

# Programming the DSP MMU in the OMAP5910 Device

OMAP)))

Matthias Kassner

#### ABSTRACT

The OMAP5910 device from Texas Instruments (TI) has a new dual-core architecture that is optimized for multimedia applications in a low-power environment. It couples two processors—a TI-enhanced TI925T general-purpose processor and an ultralow-power TMS320C55x<sup>™</sup> (C55x<sup>™</sup>) digital signal processor (DSP)—with a rich set of peripherals and powerful interfaces to achieve optimal performance.

Within the OMAP5910 device, the C55x DSP can use its memory management unit (MMU) to access the DSP external memory space. This application note discusses the basic DSP MMU architecture and the benefits of using an MMU. It also explains how the DSP MMU is programmed and provides programming examples.

#### Contents

1	OM	AP5910 Hardware Architecture	. 3
	1.1	High-Level Overview	. 3
	1.2	DSP Megacell Architecture	. 4
		1.2.1 DSP Memory Interface	. 5
2	MM	U Functional Description	. 6
	2.1	MMU Benefits	. 6
	2.2		
	2.3		
	2.4		
		2.4.1 Translation Table Hierarchy	
		2.4.2 First-level Translation Table	
		2.4.3 First-level Descriptor Format	
		2.4.4 Two-level Translation	
	2.5	Translation Look-aside Buffer	
		2.5.1 TLB Entry Format	
3		gramming the DSP MMU	
	3.1	Writing, Reading, and Deleting TLB Entries	
		3.1.1 Writing TLB Entries	
		3.1.2 Protecting TLB Entries	
		3.1.3 Reading TLB Entries	
		3.1.4 Deleting TLB Entries	
		3.1.5 Enabling the MMU	18
	3.2	MMU Error Handling	18
4	MM	U Register Manual	20

OMAP, TMS320C55x, C55x, and Code Composer Studio are trademarks of Texas Instruments Incorporated. OMAP1510, OMAP1610, OMAP1710, OMAP5910, and OMAP5912 are members of Texas Instruments OMAP™ family of products.

All other trademarks are the property of their respective owners.

SWPA038



	4.1 MMU Register Set	
	4.2 MMU Register Descriptions	
5	Conclusion	
6	References	
	A.1 Example Projects	
	A.2 Configuring the DSP MMU by Programming the TLB	
	A.2.1 Verification Procedure	
	A.3 Configuring the DSP MMU by Building a Tanslation Table	
	A.3.1 Verification Procedure	

## Figures

Figure 1.	OMAP5910 System Building Blocks	. 4
Figure 2.	C55x DSP Megacell	
Figure 3.	C55x Megacell Memory Interface	
Figure 4.	MMU Address Translation	
Figure 5.	MMU Usage Examples	
Figure 6.	MMU Architecture	
Figure 7.	Translation Process	. 8
Figure 8.	Translation Hierarchy	
Figure 9.	First-level Descriptor Address Calculation	10
Figure 10.	Detailed First-level Descriptor Address Calculation	10
Figure 11.	Section Translation Summary	
Figure 12.	Two-level Translation	
Figure 13.	Large- and Small-Page Translation Summary	14
Figure 14.	Fine-Page Translation Summary	
Figure 15.	TLB Entry Lock Mechanism	15
Figure 16.	TLB Entry Structure	16
Figure 17.	MMU Translation Process and Fault Types	19

## Tables

Table 1.	First-level Descriptor Format	. 11
Table 2.	Second-level Descriptor Format	
Table 3.	MMU Programming Register Summary	. 20
Table 4.	Prefetch Register (PREFETCH_REG)	. 21
Table 5.	Prefetch Status Register (WALKING_ST_REG)	. 21
Table 6.	Control Register (CNTL_REG)	. 22
Table 7.	Fault Address Register MSB (FAULT_AD_H_REG)	. 22
Table 8.	Fault Address Register LSB (FAULT_AD_L_REG)	. 22
Table 9.	Fault Status Register (F_ST_REG)	. 23
Table 10.	Interrupt Acknowledge Register (IT_ACK_REG)	. 23
Table 11.	Translation Table Base Register (MSB) (TTB_MSB_REG)	. 23
Table 12.	Translation Table Base Register (LSB) (TTB_LSB_REG)	
Table 13.	Lock/Protect Entry Register (LOCK_REG)	. 24
Table 14.	Read/Write TLB Entry Register (LD_TLB_REG)	. 24
Table 15.	CAM Entry Register (MSB) (CAM_H_REG)	. 25
Table 16.	CAM Entry Register (LSB) (CAM_L_REG)	. 25
Table 17.	RAM Entry Register (MSB) (RAM_H_REG)	. 25

Table 18.	Lock/Protect Entry Register (LSB) (RAM_L_REG)	
Table 19.	Global Flush Register (GFLUSH REG)	
Table 20.	Flush Entry Register (FLUSH_ENTRY_REG)	
Table 21.	CAM Entry Read Register (MSB) (READ_CAM_H_REG)	
Table 22.	CAM Entry Read Register (LSB) (READ_CAM_L_REG)	
Table 23.	RAM Entry Read Register (MSB) (READ_RAM_H_REG)	
Table 24.	Lock/Protect Entry Register (LSB) (READ_RAM_L_REG)	
Table A-1.	TLB Entry Properties	29
Table A-2.	First-level Translation Table Entry Properties	

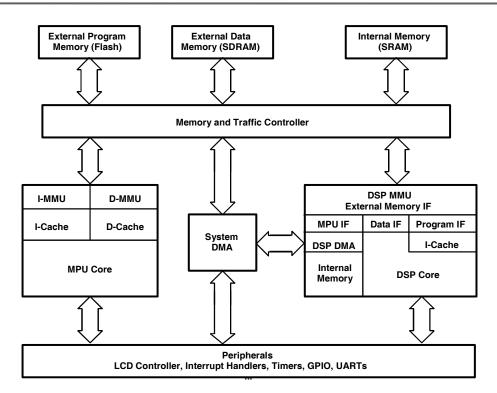
## 1 OMAP5910 Hardware Architecture

#### 1.1 High-Level Overview

The OMAP5910 architecture consists of several building blocks. The TI-enhanced TI925T microprocessor unit (MPU) is the master of the system and provides control and initialization for all components within OMAP5910. The C55x DSP acts as a programmable accelerator to the MPU for efficient acceleration of real-time algorithms.

A dedicated direct memory access (DMA) engine transfers data between various sources and destinations. The memory and traffic controller handles all accesses to shared internal and external memories. A rich set of peripherals is provided to communicate with external devices.

Figure 1 gives a high-level view of the OMAP5910 architecture. For the sake of simplicity, this figure contains only the most important functional units. Note that the MPU, the system DMA, and the DSP can access the external memory and the internal static random access memory (SRAM) using the traffic controller.



#### Figure 1. OMAP5910 System Building Blocks

For more details about the OMAP5910 architecture, refer to the OMAP5910 Dual-Core Processor Data Manual (SPRS197).

