OMAP5910/5912 Multimedia Processor DSP Subsystem Reference Guide

Literature Number: SPRU890A
May 2005

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

Preface

Read This First

About This Manual

This document describes the OMAP5910/5912 multimedia processor DSP subsystem.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.

Related Documentation From Texas Instruments

Documentation that describes the OMAP5910/5912 devices, related peripherals, and other technical collateral, is available in the OMAP5910 Product Folder on TI's website: www.ti.com/omap5910, and in the OMAP5912 Product Folder on TI's website: www.ti.com/omap5912.

Trademarks

OMAP and the OMAP symbol are trademarks of Texas Instruments.

Contents

Figures

Tables

DSP Subsystem

1 Digital Signal Processor Subsystem Overview

1.1 Architecture Overview

The Digital Signal Processor (DSP) Subsystem is a collection of modules which include the TMS320C55x CPU processor along with its hardware accelerators, tightly coupled memory, instruction cache, and dedicated DMA, the interfaces it uses to communicate with rest of the OMAP device, as well as a number of peripherals.

The TMS320C55x core processor (also referred to as the DSP core) and the peripherals included in the DSP subsystem communicate with:

- The MPU core via the microprocessor unit interface (MPUI)
- Various standard memories via the external memory interface (EMIF)
- Various system peripherals via two TI peripheral bus (TIPB) bridges

Figure 1 and Figure 2 in section 1.4 show block diagrams for the OMAP5910 and OMAP5912 DSP subsystems.

1.2 Features

The DSP subsystem is composed of several portions: the DSP module, the peripherals that surround that module, and several interfaces used to communicate with the rest of the OMAP modules. Each portion has the following components:

- DSP module:
  - TMS320C55x (C55x) DSP core
  - Tightly coupled hardware accelerators: discrete cosine transform/inverse discrete cosine transform (DCT/IDCT), motion estimation, and half-pixel interpolation
  - Tightly coupled memories and their interfaces: dual-access RAM (DARAM), single-access RAM (SARAM), programmable dynamic ROM, and an instruction cache (I-Cache)
  - Six-channel DMA controller that can copy memory contents from one address to another without DSP core intervention

- DSP subsystem interfaces:
  - External memory interface (EMIF) that connects the DSP core to external and loosely coupled memories
  - MPUI port that permits access to DSP resources by the MPU and system DMA
  - TIPB that provides two external bus interfaces for private and public peripherals
- DSP subsystem peripherals:
  - Private peripherals are on the DSP private peripheral bus, and can only be accessed by the DSP core. DSP private peripherals include:
    - Three 32-bit timers
    - Watchdog timer
    - Interrupt handlers
  - Public peripherals are on the DSP public peripheral bus. These peripherals are directly accessible by the DSP core and DSP DMA. The MPU core can also access these peripherals through the MPUI port. DSP public peripherals include:
    - Two multichannel buffered serial ports (McBSPs)
    - Two multichannel serial interfaces (MCSIs)
  - The DSP core and DMA controller also have access to system peripherals (also referred to as shared peripherals). Shared peripherals are connected to both the MPU public peripheral bus and the DSP public peripheral bus. Shared peripherals include:
    - Mailbox module to permit interrupt-based signaling between the DSP and MPU cores
    - Three universal asynchronous receiver/transmitter (UART) modules
    - General-purpose input/output (GPIO) module
  - The OMAP5912 also adds these shared peripherals:
    - Eight general purpose timers
    - Serial port interface (SPI)
    - I2C master/slave interface
    - Extra McBSP
    - Multimedia card/secure digital interface (MMC/SDIO)
    - 32-KHz synchronization counter

This document describes all of the DSP module components listed above. The DSP subsystem peripherals are described in separate documents.

1.3 Differences Between the OMAP5910 and OMAP5912 DSP Subsystems

The OMAP5910 and OMAP5912 DSP subsystems are very similar. The difference between the subsystems lies in the mix of the MPU/DSP shared peripherals.

1.4 Functional Block Diagrams

Figure 1 and Figure 2 show functional block diagrams of the OMAP5910 and OMAP5912 DSP subsystems.

Figure 1. OMAP5910 DSP Subsystem and Modules

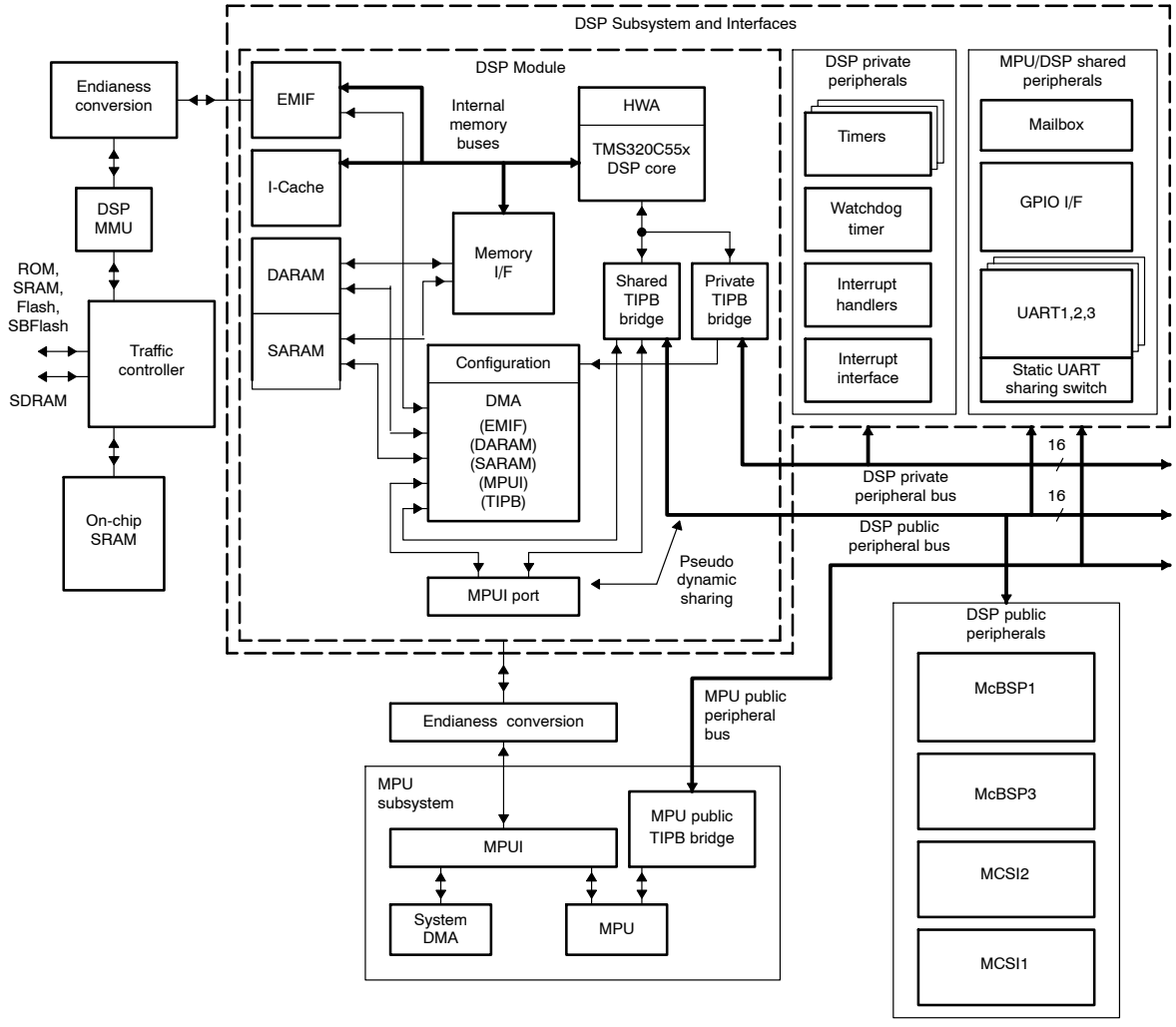
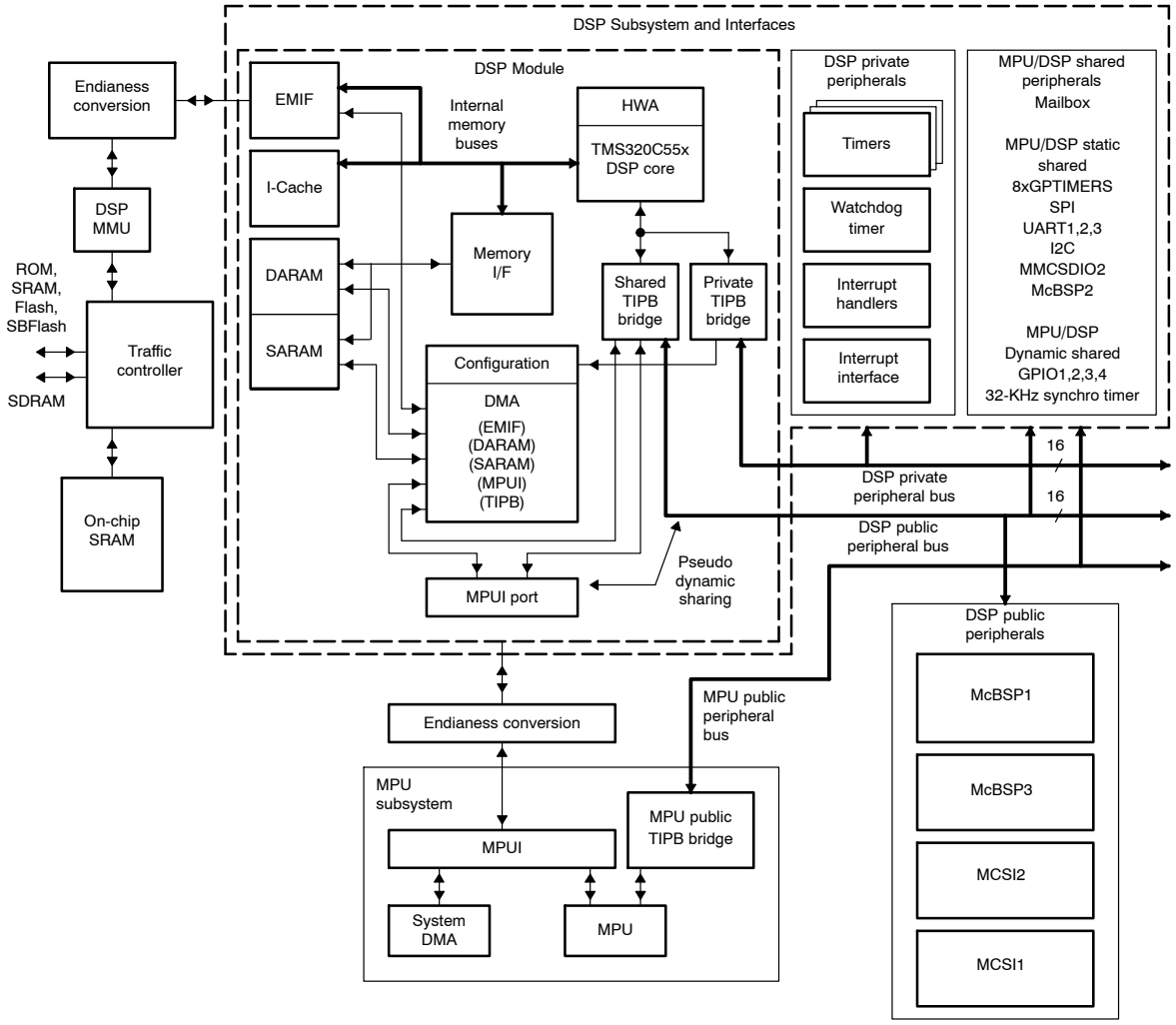


Figure 2. OMAP5912 DSP Subsystem and Modules



2 C55x DSP Core Overview

The DSP subsystem is based on the TMS320C55x DSP generation processor core. This section is intended to give a mere overview of the C55x DSP core. For detailed information, see the *TMS320C55x DSP CPU Reference Guide* (SPRU371).

2.1 DSP Core Features

Features of the high-performance, low-power DSP core include:

- ❏ Advanced multiple-bus architecture with one internal program memory bus and five internal data buses (three dedicated to reads and two dedicated to writes)
- ❏ Unified program/data memory architecture
- ❏ Dual 17-bit x 17-bit multipliers coupled to 40-bit dedicated adders for non-pipelined single-cycle multiply accumulate (MAC) operations
- ❏ Two address generators with eight auxiliary registers and two auxiliary register arithmetic units
- ❏ 8M x 16 bits (16M bytes) of total addressable memory space
- ❏ Single-instruction repeat or block repeat operations for program code
- ❏ Conditional execution
- ❏ Seven-stage pipeline for high instruction throughput
- ❏ Instruction buffer unit that loads, parses, queues, and decodes instructions to decouple the program fetch function from the pipeline
- ❏ Program flow unit that coordinates program actions among multiple parallel DSP core functional units
- ❏ Address data flow unit that provides data address generation and includes a 16-bit arithmetic unit capable of performing arithmetical, logical, shift, and saturation operations
- ❏ Data computation unit containing the primary computation units of the DSP core, including a 40-bit arithmetic logic unit, two MAC units, and a shifter
- ❏ Software-programmable idle domains that provide configurable low-power modes
- ❏ Automatic power management

2.2 Introduction to the DSP Core

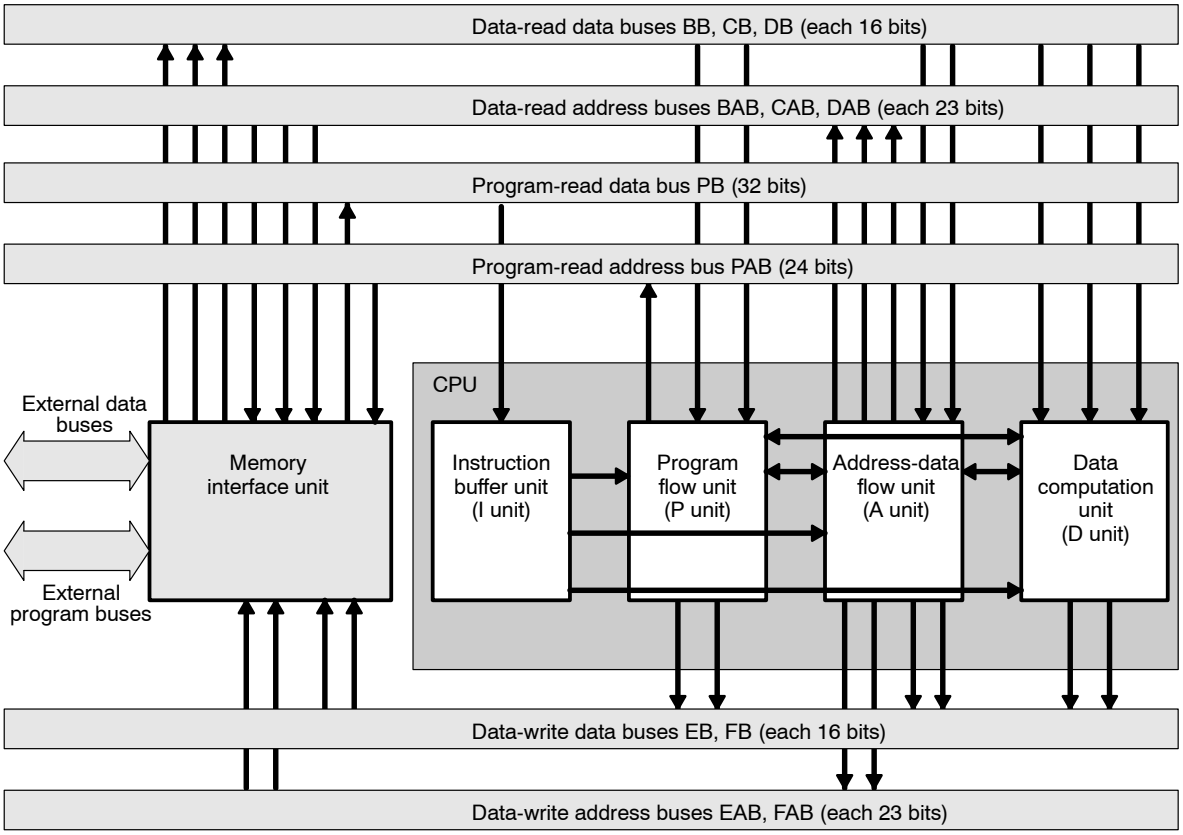
The DSP core supports an internal bus structure composed of one program bus, three data read buses, two data write buses, and additional buses dedicated to peripheral and DMA controller activity. These buses provide the ability to perform up to three data reads and two data writes in a single cycle.

The DSP core provides two multiply-accumulate (MAC) units, each capable of 17-bit x 17-bit multiplication in a single cycle. A central 40-bit arithmetic/logic unit (ALU) is supported by an additional 16-bit ALU. Use of the ALUs is under instruction set control, providing the ability to optimize parallel activity and power consumption. These resources are managed in the address unit (AU) and data unit (DU) of the DSP core.

The DSP core supports a variable byte width instruction set for improved code density. The instruction unit (IU) performs 32-bit program fetches from internal or DSP external memory and queues instructions for the program unit (PU). The program unit decodes the instructions, directs tasks to AU and DU resources, and manages the fully protected pipeline. Predictive branching capability avoids pipeline flushes on execution of conditional instructions.

Figure 3 shows a conceptual block diagram of the DSP core. Detailed information on each of the buses and units represented in this figure are given in the *TMS320C55x DSP CPU Reference Guide* (SPRU371).

Figure 3. DSP Core Diagram



Other useful documents include:

- *TMS320C55x DSP Mnemonic Instruction Set Reference Guide* (SPRU374): Describes the mnemonic instructions individually. It also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the algebraic instruction set.
- *TMS320C55x Programmer's Guide* (SPRU376): Describes ways to optimize C and assembly code for the TMS320C55x DSPs and explains how to write code that uses the special features and instructions of the DSP.
- *TMS320C55x Optimizing C Compiler User's Guide* (SPRU281): Describes the TMS320C55x C Compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for TMS320C55x devices.

- *TMS320C55x Assembly Language Tools User's Guide* (SPRU280): Describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for TMS320C55x devices.

2.3 Introduction to the Hardware Accelerators

Three powerful C55x hardware accelerator modules assist the DSP core in implementing algorithms that are commonly used in video compression applications such as MPEG4 encoders/decoders. These accelerators allow implementation of such algorithms using fewer DSP instruction cycles and dissipating less power than if the DSP core were operating alone. The hardware accelerators are utilized via functions from the TMS320C55x Image/Video Processing Library available from Texas Instruments.

The Image/Video Processing Library implements many useful functions utilizing the hardware accelerators, including:

- Forward and Inverse Discrete Cosine Transform (DCT) (used for video compression/decompression)
- Motion Estimation (used for compression standards such as MPEG video encoding and H.26x encoding)
- Pixel Interpolation (enabling high-performance fractal pixel motion estimation)
- Quantization/Dequantization (useful for JPEG, MPEG, H.26x encoding/decoding)
- Flexible 1D/2D Wavelet Processing (useful for JPEG2000, MPEG4, and other compression standards)
- Boundary and Perimeter Computation (useful for Machine Vision applications)
- Image Threshold and Histogram Computations (useful for various Image Analysis applications)

More information on the C55x Image/Video Processing Library can be found in the *TMS320C55x Image/Video Processing Library Programmer's Reference* (SPRU037).

There are three hardware accelerators included along with the C55x DSP core:

- DCT/IDCT Accelerator: This hardware accelerator implements Forward and Inverse DCT algorithms. These DCT/IDCT algorithms can enable a wide range of video compression standards including JPEG Encode/Decode, MPEG Video Encode/Decode, and H.26x Encode/Decode.
- Motion Estimation Accelerator: This hardware accelerator implements a high-performance motion estimation algorithm, enabling MPEG Video encoder or H.26x encoder applications. Motion estimation is typically one of the most computation-intensive operations in video-encoding systems.
- Pixel Interpolation Accelerator: This hardware accelerator enables high-performance pixel-interpolation algorithms, which allow for powerful fractal pixel motion estimation when used in conjunction with the Motion Estimation Accelerator. Such algorithms provide significant improvement to video-encoding applications.

Detailed information on the C55x Hardware Accelerators can be found in the *TMS320C55x Hardware Extensions for Image/Video Applications Programmer's Reference* (SPRU098).

3 DSP Subsystem Memory

The DSP subsystem requires access to three different types of memory: program memory, data memory, and I/O memory. The DSP subsystem architecture uses a unified program and data memory space composed of memory internal and external to the DSP subsystem. Internal memory is made up of tightly coupled memory blocks, whereas DSP external memory is mapped to OMAP system memory. The DSP subsystem architecture provides access to a maximum of 8M words (16M bytes) of program/data memory space.

The DSP subsystem I/O memory space is separate from the data/program memory space. The I/O space includes the configuration and data registers for all peripherals accessible by the DSP subsystem.

3.1 Internal Memory Space

The DSP subsystem memory consists of four types of tightly coupled memories which provide the DSP core with maximum efficiency.

- Dual-access RAM (DARAM)

  The DARAM memory consists of 8 blocks of 8K bytes each. The DARAM (64K bytes) can support up to two memory accesses into each RAM block in one DSP core clock cycle. Accesses can be made from any internal data, program, or DMA bus.

- Single-access RAM (SARAM)

  The SARAM memory consists of 12 blocks of 8K bytes each. The SARAM (96K bytes) can support one memory access into each RAM block in one DSP core clock cycle. This access can be a 32-bit value. Accesses can be made from any internal data, program, or DMA bus.

- Programmable dynamic ROM (PDROM)

  The PDROM memory consists of 1 block of 32K bytes. The programmable dynamic ROM (32K bytes) can support one memory read in one DSP core clock cycle. This access can be a 32-bit value. Accesses can be made from any internal data read or program bus.

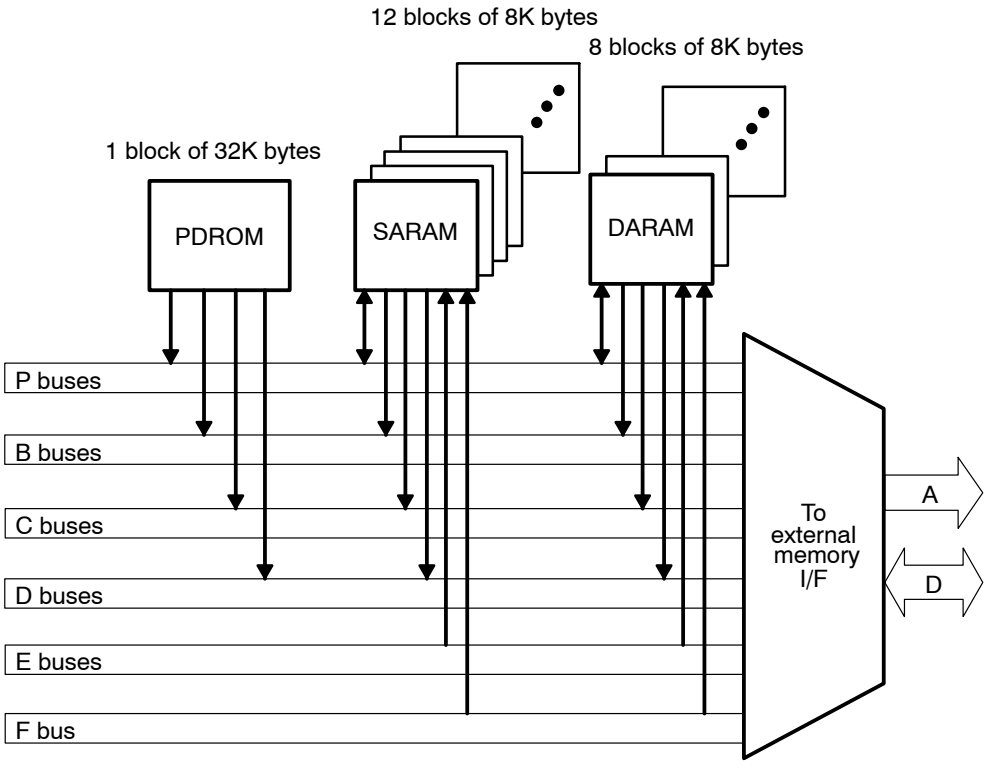
  The PDROM contains a program called a bootloader, which is executed by the DSP core when it is taken out of reset. Depending on the boot mode selected, the DSP core will either branch to an internal or DSP external memory address, or go into idle. Note that the memory at the destination address must be initialized with valid code before the bootloader is executed. Selecting boot mode 000b will disable the PDROM. The MPU core specifies the boot mode through the DSP_BOOT_CONFIG register. For more information on the DSP subsystem bootloader and the DSP_BOOT_CONFIG register, see section 12.4.

- Configurable I-Cache structure

  The DSP instruction cache (I-Cache) module is a special-purpose, tightly coupled, RAM-based program memory. The module is designed to significantly improve DSP core performance by buffering the instructions most recently fetched from DSP external memory. The entire external program memory space is cacheable. Section 4 describes the I-Cache in more detail.

Figure 4 shows the connections between the internal memory blocks and the buses of the DSP core.

Figure 4. Internal Memory Connections in the DSP Subsystem



The DSP core uses the six sets of buses to simultaneously fetch up to 32 bits of program code and to read up to 48 bits of data from memory (or to write up to 32 bits of data to memory). To achieve maximum performance from the architecture, pay close attention to placement of code and data structures within the on-chip memory resources. For more details, see the *TMS320C55x Programmer's Guide* (SPRU376).

3.2 DSP External Memory Space

The DSP core and DMA controller use the external memory interface (EMIF) to access the DSP external memory. External memory for the DSP subsystem ranges from byte address 0x02 8000 to 0xFF 8000 if the internal PDROM is enabled, or to 0xFF FFFF if the PDROM is not enabled. See Figure 18 for more details.

Note:

The term *DSP external memory* refers to memory outside of the DSP subsystem internal memory space. This includes program addresses in the range of 0x02 8000 to 0xFF 8000 if the internal PDROM is enabled, or to 0xFF FFFF if the PDROM is not enabled.

All DSP external memory access requests are passed through the DSP memory management unit (MMU). If this unit is enabled and configured by the MPU core, it translates the DSP external memory access request address, also called a virtual address, into a system memory address, also called a physical address, that is then passed to the traffic controller. The traffic controller completes the memory access through one of the three system memory interfaces: internal memory (IMIF), slow external memory (EMIFS), or fast external memory (EMIFF).

If the MMU is not enabled, then the access request is passed directly to the system traffic controller. In this case, the DSP virtual address is mapped to the first 16M bytes of chip select space 0 (CS0) of the system memory.

3.3 I/O Memory Space

The DSP subsystem I/O space is a separate address space from the data/program memory space. Configuration and data registers for all peripherals reside in the DSP subsystem I/O space, which consists of 64K-word addresses. Each peripheral maps into a 1K-word section of I/O memory.

OMAP devices include sets of peripherals grouped into three main categories: shared, public, or private.

- DSP/MPU shared peripherals are connected to both the MPU public peripheral bus and the DSP public peripheral bus. Connections are routed through a TI peripheral bus switch, which must be configured to allow MPU domain or DSP domain access. Some shared peripherals have permanent connections to both public peripheral buses, although read and write accesses to each peripheral register may differ.

- DSP public peripherals are connected to the DSP public peripheral bus and are directly accessible by the DSP core and DSP DMA. These peripherals may also be accessed by the MPU core and system DMA controller via the MPUI.
- DSP private peripherals are on the DSP private peripheral bus, and thus, can only be accessed by the DSP core.

To read or write to these registers, you must access the DSP subsystem I/O space either through C language constructs or, in the case of assembly-language code, by using a special instruction qualifier called the memory-mapped register access qualifier. For more details about this qualifier, see *TMS320C55x DSP Mnemonic Instruction Set Reference Guide* (SPRU374).

Note:

Byte access to I/O space is not supported.

The TI peripheral bus bridges manage accesses to the I/O memory space via two peripheral buses: a private TI peripheral bus and a public TI peripheral bus. Section 8 describes the TI peripheral bus bridges and their buses.

3.4 Memory Maps

Table 1 shows the high-level program/data memory map for the DSP subsystem. DSP core data accesses utilize 16-bit word addresses, while DSP core program fetches utilize byte addressing. DSP DMA data fetches always use byte addresses.

Table 1. OMAP5910/5912 DSP Subsystem Global Memory Map

Byte Address Range	Word Address Range	Internal Memory	DSP External Memory†
0x00 0000-0x00 FFFF	0x00 0000-0x00 7FFF	DARAM 64K bytes	
0x01 0000-0x02 7FFF	0x00 8000-0x01 3FFF	SARAM 96K bytes	
0x02 8000-0xFF 7FFF	0x01 4000-0x7F BFFF		Managed by DSP MMU
0xFF 8000-0xFF FFFF	0x7F C000-0x7F FFFF	PDROM (MPNMC = 0)	Managed by DSP MMU (MPNMC = 1)

† This space could be DSP external memory or internal shared system memory, depending on the DSP MMU configuration.

The I/O memory map varies from device to device, due to the different peripheral mixes. For a detailed I/O memory map, see the device-specific data manual.

4 Instruction Cache

4.1 Introduction

On the OMAP5912/10 applications processors, instructions for the C55x DSP core can reside in internal memory or in DSP external memory. When instructions reside in DSP external memory, the instruction cache (I-Cache) can improve the overall system performance by buffering the most recent instructions accessed by the DSP core.

Note:

The term *DSP external memory* refers to memory outside of the DSP subsystem internal memory space. This includes program addresses in the range of 0x02 8000 to 0xFF 8000 if the internal PDROM is enabled, or to 0xFF FFFF if the PDROM is not enabled.

4.1.1 Features

For storing instructions, the I-Cache contains:

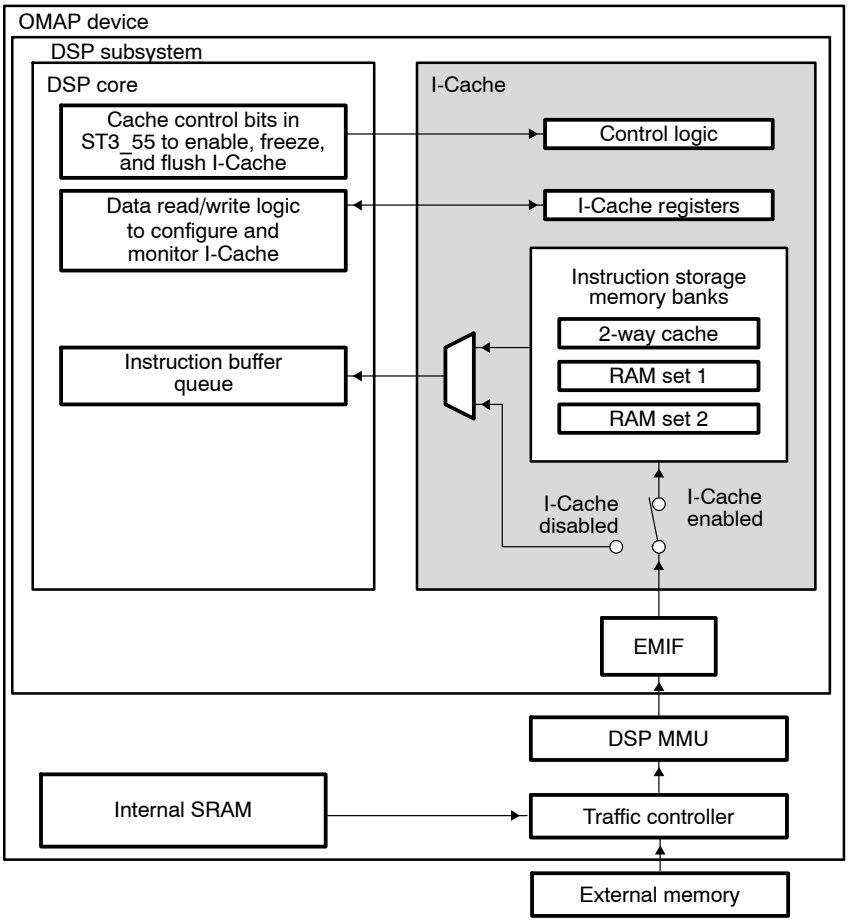
- One 2-way cache. The 2-way cache uses 2-way set associative mapping and holds up to 16K bytes: 512 sets, two lines per set, four 32-bit words per line. In the 2-way cache, each line is identified by a unique tag.
- Two RAM sets (1 and 2). These two banks of RAM are available to hold blocks of code. Each RAM set holds up to 4K bytes: 256 lines, four 32-bit words per line. Each RAM set uses a single tag to identify a continuous range of memory addresses that is represented in the RAM set. Before enabling the I-Cache, configure the I-Cache to use zero, one, or both RAM sets.

The DSP core status register, ST3_55, contains three cache control bits for enabling, freezing, and flushing the I-Cache (see section 4.2.4). To configure the I-Cache and check its status, the DSP core accesses a set of registers in the I-Cache (see section 4.6).

4.1.2 Functional Block Diagram

Figure 5 shows how the I-Cache fits into the DSP subsystem.

Figure 5. Conceptual Block Diagram of the I-Cache in the DSP Subsystem



4.1.3 Supported Cache Configurations

The I-Cache supports the following configurations:

- ❑ 2-way 16KB cache with no RAM set blocks
- ❑ 2-way 16KB cache with one 4KB RAM set block
- ❑ 2-way 16KB cache with two 4KB RAM set blocks

Sections 4.3, 4.4, and 4.5 detail the steps required to implement these cache configurations.

4.2 Instruction Cache Architecture

4.2.1 Introduction to the I-Cache

When the DSP core requests instructions, it requests 32 bits at a time. To initiate an instruction fetch, the DSP core sends a fetch request and a fetch address to the I-Cache.

If the I-Cache is enabled, it handles the fetch request as follows. If the requested word is in the I-Cache (a hit), the I-Cache delivers the word to the DSP core. If the requested word is not in the I-Cache (a miss), the I-Cache uses the external memory interface (EMIF) to fetch the 4-word DSP external memory block that contains the requested word. As soon as the requested word arrives in the I-Cache, it is delivered to the DSP core. Section 4.2.10 describes timing information for I-Cache hits and misses.

If the I-Cache is disabled, it is not checked. Instead, the fetch request and fetch address are passed to the EMIF. Once fetched by the EMIF, the requested 32-bit word is passed directly to the DSP core.

Notes:

1) The DSP external memory address generated by the EMIF is a virtual address. This virtual address is mapped to a physical address within the memory space of the OMAP device by the DSP Memory Management Unit (MMU). Before enabling the I-Cache, you must configure the DSP MMU such that the correct physical address is read during line-fill operations. Section 6 describes the DSP MMU.

2) The I-Cache does not automatically maintain coherency. If you write to a location in program memory, the corresponding line in the I-Cache is not updated. To regain coherency you must flush the I-Cache as described in section 4.2.4.2.

4.2.2 Instruction Cache Blocks

4.2.2.1 2-Way Cache

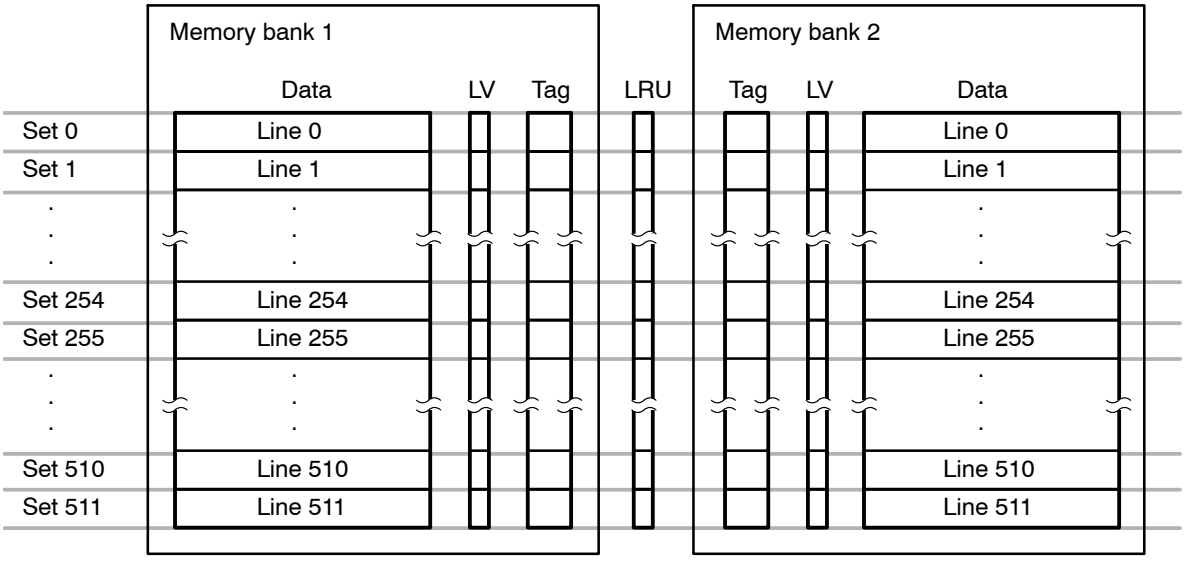
As shown in Figure 6, the 2-way cache has two memory banks. Each memory bank includes a:

- Data array. Each data array contains 512 lines (0 through 511) that the I-Cache can fill individually in response to misses in the 2-way cache.
- Line valid (LV) bit array. Each line has a line valid bit. Once a line has been loaded, its line valid bit is set. Whenever the I-Cache is flushed, all 512 line valid bits are cleared, invalidating all the lines. For more information on flushing the I-Cache, see section 4.2.4.2.

