DESIGN DOCUMENT

DSP/BIOS™ LINK

DYNAMIC CONFIGURATION

LNK 137 DES

Version 0.50

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 & Scope

This document describes the design of dynamic configuration for DSP/BIOS™ LINK.

The document is targeted at the development team of DSP/BIOS™ LINK.

1.2 Terms & Abbreviations

DSPLINK	DSP/BIOS™ LINK
⚐	This bullet indicates important information. Please read such text carefully.
❑	This bullet indicates additional information.

1.3 References

1.	LNK 084 PRD	DSP/BIOS™ LINK Product Requirement Document

1.4 Overview

DSP/BIOS™ LINK is runtime software, analysis tools, and an associated porting kit that simplifies the development of embedded applications in which a general-purpose microprocessor (GPP) controls and communicates with a TI DSP. DSP/BIOS™ LINK provides control and communication paths between GPP OS threads and DSP/BIOS™ tasks, along with analysis instrumentation and tools.

This module provides the design for Dynamic Configuration of *DSPLINK*.

This document gives an overview and detailed design of Dynamic Configuration on the GPP and DSP-sides of *DSPLINK*.

2 Requirements

Please refer to section 16.7 of LNK 084 PRD - DSP/BIOS™ LINK Product Requirement Document.

Several users have expressed the need to change the memory map used by DSP/BIOS™ Link dynamically. To enable such use cases, this release shall provide the capability to change memory map used by Link in a more dynamic manner.

R119 This release shall allow applications to configure the memory map used by the product without the need to fully recompile the sources.

R120 Applications shall also be allowed to partially change the memory map during runtime. This shall be allowed only after the application releases all resources.

In addition, the Dynamic Configuration component must meet the following generic requirements:

1. All major configurable items within the existing *DSPLINK* static configuration must be made dynamic without a need to rebuild the GPP-side kernel component.
2. The DSP-side *DSPLINK* libraries must not be required to be rebuilt on changing the values of any of the configuration items.

3 Assumptions

The Dynamic Configuration design makes the following assumptions:

1. The hardware supports multiple processors having shared access to a memory region.

4 Constraints

None.

5 High Level Design

The static configuration of *DSPLINK* is achieved through a textual configuration file (CFG_<PLATFORM>.TXT), which is processed during the build step of *DSPLINK*, to generate configuration header and source files for both the GPP and DSP-sides of *DSPLINK*. These generated files are compiled along-with the *DSPLINK* GPP-side kernel module, and DSP-side dsplink library.

The Dynamic Configuration of *DSPLINK* shall be achieved through configuration items made available to *DSPLINK* from the user-side on the GPP-side only. The GPP-side kernel module and DSP-side library shall not need to be rebuilt.

5.1 Configuration file

A pre-defined "C" source file shall be provided with configuration values defined within a fixed structure format. This shall be compiled with the *DSPLINK* user library by default.

5.2 Change in configuration

`PROC_Setup ()` shall be modified to optionally take a pointer to a configuration structure in the same format as the provided configuration source file. If a valid pointer is provided, the configuration values provided by the application are used. If none is provided, the default configuration is used. This ensures backward compatibility of existing applications.

No configuration source or header files shall be generated, resulting in the GPP-side kernel module and DSP-side *DSPLINK* library not requiring to be rebuilt.

The configuration taken during `PROC_Setup ()` shall be stored on the kernel-side as a pointer to the configuration structure.

5.3 Shared memory configuration

The DRV component shall manage a module called the Shared Memory Manager (SMM), which shall manage memory regions with shared access across processors. The SMM component is a simple offset-based memory manager. It configures itself to manage all memory segments defined in the configuration, which are marked as shared regions. Based on the memory segment ID provided, it returns a pointer to the next chunk of available memory within the memory segment, and increments the internal used offset by specified size. This component expects that the memory acquired is released in the reverse order of acquires. The functionality supported by the SMM component includes:

- Initialize the SMM component
- Finalize the SMM component
- Allocate shared memory
- Free shared memory

For the DM642_PCI platform, the shared memory manager manages the segment(s) of memory that have been mapped for direct access through PCI. A configuration flag in each memory segment indicates whether the memory region allows shared access. Only such memory segments are managed by the SMM component. An example layout of a single memory section shared with a DSP, with memory for the different components, is shown in the following figure:

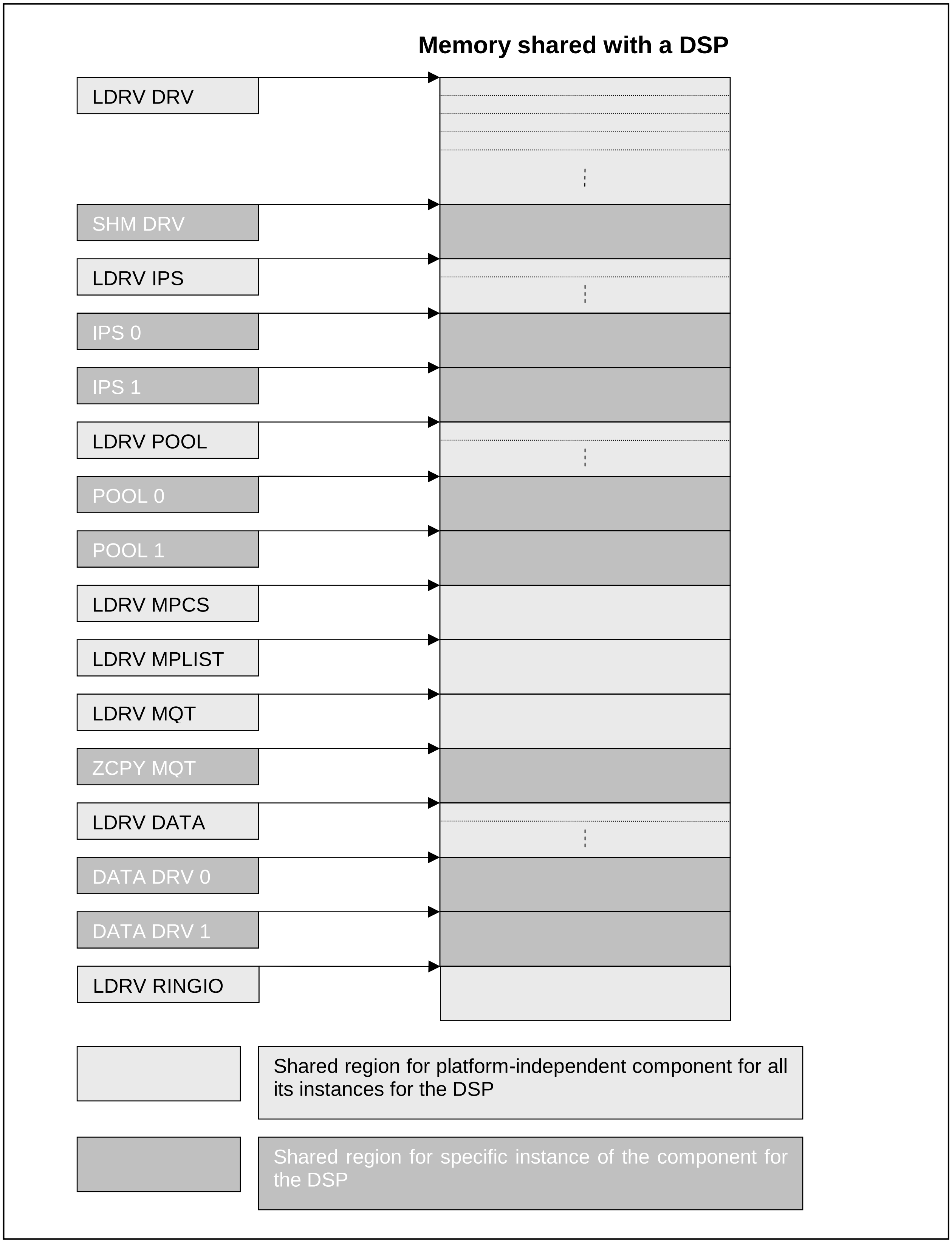

Figure 1. Shared memory layout for one DSP

5.4 Configuration of components

The LDRV component manages the kernel-side configuration structure. Within the initialization function called during `PROC_setup ()`, the kernel-side configuration structure is initialized. Module initialization functions for all LDRV sub-components are also called at this time. The module initialization functions for each sub-component setup their configuration and validate the provided configuration values.

