DSP/BIOS™ LINK

SHARED MEMORY IOM DRIVER FOR OMAP 5910/5912

LNK 019 DES

Version 1.02

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 explains the design of DSP/BIOS™ LINK for OMAP5910/OMAP5912 using shared memory. Its intended audience is DSP/BIOS™ LINK development team. This document describes the shared memory protocol used for data transfer between GPP and DSP and also outlines the lower level design for IOM based DSP/BIOS™ LINK driver.

1.2 Terms and Abbreviations

OMAP5910 / OMAP5912	TI's dual core chip having ARM and DSP cores.
PCI	Peripheral Component Interconnect
USB	Universal Serial Bus
	This is not used anywhere in the doc.
	This is not used anywhere in the doc.
DSP/BIOS™	TI's OS for DSPs
GPP	Micro-Processor Unit (which controls the DSP)
DSP	Digital Signal Processor
IOM	Input Output Manager
SIO	Standard Input Output
HAL	Hardware Abstraction Layer (of LINK Implementation)

1.3 References

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

1.4 Overview

DSP/BIOS LINK is runtime software, and associated porting kit that simplifies the development of embedded applications in which a general-purpose microprocessor 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.

The purpose of this product is to provide customers with a standard GPP-DSP communication link that also includes support for common operations such as booting and overlay management. This eliminates the need for a customer to develop it from scratch.

This document presents the design of DSP side components of DSP/BIOS LINK.

2 High Level Design

DSP side of DSP/BIOS™ LINK is implemented as a BIOS driver that communicates with the GPP side driver. DSP side user applications of DSP/BIOS™ LINK can use either SIO or IOM or any other class driver API for accessing LINK services.

Communication channels are conceptual entities in DSP/BIOS™ LINK, which are conduits used to communicate data between GPP and DSP. Channels can be addressed by specifying their number. These channels are unidirectional, which means a single channel can transfer data either from GPP to DSP or from DSP to GPP. DSP/BIOS™ LINK supports multiple links (communication hardware components) for transfer of data. Some examples of these links are USB, PCI, Serial Port, Shared Memory, Shared Memory with DMA, Shared Memory using pointer passing etc. The hardware to be used for data transfer is decided based on the channel identifier.

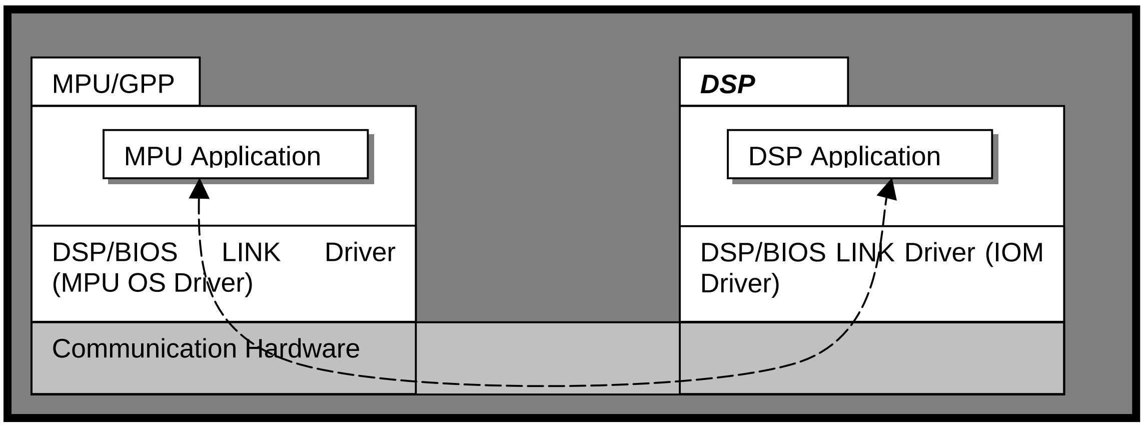

Figure 1. DSP/BIOS LINK provides uniform API for communication irrespective of the underling hardware/method used for communication

In first phase of development only shared memory based communication using processor copy is supported.

LINK Drivers on the DSP are of two kinds:

1. Shared Memory based LINK Drivers
2. Message based LINK Drivers (i.e. PCI, USB etc)

This document discusses the shared memory based link drivers. The fundamental difference between the two types of drivers is the protocol used for communication.