#### 1.2 DSP Megacell Architecture

The DSP megacell consists of the C55x DSP core, the DSP internal memory, the DSP DMA, multimedia hardware accelerators, the instruction cache, and several interfaces to other OMAP<sup>™</sup> units (see Figure 2).

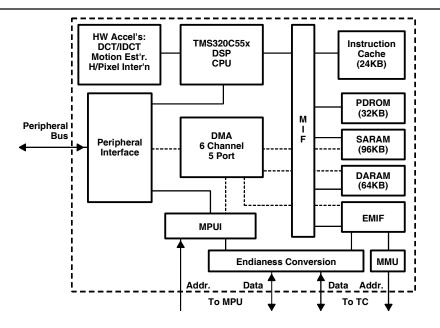


Figure 2. C55x DSP Megacell

#### 1.2.1 DSP Memory Interface

Accesses from DSP internal requestors (DSP core and DSP DMA) to external memory are handled by the DSP external memory interface (EMIF) in conjunction with the DSP MMU. The DSP MMU maps the 24-bit DSP address space into the 32-bit OMAP physical address space, provides fault and permission checking, and performs endianess conversion. It is configured by the MPU.

Figure 3 shows the role of the DSP MMU within the C55x megacell memory interface.

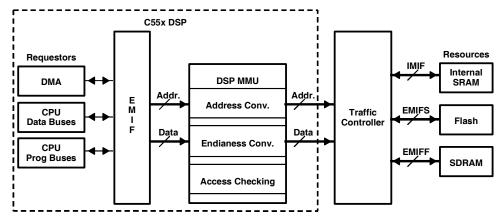


Figure 3. C55x Megacell Memory Interface

## 2 MMU Functional Description

The MMU handles the translation from virtual into physical addresses. Virtual addresses are issued by the DSP to the MMU, which converts them into physical addresses. These physical addresses are used to access the actual resource (memory).

This relationship between virtual addresses, physical addresses, and the MMU is shown in Figure 4.

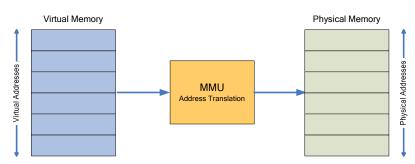


Figure 4. MMU Address Translation

#### 2.1 MMU Benefits

The use of an MMU offers two major benefits:

- Memory defragmentation: Fragmented physical memory can be translated into continuous virtual memory without moving any data.
- Memory protection: Illegal, non-allowed accesses to memory locations can be detected and prevented.

Figure 5 shows two typical MMU use cases.

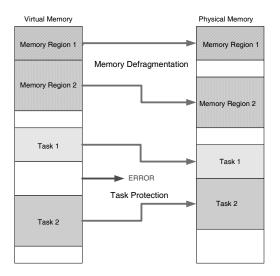


Figure 5. MMU Usage Examples

Figure 5 illustrates the benefits of using an MMU. Notice that memory region 1 and memory region 2 are fragmented in physical memory. Using the MMU, they can be translated so that they appear as one contiguous memory region in the virtual memory space.

On the other hand, task 1 and task 2 are located adjacent in physical memory. In systems without MMU, there is a danger that task 1 will accidentally write into the memory area allocated to task 2, and vice versa. Mapping the tasks to two separate memory regions prevents errors because a region of unmapped memory separates them. Any erroneous access to this region can be easily detected.

### 2.2 MMU Architecture

The MMU translation process is based on a set of translation entries, or *descriptors*. These descriptors are stored in translation tables. One first-level translation table and, optionally, several second-level translation tables exist.

Each translation table entry describes the translation of one contiguous memory page. For a description of the structure of these tables, see Section 2.4.

Two major functional units exist in the MMU to automatically provide address translation based on the table entries:

- The table walker automatically retrieves the correct translation table entry for a requested translation. If two-level translation is used, the table walker also automatically reads the required second-level translation table entry.
- The translation look-aside buffer (TLB) stores recently used translations. The TLB acts like a cache of recently read translation table entries.

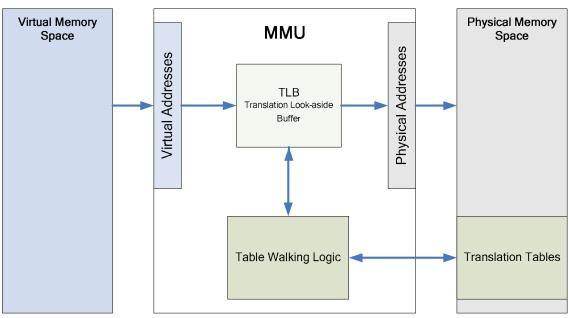


Figure 6 shows the basic MMU architecture.



#### 2.3 MMU Address Translation Process

Whenever an address translation is requested (that is, for every memory access with the MMU enabled), the MMU checks first to see whether the TLB contains the requested translation.

As mentioned previously, the TLB acts like a cache, storing recent translations. It can also be manually programmed to ensure that time-critical data can be translated without delay.

If the requested translation is not in the TLB, the table walking logic retrieves the translation by reading the corresponding translation table entry/entries. After the logic has retrieved the correct translation, it updates the TLB and the address translation is performed. This process is summarized in Figure 7.

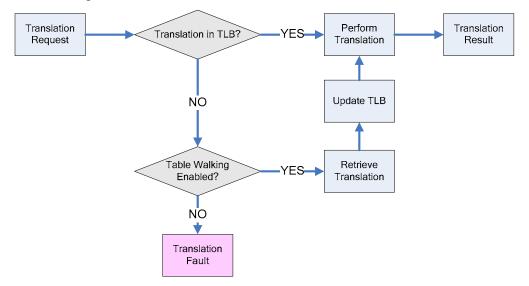


Figure 7. Translation Process

#### 2.4 Translation Table Structure

The translation of virtual to physical addresses is based on entries in translation tables. These entries define the following properties:

- Address translation: Correspondence between virtual and physical addresses
- Access rights: Read and write, read-only, or no-access rights
- Size of the memory region this entry translates

The translation tables are indexed by the virtual address. Each virtual address corresponds to exactly one entry in a translation table. Conversely, each translation table entry describes the translation properties of a range of virtual addresses.

## 2.4.1 Translation Table Hierarchy

One of the most important design parameters in developing a table-based address translation scheme is the memory page size, that is, the size of the memory region each translation table entry describes.

Using large pages results in a smaller translation table. In contrast, using small pages greatly increases the efficiency of dynamic memory allocation and defragmentation. That is why many operating systems (OSs) can operate on memory blocks as small as 4K bytes. This small size, however, also implies more complex (and larger) translation tables.

A quick calculation shows that a translation table using 1K-byte large memory pages requires more than four million entries to span the entire 4G-byte address range. Such a table itself would be 16M bytes in size. Obviously, this is not feasible.

Using larger pages, however, greatly reduces the functionality of the OS memory management. In order to reconcile these two requirements, a two-level hierarchy is usually implemented.

Within this hierarchy, one first-level translation table describes the translation properties based on 1M-byte large memory regions. Each entry in the first-level translation table describes one such region.

Each of the entries in this first-level translation table can specify:

• The translation properties for a memory section:

Memory sections are 1M-byte large and all necessary translation parameters are specified in the first-level translation table entry.