The DRV component provides the interface to all other sub-components within *DSPLINK*.

Initialization functions of all the sub-components are called by the DRV component during `PROC_attach ()`.

Each sub-component initialization function allocates and initializes the chunk of shared memory it requires from the SMM component for the memory segment ID specified in the configuration. It writes the configuration information required to be shared with the DSP within this region. It also returns the address of the shared region to the DRV component (caller), so that the DRV component can set this information within its shared region. The DRV shared region containS pointers to the shared regions of all the sub-components managed by it.

The sub-components managed by the DRV component are:

- **IPS:** The LDRV_IPS component manages one or more instances of IPS present between the GPP and DSP. Its shared memory control region points to the individual shared memory regions required by each IPS instance. The information shared between individual IPS instances consists of the following:
    - Configuration information including information about the GPP and DSP interrupt IDs.
    - IPS control structure with event registration masks, event charts etc.
- **POOL:** The LDRV_POOL component manages one or more instances of POOL present between the GPP and DSP. Its shared memory control region points to the individual shared memory regions required by each POOL instance. The information shared between individual POOL instances consists of the following:
    - Configuration information (if any).
    - POOL control structure with MPCS object, number of buffer sizes, pointer to array of buffer pools for different sizes, etc.
- **MPCS:** The LDRV_MPCS component manages a number of instances of MPCS objects. Its shared memory control region contains the following:
    - Configuration information including maximum number of MPCS instances.
    - MPCS control region containing information about all MPCS instances in the system.
- **MPLIST:** The LDRV_MPLIST component manages a number of instances of MPLIST objects. Its shared memory control region contains the following:
    - Configuration information including maximum number of MPLIST instances.
    - MPLIST control region containing information about all MPLIST instances in the system.

- **MQT:** The LDRV_MQT component manages only a single instance of MQT between the GPP and DSP. It does not have any shared memory requirement of its own, and its shared control region contains the following information specific to the MQT instance used between the GPP and DSP:
  - Configuration information including IPS ID and IPS event number used by the MQT.
  - MQT control structure with MPCS objects and shared lists used for transferring messages between the GPP and the DSP.
- **DATA:** The LDRV_DATA component manages one or more instances of data drivers present between the GPP and DSP. Its shared memory control region points to the individual shared memory regions required by each data driver instance. The information shared between individual data driver instances consists of the following:
  - Configuration information including IPS ID and IPS event number used by the data driver, number of logical channels supported by the data driver, maximum buffer size supported on the channels by the data driver etc.
  - Data driver control structure with masks, MPCS objects and shared lists used for transferring messages between the GPP and the DSP etc.
- **RINGIO:** The LDRV_RINGIO component manages a number of instances of RINGIO objects. Its shared memory control region contains the following:
  - Configuration information including maximum number of RINGIO instances.
  - RINGIO control region containing information about all RINGIO instances in the system.

5.5 DSP-side configuration

Within `PROC_Start ()`, the DRV component's API to setup the handshake is called. This function makes a call to `PMGR_PROC_GetSymbolAddress ()` to get the address of the DSP-side global variable within DRV, where the address of the GPP-side DRV component's shared memory region is to be set using `LDRV_PROC_Write ()`. After starting the DSP, the DRV component then waits for the handshake indicating that the DSP-side initialization is complete for all sub-components based on the scalability configuration selected.

On the DSP-side, a new function `DSPLINK_init ()` shall be added, which shall call `DSPLINK_init ()`. This function shall be called by applications within their `main ()` function. For modules integrated within DSP/BIOS™, this function shall also be called from their initialization functions (POOL, MSGQ, and CHNL). The application shall no longer call individual `<MOD>_init ()` functions. The `DSPLINK_init ()` function shall ensure that the code within it is called only once.

`DSPLINK_init ()` shall initialize itself and call sub-component initialization functions for all the other components based on the configuration flags, and if the module has been configured on the GPP-side (as indicated by the GPP-side).

Each sub-component shall initialize itself using the shared memory address passed to it by the DRV component. All the configuration information required by each sub-component would be present at this address in a fixed format shared between the GPP and DSP-side sub-component.

`DSPLINK_init ()` shall then call `DSPLINK_handshake ()`, which would indicate to the GPP that the initialization of the modules specified by the provided mask is complete. Following this, GPP-side driver handshake would complete with success or failure depending on whether the DSP-side configuration matches the GPP-side expectation, and also whether the DSP-side initialization is complete within the maximum poll count used for the handshake. This poll count shall be specified by the user as part of the dynamic configuration. This would allow the user to debug DSP-side without GPP-side timing out, if so required, by changing the timeout value to a special value indicating infinite wait (`-1`).

To support usage of the RINGIO and MPCS components in DSP-DSP configuration only (without presence of GPP-side component), these components shall check whether their initialization is complete from GPP-side. If not, they shall perform their own initialization. A GEL file can be used to initially reset the initialization flag and set configuration information required by these modules at the location expected by them.

6 Sequence Diagrams

The following sequence diagrams show the control flow for the configuration setup, initialization, and finalization of the various components on GPP and DSP-sides. A sequence diagram showing the handshake between the GPP and DSP-sides of *DSPLINK* is also given.

6.1 Configuration setup

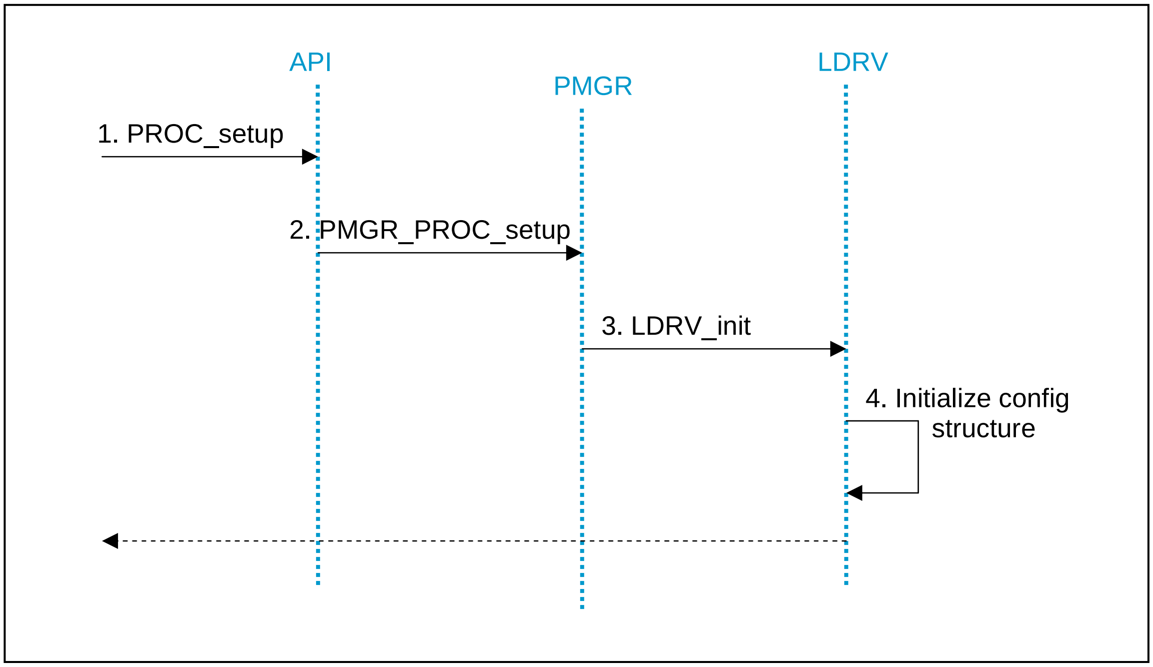

Figure 2. GPP-side configuration setup

1. A pointer to the configuration information is optionally provided by the application during `PROC_setup ()`. If provided, this structure replaces the existing default configuration built within the *DSPLINK* API library.
2. The configuration structure is passed on by PMGR layer to LDRV, where an OS-specific implementation of `LDRV_init ()` initializes the kernel-side pointer to the configuration structure. If an OS such as PrOS is used, which does not have a user-kernel separation, the address of the user-side structure is directly used as the configuration. Otherwise a newly allocated kernel-side configuration structure is initialized with the contents of the user-side configuration structure.