- Tag array. Each line has a tag field. When the I-Cache receives a 24-bit fetch address from the DSP core, the I-Cache interprets bits 23-13 as a tag. When a line gets filled, the associated tag is stored in the tag field for that line.

Across the two memory banks, every two lines with the same number belong to one set. For example, line 0 of memory bank 1 and line 0 of memory bank 2 belong to set 0. When the I-Cache receives a fetch address, the I-Cache finds the set number in bits 12-4. If the I-Cache must replace one of the lines in the set, it uses a least-recently used (LRU) algorithm: The line replaced is the one that has been unused for the longest time. Each set has an LRU bit that is toggled to indicate which line should be replaced.

Figure 6. 2-Way Cache



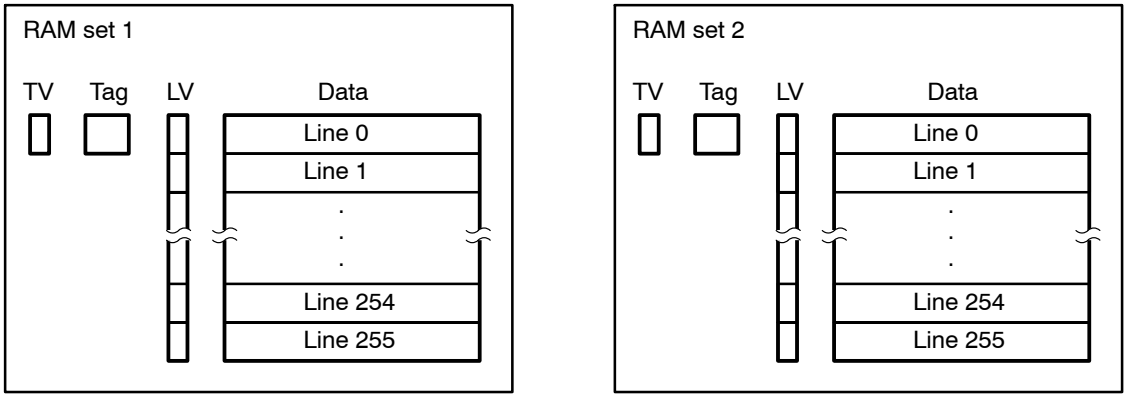
4.2.2.2 RAM Set Blocks

As shown in Figure 7, RAM set 1 and RAM set 2 each include the following parts:

- Data array. The data array contains 256 lines (0 through 255).
- Line valid (LV) bit array. Each line has a line valid bit. When a line has been loaded, its line valid bit is set. Whenever the I-Cache is flushed, all 256 line valid bits are cleared, invalidating all the lines. For more information on flushing the I-Cache, see section 4.2.4.2.

- Tag field. The RAM set has one 12-bit tag field that indicates which range of DSP external memory addresses are mapped to the RAM set. To select a tag for RAM set n (1 or 2), write to RAM set tag register n. When you write to the tag register, the I-Cache immediately fills the RAM set with all the 32-bit words in the address range specified by the tag. As each line is loaded, the associated line valid bit is set.
- Tag valid (TV) bit. The RAM set has one tag valid bit. Just before filling the RAM set, the I-Cache clears the tag valid bit. When the filling is complete, the I-Cache sets the tag valid bit. For RAM set n (1 or 2), the tag valid bit is reflected in RAM set control register n.

Figure 7. RAM Sets 1 and 2



The code that loads the RAM sets cannot be read from DSP external memory at the same time that the RAM sets are being loaded from memory. Therefore, place the RAM-set load code in internal memory.

The following pseudo-code example demonstrates the correct way to load the RAM set blocks.

```
Address Type     Pseudo Instruction
...
Ext Memory    DSP code
Ext Memory    GCR = #0xce2f          ; Select 2-way cache and two RAM sets
Ext Memory    NWCR = #0x000f         ; Initialize logic for 2-way cache
Ext Memory    RCR1 = #0x000f         ; Initialize logic for RAM set 1
Ext Memory    RCR2 = #0x000f         ; Initialize logic for RAM set 2
Ext Memory    Set CAEN in ST3_55     ; Turn on I-Cache
Ext Memory    Poll ENABLE bit of ISR ; Wait until cache is enabled
Ext Memory    goto Load_RAM_sets
...

Load_RAM_sets:
Int Memory    RTR1 = #0x0800         ; Update RAM set tag for bank1
Int Memory    Poll TAG_VALID in RCR1 ; Wait until line is filled in RAM set 1
Int Memory    RTR2 = #0x0801         ; Update RAM set tag for bank2
Int Memory    Poll TAG_VALID in RCR2 ; Wait until line is filled in RAM set 2
Int Memory    goto Back_from_RAM_set_preload
...

Back_from_RAM_set_preload:
Ext Memory    DSP code
Ext Memory    DSP code
...
```

4.2.3 Instruction Cache Operation

When the DSP core requests instructions, it requests 32 bits at a time. With each request, the DSP core sends a fetch address that indicates where to read the 32 bit requested word. When a fetch request arrives, the I-Cache performs an instruction presence check; that is, it determines whether the requested word is available in the 2-way cache and/or any RAM sets included in the I-Cache configuration.

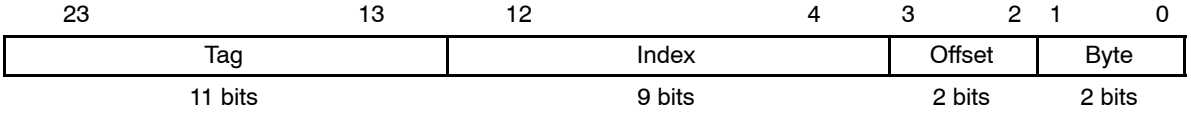
Because the 2-way cache and RAM-set architectures are different, the I-Cache interprets the fetch address differently when searching the 2-way cache and when searching the RAM set. Section 4.2.3.1 explains the differences.

Section 4.2.3.2 describes the steps of the instruction presence check and explains the factors that determine whether the I-Cache fetches the requested word from a RAM set, from the 2-way cache, or from DSP external memory. Whenever possible, the I-Cache gets the requested word from a RAM set. If the requested word is in a RAM set but not in the 2-way cache, the word is fetched from the RAM set and the 2-way cache is not loaded with that word.

4.2.3.1 How the I-Cache Uses the DSP core Fetch Address

Figure 8 and Table 2 describe how the I-Cache uses the fetch address for the 2-way cache. Figure 9 and Table 3 describe the same for a RAM set.

Figure 8. Fetch Address Fields for the 2-Way Cache Register

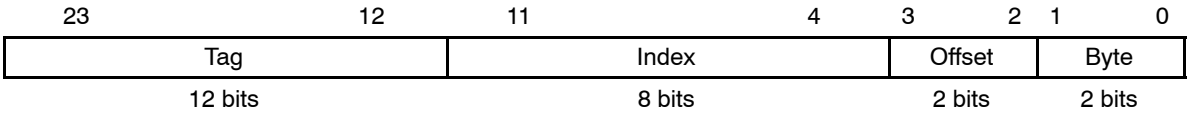


Note: R = Read, W = Write

Table 2. Fetch Address Field Descriptions for the 2-Way Cache Register Field Descriptions

Bits	Field	Value	Description
23–13	Tag		Whenever a line of the 2-way cache is loaded from DSP external memory, the tag portion of the fetch address is stored with the line (in the tag array). During an instruction presence check, the I-Cache uses the Index field to find the addressed set and then compares both tags in the set with the tag portion of the fetch address.
12–4	Index		This 9-bit value references one of the 512 sets of the 2-way cache. As shown in Figure 6, each set has two lines.
3–2	Offset		When the I-Cache must read a 32-bit word from one of the lines of the 2-way cache, the offset field indicates which of the four 32-bit words in the line should be read.
1–0	Byte		This field is not used by the I-Cache but is the part of the fetch address that indicates the specific byte being addressed.

Figure 9. Fetch Address Fields for a RAM Set



Note: R = Read, W = Write

Table 3. Fetch Address Field Descriptions for a RAM Set

Bits	Field	Value	Description
23–13	Tag		During an instruction presence check, the I-Cache compares the tag portion of the fetch address with the tag defined in the RAM-set tag register.
11–4	Index		This 8-bit value references one of the 256 lines of the RAM set.
3–2	Offset		When the I-Cache must read a 32-bit word from one of the lines of the RAM set, the offset field indicates which of the four 32-bit words in the line should be read.
1–0	Byte		This field is not used by the I-Cache but is the part of the fetch address that indicates the specific byte being addressed.

4.2.3.2 Instruction Presence Check and Corresponding I-Cache Response

When a fetch request arrives, the I-Cache performs an instruction presence check to determine whether the 32-bit requested word is available in the I-Cache. During the instruction presence check, the I-Cache performs two operations on both the 2-way cache and the RAM sets:

1) Compares the tag portion of the fetch address with the tag in the data array at the location referenced by the Index portion of the fetch address.

2) Checks the line valid bit at the referenced location to determine whether the line associated with the tag is valid.

If the tag comparison fails and/or the line valid bit is 0, this qualifies as a miss. If the instruction presence check finds a tag match and the line valid bit is 1, this qualifies as a hit. Table 4 summarizes the possible presence check cases (1 through 6) and the corresponding I-Cache responses. Whenever a line in the I-Cache must be loaded from DSP external memory (cases 1, 2, and 5), the I-Cache uses the line load process described in section 4.2.3.3.

Table 4. Instruction Presence Check and I-Cache Response

Case	2-Way Case	RAM Sets	Presence	I-Cache Response
1	Miss	Miss (no tag match)	True	2-way cache line loaded from DSP external memory, requested 32-bit word delivered to DSP core
2	Miss	Miss but tag match	True	RAM set line loaded from DSP external memory, requested 32-bit word delivered to DSP core
3	Miss	Hit	True	Requested 32-bit word taken directly from RAM set; 2-way cache line not loaded
4	Hit	Miss (no tag match)	True	Requested 32-bit word taken directly from 2-way cache
5	Hit	Miss but tag match	True	RAM set line loaded from DSP external memory, requested 32-bit word delivered to DSP core
6	Hit	Hit	True	Requested 32-bit word taken directly from RAM set

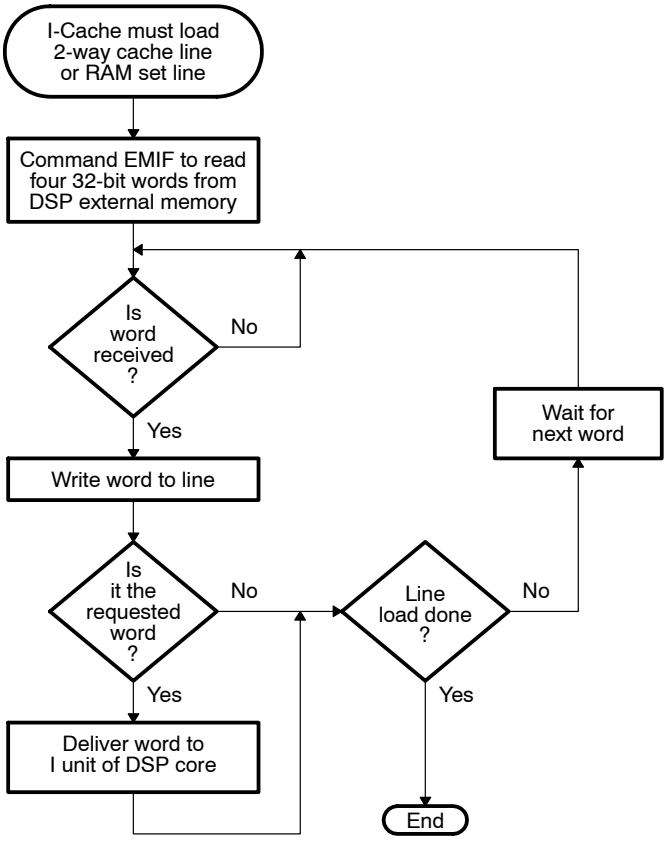
4.2.3.3 Line Load Process

When an instruction presence check results in a fetch from the DSP external memory, the 4-word DSP external memory block that contains the requested word is fetched and loaded into a line in the I-Cache. Figure 10 illustrates this line load process. The I-Cache uses the external memory interface (EMIF) to fetch the 4-word block. These four 32-bit words are written to the line in the I-Cache one word at a time. The I-Cache delivers the requested word to the DSP core as soon as the word arrives in the data array, even if the rest of the line is still being loaded. When the entire line is loaded in the data array, the corresponding tag is written to the tag array and the line valid bit is set to validate the line.

Note:

The DSP external memory address generated by the EMIF is a virtual address. This virtual address is mapped to a physical address within the memory space of the OMAP device by the DSP Memory Management Unit (MMU). Before enabling the I-Cache, you must configure the DSP MMU such that the correct physical address is read during line fill operations. Section 6 describes the DSP MMU).

Figure 10. Flow Chart of the Line Load Process



4.2.4 DSP Core Bits for Controlling the I-Cache

The I-Cache is controlled not only through the I-Cache registers but also through three bits located in status register ST3_55 of the DSP core. These bits are highlighted in Figure 11. For more details about ST3_55, see the *TMS320C55x DSP CPU Reference Guide* (SPRU371).

Figure 11. CAFRZ, CAEN, and CACLR Bits in ST3_55

15	14	13	12	11	10	9	8
CAFRZ	**CAEN**	**CACLR**	HINT†	Reserved‡		HOM_R	HOM_P
RW-0	RW-0	RW-0	RW-1	RW-11b		RW-x	RW-x
7	6	5	4	3	2	1	0
CBERR	MPNMC	SATA	Reserved	Reserved	CLKOFF§	SMUL	SST
RW-0	RW-x	RW-0	RW-0	R-0	RW-0	RW-0	RW-0

† This bit is not used in OMAP5910/5912, always keep this bit as 1.
‡ Always write 11b to these bits.
§ This bit must always be kept as 0.

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

4.2.4.1 CAEN to Enable and Disable the I-Cache

To enable the I-Cache, set the cache enable (CAEN) bit of ST3_55. To disable the I-Cache, clear the CAEN bit. When disabled, the lines of the I-Cache data arrays are not checked; instead, the I-Cache forwards instruction-fetch requests directly to the external memory interface (EMIF).

For proper I-Cache operation, configure the I-Cache before enabling it and disable the I-Cache before making any changes to its configuration. The procedures for configuring and enabling the I-Cache are in sections 4.3, 4.4, and 4.5.

A DSP subsystem reset forces CAEN = 0 (I-Cache disabled).

Note:

The DSP external memory address generated by the EMIF is a virtual address. This virtual address is mapped to a physical address within the memory space of the OMAP device by the DSP Memory Management Unit. Before enabling the I-Cache, you must configure the DSP MMU such that the correct physical address is read during line fill operations. Section 6 describes the DSP MMU).

4.2.4.2 CACLR Bit to Flush the I-Cache

The flush operation is defined as the invalidation of all of the lines in the I-Cache.

To flush the I-Cache, write 1 to the cache clear (CACLR) bit of ST3_55. In response, all the line valid bits of the 2-way cache and of the RAM sets are cleared. In addition, the tag valid bit of each RAM set is cleared. The CACLR bit remains 1 until the flush process is complete, at which time CACLR is automatically reset to 0.

A DSP subsystem reset forces CACLR = 0 (no flush in process).

4.2.4.3 CAFRZ Bit to Freeze the Contents of the I-Cache

When you write 1 to the cache freeze (CAFRZ) bit of ST3_55, the contents of the I-Cache are locked. Instruction words that were cached prior to the freeze are still accessible in the case of an I-Cache hit, but the data arrays are not updated in response to an I-Cache miss. To re-enable updates, clear CAFRZ.

A DSP subsystem reset forces CAFRZ = 0 (I-Cache not frozen).

Note:

When the I-Cache is frozen (CAFRZ = 1), each I-Cache miss still causes a 4-word (16-byte) fetch cycle in the EMIF. One of those words is returned to the DSP core and the rest are discarded. It is recommended that you profile your code to minimize the number of misses during an I-Cache freeze.

4.2.5 Initialization

Sections 4.3, 4.4, and 4.5 outline the procedures for configuring and enabling the I-Cache for the three I-Cache configurations:

- 2-way 16KB cache with no RAM set blocks
- 2-way 16KB cache with one 4KB RAM set block
- 2-way 16KB cache with two 4KB RAM set blocks

Section 4.6 describes the I-Cache registers. Section 4.2.4.1 describes the cache enable (CAEN) bit that is used to enable and disable the I-Cache.

Write to the control registers (GCR, NWCR, RCR1, and RCR2) only when the I-Cache is disabled (CAEN = 0 in ST3_55).

Write to the RAM-set tag registers (RTR1 and RTR2) only when the I-Cache is enabled (after making CAEN = 1 in ST3_55, wait for ENABLE = 1 in ISR).

4.2.6 Reset Considerations

After a DSP subsystem reset, the I-Cache is not automatically reconfigured for use. Make sure that your DSP initialization code configures the I-Cache as described in sections 4.3, 4.4, and 4.5 after every reset.

4.2.7 Clock Control

The DSP I-Cache is part of the DSP module within the DSP subsystem (see section 1.2) and is therefore clocked by the DSP subsystem master clock, DSP_CK. Section 12.2 describes the DSP subsystem master clock.

4.2.8 Power Management

If you want to temporarily halt the I-Cache to reduce power, you can place its domain in idle mode:

1) Select the idle mode for the I-Cache domain by making CACHEI = 1 in the idle configuration register (ICR) of the DSP subsystem. Section 12.3.2.8 describes ICR.

2) Execute the IDLE instruction from the DSP core.

When the I-Cache is in its idle mode or is disabled, instruction-fetch requests are handled by the external memory interface (EMIF).

To wake the I-Cache from its idle mode:

1) Deselect the idle mode by making CACHEI = 0 in ICR.

2) Execute the IDLE instruction.

4.2.9 Emulation Considerations

The software emulator reads the contents of the I-Cache during the debug mode. The contents of the I-Cache are not modified by emulator read operations.

If you set or remove a software breakpoint at an instruction during emulation, the corresponding line in the I-Cache is automatically invalidated.

4.2.10 Timing Considerations

As the I-Cache fetches and returns 32-bit words requested by the DSP core, two key time periods affect the speed of the I-Cache: hit time and miss penalty.

4.2.10.1 Hit Time

The hit time is the time required for the I-Cache to deliver the 32-bit requested word to the DSP core in the case of a hit (when the word is present in the I-Cache). The hit time is either 1 or 2 DSP core clock cycles:

- An initial request (a request that follows a period of inactivity) has a hit time of 2 cycles.
- Subsequent requests have a hit time of 1 cycle if:
  - The requests are consecutive (no inactivity in between) and
  - The requests are to sequential addresses
- Subsequent requests have a hit time of 2 cycles if:
  - The requests are not consecutive or
  - The requests are to non-sequential addresses

4.2.10.2 Miss Penalty

The miss penalty is the time required for the I-Cache to deliver the 32-bit requested word to the DSP core in the case of a miss (when the word must be fetched from DSP external memory). In response to a miss, the I-Cache requests four words from the external memory interface (EMIF) to load the appropriate line.

The miss penalty due to an initial request to the EMIF is:

1) Four cycles for the I-Cache to receive the fetch request, detect an I-Cache miss, and forward the fetch request to the EMIF.

2) X cycles for the EMIF to get the requested word to the I-Cache, where X depends on factors such as:

   a) The access latency introduced by the traffic controller.

   b) The position of the requested word in the I-Cache line. For example, if the requested word is the third word of the line, two words are fetched before the requested word.

   c) Whether the four words are fetched in a burst access (if synchronous memory is used).

3) Three cycles for the I-Cache to get the requested 32-bit word to the instruction fetch unit (I unit) of the DSP core.

Subsequent requests can incur a smaller miss penalty if the DSP external memory is synchronous. After accessing the first word from synchronous memory, the EMIF can return each of the remaining words in a single cycle.

The I-Cache includes a feature that reduces overall miss penalties. The I-Cache gives the requested word to the DSP core as soon as it arrives in the I-Cache line, rather than after the whole line is loaded.

Note:

The DSP external memory address generated by the EMIF is a virtual address. This virtual address is mapped to a physical address within the memory space of the OMAP device by the DSP Memory Management Unit (MMU). Before enabling the I-Cache, you must configure the DSP MMU such that the correct physical address is read during line fill operations. section 6 describes the DSP MMU).

4.3 Configuring the I-Cache With the 2-Way Cache and No RAM Set Blocks

The instruction cache is used to store recently-used instructions stored in DSP external memory. The I-Cache automatically fills its 2-way cache with instructions accesses from DSP external memory, in this manner subsequent accesses are essentially fetched from internal memory.

This section describes how to configure the I-Cache such that the 16KB 2-way cache is enabled with no RAM set blocks.

4.3.1 Architectural/Operational Description

When the DSP core fetches an instruction from DSP external memory, the I-Cache performs an instruction presence check to determine whether the 32-bit requested word is available in the I-Cache. If the instruction is found, the I-Cache returns the requested instruction to the DSP core; otherwise a DSP external memory access request is forwarded to the external memory interface (EMIF). The EMIF passes that request to the DSP Memory Management Unit (if enabled). After address translation, the DSP MMU places a request to the traffic controller which accesses shared memory via the OMAP external memory interfaces (EMIFF and EMIFS).

4.3.2 Software Configuration

Follow this procedure to select the 2-way cache and no RAM sets:

1) Write to the appropriate control registers:
   a) Write CA0Fh to GCR to indicate N-way cache is used in a 2-way configuration and that no RAM sets are needed.
   b) Write 000Fh to NWCR to initialize the logic for the 2-way cache.
2) Set the cache enable bit (CAEN) bit of DSP core status register ST3_55 to send an enable request to the I-Cache.
3) Poll the I-Cache-enabled (ENABLE) bit of ISR until ENABLE = 1. (The I-Cache is not instantaneously enabled.)

4.3.3 System Traffic Considerations

All DSP subsystem accesses to DSP external memory eventually go through the traffic controller. The access time for a DSP external memory request will depend on the amount of competing accesses in the traffic controller as well as the configurations of the OMAP external memory interfaces (EMIFF and EMIFS).

4.4 Configuring the I-Cache With the 2-Way Cache and One RAM Set

The instruction cache is used to store recently-used instructions in the DSP external memory. The I-Cache automatically fills its two-way cache with instruction accesses from DSP external memory, thus, subsequent accesses are essentially fetched from internal memory. Blocks of instructions can also be pre-fetched into the RAM set blocks.

This section describes how to configure the I-Cache such that the 16KB two-way cache is enabled with one 4KB RAM set block.

4.4.1 Architectural/Operational Description

When the DSP core fetches an instruction from DSP external memory, the I-Cache performs an instruction presence check to determine whether the 32-bit requested word is available in the I-Cache. If the instruction is found, the I-Cache returns the requested instruction to the DSP core. Otherwise, a DSP external memory access request is forwarded to the external memory interface (EMIF). The EMIF passes that request to the DSP Memory Management Unit (if enabled). After address translation, the DSP MMU places a request to the traffic controller which accesses shared memory via the OMAP external memory interfaces (EMIFF and EMIFS).

4.4.2 Software Configuration

Follow this procedure to configure with 2-way cache and one RAM set:

1) Write to the appropriate control registers:
   Write CE0Fh to GCR to indicate one RAM set.
   Write 000Fh to NWCR to initialize the logic for the 2-way cache.
   Write 000Fh to RCR1 to initialize the logic for RAM set 1.

2) Set the cache enable bit (CAEN) bit of DSP core status register ST3_55 to send an enable request to the I-Cache.

3) Poll the I-Cache-enabled (ENABLE) bit of ISR until ENABLE = 1. (The I-Cache is not instantaneously enabled.)

4) Write the desired tag to RTR1. When you write to the tag register, the tag is used to immediately fill RAM set 1 from DSP external memory.

   While the I-Cache is enabled, you can write to the tag register at any time to change the RAM-set address range. Each time you load the tag register, RAM set 1 is immediately filled from the selected address range.

5) To monitor the RAM-set filling, poll the tag-valid bit: When TAG_VALID = 1 in RCR1, the I-Cache has finished filling RAM set 1.

Note:

The code that loads the RAM sets cannot be read from DSP external memory at the same time that the RAM sets are being loaded from memory. Therefore, place the RAM-set load code in memory that is internal to the DSP subsystem.

4.4.3 System Traffic Considerations

All DSP subsystem accesses to DSP external memory eventually go through the traffic controller. The access time for a DSP external memory request will depend on the amount of competing accesses in the traffic controller, as well as the configurations of the OMAP external memory interfaces (EMIFF and EMIFS).

4.5 Configuring the I-Cache With the 2-Way Cache and Two RAM Sets

The instruction cache is used to store recently-used instructions stored in DSP external memory. The I-Cache automatically fills its two-way cache with instruction accesses from DSP external memory, thus, subsequent accesses are essentially fetched from internal memory. Blocks of instructions can also be pre-fetched into the RAM set blocks.

This section describes how to configure the I-Cache such that the 16KB two-way cache is enabled with two 4KB RAM set blocks.

4.5.1 Architectural/Operational Description

When the DSP core fetches an instruction from DSP external memory, the I-Cache performs an instruction presence check to determine whether the 32-bit requested word is available in the I-Cache. If the instruction is found, the I-Cache returns the requested instruction to the DSP core, otherwise a DSP external memory access request is forwarded to the DSP external memory interface (EMIF). The EMIF passes that request to the DSP Memory Management Unit (if enabled). After address translation, the DSP MMU places a request to the traffic controller, which accesses shared memory via the OMAP external memory interfaces (EMIFF and EMIFS).

4.5.2 Software Configuration

Follow this procedure to configure with 2-way cache and two RAM sets:

1) Write to the appropriate control registers:
   Write CE2Fh to GCR to indicate two RAM sets.
   Write 000Fh to NWCR to initialize the logic for the 2-way cache.
   Write 000Fh to RCR1 to initialize the logic for RAM set 1.
   Write 000Fh to RCR2 to initialize the logic for RAM set 2.
2) Set the cache enable bit (CAEN) bit of DSP core status register ST3_55 to send an enable request to the I-Cache.
3) Poll the I-Cache-enabled (ENABLE) bit of ISR until ENABLE = 1, indicating that the I-Cache is enabled. (The I-Cache is not instantaneously enabled.)
4) Write to the RAM set tag registers:
   a) Write the desired tag to RTR1. When you write to the tag register, the tag is used to immediately fill RAM set 1 from DSP external memory.
   b) Write the desired tag to RTR2. When you write to the tag register, the tag is used to immediately fill RAM set 2 from DSP external memory.

   While the I-Cache is enabled, you can write to a tag register at any time to change the address range as necessary. Each time you load a tag register, the corresponding RAM set is immediately filled from the selected address range.
5) To monitor the RAM-set filling, poll the tag-valid bits:
   a) When TAG_VALID = 1 in RCR1, the I-Cache is done filling RAM set 1.
   b) When TAG_VALID = 1 in RCR2, the I-Cache is done filling RAM set 2.

Notes:

1) Do not write the same value to both RAM set tag registers.
2) The code that loads the RAM sets cannot be read from DSP external memory at the same time that the RAM sets are being loaded from memory. Therefore, place the RAM-set load code in memory that is internal to the DSP subsystem.

4.5.3 System Traffic Considerations

All DSP subsystem accesses to DSP external memory eventually go through the traffic controller. The access time for a DSP external memory request will depend on the amount of competing accesses in the traffic controller, as well as the configurations of the OMAP external memory interfaces (EMIFF and EMIFS).

4.6 Instruction Cache Registers

4.6.1 Overview

Control of the I-Cache is maintained through a set of registers within the I-Cache. These registers are accessible only at addresses in the I/O memory space of the DSP subsystem.

Note:

Not every function documented in these registers is supported on OMAP5910 and OMAP5912. The functions not supported are listed in the section describing each register. Sections 4.3, 4.4, and 4.5 detail the steps needed to correctly configure and initialize the DSP I-Cache in the three supported modes of operation.

Table 5. Summary of the I-Cache Registers

Name	Description	DSP I/O Address†	See Section
GCR	Global control register. Use this register to select the number of active RAM sets.	0x1400	4.6.2
FLR0 FLR1	Flush line registers. Use these registers to flush a line from the cache.	0x1401 0x1402	4.6.3
NWCR	N-way control register. Use this register to initialize the logic for the 2-way cache.	0x1403	4.6.4
RCR1	RAM set 1 control register. Use this register to initialize the logic for RAM set 1 and to check the corresponding tag-valid flag.	0x1405	4.6.5
RCR2	RAM set 2 control register. Use this register to initialize the logic for RAM set 2 and to check the corresponding tag-valid flag.	0x1407	4.6.5
RTR1	RAM set 1 tag register. Use the register to define the 12-bit tag for RAM set 1.	0x1406	4.6.6
RTR2	RAM set 2 tag register. Use this register to define the 12-bit tag for RAM set 2.	0x1408	4.6.6
ISR	Status register. Use this register to verify that the I-Cache is enabled before you write to either of the RAM set tag registers.	0x1404	4.6.7

† DSP I/O addresses apply to both OMAP5910 and OMAP5912.

4.6.2 I-Cache Global Control Register (GCR)

Before enabling the I-Cache (by setting CAEN = 1), use the global control register (GCR) to select from the different cache options.

Note that not all functions described in the GCR are supported on OMAP5912 and OMAP5910. For example, the I-Cache supports the 2-way option for the N-way cache and zero, one, or two RAM sets. The following bits must be set as specified:

- CUT_CLOCK = 1
- AUTO_GATING = 1
- FLUSH_LINE = 0; line flushing is not supported, instead the entire cache must be flushed as a whole.
- GLOBAL_FLUSH = 1; flushing individual portions of the cache is not supported.
- WAY_NUMR = X1b; only 2-way cache is supported.
- GLOBAL_ENABLE = 1; enabling individual portions of the cache separately is not supported, instead the entire cache must be enabled as a whole.

Figure 12. I-Cache Global Control Register (GCR)

15	14	13	12	11	10	9	8
CUT_ CLOCK	AUTO_ GATING	Reserved	FLUSH_ LINE	GLOBAL_ FLUSH	HLFRAM-SET_ PRESENCE	WAY_ PRESENCE	HLFRAM-SET_ NUMR
RW-1	RW-1	RW-0	RW-0	RW-0	RW-0	RW-0	RW-00

7	5	4	3	2	1	0
HLFRAMSET_ NUMR		WAY_NUMR		STREAM-ING	RAM_FILL_ MODE	GLOBAL_ ENABLE
RW-00		RW-00		RW-1	RW-1	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined

Table 6. I-Cache Global Control Register (GCR) Bits Field Descriptions

Bits	Field	Value	Description
15	CUT_CLOCK		This bit determines whether the I-Cache module clock is disabled or enabled when the I-Cache is disabled.
		0	Disabled.
		1	Enabled.
14	AUTO_GATING		Enables automatic clock gating
		0	Disabled.
		1	Enabled.
13	Reserved		This reserved bit must be kept as 0.
12	FLUSH_LINE		Setting this bit flushes the lines specified by the flush line registers.
		0	No flush.
		1	Flush the specified line. Once the line flush occurs, the flush line bit is automatically cleared by the I-Cache.
11	GLOBAL_FLUSH		Setting the CACLR bit of the DSP core ST3_55 register begins a flush process within the I-Cache. The N-way cache and the two RAM set blocks contain a flush bit in their control registers, NWCR and RCR1/2, respectively. The GLOBAL_FLUSH bit determines whether the local flush bits are taken into consideration when CACLR is set.
		0	The N-way cache and the RAM set blocks are flushed when CACLR is set only if their local flush bits are set.
		1	The entire cache is flushed when CACLR is set; the local flush bits are ignored.
10	HLFRAMSET_ PRESENCE		This bit is used to enable the RAM set blocks. The number of RAM set blocks that are enabled is specified through the HLFRAMSET_NUMR bits.
		0	RAM set blocks are disabled.
		1	RAM sets blocks are enabled.
9	WAY_PRESENCE		This bit is used to enable the N-way cache block. The number of ways is specified in through the WAY_NUMR bits.
		0	N-way cache block is disabled.
		1	N-way cache block is enabled.

Table 6. I-Cache Global Control Register (GCR) Bits Field Descriptions (Continued)

Bits	Field	Value	Description
8–5	HLFRAMSET_NUMR		Specifies the number of RAM set blocks to enable when HLFRAMSET_PRESENCE is set.
		0000b	Enable only RAM set block 1.
		xxx1b	Enable both RAM set block1 and 2.
4–3	WAY_NUMR		Sets the number of ways active in the N-way cache block.
		x0b	Set N-way cache as 1-way (direct-mapped).
		x1b	Set N-way cache as 2-way (set-associative).
2	STREAMING		The principle of streaming is used in order to reduce the miss penalty: when a read miss occurs, a line load from external memory is started, and as soon as the requested word of the line arrives, it is sent to the DSP core. In this manner, the DSP core can continue its execution before the entire line is loaded. This bit must always be set.
		0	Disabled.
		1	Enabled. You must always set this bit.
1	RAM_FILL_MODE		
		0	This bit must always be set.
		1	Always set this bit to 1.
0	GLOBAL_ENABLE		Setting the CAEN bit of the DSP core ST3_55 register enables the I-Cache. The N-way cache and the two RAM set blocks contain a local enable bit in their control registers, NWCR and RCR1/2, respectively. The GLOBAL_ENABLE bit determines whether the local enable bits are taken into consideration when CAEN is set.
		0	The N-way cache and the RAM set blocks are enabled when CAEN is set only if their local enable bits are set.
		1	The entire cache is enabled when CAEN is set; the local enable bits are ignored.

4.6.3 I-Cache Line Flush Registers (FLR0, FLR1)

The I-Cache line flush registers are used to specify the address to be flushed from the cache.

Note:

These registers are not used, as line flushing is not supported on OMAP5910 and OMAP5912.

Figure 13. I-Cache Line Flush Registers (FLR0, FLR1)

FLR0

15	0
LINE_ADDRS_LOWER	
RW-0	

FLR1

15	8	7	0
Reserved		LINE_ADDRS_UPPER	
R-0		RW-0	

Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 7. I-Cache Line Flush Register 0 (FLR0) Field Descriptions

Bits	Field	Value	Description
15–0	LINE_ADDRS_LOWER	0000h–FFFFh	Lower address bits of the line to be flushed.

Table 8. I-Cache Line Flush Register 1 (FLR1) Field Descriptions

Bits	Field	Value	Description
15–8	Reserved		These bits are not used.
7–0	LINE_ADDRS_UPPER	00h–FFh	Upper address bits of the line to be flushed.

4.6.4 I-Cache N-Way Control Register (NWCR)

The N-way control register (NWCR) controls certain features of the N-way cache. You must configure this register before enabling the I-Cache through CAEN.

The size of each way in the N-way cache must always be set to 8KB for all devices.

