

DSP/BIOS™ LINK

MMU Dynamic entry support (OMAP)

LNK 181 DES

Version 1.60

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

Copyright ©. 2003, Texas Instruments Incorporated

This page has been intentionally left blank.

TABLE OF CONTENTS

1	Introduction	7
	1.1 Purpose & Scope	7
	1.2 Terms & Abbreviations	7
	1.3 References	7
	1.4 Overview	7
2	Requirements	8
3	Assumptions	8
4	Constraints	8
5	High Level Design	8
	5.1 Static entries	8
	5.2 Dynamic entries	9
6	Design	9
	6.1 Dynamic addition of entries in the TLB	10
	6.2 Deletion of entries	10
7	API Usage	12
8	Constants & Enumerations	13
9	API definition	14
	9.1 Internal functions	14
	9.2 Exported API	16

TABLEOFFIGURES

Error!Notableoffiguresentriesfound.

1 Introduction

1.1 Purpose&Scope

This document describes the design of MMU entries configuration dynamically for DSP/BIOS™ LINK. The document is targeted at the development team of DSP/BIOS™ LINK.

1.2 Terms&Abbreviations

DSPLINK	DSP/BIOS™ LINK
MMU	Memory Management Unit
FLT	First Level Table
SLT	Second Level Table
TLB	Translation look-aside Buffer
TWL	Table walking logic
Ⓢ	This bullet indicates important information. Please read such text carefully.
□	This bullet indicates additional information.

<

Ⓢ *This is important information.*

□ *This is additional information.*

>

1.3 References

1.	TRM	OMAP3430 Multimedia Device Silicon Revision 1.0
----	-----	---

1.4 Overview

MMU instances handle translation from virtual into physical addresses. Virtual addresses are issued by the DSP subsystems to the MMU, which converts them into physical addresses. These physical addresses correspond to actual memory resource. MMU instances can be used dynamically or statically, in other words, MMU can be managed by the MPU software or configured directly.

An MMU can be configured dynamically when a software subroutine writes translation tables into the appropriate physical memory space. Translation tables are most likely stored in external SDRAM.

This document provides a detailed description of translation table and the MMU dynamic configuration driver design.

2 Requirements

SR132.27: Support for dynamically creating DSP MMU entries on OMAP.

On OMAP devices (OMAP3530 and OMAP2530), it must be possible to use the PROC_control API to dynamically add/delete DSP MMU entries. It must also be possible to dynamically map/unmap the DSP regions into user-space.

3 Assumptions

The MMU design makes the following assumptions:

- The hardware provides a shared memory area, to which both the GPP and the DSP have access.
- The following page sizes are supported:
 - Super section: 16M bytes
 - Section : 1M bytes
 - Large page : 64K bytes
 - Small page : 4K bytes

4 Constraints

For dynamic addition of entries, any memory overlap cases will treat as error case.

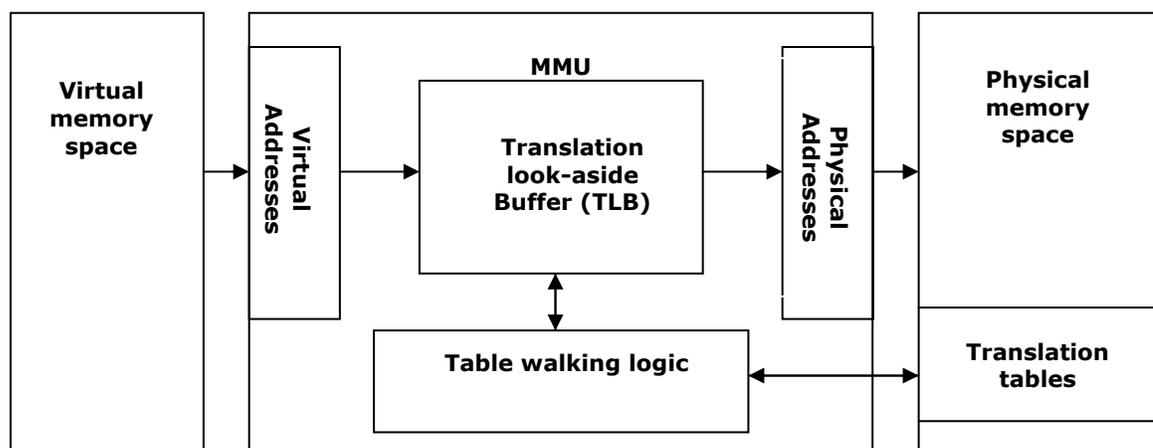
For optimization of TLB entries please make sure