• A pointer to a second-level translation table:

The second-level translation table subdivides the 1M-byte section into smaller regions, or *pages*. These pages can be 64K bytes (large page), 4K bytes (small page), or 1K byte (tiny page) in size. Each of these pages can have its individual translation properties.

This hierarchical approach means that additional translation information for smaller pages must be provided only when such pages are actually used. Figure 8 shows this hierarchy.

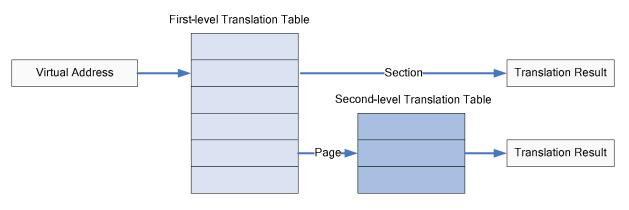


Figure 8. Translation Hierarchy



The structure of the first- and second-level translation tables and their entries is described in more detail in subsection 2.4.2 through subsection 2.4.4.3.

#### 2.4.2 First-level Translation Table

The first-level translation table describes the translation properties based on sections of 1M byte. Sixteen 32-bit entries, or *first-level descriptors*, are required to describe the entire 16M-byte DSP virtual-address space.

The start address of the first-level translation table must be aligned on a 128-byte boundary; that is, at least the last seven address bits of the start address must be zero. This start address is called the *translation table base* and is a configurable parameter.

The table is indexed by the upper 4 bits of the virtual address. This mechanism is shown in Figure 9.

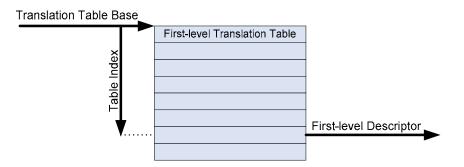
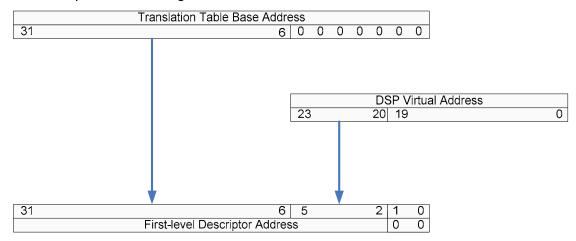


Figure 9. First-level Descriptor Address Calculation

In summary, the translation table base and the translation table index together define the firstlevel descriptor address. Figure 10 outlines the mechanism used to calculate this address.





#### 2.4.3 First-level Descriptor Format

Each first-level descriptor provides either the complete address translation for a 1M-byte section or a pointer to a second-level translation table. Table 1 shows the first-level descriptor format.

First-level Descriptor Format													
31:20	19:12	11:10	9:2	1	0								
Х				0	0	Fault							
Coarse-page table base address			Х	0	1	Coarse page table							
Section base address	X	AP	Х	1	0	Section							
Fine-page table base address		Х		1	1	Fine page table							
AP = Access permissions X = Don't care	0x = No access, 10 = Read-only acce	ess, 11 = F	Full acc	ess (	read	and write)							

#### 2.4.3.1 First-level Page Descriptor Format

If a translation granularity smaller than 1M byte is required, a two-level translation process is used. In this case, the first-level descriptor specifies only the start address of a second-level translation table used to allocate subsections, or pages, within the 1M-byte region the first-level descriptor descriptor describes.

The entries of the specified second-level translation table then define the translation properties of these pages. There are two types of second-level tables: coarse tables and fine tables. See subsection 2.4.4 for information on their differences.

#### 2.4.3.2 First-level Section Descriptor Format

Each section descriptor in the first-level translation table specifies the complete translation properties for a 1M-byte memory section. Two parameters are defined by a section descriptor:

- Address translation: The physical start address of the section
- Access rights: No access, read-only, or read and write access

#### 2.4.3.3 Section Translation Summary

Sections are translated based solely on the information provided by the first-level descriptor. Figure 11 summarizes the address translation process for a section.

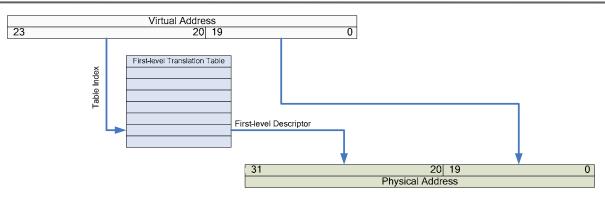


Figure 11. Section Translation Summary

### 2.4.4 Two-level Translation

Two-level translation is used when a translation granularity smaller than 1M byte is required. In that case, the first-level descriptor provides a pointer to the base address of a second-level translation table.

Two types of second-level tables can be used:

• Coarse page tables with 256 entries:

Each entry in a coarse page table describes the translation properties of 4K bytes of memory.

• Fine page tables with 1024 entries:

Each entry in a fine page table describes the translation properties of 1K byte of memory.

The two least significant bits of the first-level descriptor indicate whether the second-level table is a coarse table or a fine table. The second-level table is indexed by bits 19:12 (coarse page table) or bits 19:10 (fine page table) of the virtual address. This indexing mechanism is shown in Figure 12.

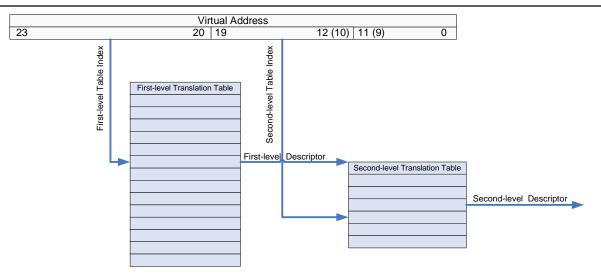


Figure 12. Two-level Translation

Coarse table entries describe the translation of 1M bytes of address space in pages of 4K bytes (small page). They can also define the translation of a 64K-byte large page. In this case, the same descriptor must be repeated 16 times in the coarse page table.

Having large pages provides the advantage that they can be cached using only one entry in the translation look-aside buffer (TLB), even though they require 16 entries in the coarse page table.

Note that the start addresses of both coarse and fine tables must be aligned according to their respective size. This means that a coarse table with 256 entries of 4 bytes each must be aligned on a 1024-byte boundary; that is, the last 10 bits of its start address must be zero.

Similarly, a fine table with 1024 entries of 4 bytes each must be aligned on a 4096-byte boundary, that is, the last 12 bits of its start address must be zero.

#### 2.4.4.1 Second-level Descriptor Format

Second-level descriptors provide all the necessary information for the translation of a large, small, or tiny page. In this, they are similar to first-level section descriptors. Table 2 shows the format of such a second-level descriptor.