The local flush and enable capabilities of the N-way cache are not supported on OMAP5912 and OMAP5910. Always use the following configuration for the N-way Control Register:

- ❏ FLUSH = 1; the N-way cache is always flushed when CACLR is set.
- ❏ ENABLE = 1; the N-way cache is always enabled when CACLR is set.

Any other setting for these bits is not supported.

Figure 14. I-Cache N-Way Control Register (NWCR)

Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 9. I-Cache N-way Control Register (NWCR) Field Descriptions

Bits	Field	Value	Description
15–5	Reserved		These bits are not used.
4–2	WAY_SIZE		These bits set the size of each way in the N-way cache. The size must always be set to 8Kbytes.
		011b	Each way size is set to 8Kbytes.
1	FLUSH		This bit determines whether the N-way cache is flushed when the CACLR bit of the DSP core ST3_55 register is set. These bit is ignored (N-way cache is always flushed) when GLOBAL_FLUSH is set.
		0	The N-way cache is not flushed when CACLR is set.
		1	The N-way cache is flushed when the CACLR is set.
0	ENABLE		This bit determines whether the N-way cache is enabled when the CAEN bit of the DSP core ST3_55 register is set. This bit is ignored (N-way cache is always enabled) when GLOBAL_ENABLE is set.
		0	The N-way cache is not enabled when CACLR is set.
		1	The N-way cache is enabled when the CACLR is set.

4.6.5 I-Cache RAM Set Control Registers (RCR1 and RCR2)

Each RAM set control register contains two initialization fields and a tag-valid bit.

- Initialization fields (FLUSH and ENABLE). If you have selected one RAM set with the global control register, you must initialize the logic for RAM set 1 before you enable the I-Cache. If you have selected two RAM sets with the global control register, you must also initialize the logic for RAM set 2. To perform the initialization for each RAM set, write the appropriate value (000Fh) to its RAM set control register before enabling the I-Cache.

- Tag-valid bit (TAG_VALID). When the I-Cache completes the process of filling a RAM set, the I-Cache sets TAG_VALID in that RAM set's control register. You can poll this bit to determine when the RAM set is ready.

Note:

On OMAP5910 and OMAP5912, you must always set FLUSH and ENABLE in RCR1 and RCR2.

Figure 15. I-Cache RAM Set Control Registers (RCR1 and RCR2)

RCR1

15	14 – 2	1	0
TAG_VALID	Reserved	FLUSH	ENABLE
R-0	R-0x3	RW-0	RW-1

RCR2

15	14 – 2	1	0
TAG_VALID	Reserved	FLUSH	ENABLE
R-0	R-0x3	RW-0	RW-1

Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 10. I-Cache RAM Set 1 Control Register (RCR1) and RAM Set 2 Control Register (RCR2) Field Descriptions

Bits	Field	Value	Description
15	TAG_VALID		RAM set tag-valid bit. Check this bit to determine when the I-Cache has completed the process of filling the RAM set.
		0	The fill is not started or is not complete.
		1	The fill is complete.
14–2	Reserved		These read-only bits are not used.
1	FLUSH		This bit determines whether the RAM set is flushed when the CACLR bit of the DSP core ST3_55 register is set. These bit is ignored (RAM set is always flushed) when GLOBAL_FLUSH is set.
		0	The RAM set is not flushed when CACLR is set.
		1	The RAM set is flushed when the CACLR is set.
0	ENABLE		This bit determines whether the RAM set is enabled when the CAEN bit of the DSP core ST3_55 register is set. This bit is ignored (RAM set is always enabled) when GLOBAL_ENABLE is set.
		0	The RAM set is not enabled when CACLR is set.
		1	The RAM set is enabled when the CACLR is set.

4.6.6 I-Cache RAM Set Tag Registers (RTR1 and RTR2)

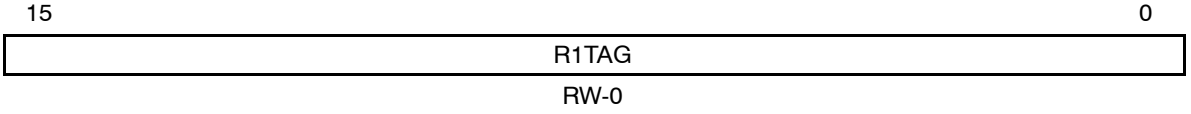
For each active RAM set (selected with the global control register), you must give the I-Cache a 12-bit tag that defines the range of addresses assigned to that RAM set. Load the tag into the appropriate RAM set tag register. Write a value with zeros in bits 15-12 and the tag in bits 11-0.

Note:

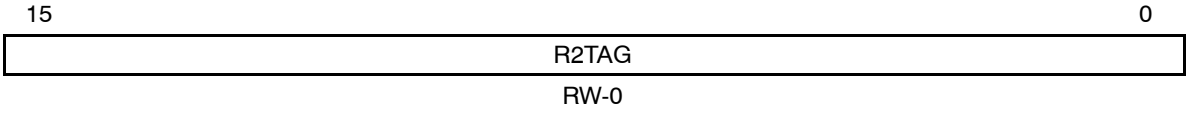
Do not set the RTR1 and RTR2 registers to the same value.

Figure 16. I-Cache RAM Set Tag Registers (RTR1 and RTR2)

RTR1



RTR2



Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 11. I-Cache RAM Set 1 Tag Register (RTR1) Field Descriptions

Bits	Field	Value	Description
15–0	R1TAG	0000h–0FFFh	RAM set 1 tag bits. Write a value with zeros in bits 15-12 and the tag in bits 11-0. This register is only applicable if you have selected one or two RAM sets with the global control register.

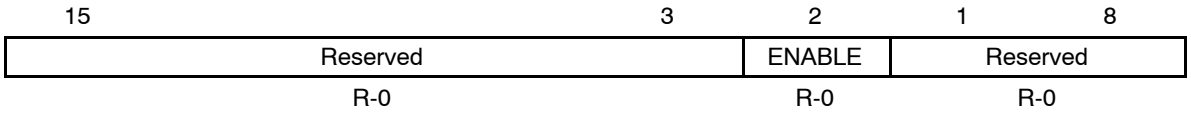
Table 12. I-Cache RAM Set 2 Tag Register (RTR2) Field Descriptions

Bits	Field	Value	Description
15–0	R2TAG	0000h–0FFFh	RAM set 2 tag bits. Write a value with zeros in bits 15-12 and the tag in bits 11-0. This register is only applicable if you have selected one or two RAM sets with the global control register.

4.6.7 I-Cache Status Register (ISR)

The status register contains the ENABLE bit that indicates when the I-Cache is enabled. When you send an enable request to the I-Cache (CAEN = 1 in the DSP core status register ST3_55), poll for ENABLE = 1 before writing to either of the RAM set tag registers.

Figure 17. I-Cache Status Register (ISR)



Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 13. I-Cache Status Register (ISR) Field Descriptions

Bits	Field	Value	Description
15–3	Reserved		These read-only bits are not used.
2	ENABLE		I-Cache-enabled bit. When you send an enable request to the I-Cache, poll for ENABLE = 1 before writing to either of the RAM set tag registers.
		0	The I-Cache is disabled.
		1	The I-Cache is enabled.
1–0	Reserved		These bits are not used.

5 DSP External Memory Interface

5.1 Overview

The external memory interface (EMIF) gives the DSP core and the DSP DMA controller access to the shared system memory managed by the traffic controller. The EMIF interfaces directly to a 32-bit-wide system bus. This bus can operate at the DSP subsystem clock rate with sustained throughput during burst accesses.

Note:

Internally, 8-bit data read requests from DSP external memory are converted to 16-bit data read requests by the EMIF. The appropriate byte is fetched from this read request and placed in internal memory.

The relationship of the DSP EMIF to other DSP subsystem modules can be seen from the system block diagrams in section 1.4.

5.2 Peripheral Architecture

5.2.1 Clock Control

The EMIF is clocked by the DSP subsystem clock DSP_CK (see section 12.2 for more details).

5.2.2 Memory Map

The EMIF controls accesses to DSP subsystem external memory. Section 3.4 details the memory map of the DSP subsystem.

5.2.3 DSP External Memory Accesses

Four major steps are taken when the DSP subsystem accesses DSP external memory.

1) The DSP core or the DSP DMA requests an access to DSP external memory.

2) The DSP EMIF receives that request and forwards it to the DSP MMU.

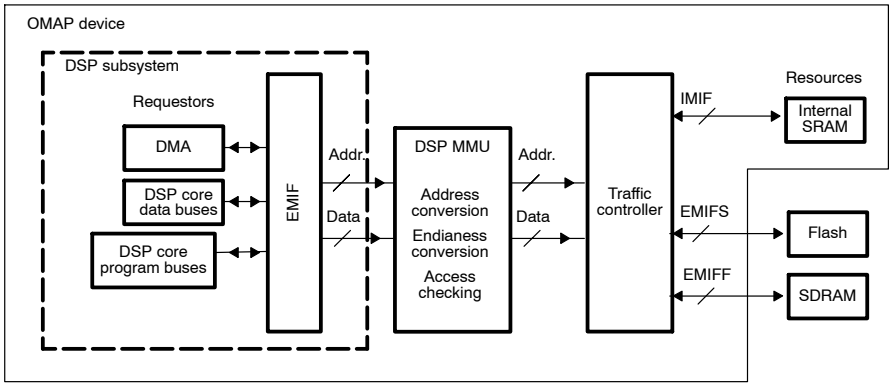
3) The MMU checks its translation look-aside buffer (TLB, section 6.2.2) for a match on the virtual address tag. If there is a TLB hit and the correct access permissions for the type of access (read or write) are found, the MMU translates the virtual address from the EMIF into a physical address and forwards the request to the traffic controller with the appropriate endianess conversion.

   If the virtual address tag is not found, the MMU uses its table walking logic to fetch the translation from translation tables and updates the TLB. If correct access permissions are found, the MMU carries out the virtual-to-physical address translation and forwards the request to the traffic controller. If the correct access permissions are not found, MMU generates an interrupt to the MPU core and stalls the DSP EMIF until the error is cleared. When the MPU core clears this error, the DSP MMU repeats this entire step.

4) The traffic controller accesses the actual OMAP resource.

Figure 18 shows the major blocks involved during an access to DSP external memory by the DSP subsystem.

Figure 18. DSP Subsystem External Memory Connections



5.2.4 EMIF Requests

The EMIF services the requests shown in Table 14. If multiple requests arrive simultaneously, the EMIF prioritizes them as shown in the Priority column.

Table 14. EMIF Requests and Their Priorities

EMIF Requester	Priority	Description
E bus	1 (highest)	A write request from the E bus of the DSP core.
F bus	2	A write request from the F bus of the DSP core.
D bus	3	A read request from the D bus of the DSP core.
C bus	4	A read request from the C bus of the DSP core.
P bus	5	An instruction fetch request from the DSP core or instruction cache. In the core, instructions are received on the P bus.
DMA controller	6	A write or read request from the DSP DMA controller.

As shown in Table 15, there is a subtle difference between dual data accesses and long data accesses requested by the DSP core. The following two instructions are examples of these access types:

```
ADD *AR0, *AR1, AC0    ; Dual data access. Two separate
                       ; 16-bit values referenced by
                       ; pointers AR0 and AR1.
ADD dbl(*AR2), AC1     ; Long data access. One 32-bit
                       ; value referenced by pointer AR2.
```

Both access types require two 16-bit data buses in the DSP core, but they require different numbers of EMIF requests. A dual data access involves two separate 16-bit values and, therefore, requires two EMIF requests. A long data access involves a single 32-bit value and, therefore, a single EMIF request. This EMIF request corresponds to the address bus used. For example, if a long data read is performed, the DAB address bus is used, and the EMIF receives a D-bus request.

Table 15. EMIF Requests Associated with Dual and Long Data Accesses

Access Type	DSP Core Data Buses Used	DSP Core Address Bus(es) Used	Request(s) Sent To EMIF
Dual data read	CB and DB (carrying two 16-bit values)	CAB and DAB	C-bus request to read 16 bits D-bus request to read 16 bits
Dual data write	EB and FB (carrying two 16-bit values)	EAB and FAB	E-bus request to write 16 bits F-bus request to write 16 bits
Long data read	CB and DB (carrying one 32-bit value)	DAB	D-bus request to read 32 bits
Long data write	EB and FB (carrying one 32-bit value)	EAB	E-bus request to write 32 bits

5.2.5 Write Posting: Buffering Write to DSP External Memory

Typically, when a DSP core write request arrives at the EMIF, the EMIF does not send acknowledgment to the DSP core until the EMIF has driven the data on the external bus. As a result, the DSP core does not begin the next operation until the data is actually sent to the DSP external memory.

If write posting is enabled, the EMIF acknowledges the DSP core as soon as the EMIF receives the address and data. The address and data are stored in dedicated write posting registers in the EMIF. When a time slot becomes available, the EMIF runs the posted write operation. If the next DSP core access is not for the EMIF and is for internal memory, that access is able to run concurrently with the posted write operation.

The EMIF supports two levels of write posting. That is, the write posting registers can hold data and addresses for up to two DSP core accesses at a time. The EMIF allocates the write posting registers on a first requested, first served basis. However, if the E bus and the F bus make requests simultaneously, the E bus is given priority.

To enable write posting for all accesses to DSP external memory, set the WPE bit in the EMIF global control register. It might be useful to disable write posting (WPE = 0) during debugging.

There are no write posting registers for requests from the DMA controller. However, the EMIF sends acknowledgement to the DSP DMA controller prior to the actual write to DSP external memory. This early acknowledgement allows the DMA controller to transfer the next address early, avoiding dead cycles during burst transfers or between back-to-back single transfers.

5.2.6 Reset Considerations

The EMIF registers can be reset by hardware and software resets. Section 5.3 details the contents of the EMIF configuration registers after reset.

5.2.6.1 Effect of Hardware Reset

The EMIF configuration registers are always reset by an OMAP hardware reset. Section 12.1 describes OMAP hardware resets.

5.2.6.2 Effect of Software Reset

The DSP_RST bit of the ARM_RSTCT1 register controls whether the priority registers of the TIPB module, the EMIF configuration registers, and the MPUI control logic (partially) in the DSP subsystem are reset when the DSP_EN bit (also in ARM_RSTCT1) is cleared. See your device-specific data manual for more information on ARM_RSTCT1. Clearing the DSP_EN bit always resets the DSP subsystem. When DSP_RST = 0, clearing the DSP_EN bit resets the DSP subsystem and also the priority registers, the EMIF configuration registers, and the MPUI control logic. If DSP_RST = 1, the registers are not reset.

The DSP_RST bit of the ARM_RST1 register must be set before the DSP subsystem is taken out of reset.

5.2.7 Power Management

If you want to temporarily turn off the clock to the EMIF module to reduce power, you can place its domain in idle mode:

1) Select the idle mode for the EMIF domain by making EMIFI = 1 in the idle configuration register (ICR) of the DSP subsystem (see section 12.3.2.8).

2) Execute the IDLE instruction in the DSP core.

External memory requests should not be made when the EMIF is in its idle mode.

To wake the EMIF from its idle mode:

1) Deselect the idle mode by making EMIFI = 0 in ICR.

2) Execute the IDLE instruction.

5.3 EMIF Registers

5.3.1 Overview

Control of the EMIF is maintained through a set of registers within the EMIF. These registers are accessible only at addresses in the I/O memory space of the DSP subsystem.

Table 16. Summary of the EMIF Registers

Name	Description	DSP I/O Address†	See Section
GCR	Global control register. Use this register to enable or disable write-posting.	0x0800	5.3.2
GRR	Global reset register. Use this register to reset the EMIF state machine.	0x0801	5.3.3

† DSP I/O addresses apply to both OMAP5910 and OMAP5912.

5.3.2 EMIF Global Control Register (GCR)

The EMIF Global Control Register is used to enable or disable write-posting.

Figure 19. EMIF Global Control Register (GCR)

Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

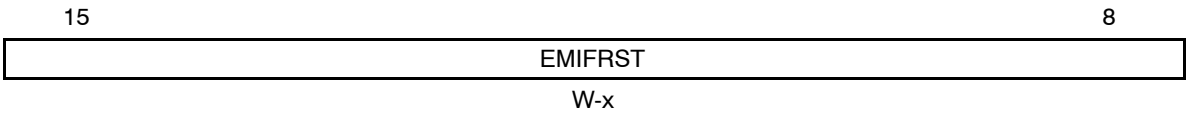
Table 17. EMIF Global Control Register (GCR) Field Descriptions

Bits	Field	Value	Description
15–8	Reserved		These bits are not used. Writable bits should be kept as 0 during writes to this register.
7	WPE		Write posting enable bit. Use WPE to enable or disable the write posting feature of the EMIF. WPE affects all accesses to DSP external memory.
		0	Disabled.
		1	Enabled.
6–0	Reserved		These bits are not used. Writable bits should be kept as 0 during writes to this register.

5.3.3 EMIF Global Reset Register (GRR)

The EMIF Global Reset Register is used to reset the EMIF state machine.

Figure 20. EMIF Global Reset Register (GRR)



Note: R = Read, W = Write; –n = Value after reset;, –x = Value after reset is not defined

Table 18. EMIF Global Reset Register (GRR) Field Descriptions

Bits	Field	Value	Description
15–0	EMIFRST		Any write to this register resets the EMIF state machine.

6 DSP Memory Management Unit

6.1 Overview

DSP core and DSP DMA accesses to DSP external memory are handled by the DSP external memory interface (EMIF) in conjunction with the DSP Memory Management Unit (MMU). The DSP MMU maps external memory requests to the OMAP physical address space. The MMU also provides fault and permission checking, and performs endianess conversion. It is configured by the MPU core. Section 10 describes MMU endianess.

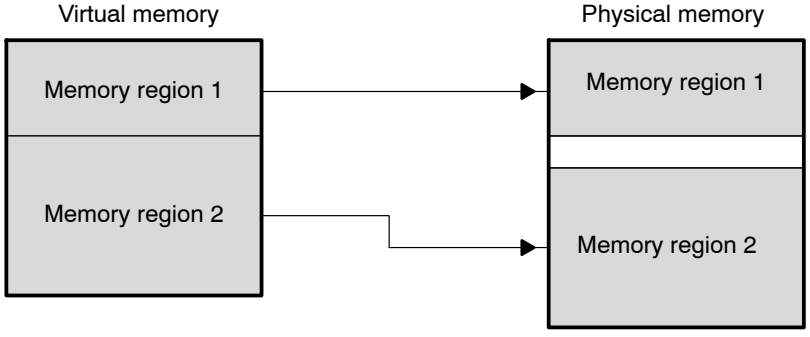
6.1.1 Purpose of the MMU

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.
- Task protection: Illegal, non-allowed accesses to memory locations can be detected and prevented.

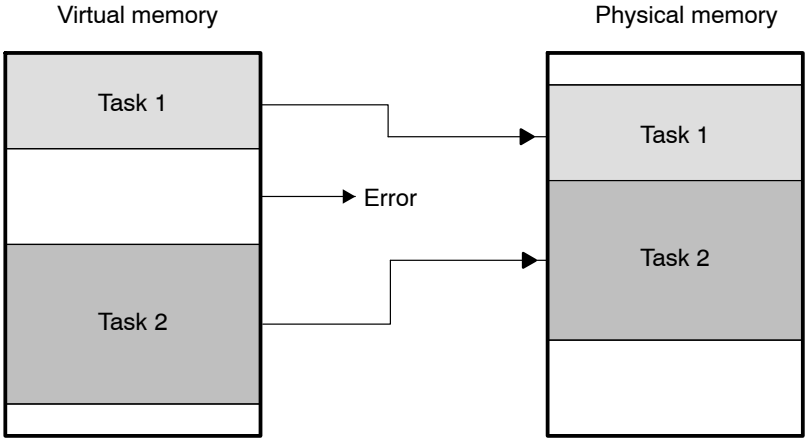
Figure 21 and Figure 22 illustrate the benefits of using an MMU.

Figure 21. Memory Defragmentation



In Figure 21, memory region 1 and memory region 2 are fragmented in physical memory. Using the MMU, they can be translated to appear as one contiguous memory region in the virtual memory space.

Figure 22. Task Protection



In Figure 22, task 1 and task 2 are located adjacent in physical memory. In systems without an MMU, there is a danger that task 1 will accidentally write into the memory area allocated to task 2, and vice versa. Using an MMU, unmapped memory regions can be placed between tasks. Therefore, the MMU can easily detect any erroneous accesses to unmapped memory regions in the virtual address space.

6.1.2 Features

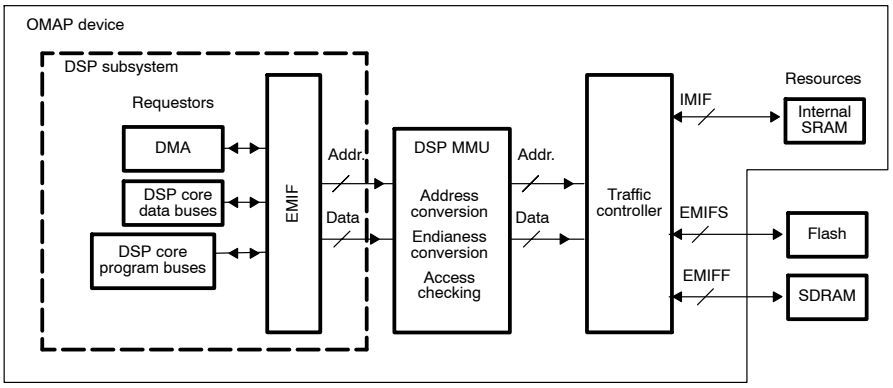
The DSP MMU in OMAP5910 and OMAP5912 devices includes the following features:

- A translation look-aside buffer (TLB), which stores recently-used translations. The TLB acts like a cache of recently read translation table entries. Translations can also be manually written to the TLB by the MPU core.
- Table walking logic, which automatically retrieves a translation from a set of translation tables and updates the TLB.

6.1.3 Functional Block Diagram

Figure 23 shows the role of the DSP MMU within the DSP subsystem memory structure.

Figure 23. DSP Subsystem Memory Interface



6.1.4 Supported Usage of the DSP MMU

There are two ways to use the MMU:

- The contents of the TLB can be written manually by the MPU core.

  Using this approach does not require any translation tables. However, the MPU core has to update the TLB when no valid address translation is found (TLB miss).

- The MMU table walking logic can be enabled to automatically update the TLB by reading a structure of translation tables.

  The translation table structure has to be set up by the MPU core before the MMU is enabled. However, no action from the MPU core is required on a TLB miss.

You can also combine these two options. For instance, the MPU core can set up time-critical translations in the TLB and other non-time-critical address translations in translation tables, which the table walking logic reads later. The DSP MMU can also be disabled, in which case all DSP subsystem external memory requests would be mapped to the first 16M-bytes of OMAP system memory (CS0).

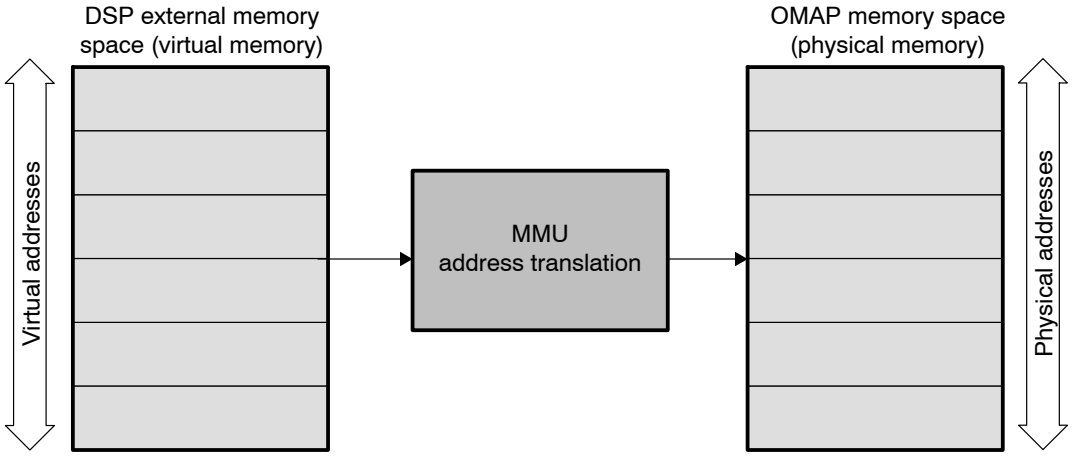
Sections 6.3 and section 6.4 give more detail on using one of these two supported usage options.

6.2 MMU Architecture

6.2.1 Summary of Address Translation Process

As shown in Figure 24, the MMU translates virtual addresses generated by the DSP EMIF to physical addresses. These physical addresses are used to access the actual OMAP resource.

Figure 24. MMU Address Translation



Whenever an address translation is requested (that is, for every memory access with the DSP MMU enabled), the DSP MMU checks first to see whether the TLB contains the requested translation. The TLB acts like a cache, storing recent translations.

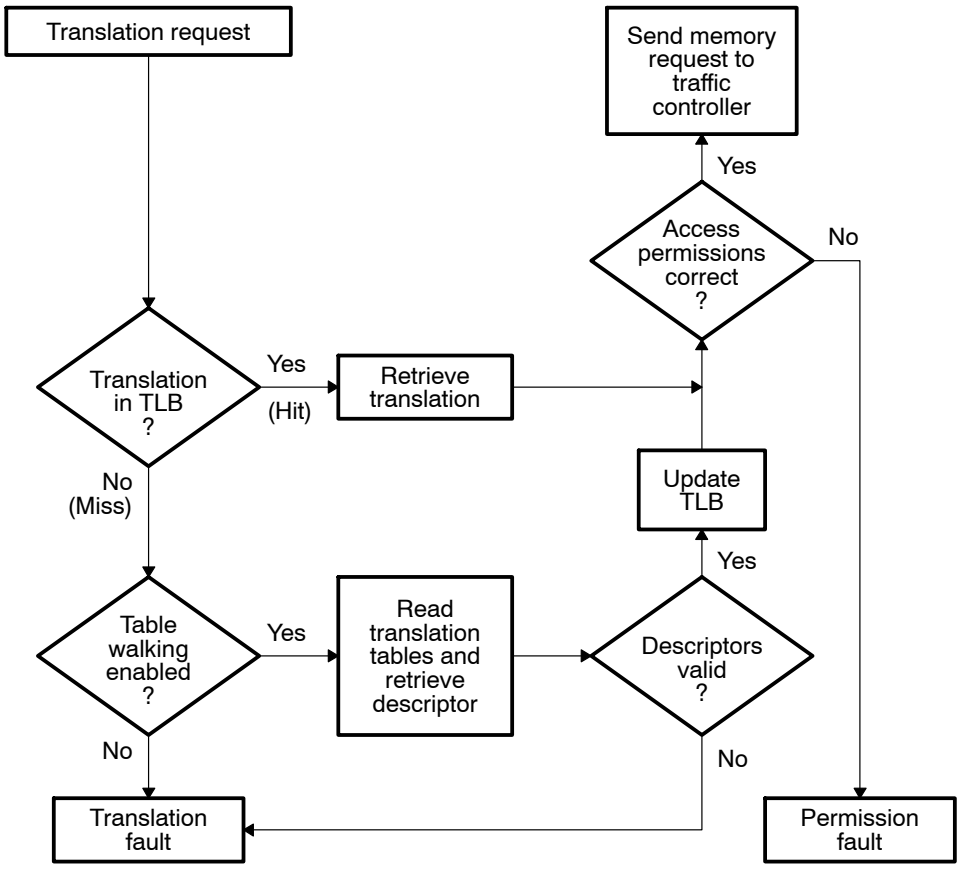
If the translation is contained in the TLB and the access permissions are correct, the corresponding physical address is calculated and the memory request is forwarded to the traffic controller. If the memory request lacks the correct access permissions, the MMU generates a fault interrupt to the MPU core.

When the requested translation is not in the TLB, the table walking logic (if enabled) retrieves the translation by reading a set of translation tables. If the table walking logic is disabled, the MMU generates a fault interrupt to the MPU core.

When the table walking logic finds a valid translation, it updates the TLB and, if the access permissions are correct, the corresponding physical address is calculated and the memory request is sent to the traffic controller. If the request does not have the correct permissions, or if no valid translation is found in the translation tables, then the MMU generates a fault interrupt to the MPU core.

Figure 25 summarizes the entire DSP MMU translation process.

Figure 25. MMU Translation Process



6.2.2 Translation Look-Aside Buffer (TLB)

To increase the virtual-to-physical address translation process speed, a cache mechanism (the TLB) is 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 the translation is in the TLB (a TLB hit), then this translation is used. If the address translation is not in the TLB (a TLB miss), the table walking logic (described in section 6.2.3) retrieves the address translation from the translation tables and updates the TLB. If the table walking logic is disabled, a translation fault is generated and the MPU core is interrupted.

Entries in the TLB are replaced, or evicted, by the table walking logic when the TLB is full. The table walking logic selects the entry to be replaced at random.

Entries in the TLB can be protected, or locked, against being overwritten if necessary. 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. Section 6.2.2.4 describes the locking process, while section 6.2.2.2 describes the process for writing entries into the TLB.

When time-critical program routines are used, it is preferable to avoid the performance impact of retrieving the translations via table walking logic by locking TLB entries.

The MPU core can manually write address translations to the TLB. Alternatively, table walking logic can be used to automatically carry out the address translation (using the translation tables) and update the TLB.

The TLB entries can be read to determine the currently buffered translations (section 6.2.2.5). Unused translations can be deleted (section 6.2.2.6).

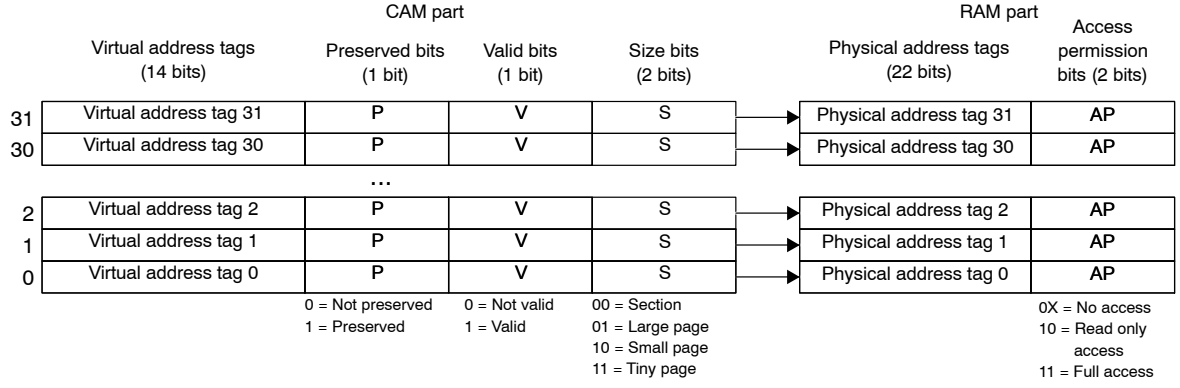
6.2.2.1 TLB Entry Format

TLB entries consist of two parts:

- CAM. Contains a 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 memory block size (section, large page, small page, or tiny page) and the preserved and valid flags.
- RAM. Contains the address translation that belongs to the virtual address tag. It also contains the access permissions (no access, read-only access, and full access).

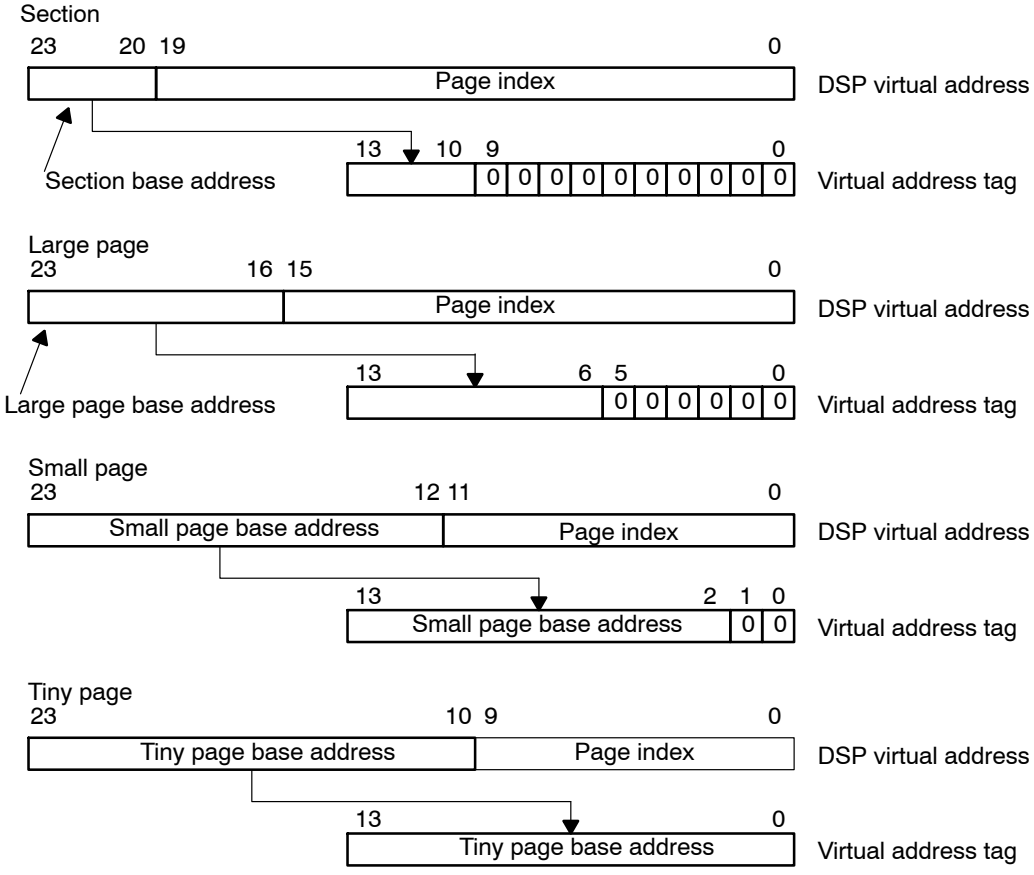
The TLB entry structure is shown in Figure 26.

Figure 26. TLB Entry Structure



The virtual address tag is a 14-bit field derived from the virtual address of the memory request being processed. Not all the bits in the virtual address tag are needed for translation. Instead, the size of the memory block described by the entry determines the number of bits used. For example, only bits 13:10 of the virtual address tag are used for a section. When writing entries to the TLB, unused bits in the virtual address tags must always be kept as zeros. The read value of unused bits is not predictable. Figure 27 shows how to determine the virtual address tag from the DSP virtual address. Note that a section corresponds to 1 Mbytes of memory, a large page corresponds to 64 Kbytes of memory, a small page corresponds to 4 Kbytes of memory, and a tiny page corresponds to 1 Kbyte of memory.

Figure 27. Determining Virtual Address Tags for TLB CAM Entries



The valid parameter of the TLB entry value specifies whether an entry is valid. The table walking logic can overwrite non-valid entries. The table walking logic first attempts to fill all non-protected, non-valid entries before replacing valid entries.