- Requested entries should be continuous.
- Requested entries should be aligned with 16M or 1M or 64K or 4K.

5 HighLevelDesign

The MMU handle the translation from virtual into physical addresses. The requestor DSP issues virtual addresses to the respective MMU. The MMU translates these virtual addresses into physical addresses to access the actual resource (memory).

MMU instances (MMU entries) can be used dynamically or statically, in other words, MMU can be managed by the GPP software or configured directly.



5.1 Staticentries

This method avoids the need to write Translation tables in memory and is commonly used for relatively small address spaces. When an MMU is configured statically,

translation tables are not used, and TLB entries are written directly by the programmer. It ensures that the translation of time-critical data accesses execute as fast as possible with entries already present in the TLB. These entries must be locked to prevent them from being overwritten. N entries fully associative translation look-aside buffer (TLB) with $N = 32$ for the DSP. The entries must be locked to prevent them from being overwritten.

5.2 Dynamic entries

An MMU is configured dynamically when a software subroutine writes translation tables into the appropriate physical memory space. Whenever an address translation is requested (that is, for every access with the MMU enabled), the MMU first checks whether the translation is already contained in the TLB, which acts like a cache storing recent translations. If the requested translation is not in the TLB, the table-walking logic retrieves this translation from the translation table(s), and then updates the TLB.

6 Design

In initial set, some of the DSPLink memory region will be written in the TLB and TLB will be set to protect these entries, so that these entries are not over written. This is done, so that GPP side and DSP side has permanent access to the shared memory region and code/data region as well. Apart from these, GPP side can also dynamically create/write entries in the TLB, upon user requests.

Whenever a user wants to map some memory address to the DSP address space, GPP side logic checks for overlaps:

The following are set of overlap cases:

1. Any existing entry's start address is covered by given entry's start address and end address.
(Given Start Address \leq Entry's start address \leq Given End address)
2. Any existing entry's end address is covered by given entry's start address and end address.
(Given Start Address \leq Entry's end address \leq Given End address)
3. Given entry's start address is covered by any existing entry's start address and end address.
(Entry's Start Address \leq given start address \leq Entry End address)
4. Given entry's end address is covered by any existing entry's start address and end address.
(Entry's Start Address \leq given end address \leq Entry End address)

If all of the above conditions are false, then entry is updated in the TLB.

On any overlap case, DSPLink GPP side will return error.

To check the above conditions, a linked-list is maintained in GPP-side so that GPP does not read the TLB area directly, as the direct access to TLB will require reading TLB register in the hardware documented way. This linked-list is a replica of TLB, all entries which exists in the TLB also exists in this linked-list. Also any update made to TLB is also updated in the linked-list. This linked-list acts like a cache for TLB in GPP.

For deleting entries, GPP side simply checks whether entry existing the linked-list, if it exists then it check if the entry is protected or not. If the entry is not protected then it is deleted from the TLB and linked-list.

6.1 Dynamic addition of entries in the TLB

TLB entries consist of two parts:

The CAM part contains the virtual address tag used to determine if a virtual address translation is in the TLB. The TLB acts like a fully associative cache addressed by the virtual address tag. The CAM part also contains the section/page size, as well as the preserved and the valid parameters.

The RAM part contains the address translation that belongs to the virtual address tag as well as the endianness, element size, and mixed parameters.