Note that DSP accesses the shared memory using EMIF hardware. Depending upon the DSP chip that is used, there can be different versions of EMIF hardware. DSP side IOM driver is designed such that if EMIF changes, it impacts minimum code.

On DSP/BIOS, every link driver that uses different hardware will exist as different IOM driver.

The following diagram illustrates the relationship between the different subcomponents of LINK driver and its interfaces with the external components. As shown in the diagram LINK mini-driver internally consists of mainly three modules as shown in Figure 2. Each of the module exports an interface to other module or external entity like IOM.

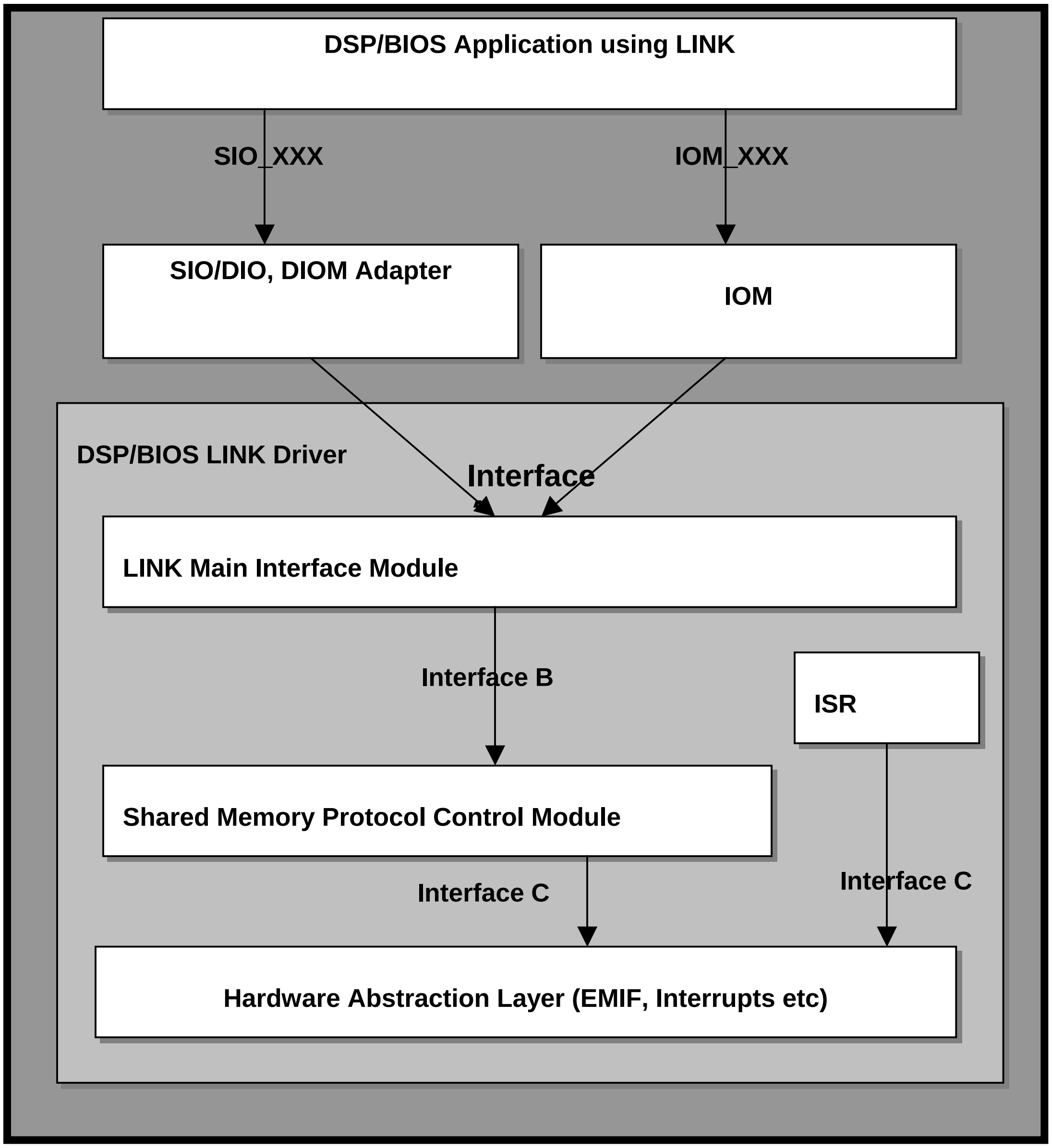

Figure 2. Overall Structure of DSP/BIOS LINK driver for shared memory based Links

The following table describes the key responsibilities of each subcomponent in brief.