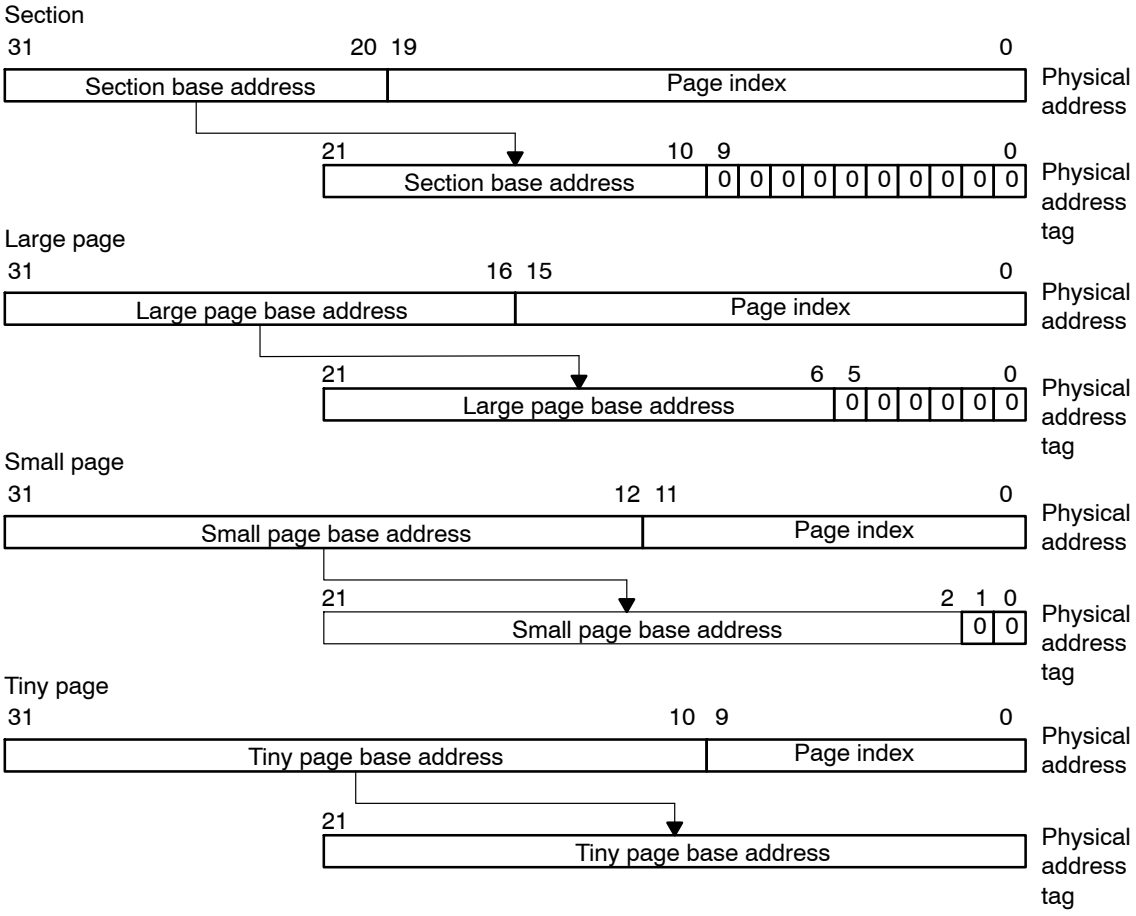
The preserved parameter of the TLB entry value determines the behavior of an entry in the event of a TLB flush. If an entry is preserved, it is not deleted upon a TLB global flush. Section 6.2.2.6 describes the TLB flushing mechanism.

The size bits determine the range of memory addresses to which the TLB entry corresponds. All addresses that fall within the same range will have the same section or base address. For example, external memory addresses between virtual address 0x10 0000 – 0x1F FFFF will have the same section base address (0x1).

The physical address tag of the RAM value is a 22-bit field which is used in the virtual-to-physical address translation, as described in section 6.2.2.2. The physical address tag is derived from the physical address corresponding to the virtual address. Note that, like the virtual address tag, not all the bits in the physical address tag are used. When writing RAM entries to the TLB, unused bits in the physical address tags must always be kept as zeros. Figure 28 shows how to determine the physical address tag from the physical address.

The access permission bits of the RAM value define the type of access that is permitted to the physical memory range described by the TLB entry. The memory range is specified by the physical address tag and the size bits. A forbidden access to the physical memory will cause the MMU to generate a permission fault and an interrupt to the MPU core.

Figure 28. Determining Physical Address Tags for TLB RAM Entries



6.2.2.2 TLB Address Translation Process

When an external memory request is generated by the DSP EMIF, the DSP MMU first checks the contents of the TLB to determine whether a corresponding address translation is present. To determine if the address translation is in the TLB, the MMU performs these steps:

1) Generates a virtual address tag by taking the 14 most-significant address bits of the virtual address (bits 23:10).

2) Compares the virtual address tag to the tags contained in valid TLB entries

   Note that although the number of bits needed for an address translation depends on the size of the memory block described by the entry, the entire contents of the virtual address tags are compared. Therefore, as described in section 6.2.2.1, it is important to keep unneeded bits as zero when writing entries in the TLB.

3) Reads the corresponding physical address tag from the TLB entry.

4) Checks the access permission bits.

5) Generates a corresponding physical address by using the physical address tag and the page index (taken from the virtual address).

   The number of physical address tags and virtual address bits used in this step depends on the size field of the TLB entry. Figure 29 through Figure 32 illustrate how a physical address is generated from the physical address tag and the page index.

If the access permission bits do not allow the type of access being requested, the MMU generates a permission fault and interrupts the MPU core. An interrupt is also generated if no matching virtual address tag is found and the table walking logic is disabled (translation fault).

Figure 29. Physical Address Generation Using TLB Entry with Size = 00b (Section)

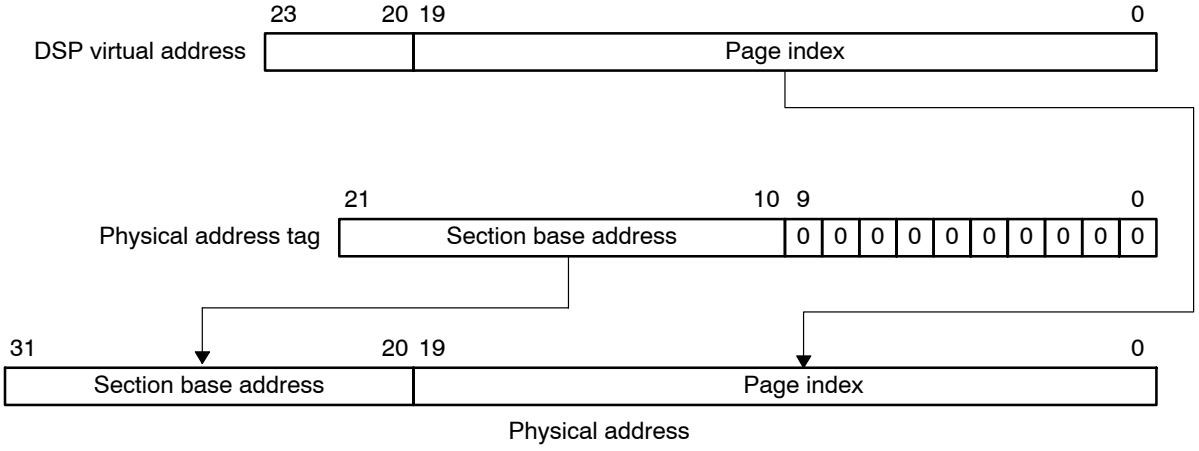


Figure 30. Physical Address Generation Using TLB Entry with Size = 01b (Large Page)

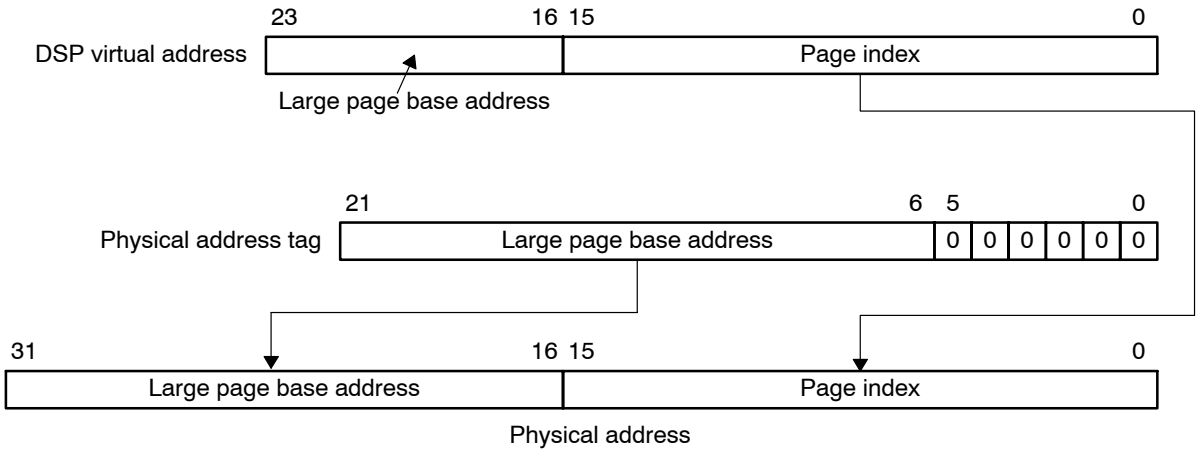


Figure 31. Physical Address Generation Using TLB Entry with Size = 10b (Small Page)

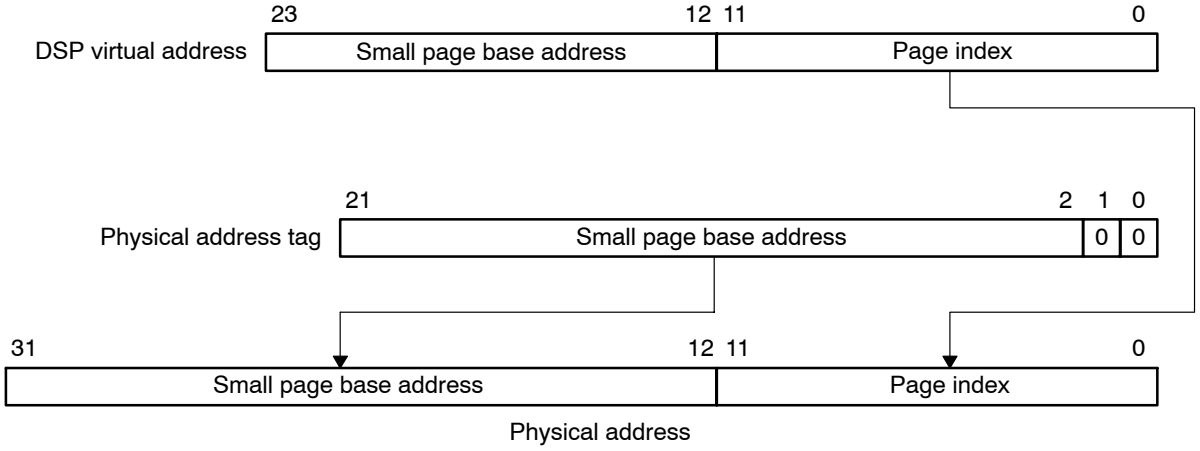
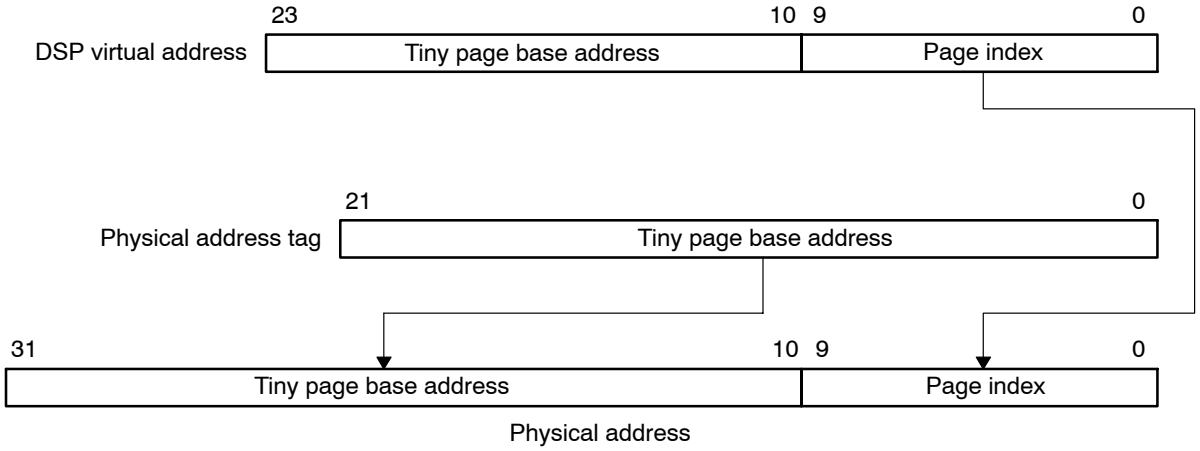


Figure 32. Physical Address Generation Using TLB Entry with Size = 11b (Tiny Page)



6.2.2.3 Writing Entries to the TLB

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

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.

A victim pointer identifies the next entry to be written. The same victim pointer can be used to select an entry to be read. The victim pointer is controlled through the Lock/Protect Entry Register (LOCK_REG). The Lock/Protect Entry Register can only be modified by the MPU core when the table walking logic is disabled.

To write an entry to the TLB, follow these steps:

1) Disable the table walking logic by clearing the TWL_EN bit in the Control Register (CNTL_REG).
2) Determine CAM and RAM parameters and write them into the CAM and RAM registers (CAM_H_REG, CAM_L_REG, RAM_H_REG, and RAM_L_REG).

3) Select the TLB entry to be written by setting the victim pointer through the Lock/Protect Entry Register (LOCK_REG). For example, to update entry 0 in the TLB, write 0 to the victim pointer field of LOCK_REG.

4) Set the WRITE_ENTRY bit in the Read/Write TLB Entry Register (LD_TLB_REG).

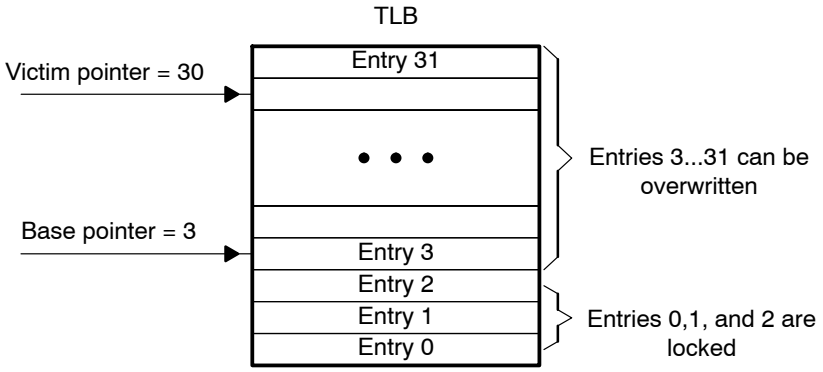
5) Enable the table walking logic by setting the TWL_EN bit in the Control Register (CNTL_REG). This step can be omitted if the table walking logic is not used.

Section 6.5 gives detailed descriptions of all TLB control registers.

6.2.2.4 Protecting TLB Entries

The first n TLB entries (with n < 32) can be protected, or locked, against being overwritten with new translations retrieved by the table walking logic. This is done by setting the TLB base pointer to n (see Figure 33). The remaining entries are overwritten, if necessary, on a random basis. The victim pointer indicates the next TLB entry to be read/written.

Figure 33. TLB Entry Lock Mechanism



Locking TLB entries ensures 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.

To protect the first n TLB entries, follow these steps:

1) Disable the table walking logic by clearing the TWL_EN bit in the Control Register (CNTL_REG).

2) Set the base pointer field in the Lock/Protect Entry Register (LOCK_REG) to n, and set the current victim pointer (also in the Lock/Protect Entry Register) to a value equal to or greater than n. For example, to protect

entries 0 through 10 of the TLB, write 11 to the base pointer field and load the victim pointer field with a value from 11 to 31.

3) Enable the table walking logic by setting the TWL_EN bit in the Control Register (CNTL_REG).

Locking entries in the TLB does not protect against a TLB global flush operation. Therefore, when locking entries in the TLB, it is recommended that all locked entries be written with their preserved bit set. Section 6.2.2.2 describes the process for writing entries into the TLB.

6.2.2.5 Reading TLB Entries

Entries in the TLB can be read by using the victim pointer to specify the entry number. The entry is read via the CAM/RAM read registers (READ_CAM_H_REG, READ_CAM_L_REG, READ_RAM_H_REG, and READ_RAM_L_REG).

To read an entry from the TLB, follow these steps:

1) Disable the table walking logic by clearing the TWL_EN bit in the Control Register (CNTL_REG).

2) Select the TLB entry to be read by setting the victim pointer through the Lock/Protect Entry Register (LOCK_REG). For example, to read entry 0, write 0 to the victim pointer field in the Lock/Protect Register.

3) Set the read TLB-entry bit in the Read/Write TLB Entry Register (LD_TLB_REG).

4) Read the CAM and RAM parameters from the CAM and RAM read registers (READ_CAM_H_REG, READ_CAM_L_REG, READ_RAM_H_REG, and READ_RAM_L_REG).

5) Enable the table walking logic by setting the TWL_EN bit in the Control Register (CNTL_REG). This step can be skipped if the table walking logic is not being used.

Section 6.5 summarizes the relevant TLB control registers.

6.2.2.6 Deleting TLB Entries

Two mechanisms exist to delete (flush) TLB entries. Invoking a TLB global flush deletes all unpreserved TLB entries (TLB entries that were written with the preserved bit as zero). The flush is invoked by setting the global flush bit in the TLB Global Flush Register (GFLUSH_REG).

An individual TLB entry can be flushed, regardless of its preserved bit setting, by selecting it using the victim pointer (LOCK_REG) and setting the flush entry

bit in the Flush Entry Register (FLUSH_ENTRY_REG). The valid and preserved bits of the TLB entry are cleared when the flush command is completed.

To flush individual entries from the TLB, follow these steps:

1) Disable the table walking logic by clearing the TWL_EN bit in the Control Register (CNTL_REG).

2) Select the TLB entry to be flushed by setting the victim pointer through the Lock/Protect Entry Register (LOCK_REG). For example, to flush entry 0, write 0 to the victim pointer field in the Lock/Protect Register.

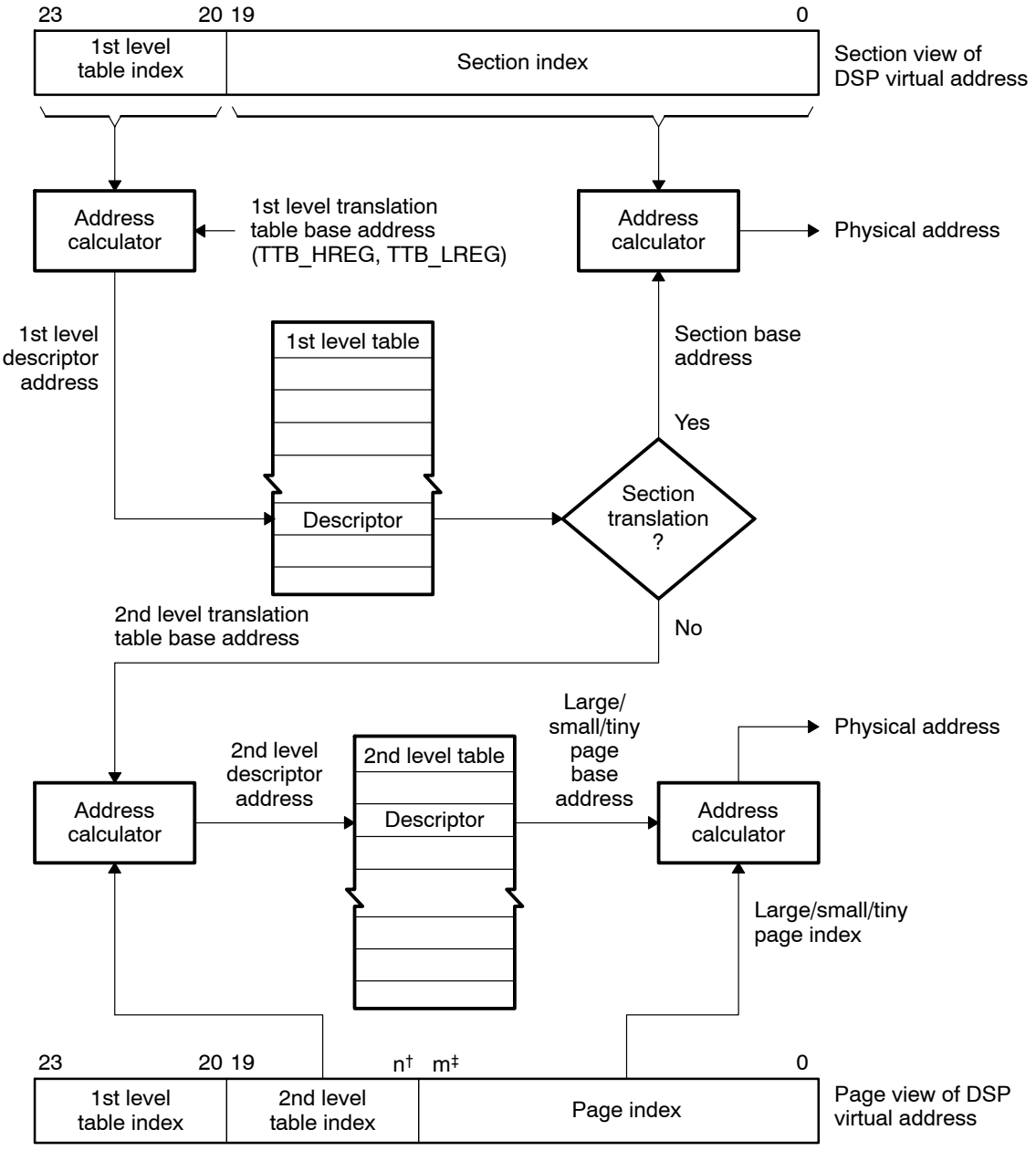
3) Set the flush entry bit in the Flush Entry Register (FLUSH_ENTRY_REG).

4) Enable the table walking logic by setting the TWL_EN bit in the Control Register (CNTL_REG). This step can be skipped if the table walking logic is not being used.

6.2.3 Table Walking Logic

When an address translation is not present in the TLB (a TLB miss), the table walking logic automatically carries out the address translation using the translation tables and then updates the TLB. Figure 34 is a flow diagram of the steps taken by the table walking logic to translate a virtual address to a physical address using the translation tables. Details on each step can be found in the indicated sections.

Figure 34. Physical Address Calculation



† The value of n depends on the type of table being accessed (see sections 6.2.6.5 and 6.2.6.6).
‡ The value of m depends on the type of page being accessed (see sections 6.2.6.2 through 6.2.6.4).

The table walking logic starts an address translation by accessing a descriptor from a first-level translation table (section 6.2.5). To determine the address of the descriptor, add a first-level table index (taken from the virtual address) and the base address of the first-level translation table (taken from the translation table base registers TTB_MSB_REG and TTB_LSB_REG). The first-level translation table divides the DSP memory space into 16 1MB-sections.

The contents of the first-level descriptor determine whether the section to which the virtual address corresponds is further divided into pages or if the section is directly linked to a physical memory section. In the latter case, the descriptor provides a section base address which is joined to a section index (taken from the virtual address) to generate a physical address.

When a section is further divided into pages, the first-level descriptor provides a base address for a second-level translation table (section 6.2.6). A descriptor is accessed from the second-level translation table to determine the page base address corresponding to the virtual address. Determine the base address of the descriptor by adding a second-level table index (taken from the virtual address) and the second-level translation table base address. Finally, the physical address is determined by adding a page base address provided by the descriptor and a page index taken from the virtual address.

After an address translation has been carried out, the table walking logic updates the selected TLB entry with the translation result. The victim pointer selects this TLB entry, then selects the next unlocked entry to be replaced.

The table walking logic is enabled through the Control Register (CNTL_REG). If the table walking logic is not enabled, an interrupt will be generated to the MPU core on every TLB miss, see section 6.2.7 for more details.

Note:

When the table walking logic is enabled, the TLB cannot be manually updated; you should not write to the LD_TLB_REG, TTB_H_REG, TTB_L_REG, and LOCK_REG.

The DSP core can force the table walking logic to perform an address translation pre-fetch before the occurrence of a TLB miss. For this, a pre-fetch register is visible in DSP I/O memory space. The DSP initiates a pre-fetch by writing the DSP virtual address tag to the pre-fetch register.

Note:

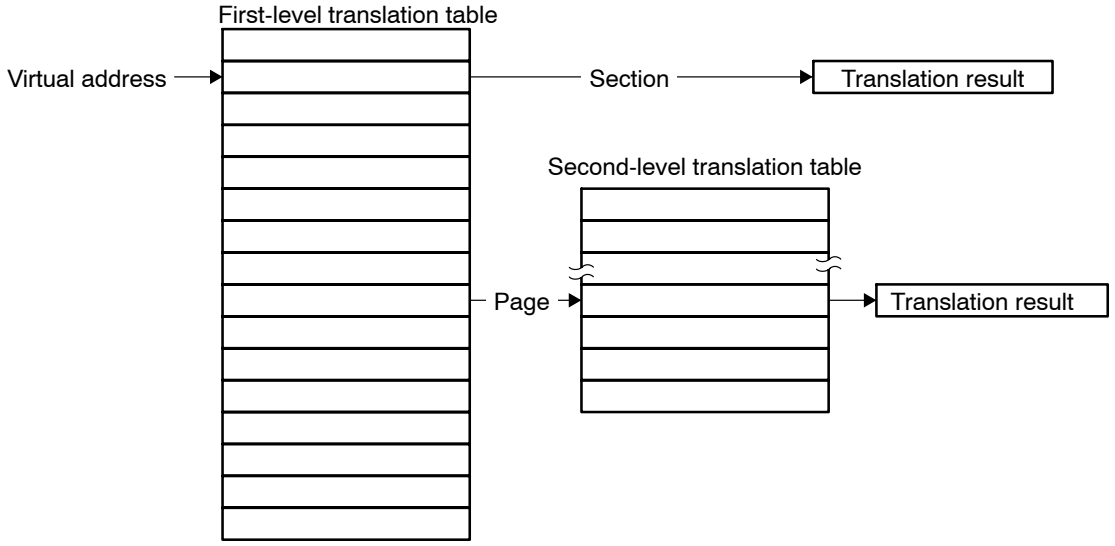
The table walking logic must be enabled to carry out the pre-fetch request.

6.2.4 Memory Address Translation

The table walking logic carries out address translations by accessing a first-level translation table and (if necessary) multiple second-level translation tables. Each translation table is made up of descriptors containing the information needed to map a range of virtual memory addresses to a corresponding range of physical memory addresses.

Figure 35 shows a sample translation table hierarchy.

Figure 35. Sample Translation Table Hierarchy



The first-level translation table divides the DSP virtual address space (16MB) into 16 sections (1MB per section). It contains first-level descriptors, each of which can specify one of two types of information:

- The translation information for a virtual memory section.

  The descriptor provides the base address for the 1MB physical memory section assigned to that virtual memory section. The table entry also specifies all access permission information.

- A pointer to a second-level translation table.

  Second-level translation tables are used when a translation granularity smaller than the size of one section is desired.

Two types of second-level tables can be used:

- Coarse page tables with 256 entries.

  Each entry in a coarse page table contains a descriptor which describes the translation information for either a large page (64KB) or a small page (4KB) of memory.

  Notice that 256 small pages is the equivalent of a section, yet 256 large pages is the equivalent of 16 sections. As described in section 6.2.6.5, the descriptor must be copied 16 times in the course page table when using a descriptor to map a large page.

- Fine page tables with 1024 entries.

  Each entry in a fine page table contains a descriptor which contains the translation information for either a large page (64KB), a small page (4KB), or a tiny page (1KB) of memory.

  As for coarse page tables, descriptors used to map large pages must be copied 64 times in the fine page table and descriptors used to map small pages must be copied 16 times. This requirement is described in section 6.2.6.6.

One of the most important parameters in developing a table-based address translation scheme is the memory page size, that is, the size of the memory region described by each translation table entry. Using large pages results in a smaller translation table, whereas using small pages greatly increases the efficiency of dynamic memory allocation and defragmentation. However, this small size also implies more complex (and larger) translation tables.

Sections 6.2.5 and 6.2.6 describe the structure of the first- and second-level translation tables, as well as the descriptors format.

6.2.5 First-Level Translation Table

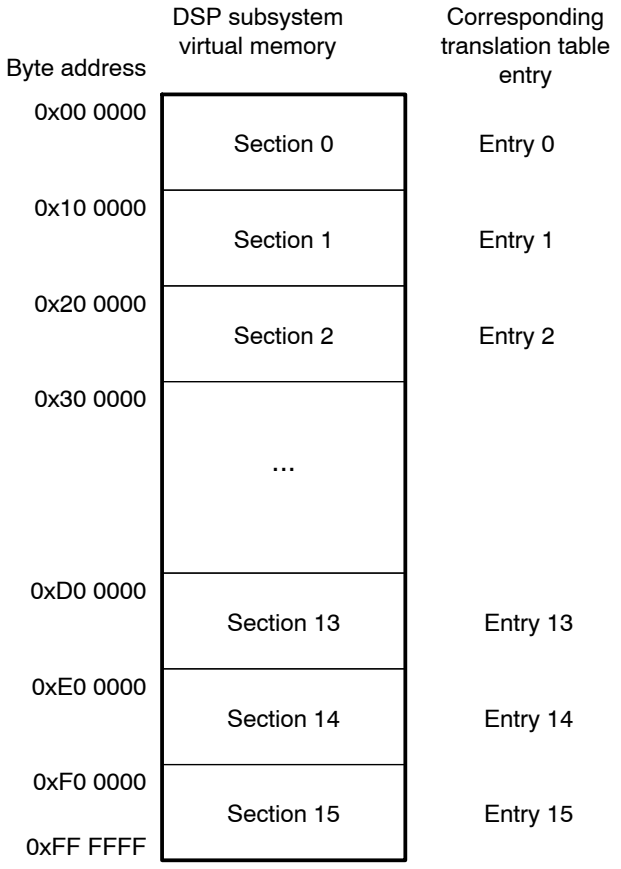
The first-level translation table describes the translation properties of the DSP subsystem virtual address space by dividing it into 1M-byte sections. Sixteen sections are needed to encompass the entire 16M-byte virtual address space. The translation table contains sixteen entries, each of which carries a four-byte first-level descriptor.

Figure 36 shows the virtual address space of the DSP subsystem divided into sections and their relationship to the entries in the first-level translation table.

Virtual memory address range 0x00 0000 through 0x02 8000 corresponds to the DSP subsystem internal memory; therefore, section 0 is not a full 1MB. The DSP MMU only controls the mapping of addresses considered external to the DSP subsystem.

If MPNMC in ST3_55 is 0, the virtual memory address range 0xFF 8000 through 0xFF FFFF will be mapped to the DSP subsystem internal PDROM. Conversely, if MPNMC = 1, the internal PDROM will be disabled and the addresses will be mapped to external memory. The DSP MMU only controls the mapping of these addresses when MPNMC = 1.

Figure 36. DSP Subsystem Virtual Address Space Divided Into Sections

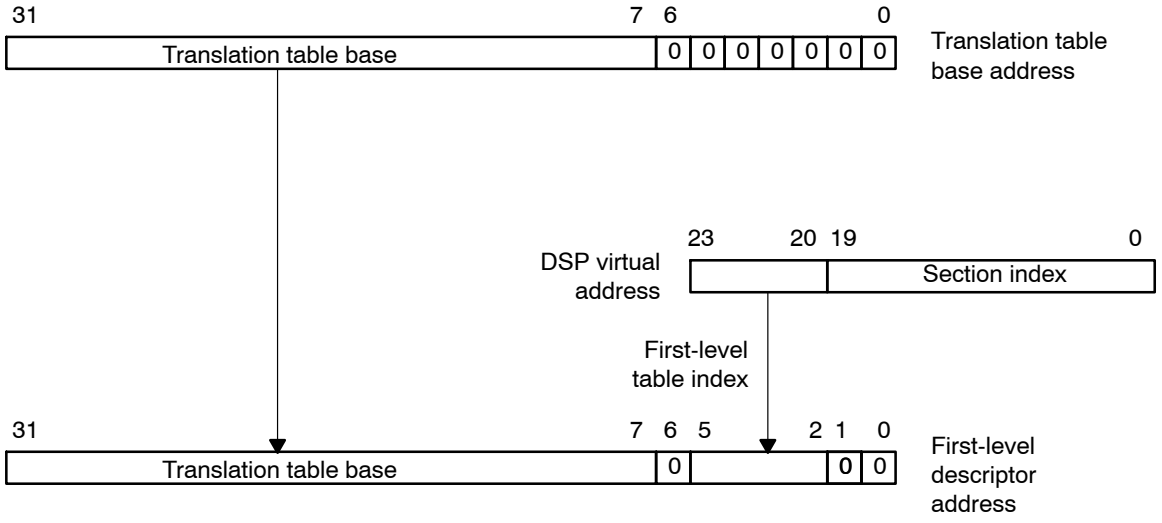


The following restrictions apply when using a first-level translation table:

- A total of 64-bytes (four bytes per descriptor) of memory must be allocated for the table.
- The start address of the translation table must be aligned to a 128-byte boundary; that is, the least significant seven address bits of the 32-bit start address must be zeros.

The 25 most-significant bits of the first-level translation table start address are called the translation table base. The translation table base is set by writing to the MMU Translation Table Registers (TTB_H_REG and TTB_L_REG). The four most-significant bits of the DSP virtual address are called the table index. The translation table base and the table index are used to calculate the address of the first-level descriptor (see Figure 37).

Figure 37. First-Level Descriptor Address Calculation



Notice that first-level descriptors have 32-bit addresses; consequently, they are aligned on 4-byte boundaries and the two least-significant bits of their addresses are zeros.

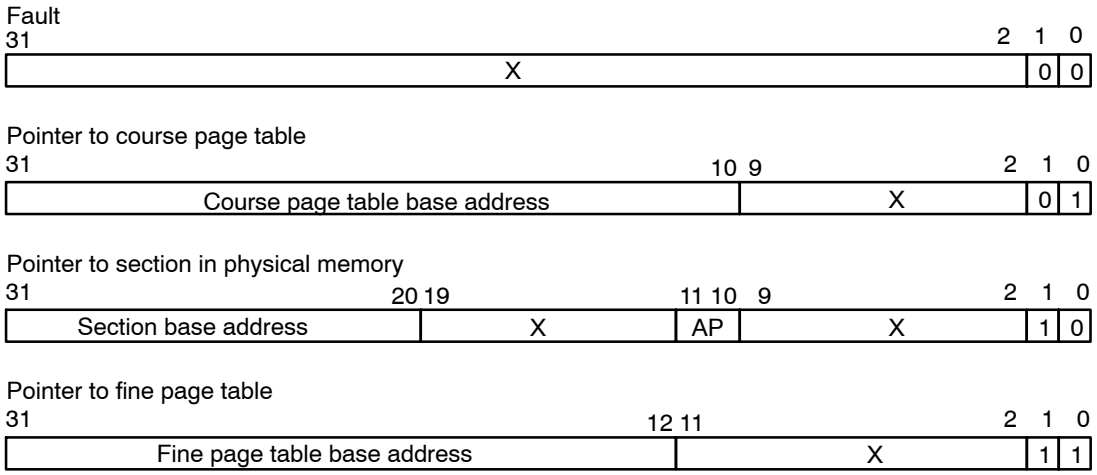
Once the descriptor address is known, the descriptor contents can be decoded to determine the translation information for the section. The next section describes the information contained in the descriptor.

6.2.5.1 First-Level Descriptor

Each first-level descriptor provides either the complete address translation for a section or a pointer to a second-level translation table. The descriptor can also indicate that a fault error must be generated if the section in virtual memory is accessed.

The least-significant two bits of the descriptor contents determine the type of information contained in the descriptor. Figure 38 shows how the contents of the first-level descriptor are interpreted based on the two least-significant bits. Table 19 further explains the meaning of each combination.

Figure 38. First-Level Descriptor Format Based on Two Least-Significant Bits



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

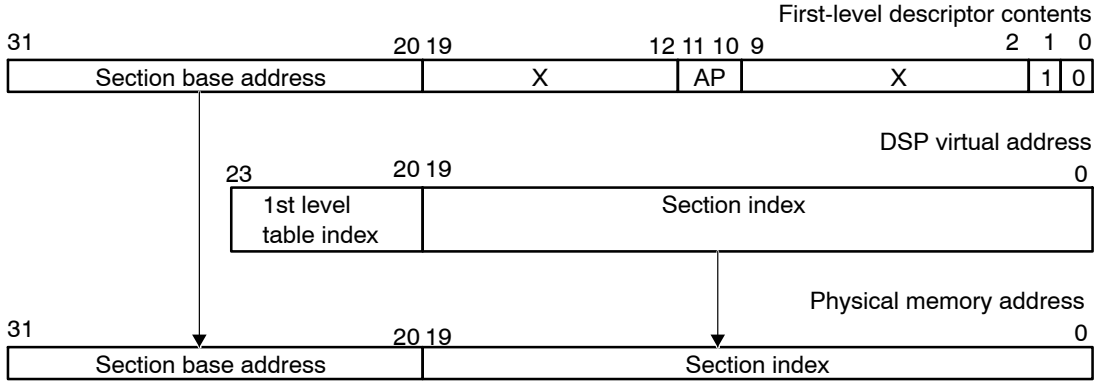
Table 19. First−Level Descriptor Contents

Least-Significant Two Bits of Descriptor Contents	Descriptor Contents Meaning
00b	Any access to this section in virtual memory will generate a fault error. As described in section 6.2.7, the fault error must be addressed by the MPU core. Until the error is cleared, the DSP EMIF will be stalled, therefore stalling the original requestor (either the DSP core or DMA).
01b	The descriptor contains the base address for a coarse page table. The coarse page table base address is used in conjunction with a second-level table index to determine the address of a second−level descriptor. The second-level descriptor provides the translation information for either a large page or a small page. Section 6.2.6.5 describes coarse page tables.
10b	The descriptor contains the base address for a section in physical memory. The section base address and the section index (bits 19−0 of the virtual address) are used determine the physical memory address. Section 6.2.5.2 describes this process.
11b	The descriptor contains the base address for a fine page table. The fine page table base address is used in conjunction with a second-level table index to determine the address of a second-level descriptor. The second-level descriptor provides the translation information for a large page, a small page, or a tiny page. Section 6.2.6.6 describes fine page tables.