	Sec	ond-le	vel De	escrip	tor Forn	nat				
31:16	15:12	11	10	9	8:6	5:4	3:2	1	0	
								0	0	Fault
Large-page base address	Х					AP	Х	0	1	Large page
Small-page base address		Х				AP		1	0	Small page
Tiny-page base address				Х		AP		1	1	Tiny page
AP = Access permissions	0x = No	acces	s, 10=	Read	-only ac	cess, 11 = I	Full acc	ess (	read	and write)
X = Don't care										

Table 2.	Second-level	Descriptor	Format

#### 2.4.4.2 Large- and Small-Page Translation Summary

The translation process for large and small pages using a second-level coarse page table is summarized in Figure 13. The numbers in parentheses refer to small pages.

Note that large descriptors must be repeated 16 times and that large pages must be aligned according to their size, which means that a large page always starts at a coarse table entry that is a multiple of 16.

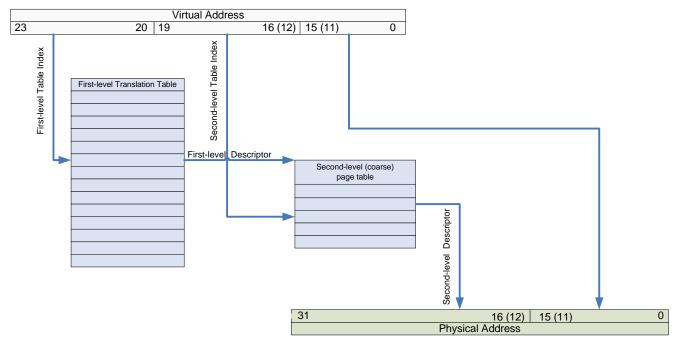


Figure 13. Large- and Small-Page Translation Summary

#### 2.4.4.3 Tiny-Page Translation Summary

The translation of tiny pages is defined by a fine page table. It contains 1024 entries, each describing the translation properties of 1K byte of memory.

**NOTE**: Most operating systems do not support such small allocation sizes; therefore, the support of 1K-byte memory pages will be discontinued in future OMAP generations.

Fine page tables are indexed using bits 19:10 of the virtual address. This is shown in Figure 14 .

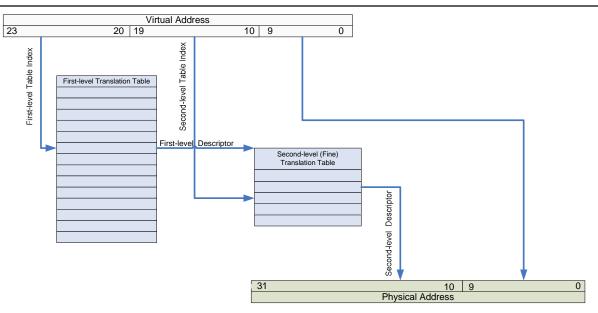


Figure 14. Fine-Page Translation Summary

### 2.5 Translation Look-aside Buffer

Translating virtual to physical addresses is required for each memory access in systems using an MMU. In order to speed up this translation process, a cache mechanism (the TLB) has been introduced to store the results of recent translations.

For every translation request, the MMU internal logic checks first whether this translation already exists in the TLB. If it does, then this translation is used. Otherwise, the translation is retrieved from the translation tables, and the TLB is updated. If the TLB is full, one of the older entries must be replaced. The entry is selected randomly.

The first *n* TLB entries (with n < total number of TLB entries) can be protected, or *locked*, against being overwritten. This is done by setting the TLB base pointer to *n* (see Figure 15). The remaining entries are overwritten, if necessary, on a random basis. The victim pointer indicates the next TLB entry to be written.

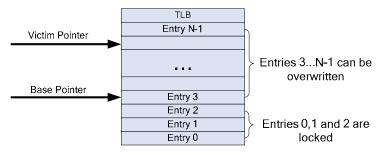


Figure 15. TLB Entry Lock Mechanism



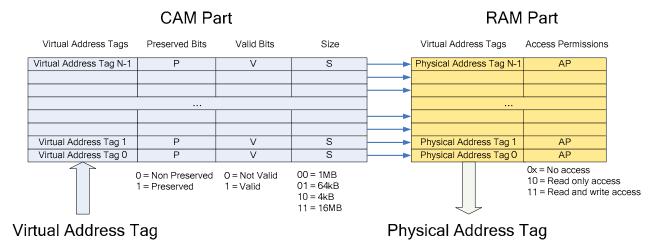
When table walking is enabled, TLB entries are automatically written by the table walking logic. Alternatively, entries can also be manually written by the user. This is done to ensure that the time-critical data accesses execute as fast as possible. Such user-defined entries are typically locked to prevent them from being overwritten subsequently.

#### 2.5.1 TLB Entry Format

TLB entries consist of two parts:

- CAM part: Contains the virtual address tag used to locate the translation in the TLB. The TLB acts as a fully associative cache addressed by the virtual address tag. The CAM part also contains the entry type (section, pointer to coarse table, or pointer to fine table) and the *preserved* and *valid* flags.
- RAM part: Contains the address translation that belongs to the virtual address tag. It also contains the access permissions (no access, read-only access, and read and write access).

The valid parameter specifies whether an entry is valid or not. The preserved parameter determines the behavior of an entry in the event of a TLB flush. If an entry is set as preserved, it is not deleted upon a TLB flush. Such preserved entries must be deleted manually.



The TLB entry structure is shown in Figure 16.



## 3 Programming the DSP MMU

The DSP MMU can be programmed in two ways: by building translation tables or by writing all required entries directly into the TLB. A mix of these two ways is also possible. In that case, translation tables are used but some of the most time-critical translations are preloaded into the TLB.

## 3.1 Writing, Reading, and Deleting TLB Entries

For time-critical program routines, it is often desirable to ensure that the required MMU translations are always present in the TLB. This avoids the performance impact of retrieving these translations via a table walk.

It is therefore possible for the user to write some or all of the required translations manually into the TLB and to protect them against being overwritten. A maximum of 31 of the 32 TLB entries can be user-written and protected. One entry must always remain unprotected for use by the table walking logic.

The TLB entries can be read by the user to determine the currently buffered translations. Unused translations can be deleted.

#### 3.1.1 Writing TLB Entries

TLB entries are written using a set of configuration registers. Four registers (CAM\_H\_REG, CAM\_L\_REG, RAM\_H\_REG, and RAM\_L\_REG) are used to store the CAM and RAM parts of a TLB entry that will be written. The CAM and RAM parts are allocated in two registers (high and low) because they are bigger than 16 bits.

The CAM registers hold the virtual tag, that is, the 14 most significant bits of the virtual address. Additionally, they contain some status bits that define whether the entry is to be preserved upon a TLB flush operation, is valid or contains only random uninitialized content, and the type of memory block (section, large, small, or tiny page) this entry describes.

The RAM registers hold the physical address tag, that is, the 22 most significant bits of the physical address. Additionally, they define the access permissions of the memory region. Both the virtual and physical start addresses of sections and pages must be aligned according to their respective size.

This means, for instance, that for a 1M-byte section only address bits 23:20 (virtual) and 31:20 (physical) are significant.

In order to write a TLB entry, the user must first calculate the correct CAM and RAM parameters and place them into the corresponding CAM and RAM registers. Next, the lock register (LOCK\_REG) is used to select (and, optionally, protect) the entry that will be written. Finally, the TLB entry is written to the specified location by setting the write TLB-entry bit in the LD\_TLB\_REG register.