Module	Functionality	Exported Interface
LINK Main Interface Module	1. Implements mini-driver interface as required by the IOM model. 2. Handles buffering of IO requests. 3. Handles synchronization issues.	Interface-A (see section 3.2; also see BIOS IOM documentation for details of this interface)
Shared Memory Protocol Control Module	1. Provides interface to LINK Main Interface Module for shared memory protocol specific information. 2. Any small change or enhancement in the protocol will only be absorbed here. 3. Implements shared memory protocol state machine. 4. It gives functionality to read/write buffers by using interface-C of Physical Access Module.	Interface-B (see section 3.3 for more details of this interface)
Hardware Abstraction Layer	1. Primary purpose of this module is to isolate all the hardware specific details from the rest of the module and provide clean interface to read/write shared memory. 2. Provides interface to register ISRs.	Interface-C (see section 3.4 for more details of this interface)

3 Low Level Design

3.1 Type Definitions

3.1.1 LINK_DevParams

Device parameters of LINK.

Definition

```
typedef struct LINK_DevParams
{
Int pid          ;
Int numChannels  ;
Ptr shmConfig    ;
} LINK_DevParams;
```

Fields

`pid`	Processor identifier. This is not used currently.
`numChannels`	Number of channels that can be opened for this device.
`shmConfig`	Optional configuration of shared memory hardware i.e. EMIF. Not supported currently. NULL gives default values.

Comments

None.

Constraints

None.

See Also

None.

3.1.2 LINK_ShmConfig

Configuration parameters of shared memory.

Definition

```
typedef struct LINK_ShmConfig
{
Ptr startAddress  ;
Int maxMemSize    ;
Ptr shmHwConfig   ;
} LINK_ShmConfig;
```

Fields

`startAddress`	Starting address of the shared memory control structure.
`maxMemSize`	Configuration parameters of shared memory.
`shmHwConfig`	Pointer to the EMIF configuration structure. Pass NULL to use default configuration.

Comments

None.

Constraints

None.

See Also

None.

3.1.3 LINK_ChanParams

Channel parameters to be passed while creating channels.

Definition

```
typedef struct LINK_ChanParams {Int maxBufferSize;
Int maxPendingIOs;
} LINK_ChanParams;
```

Fields

maxBufferSize	Maximum size of buffer for this channel.
maxPendingIOs	Maximum IO requests that can spend on this channel.

Comments

None.

Constraints

None.

See Also

None.

3.1.4 LINK_ChannelObject

Channel object of LINK device.

Definition

```
typedef struct LINK_ChannelObject {Uns inUse                      ;
    Uns chanId                      ;
    Uns mode                        ;
    struct LINK_DevObject_tag *dev  ;
    QUE_Obj pendingIOQue            ;
    Uns maxBufferSize               ;
    Uns maxPendingIOs               ;
    Uns currentPendingIOs           ;
    IOM_TiomCallback cbFxn          ;
    Ptr cbArg                       ;
} LINK_ChannelObject;
```

Fields

inUse	Non-zero value means this channel is in use.

`chanId`	Channel identifier
`mode`	Mode of channel. Mode can be input or output.
`dev`	Reference to LINK device structure.
`pendingIOQue`	Queue for pending IO packets.
`maxBufferSize`	Maximum size of buffer that this channel supports.
`maxPendingIOs`	Maximum number of IOs that this channel can have.
`currentPendingIOs`	Number of pending IO request on this channel.
`cbFxn`	IOM callback function.
`cbArg`	Argument to callback function.

Comments

None.

Constraints

None.

See Also

None.

3.1.5 LINK_DevObject

LINK device structure.

Definition

```
typedef struct LINK_DevObject{Uns inUse   ;
Uns numChannels                           ;
Uns dspDataMask                           ;
Uns lastOutputChannel                     ;
LINK_ChannelObject chanObj [MAX_CHANNELS] ;
} LINK_DevObject;
```

Fields

`inUse`	Non-zero value means this LINK device is in use.
`numChannels`	Maximum channels supported by this device.
`dspDataMask`	Tells on which channels output buffer available.
`lastOutputChannel`	Variable indicating on which channel last output was done.
`chanObj`	Array of channel objects that belong to this device.

Comments

None.

Constraints

None.

See Also

None.