6.2.5.2 Translating Sections

When the first-level descriptor contains a pointer to a section in physical memory, the section base address contained in the descriptor is used to calculate the physical memory address for the original DSP virtual address (Figure 39).

Figure 39. Translation for a Virtual Memory Section



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

Once the physical address is known, the data is accessed from physical memory, assuming the AP bits provide the correct access permissions.

6.2.6 Second-Level Translation Tables

First-level descriptors can provide a pointer to the base address of a second-level translation table. Second-level translation tables are used when a granularity smaller than a section is required.

There are two types of second-level translation tables:

- Coarse page tables with 256 entries.

  Descriptors for large and small pages can be used with coarse page tables.

- Fine page tables with 1024 entries.

  Descriptors for large, small, and tiny pages can be used with fine page tables.

The type of second-level translation table used depends on the system requirements. Fine page tables provide a finer granularity, plus they support all three page sizes; however, they require more space in memory. Coarse page tables require less space; however, they do not support tiny pages.

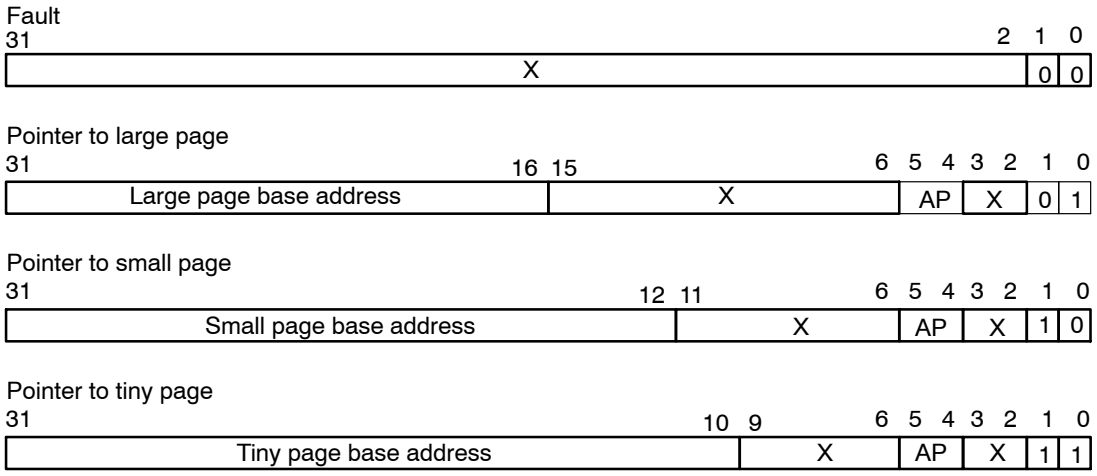
Both types of page tables contain second-level descriptors which provide the translation information for a large page, a small page, or a tiny page. Note that the format of the second-level descriptor is the same regardless of the type of second-level page table in which it is used. The type of the page table, however, determines the total number of descriptors needed.

6.2.6.1 Second-Level Descriptors

Second-level descriptors provide all the necessary information for the translation of a large, small, or tiny page. The descriptor can also indicate that a fault error must be generated if the page is accessed in virtual memory, similar to first-level section descriptors.

As with first-level descriptors, the least-significant two bits of the second-level descriptor contents determine the type of information contained in the descriptor. Figure 40 shows how the contents of the second-level descriptor are interpreted based on the two least-significant bits. Table 20 further explains the meaning of each combination.

Figure 40. Second-Level Descriptor Format Based on Two Least-Significant Bits



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

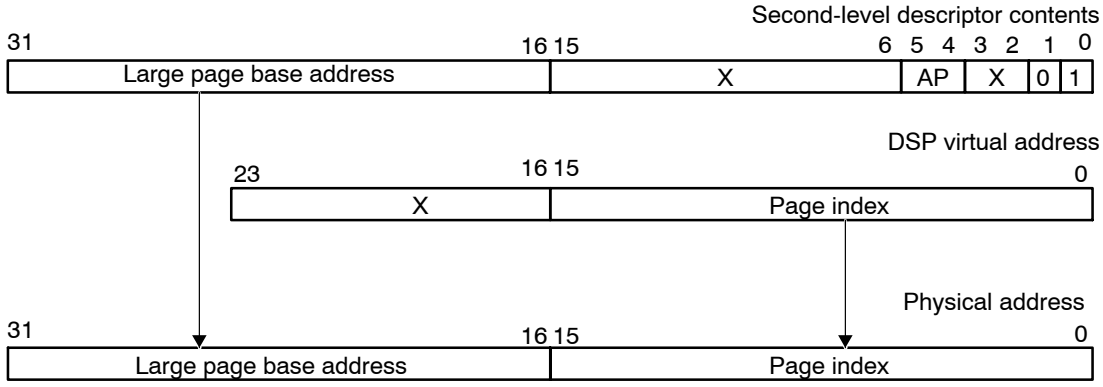
Table 20. First−Level Descriptor Contents

Least-Significant Two Bits of Descriptor Contents	Descriptor Contents Meaning
00b	Any access to the page in virtual memory corresponding to this descriptor will generate a fault. As described in section 6.2.7, the fault error must be addressed by the MPU core. Until the error is cleared, the DSP EMIF will be stalled, therefore stalling the original requestor (either the DSP core or DMA).
01b	The descriptor contains the base address for a large page. Section 6.2.6.2 describes the translation process for a large page.
10b	The descriptor contains the base address for a small page. Section 6.2.6.3 describes the translation process for a small page.
11b	The descriptor provides the base address of a tiny page (fine page tables only). Section 6.2.6.3 describes the translation process for a tiny page.

6.2.6.2 Translating Large Pages

Figure 41 describes how the contents of a large page descriptor are used to calculate the physical address of the DSP virtual address.

Figure 41. Translation for a Large Page

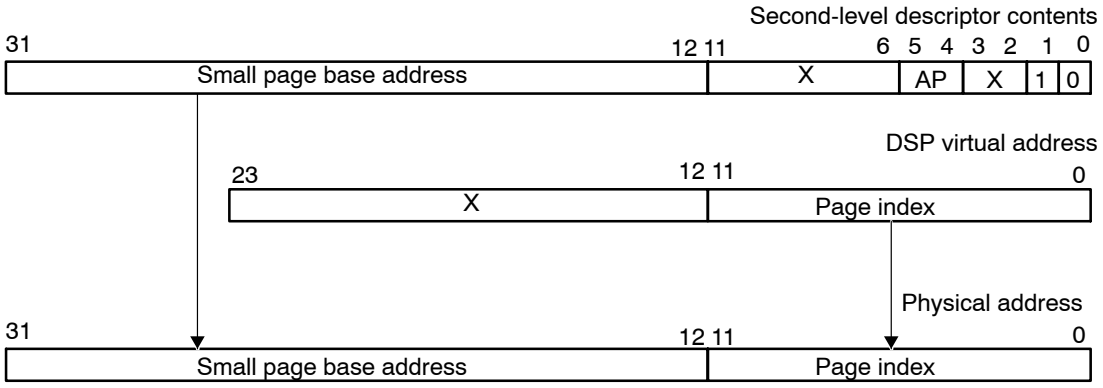


Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

6.2.6.3 Translating Small Pages

Figure 42 describes how the contents of a small page descriptor are used to calculate the physical address of the DSP virtual address.

Figure 42. Translation for a Small Page

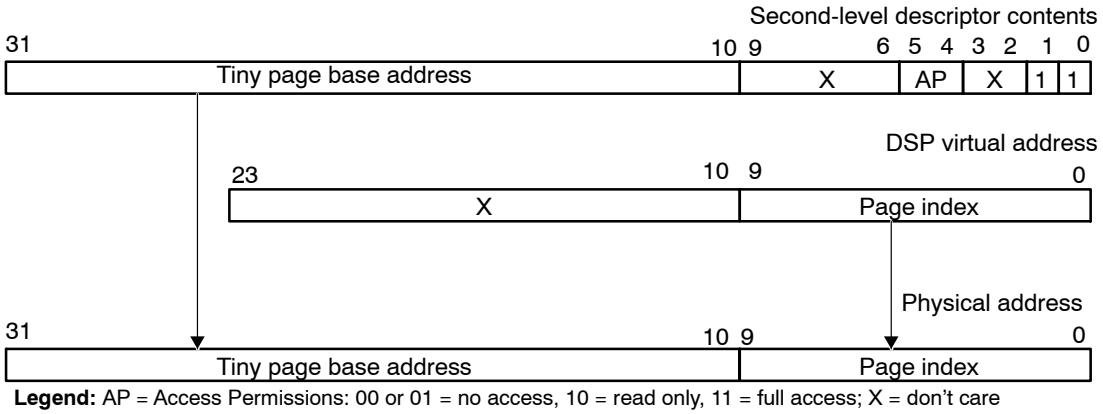


Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

6.2.6.4 Translating Tiny Pages

Figure 43 describes how the contents of a tiny page descriptor are used to calculate the physical address of the DSP virtual address.

Figure 43. Translation for a Tiny Page



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

6.2.6.5 Coarse Page Tables

Coarse page tables can be used to map large and small pages of virtual memory to physical memory. Each coarse table must contain 256 entries.

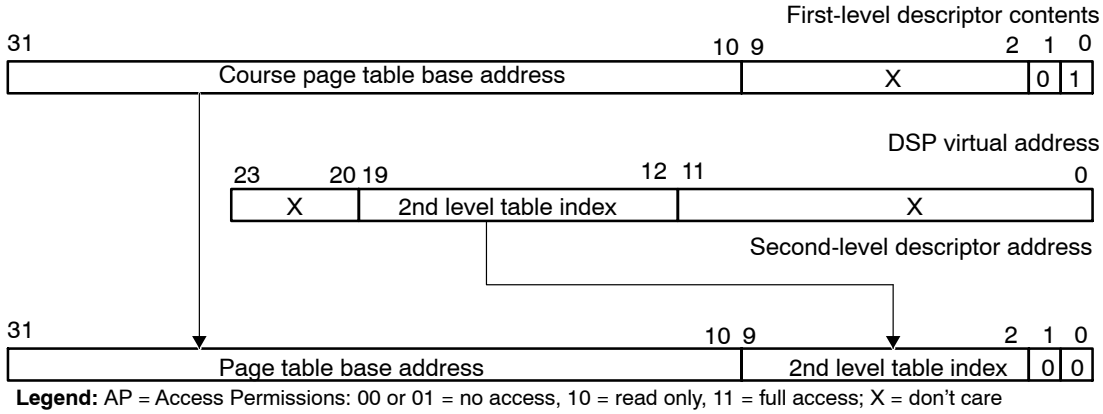
Follow these rules when using coarse page tables:

- The start address of a coarse page table must be aligned on a 1024-byte boundary; that is, the last 10 bits of its start address must be zeros.
- A descriptor for a large page must be repeated sixteen times. The repeated descriptor must start at an entry number that is a multiple of sixteen. As described in section 6.2.2, only one entry is required in the TLB to translate a large page.
- Descriptors for tiny pages cannot be used.

The address of the second-level descriptor is determined by using the course page table base address (contained in the first-level descriptor) and a second-level table index. The second-level table index is taken from the DSP virtual address.

Figure 44 describes how to generate the descriptor address for coarse page tables.

Figure 44. Calculating the Descriptor Address in a Coarse Page Table



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

Notice that the MMU indexes the coarse table as if the entries were specifying small pages. That is, it always selects 1 of 256 entries. However, the MMU uses 16 bits from the second-level descriptor as a base address for a large page and 20 bits for a small page (see Figure 41 and Figure 42, respectively). This behavior means that when large pages are used, the descriptor for a large page must be repeated sixteen times in the coarse page table.

As described in section 6.2.2, the TLB can be used to bypass the translation tables. Using this approach, only one TLB entry is required to translate a large page.

6.2.6.6 Fine Page Tables

Fine page tables can be used to map large, small, and tiny pages of virtual memory to physical memory. The added granularity comes at a cost, because each fine page table must contain 1024 entries.

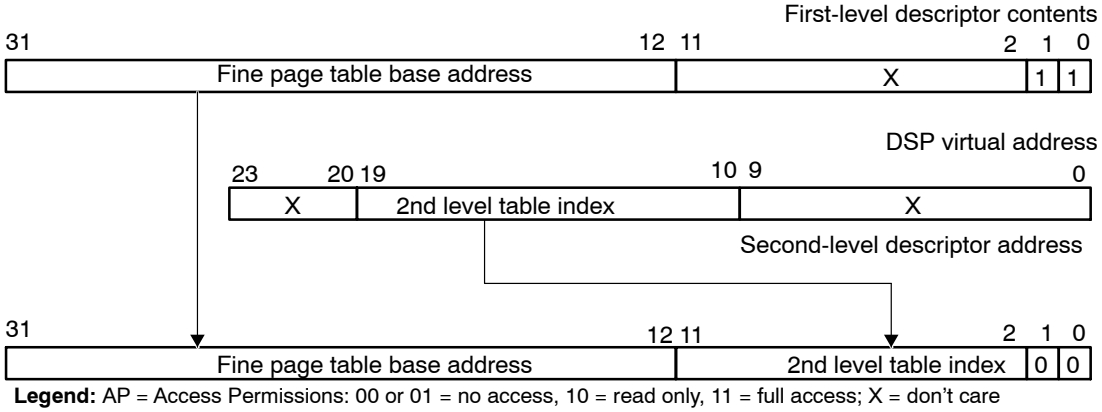
Follow these rules when using fine page tables:

- The start address of fine tables must be aligned on a 4096-byte boundary; that is, the last 12 bits of its start address must be zeros.
- A descriptor for a large page must be repeated 64 times. The repeated descriptor must start at an entry number that is a multiple of 64. As described in section 6.2.2, only one entry is required in the TLB to translate a large page.
- A descriptor for a small page must be repeated four times. The repeated descriptor must start at an entry number that is a multiple of four. As described in section 6.2.2, only one entry is required in the TLB to translate a small page.

The address of the second-level descriptor is determined by using the fine page table base address (contained in the first-level descriptor) and a second-level table index. The second-level table index is taken from the DSP virtual address.

Figure 45 describes how the descriptor address is generated for fine page tables.

Figure 45. Calculating the Descriptor Address in a Fine Page Table



Legend: AP = Access Permissions: 00 or 01 = no access, 10 = read only, 11 = full access; X = don't care

Notice that the MMU indexes the coarse table as if the entries were specifying tiny pages. That is, it always selects 1 of 1024 entries. However, the MMU uses 16 bits from the second−level descriptor as a base address for a large page and 22 bits for a tiny page (see Figure 41 and Figure 43, respectively). This behavior means that when large pages are used, the descriptor for a large page must be repeated 64 times in the coarse page table. For similar reasons, a descriptor for a small page must be repeated 16 times in the coarse page table.

As described in section 6.2.2, the TLB can be used to bypass the translation tables. Using this approach, only one TLB entry is required to translate a large page or a small page.

6.2.7 MMU Error Handling

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

- Pre-fetch error.

  An error occurred during an address-translation pre-fetch request from the DSP core. The error may have occurred due to a TLB miss or a translation fault as described below.

- 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, an interrupt is signaled to the MPU core. 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 ISR can determine the cause of the interrupt by reading the fault status register (FAULT_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).

Note:

The DSP EMIF will be stalled, thus stalling the original requestor (either the DSP core or DMA), while the error is cleared by the MPU core.

The ISR can service each error as follows:

- For a pre-fetch or translation fault, the ISR must write a valid entry to the TLB and acknowledge the interrupt through the interrupt acknowledge register (IT_ACK_REG). The translation table(s) can also be updated such that the error is not generated again if the TLB entry is evicted or flushed.
- For a TLB miss, the ISR must write a valid entry to the TLB and acknowledge the interrupt through the interrupt acknowledge register (IT_ACK_REG).
- For a permission fault, the MPU core must write a valid entry to the TLB to allow for the requested access type and then acknowledge the interrupt through the interrupt acknowledge register (IT_ACK_REG). The translation table(s) can also be updated such that the error is not generated again if the TLB entry is evicted or flushed.

The ISR may also reset the DSP subsystem in response to any MMU interrupt.

6.2.8 Reset Considerations

6.2.8.1 Software Reset Considerations

A software reset of the DSP MMU can be initiated by setting the MMU_RESET bit in the CNTL_REG register of the MMU. After a software reset, the preserved and valid bits of all the entries in the TLB are cleared. The victim and base pointers are not affected by a software reset. Also, the table walking logic does not become disabled after a software reset.

6.2.8.2 Hardware Reset Considerations

After a hardware reset (section 12.1), the MMU is disabled and the DSP external memory space is mapped to the first 16M bytes of system memory. Also, the MMU does not perform any permission checks on DSP external memory accesses. All MMU registers return to their default state as indicated in section 6.5.

6.2.9 Clock Control

The DSP MMU module is clocked by the DSPMMU_CK included in the DSP clock domain. The DSP domain clock can be divided by 1, 2, 4, or 8 to generate the MMU clock by using the DSPMMUDIV bits of the ARM_CKCTL register. By default, the DSPMMUDIV bits are set to divide-by-one mode. DSPMMU_CK can be shut off by setting the GL_PDE bit of the DSPMMU_IDLE_CTRL register (section 6.5.17).

Note:

The DSP MMU clock must follow these rules:

- The DSP MMU clock frequency must be greater than or equal to the traffic controller clock frequency.
- The DSP MMU clock frequency must be 1 or 1/2 times the DSP subsystem clock frequency.

6.2.10 Initialization

The DSP MMU clock must be configured as described in section 6.2.9 before programming the DSP MMU.

Preferably, the DSP MMU should be configured before the DSP core is taken out of reset. Note that the LD_TLB_REG, TTB_H_REG, TTB_L_REG, and the LOCK_REG registers cannot be written to once the table walking logic has been enabled (TWL_EN = 1 in CNTL_REG).

6.2.11 Interrupt Support

6.2.11.1 Interrupt Events and Requests

The DSP MMU generates a single interrupt to the MPU core in response to a translation error. The ISR then determines the cause of the interrupt by reading the fault status register (FAULT_ST_REG). The ISR may take one of two actions to clear the interrupt from the MMU:

- The ISR may clear the error condition as described in section 6.2.7.
- The ISR may reset the DSP subsystem through the DSP_EN bit of the MPU-Reset-Control-1 Register (ARM_RSTCT1) and reset the DSP MMU through the MMU_RESET bit of the control register (CNTL_REG).

6.2.11.2 Interrupt Multiplexing

The DSP MMU interrupt is managed by the MPU level 2 interrupt handler. Before the MPU core can see the DSP MMU interrupt, DSP_MMU_IRQ (IRQ_28) must be enabled and configured as a level-sensitive interrupt. More information on the MPU level 2 interrupt handler can be found in the *OMAP5912 Multimedia Processor Interrupts Reference Guide* (SPRU757).

6.2.12 Power Management

The clock to the DSP MMU can be shut off to save power. The GL_PDE bit of the DSPMMU_IDLE_CTRL register can be set to completely shut off the clock to the DSP MMU. Alternatively, the AUTOGATING_EN bit can be set such that the clock to the DSP MMU is only shut off when DSP MMU is not active.

6.3 Using the MPU to Manage the TLB

The DSP MMU generates a physical address for every virtual address generated by the DSP external memory interface (EMIF) by using address-translation information stored in its TLB. The DSP MMU includes table walking logic, which automatically fetches the address-translation information from a set of translation tables and updates the TLB. As an alternative to using the table walking logic, the MPU core can be used to write entries to the TLB. No translation tables are needed when using this approach.

6.3.1 Architectural/Operational Description

Four major steps are taken when the DSP subsystem accesses DSP external memory.

1) The DSP core or the DSP DMA requests an access to DSP external memory.

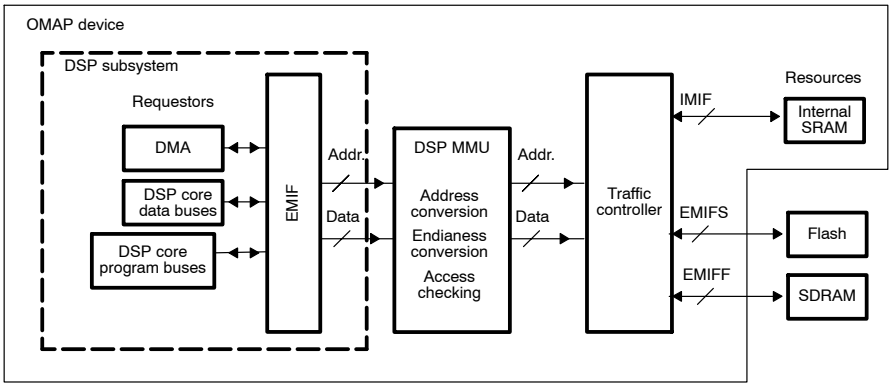
2) The DSP EMIF receives that request and forwards it to the DSP MMU.

3) The MMU checks its TLB for a match on the virtual address tag. If there is a TLB hit and the correct access permissions for the type of access (read or write) are present, the MMU translates the virtual address from the EMIF into a physical address and forwards the request to the traffic controller with the appropriate endianess conversion. If the virtual address tag is not found or if incorrect access permissions are present, the MMU generates an interrupt to the MPU core and stalls the DSP EMIF until the error is cleared. When the MPU core clears this error, the DSP MMU repeats this entire step.

4) The traffic controller accesses the actual OMAP resource.

Figure 46 shows the major blocks involved during an access to DSP external memory.

Figure 46. DSP Subsystem External Memory Interface



6.3.2 Software Configuration

The DSP MMU is initialized by the MPU core. To prevent a DSP access to DSP external memory while the MMU is disabled, it is recommended that the MMU be initialized and enabled before the DSP subsystem is taken out of reset.

The MPU core must follow these steps to initialize and enable the DSP MMU:

1) Configure and enable the DSP MMU clock:
   a) The MMU clock is derived from the CK_GEN2 clock domain. The DSPMMUDIV bits of the ARM_CKCTL register are used to divide the CK_GEN2 clock by 1, 2, 4, or 8. The MMU clock has specific restrictions. See section 6.2.9 for more details.
2) Take the DSP MMU out of reset by setting the MMU_RESET of the CNTL_REG.
3) Write entries to the TLB.
   a) Determine CAM and RAM parameters and write them into the CAM and RAM registers (CAM_H_REG, CAM_L_REG, RAM_H_REG, and RAM_L_REG). See section 6.2.2.1 for information on CAM and RAM values.
   b) Select the TLB entry to be written by setting the victim pointer through the Lock/Protect Entry Register (LOCK_REG). For example, to update entry 0 in the TLB, write 0 to the victim pointer field of LOCK_REG.
   c) Set the WRITE_ENTRY bit in the Read/Write TLB Entry Register (LD_TLB_REG).
   d) Repeat these steps for every entry that is to be written to the TLB.

4) Configure the MPU level 2 interrupt handler such that DSP MMU interrupts are enabled and can be serviced by the MPU core. More information on the MPU level 2 interrupt handler can be found in the *OMAP5912 Multimedia Processor Interrupts Reference Guide* (SPRU757).

5) Enable the DSP MMU by setting the MMU_EN bit in CNTL_REG.

6) Take the DSP subsystem out of reset by setting the DSP_EN bit in the MPU-Reset-Control-1 Register (ARM_RSTCT1).

6.3.3 System Traffic Considerations

All DSP subsystem accesses to DSP external memory eventually go through the traffic controller. The access time for a DSP external memory request will depend on the amount of competing accesses in the traffic controller, as well as the configurations of the OMAP external memory interfaces (EMIFF and EMIFS).

6.4 Using Table Walking Logic to Manage the TLB

The DSP MMU generates a physical address for every virtual address generated by the DSP external memory interface (EMIF) by using address-translation information stored in its TLB. The DSP MMU includes table walking logic, which automatically fetches the address-translation information from a set of translation tables and updates the TLB. This section describes the steps needed to set up the table walking logic to manage the TLB.

6.4.1 Architectural/Operational Description

Four major steps are taken when the DSP subsystem accesses DSP external memory.

1) The DSP core or the DSP DMA requests an access to DSP external memory.

2) The DSP EMIF receives that request and forwards it to the DSP MMU.

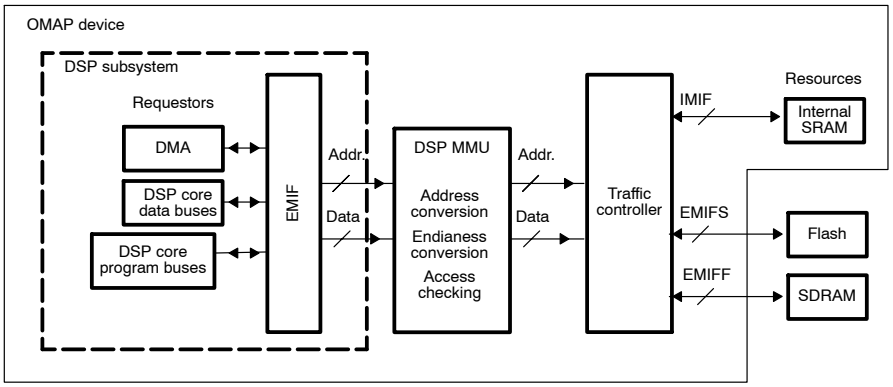
3) The MMU checks its TLB for a match on the virtual address tag. If there is a TLB hit and the correct access permissions for the type of access (read or write) are found, the MMU translates the virtual address from the EMIF into a physical address and forwards the request to the traffic controller with the appropriate endianess conversion.

   Otherwise, if the virtual address tag is not found, the MMU uses its table walking logic to fetch the translation from translation tables and updates the TLB. If correct access permissions are found, the MMU carries out the virtual-to-physical address translation and forwards the request to the traffic controller. If the correct access permissions are not found, MMU generates an interrupt to the MPU core and stalls the DSP EMIF until the error is cleared. When the MPU core clears this error, the DSP MMU repeats this entire step.

4) The traffic controller accesses the actual OMAP resource.

Figure 47 shows the major blocks involved during an access to DSP external memory by the DSP subsystem.

Figure 47. DSP Subsystem External Memory Interface



6.4.2 Software Configuration

The DSP MMU is initialized by the MPU core. To prevent a DSP access to DSP external memory while the MMU is disabled, it is recommended that the MMU be initialized and enabled before the DSP subsystem is taken out of reset.

The MPU core must follow these steps to initialize and enable the DSP MMU:

1) Set up the translation tables.

   The translation tables can be placed anywhere in shared memory (CS0, CS1, etc.). Depending on the table structure selected, one or more tables may be needed.

   See sections 6.2.5 and 6.2.6 for more information on first- and second-level translation tables.

2) Configure and enable the DSP MMU clock.

   The MMU clock is derived from the CK_GEN2 clock domain. The DSPMMUDIV bits of the ARM_CKCTL register are used to divide the CK_GEN2 clock by 1, 2, 4, or 8.

   The MMU clock has specific restrictions, see section 6.2.9 for more details.

3) Take the DSP MMU out of reset by setting the MMU_RESET of the CNTL_REG.

4) Write the first-level translation table base address to the Translation Table Registers (TTB_H_REG and TTB_L_REG).

   The table base address corresponds to the 25 most-significant bits of the 32-bit shared memory address of the first-level translation table.

5) Configure the MPU level 2 interrupt handler such that DSP MMU interrupts are enabled and can be serviced by the MPU core. More information on the MPU level 2 interrupt handler can be found in the *OMAP5912 Multimedia Processor Interrupts Reference Guide* (SPRU757).

6) Enable the DSP MMU and the table walking logic by setting both the MMU_EN bit and the TWL_EN bit in CNTL_REG.

7) Take the DSP subsystem out of reset by setting the DSP_EN bit in the MPU-Reset-Control-1 Register (ARM_RSTCT1).

Notice that TLB entries can also be written to the TLB before the MMU is enabled. In this case, make sure to set the base pointer such that the entries written to the TLB are protected from eviction by the table walking logic (see section 6.2.2.4 for more details).

6.4.3 System Traffic Considerations

All DSP subsystem accesses to DSP external memory eventually go through the traffic controller. The access time for a DSP external memory request will depend on the amount of competing accesses in the traffic controller, as well as the configurations of the OMAP external memory interfaces (EMIFF and EMIFS).

6.5 DSP MMU Registers

6.5.1 Overview

The DSP MMU is programmed by the MPU core via a set of configuration registers. Table 21 shows these registers, their access types, and their addresses.

Table 21. Summary of DSP MMU Registers

Name	Description	MPU Byte Address†	See Section
PREFETCH_REG‡	Pre-fetch register. An entry for the TLB can be pre-fetched by writing a virtual address tag into this register.	0xFFFE D200	6.5.2
WALKING_ST_REG	Pre-fetch status register. Use this register to determine if the table walking logic is performing an address translation or if a pre-fetch operation has completed.	0xFFFE D204	6.5.3
CNTL_REG	Control register. Use this register to enable the MMU and the table walking logic, and to reset the MMU.	0xFFFE D208	6.5.4
FAULT_AD_H_REG FAULT_AD_L_REG	Fault address registers. These registers display the virtual address of an access which caused an MMU error.	0xFFFE D20C 0xFFFE D210	6.5.5
FAULT_ST_REG	Fault status register. When an MMU error is generated, use this register to determine the cause of the error.	0xFFFE D214	6.5.6
IT_ACK_REG	Interrupt acknowledge register. Use this register to notify the MMU that an error has been corrected.	0xFFFE D218	6.5.7
TTB_H_REG TTB_L_REG	TTB registers. Use these registers to specify the first-level translation table base address.	0xFFFE D21C 0xFFFE D220	6.5.8
LOCK_REG	Lock/protect entry. Use this register to set the base pointer and the victim pointer of the TLB.	0xFFFE D224	6.5.9
LD_TLB_REG	Read/write TLB entry. Use this register to start a TLB entry read or TLB entry write operation.	0xFFFE D228	6.5.10
CAM_H_REG CAM_L_REG	CAM entry registers. Use these registers to specify the CAM value to be written to the entry pointed to by the victim pointer.	0xFFFE D22C 0xFFFE D230	6.5.11
RAM_H_REG RAM_L_REG	RAM entry registers. Use these registers to specify the RAM value to be written to the entry pointed to by the victim pointer.	0xFFFE D234 0xFFFE D238	6.5.12

† MPU byte addresses apply to both OMAP5910 and OMAP5912.
‡ This register is accessible by the DSP core at I/O address 0x4400.

Table 21. Summary of DSP MMU Registers (Continued)

Name	Description	MPU Byte Address†	See Section
GFLUSH_REG	Global flush register. Use this bit to flush all non-preserved entries from the TLB.	0xFFFE D23C	6.5.13
FLUSH_ENTRY_REG	Individual flush register. Use this register to flush a single entry from the TLB.	0xFFFE D240	6.5.14
READ_CAM_H_REG READ_CAM_L_REG	Read CAM registers. When reading from the TLB, the CAM value is retrieved from these registers.	0xFFFE D244 0xFFFE D248	6.5.15
READ_RAM_H_REG READ_RAM_L_REG	Read RAM registers. When reading from the TLB, the RAM value is retrieved from these registers.	0xFFFE D24C 0xFFFE D250	6.5.16
DSPMMU_IDLE_CTRL	MMU Idle Control register. Use this register to control the power-down capabilities of the DSP MMU.	0xFFFE D254	6.5.17

† MPU byte addresses apply to both OMAP5910 and OMAP5912.
‡ This register is accessible by the DSP core at I/O address 0x4400.

6.5.2 MMU Pre-Fetch Register (PREFETCH_REG)

The table walking logic automatically fetches an entry for the TLB when a TLB miss is generated. The DSP core can force the table walking logic to pre-fetch an entry for the TLB by writing a virtual address tag to the PREFETCH_REG. Note that the virtual address tag corresponds to the 14 most-significant bits of a virtual address.

The status of the pre-fetch operation is shown in the PREFETCH_ON bit of the WALKING_ST_REG.

This register is visible from both the DSP side and the MPU side; however, on the DSP side this register is write-only; and on the MPU side this register is read-only.

Figure 48. MMU Pre-Fetch Register (PREFETCH_REG)

DSP Side

16	14	13	0
Reserved		PREFETCH_ADDR	
R-0		W-0	

MPU Side

31	14	13	0
Reserved		PREFETCH_ADDR	
R-0		R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 22. MMU Pre-Fetch Register (PREFETCH_REG) Field Descriptions

Bits	Field	Value	Description
31–14	Reserved		These bits are not used.
13–0	PREFETCH_ADDR		Virtual address tag of the TLB entry to be pre-fetched.

6.5.3 MMU Pre-Fetch Status Register (WALKING_ST_REG)

Use the WALKING_ST_REG to determine when the table walking logic has completed a TLB-entry pre-fetch operation. This register can also be used to determine when the table walking logic is busy handling a miss in the TLB.

Figure 49. MMU Pre-Fetch Status Register (WALKING_ST_REG)

31	2	1	0
Reserved		WALK_ WORKING	PREFETCH_ ON
R-0		R-0	R-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 23. MMU Pre-Fetch Status Register (WALKING_ST_REG) Field Descriptions

Bits	Field	Value	Description
31–2	Reserved		These bits are not used.
1	WALK_WORKING		This bit is used to indicate when the table walking logic is performing an address translation after a miss in the TLB.
		0	Table walking logic is not performing any action.
		1	Table walking logic is performing an address translation.
0	PREFETCH_ON		This bit is used to indicate the status of a TLB-entry pre-fetch operation. When a value is written into the PREFETCH_REG register, the pre-fetch operation is started and this bit is set. When the pre-fetch operation finishes, this bit is automatically cleared by the MMU.
		0	The pre-fetch operation has been completed.
		1	A value has been written to the PREFETCH_REG and the table walking logic is fetching the entry for the TLB.

6.5.4 MMU Control Register (CNTL_REG)

The Control Register (CNTL_REG) is used to reset and enable the DSP MMU module and to enable the table walking logic.

Note:

The DSP MMU module must be reset through the MMU_RESET bit of the CNTL_REG register before the MMU is enabled.

Figure 50. MMU Control Register (CNTL_REG)

31	3	2	1	0
Reserved		TWL_EN	MMU_EN	MMU_ RESET
R-0		RW-0	RW-0	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 24. Control Register (CNTL_REG) Field Descriptions

Bits	Field	Value	Description
31–3	Reserved		These bits are not used.
2	TWL_EN		Enables the table walking logic. **Note:** When the table walking logic is enabled, the TLB cannot be manually updated; you should not write to the LD_TLB_REG, TTB_H_REG, TTB_L_REG, and LOCK_REG.
		0	Table walking logic is disabled; access to the TLB is permitted.
		1	Table walking logic is enabled; access to the TLB is not permitted.
1	MMU_EN		Enables the MMU. **Note:** Before enabling the MMU, you must reset it using the MMU_RESET bit.
		0	The MMU is disabled.
		1	The MMU is enabled.
0	MMU_RESET		Resets the MMU module. Writing a 0 to this bit resets the MMU to its default configuration. **Note:** You must clear this bit before enabling the MMU.
		0	The MMU is in reset.
		1	The MMU has been reset successfully.