6.2 GPP-side Initialization

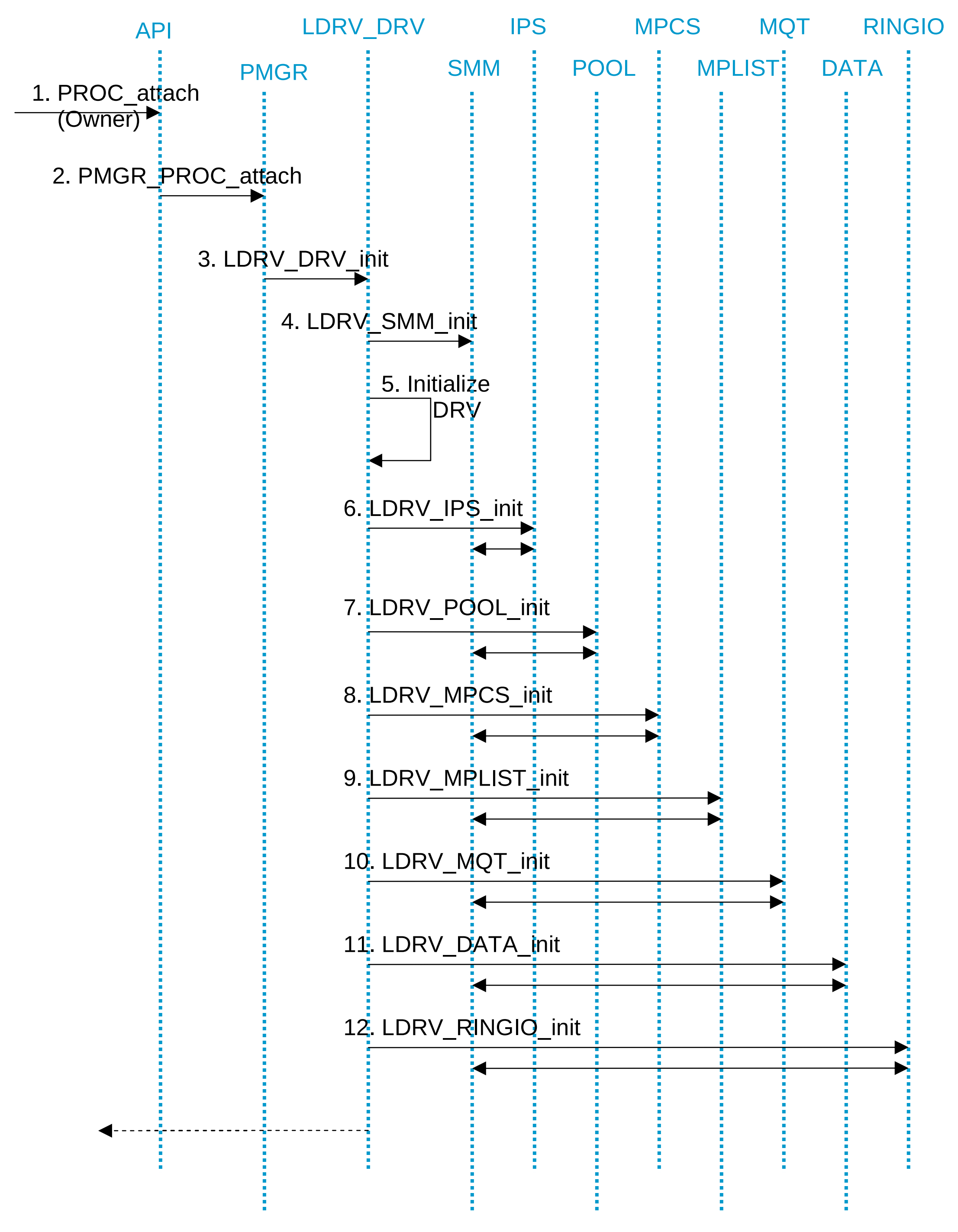

Figure 3. GPP-side initialization

6.3 GPP-side Handshake

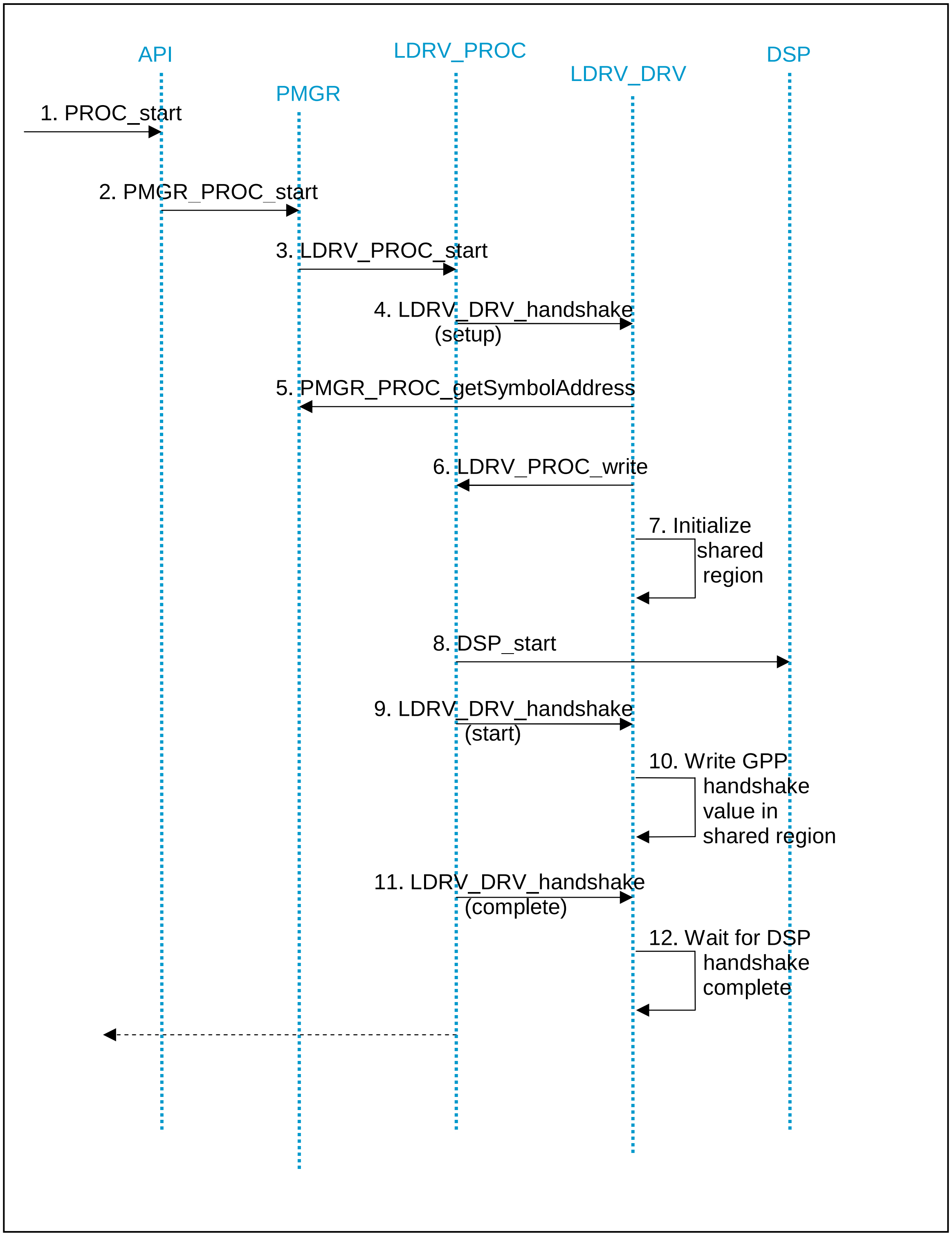
API
LDRV_PROC
LDRV_DRV
DSP
PMGR
1. PROC_start
2. PMGR_PROC_start
3. LDRV_PROC_start
4. LDRV_DRV_handshake (setup)
5. PMGR_PROC_getSymbolAddress
6. LDRV_PROC_write
7. Initialize shared region
8. DSP_start
9. LDRV_DRV_handshake (start)
10. Write GPP handshake value in shared region
11. LDRV_DRV_handshake (complete)
12. Wait for DSP handshake complete