4 LINK Main Interface Module

Primary functionality of this module is to implement the mini-driver interface functions (interface-A). These functions handle queuing of requests and calling callback functions of higher entities (IOM, DIOM etc).

This module calls the functions of Shared Memory Protocol Control Module though the interface-B. Following is the detailed description of the interface of this module:

4.1 API Definition

4.1.1 LINK_mdBindDev

Allocates resources needed for initialization of this device.

Syntax

```
Int LINK_mdBindDev (Ptr *devp, Int devid, Ptr devParams);
```

Arguments

OUT	Ptr	devp
	Device structure handle.	
IN	Int	devid
	Device Identifier.	
IN	Ptr	devParams
	Device parameters	

Return Values

IOM_EINUSE	device already in use.
IOM_EBADIO	General failure during initialization.
IOM_COMPLETED	Successful initialization.
IOM_EBADARGS	Invalid argument passed.

Comments

This function returns IOM_EINUSE error code in case device is already in use. Otherwise it will mark the device to be in use. It will return a pointer to structure of type `LINK_DevObject`. This structure can be allocated dynamically or statically based upon what type of SWI support we want to provide.

Constraints

None.

See Also

None.

4.1.2 LINK_ mdCreateChan

Creates a new channel on given device.

Syntax

```
Int LINK_mdCreateChan (Ptr *chanp, Ptr devp, String name, Int mode, Ptr
chanParams,  IOM_TiomCallback cbFxn, Ptr cbArg) ;
```

Arguments

OUT	Ptr	chanp
	Channel handle to be created.	
IN	devp	devp
	Device on which to create the channel.	
IN	String	name
	Channel number as character string.	
IN	Int	mode
	Mode of the channel	
IN	Ptr	chanParams
	Channel parameters	
IN	IOM_TiomCallback	cbFxn
	IOM callback function	
IN	Ptr	cbArg
	Argument to IOM callback function	

Return Values

IOM_EBADARGS	Invalid/Unsupported mode or channel id passed
IOM_EINUSE	Specified channel is already in use.
IOM_COMPLETED	Function successfully completed.
IOM_EBADIO	General failure during operation.

Comments

This function creates a channel to GPP. We can create either input or output channel.

Constraints

None.

See Also

None.

4.1.3 LINK_ mdDeleteChan

Deletes specified channel.

Syntax

```
Void LINK_mdDeleteChan (Ptr chanp);
```

Arguments

`OUT`	`Ptr`	`chanp`
	Channel to be deleted.	

Return Values

`IOM_COMPLETED`	Function successfully completed.
`IOM_EBADARGS`	Invalid argument passed.
`IOM_EINUSE`	Device already in use.
`IOM_EBADIO`	General failure during operation.

Comments

None.

Constraints

None.

See Also

None.

4.1.4 LINK_mdSubmitChan

Submits a command or IO request on a channel.

Syntax

```
Int LINK_mdSubmitChan (Ptr chanp, IOM_Packet * packet);
```

Arguments

`OUT`	`Ptr`	`chanp`
	Channel handle to be created.	
`IN`	`IOM_Packet *`	`packet`
	IO request packet.	

Return Values

`IOM_ENOTIMPL`	IOM command specified in packet is not implemented.
`IOM_COMPLETED`	function completed successfully
`IOM_PENDING`	IO request has been queued for future excecution.

Comments

This function queues the IOM packet request for commands `IOM_READ` and `IOM_WRITE`. It discards all the pending requests in case of `IOM_ABORT` and `IOM_FLUSH`.

Constraints

None.

See Also

None.

4.1.5 LINK_init

Initializes LINK data structures before bind function

Syntax

```
Void LINK_init();
```

Arguments

None.

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

5 Shared Memory Protocol Control Module

5.1 Type Definitions

5.1.1 SHM_FieldId

Defines Id values for control fields.

Definition

```
typedef enum {
    SHM_handshakeGPP = 0,
    SHM_handshakeDSP,
    SHM_dspFreeMask,
    SHM_gppFreeMask,
    SHM_inputFull,
    SHM_inputId,
    SHM_inputSize,
    SHM_outputFull,
    SHM_outputId,
    SHM_outputSize
} SHM_FieldId;
```

Fields

SHM_handshakeGPP	GPP's handshake field.
SHM_handshakeDSP	DSP's handshake field.
SHM_dspFreeMask	k'th bit set means DSP is ready to receive on k'th channel.
SHM_gppFreeMask	k'th bit set means GPP is ready to receive on k'th channel.
SHM_inputFull	There is something in the input buffer for DSP to read.
SHM_inputId	Channel number for which input buffer is available.
SHM_inputSize	Size of the buffer.
SHM_outputFull	Output is full and GPP should read it.
SHM_outputId	Channel number for which output is done
SHM_outputSize	Size of output buffer.