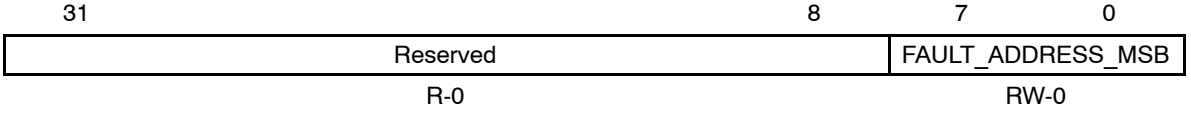
6.5.5 MMU Fault Address Registers (FAULT_AD_H_REG, FAULT_AD_L_REG)

When a fault is generated, the fault address registers are used to determine the virtual address that generated the fault. The eight most-significant bits of the 24-bit virtual address are displayed in FAULT_AD_H_REG, while the rest of the address bits are displayed in FAULT_AD_L_REG.

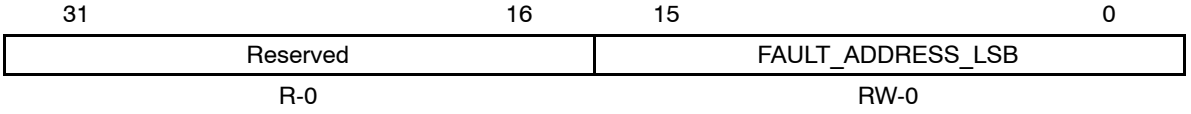
To determine the type of fault generated, see the Fault Status Register (FAULT_ST_REG) in section 6.5.6.

Figure 51. MMU Fault Address Registers (FAULT_AD_H_REG, FAULT_AD_L_REG)

FAULT_AD_H_REG



FAULT_AD_L_REG



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 25. MMU MSB Fault Address Register (FAULT_AD_H_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–0	FAULT_ADDRESS_MSB		Most-significant bits of the 24-bit virtual address which caused a fault.

Table 26. MMU LSB Fault Address Register (FAULT_AD_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–0	FAULT_ADDRESS_LSB		Least-significant bits of the 24-bit virtual address which caused a fault.

6.5.6 MMU Fault Status Register (FAULT_ST_REG)

When an error is generated by the MMU, the MMU Fault Status Register (FAULT_ST_REG) determines the cause of the error. The MMU generates errors based on the following conditions:

- TLB miss (table walker disabled):

  No translation is found in the TLB for the virtual address issued. The hardware table walker is disabled, and thus, 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.

- Table walker logic pre-fetch error:

  An error occurred during an address-translation pre-fetch request from the DSP core. The error may have occurred due to a TLB miss or a translation fault.

After the MPU core clears the error condition, it must acknowledge the error using the Interrupt Acknowledge Register (IT_ACK_REG). Section 6.2.7 describes the steps needed to clear the MMU error conditions.

Figure 52. MMU Fault Status Register (FAULT_ST_REG)

31 4	3	2	1	0
Reserved	PREFETCH_ ERR	PERM_ FAULT	TLB_ MISS	TRANS_ FAULT
R-0	R-0	R-0	R-0	R-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 27. MMU Fault Status Register (FAULT_ST_REG) Field Descriptions

Bits	Field	Value	Description
31–4	Reserved		These bits are not used.
3	PREFETCH_ERR		This bit indicates when table walking logic was unable to pre-fetch an entry for the TLB.
		0	No pre-fetch error has occurred.
		1	A pre-fetch error has occurred.

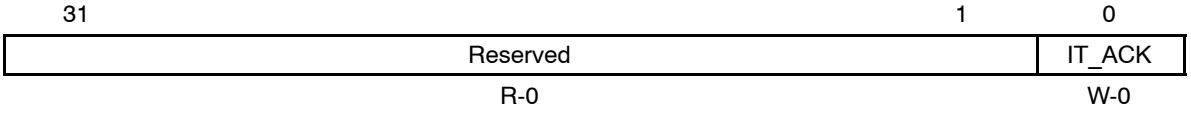
Table 27. MMU Fault Status Register (FAULT_ST_REG) Field Descriptions (Continued)

Bits	Field	Value	Description
2	PERM_FAULT		This bit indicates when the DSP core attempted to access a section/page without the proper access permissions.
		0	No permission fault exists.
		1	A permission fault has been generated.
1	TLB_MISS		This bit indicates a TLB miss has been generated and the table walking logic is disabled.
		0	No error of this type has occurred.
		1	A TLB miss has been generated and the table walking logic is not enabled.
0	TRANS_FAULT		This bit indicates a TLB miss has been generated and the table walking logic was unable to find a valid section/page table entry for the given virtual address.
		0	No error of this type has occurred.
		1	The table walking logic was not able to find a valid section/page table entry to service the TLB miss.

6.5.7 MMU Interrupt Acknowledge Register (IT_ACK_REG)

Use this register to signal to the MMU that the MPU core has taken care of the error condition displayed by the Fault Status Register (FAULT_ST_REG). See section 6.2.7 for more details.

Figure 53. MMU Interrupt Acknowledge Register (IT_ACK_REG)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 28. MMU Interrupt Acknowledge Register (IT_ACK_REG) Field Descriptions

Bits	Field	Value	Description
31–1	Reserved		These bits are not used.
0	IT_ACK		The MPU core must write a 1 to this bit to acknowledge the interrupt from the DSP MMU.
		0	Writing 0 has no effect.
		1	Writing a 1 to this bit acknowledges the interrupt from the DSP MMU.

6.5.8 MMU Translation Table Registers (TTB_H_REG, TTB_L_REG)

These registers together specify the base address of the first-level translation table. The base address corresponds to the 25 most-significant bits of the 32-bit address of the first-level translation table.

Note:

TTB_H_REG and TTB_L_REG can only be modified by the MPU core when the table walking logic is disabled.

Figure 54. MMU Translation Table Registers (TTB_H_REG, TTB_L_REG)

TTB_H_REG

31	16	15	0
Reserved		TTB_H	
R-0		RW-0	

TTB_L_REG

31	16
Reserved	
R-0	

15	7	6	0
TTB_L		Reserved	
RW-0		R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 29. MMU MSB Translation Table Register (TTB_H_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–0	TTB_H	0x0000–0xFFFF	Most-significant bits of the 25-bit base address of the first-level translation table.

Table 30. MMU LSB Translation Table Register (TTB_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–7	TTB_L	0x0000–0x01FF	Least-significant bits of the 25-bit base address of the first-level translation table.
6–0	Reserved		These bits are not used.

6.5.9 MMU Lock/Protect Entry Register (LOCK_REG)

The Lock/Protect Entry Register is used to set the victim and base pointers of the TLB.

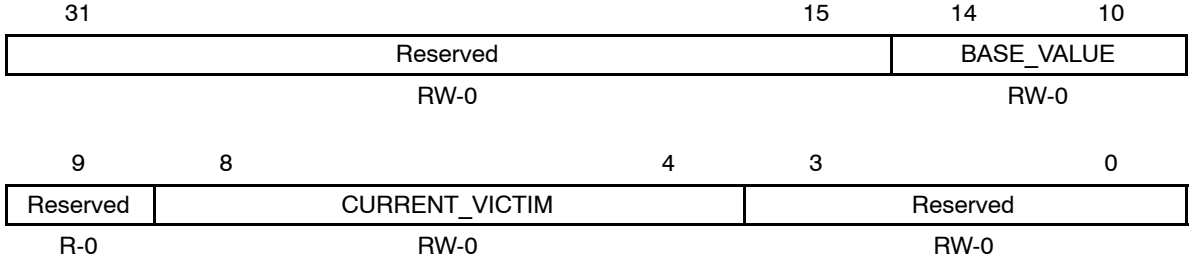
The victim pointer identifies the TLB entry which is due for eviction by the table walking logic. The MPU core can also set it to select a TLB entry for reading or writing.

The base pointer specifies which entries in the TLB are protected against eviction by the table walking logic.

Note:

The LOCK_REG can only be modified by the MPU core when the table walking logic is disabled.

Figure 55. MMU Lock/Protect Entry Register (LOCK_REG)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 31. MMU Lock/Protect Entry Register (LOCK_REG) Field Descriptions

Bits	Field	Value	Description
31–15	Reserved		These bits are not used. Always write 0 to these bits.
14–10	BASE_VALUE	0–31	TLB lock base pointer. The value, n, that is written to these bits locks the first n TLB entries. Entries 0 to n-1 will not be evicted by the table walking logic when the TLB becomes full. Notice that n must be less than or equal to 31; that is, at least one entry must always be unprotected in the TLB.
9	Reserved		This bit is not used.
8–4	CURRENT_VICTIM	0–31	TLB victim pointer. This field displays the entry number currently selected by the victim pointer. The victim pointer identifies the TLB entry which is next to be overwritten by the table walking logic. The victim pointer is also used when reading or writing TLB entries.
3–0	Reserved		These bits are not used. Always write 0 to these bits.

6.5.10 MMU Read/Write TLB Entry Register (LD_TLB_REG)

The Read/Write TLB Entry Register is used to read or write entries from or to the TLB. The victim pointer (set through the Lock/Protect Entry Register) selects the entry to be read or written.

When the WRITE_ENTRY bit is set, the values in the CAM Entry Registers and the RAM Entry Registers are written to the entry pointed to by the TLB.

When the READ_ENTRY bit is set, the TLB entry CAM and RAM values are copied to the CAM Entry Read Registers and the RAM Entry Read Registers.

Note:

LD_TLB_REG can only be modified by the MPU core when the table walking logic is disabled.

Figure 56. MMU Read/Write TLB Entry Register (LD_TLB_REG)

31	2	1	0
Reserved		READ_ ENTRY	WRITE_ ENTRY
R-0		RW-0	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 32. MMU Read/Write TLB Entry Register (LD_TLB_REG) Field Descriptions

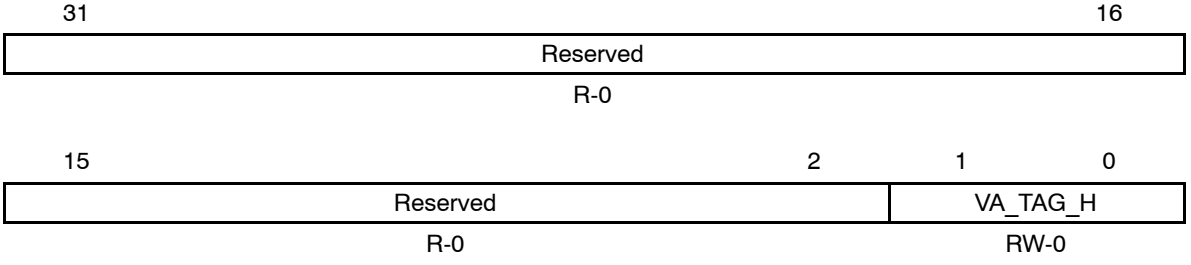
Bits	Field	Value	Description
31–2	Reserved		These bits are not used.
1	READ_ENTRY		Read the TLB entry. Writing 1 to this field causes an entry to be read from the TLB. This bit is always 0 when read.
		0	Writing a 0 to this bit has no effect.
		1	Read the TLB entry specified by the victim pointer.
0	WRITE_ENTRY		Write the TLB entry. Writing 1 to this field causes an entry to be loaded into the TLB. This bit is always 0 when read.
		0	Writing a 0 to this bit has no effect.
		1	Write the programmed entry to the TLB at location specified by the victim pointer.

6.5.11 MMU CAM Entry Registers (CAM_H_REG, CAM_L_REG)

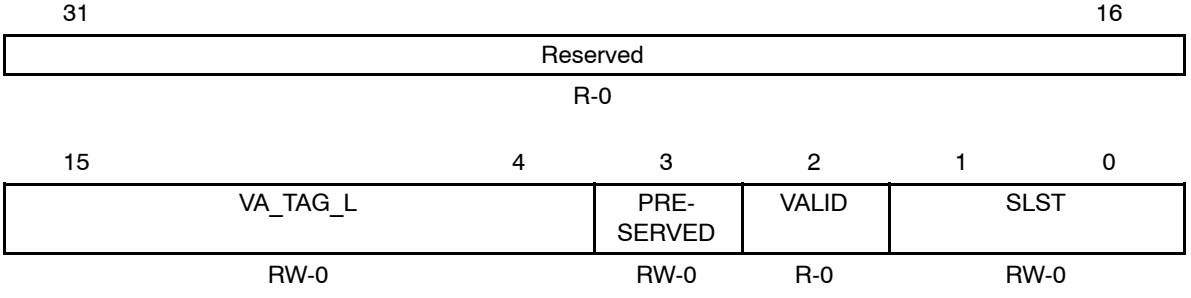
The CAM Entry Registers specify a CAM value to be written into the TLB.

Figure 57. MMU CAM Entry Registers (CAM_H_REG, CAM_L_REG)

CAM_H_REG



CAM_L_REG



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 33. MMU MSB CAM Entry Register (CAM_H_REG) Field Descriptions

Bits	Field	Value	Description
31–2	Reserved		These bits are not used.
1–0	VA_TAG_H	0x0–0x3	Most-significant bits of the virtual address tag. The VA_TAG bits correspond to bits 23–10 of the DSP virtual address. Note that, depending on the page size, not all of the VA_TAG bits are needed; these unneeded bits must be written as zeros.

Table 34. MMU LSB CAM Entry Register (CAM_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–4	VA_TAG_L		Least-significant bits of the virtual address tag. The VA_TAG bits correspond to bits 23–10 of the DSP virtual address. Note that, depending on the page size, not all of the VA_TAG bits are needed; these unneeded bits must be written as zeros.
3	PRESERVED		Preserve bit for the TLB entry. This bit specifies whether the TLB entry should be kept during a TLB global flush.
		0	TLB entry is not preserved during a TLB global flush.
		1	TLB entry is preserved during a TLB global flush.
2	VALID		Valid bit for the TLB entry. This bit specifies whether the TLB entry is valid.
		0	TLB entry is not valid.
		1	TLB entry is valid.
1–0	SLST		Size of physical memory covered by the TLB entry.
		00	TLB entry covers an entire section.
		01	TLB entry covers a large page.
		10	TLB entry covers a small page.
		11	TLB entry covers a tiny page.

6.5.12 MMU RAM Entry Registers (RAM_H_REG, RAM_L_REG)

The RAM Entry Registers specify a RAM value to be written into the TLB.

Figure 58. MMU RAM Entry Registers (RAM_H_REG, RAM_L_REG)

RAM_H_REG

31	16	15	0
Reserved		PHYS_TAG_H	
R-0		RW-0	

RAM_L_REG

31	16
Reserved	
R-0	

15	10	9	8	7	0
PHYS_TAG_L		AP		Reserved	
RW-0		RW-0		RW-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 35. MMU MSB RAM Entry Register (RAM_H_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–0	PHYS_TAG_H		These are the most-significant bits of the physical address tag corresponding to the TLB entry. The PHYS_TAG bits correspond to bits 31–10 of the physical memory address. Note that, depending on the page size, not all of the PHYS_TAG bits are needed; these unneeded bits must be written as zeros.

Table 36. MMU LSB RAM Entry Register (RAM_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–10	PHYS_TAG_L		These are the least-significant bits of the physical address tag corresponding to the TLB entry. The PHYS_TAG bits correspond to bits 31–10 of the physical memory address. Note that, depending on the page size, not all of the PHYS_TAG bits are needed; these unneeded bits must be written as zeros.

Table 36. MMU LSB RAM Entry Register (RAM_L_REG) Field Descriptions (Continued)

Bits	Field	Value	Description
9–8	AP		Access permission bits. These bits determine the access permission for the physical memory covered by the TLB entry.
		00 or 01	No access.
		10	Read-only access.
		11	Full access.
7–0	Reserved		These bits are not used.

6.5.13 MMU TLB Global Flush Register (GFLUSH_REG)

The Global Flush Register flushes all TLB entries that are not preserved. When the GLOBAL_FLUSH bit is set, the VALID bit of all entries with PRESERVED = 0 is cleared.

Note:

A global flush does not change the first-level table base address or the victim pointer and base pointer.

Figure 59. MMU TLB Global Flush Register (GFLUSH_REG)

31	1	0
Reserved		GLOBAL_ FLUSH
R-0		RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 37. MMU TLB Global Flush Register (GFLUSH_REG) Field Descriptions

Bits	Field	Value	Description
31–1	Reserved		These bits are not used.
0	GLOBAL_FLUSH		TLB global flush. Setting this bit flushes all TLB entries that are not preserved. After the flush operation has completed, this bit is automatically cleared.
		0	TLB global flush completed.
		1	Flush the TLB.

6.5.14 MMU TLB Entry Flush Register (FLUSH_ENTRY_REG)

The TLB Entry Flush Register deletes individual entries from the TLB. When the FLUSH_ENTRY bit is set, the preserved and valid bits of the entry pointed to by the victim pointer are cleared.

Figure 60. MMU TLB Entry Flush Register (FLUSH_ENTRY_REG)

31	1	0
Reserved		FLUSH_ENTRY
R-0		RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 38. MMU TLB Entry Flush Register (FLUSH_ENTRY_REG) Field Descriptions

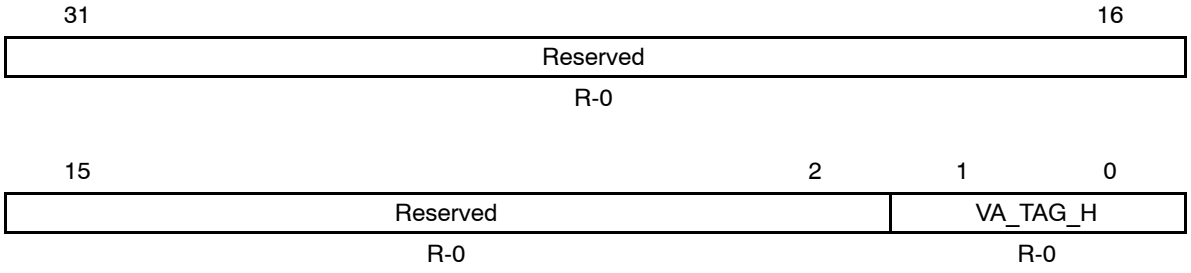
Bits	Field	Value	Description
31–1	Reserved		These bits are not used.
0	FLUSH_ENTRY		TLB entry flush. Setting this bit flushes the entry pointed to by the victim pointer.
		0	The TLB entry flush is complete.
		1	Flush the TLB entry pointed to by the victim pointer.

6.5.15 MMU Read CAM Entry Registers (READ_CAM_H_REG, READ_CAM_L_REG)

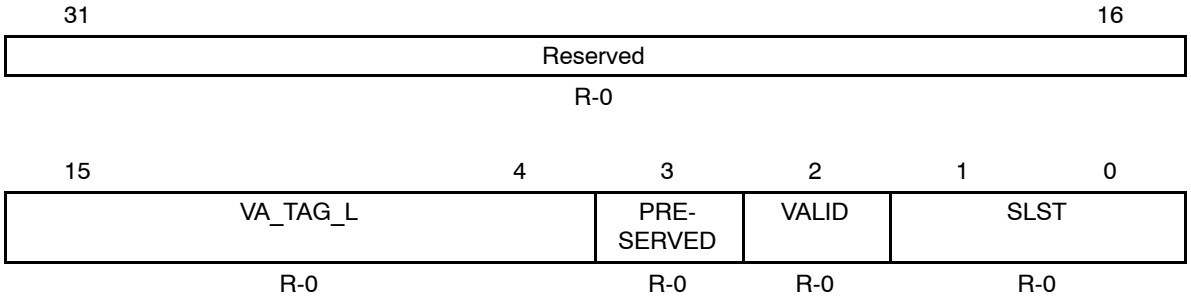
The Read CAM Entry Registers hold the last CAM value read from the TLB.

Figure 61. MMU CAM Entry Read Registers (READ_CAM_H_REG, READ_CAM_L_REG)

READ_CAM_H_REG



READ_CAM_L_REG



Note: R = Read; W = Write; −n = Value after reset; −x = Value after reset is not defined.

Table 39. MMU MSB CAM Entry Read Register (READ_CAM_H_REG) Field Descriptions

Bits	Field	Value	Description
31−2	Reserved		These bits are not used.
1-0	VA_TAG_H		Most-significant bits of the virtual address tag. The VA_TAG bits correspond to bits 23−10 of the DSP virtual address.

Table 40. MMU LSB CAM Entry Read Register (READ_CAM_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–4	VA_TAG_L		Least-significant bits of the virtual address tag. The VA_TAG bits correspond to bits 23–10 of the DSP virtual address.
3	PRESERVED		Preserve bit for the TLB entry. This bit specifies whether the TLB entry should be kept during a TLB global flush.
		0	TLB entry is not preserved during a TLB global flush.
		1	TLB entry is preserved during a TLB global flush.
2	VALID		Valid bit for the TLB entry.
		0	TLB entry is not valid.
		1	TLB entry is valid.
1–0	SLST		Size of physical memory covered by the TLB entry.
		00	TLB entry covers an entire section.
		01	TLB entry covers a large page.
		10	TLB entry covers a small page.
		11	TLB entry covers a tiny page.

6.5.16 MMU Read RAM Entry Registers (READ_RAM_H_REG, READ_RAM_L_REG)

The Read RAM Entry Registers hold the last RAM value read from the TLB.

Figure 62. MMU Read RAM Entry Registers (READ_RAM_H_REG, READ_RAM_L_REG)

READ_RAM_H_REG

31	16	15	0
Reserved		PHYS_TAG_H	
R-0		R-0	

READ_RAM_L_REG

31	16
Reserved	
R-0	

15	10	9	8	7	0
PHYS_TAG_L		AP		Reserved	
R-0		R-0		R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 41. MMU MSB RAM Entry Read Register (READ_RAM_H_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–0	PHYS_TAG_H		These are the most-significant bits of the physical address tag corresponding to the TLB entry. The PHYS_TAG bits correspond to bits 31–10 of the physical memory address.

Table 42. MMU LSB RAM Entry Read Register (READ_RAM_L_REG) Field Descriptions

Bits	Field	Value	Description
31–16	Reserved		These bits are not used.
15–10	PHYS_TAG_L		These are the least-significant bits of the physical address tag corresponding to the TLB entry. The PHYS_TAG bits correspond to bits 31–10 of the physical memory address.
9–8	AP		Access permission bits. These bits determine the access permission for the physical memory covered by the TLB entry.
		00 or 01	No access.
		10	Read-only access.
		11	Full access.
7–0	Reserved		These bits are not used.

6.5.17 MMU Idle Control Register (DSPMMU_IDLE_CTRL)

The Idle Control Register controls the DSP MMU clock.

Figure 63. MMU Idle Control Register (DSPMMU_IDLE_CTRL)

31	2	1	0
Reserved		GL_PDE	AUTOGATING_EN
R-0		R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 43. MMU Idle Control Register (DSPMMU_IDLE_CTRL) Field Descriptions

Bits	Field	Value	Description
31–2	Reserved		These bits are not used.
1	GL_PDE		Global power-down enable bit. This bit is used to shut down the clock feeding the DSP MMU module.
		0	The DSP MMU clock is running.
		1	The DSP MMU clock is disabled.
0	AUTOGATING_EN		Autogating enable bit
		0	Autogating is disabled.
		1	Autogating is enabled.

7 DSP DMA

7.1 Overview

7.1.1 Purpose of the DSP DMA

Acting in the background of DSP core operation, the DMA controller can:

- ❏ Transfer data among internal memory, DSP external memory, and peripherals residing on the DSP public peripheral bus
- ❏ Transfer data between the Microprocessor Unit Interface (MPUI) and memory internal to the DSP subsystem

7.1.2 Features

The DMA controller has the following important features:

- ❏ Operation that is independent of the DSP core
- ❏ Four standard ports, one for each data resource: internal dual-access RAM (DARAM), internal single-access RAM (SARAM), DSP external memory (via the External Memory Interface [EMIF]), and peripherals (via the shared TI peripheral bus bridge)
- ❏ An auxiliary port to enable certain transfers between the MPUI and memory
- ❏ Six channels, which allow the DMA controller to keep track of the context of six independent block transfers among the standard ports
- ❏ Bits for assigning each channel a low priority or a high priority. For details, see section 7.2.6, *Service Chain*.
- ❏ Event synchronization. DMA transfers in each channel can be made dependent on the occurrence of selected events. For details, see section 7.2.12, *Synchronizing Channel Activity*.
- ❏ An interrupt for each channel. Each channel can send an interrupt to the DSP core on completion of certain operational events. See section 7.2.15, *Interrupt Support*.
- ❏ Software-selectable options for updating addresses for the sources and destinations of data transfers.
- ❏ A dedicated idle domain. The DMA controller can be put into a low-power state by turning off this domain. Each multichannel buffered serial port (McBSP) has the ability to temporarily take the DMA domain out of this idle state when the McBSP needs the DMA controller. See *Power Management* in section 7.2.16.

To read about the registers that program the DMA controller, see section 7.3.

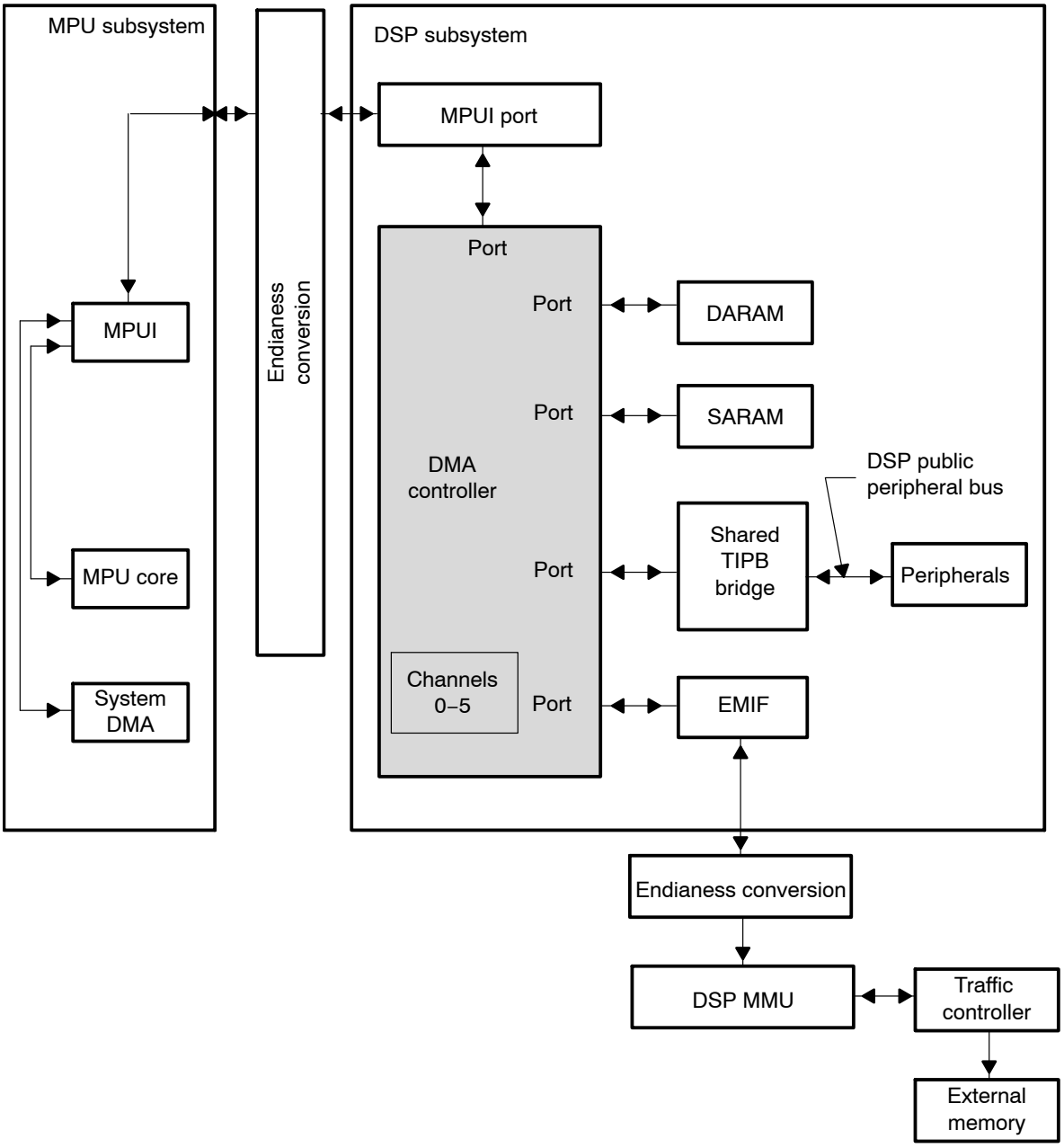
7.1.3 Block Diagram of the DMA Controller

Figure 64 is a conceptual diagram of connections between the DMA controller and other parts of the DSP subsystem. The DMA controller ports in the diagram are:

- Four standard ports. The DMA controller has a standard port for each of the following resources: internal dual-access RAM (DARAM), internal single-access RAM (SARAM), DSP external memory, and peripherals. Data transfers among the standard ports occur in the six DMA channels. (The DMA channels are described in section 7.2.3).
- Auxiliary port. A fifth port supports data transfers between memory and the MPUI. The MPUI cannot access the peripheral port. Transfers between the MPU and the DSP peripherals are supported through a direct connection that does not involve the DSP DMA controller. Transfers between the MPUI and the memory ports are handled by the DMA controller but do not use a DMA channel.

It is possible for multiple channels (or for one or more channels and the MPUI) to request access to the same standard port at the same time. To arbitrate simultaneous requests, the DMA controller has one programmable service chain configuration that is used by each of the standard ports. For details on the service chain, see section 7.2.6.

Figure 64. Conceptual Block Diagram of the DMA Controller Connections



7.2 DSP DMA Controller Architecture

7.2.1 Clock Control

The DSP DMA controller is part of the DSP module within the DSP subsystem (see section 1.2) and thus is clocked by the DSP subsystem clock, DSP_CK. Section 12.2 describes the DSP subsystem clock.

7.2.2 Memory Map

Figure 65 is a high-level memory map for the DSP subsystem data memory space. The diagram shows both the word addresses (23-bit addresses) used by the DSP core and byte addresses (24-bit addresses) used by the DMA controller.

Note:

Word addresses 00 0000h – 00 005Fh (which correspond to byte addresses 00 0000h – 00 00BFh) are reserved for the memory-mapped registers (MMRs) of the DSP core.

Figure 65. High-Level Data Memory Map for DSP Subsystem

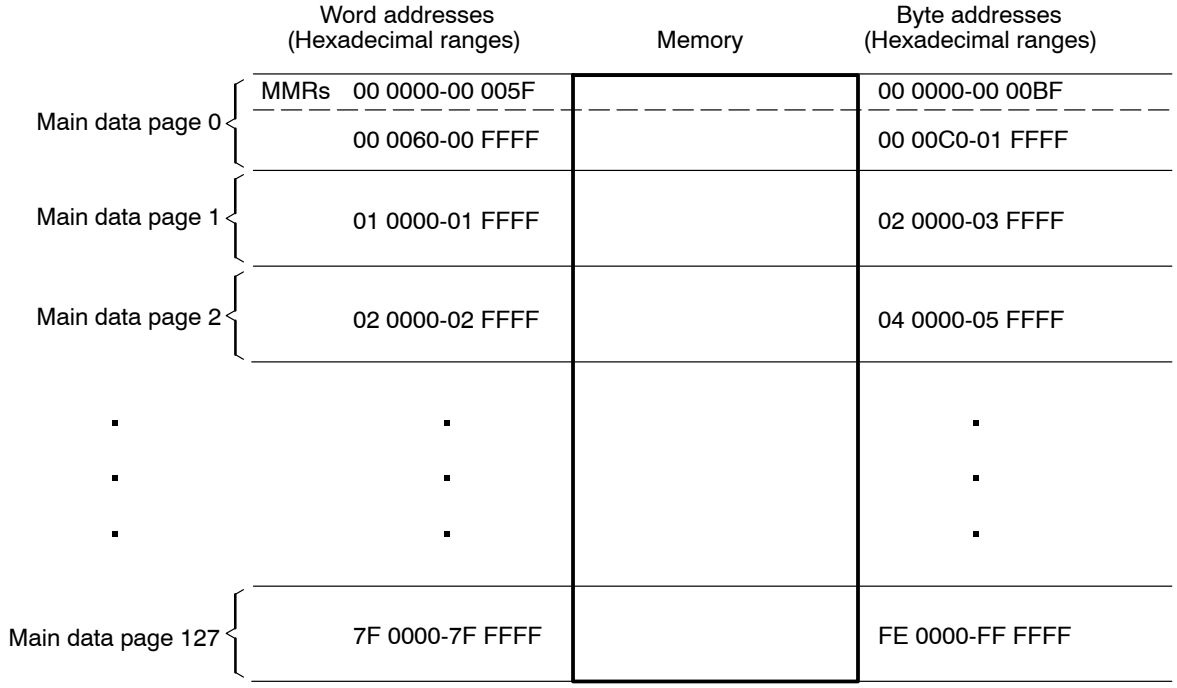


Figure 66 is an I/O space map for the DSP subsystem. The diagram shows both the word addresses (16-bit addresses) used by the DSP core and byte addresses (17-bit addresses) used by the DMA controller.

Note:

The I/O memory map varies from device to device due to the different mixes of peripherals. For a detailed I/O memory map, see the device-specific data manual.

Figure 66. High-Level I/O Memory Map for DSP Subsystem

Word addresses (Hexadecimal range)	I/O space	Byte addresses (Hexadecimal range)
0000-FFFF		0 0000-1 FFFF

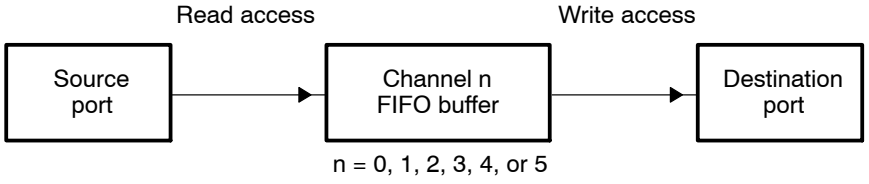
7.2.3 Channels and Port Accesses

The DMA controller has six paths, called channels, to transfer data among the four standard ports (for DARAM, SARAM, DSP external memory, and peripherals). Each channel reads data from one port (from the source) and writes data to that same port or another port (to the destination).

Each channel has a first in, first out (FIFO) buffer that allows the data transfer to occur in two stages (see Figure 67):

- Port read access: Transfer of data from the source port to the channel FIFO buffer.
- Port write access: Transfer of data from the channel FIFO buffer to the destination port.

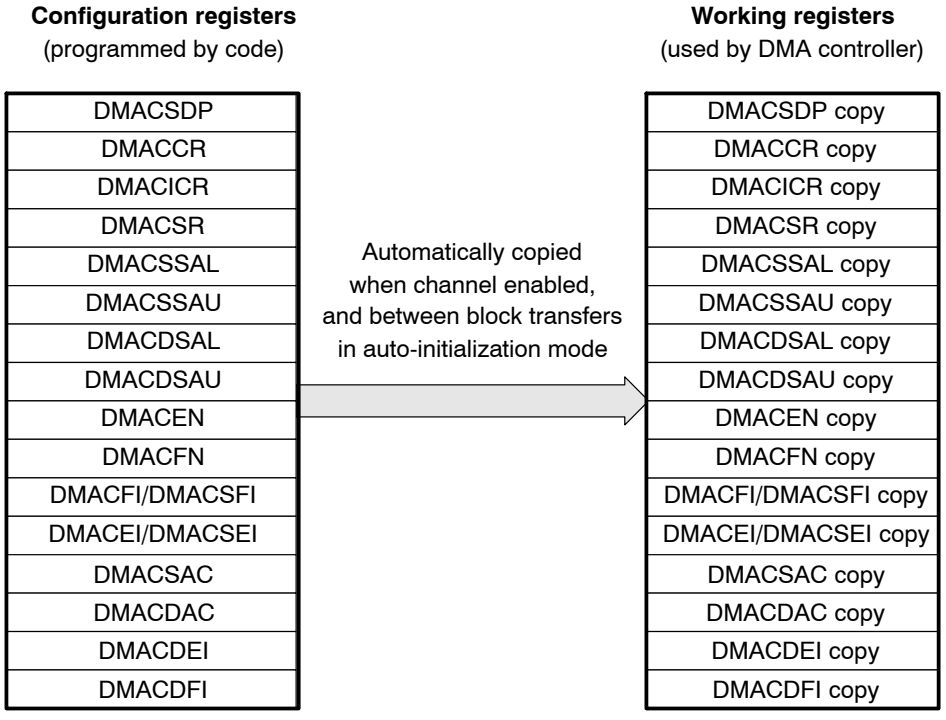
Figure 67. The Two Parts of a DMA Controller Transfer



The set of conditions under which transfers occur in a channel is called the channel context. Each of the six channels contains a register structure for programming and updating the channel context (see Figure 68). The programming code modifies the configuration registers. When it is time for data transferring, the contents of the configuration registers are copied to the working registers, and the DMA controller uses the working register values to control channel activity. The copy from the configuration registers to the working registers occurs whenever the code enables the channel (EN = 1 in DMACCR). In addition, if the auto-initialization mode is on (AUTOINIT = 1 in DMACCR), the copy occurs between block transfers. For more information about the auto-initialization mode, see section 7.2.4.