Figure 4. Handshake between GPP and DSP-side

6.4 DSP-side Initialization

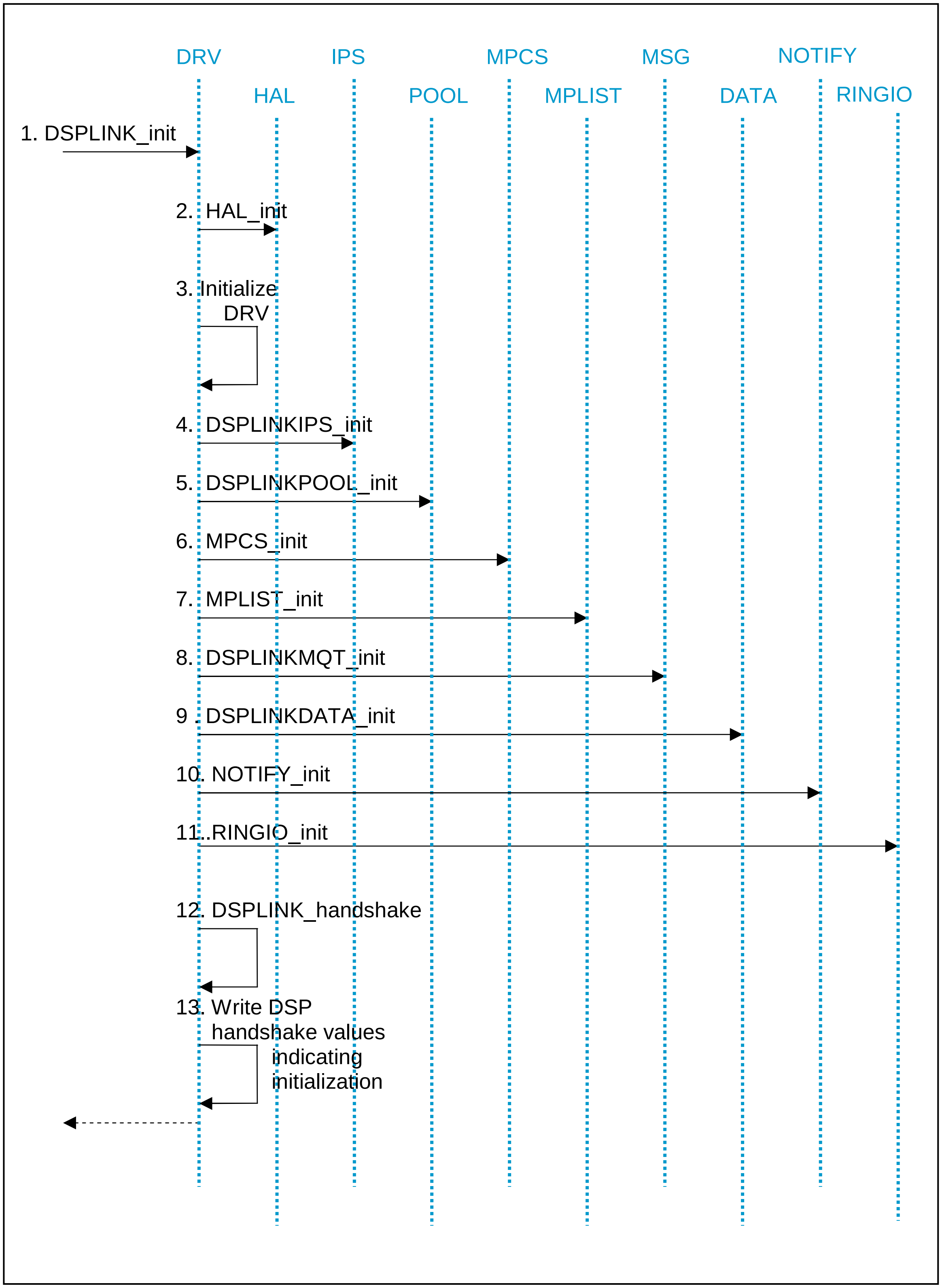

Figure 5. DSP-side initialization

7 API

This section describes the low-level design for the *DSPLINK* dynamic configuration within the API layer.

7.1 Constants & Enumerations

None.

7.2 Typedefs & Data Structures

7.2.1 LINKCFG_Object

This structure defines the object containing all configuration items for DSP/BIOS LINK.

Definition

```
typedef struct LINKCFG_Object_tag {
    LINKCFG_Gpp *             gppObject ;
    Uint32                    numDsps ;
    LINKCFG_Dsp *             dspObjects ;
    LINKCFG_LinkDrv *         linkDrvObjects ;
    Uint32                    numMemTables ;
    LINKCFG_MemEntry **       memTables ;

    Uint32                    numIpsTables ;
    LINKCFG_Ips **            ipsTables ;

    Uint32                    numPoolTables ;
    LINKCFG_Pool **           poolTables ;

    Uint32                    numDataTables ;
    LINKCFG_DataDrv **        dataTables ;

    Uint32                    numMqts ;
    LINKCFG_Mqt *             mqtObjects ;

    Uint32                    numRingIo ;
    LINKCFG_RingIo *          ringIoObjects ;

    Uint32                    numMpList ;
    LINKCFG_MpList *          mplistObjects ;

    Uint32                    numMpcs ;
    LINKCFG_Mpcs *            mpcsObjects ;

    LINKCFG_Log *             logObject ;
} LINKCFG_Object ;
```

Fields

`gppObject`	Pointer to the GPP object.
`numDsps`	Number of DSPs connected to the GPP.
`dspObjects`	Pointer to the array of DSP objects.

`linkDrvObjects`	Pointer to the array of link objects.
`numMemTables`	Number of MEM tables specified in configuration database.
`memTables`	Pointer to the array of memory information table arrays.
`numIpsTables`	Number of IPS tables.
`ipsTables`	Pointer to the array of IPS table arrays.
`numPoolTables`	Number of POOL tables.
`poolTables`	Pointer to the array of POOL table arrays.
`numDataTables`	Number of data tables.
`dataTables`	Pointer to the array of data table arrays.
`numMqts`	Number of Message Queue Transports.
`mqtObjects`	Pointer to the array of MQT objects.
`numRingIo`	Number of RingIO tables.
`ringIoObjects`	Pointer to the array of RingIO tables.
`numMpList`	Number of MPLIST tables.
`mpListObjects`	Pointer to the array of MPLIST tables.
`numMpcs`	Number of MPCS tables.
`mpcsObjects`	Pointer to the array of MPCS tables.
`logObject`	Pointer to the LOG object.

Comments

An instance of the `LINKCFG_Object` is linked into the *DSPLINK* user-side library by default. This contains the default configuration provided with *DSPLINK*. The application may override this configuration by providing a pointer to its own instance of the `LINKCFG_Object` structure.

Constraints

None.

See Also

`LINKCFG_Gpp`
`LINKCFG_Dsp`
`LINKCFG_LinkDrv`
`LINKCFG_MemEntry`
`LINKCFG_Ips`
`LINKCFG_Pool`
`LINKCFG_DataDrv`
`LINKCFG_Mqt`
`LINKCFG_RingIo`

```
LINKCFG_MpList
LINKCFG_Mpcs
LINKCFG_Log
```

7.2.2 LINKCFG_Gpp

This structure defines the configuration structure for the GPP.

Definition

```
typedef struct LINKCFG_Gpp_tag {
    Char8              name [DSP_MAX_STRLEN] ;
    Uint32             maxMsgqs ;
    Uint32             maxChnlQueue ;
    Uint32             poolTableId ;
    Uint32             numPools ;
    Uint32             probeRtcId ;
    Uint32             probeIntId ;
} LINKCFG_Gpp ;
```

Fields

`name`	Name of GPP Processor.
`maxMsgqs`	Maximum MSGQs that can be opened on the GPP.
`maxChnlQueue`	Maximum Queue Length for all channels created on the GPP.
`poolTableId`	POOL table ID to be used for intra-GPP communication. A value of -1 indicates that no POOL is required by the GPP.
`numPools`	Number of POOLs within the pool table for the GPP.
`probeRtcId`	Real Time Clock ID for PROBE.
`probeIntId`	Interrupt ID for PROBE.

Comments

An instance of the `LINKCFG_Gpp` object is provided, with information about the GPP object in the system.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.3 LINKCFG_Dsp

This structure defines the configuration structure for the DSP.