5.1.2 SHM_Control

Control structure of shared memory.

Definition

```
typedef struct SHM_Control {
    volatile Uns  handshakeGPP;
    volatile Uns  handshakeDSP;
    volatile Uns  dspFreeMask;
    volatile Uns  gppFreeMask;
    volatile Uns  outputFull;
    volatile Uns  outputId;
```

```
    volatile Uns  outputSize;
    volatile Uns  inputFull;
    volatile Uns  inputId;
    volatile Uns  inputSize;
    volatile Uns  argv;
    volatile Uns  resv;
} SHM_Control;
```

Fields

`handshakeGPP`	Handshake field updated by GPP. DSP waits on this field during initialization to synchronize with GPP.
`handshakeDSP`	Handshake field updated by DSP. DSP writes this field after reading 'handshakeGPP' field to unblock GPP.
`dspFreeMask`	k'th bit set means DSP is ready to receive on k'th channel.
`gppFreeMask`	k'th bit set means GPP is ready to receive on k'th channel.
`inputFul`	There is something in the input buffer for DSP to read.
`inputId`	Channel number for which input buffer is availble.
`inputSize`	Size of the buffer.
`outputFull`	Output is full and GPP should read it.
`outputId`	Channel number for which output is done
`outputSize`	Size of output buffer.
`argv`	Reserved
`resv`	Reserved

Comments

This structure defines control structure of the shared memory.

5.2 API Definition

5.2.1 SHM_init

Initializes shared memory. It also initializes the hardware abstraction module.

Syntax

```
Int SHM_init (Ptr config);
```

Arguments

`IN`	`Ptr`	`config`
	Configuration parameter for shared memory driver. Specify NULL for default configuration.	

Return Values

`SYS_OK`	Initialization successful.
`SYS_BADIO`	Initialization failed.

Comments

None.

Constraints

None.

See Also

None.

5.2.2 SHM_readCtlParam

Reads a control parameter from the shared memory control structure.

Syntax

```
Uns SHM_readCtlParam (Uns param);
```

Arguments

`IN`	`Uns`	`param`
	Parameter Id.	

Return Values

`0-max integer value`	Uns value read from the address specified as parameter.

Comments

None.

Constraints

None.

See Also

None.

5.2.3 SHM_writeCtlParam

Write shared memory control structure parameter.

Syntax

```
Void SHM_writeCtlParam (Uns param, Uns Value);
```

Arguments

IN	Uns	param
	Parameter Id.	
IN	Uns	value
	Parameter value to be written.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

5.2.4 SHM_writeOutputBuffer

Writes shared memory output buffer.

Syntax

```
Void SHM_writeOutputBuffer(Ptr buffer, Uns size) ;
```

Arguments

IN	Ptr	buffer
	Buffer to be written to shared memory.	
IN	Uns	size
	Size of the buffer.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

5.2.5 SHM_readInputBuffer

Read input data buffer from shared memory.

Syntax

```
Void SHM_readInputBuffer (Ptr buffer, Uns size) ;
```

Arguments

IN	Ptr	buffer
	Buffer where to put the read data.	
IN	Uns	size
	Maximum size of the buffer.	

Return Values

0-max integer value	Number of MAUs actually read

Comments

None.

Constraints

None.

See Also

None.

5.2.6 SHM_disableGPPInt

Disables the GPP interrupt.

Syntax

```
Uns SHM_disableGPPInt();
```

Arguments

None.

Return Values

0-max integer value	Key to enable interrupt with a subsequent call to SHM_enableGPPInt

Comments

None.

Constraints

None.

See Also

None.

5.2.7 SHM_enableGPPInterrupt

Enables GPP interrupt.

Syntax

```
Void SHM_enableGPPInt (key);
```

Arguments

IN	Uns	key
	Key to enable interrupt.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

5.2.8 SHM_registerGPPISR

Register ISR for GPP interrupt.