Some configuration registers can be programmed for the next block transfer while the DMA controller is still running the current context from the working registers. The next transfer will use the new configuration without stopping the DMA controller. The registers DMACSDP, DMACCR, DMACICR, DMACSR, DMAGCR, DMAGSCR, and DMAGTCR should not be configured in this manner. Modification of these registers while the DMA channel is running may cause unpredictable channel operation.

Figure 68. Registers for Controlling the Context of a Channel



7.2.4 Channel Auto-Initialization Capability

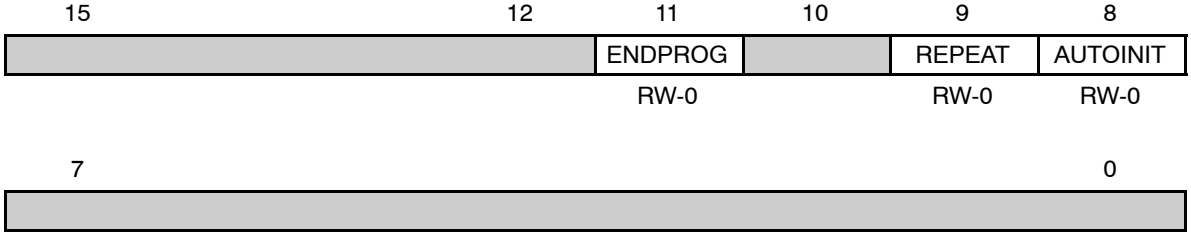
After a block transfer is completed (all of the elements and frames in a block have been moved), the DMA controller automatically disables the channel. If the channel must be used again, the DSP core can reprogram the new channel context and re-enable the DMA channel, or the DMA controller can automatically initialize the new context and re-enable the channel.

When auto-initialization is used, the DMA controller automatically copies the channel context (identical or new) after each block transfer is completed from the configuration registers to the working registers and re-enables the channel, allowing the channel to run again. Auto-initialization is enabled by setting the AUTOINIT bit in the channel controller register (DMACCR).

Two additional bits in DMACCR, REPEAT and ENDPROG, are used during the auto-initialization operation. REPEAT controls whether the DMA controller waits for an indication from the DSP core that the configuration registers are ready to be copied. ENDPROG is a handshaking bit used to communicate between the DSP core and the DMA controller regarding the state of the register copy process. Figure 69 shows DMACCR and Table 44 describes AUTOINIT, REPEAT, and ENDPROG. For a complete description of DMACCR, see section 7.3.5.

There are two methods for using auto-initialization. The same channel context can be repeated on each block transfer, or a new context can be provided for each transfer. The following sections explain these two cases are explained.

Figure 69. DMA Channel Control Register (DMACCR)



Note: R = Read, W = Write, –n = Value after DSP reset

Table 44. DMA Channel Control Register (DMACCR) Field Descriptions

Bits	Field	Value	Description
15–12	Reserved		Reserved
11	ENDPROG		End-of-programming bit. Each DMA channel has two sets of registers: configuration registers and working registers. When block transfers occur repeatedly because of auto-initialization (AUTOINIT = 1), you can change the context for the next DMA transfer by writing to the configuration registers during the current block transfer. At the end of the current transfer, the contents of the configuration registers are copied into the working registers, and the DMA controller begins the next transfer using the new context. For proper auto-initialization, the DSP core must finish programming the configuration registers before the DMA controller copies their contents. The DMA controller automatically clears the ENDPROG bit after copying the configuration registers to the working registers. The DSP core can then program the DMA channel context for the next iteration of the transfer by programming the configuration registers. To ensure that auto-initialization waits for the DSP core, follow this procedure: 1) Make auto-initialization wait for ENDPROG = 1 by clearing the REPEAT bit (REPEAT = 0) 2) Poll for ENDPROG = 0, which indicates that the DMA controller has finished copying the previous context. The configuration registers can now be programmed for the next iteration. 3) Program the configuration registers. 4) Set ENDPROG (ENDPROG = 1) to indicate the end of register programming.
		0	Configuration registers ready for programming/Programming in progress.
		1	End of programming.
10	Reserved		Reserved.

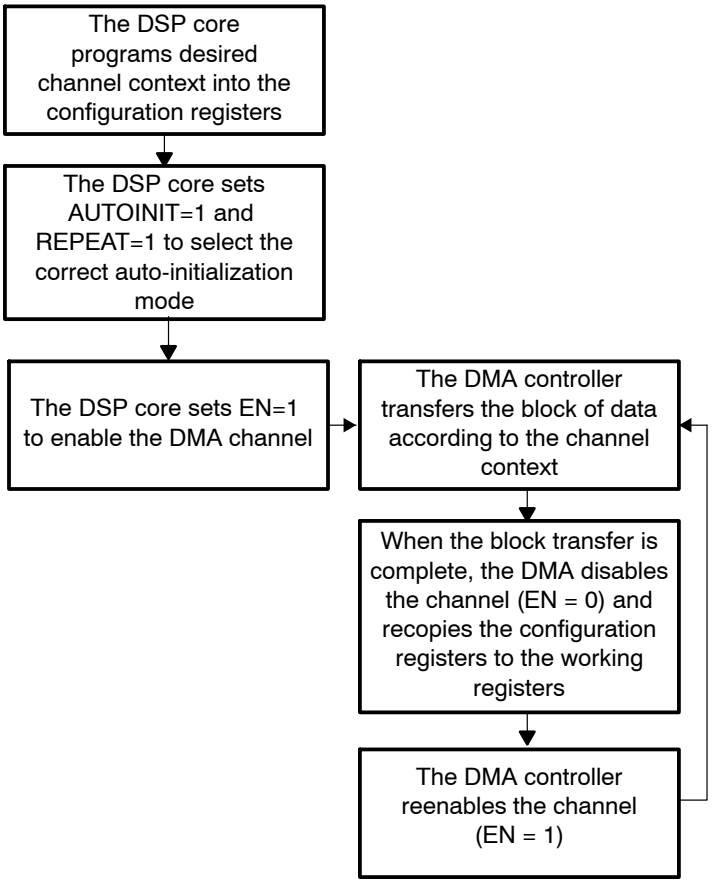
Table 44. DMA Channel Control Register (DMACCR) Field Descriptions (Continued)

Bits	Field	Value	Description
9	REPEAT		Repeat condition bit. If auto-initialization is selected for a channel (AUTOINIT = 1), REPEAT specifies one of two special repeat conditions:
		0	Repeat only if ENDPROG = 1. Once the current DMA transfer is complete, auto-initialization will wait for the end-of-programming bit (ENDPROG) bit to be set.
		1	Repeat regardless of ENDPROG. Once the current DMA transfer is complete, auto-initialization occurs regardless of the ENDPROG bit state.
8	AUTOINIT		Auto-initialization bit. The DMA controller supports auto-initialization, which is the automatic reinitialization of the channel between DMA block transfers. Use AUTOINIT to enable or disable this feature.
		0	Auto-initialization is disabled. Activity in the channel stops at the end of the current block transfer. To stop a transfer immediately, clear the channel enable bit (EN).
		1	Auto-initialization is enabled. Once the current block transfer is complete, the DMA controller reinitializes the channel and starts a new block transfer. To stop activity in the channel you have two options: ❑ To stop activity immediately, clear the channel enable bit (EN = 0). ❑ To stop activity after the current block transfer, clear AUTOINIT (AUTOINIT= 0).
7–0	Reserved		Reserved.

7.2.4.1 Auto-initialization With Unchanging Context

If the desired context for the channel needs to be repeated but does not need to be changed, then the DMA controller is configured with AUTOINIT = 1 and REPEAT = 1. When REPEAT = 1, the DMA controller ignores the state of the ENDPROG handshaking bit. After the DSP core has initially configured the DMA channel, no other DSP core intervention is required to keep the channel running. Figure 70 shows a detailed sequence of events in this mode.

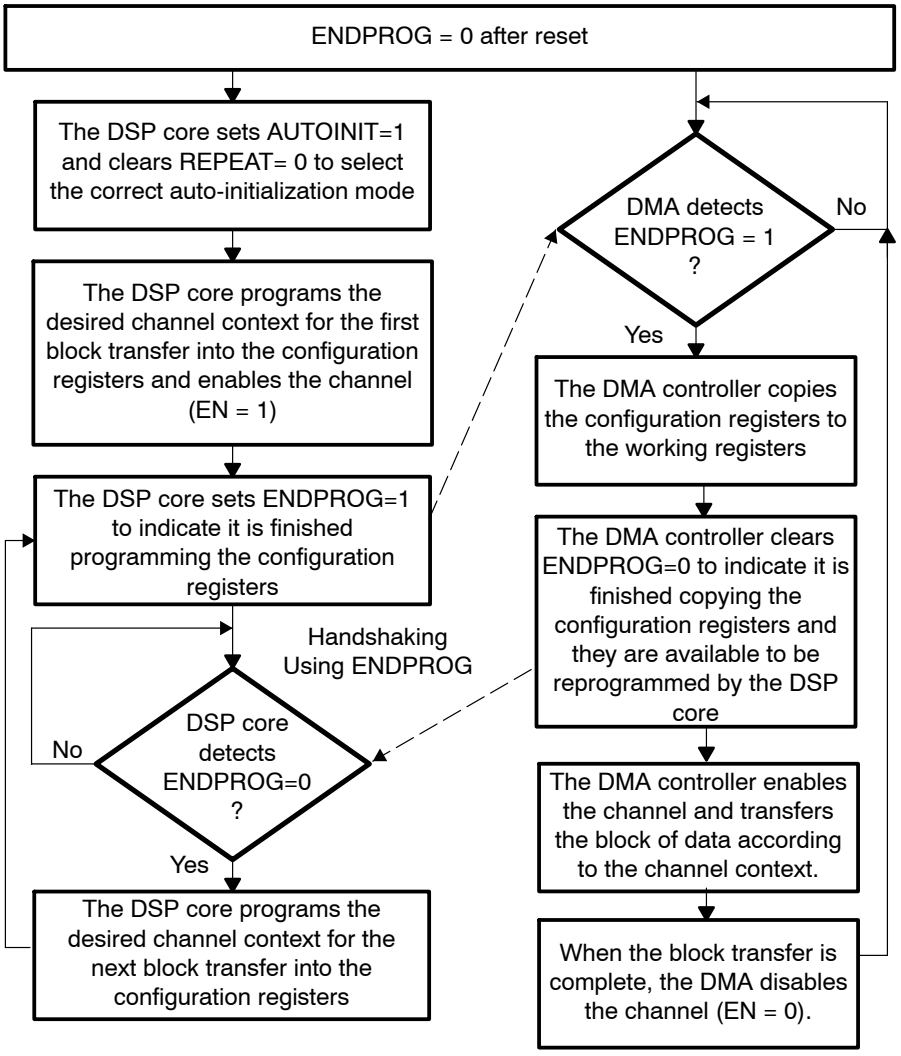
Figure 70. Auto-Initialization Sequence With Unchanging Context (REPEAT = 1)



7.2.4.2 Auto-Initialization With Changing Context

If the desired context for the channel needs to be repeated and is not the same on each block transfer, then the DMA controller must be configured with AUTOINIT = 1 and REPEAT = 0. When REPEAT = 0, the DMA controller waits for the DSP core to write ENDPROG = 1 before it copies the configuration registers. This provides handshaking for the DMA controller to prevent it from copying the registers while they are still being configured by the DSP core. Figure 71 shows a detailed sequence of events in this mode.

Figure 71. Auto-initialization Sequence With Changing Context (REPEAT = 0)

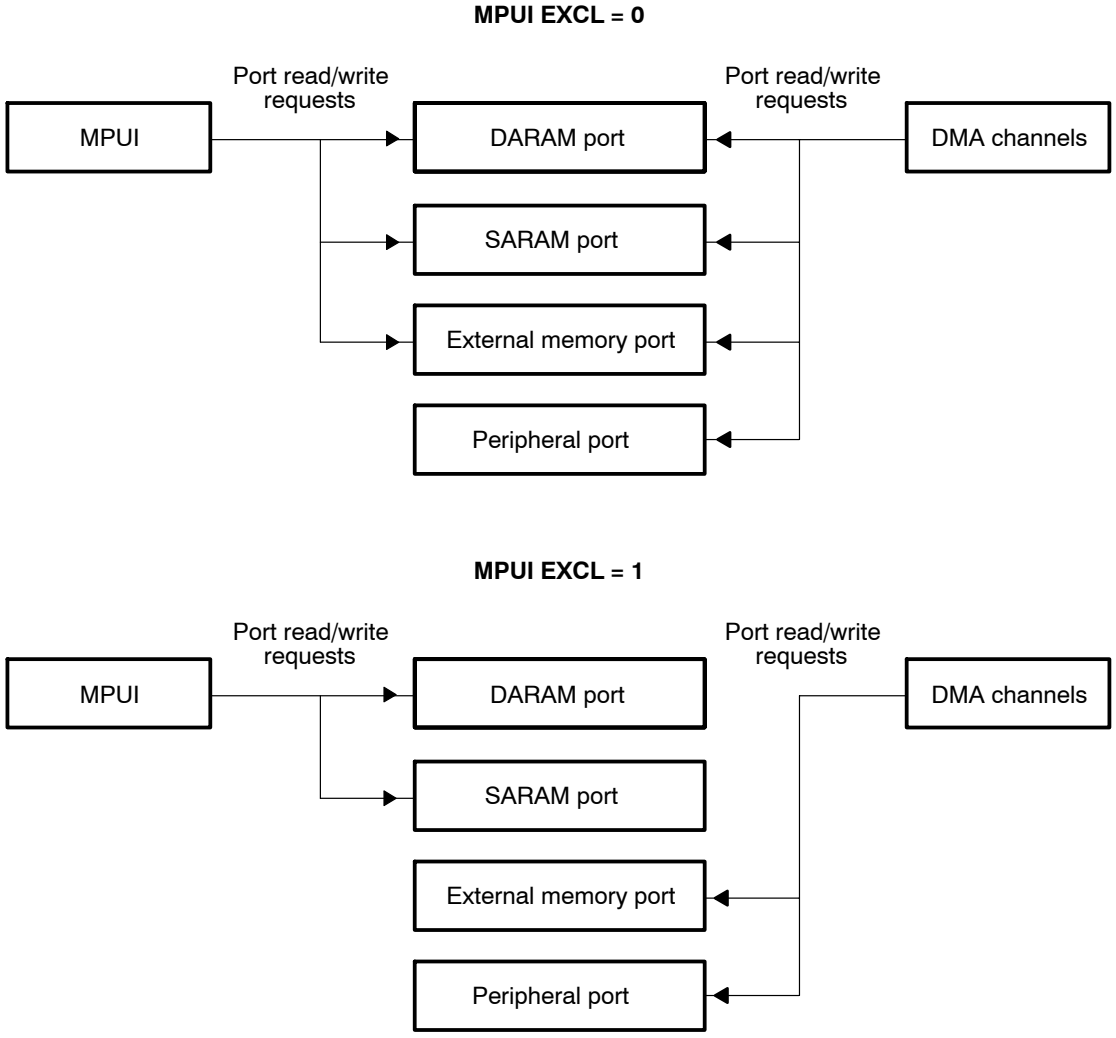


7.2.5 MPUI Access Configurations

As shown in Figure 72, the MPUI_EXCL bit in DMAGCR determines the relationship between the MPUI and the DMA channels:

- When MPUI_EXCL = 0, the MPUI shares memory with the channels.
- When MPUI_EXCL = 1, the MPUI cannot access DSP external memory, but it can access internal RAM without interruptions from the channels. The DARAM port and the SARAM port operate as if all the channels were disconnected from the service chain. Section 7.2.6 describes the service chain.

Figure 72. MPUI Access Configurations

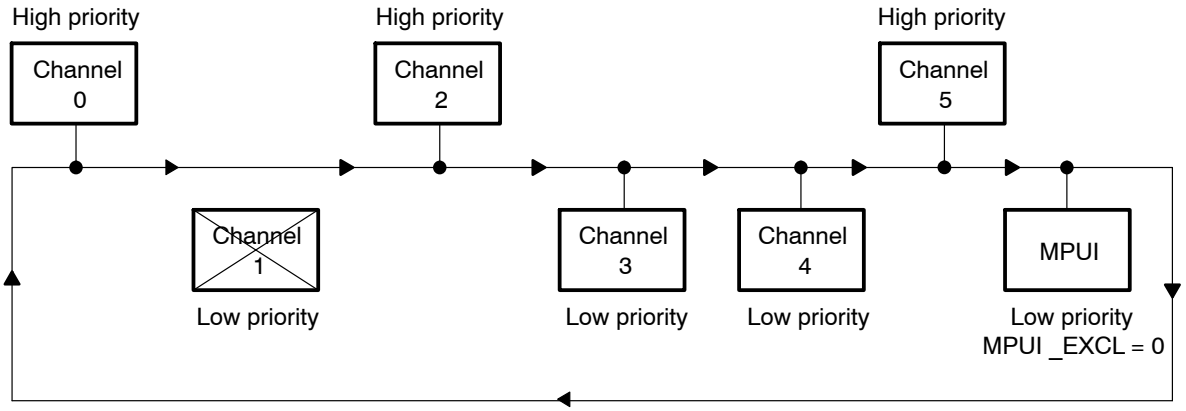


7.2.6 Service Chain

Each of the standard ports can arbitrate simultaneous access requests sent by the six DMA channels and the Microprocessor Unit Interface (MPUI). Each of the standard ports has an independently functioning service chain, which is a software and hardware controlled scheme for servicing access requests. Although the four service chains function independently, they share a common configuration. For example, if channel 2 is disabled, it is disabled in all four ports, and if channel 4 is made a high priority, it is high priority in all four of the ports. One possible configuration for the service chains is shown in Figure 73. Important characteristics of the service chain are listed after the figure.

Section 7.2.6.1 contains an example that shows a service chain configuration applied to three ports.

Figure 73. One Possible Configuration for the Service Chains



- The channels and the MPUI have a programmable priority level. Each channel has a PRIO bit in DMACCR for selecting a high priority or a low priority. The MPUI is assigned a high or low priority with the MPUI_PRIO bit in DMAGCR. The DMA controller only services the low-priority items when all the high-priority items are done or stalled. After a DSP subsystem reset, all channels and the MPUI are low priority.

  In the figure, channels 0, 2, and 5 are high-priority (in each of these channels, PRIO = 1). DMA channels 1, 3, and 4 and the MPUI are low priority (in each of these channels, PRIO = 0, and for the MPUI, MPUI_PRIO = 0).

- The channels and the MPUI have fixed positions in the service chain. The port checks the channels and the MPUI in a repeating circular sequence: 0, 1, 2, 3, 4, 5, MPUI, 0, 1, 2, 3, 4, 5, MPUI, and so on. At each position in the service chain, the port checks whether the channel/MPUI is ready and able to be serviced. If the channel is ready to be serviced and a higher priority request is not pending in another channel, it is serviced; otherwise, the port skips to the next position. After a DSP subsystem reset, the port restarts its circular sequence, beginning with channel 0.

- The channels can be individually connected or disconnected from the service chain through software. If a channel is enabled (EN = 1 in DMACCR), it is connected to the service chain; if it is disabled (EN = 0), it is disconnected. After a DSP subsystem reset, all channels are disconnected. In the figure, only channel 1 is disconnected. As a port checks the channels and the MPUI in its repeating circular sequence, it will keep skipping channel 1 until the channel is reconnected.

  The MPUI cannot access the peripheral port. The peripheral port operates as if the MPUI is disconnected from the service chain.

- Writing a 1 to the MPUI_EXCL bit in DMAGCR gives the MPUI exclusive access to the DARAM and SARAM ports. The DARAM and SARAM ports operate as if only the MPUI is connected to the service chain (as if none of the channels are connected, regardless of whether the channels are enabled). For more details, see MPUI Access Configurations in section 7.2.5.

  In the figure, MPUI_EXCL = 0. The MPUI shares the RAM ports with the channels.

- If a channel is tied to a synchronization event, the channel does not generate a DMA request (and, therefore, cannot be serviced) until the synchronization event occurs.

7.2.6.1 Service Chain Example

Figure 74 shows a DMA service chain configuration applied to the DARAM port, the DSP external memory port, and the peripheral port. Each service chain has the following programmed characteristics.

- Channels 0, 2, and 5 are high-priority (PRIO = 1 in DMACCR). Channels 1, 3, and 4 are low-priority (PRIO = 0).

- Channels 1, 2, and 4 are enabled (EN = 1 in DMACCR). Channels 0, 3, and 5 are disabled (EN = 0).

- The MPUI is sharing the internal memory with the channels (MPUI_EXCL = 0 in DMAGCR) and is treated like a low-priority channel (MPUI_PRIO = 0 in DMAGCR). Notice that the MPUI is shown as disconnected in the peripheral port. This is because the MPUI cannot access the peripheral port.

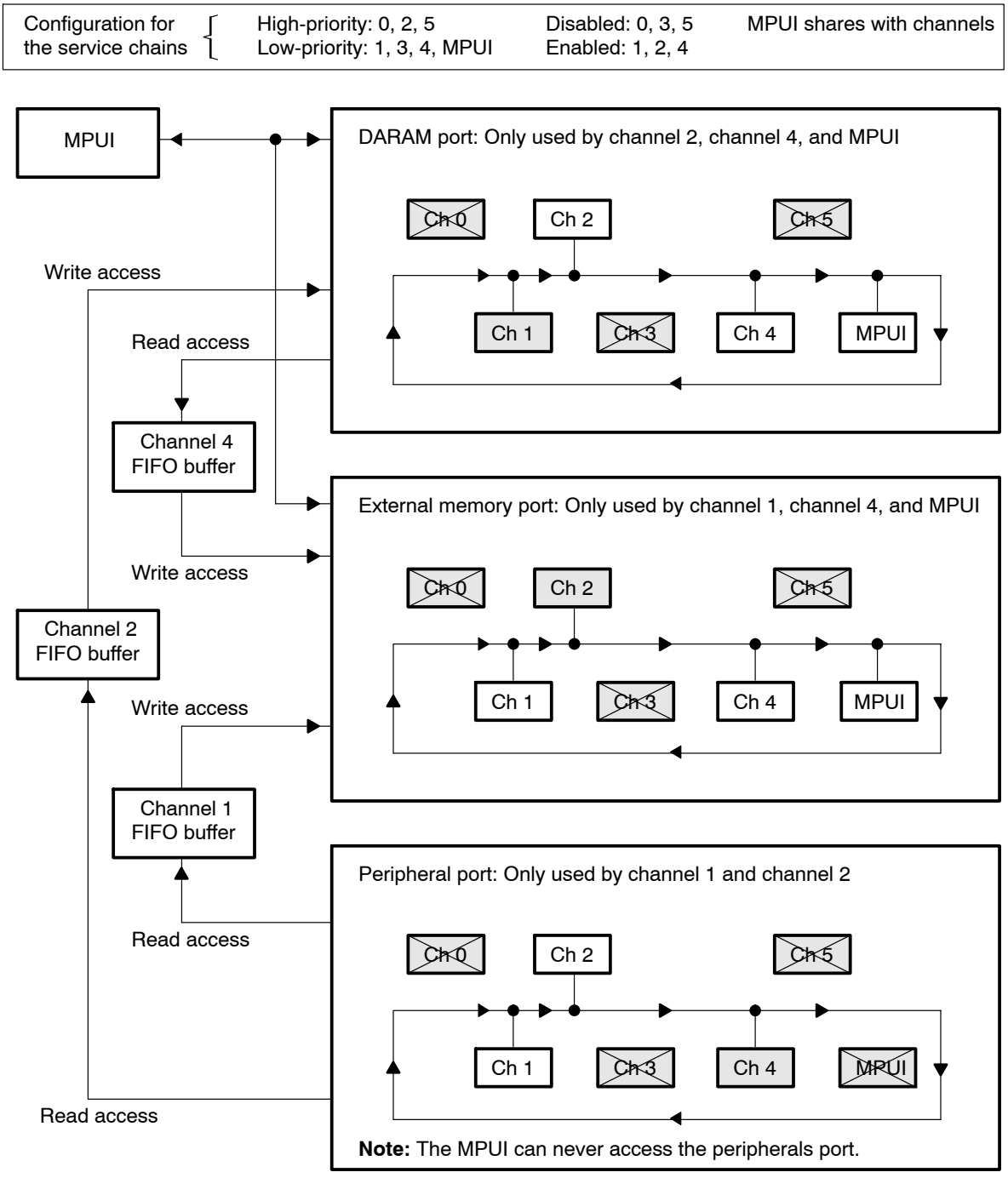
Table 45 summarizes the activity at the ports in Figure 74.

Table 45. Activity Shown in Figure 74

Port	This Port Arbitrates
DARAM	Write access requests from channel 2 Read access requests from channel 4 Read or write access requests from the MPUI
External Memory	Write access requests from channel 1 Write access requests from channel 4 Read or write access requests from the MPUI
Peripheral	Read access requests from channel 1 Read access requests from channel 2

Finally, notice that for each port in the figure, there is a channel that is connected to the service chain but does not use the port. For example, the peripheral port is not used by channel 4. If channel 4 were redefined to include the peripheral port as source or destination, the port would handle channel 4 according to its position and priority in the service chain.

Figure 74. Service Chain Applied to Three DMA Ports



7.2.7 Units of Data: Byte, Element, Frame, and Block

This documentation on the DMA controller refers to data in four levels of granularity:

- Byte: An 8-bit value. A byte is the smallest unit of data transferred in a DMA channel.
- Element: One or more bytes to be transferred as a unit. Depending on the programmed data type, an element is an 8-bit, 16-bit, or a 32-bit value. An element transfer cannot be interrupted; all of its bytes are transferred to a port before another channel or the MPUI can take control of the port.
- Frame: One or more elements to be transferred as a unit. A frame transfer can be interrupted between element transfers.
- Block: One or more frames to be transferred as a unit. Each channel can transfer one block of data (once or multiple times). A block transfer can be interrupted between frame transfers and element transfers.

For each of the six DMA channels, you can define the number of frames in a block (with DMACFN), the number of elements in a frame (with DMACEN), and the number of bytes in an element (with the DATATYPE bits in DMACSDP). For descriptions of DMACFN, DMACEN, DMACSDP, and other registers of the DMA controller, see section 7.3.

7.2.8 Start Address in a Channel

During a data transfer in a DMA channel, the first address at which data is read is called the source start address. The first address to which the data is written is called the destination start address. These are byte addresses. From the standpoint of the DMA controller, every 8 bits in memory or I/O space has its own address. Each channel contains the following registers for specifying the start addresses:

Table 46. Registers Used to Define the Start Addresses for a DMA Transfer

Register	Load With:
DMACSSAL	Source start address (lower part)
DMACSSAU	Source start address (upper part)
DMACDSAL	Destination start address (lower part)
DMACDSAU	Destination start address (upper part)

The following sections explain how to load the start address registers for memory accesses and I/O accesses. The DMA controller can access all of the internal and DSP external memory and all of I/O space (which contains registers for the DSP peripherals).

7.2.8.1 Start Address in DSP Subsystem Data Memory

Figure 65 is a high-level memory map for the DSP subsystem. The diagram shows both the word addresses (23-bit addresses) used by the DSP core and byte addresses (24-bit addresses) used by the DMA controller. To load the source/destination start address registers:

1) Identify the correct start address. Check for any alignment constraint for the data type; see the description for the DATATYPE bits of DMACSDP (section 7.3.7). If you have a word address, shift it left by 1 bit to form a byte address with 24 bits. For example, word address 02 4000h should be converted to byte address 04 8000h.

2) Load the 16 least significant bits (LSBs) of the byte address into DMACSSAL (for source) or DMACDSAL (for destination).

3) Load the 8 most significant bits (MSBs) of the byte address into the 8 LSBs of DMACSSAU (for source) or DMACDSAU (for destination).

Note:

Word addresses 00 0000h-00 005Fh (which correspond to byte addresses (00 0000h-00 00BFh) are reserved for the memory-mapped registers (MMRs) of the DSP core.

7.2.8.2 Start Address in I/O Space

Figure 66 is an I/O space map for the DSP subsystem. The diagram shows both the word addresses (16-bit addresses) used by the DSP core and byte addresses (17-bit addresses) used by the DMA controller. To load the source/destination start address registers:

1) Identify the correct start address. Check for any alignment constraint for the data type; see the description for the DATATYPE bits of DMACSDP (section 7.3.7). If you have a word address, shift it left by 1 bit to form a byte address with 17 bits. For example, word address 8000h should be converted to byte address 1 0000h.

2) Load the 16 least significant bits (LSBs) of the byte address into DMACSSAL (for source) or DMACDSAL (for destination).

3) Load the most significant bit (MSB) of the byte address into the LSB of DMACSSAU (for source) or DMACDSAU (for destination).

7.2.9 Updating Addresses in a Channel

During data transfers in a DMA channel, the DMA controller begins its read and write accesses at the start addresses you specify (see section 7.2.8). In many cases, these addresses must be updated so that data is read and written at consecutive or indexed locations after a data transfer has begun. You can configure address updates at two levels:

- Block-level address updates. In the auto-initialization mode (AUTOINIT = 1 in DMACCR), block transfers can occur one after another until you turn off auto-initialization or disable the channel. If you want different start addresses for the block transfers, you can update the start addresses between the block transfers.
- Element-level address updates. You can have the DMA controller update the source address and/or the destination address after each element transfer. At the end of an element transfer, the source address held by the DMA controller is the address of the last byte that was read from the source. Likewise, after a transfer, the destination address held by the DMA controller is the address of the last byte that was modified at the destination. Through software control, you can make sure the source address points to the start of the next element, and you can make sure the element will be precisely positioned at the destination. Choose an addressing mode for the source with the SRCAMODE bits in DMACCR. Choose an addressing mode for the destination with the DSTAMODE bits in DMACCR. If you choose a single-index or double-index addressing mode, you must load the appropriate index register or registers (see section 7.3.11).

7.2.10 Data Packing Capability

The five DMA controller ports have various widths and support various sizes of data accesses.

Table 47. DMA Controller Ports

DMA Port	Port Width	Access Sizes Supported (bytes)
SARAM	32	2, 4
DARAM	32	2, 4
EMIF	32	1, 2, 4
Peripheral	16	2, 4
MPUI	16	2

A DMA channel has the ability to pack and unpack.

- Pack: Pack several consecutive element transfers into wider accesses. For example, if the element size is 16 bits, the 32-bit-wide SARAM port can pack two accesses so that 4 bytes at a time are written into the channel FIFO. Packing effectively reduces the frequency at which that channel must be serviced by the port service chain. This can reduce overhead and improve channel throughput in some cases. Packing options are determined by port access capabilities and element size. Packing at the source or destination port can be disabled by software control.
- Unpack: Split a single element transfer into several byte accesses. This occurs when the DMA port size is less than the size of the element type. For example, when the element size is programmed as 32 bits and the destination port is 16 bits wide, the transfer write operation is split into two 16-bit write operations.

The value programmed for channel element size determines the width of the read access at the source port and write access at the destination port. If the element size is smaller than the source port width and source packing is enabled, the source port controller receives 4 bytes per service-chain request. This reduces overhead because four times as much data is written into the channel FIFO per service-chain cycle. Similarly, if packing is enabled at the destination port, 4 bytes are written per iteration of the destination port service chain.

The DMA controller performs packing as shown in Table 48.

Table 48. DMA Controller Data Packing

Data Type	Port Bus Size	Data Packing
8-bit	16-bit	Two data values packed into 16 bits
8-bit	32-bit	Four data values packed into 32 bits
16-bit	32-bit	Two data values packed into 32 bits

7.2.11 Data Burst Capability

Data bursts can be used to improve DMA controller throughput if one or both of the ports associated with the DMA channel support burst capability. When bursting is enabled, the DMA controller executes a burst of four elements each time a channel is serviced instead of moving a single element. The SARAM and DARAM ports support burst capability. The EMIF port will burst data only if the traffic controller memory interface used to fetch the data supports bursting. The traffic controller memory interfaces include the internal memory interface (IMIF on OMAP5910 and OCP-T1 on OMAP5912), the external memory interface slow (EMIFS), and external memory interface fast (EMIFF). If the traffic controller interface does not support bursting, the DMA controller will perform four single accesses to move the burst of data. The peripheral port does not support burst capability; therefore, the DMA controller will perform four single peripheral port accesses to move the burst data. More information on the traffic controller interfaces can be found in the *OMAP5910 Dual-Core Processor Memory Interface Traffic Controller Reference Guide* (SPRU673) and the *OMAP5912 Multimedia Processor OMAP3.2 Subsystem Reference Guide* (SPRU749).

If bursting is used, the start addresses for the source and destination should be aligned on a burst boundary. Burst boundaries correspond to byte addresses with 0h as the least significant four bits.

To use bursting, the following conditions should be met:

- The start address for the port with bursting enabled should be on a burst boundary.
- The element index should be 1.
- The frame index should cause each burst access to align on a burst boundary.
- The result of the equation (Element number x Element size) should align on a burst boundary. This means at the end of each frame, the address should be aligned on a burst boundary.

If both the source and destination have bursting enabled, but the source address does not start on a burst boundary, the source burst will be automatically disabled internally. The source will load the channel FIFO and, when enough data is available, a destination burst will be executed. If the destination does not start on a burst boundary, the destination accesses will be performed as single accesses.

If the frame size is not a multiple of 4 elements, the remaining 1 to 3 elements at the end of the frame will be transferred in single (nonburst) accesses.

Burst mode is not supported when the source and destination are both configured to be the EMIF port.

7.2.12 Synchronizing Channel Activity

Activity in a channel can be synchronized to a request from a DSP public peripheral or MPU/DSP shared peripheral. Each peripheral request is tied to a DMA synchronization event. You can use a DMA synchronization event to trigger channel activity using the SYNC bits of DMACCR.

Each channel has an FS bit in DMACCR that allows you to choose between two synchronization modes:

- Element synchronization mode (FS = 0) requires one event per element transfer. When the selected synchronization event occurs, a read access request is sent to the source port and then a write access request is sent to the destination port. When all the bytes of the current element are transferred, the channel makes no more requests until the next synchronization event occurrence.
- Frame synchronization mode (FS = 1) requires one event to trigger an entire frame of elements. When the event occurs, the channel sends a read access request and a write access request for each element in the frame. When all the elements are transferred, the channel makes no more requests until the next occurrence of the event.

If you specify a synchronization event, the DMA access to the source and destination are handled as described in section 7.2.12.1. Once the request is received, it is handled according to the predefined position and the programmed priority of the channel in the DMA service chain (section 7.2.6).

If you choose not to synchronize the channel (SYNC = 00000b), the channel sends an access request to the source port as soon as the channel is enabled (EN = 1 in DMACCR). Setting EN = 1 initiates the transfer of the entire block defined for the channel.

If the DMA controller is configured to recognize a synchronization event (SYNC is something other than 00000b) and the synchronization event occurs before the channel is enabled, the synchronization event will be latched and serviced as soon as the channel is enabled. Set the SYNC field to 00000b while the channel is disabled to ignore the synchronization events that occur before the channel is enabled.

7.2.12.1 DMA Channel Read Synchronization vs. Write Synchronization

When a DMA channel is configured for synchronization, the synchronization event is tied to the element read operation or the element write operation depending on the source and destination ports.

There are three general cases (see Table 49).

- Case 1: Source port is peripheral; destination port is SARAM, DARAM, or EMIF.