Definition

```
typedef struct LINKCFG_Dsp_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   dspArch ;
    Char8                    loaderName [DSP_MAX_STRLEN] ;
    Bool                     loadSymbols ;
    Bool                     autoStart ;
    Char8                    execName [DSP_MAX_STRLEN] ;
    Bool                     doPowerCtrl ;
    Uint32                   resumeAddr ;
    Uint32                   resetVector ;
    Uint32                   resetCodeSize ;
    Uint32                   maduSize ;
    Uint32                   endian ;
    Uint32                   wordSwap ;
    Uint32                   memTableId ;
    Uint32                   memEntries ;
    Uint32                   linkDrvId ;
} LINKCFG_Dsp ;
```

Fields

`name`	Name of DSP processor.
`dspArch`	Architecture of the DSP.
`loaderName`	Name of loader to be used for loading the DSP executable.
`loadSymbols`	Indicates whether symbols from the DSP executable should be loaded.
`autoStart`	AutoStart flag indicating whether a default DSP image should be loaded on startup. Currently not supported.
`execName`	Name of executable to load in case autostart is used.
`doPowerCtrl`	Indicates whether DSP/BIOS LINK should do the power control for the DSP.
`resumeAddr`	The resume address after hibernating.
`resetVector`	Address of reset vector of DSP.
`resetCodeSize`	Size of code at DSP Reset Vector.
`maduSize`	Minimum addressable unit on the DSP.
`endian`	Endianism info of DSP.
`wordSwap`	Indicates whether words need to be swapped while writing into the memory for the DSP.
`memTableId`	Table number of the MEM entries for this DSP.

`memEntries`	Number of entries in the MEM table.
`linkDrvId`	Link Driver table identifier for this DSP.

Comments

An instance of the `LINKCFG_Dsp` object is provided for each DSP in the system.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.4 LINKCFG_MemEntry

This structure defines an entry in the MEM table.

Definition

```
typedef struct LINKCFG_MemEntry_tag {
    Uint32                   entry ;
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   physAddr ;
    Uint32                   dspVirtAddr ;
    Uint32                   gppVirtAddr ;
    Uint32                   size ;
    Bool                     shared ;
} LINKCFG_MemEntry ;
```

Fields

`entry`	Entry number in the memory table.
`name`	Name identifying the memory region.
`physAddr`	Physical address of the memory region.
`dspVirtAddr`	DSP virtual address of the memory region.
`gppVirtAddr`	GPP virtual address of the memory region. If specified as -1, the GPP virtual address is assumed to be invalid, and shall be set internally within the *DSPLINK* driver.
`size`	Size of the memory region.
`shared`	Flag indicating whether the memory region is shared between GPP and DSP.

Comments

The configuration contains one or more memory tables, each containing a number of memory entries of the type `LINKCFG_MemEntry`. Each DSP object contains information about the memory table for the DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.5 LINKCFG_LinkDrv

This structure defines the configuration information for the physical link driver.

Definition

```
typedef struct LINKCFG_LinkDrv_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   hshkPollCount ;
    Uint32                   memEntry ;
    Uint32                   ipsTableId ;
    Uint32                   numIpsEntries ;
    Uint32                   poolTableId ;
    Uint32                   numPools ;
    Uint32                   dataTableId ;
    Uint32                   numDataDrivers ;
    Uint32                   mqtId ;
    Uint32                   ringIoTableId ;
    Uint32                   mplistTableId ;
    Uint32                   mpcsTableId ;
} LINKCFG_LinkDrv ;
```

Fields

`name`	Name of the physical link driver.
`hshkPollCount`	Poll count to be used for the handshake between GPP and DSP. The GPP spins in a loop for number of iterations equal to the handshake poll count or till the DSP completes its handshake. A value of -1 indicates infinite wait.
`memEntry`	MEM Entry for the memory area used by this physical link. This field is optional and can be specified as -1 in case a MEM entry is not required.
`ipsTableId`	IPS table ID.
`numIpsEntries`	Number of IPS table entries for this link driver.
`poolTableId`	POOL table ID.
`numPools`	Number of POOLs within the pool table for this link driver.
`dataTableId`	Table number of the data driver(s) used with this DSP.
`numDataDrivers`	Number of data drivers used with this DSP.
`mqtId`	The ID of the MQT which is to be used for this DSP.
`ringIoTableId`	Table number of the RINGIO object(s) for this DSP.
`mplistTableId`	Table number of the MPLIST object(s) for this DSP.
`mpcsTableId`	Table number of the MPCS object(s) for this DSP.

Comments

An instance of the `LINKCFG_LinkDrv` object is provided for every DSP in the system, with information about the physical link connecting the GPP and the DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.6 LINKCFG_Ips

This structure defines the Configuration information for the Inter-processor Signaling Component.

Definition

```
typedef struct LINKCFG_Ips_tag {
    Char8                     name [DSP_MAX_STRLEN] ;
    Uint32                    numIpsEvents ;
    Uint32                    memEntry ;
    Uint32                    gppIntId ;
    Uint32                    dspIntId ;
    Uint32                    arg1 ;
    Uint32                    arg2 ;
} LINKCFG_Ips ;
```

Fields

`name`	Name of the IPS component.
`numIpsEvents`	Number of events supported by the IPS.
`memEntry`	MEM entry for the memory area used for this IPS component. This field is optional and can be specified as -1 in case a MEM entry is not required.
`gppIntId`	Interrupt Number to be used by the IPS on GPP-side.
`dspIntId`	Interrupt Number to be used by the IPS on DSP-side.
`arg1`	First IPS-specific optional argument.
`arg2`	Second IPS-specific optional argument.

Comments

The physical link driver between GPP and DSP may have one or more Inter-processor Signaling components connecting the GPP and DSP. Information about each IPS component is provided within the `LINKCFG_Ips` object. Tables of IPS objects are present within the configuration, and each link driver indicates the IPS table ID used by it.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.7 LINKCFG_Pool

This structure defines the configuration information for a buffer pool, from which buffers for use with DSPLINK can be allocated through the POOL interface(s).

Definition

```
typedef struct LINKCFG_Pool_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   memEntry ;
    Uint32                   poolSize ;
    Uint32                   ipsId ;
    Uint32                   ipsEventNo ;
    Uint32                   arg1 ;
    Uint32                   arg2 ;
} LINKCFG_Pool ;
```

Fields

`name`	Name of the pool.
`memEntry`	MEM Entry for the memory area used by this buffer pool. This field is optional and can be specified as -1 in case a MEM entry is not required.
`poolSize`	Size of the buffer pool.
`ipsId`	ID of the IPS used (if any). A value of -1 indicates that no IPS is required by the pool.
`ipsEventNo`	IPS Event number associated with POOL (if any). A value of -1 indicates that no IPS is required by the pool.
`arg1`	First optional pool-specific argument.
`arg2`	Second optional pool-specific argument.

Comments

Multiple pools may be configured within the system. Configuration information about one pool instance is present in an instance of the `LINKCFG_Pool` object. The pool instance is associated with pool ID matching the index number of the pool in the pool table.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.8 LINKCFG_DataDrv

This structure defines the configuration structure for the data streaming driver using the CHNL component.

Definition

```
typedef struct LINKCFG_DataDrv_tag {
    Char8                     name [DSP_MAX_STRLEN] ;
    Uint32                    baseChnlId ;
    Uint32                    numChannels ;
    Uint32                    maxBufSize ;
    Uint32                    memEntry ;
    Uint32                    poolId ;
    Uint32                    queuePerChnl ;
    Uint32                    ipsId ;
    Uint32                    ipsEventNo ;
    Uint32                    arg1 ;
    Uint32                    arg2 ;
} LINKCFG_DataDrv ;
```

Fields

`name`	Name of the data transfer driver.
`baseChnlId`	Base channel ID for this data driver.
`numChannels`	Number of logical channels for this data driver.
`maxBufSize`	Maximum size of data buffer supported by this data driver. If no limit is imposed by the driver, a value of -1 can be specified.
`memEntry`	MEM entry for the memory area for data streaming driver. This field is optional and can be specified as -1 in case a MEM entry is not required.
`poolId`	Identifier for the pool from where buffers are allocated.
`queuePerChnl`	Buffer Queue length on each channel supported by the data driver.
`ipsId`	ID of the IPS to be used (if any). A value of -1 indicates that no IPS is required by the data driver.
`ipsEventNo`	IPS Event number associated with data driver (if any). A value of -1 indicates that no IPS is required by the data driver.
`arg1`	First optional data driver specific argument. The significance of this argument is specific to a data driver.
`arg2`	Second optional data driver specific argument 2. The significance of this argument is specific to a data driver.