Syntax

```
Void SHM_registerGPPISR (Ptr func, Ptr arg);
```

Arguments

IN	Ptr	func
	Function to register.	
IN	Ptr	arg
	Argument to function.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

`SHM_init`

5.2.9 SHM_sendInt

Send interrupt to GPP.

Syntax

```
Void SHM_sendInt (Ptr arg);
```

Arguments

IN	Ptr	arg
	Argument containing interrupt specific information.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

5.2.10 SHM_getMaxBufferSize

Function to get maximum size of shared memory buffer.

Syntax

```
Uns SHM_getMaxBufferSize ();
```

Arguments

None.

Return Values

None.

Comments

None.

Constraints

None.

See Also

`SHM_init`

6 Hardware Abstraction Layer (HAL)

This module abstracts the hardware specific things of shared memory access and interrupts handling. This section explains implementation of this module interface for OMAP 5910/ OMAP 5912 device.

6.1 API Definition

6.1.1 HAL_init

Initialization function of HAL module. It initializes EMIF and interrupt hardware.

Syntax

```
Int HAL_init (Ptr config) ;
```

Arguments

IN	Ptr	config
	Configuration for initialization. Specify NULL for default configuration.	

Return Values

SYS_OK	Operation successful.
SYS_BADIO	Operation failed.

Comments

None.

Constraints

None.

See Also

None.

6.1.2 HAL_memWrite

Writes to the memory from given buffer.

Syntax

```
Void HAL_memWrite(Ptr fromBuf,Ptr toBuffer,maus);
```

Arguments

IN	Ptr	fromBuf
	From where to read data to be written.	
OUT	Ptr	toBuffer
	Address where data is to be written. This must be in EMIF memory address space.	
IN	Ptr	maus

Number of MAUS to be written.

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

6.1.3 HAL_memRead

Reads data from `fromBuf` to `toBuf`, maus are number of 16 bit words to be copied.

Syntax

```
Void HAL_memRead(Ptr fromBuf,Ptr toBuffer,Ptr maus) ;
```

Arguments

IN	Ptr	fromBuf
	Buffer from where to read data to write. This should be in EMIF memory address space.	
OUT	Ptr	toBuffer
	Address where data is to be written.	
IN	Ptr	maus
	Number of MAUS to be read.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

6.1.4 HAL_disableGPPInt

Disables the GPP interrupt.

Syntax

```
Uns HAL_disableGPPInterrupt();
```

Arguments

None.

Return Values

`0-max integer value`	key to enable GPP interrupt again.

Comments

None.

Constraints

None.

See Also

None.

6.1.5 HAL_enableGPPInt

Enables the GPP interrupt.

Syntax

```
Void HAL_enableGPPInterrupt (Uns key);
```

Arguments

`IN`	`Uns`	`key`
	Key to enable interrupt.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

6.1.6 HAL_registerGPPISR

Registers ISR for GPP interrupt.

Syntax

```
Void HAL_registerGPPInterruptISR (Ptr func, Ptr arg);
```

Arguments

`IN`	`Ptr`	`func`
	Function to register.	
`IN`	`Ptr`	`arg`
	Argument to function.	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

6.1.7 HAL_sendInt

Interrupts the GPP.

Syntax

```
Void HAL_sendInt (Ptr arg);
```

Arguments

`IN`	`Ptr`	`arg`
	Argument containing interrupt specific information	

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

7 Shared Memory Protocol

This section describes the protocol used to exchange data though shared memory. We will explain the protocol functioning from the DSP side, although GPP side is also similar. Figure 3 shows shared memory layout (for DSP/BIOS LINK). Add handshake field and reserved fields in diagram.