Procedure to create TLB entries are as follows:

1. Reset the first bit of MMU_CNTL register. It disables the MMU.
2. Load the Virtual Address (VATAG), the preserved (P=1) and valid (V=1) bits and the page size (small or large page, section, super section) into MMU_CAM register.
3. Load the Physical Address (PHYSICALADDRESS), the endianness (ENDIANNESS=0), element size (ELEMENTSIZE) and mixed page attributes bits (MIXED) into MMU_RAM register.
4. Specify the TLB entry you want to write by setting the MMU_LOCK.CURRENTVICTIM pointer. Start with TLB Entry 0 and increment this pointer for each subsequent entry you want to write.
5. Load the specified entry in the TLB by setting MMU_LD_TLB.LDTLBITEM=1.
6. Set the first bit of MMU_CNTL register. It enables the MMU.

After writing the CAM and RAM registers, set the entry into the TLB.

The first n TLB entries (with n < total number of TLB entries) can be protected from being overwritten with new translations. This is useful to ensure that certain commonly used or time critical translations are always in the TLB and do not require retrieval via the table walking process.

To protect the first n TLB entries, set the MMU1.MMU_LOCK [12:10] BASEVALUE field for the IVA2.2 MMU to n.

6.2 Deletion of entries

Two mechanisms exist to delete TLB entries. All unpreserved TLB entries, i.e., TLB entries that were written with the preserved bit set to zero, can be deleted by invoking a TLB flush. Such a TLB flush is invoked by setting the MMUn.MMU_GFLUSH[0] GLOBALFLUSH bit.

Individual TLB entries can be flushed, regardless of the preserved bit setting, by specifying its virtual address in the MMUn.MMU_CAM register and setting the MMUn.MMU_FLUSH_ENTRY[0] FLUSHENTRY bit.

APIs are as follows:

OMAP3530_halMmuAddEntry	This API would add the entry at run time.
OMAP3530_halSearchMmuEntry	This API would match an event listener with a list

	element. And also verify the boundary conditions.
OMAP3530_halMmuDeleteEntry	This API would delete the entry from TLB and list.
OMAP3530_halCheckMmuEntry	This API would match an event listener with a list element.
PROC_control	This API is user interface to map, unmap, add and delete the DSP memory.

7 APIUsage

If DSP access an area whose mapping is not present in the TLB, then DSP's MMU generates an interrupt of PAGE fault type. This interrupt is generated on ARM (HOST). This happens only in case when table walking logic is disabled. Otherwise, if it is enabled, MMU checks with tables present in the memory.

To map some area into the DSP's address space, user can call PROC_control API with respective commands from GPI side. Below shows a usage scenario of mapping an area into DSP's address space.

Add the MMU entry:

```
ProcMemMapInfo    mapInfo    ;

mapInfo.dspAddr  = DSP_ADDR1 ;
mapInfo.size     = 0x80000   ;

status = PROC_control (ID_PROCESSOR,
                      PROC_CTRL_CMD_MMU_ADD_ENTRY,
                      &mapInfo) ;
```

Here PROC_CTRL_CMD_MMU_ADD_ENTRY is an enumerated type, directing GPP side logic to add user given entries to the DSP's TLB. Now, User may want to write some information on the area, to be given to DSP. For this user has to map the area into GPP user/kernel address space. For this below example is useful:

```
ProcMemMapInfo    mapInfo    ;

mapInfo.dspAddr  = DSP_ADDR1 ;
mapInfo.size     = 0x80000   ;

status = PROC_control (ID_PROCESSOR,
                      PROC_CTRL_CMD_MAP_DSPMEM,
                      &mapInfo) ;
```

Once the user has done with his/her protocol, he/she may want to unmap the area from GPP and DSP address space as well. Below code how to achieve this:

Delete the MMU entry:

```
ProcMemMapInfo    unMapInfo  ;

unMapInfo.dspAddr = DSP_ADDR1 ;
unMapInfo.size    = 0x80000   ;

status = PROC_control (ID_PROCESSOR,
                      PROC_CTRL_CMD_UNMAP_DSPMEM,
                      &unMapInfo) ;

status = PROC_control (ID_PROCESSOR,
                      PROC_CTRL_CMD_MMU_DEL_ENTRY,
                      &unMapInfo) ;
```