A summary of all relevant TLB control registers is given in Section 4.

### 3.1.2 Protecting TLB Entries

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

The entry protection mechanism is shown in Figure 15. To protect the first *n* TLB entries, set the base pointer field in the MMU lock register (LOCK\_REG) to *n*, and set the current victim pointer (also in the MMU lock register) to a value greater than or equal to *n*.

#### 3.1.3 Reading TLB Entries

Entries in the TLB can be read by using the victim pointer to specify the entry number. The parameters of this entry are then accessible in the CAM/RAM read registers (READ\_CAM\_H\_REG, READ\_CAM\_L\_REG, READ\_RAM\_H\_REG, and READ\_RAM\_L\_REG).

#### 3.1.4 Deleting TLB Entries

Two mechanisms exist to delete TLB entries. All unpreserved TLB entries, that is, TLB entries that were written with the preserved bit set to zero, can be deleted by invoking a TLB flush. Such a flush is invoked by setting the flush bit in the TLB global flush register (GFLUSH\_REG).

An individual TLB entry can be flushed, regardless of its preserved bit setting, by specifying its virtual address in the MMU CAM registers, and by setting the flush entry bit in the MMU flush entry register (FLUSH\_ENTRY\_REG).

The preserved bit should only be set for protected TLB entries, as it does **not** prevent replacement of the entry by the table walking logic.

#### 3.1.5 Enabling the MMU

The MMU is enabled by setting the enable bit in the MMU control register (CNTL\_REG). Clearing this bit disables the MMU. When the MMU is disabled, no translation is performed, meaning that virtual addresses are treated as physical addresses.

#### 3.2 MMU Error Handling

The following types of faults can occur in the address translation process:

• TLB miss (table walker disabled):

No translation is found in the TLB for the virtual address issued. The hardware table walker is disabled, and hence the translation cannot be retrieved from the translation table(s).

• Translation fault (table walker enabled):

No translation is found for the virtual address required (TLB miss). The table walker is enabled, but no valid page table entry exists for the given virtual address.

• Permission fault:

The section/page access permissions do not match the access type.

When a fault occurs and its corresponding interrupt is enabled, an interrupt is signaled to the ARM. The interrupt service routine (ISR) is then responsible for fault recovery. For example, for a TLB miss, the ISR might load the missing entry from a page table. The DSP memory interface is stalled by the DSP MMU while the fault is handled.

The ISR can determine the cause of the abort interrupt by reading the fault status register (F\_ST\_REG). The virtual address that caused the fault can be determined by reading the fault address registers (FAULT\_AD\_H\_REG and FAULT\_AD\_L\_REG).

In the case of a TLB miss, the MMU continues servicing the request as soon as a valid TLB entry has been written and the interrupt has been acknowledged via the acknowledge interrupt bit in the interrupt acknowledge register (IT\_ACK\_REG).

In the case of a permission fault, the corresponding TLB/translation table entry must be modified to allow for the requested access type before acknowledging the interrupt.

Figure 17 summarizes the MMU translation process and highlights the different error scenarios.

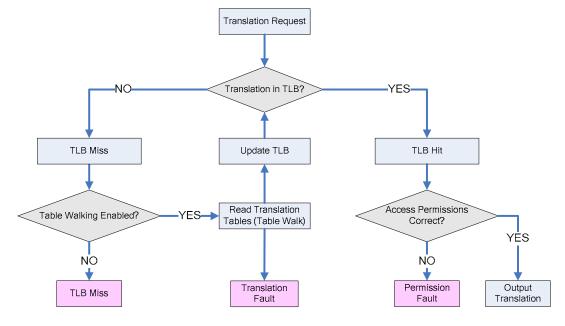


Figure 17. MMU Translation Process and Fault Types

## 4 MMU Register Manual

#### 4.1 MMU Register Set

The DSP MMU is programmed by the ARM via a set of 21 configuration registers. These registers, their access type, and their address are shown Table 3.

Register Name	Register Function	Address	Access Type
PREFETCH REG	Prefetch register	FFFE:D200	B/W
WALKING_ST_REG	Prefetch status register	FFFE:D204	R
CNTL_REG	Control register	FFFE:D208	R/W
FAULT_AD_H_REG	Fault address register MSB	FFFE:D20C	R
FAULT_AD_L_REG	Fault address register LSB	FFFE:D210	R
F_ST_REG	Fault status register	FFFE:D214	R
IT_ACK_REG	Interrupt acknowledge register	FFFE:D218	W
TTB_H_REG	TTB register MSB	FFFE:D21C	R/W
TTB_L_REG	TTB register LSB	FFFE:D220	R/W
LOCK_REG	Lock counter	FFFE:D224	R/W
LD_TLB_REG	Load entry in TLB	FFFE:D228	R/W
CAM_H_REG	CAM entry register MSB	FFFE:D22C	R/W
CAM_L_REG	CAM entry register LSB	FFFE:D230	R/W
RAM_H_REG	RAM entry register MSB	FFFE:D234	R/W
RAM_L_REG	RAM entry register LSB	FFFE:D238	R/W
GFLUSH_REG	Global flush register	FFFE:D23C	R/W
FLUSH_ENTRY_REG	Individual flush register	FFFE:D240	R/W
READ_CAM_H_REG	Read CAM register MSB	FFFE:D244	R
READ_CAM_L_REG	Read CAM register LSB	FFFE:D248	R
READ_RAM_H_REG	Read RAM register MSB	FFFE:D24C	R
READ_RAM_L_REG	Read RAM register LSB	FFFE:D250	R

 Table 3.
 MMU Programming Register Summary

## 4.2 MMU Register Descriptions

Table 4 through Table 24 describe the individual MMU registers.

Name	PREFETCH	I_RE	G	Phy	ysical	addro	ess	0xFFFE:D200			Access Type			R/W	
<b>Description</b> Prefetch register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	1 Program/data	Virtual address tag											•	•	
Bits	Field Name					[	Descr	iption						Тур	be
15	Reserved	Reserved													
14	Must always be programmed as 0 (program)												R/W		
13:0 Virtual address tag			Virtual address tag of the TLB entry to be prefetched											R/W	

#### Table 4. Prefetch Register (PREFETCH\_REG)

#### Table 5. Prefetch Status Register (WALKING\_ST\_REG)

N	Name WALKING_S			ST_R	EG Physical Address 0xFFFE:D204							Access Typ	be	R		
Desc	Description Prefetch status register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	
	Reserved Table walk													Ρ	refetch	
Bits		Fie	eld Na	me					[	Descri	iption					Туре
15:2		Reserv	/ed		Res	erved										
1	-	Table v	walk		1: T	able v	valk is	runni	ng.							R
0 Prefetch Set by writi Reset upor																R