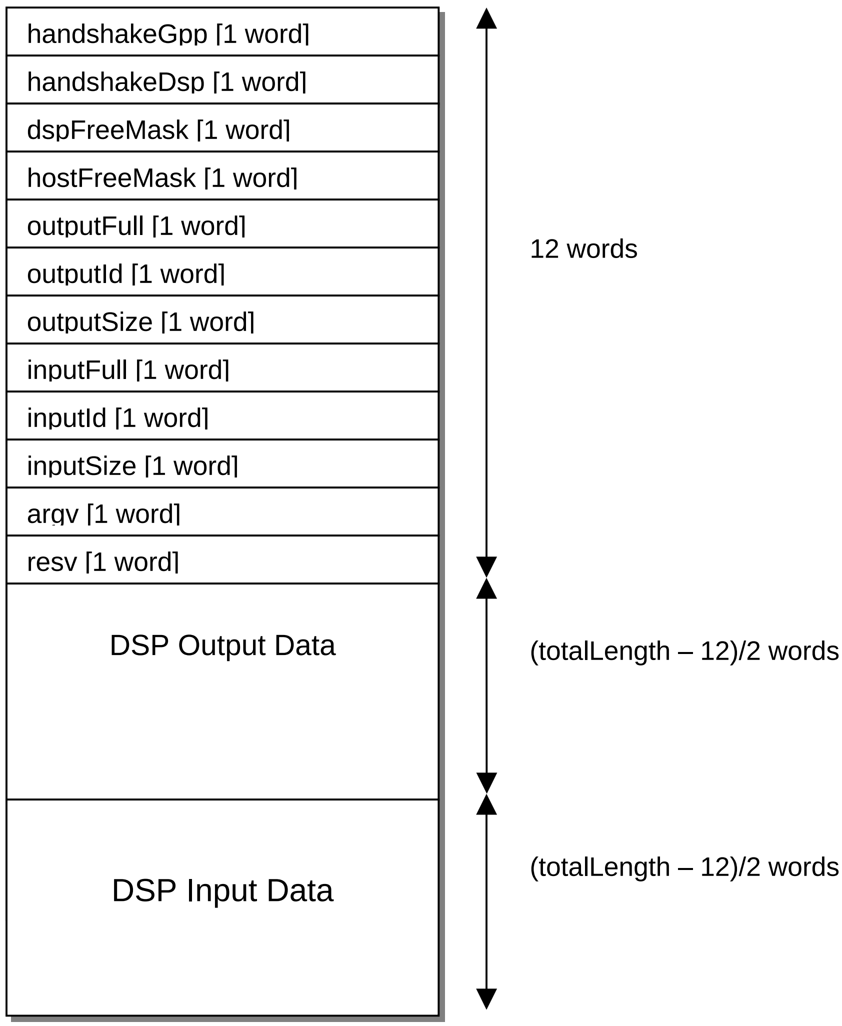

Figure 3. Shared memory layout (totalLength in figure is total length of shared memory)

The table given below explains the explanation of various parameters of shared memory.

Name	Size in Words	DSP Access	Host Access	Description
dspFreeMask	1	R/W	R	Bit N set: DSP has requested data on channel N
hostFreeMask	1	R	R/W	Bit N set: Host has requested data on channel N
outputFull	1	R/W	R/W	DSP output buffer has unread

				data
outputId	1	R/W	R/W	Channel id for DSP output buffer
outputSize	1	R/W	R/W	Size of data block in DSP output buffer
inputFull	1	R/W	R/W	DSP input buffer has unread data
inputId	1	R/W	R/W	Channel id for DSP input buffer
inputSize	1	R/W	R/W	Size of data block in DSP input buffer
DSP Output Data	N *)	R/W	R	DSP output buffer
DSP Input Data	N *)	R	R/W	DSP input buffer

Note that the size of 1 word on DSP in OMAP is 16 bits. So maximum number of channels is limited to 16.

Following section describes the typical operations that are done on shared memory when sending/receiving the buffer by DSP.

Scenario-1: DSP application demands data on a channel k**:**

1. Set bit k of dspFreeMask.
2. Send interrupt to GPP/GPP.

Scenario-2: DSP receives interrupt from GPP/GPP

If inputFull = 1 do following

If buffer (IOM packet) is available for channel inputId

a. Copy the data to buffer of channel inputId. Read the length of buffer form inputSize.
b. Set inputFull to 0
c. Set bit inputId of dspFreeMask to zero if no more data is required for this channel.

Else if buffer (IOM packet) is not available or channel k is not in ready state

a. Set inputFull to 0
b. Set bit inputId of dspFreeMask to zero if no more data is required for this channel.

It then follows the procedure of Scenario-3 for each channel.

Scenario-3: DSP application sends data to GPP/GPP

Check the following three conditions:

1. GPP/GPP is ready to receive the data buffer on channel outputId. We check this by checking the appropriate bit of gppFreeMask.
2. outputFull is not 1.
3. Data is available for transfer to GPP/GPP on the channel.

If above conditions get satisfied we do following

1. Copy the data to output data buffer in shared memory.

2. Set outputSize to appropriate size.
3. Set outputId to appropriate channel number.
4. Set outputFull to 1.
5. Send interrupt to GPP/GPP.