8 Constants & Enumerations

Definition

```
#define PROC_CTRL_CMD_MAP_DSPMEM
    (PROC_CTRL_CMD_GEN_BASE + 0x00000001u)
#define PROC_CTRL_CMD_UNMAP_DSPMEM
    (PROC_CTRL_CMD_GEN_BASE + 0x00000002u)
#define PROC_CTRL_CMD_MMU_ADD_ENTRY
    (PROC_CTRL_CMD_GEN_BASE + 0x00000003u)
#define PROC_CTRL_CMD_MMU_DEL_ENTRY
    (PROC_CTRL_CMD_GEN_BASE + 0x00000004u)
```

Comments

PROC_CTRL_CMD_MMU_ADD_ENTRY	This command used to add the entry at run time.
PROC_CTRL_CMD_MAP_DSPMEM	This command used to map some area into the DSP's address space.
PROC_CTRL_CMD_UNMAP_DSPMEM	This command used to unmap the area from GPP and DSP's address space.
PROC_CTRL_CMD_MMU_DEL_ENTRY	This command used to delete the entry from TLB and list.

9 APIdefinition

9.1 Internalfunctions

9.1.1 OMAP3530_halMmuAddEntry

This function adds the run time requested entry. Also calls the OMAP3530_OMAP3530_halSearchMmuEntry function from the attached interface.

Syntax

```
NORMAL_API
DSP_STATUS
OMAP3530_halMmuAddEntry (IN Uint32 tlbIndex,
                          IN Bool   type,
                          IN Uint32 physAddr,
                          IN Uint32 dspVirtAddr,
                          IN Uint32 size)
```

Arguments

IN	Uint32	tlbIndex
		TLB entry number.
IN	Bool	type
		To identify the type of entry (static or run time entry)
IN	Uint32	physAddr
		Physical address.
IN	Bool	dspVirtAddr
		DSP virtual address.
IN	Uint32	size
		Size of requested entry.

ReturnValue

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_SEXISTS	Entry exists.

Comments

None.

Constraints

None.

9.1.2 OMAP3530_halSearchMmuEntry

This function search the run time requested entry.

Syntax

```

NORMAL_API
DSP_STATUS
OMAP3530_halSearchMmuEntry (IN ListElement * elem, Pvoid data, Bool
type)

```

Arguments

IN	ListElement	elem
	List of TLB entries.	
IN	Pvoid	data
	Requested entry.	

ReturnValue

DSP_SOK	Operation successfully completed.
DSP_SEXISTS	Entry exists.
DSP_ERANGE	

Comments

None.

Constraints

None.

9.1.3 OMAP3530_halMmuDeleteEntry

This function deletes the run time requested entry. Also calls the OMAP3530_halSearchMmuEntry function.

Syntax

```

EXPORT_API
DSP_STATUS
OMAP3530_halMmuDeleteEntry (IN Pvoid halObj, IN ProcMemMapInfo *
mmuInfo)

```

Arguments

IN	Pvoid	halObj
	Hardware Abstraction object.	
IN	ProcMemMapInfo*	mmuInfo
	Control information for mapping DSP memory region in GPP's address space	

ReturnValue

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.

DSP_ECONFIG

Operation failed due to a config error.

Comments

None.

Constraints

None.

9.1.4 OMAP3530_halCheckMmuEntry

This function check the run time requested entry to be deleting.

Syntax

```
NORMAL_API
Bool
OMAP3530_halCheckMmuEntry (IN ListElement * elem, Pvoid data)
```

Arguments

IN	ListElement	elem
	List of TLB entries.	
IN	Pvoid	data
	Requested entry.	

ReturnValue

DSP_SOK	Operation successfully completed.
DSP_ERANGE	Entry not exists

Comments

None.

Constraints

None.

9.2 ExportedAPI

9.2.1 PROC_control

Provides a hook to perform device dependent control operations on the DSP.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_control (IN ProcessorId procId,
              IN Int32 cmd,
              OPT Pvoid arg)
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	