Nam	ne		CNT	L_RE	EG				Phys	sical Add	lress	0xFF	FE:D208	Acc	cess Type	R/W
Des	cript	ion	Cont	trol re	egiste	r										
15	14	13	12	11	10	9	8	7	6	5	4	3	2		1	0
				Rese	erved					Burst	Rese	erved	Table wa	lk	Enable	Reset
											•					
Bits	Bits Field Name Description											Туре				
15:6		Res	serve	d		R	eserv	/ed								
5		Bur	st			1:	Ena	ble 1	16-bit	burst ma	Inagen	nent				R/W
4:3		Res	serve	d		R	eserv	/ed								
2		Reserved     Reserved       Table walk     1: Enable table walk       0: Enable user access to TLB													R/W	
1		Ena	able			1:	Ena	ble <b>I</b>	MMU							R/W
0		Res	set			0:	Res	et M	MU b	ack to de	efault c	onfigu	iration			R/W

## Table 6. Control Register (CNTL\_REG)

## Table 7. Fault Address Register MSB (FAULT\_AD\_H\_REG)

Nam	e		FAU	LT_A	D_H	RE	G Physical A	ddress	0xFF	FE:D2	0C	Access	Туре	R	1
Des	cript	ion	Fault	t add	ress	egis	ter MSB								
15	14 13 12 11 10 Reserved						8	7	6	5	4	3	2	1	0
	Reserved						Data/program			Fa	ault a	ddress M	SB		
Bits	ts Field Name							Des	criptio	n				Ту	/pe
15:9							eserved								
8	Data/program						ways reads as 0 (pro	gram)						ŀ	R
7:0	Fault address MSB					B M	SB of the address that	t cause	d the fa	ault				I	R

#### Table 8. Fault Address Register LSB (FAULT\_AD\_L\_REG)

Name		FAULT	ſ_AD_L	REG	ì	Phys	ical Ad	dress	0xFF	FE:D21	0	Access	Туре	R
Descr	iption	Fault a	address	regist	er LSB									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
			Fault	addres	ss LSB							Reserve	d	
Bits		Field	Name					Des	criptio	n				Туре
15:7	Fai	ult addro	ess LSE	3 LS	B of the	addre	ss that	caused	the fa	ult				R
6:0	Re	served		Re	served									

Nam	ne		F_S	T_RE	EG		Ρ	hysio	cal A	ddre	SS	0xFFFE:D	214	Access Typ	be	R
Des	cript	ion	Faul	t stat	us re	giste	r									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	(	)
					Rese	rved						Prefetch	Permission	TLB mss	Translat	ion fault
Bits			Field	d Na	me						0	Description	า		Ту	ре
15:4		Res	serve	d		R	eser	ved								
3		Pre	fetch			1:	Erro	or oc	curre	d du	ring a	a prefetch.			F	3
2	missi		1:	Erro	or is o	cause	ed by	ana	access peri	mission fault		F	3			
1 TLB miss 1: Error was caused										used	by a	TLB miss	(table walk d	isabled).	F	7
0	0 Translation fault 1: Err									used	by a	translation	ı fault.		F	3

#### Table 9. Fault Status Register (F\_ST\_REG)

## Table 10. Interrupt Acknowledge Register (IT\_ACK\_REG)

Name	е	IT_	ACK_	REG		Phy	sical	Addre	SS	0xFFFE	E:D218	8	Acc	ess T	уре	W
Desc	riptio	on Inte	errupt a	acknov	wledge	e regis	ter									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Reserved														Acknov	vledge
Bits		Fie	ld Na	me					Des	criptio	n				Ту	ре
15:1		Reserv	ed		Rese	erved										
<u> </u>		Aaluaau	با م ما م م		Writi	ng 0:	No eff	ect							14	,
0	4	Acknov	viedge		Writi	ng 1:	Ackno	wledg	es int	errupt,	clears	the st	atus t	oit	W	1

### Table 11. Translation Table Base Register (MSB) (TTB\_MSB\_REG)

Name		TTB_N	/ISB_R	EG	Phy	sical A	ddres	s (	)xFFFE:	D21C	Ac	cess T	уре	F	R,R/W
Descr	ription	Transl	ation ta	ble ba	se regis	ster (MS	SB)								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TTB MSB														
Bits		Field	Name					Desc	ription					Туре	9
15:0														(user m	node)
15:0	11			IVIC		e trans	ation ta	able b	ase audi	less			R/W	/ (supe	ervisor)



Table 12.	Translation	<b>Table Base</b>	Register	(LSB)	(TTB_	LSB_	REG)
-----------	-------------	-------------------	----------	-------	-------	------	------

Name		TTB_LS	SB_REG	à		Physica	al Addro	ess	0xFFFE	:D220	Acce	ess Ty	ре	F	R/RW
Descri	ption	Transla	tion tab	e base	e regist	ter (LSB)									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			T	TB LSE	3						Re	eserve	d		
Bits		Field N	lame				De	escrip	otion					Туре	e
15:7	<b>T</b> T	B LSB			of the	baar	e address				R (	user n	node)		
15.7	'''	DLJD		LOD	or the	liansiali		, nase	e audres:	5			R/W	(supe	ervisor)
6:0	Re	served													

### Table 13. Lock/Protect Entry Register (LOCK\_REG)

Na	me	LOCK	REG			Physi	cal Ad	dress		0xl	FFE:D	224	Access T	Гуре	R/W
Descr	iption	Lock/p	protect	entry r	egister										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ba	ise					Vic	tim				Rese	erved	
Bits		Field	Name					l	Descri	ption					Туре
15:10	Bas	se		Lo	cked e	ntry ba	se poir	nter							R/W
9:4	Vic	tim		Ne	ext entr	y to be	read/w	/ritten							R/W
3:0	Re	served													

#### Table 14. Read/Write TLB Entry Register (LD\_TLB\_REG)

Nam	ne		LD_T	LB_I	REG			Phy	/sica	l Ade	dress	5	0xFF	FE:D228	Access Type	R/W
Des	cript	ion	Read	l/write	e TLE	3 ent	ry re	giste	r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
						Rese	erved							Read TLB entry	Write TLB	entry
Bits			Field	l Nar	ne							C	escr	iption		Туре
15:2		Res	erveo	ł												
1		Rea	d TL	B ent	try	1:	Read	d TLE	3 ent	ry sp	ecifie	ed by	victir	m pointer		R/W
0		Writ	te TLE	3 ent	ry	1:	Write	e TLE	3 ent	ry sp	ecifie	ed by	victir	n pointer		R/W

Nam	е	(	CAM_H	_REG			Physi	cal A	ddres	S	0xFFF	E:D2	2C	Access	Туре	RW
Desc	ripti	on	CAM en	try reg	ister	(high	part)									
15	14	1:	3 12	11	10	9	8	7	6	5	4	3	2	1		0
						Res	erved							Virtual	ag MS	В
Bits			Field N	ame						[	Descri	otion				Туре
15:2		Res	erved													
1:0		Virtu	ial tag N	1SB	21	MSB c	of the v	virtual	addre	ess ta	g					R/W

