control

Hook for performing device dependent control operation.

Syntax

```
DSP_STATUS DSP_Control (IN  ProcessorId dspId,
                        IN  DspObject * dspObj,
                        IN  Int32       cmd,
                        OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

IN	Int32	cmd
	Command id.	
IN	Pvoid	Arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified

Comments

This function performs the necessary endianism conversion on the data before writing it to the target memory.

Constraints

DSP_Setup () must be called before calling this function.

See Also

DspObject
DSP_Write

9 Physical Link Driver

This subcomponent implements the physical link driver for data communication between the GPP and target DSP.

This section illustrates the API for a generic link driver – 'GLINK'. The actual link driver shall prefix more specific name in this API e.g. Shared Memory link driver may use prefix SHM.

9.1 Dependencies

HW resources specific to the physical link.

9.1.1 Subordinates

This subcomponent uses DSP subcomponent for interacting with the DSP.

9.2 Description

This subcomponent is HW dependent. It uses either the Hardware Abstraction functions or the function pointer interface exported by the DSP subcomponent.

9.3 Typedefs and Data Structures

9.3.1 GLINK_Control

This structure defines the link specific control information used by data transfer protocol.

Definition

```
typedef struct GLINK_Control_tag {
...
...
} GLINK_Control ;
```

Fields

The definition of fields is specific to a link driver.

Comments

None.

9.3.2 GLINK_ DriverInfo

This structure defines driver information object for this link driver.

Definition

```
typedef struct GLINK_DriverInfo_tag {
...
...
} GLINK_ DriverInfo_Control ;
```

Fields

The definition of fields is specific to a link driver.

Comments

None.

9.4 API Definition

9.4.1 GLINK_Initialize

This function initializes the resources required by this link driver. It initializes necessary hardware modules and installs appropriate Interrupt Service Routine(s) and Delayed Procedure Call(s) for data transfer across the physical link.

Syntax

```
DSP_STATUS LDRV_IO_Initialize (ProcessorId  procId, LinkId lnkId)
```

Arguments

IN	LinkId	lnkId
	Identifier for the physical link	
IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

None.

See Also

LDRV_IO_Initialize
GLINK_Finalize

9.4.2 GLINK_Finalize

This function releases the resources allocated earlier by this link driver in call to GLINK_Initialize (). It resets necessary hardware modules and un-installs all Interrupt Service Routine(s) and Delayed Procedure Call(s).

Syntax

```
DSP_STATUS GLINK_Finalize (ProcessorId  procId, LinkId lnkId)
```

Arguments

IN	LinkId	lnkId
	Identifier for the physical link	
IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_IO_Finalize
GLINK_Initialize

9.4.3 GLINK_OpenChannel

This function opens the physical channel corresponding to the specified logical channel.

Syntax

```
DSP_STATUS GLINK_OpenChannel (ProcessorId procId, ChannelId chnlId) ;
```

Arguments

IN	ChannelId	chnlId
	Identifier for the channel	
IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

GLINK_Initialize () must be called before this function.

See Also

LDRV_IO_OpenChannel

GLINK_Initialize
GLINK_CloseChannel

9.4.4 GLINK_CloseChannel

This function closes the physical channel corresponding to the specified logical channel.

Syntax

```
DSP_STATUS GLINK_CloseChannel (ProcessorId procId,
                               ChannelId   chnlId) ;
```

Arguments

IN	ChannelId	chnlId
	Identifier for the channel	
IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_IO_CloseChannel
GLINK_Initialize
GLINK_OpenChannel

9.4.5 GLINK_Request

This function sends an IO request on specified channel. The actual mechanism for sending the request depends upon the specific protocol used by link driver.

Syntax

```
DSP_STATUS GLINK_Request (ProcessorId dspId, ChannelId chnlId) ;
```

Arguments

IN	ChannelId	chnlId
	Identifier for the channel	
IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_IO_Request

9.4.6 GLINK_ScheduleDpc

This function schedules the DPC to perform IO on specified channel.

Syntax

```
DSP_STATUS GLINK_ScheduleDpc (ProcessorId dspId, ChannelId chnlId) ;
```

Arguments

IN	ChannelId	chnlId
	Identifier for the channel	
IN	ProcessorId	procId
	Identifier for the DSP	

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_ScheduleDPC
GLINK_DPC

9.4.7 GLINK_HandshakeStart

This function initiates the handshake on the physical link with the target DSP. The actual mechanism for handshake depends upon the specific protocol used by link driver.

Syntax

```
DSP_STATUS GLINK_HandshakeStart (ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

LDRV_Handshake
GLINK_HandshakeComplete

9.4.8 GLINK_HandshakeComplete

This function completes the handshake on the physical link with the target DSP. The actual mechanism for handshake depends upon the specific protocol used by link driver.

This function block until handshake completes.

Syntax

```
DSP_STATUS GLINK_HandshakeComplete (ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

`LDRV_Initialize ()` must be called before this function.

See Also

`LDRV_Handshake`
`GLINK_HandshakeStart`

9.4.9 GLINK_ISR

This function is interrupt service routine for interrupt used for data transfer on the physical link between GPP and target DSP.

This function is not exported to LDRV_IO.

Syntax

```
DSP_STATUS GLINK_ISR (Pvoid refData) ;
```

Arguments

`IN` `Pvoid refData` `Pvoid refData`

A void pointer to the link driver object

Return Values

`DSP_SOK`	Operation completed successfully
`DSP_EMEMORY`	Out of memory
`DSP_EFAIL`	General failure returned from GPP OS

Comments

None.

Constraints

`LDRV_Initialize ()` must be called before this function.

See Also

`GLINK_DPC`

9.4.10 GLINK_DPC

This function is delayed procedure for performing actual data transfer on the physical link between GPP and target DSP. This function runs at a priority level higher than user threads/ clients (but lower than interrupt context).

This function is not exported to LDRV_IO.

Syntax

```
DSP_STATUS GLINK_DPC (Pvoid refData) ;
```

Arguments

IN	Pvoid refData	Pvoid refData

A void pointer to the link driver object

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Out of memory
DSP_EFAIL	General failure returned from GPP OS

Comments

None.

Constraints

LDRV_Initialize () must be called before this function.

See Also

GLINK_DPC

9.4.11 GLINK_Debug

This is a debug mode function. It prints the debug information regarding the physical link toward a specified DSP.

Syntax

```
Void GLINK_Debug (IN ProcessorId dspId)
```

Arguments

IN	ProcessorId	procId

Identifier for the DSP

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.