Comments

Each instance of the `LINKCFG_DataDrv` object contains configuration information about a data streaming driver connecting the GPP and a DSP. The data driver is used by the CHNL component. Each GPP-DSP connection may have one or more data drivers, indicated by the data table ID within the DSP object. The data driver to be used between the GPP and DSP is identified on the basis of the channel ID used. The base channel ID and maximum channels are specified by the data driver as part of its configuration.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.9 LINKCFG_Mqt

This structure defines the configuration structure for the Message Queue Transport.

Definition

```
typedef struct LINKCFG_Mqt_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   memEntry ;
    Uint32                   maxMsgSize ;
    Uint32                   ipsId ;
    Uint32                   ipsEventNo ;
    Uint32                   arg1 ;
    Uint32                   arg2 ;
} LINKCFG_Mqt ;
```

Fields

`name`	Name of the MQT.
`memEntry`	MEM entry id for the memory area used by this MQT. This field is optional and can be specified as -1 in case a MEM entry is not required.
`maxMsgSize`	Maximum size of message supported by MQT. May be -1 if there is no limit on maximum message size for the MQT.
`ipsId`	ID of the IPS to be used (if any). A value of -1 indicates that no IPS is required by the MQT.
`ipsEventNo`	IPS Event number associated with MQT (if any). A value of -1 indicates that no IPS is required by the MQT.
`arg1`	First optional argument for this MQT. The significance of this argument is specific to the MQT.
`arg2`	Second optional argument for this MQT. The significance of this argument is specific to the MQT.

Comments

Each instance of the `LINKCFG_Mqt` object provides configuration information about a Message Queue Transport connecting the GPP and a DSP. There can be only one MQT configured at a time between the GPP and each DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.10 LINKCFG_RingIo

This structure defines the configuration structure for the RingIO component.

Definition

```
typedef struct LINKCFG_RingIo_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   memEntry ;
    Uint32                   maxEntries ;
    Uint32                   ipsId ;
    Uint32                   ipsEventNo ;
} LINKCFG_RingIo ;
```

Fields

`name`	Name of the RingIO.
`memEntry`	MEM entry ID for the memory area used by this RingIO.
`maxEntries`	Maximum number of RingIO instances supported by the RingIO.
`ipsId`	ID of the IPS to be used.
`ipsEventNo`	IPS Event number associated with the RingIO.

Comments

Each instance of the `LINKCFG_RingIo` object provides configuration information about a RingIO transport connecting the GPP and a DSP. There can be only one RingIO transport configured at a time between the GPP and each DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.11 LINKCFG_MpList

This structure defines the configuration structure for the MPLIST component.

Definition

```
typedef struct LINKCFG_MpList_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   memEntry ;
    Uint32                   maxEntries ;
    Uint32                   ipsId ;
    Uint32                   ipsEventNo ;
} LINKCFG_MpList ;
```

Fields

`name`	Name of the MPLIST.
`memEntry`	MEM entry ID for the memory area used by this MPLIST.
`maxEntries`	Maximum number of MPLIST instances supported by the MPLIST.
`ipsId`	ID of the IPS to be used (if any). A value of -1 indicates that no IPS is required by the MPLIST.
`ipsEventNo`	IPS Event number associated with MPLIST (if any). A value of -1 indicates that no IPS is required by the MPLIST.

Comments

Each instance of `the LINKCFG_MpList` object provides configuration information about an MPLIST transport connecting the GPP and a DSP. There can be only one MPLIST transport configured at a time between the GPP and each DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.12 LINKCFG_Mpcs

This structure defines the configuration structure for the MPCS component.

Definition

```
typedef struct LINKCFG_Mpcs_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    Uint32                   memEntry ;
    Uint32                   maxEntries ;
    Uint32                   ipsId ;
    Uint32                   ipsEventNo ;
} LINKCFG_Mpcs ;
```

Fields

`name`	Name of the MPCS.
`memEntry`	MEM entry ID for the memory area used by this MPCS.
`maxEntries`	Maximum number of MPCS instances supported by the MPCS.
`ipsId`	ID of the IPS to be used (if any). A value of -1 indicates that no IPS is required by the MPCS.
`ipsEventNo`	IPS Event number associated with MPLIST (if any). A value of -1 indicates that no IPS is required by the MPCS.

Comments

Each instance of the `LINKCFG_Mpcs` object provides configuration information about an MPCS transport connecting the GPP and a DSP. There can be only one MPCS transport configured at a time between the GPP and each DSP.

Constraints

None.

See Also

`LINKCFG_Object`

7.2.13 LINKCFG_Log

This structure defines the configuration structure for the LOG component.

Definition

```
typedef struct LINKCFG_Log_tag {
    Bool                    gdMsgqPut ;
    Bool                    gdMsgqSendInt ;
    Bool                    gdMsgqIsr ;
    Bool                    gdMsgqQue ;

    Bool                    dgMsgqPut ;
    Bool                    dgMsgqSendInt ;
    Bool                    dgMsgqIsr ;
    Bool                    dgMsgqQue ;

    Bool                    gdChnlIssueStart ;
    Bool                    gdChnlIssueQue ;
    Bool                    gdChnlIssueCompl ;

    Bool                    gdChnlXferStart ;
    Bool                    gdChnlXferProcess ;
    Bool                    gdChnlXferCompl ;

    Bool                    gdChnlReclStart ;
    Bool                    gdChnlReclPend ;
    Bool                    gdChnlReclPost ;
    Bool                    gdChnlReclCompl ;

    Bool                    dgChnlIssueQue ;

    Bool                    dgChnlXferStart ;
    Bool                    dgChnlXferProcess ;
    Bool                    dgChnlXferCompl ;

    Bool                    dgChnlReclPend ;
    Bool                    dgChnlReclPost ;

    Uint32                  msgIdRangeStart ;
    Uint32                  msgIdRangeEnd ;
} LINKCFG_Log ;
```

Fields

`gdMsgqPut`	GPP->DSP MSG Transfer - MSGQ_Put call.
`gdMsgqSendInt`	GPP->DSP MSG Transfer - GPP sends interrupt.
`gdMsgqIsr`	GPP->DSP MSG Transfer - DSP receives interrupt.
`gdMsgqQue`	GPP->DSP MSG Transfer - Message queued at DSP.
`dgMsgqPut`	DSP->GPP MSG Transfer - MSGQ_Put call.
`dgMsgqSendInt`	DSP->GPP MSG Transfer - DSP sends interrupt.
`dgMsgqIsr`	DSP->GPP MSG Transfer - GPP receives interrupt.

`dgMsgqQue`	DSP->GPP MSG Transfer - Message queued at GPP.
`gdChnlIssueStart`	GPP->DSP CHNL Transfer - Entering inside ISSUE call.
`gdChnlIssueQue`	GPP->DSP CHNL Transfer - ISSUE: Buffer is queued in internal
`gdChnlIssueCompl`	GPP->DSP CHNL Transfer - ISSUE call completed.
`gdChnlXferStart`	GPP->DSP CHNL Transfer - Initiating a buffer transfer by GPP.
`gdChnlXferProcess`	GPP->DSP CHNL Transfer - Actual transfer of buffer is going to take place.
`gdChnlXferCompl`	GPP->DSP CHNL Transfer - Buffer transfer is complete.
`gdChnlReclStart`	GPP->DSP CHNL Transfer - Entering RECLAIM call.
`gdChnlReclPend`	GPP->DSP CHNL Transfer - RECLAIM: Wait on a semaphore.
`gdChnlReclPost`	GPP->DSP CHNL Transfer - RECLAIM: Posting the Semaphore.
`gdChnlReclCompl`	GPP->DSP CHNL Transfer - RECLAIM call completed.
`dgChnlIssueQue`	DSP->GPP CHNL Transfer - ISSUE: Buffer is queued in internal structure on DSP.
`dgChnlXferStart`	DSP->GPP CHNL Transfer - Initiating a buffer transfer by DSP.
`dgChnlXferProcess`	DSP->GPP CHNL Transfer - Actual transfer of buffer is going to take place.
`dgChnlXferCompl`	DSP->GPP CHNL Transfer - Buffer transfer is complete.
`dgChnlReclPend`	DSP->GPP CHNL Transfer - RECLAIM: Wait on a semaphore.
`dgChnlReclPost`	DSP->GPP CHNL Transfer - RECLAIM: Posting the Semaphore.
`msgIdRangeStart`	MSG ID range: lower limit.
`msgIdRangeEnd`	MSG ID range: upper limit.