#### Table 15. CAM Entry Register (MSB) (CAM\_H\_REG)

## Table 16. CAM Entry Register (LSB) (CAM\_L\_REG)

Nam	ne		CAN	1_L_	REG		Ρ	hysio	cal A	ddre	ss	0xFFFE:D230		Access 1	Гуре	R/W	I
Des	cript	ion	CAN	1 ent	ry reę	gister	· (low	part	)								
15	14	13	12	11	10	9	8	7	6	5	4	3	:	2	1	0	
				Vi	rtual	tag L	SB					Preserved	Va	alid		Туре	
Bits	Bits Field Name Description												Туре				
15:4														R/W			
3		Pre	serve	ed								upon TLB flush I upon TLB flus				R/W	
2	1: ILB entry is preserved upon ILB flush.												R/W				
1:0		Тур	e			0 1	1: La 0: Sr	ectior irge p nall p ny pa	age age	(64K (4K k	byte bytes					R/W	

#### Table 17. RAM Entry Register (MSB) (RAM\_H\_REG)

Name				ì	Phy	sical A	Addres	<b>s</b> (	xFFFE:	D234	Ac	cess T	уре	R/W	
Descr	Description RAM entry register (high part)														
15	14	14 13 12 11 10 9 8 7 6 5 4 3							2	1	0				
						Р	hysical	tag M	SB						
Bits Field Name Description							Туре								
15:0 Physical tag MSB 16 MSB of the					of the physical tag				R/W						



Table 18.	Lock/Protect Entry	Register	(LSB) (RAM_L	_REG)
-----------	--------------------	----------	--------------	-------

Name	•	RAN	1_L_R	EG		Physical A	ddress	0xFFF	E:D23	88		Acce	ess Ty	ре	R/W
Desc	riptio	n Lock	/prote	ct entr	y regi	ster									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Physical tag LSB					Access permissions Reserved									
Bits		Fiel	d Nam	ne			I	Descrip	otion						Туре
15:10	P	hysical	tag L	SB	6 LSE	3 of the phys	sical tag								R/W
9:8     Access permissions     00, 01: No access 10: Read only access 11: Read and write access						cess								R/W	
7:0 Reserved															

#### Table 19. Global Flush Register (GFLUSH\_REG)

Nam							Phy	sica	l Add	ress		0xFF	FE:D	23C	Access Type	R/W
Des	escription Global flush register					ster										
15	14	1:	3 12	11	10	9	8	7	6	5	4	3	2	1	0	
						Re	eserv	ed							Global flus	h
Bits			Field	Name	e						De	scrip	tion			Туре
15:1		Re	served													
0	Global flush					1: Flu reset	: Flush all non-protected TLB entries. This bit is automatically eset.							is automatically	R/W	

#### Table 20. Flush Entry Register (FLUSH\_ENTRY\_REG)

Nam	е		FLUSI	H_EN	ITRY	REG	Phy	sical	Add	ress		0xFF	FE:D	240	Access Typ	e R/W
Desc	Description Flush entry register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Reserved Flush entry										/					
Bits			Field	Name	е						De	scrip	tion			Туре
15:1 Reserved																
0Flush entry1: Flush the TLB entry specified by the virtual address tag in the CAM registers. This bit is automatically reset.									R/W							

Nam	е	RE	EAD_C	CAM_I	H_RE	G I	Physi	cal A	ddres	s	0xFFF	E:D2	44	Access	Туре	R
Desc	Description CAM entry read register (high part)															
15	14	13	13 12 11 10 9 8 7 6 5 4 3 2 1									0				
	Reserved Virtual tag I										B					
														viituari	ay wo	
						11000	erveu							Viituari	ay wo	5
Bits		Fi	eld Na	ame		1000				Des	criptic	on		Viituari		Гуре
<b>Bits</b> 15:2		<b>Fi</b> Reser		ame						Des	criptic	on				

#### Table 21. CAM Entry Read Register (MSB) (READ\_CAM\_H\_REG)

## Table 22. CAM Entry Read Register (LSB) (READ\_CAM\_L\_REG)

Name	READ_CAM_L	REG Physical address	0xFFFE:D24	Access 1	Гуре	R		
Descript	ion CAM entry read	register (low part)						
15 14	13 12 11 10	9 8 7 6 5 4	3	2	1	0		
	Virtual ta	lg LSB	Preserved	Valid	Г	Гуре		
Bits	Field Name		Description			Туре		
15:4	Virtual tag LSB	12 LSB of the virtual tag				R		
3	Preserved	0: Entry is not preserved up 1: TLB entry is preserved up				R		
2	Valid	0: Entry is not valid						
1:0Type00: Section (1M bytes) 01: Large page (64K bytes) 10:Small page (4K bytes) 11: Tiny page (1K bytes)								

#### Table 23. RAM Entry Read Register (MSB) (READ\_RAM\_H\_REG)

Name		READ	_RAM_	H_REG	à Phy	sical A	ddres	<b>s</b> (	xFFFE:	D24C	Ac	cess T	уре	R	1
Descr	Description RAM entry read register (high part)														
15	14	13	13 12 11 10 9 8 7 6 5 4 3 2							1	0				
						Р	hysical	tag M	SB						
Bits Field Name Description							Ту	pe							
15:0 Physical tag MSB 16 MSB of the physical tag						F	٦								



Name	Name RAM_L		_RE/	AD_I	REG	Physical A	ddress	0xFFF	E:D25	50		Ассе Туре		R		
Desc	Description Lock/protect entry re					y regis	ster									
15	14	13	12	2	11	10	9	8	7 6 5 4 3					2	1	0
	Physical tag LSB						Access pe	Reserved								
	1 Hjoloal ag 200								•							
Bits		Fi	eld Na	ame	;				Descri	ption					٦	Гуре
15:10	I	Physic	al tag	LSE	В	6 LSE	LSB of the physical tag								R	
9:8	9:8 Access permissions 00, 10: 11:				10: R	I: No access ead-only ac ead and wri	cess								R	
7:0	7:0 Reserved															

#### Table 24. Lock/Protect Entry Register (LSB) (READ\_RAM\_L\_REG)

## 5 Conclusion

The OMAP5910 architecture provides a flexible way for the DSP to access external memory. The DSP memory management unit enables DSP access to the entire 32-bit OMAP physical address space. The address mapping between DSP and OMAP addresses is flexible and can be configured at run time to enable dynamic memory allocation. The DSP MMU implements memory protection, adding additional security against unauthorized memory accesses by the DSP.

#### 6 References

1. OMAP5910 Dual-Core Processor Data Manual (SPRS197)



## Appendix A. Demonstration Software

#### A.1 Example Projects

Two example projects are provided to demonstrate the DSP MMU setup. The first example shows how to configure the DSP MMU by writing the TLB entries directly. This is the most common use case.

The second example shows how to program the DSP MMU using a first-level translation table. For the sake of simplicity, no two-level translation process is demonstrated. Also, the focus of these projects is rather on demonstrating the steps required to program the DSP MMU than on providing the most efficient implementation.

The projects are written for the Code Composer Studio<sup>™</sup> Integrated Development Environment (IDE) for OMAP, and the explanations assume that you are familiar with the Code Composer Studio<sup>™</sup> IDE. The source code, on the other hand, is generic and can be ported easily to any other build/debug environment.

To use and debug these projects, extract the source code provided with this application note into a directory of your choice using a zip extraction utility.