Comments

An instance of the `LINKCFG_Log` object is provided, with information about the LOG component in the system.

Constraints

None.

See Also

`LINKCFG_Object`

7.3 API Definition

7.3.1 PROC_setup

This function sets up the necessary data structures for the PROC sub-component.

Syntax

```
DSP_STATUS PROC_setup (LINKCFG_Object * linkCfg) ;
```

Arguments

```
IN OPT   LINKCFG_Object *           linkCfg
```

Pointer to the configuration information structure for DSP/BIOS™ LINK. If NULL, indicates that default configuration should be used.

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

A version of this API is also provided, which does not take any parameters, for backward compatibility with existing applications:

```
#define PROC_Setup  PROC_setup (NULL)
```

Constraints

None.

See Also

`LINKCFG_Object`

8 LDRV

This section describes the low-level design for the *DSPLINK* dynamic configuration within the link driver layer.

8.1 Constants & Enumerations

None.

8.2 Typedefs & Data Structures

8.2.1 CFGMAP_Object

This structure defines the object containing all configuration mapping information for DSP/BIOS LINK.

Definition

```
typedef struct CFGMAP_Object_tag {
    Uint32                   numDsps ;
    CFGMAP_Dsp *             dspObjects ;

    Uint32                   numLoaders ;
    CFGMAP_Loader *          loaders ;

    Uint32                   numLinkDrvs ;
    CFGMAP_LinkDrv *         linkDrvObjects ;

    Uint32                   numIps ;
    CFGMAP_Ips *             ipsObjects ;

#if defined (POOL_COMPONENT)
    Uint32                   numPools ;
    CFGMAP_Pool *            poolObjects ;
#endif /* if defined (POOL_COMPONENT) */

#if defined (CHNL_COMPONENT)
    Uint32                   numDataDrivers ;
    CFGMAP_DataDrv *         dataObjects ;
#endif /* if defined (CHNL_COMPONENT) */

#if defined (MSGQ_COMPONENT)
    Uint32                   numMqts ;
    CFGMAP_Mqt *             mqtObjects ;
#endif /* if defined (MSGQ_COMPONENT) */
} CFGMAP_Object ;
```

Fields

`numDsps`	Number of DSPs supported for this platform.
`dspObjects`	Pointer to the array of DSP configuration mapping objects.
`numLoaders`	Number of DSP executable loaders supported for this platform.

`loaders`	Pointer to the array of DSP executable loader configuration mapping objects.
`numLinkDrvs`	Number of types of link drivers supported for this platform.
`linkDrvObjects`	Pointer to the array of link driver configuration mapping objects.
`numIps`	Number of different types of IPS supported for this platform.
`ipsObjects`	Pointer to the array of IPSs configuration mapping objects.
`numPools`	Number of different types of POOLs supported for this platform. Only defined if POOL_COMPONENT is used.
`poolObjects`	Pointer to the array of POOL configuration mapping objects. Only defined if POOL_COMPONENT is used.
`numDataDrivers`	Number of different types of data drivers supported for this platform. Only defined if CHNL_COMPONENT is used.
`dataObjects`	Pointer to the array of data driver configuration mapping objects. Only defined if CHNL_COMPONENT is used.
`numMqts`	Number of different types of MQTs supported for this platform. Only defined if MSGQ_COMPONENT is used.
`mqtObjects`	Pointer to the array of MQT configuration mapping objects. Pointer to the array of MQT configuration mapping objects.

Comments

An instance of the `CFGMAP_Object` object provides information that maps function table interfaces and function pointers for required modules to the user-level configuration.

Constraints

None.

See Also

`CFGMAP_Dsp`
`CFGMAP_Loader`
`CFGMAP_LinkDrv`
`CFGMAP_Ips`
`CFGMAP_Pool`
`CFGMAP_DataDrv`
`CFGMAP_Mqt`

8.2.2 CFGMAP_Dsp

This structure defines the configuration mapping structure for the DSP.

Definition

```
typedef struct CFGMAP_Dsp_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    DSP_Interface *          interface ;
} CFGMAP_Dsp ;
```

Fields

`name`	Name of DSP processor.
`interface`	Function pointer interface table for accessing the DSP.

Comments

Each instance of the `CFGMAP_Dsp` object provides configuration mapping information for each type of DSP.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.3 CFGMAP_Loader

This structure defines the configuration mapping structure for the DSP executable loader.

Definition

```
typedef struct CFGMAP_Loader_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    LOADER_Interface *       interface ;
} CFGMAP_Loader ;
```

Fields

`name`	Name of DSP executable loader.
`interface`	Function pointer interface table for the DSP executable loader.

Comments

Each instance of the `CFGMAP_Loader` object provides configuration mapping information for each type of DSP executable loader.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.4 CFGMAP_LinkDrv

This structure defines the configuration mapping structure for the link driver.

Definition

```
typedef struct CFGMAP_LinkDrv_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    DRV_Interface *          interface ;
} CFGMAP_LinkDrv ;
```

Fields

`name`	Name of link driver.
`interface`	Name of link driver.

Comments

Each instance of the `CFGMAP_LinkDrv` object provides configuration mapping information for each type of link driver.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.5 CFGMAP_Ips

This structure defines the configuration mapping structure for the IPS component.

Definition

```
typedef struct CFGMAP_Ips_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    FnIpsInit                init ;
    FnIpsExit                exit ;
#if defined (DDSP_DEBUG)
    FnIpsDebug               debug ;
#endif /* if defined (DDSP_DEBUG) */
} CFGMAP_Ips ;
```

Fields

`name`	Name of IPS.
`init`	Function pointer for the init function.
`exit`	Function pointer for the exit function.
`debug`	Function pointer for the exit function. Defined only if DDSP_DEBUG is enabled.

Comments

Each instance of the `CFGMAP_Ips` object provides configuration mapping information for each type of IPS.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.6 CFGMAP_Pool

This structure defines the configuration mapping structure for the POOL component.

Definition

```
typedef struct CFGMAP_Pool_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    FnPoolInit               init ;
    FnPoolExit               exit ;
    POOL_Interface *         interface ;
} CFGMAP_Pool ;
```

Fields

`name`	Name of the pool.
`init`	Initialization function for the pool.
`exit`	Finalization function for the pool.
`interface`	Function pointer interface table for the pool.

Comments

Each instance of the `CFGMAP_Pool` object provides configuration mapping information for each type of pool.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.7 CFGMAP_DataDrv

This structure defines the configuration mapping structure for the DATA driver component.

Definition

```
typedef struct CFGMAP_DataDrv_tag {
    Char8                    name [DSP_MAX_STRLEN] ;
    DATA_Interface *         interface ;
} CFGMAP_DataDrv ;
```

Fields

`name`	Name of the data transfer driver.
`interface`	Function pointer interface table for the data driver.

Comments

Each instance of the `CFGMAP_DataDrv` object provides configuration mapping information for each type of data transfer driver.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.8 CFGMAP_Mqt

This structure defines the configuration mapping structure for the MQT component.

Definition

```
typedef struct CFGMAP_Mqt_tag {
    Char8                     name [DSP_MAX_STRLEN] ;
    MQT_Interface *           interface ;
} CFGMAP_Mqt ;
```

Fields

`name`	Name of the MQT.
`interface`	Function pointer interface table for the MQT.

Comments

Each instance of the `CFGMAP_Mqt` object provides configuration mapping information for each type of Message Queue Transport.

Constraints

None.

See Also

`CFGMAP_Object`

8.2.9 LDRV_SMM_MemObject

This structure defines the memory object managed by the Shared Memory Manager (SMM). This object contains all information about the shared memory region required by the SMM.

Definition

```
typedef struct LDRV_SMM_MemObject_tag {
    LINKCFG_MemEntry * memEntry ;
    Uint32             curFreeOffset ;
} LDRV_SMM_MemObject ;
```

Fields

`memEntry`	Pointer to memory entry in the configuration.
`curFreeOffset`	Current free offset within the shared memory region. The memory from this offset onwards within this region is free to be allocated.

Comments

An array of the `LDRV_SMM_MemObject` objects is maintained and managed within the SMM component.

Constraints

None.

See Also

`LDRV_SMM_Object`

8.2.10 LDRV_SMM_Object

This structure defines the Shared Memory Manager (SMM) object, which contains all state information required by the Shared Memory Manager.