### A.2 Configuring the DSP MMU by Programming the TLB

The most common way to configure the DSP MMU is by writing the TLB entries directly without building translation tables. This is usually possible because the DSP has a comparatively small address space (16M bytes), and dynamic memory allocation via the MMU is not as common on the DSP as it is on the ARM.

This example provides a set of functions to read and write TLB entries and some generic MMU control functions. Two entries will be written into the TLB with the properties listed in Table A-1.

TLB Entry No.	DSP Virtual Start Address	Physical Start Address	Size	Access Permission
0	0x80:0000 (byte) 0x40:0000 (word)	0x1000:0000 (byte)	1MB	Read and write
1	0xA0:0000 (byte) 0x50:0000 (word)	0x2000:0000 (byte)	64kB	Read only

Table A-1. TLB Entry Properties

For the virtual (DSP) addresses, both the byte and the word addresses are given.

**NOTE**: Any DSP MMU error results in a memory stall; that is, the MMU will not process any request until the cause for the error has been addressed.

Because no dynamic error-correction mechanism has been implemented in the demonstration software, an MMU error usually results in an emulation timeout that requires a target power cycle and a Code Composer Studio<sup>™</sup> IDE restart.

#### A.2.1 Verification Procedure

To verify the program functionality:

- 1. Configure Code Composer Studio<sup>™</sup> IDE for your target platform (OMAP1510, OMAP1610, OMAP1710, OMAP5910, or OMAP5912).
- 2. Start Code Composer Studio<sup>™</sup> IDE, and open the ARM debugger window.
- 3. Open the project DSPMMU\_TLB.pjt.
- 4. Load the program file DSPMMU\_TLB.out from the \debug subdirectory.
- 5. Run the program. It configures the DSP MMU and then polls for possible error conditions.
- 6. Open a memory window in the ARM debugger at start address 0x10000000.
- 7. Open the DSP debugger.
- 8. Open a memory window to view data at address 0x400000. Make sure you set the page to DATA.
- 9. Compare the data values in both memory displays (ARM and DSP). They should be the same.
- 10. Change one data value in the DSP debugger (you can do this by double-clicking on it and writing the new value into the dialog box) and then refresh the memory window in the ARM debugger. The value changes there as well.
- 11. Verify the access permission functionality. Entry 1 in the TLB is configured as having only read permissions. Trying to write to its address space, therefore, results in a permission fault.
- 12. Change the start address of the DSP memory display to 0x500000. Try to write to a value there. This attempt results in an error message.
- 13. Check the ARM debugger. An access-permission error message displays in the stdout window.
- 14. Because the MMU does not process any request until the error cause has been addressed, you must now power cycle the processor and restart Code Composer Studio<sup>™</sup> IDE. In typical user software, an error handling mechanism is included.

## A.3 Configuring the DSP MMU by Building a Tanslation Table

Although configuring the DSP MMU by writing the TLB entries directly is more common, it is also possible to configure the DSP MMU by building translation tables, as the second example shows.

This example provides a function to build the 16-entry first-level translation table. The properties of its entries are listed in Table A-2. Some generic MMU control functions are also provided. Note that the DSP MMU maps only the DSP external memory space, that is, addresses above 0x28000 (byte)/0x14000 (word). Consequently, the first section is not a full 1M byte in size.

Table	DSP Virtual	Physical Start Address	Size	Access Permission
Entry No.	Start Address	,,		
0	0x02:8000 (byte) 0x01:4000 (word)	0x1005:0000 (byte)	1MB - 160kB	Read and write
1	0x10:0000 (byte) 0x08:0000 (word)	0x1010:0000 (byte)	1MB	Read only
2	0x20:0000 (byte) 0x10:0000 (word)	0x1020:0000 (byte)	1MB	Read and write
3	0x30:0000 (byte) 0x18:0000 (word)	0x1030:0000 (byte)	1MB	Read only
4	0x40:0000 (byte) 0x20:0000 (word)	0x1040:0000 (byte)	1MB	Read and write
5	0x50:0000 (byte) 0x28:0000 (word)	0x1050:0000 (byte)	1MB	Read only
6	0x60:0000 (byte) 0x30:0000 (word)	0x1060:0000 (byte)	1MB	Read and write
7	0x70:0000 (byte) 0x38:0000 (word)	0x1070:0000 (byte)	1MB	Read only
8	0x80:0000 (byte) 0x40:0000 (word)	0x1080:0000 (byte)	1MB	Read and write
9	0x90:0000 (byte) 0x48:0000 (word)	0x1090:0000 (byte)	1MB	Read only
10	0xA0:0000 (byte) 0x50:0000 (word)	0x10A0:0000 (byte)	1MB	Read and write
11	0xB0:0000 (byte) 0x58:0000 (word)	0x10B0:0000 (byte)	1MB	Read only
12	0xC0:0000 (byte) 0x60:0000 (word)	0x10C0:0000 (byte)	1MB	Read and write
13	0xD0:0000 (byte) 0x68:0000 (word)	0x10D0:0000 (byte)	1MB	Read only
14	0xE0:0000 (byte) 0x70:0000 (word)	0x10E0:0000 (byte)	1MB	Read and write
15	0xF0:0000 (byte) 0x78:0000 (word)	0x10F0:0000 (byte)	1MB	Read only

 Table A-2.
 First-level Translation Table Entry Properties

#### A.3.1 Verification Procedure

To verify the program functionality:

- 1. Configure Code Composer Studio<sup>™</sup> IDE for your target platform (OMAP1510, OMAP1610, OMAP1710, OMAP5910, or OMAP5912).
- 2. Start Code Composer Studio<sup>™</sup> IDE, and open the ARM debugger window.
- 3. Open the project DSPMMU\_Table.pjt.
- 4. Load the program file DSPMMU\_Table.out from the \debug subdirectory.
- 5. Run the program. It builds the first-level translation table used by the DSP MMU, configures the DSP MMU accordingly, and then polls for possible error conditions.
- 6. Open a memory window in the ARM debugger at start address 0x10200000.
- 7. Open the DSP debugger.
- 8. Open a memory window to view data at address 0x100000. Make sure you set the page to DATA.
- 9. Compare the data values in both memory displays (ARM and DSP). They should be the same.
- 10. Change one data value in the DSP debugger (you can do this by double-clicking on it and writing the new value into the dialog box), and then refresh the memory window in the ARM debugger. The value changes there as well.
- 11. Verify the access permission functionality. All odd entries in the first-level translation table (those corresponding to a start byte address with an odd most-significant digit) are configured as having only read permissions. Trying to write to its address space, therefore, results in a permission fault.
- 12. Change the start address of the DSP memory display to 0x180000. This corresponds to the physical address 0x10300000 in the ARM debugger. Try to change a value in the DSP memory display. This attempt results in an error message.
- 13. Check the ARM debugger. An access-permission error message displays in the stdout window.
- Because the MMU does not process any request until the error cause has been addressed, you must power cycle the processor and restart Code Composer Studio<sup>™</sup> IDE.

#### **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.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:

Texas Instruments

Post Office Box 655303 Dallas, Texas 75265

Copyright © 2004, Texas Instruments Incorporated