Definition

```
typedef struct LDRV_SMM_Object_tag {
    Uint32                numMemEntries ;
    LDRV_SMM_MemObject *  memTable ;
} LDRV_SMM_Object ;
```

Fields

`numMemEntries`	Number of memory entries within the memory table.
`memTable`	Array of SMM memory objects.

Comments

One instance of the `LDRV_SMM_Object` is present for every DSP in the system.

Constraints

None.

See Also

`LDRV_SMM_MemObject`
`LDRV_SMM_init ()`

8.3 API Definition

8.3.1 LDRV_init

This function initializes the LDRV component. The implementation of this function is OS-independent.

Syntax

```
DSP_STATUS LDRV_init (LINKCFG_Object * linkCfg) ;
```

Arguments

`IN`	`LINKCFG_Object *`	`linkCfg`
	Pointer to the user-side configuration object.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_ECONFIG`	Incorrect configuration.
`DSP_EFAIL`	General failure.

Comments

This function initializes the LDRV component. It calls the module initialization functions for all LDRV sub-components after getting the pointer to configuration information (`LDRV_LinkCfgPtr`). The validity of configuration values provided by the user is verified during the execution of this function.

Constraints

None.

See Also

`LDRV_exit ()`

8.3.2 LDRV_exit

This function finalizes the LDRV component. The implementation of this function is OS-independent.

Syntax

```
DSP_STATUS LDRV_exit (Void) ;
```

Arguments

None.

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function clears configuration information stored within the LDRV component (`LDRV_LinkCfgPtr`) after calling the module finalization functions for all LDRV sub-components.

Constraints

None.

See Also

`LDRV_init ()`

8.3.3 LDRV_getLinkCfg

This function gets the pointer to kernel configuration structure after creating it (if required). The implementation of this function is OS-specific.

Syntax

```
DSP_STATUS LDRV_getLinkCfg (LINKCFG_Object *  linkCfg,
                            LINKCFG_Object ** knlLinkCfg) ;
```

Arguments

`IN`	`LINKCFG_Object *`	`linkCfg`
	Pointer to the user-side configuration object.	
`OUT`	`LINKCFG_Object **`	`knlLinkCfg`
	Location to receive the pointer to the kernel-side configuration object.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_ECONFIG`	Incorrect configuration.
`DSP_EFAIL`	General failure.

Comments

This function returns a pointer to the kernel-side configuration structure. The implementation of this function varies depending on the Operating System. For OSes such as PrOS with no user-kernel separation, this function simply sets the returns the specified pointer to the user-level configuration structure. For OSes such as Linux, having user-kernel separation, this function allocates memory for the configuration sub-structures in kernel memory space and copies the contents of the specified configuration structure from user-space.

Constraints

None.

See Also

`LDRV_freeLinkCfg ()`

8.3.4 LDRV_freeLinkCfg

This function frees any memory allocated for the kernel-side DSPLINK configuration structure. The implementation of this function is OS-specific.

Syntax

```
DSP_STATUS LDRV_freeLinkCfg (LINKCFG_Object *  knlLinkCfg) ;
```

Arguments

`IN`	`LINKCFG_Object *`	`knlLinkCfg`
	Pointer to the kernel-side configuration object.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_ECONFIG`	Incorrect configuration.
`DSP_EFAIL`	General failure.

Comments

This function frees any memory allocated for the kernel-side configuration structure. The implementation of this function varies depending on the Operating System. For OSes such as PrOS with no user-kernel separation, this function does not perform any actions. For OSes such as Linux, having user-kernel separation, this function frees any memory allocated for the kernel-side configuration structure and sub-structures allocated during initialization.

Constraints

None.

See Also

`LDRV_getLinkCfg ()`

8.3.5 LDRV_SMM_moduleInit

This function initializes the LDRV_SMM module.

Syntax

```
DSP_STATUS LDRV_SMM_moduleInit (Void) ;
```

Arguments

None.

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_ECONFIG`	Incorrect configuration.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function is called once for the LDRV_SMM sub-component. It is called during initialization of the LDRV component, done during `PROC_setup` ().

This function initializes the fields of LDRV_SMM state objects for all DSPs to indicate that the LDRV_SMM component is not initialized.

Constraints

None.

See Also

`LDRV_SMM_moduleExit ()`

8.3.6 LDRV_SMM_moduleExit

This function finalizes the LDRV_SMM module.

Syntax

```
DSP_STATUS LDRV_SMM_moduleExit (Void) ;
```

Arguments

None.

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_ECONFIG`	Incorrect configuration.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function is called once for the LDRV_SMM sub-component. It is called during finalization of the LDRV component, done during `PROC_destroy` ().

This function resets the fields of LDRV_SMM state objects for all DSPs.

Constraints

None.

See Also

`LDRV_SMM_moduleInit ()`

8.3.7 LDRV_SMM_init

This function initializes the Shared Memory Manger (SMM) component.

Syntax

```
DSP_STATUS LDRV_SMM_init (ProcessorId dspId) ;
```

Arguments

`IN`	`ProcessorId`	`dspId`
	ID of the DSP for which the SMM component is to be initialized.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function is called once for every DSP in the system. It is called during initialization of the LDRV_DRV component for each DSP.

This function searches within the *DSPLINK* configuration for the memory table for the specified DSP, and initializes itself for the memory regions that are marked as shared.

Constraints

None.

See Also

`LDRV_SMM_exit ()`

8.3.8 LDRV_SMM_exit

This function finalizes the Shared Memory Manger (SMM) component.

Syntax

```
DSP_STATUS LDRV_SMM_exit (ProcessorId dspId) ;
```

Arguments

`IN`	`ProcessorId`	`dspId`
	ID of the DSP for which the SMM component is to be finalized.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function is called once for every DSP in the system. It is called during finalization of the LDRV_DRV component for each DSP.

This function frees any memory allocated during initialization of the SMM component, and finalizes the state object.

Constraints

None.

See Also

`LDRV_SMM_init ()`

8.3.9 LDRV_SMM_alloc

This function allocates a chunk of memory of the requested size from the specified shared memory region.

Syntax

```
DSP_STATUS LDRV_SMM_alloc (ProcessorId dspId,
                           Uint32      memEntryId,
                           Uint32 *    physAddr,
                           Uint32 *    dspVirtAddr,
                           Uint32 *    gppVirtAddr,
                           Uint32      size) ;
```

Arguments

`IN` `ProcessorId` `dspId`

ID of the DSP with which the memory region is shared.

`IN` `Uint32` `memEntryId`

ID of the memory entry from which memory is to be allocated.

`OUT` `Uint32 *` `physAddr`

Location to receive the physical address of the allocated memory chunk. If NULL, the address is not returned.

`OUT` `Uint32 *` `dspVirtAddr`

Location to receive the DSP virtual address of the allocated memory chunk. If NULL, the address is not returned.

`OUT` `Uint32 *` `gppVirtAddr`

Location to receive the GPP kernel virtual address of the allocated memory chunk. If NULL, the address is not returned.

`IN` `Uint32` `size`

Size of the memory chunk to be allocated.

Return Value

DSP_SOK	Operation successfully completed.
DSP_EVALUE	Memory entry ID does not correspond to a shared region.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Comments

This function is called by all the *DSPLINK* components to allocate their shared memory requirements.

Constraints

None.

See Also

`LDRV_SMM_free ()`

8.3.10 LDRV_SMM_free

This function frees the chunk of memory of the requested size into the specified shared memory region.

Syntax

```
DSP_STATUS LDRV_SMM_free (ProcessorId dspId,
                          Uint32      memEntryId,
                          Uint32      size) ;
```

Arguments

`IN`	`ProcessorId`	`dspId`
	ID of the DSP with which the memory region is shared.	
`IN`	`Uint32`	`memEntryId`
	ID of the memory entry into which memory is to be freed.	
`IN`	`Uint32`	`size`
	Size of the memory chunk to be freed.	

Return Value

`DSP_SOK`	Operation successfully completed.
`DSP_EVALUE`	Memory entry ID does not correspond to a shared region.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EFAIL`	General failure.

Comments

This function is called by all the *DSPLINK* components to free their shared memory allocations.

Constraints

The allocated memory regions must be freed in the reverse order of their allocation. Due to this restriction, the address of memory chunk to be freed is not required for this function.

See Also

`LDRV_SMM_alloc ()`