  The channel waits for the synchronization event before reading from the peripheral port into the channel FIFO (source synchronization). Once the FIFO is filled, the DMA channel begins writing to the destination port to empty the FIFO.

- Case 2: Source port is SARAM, DARAM, or EMIF; destination is peripheral.

  As soon as the channel is enabled (EN = 1 in DMACCR), reads from the source port are performed to load the channel FIFO. The FIFO writes to the peripheral port do not begin until the synchronization event is detected (destination synchronization). When the channel is operating in frame-synchronization mode (FS = 1 in DMACCR), several prereads may occur that might fill the FIFO while the channel is awaiting the synchronization event.

- Case 3: Source port is SARAM, DARAM, or EMIF; destination port is SARAM, DARAM, or EMIF.

  The channel waits for the synchronization event before reading from the source port into the channel FIFO (source synchronization). Once the FIFO is filled, the DMA channel begins writing to the destination port to empty the FIFO.

Table 49. Read/Write Synchronization

SYNC Field of DMACCR	Source Port	Destination Port	Synchronization Event Triggers
00000b	Any port	Any port	No activity
Not 00000b	Peripheral port	SARAM, DARAM or EMIF port	Source read
Not 00000b	SARAM, DARAM or EMIF port	Peripheral port	Destination write
Not 00000b	SARAM, DARAM or EMIF port	SARAM, DARAM or EMIF port	Source read

7.2.12.2 Checking the Synchronization Status

Each channel has a synchronization flag (SYNC) in its status register, DMACSR. When the synchronization event occurs, the DMA controller sets the flag (SYNC = 1). The flag is cleared (SYNC = 0) when the DMA controller has completed the first read access (transfer from source port to channel FIFO) after receiving synchronization.

7.2.12.3 Dropped Synchronization Events

If a synchronization event occurs in a channel before the DMA controller is done servicing the previous event in that channel (before the DMA controller clears the SYNC bit in DMACSR), a synchronization event has been *dropped*. The DMA controller responds to an event drop in the following manner:

- After the current element transfer, the DMA controller disables the channel (EN = 0 in DMACCR) and activity in the channel stops.
- If the corresponding interrupt enable bit is set (DROPIE = 1 in DMACICR), the DMA controller also sets the event drop status bit (DROP = 1 in DMACSR) and sends an interrupt request to the DSP core. For more details on interrupts, see section 7.2.15.

7.2.12.4 Synchronization Event Sources

Table 50 and Table 51 list the source for each synchronization event on the OMAP5910 and OMAP5912 devices, respectively. To have a DMA channel transfer data based on a specific synchronization event, set the SYNC bits of the DMACCR to the value listed in these tables.

Note:

On the OMAP5910, the synchronization events of the DSP DMA controller are tied to the peripheral requests specified in Table 50. However, on the OMAP5912, the source for each synchronization event can be changed using the DSP GDMA Handler, as described in section 7.2.13. For compatibility with OMAP5910, the OMAP5912 DSP GDMA Handler defaults to the configuration shown in Table 51 after the DSP subsystem is reset.

Table 50. DSP DMA Controller Synchronization Events for OMAP5910

DMA Event	Generated By (On OMAP5910):	Setting For SYNC Bits
DMA_EVT1	MCSI1 Transmit Request (MCSI1TX)	00001b
DMA_EVT2	MCSI1 Receive Request (MCSI1RX)	00010b
DMA_EVT3	MCSI2 Transmit Request (MCSI2TX)	00011b
DMA_EVT4	MCSI2 Receive Request (MCSI2RX)	00100b
DMA_EVT5	MPU Request (through MPUIO2)	00101b
DMA_EVT6	MPU Request (through MPUIO4)	00110b
DMA_EVT7	Reserved	00111b
DMA_EVT8	McBSP1 Transmit Request (McBSP1TX)	01000b
DMA_EVT9	McBSP1 Receive Request (McBSP2RX)	01001b
DMA_EVT10	McBSP3 Transmit Request (McBSP3TX)	01010b
DMA_EVT11	McBSP3 Receive Request (McBSP3RX)	01011b
DMA_EVT12	UART1 Transmit Request (UART1TX)	01100b
DMA_EVT13	UART1 Receive Request (UART1RX)	01101b
DMA_EVT14	UART2 Transmit Request (UART2TX)	01110b
DMA_EVT15	UART2 Receive Request (UART2RX)	01111b
DMA_EVT16	Reserved	10000b
DMA_EVT17	Reserved	10001b
DMA_EVT18	UART3 Transmit Request (UART3TX)	10010b
DMA_EVT19	UART3 Receive Request (UART3RX)	10011b

Table 51. DSP DMA Controller Synchronization Events for OMAP5912†

DMA Event	Generated By (On OMAP5912):	Setting For SYNC Bits
DMA_EVT1	MCSI1 Transmit Request (MCSI1TX)	00001b
DMA_EVT2	MCSI1 Receive Request (MCSI1RX)	00010b
DMA_EVT3	MCSI2 Transmit Request (MCSI2TX)	00011b
DMA_EVT4	MCSI2 Receive Request (MCSI2RX)	00100b
DMA_EVT5	MMC/SDIO2 Transmit Request (MMC/SDIO2TX)	00101b
DMA_EVT6	MMC/SDIO2 Receive Request (MMC/SDIO2RX)	00110b
DMA_EVT7	Free‡	00111b
DMA_EVT8	McBSP1 Transmit Request (McBSP1TX)	01000b
DMA_EVT9	McBSP1 Receive Request (McBSP2RX)	01001b
DMA_EVT10	McBSP3 Transmit Request (McBSP3TX)	01010b
DMA_EVT11	McBSP3 Receive Request (McBSP3RX)	01011b
DMA_EVT12	UART1 Transmit Request (UART1TX)	01100b
DMA_EVT13	UART1 Receive Request (UART1RX)	01101b
DMA_EVT14	UART2 Transmit Request (UART2TX)	01110b
DMA_EVT15	UART2 Receive Request (UART2RX)	01111b
DMA_EVT16	I2C Receive Request (I2CRX)	10000b
DMA_EVT17	I2C Transmit Request (I2CTX)	10001b
DMA_EVT18	UART3 Transmit Request (UART3TX)	10010b
DMA_EVT19	UART3 Receive Request (UART3RX)	10011b

† This table lists the default mapping for the DMA synchronization events after a DSP subsystem reset. The source for each synchronization event can be changed through the GDMA Handler.

‡ After a DSP subsystem reset, DMA_EVT7 is not associated with any particular peripheral request. You can use the Functional Multiplexing DSP DMA Register B to associate a peripheral request (from Table 52) with this DMA synchronization event.

7.2.13 DSP GDMA Handler (OMAP5912 Only)

7.2.13.1 Operation

As mentioned in section 7.2.12.4, the source for a synchronization event can be changed on the OMAP5912 DSP DMA. Altogether, the DSP public peripherals and MPU/DSP shared peripherals can generate up to 28 different peripheral requests. The OMAP5912 DSP GDMA Handler acts as a crossbar which maps these incoming peripheral requests to the 19 DSP DMA synchronization events (see Figure 75).

Figure 75. DSP GDMA Handler

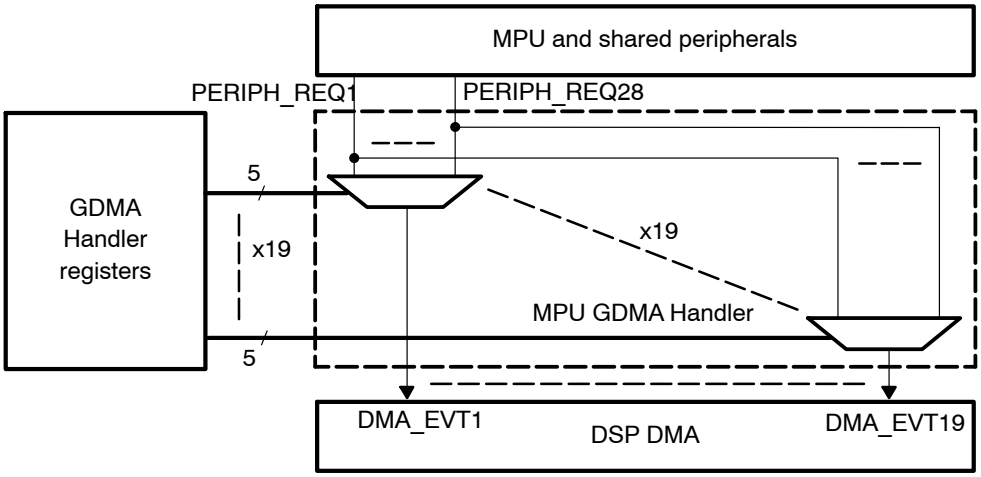


Table 52 lists the source of the 28 input lines of the GDMA Handler.

Table 52. DSP GDMA Handler Input Request Lines

Input Request Line	Generated By:
PERIPH_REQ1	MCSI1 Transmit Request (MCSI1TX)
PERIPH_REQ2	MCSI1 Receive Request (MCSI1RX)
PERIPH_REQ3	MCSI2 Transmit Request (MCSI2TX)
PERIPH_REQ4	MCSI2 Receive Request (MCSI2RX)
PERIPH_REQ5	MMC/SDIO2 Transmit Request (MMC/SDIO2TX)
PERIPH_REQ6	MMC/SDIO2 Receive Request (MMC/SDIO2RX)
PERIPH_REQ7	Reserved
PERIPH_REQ8	McBSP1 Transmit Request (McBSP1TX)
PERIPH_REQ9	McBSP1 Receive Request (McBSP2RX)
PERIPH_REQ10	McBSP3 Transmit Request (McBSP3TX)
PERIPH_REQ11	McBSP3 Receive Request (McBSP3RX)
PERIPH_REQ12	UART1 Transmit Request (UART1TX)
PERIPH_REQ13	UART1 Receive Request (UART1RX)
PERIPH_REQ14	UART2 Transmit Request (UART2TX)
PERIPH_REQ15	UART2 Receive Request (UART2RX)
PERIPH_REQ16	I2C Receive Request (I2CRX)
PERIPH_REQ17	I2C Transmit Request (I2CTX)
PERIPH_REQ18	UART3 Transmit Request (UART3TX)

Table 52. DSP GDMA Handler Input Request Lines (Continued)

Input Request Line	Generated By:
PERIPH_REQ19	UART3 Receive Request (UART3RX)
PERIPH_REQ20	CMT-APE Transmit Request , channel 1 (CMTAPE1TX)
PERIPH_REQ21	CMT-APE Receive Request, channel 1 (CMTAPE1RX)
PERIPH_REQ22	CMT-APE Transmit Request, channel 2 (CMTAPE2TX)
PERIPH_REQ23	CMT-APE Receive Request, channel 2 (CMTAPE2RX)
PERIPH_REQ24	NAND Flash End of Burst (NANDEB)
PERIPH_REQ25	SPI Transmit Request (SPITX)
PERIPH_REQ26	SPI Receive Request (SPIRX)
PERIPH_REQ27	McBSP2 Transmit Request (McBSP2TX)
PERIPH_REQ28	McBSP2 Receive Request (McBSP2RX)

7.2.13.2 Configuration

Each peripheral request can be mapped to a specific DMA synchronization event through the Functional Multiplexing DSP DMA registers. Each register contains a 5-bit field associated with each DMA synchronization event. The value written to this field represents the peripheral request associated with that synchronization event.

To associate a specific peripheral request with a DMA event, follow these steps:

1) Using Table 53, identify which functional multiplexing register controls the desired DMA event.

2) Determine the number of the peripheral request you want to associate with the DMA event using Table 52.

3) Write the peripheral request number as n-1, not n, to the bit field for the desired DMA event within the functional multiplexing register.

For example, to associate the McBSP3 transmit request with DMA event number 6 (DMA_EVT6), write 0x09 to the CONF_DSP_DMA_EVT_06 field of the FUNC_MUX_DSP_DMA_A register.

After reset, the OMAP5912 DSP GDMA Handler is configured to the default configuration shown in Table 51. However, the handler gives you the flexibility to specify the peripheral request associated with each of the 19 DMA events according to your application task requirements.

7.2.13.3 Registers

Table 53 lists the registers associated with the OMAP5912 DSP GDMA Handler. These registers can be accessed by the MPU through the OMAP15912 configuration module. The descriptions for each register follow the table.

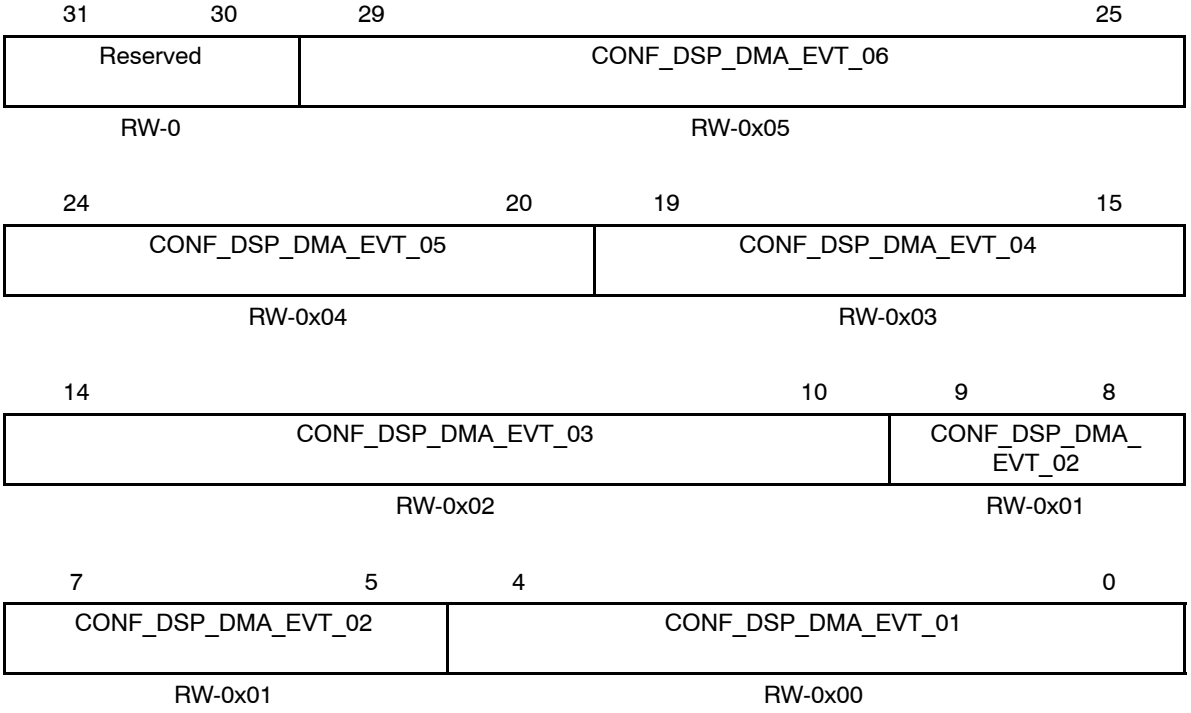
Table 53. Registers of the OMAP5912 DSP GDMA Handler

Name	Description	MPU Byte Address
FUNC_MUX_DSP_DMA_A	Functional Multiplexing DSP DMA Register A Controls the mapping for DSP DMA events 1 through 6	0xFFFE 10D0
FUNC_MUX_DSP_DMA_B	Functional Multiplexing DSP DMA Register B Controls the mapping for DSP DMA events 7 through 12	0xFFFE 10D4
FUNC_MUX_DSP_DMA_C	Functional Multiplexing DSP DMA Register C Controls the mapping for DSP DMA events 13 through 18	0xFFFE 10D8
FUNC_MUX_DSP_DMA_D	Functional Multiplexing DSP DMA Register D Controls the mapping for DSP DMA event 19	0xFFFE 10DC

Functional Multiplexing DSP DMA Register A (FUNC_MUX_DSP_DMA_A)

This global control register (see Figure 76 and Table 54) is a 32-bit read/write register. Use this OMAP configuration register to set the peripheral request associated with DMA events 1 through 6.

Figure 76. Functional Multiplexing DSP DMA Register A (FUNC_MUX_DSP_DMA_A)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 54. Functional Multiplexing DSP DMA Register A (FUNC_MUX_DSP_DMA_A) Field Descriptions

Bits	Field	Value	Description
31–30	Reserved		These read-only bits return 0s when read.
29–25	CONF_DSP_DMA_ EVT_06	0–27	Configuration bits for DMA event 6. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 6. The value n must be between 0 and 27.
24–20	CONF_DSP_DMA_ EVT_05	0–27	Configuration bits for DMA event 5. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 5. The value n must be between 0 and 27.
19–15	CONF_DSP_DMA_ EVT_04	0–27	Configuration bits for DMA event 4. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 4. The value n must be between 0 and 27.
14–10	CONF_DSP_DMA_ EVT_03	0–27	Configuration bits for DMA event 3. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 3. The value n must be between 0 and 27.

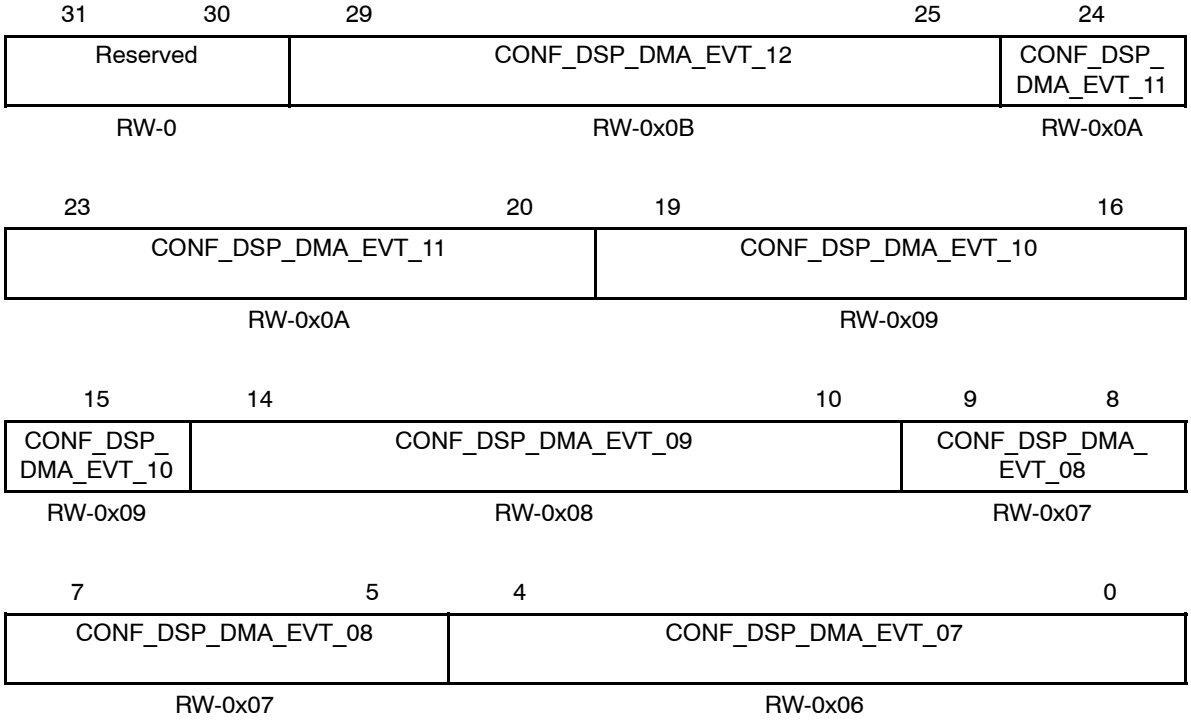
Table 54. Functional Multiplexing DSP DMA Register A (FUNC_MUX_DSP_DMA_A) Field Descriptions (Continued)

Bits	Field	Value	Description
9–5	CONF_DSP_DMA_EVT_02	0–27	Configuration bits for DMA event 2. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 2. The value n must be between 0 and 27.
4–0	CONF_DSP_DMA_EVT_01	0–27	Configuration bits for DMA event 1. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 1. The value n must be between 0 and 27.

Functional Multiplexing DSP DMA Register B (FUNC_MUX_DSP_DMA_B)

This global control register (see Figure 77 and Table 55) is a 32-bit read/write register. Use this OMAP configuration register to set the peripheral request associated with DMA events 7 through 12.

Figure 77. Functional Multiplexing DSP DMA Register B (FUNC_MUX_DSP_DMA_B)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

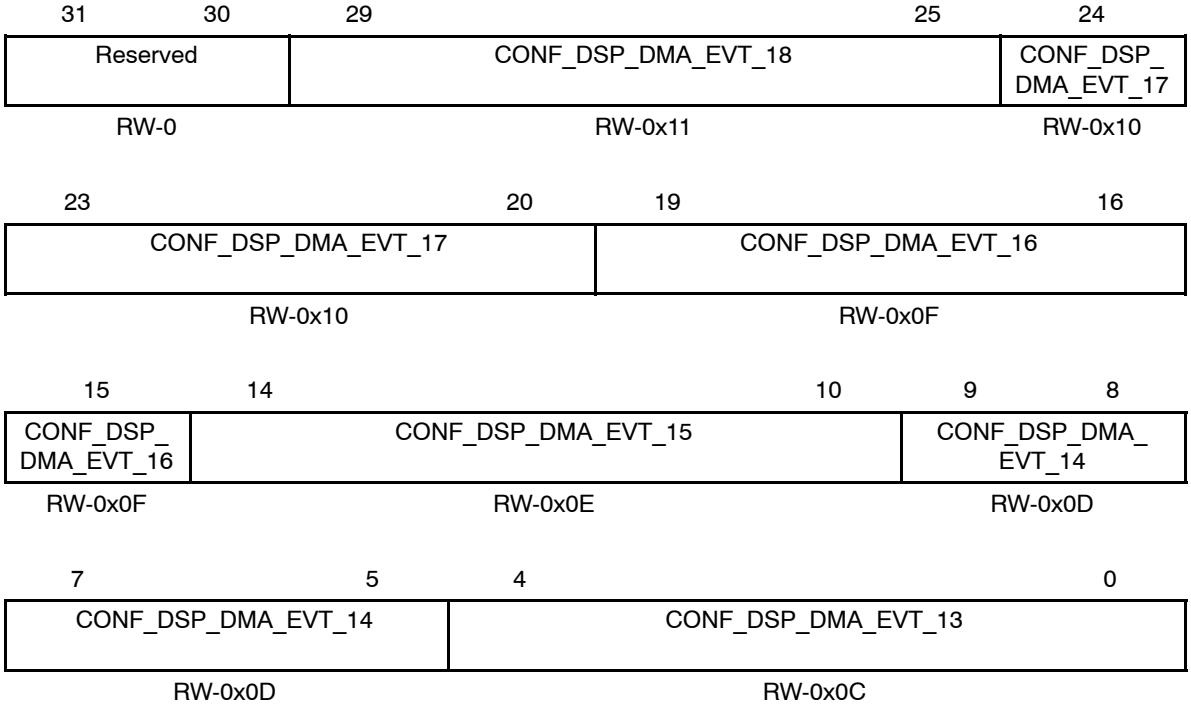
Table 55. Functional Multiplexing DSP DMA Register B (FUNC_MUX_DSP_DMA_B) Field Descriptions

Bits	Field	Value	Description
31–30	Reserved		These read-only bits return 0s when read.
29–25	CONF_DSP_DMA_EVT_12	0–27	Configuration bits for DMA event 12. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 12. The value n must be between 0 and 27.
24–20	CONF_DSP_DMA_EVT_11	0–27	Configuration bits for DMA event 11. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 11. The value n must be between 0 and 27.
19–15	CONF_DSP_DMA_EVT_10	0–27	Configuration bits for DMA event 10. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 10. The value n must be between 0 and 27.
14–10	CONF_DSP_DMA_EVT_09	0–27	Configuration bits for DMA event 9. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 9. The value n must be between 0 and 27.
9–5	CONF_DSP_DMA_EVT_08	0–27	Configuration bits for DMA event 8. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 8. The value n must be between 0 and 27.
4–0	CONF_DSP_DMA_EVT_07	0–27	Configuration bits for DMA event 7. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 7. The value n must be between 0 and 27.

Functional Multiplexing DSP DMA Register C (FUNC_MUX_DSP_DMA_C)

This global control register (see Figure 78 and Table 56) is a 32-bit read/write register. Use this OMAP configuration register to set the peripheral request associated with DMA events 13 through 18.

Figure 78. Functional Multiplexing DSP DMA Register C (FUNC_MUX_DSP_DMA_C)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 56. Functional Multiplexing DSP DMA Register C (FUNC_MUX_DSP_DMA_C) Field Descriptions

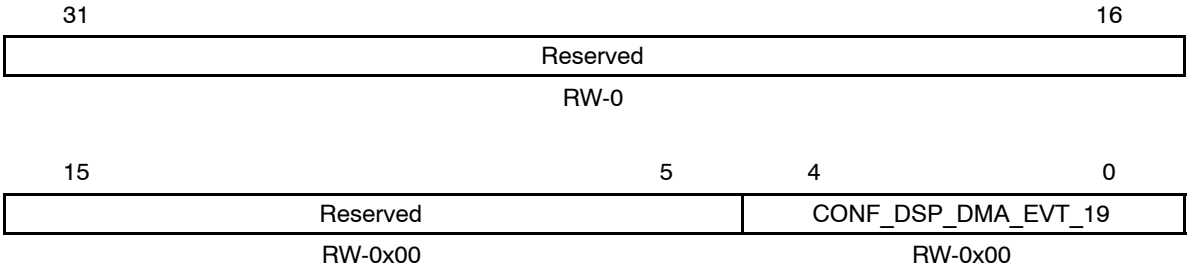
Bits	Field	Value	Description
31–30	Reserved		These read-only bits return 0s when read.
29–25	CONF_DSP_DMA_EVT_18	0–27	Configuration bits for DMA event 18. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 18. The value n must be between 0 and 27.
24–20	CONF_DSP_DMA_EVT_17	0–27	Configuration bits for DMA event 17. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 17. The value n must be between 0 and 27.
19–15	CONF_DSP_DMA_EVT_16	0–27	Configuration bits for DMA event 16. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 16. The value n must be between 0 and 27.
14–10	CONF_DSP_DMA_EVT_15	0–27	Configuration bits for DMA event 15. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 15. The value n must be between 0 and 27.

Bits	Field	Value	Description
9–5	CONF_DSP_DMA_ EVT_14	0–27	Configuration bits for DMA event 14. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 14. The value n must be between 0 and 27.
4–0	CONF_DSP_DMA_ EVT_13	0–27	Configuration bits for DMA event 13. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 13. The value n must be between 0 and 27.

Functional Multiplexing DSP DMA Register D (FUNC_MUX_DSP_DMA_D)

This global control register (see Figure 79 and Table 57) is a 32-bit read/write register. Use this OMAP configuration register to set the peripheral request associated with DMA event 19.

Figure 79. Functional Multiplexing DSP DMA Register D (FUNC_MUX_DSP_DMA_D)



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 57. Functional Multiplexing DSP DMA Register D (FUNC_MUX_DSP_DMA_D) Field Descriptions

Bits	Field	Value	Description
31–5	Reserved		These read-only bits return 0s when read.
4–0	CONF_DSP_DMA_ EVT_19	0–27	Configuration bits for DMA event 19. Writing a value n to this register maps peripheral request source n+1 to DSP DMA event 19. The value n must be between 0 and 27.

7.2.14 Reset Considerations

A DSP subsystem reset resets the DMA controller and the DMA configuration registers. Some of the registers are initialized after reset and some are not. The register definitions included in section 7.3 indicate the register contents after a DSP subsystem reset.

7.2.15 Interrupt Support

The DMA controller can send an interrupt to the DSP core in response to the operational events listed in Table 58. Each channel has interrupt enable (IE) bits in the interrupt control register (DMACICR) and some corresponding status bits in the status register (DMACSR). (DMACICR and DMACSR are described in section 7.3.6). If one of the operational events in the table occurs, the DMA controller checks the corresponding IE bit and acts accordingly:

- If the IE bit is 1 (the interrupt is enabled), the DMA controller sets the corresponding status bit and sends the associated interrupt request to the DSP core. DMACSR is automatically cleared if your program reads the register.
- If the IE bit is 0, no interrupt is sent and the status bit is not affected. DMACSR also has a SYNC bit that is used if you choose a synchronization event for the channel. SYNC indicates when the selected synchronization event has occurred (SYNC = 1) and when it has been serviced (SYNC = 0). For more details about synchronization events, see section 7.2.12.

Table 58. DMA Controller Operational Events and Their Associated Bits and Interrupts

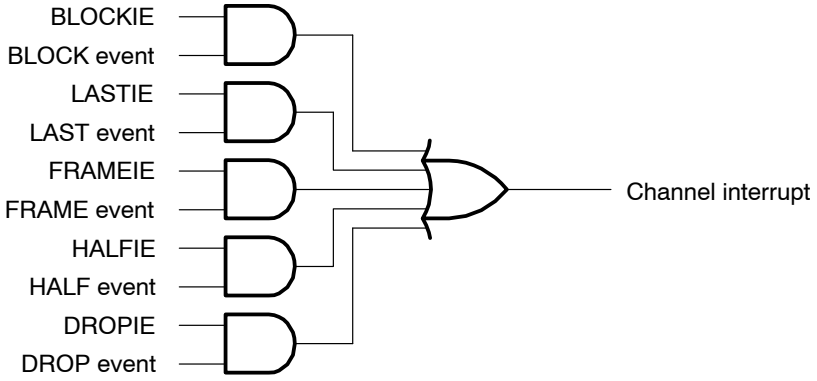
Operational Event	Interrupt Enable Bit	Status Bit	Associated Interrupt
Block transfer is complete	BLOCKIE	BLOCK	Channel interrupt
Last frame transfer has started	LASTIE	LAST	Channel interrupt
Frame transfer is complete	FRAMEIE	FRAME	Channel interrupt
First half of current frame has been transferred†	HALFIE	HALF	Channel interrupt
Synchronization event has been dropped	DROPIE	DROP	Channel interrupt
Timeout error has occurred	TIMEOUTIE	TIMEOUT	Bus-error interrupt

† For a frame with an odd number of elements, the half-frame event occurs as soon as the number of elements transferred is greater than the number that remains to be transferred. For example, for a frame of five elements, the half-frame event occurs when the DMA controller has transferred three of the elements.

7.2.15.1 Channel Interrupt

Each of the six channels has its own interrupt. As shown in Figure 80, the channel interrupt is the logical OR of all the enabled operational events except the timeout event (the timeout event generates a bus-error interrupt request). You can choose any combination of these five events by setting or clearing the appropriate interrupt enable (IE) bits in the interrupt control register (DMACICR) for the channel. You can determine which event(s) caused the interrupt by reading the bits in the status register (DMACSR) for the channel. The bits in DMACSR are not automatically cleared. A read of DMACSR clears all of the status bits. DMACSR should be read each time an interrupt occurs to clear the pending status bits.

Figure 80. Triggering a Channel Interrupt Request



As an example of using the interrupt enable bits, suppose you are monitoring activity in channel 1, and suppose that in DMACICR:

BLOCKIE = 0
LASTIE = 0
FRAMEIE = 1
HALFIE = 0
DROPIE = 1

When the current frame transfer is done or if a synchronization event is dropped (see section 7.2.12.3), the channel 1 interrupt request is sent to the DSP core. No other event can generate the channel 1 interrupt. To determine whether one or both of the events triggered the interrupt, you can read the FRAME and DROP bits in DMACSR.

The channel 1 interrupt sets its corresponding flag bit in an interrupt flag register of the DSP core. The DSP core can respond to the interrupt or ignore the interrupt.

For more details about DMACICR and DMACSR, see section 7.3.6.

7.2.15.2 Interrupt Multiplexing

Each channel interrupt is routed directly to the DSP core via the DSP Level 1 Interrupt Handler. A timeout event generates a bus error interrupt to the DSP core.

For more information on the DSP subsystem interrupt handlers, see the *OMAP5910 Dual-Core Processor DSP Subsystem Interrupts Reference Guide* (SPRU923) or the *OMAP5912 Multimedia Processor Interrupts Reference Guide* (SPRU757).

7.2.15.3 Timeout Error Conditions

A timeout error condition exists when a memory access has been stalled for too many cycles. Three of the four standard ports of the DMA controller are supported by hardware to detect a timeout error:

- DARAM port: A timeout counter in the DARAM port counts how many cycles have passed since a request was made to access the DARAM. When the counter reaches its timeout value of 255 DSP core clock cycles, the DARAM port generates an internal timeout signal. This counter can be enabled or disabled by writing to the DARAM timeout counter enable bit (DTCE in DMAGTCR). A timeout error on the DARAM port can occur because the DSP core is using the port and preventing access by the DMA controller, or because an address was specified that does not exist in the DARAM of the DSP subsystem.
- SARAM port: A timeout counter in the SARAM port counts how many cycles have passed since a request was made to access the SARAM. When the counter reaches its timeout value of 255 DSP core clock cycles, the SARAM port generates a timeout signal. This counter can be enabled or disabled by writing to the SARAM timeout counter enable bit (STCE in DMAGTCR). A timeout error on the SARAM port can occur because the DSP core is using the port and preventing access by the DMA controller, or because an address was specified that does not exist in the SARAM of the DSP subsystem.
- External memory port: The EMIF port does not support a timeout feature.
- Peripheral port: A timeout counter in the TIPB bridge module counts how many cycles have passed since a request was made to access a peripheral. When the counter reaches its timeout value of 127 DSP core clock cycles, the peripheral bus controller sends a timeout signal to the DMA controller. A timeout error on the peripheral port can occur because an address was specified that does not exist in I/O space on the DSP.

In response to a timeout signal, the DMA controller disables the channel (EN = 0 in DMACCR) and activity in the channel stops. If the corresponding interrupt enable bit is set (TIMEOUTIE = 1 in DMACICR), the DMA controller also sets the timeout status bit (TIMEOUT = 1 in DMACSR) and sends the timeout signal to the DSP core as an interrupt request. The interrupt request sets the bus-error interrupt (BERRINT) flag bit in the DSP core. The DSP core can respond to the interrupt request or ignore the interrupt request.

7.2.16 Power Management

The DSP module is divided into idle domains that can be programmed to be idle or active. The state of all domains is called the idle configuration. Any idle configuration that disables the DMA domain stops the DMA clock and, therefore, stops activity in the DMA controller.

Note:

There is no hardware handshaking to ensure all ongoing transfers are completed before the DMA controller goes into the idle state. All ongoing transfers are immediately suspended when the DMA controller is placed in the idle state.

When the DMA domain is idle, it can be temporarily reactivated without a change in the idle configuration in the following case. If one of the multichannel buffered serial ports (McBSPs) needs the DMA controller for a data transfer, the DMA controller will leave its idle state to perform the data transfer and then enter its idle state again.

7.2.17 Emulation Considerations

The FREE bit of DMAGCR controls the behavior of the DMA controller when an emulation breakpoint is encountered. If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If FREE = 1, DMA transfers are not interrupted by a breakpoint.

7.2.18 Latency in DMA Transfers

Each element transfer in a channel is composed of a read access (a transfer from the source location to the channel buffer) and a write access (a transfer from the channel buffer to the destination location). The time to complete this activity depends on factors such as:

- The selected frequency of the DSP core clock signal. This signal, as propagated to the DMA controller, determines the timing for all DMA transfers.
- Wait states or other extra cycles added by or resulting from an interface.
- Activity on other channels. As channels are serviced in a sequential order, the number of pending DMA service requests in the other channels affects how often a given channel can be serviced. For more details on how the channels are serviced, see section 7.2.6.
- Competition from the Microprocessor Port Interface (MPUI). If the MPUI is sharing internal RAM with the channels, the DMA controller allocates cycles to the MPUI like it does to channels. If the MPUI is given exclusive access to the internal RAM, no channels can access the internal RAM until the MPUI access configuration is changed (see section 7.2.5).
- Competition from the DSP core. If the DMA controller and the DSP core request access to the same internal memory block in the same cycle and the memory block cannot service both requests at the same time, the DSP core request has higher priority. The DMA request is serviced as soon as there are no pending DSP core requests.
- The timing of synchronization events (if the channel is synchronized). The DMA controller cannot service a synchronized channel until the synchronization event has occurred. For more details on synchronization, see section 7.2.12.

The minimum (best-case) latency is determined by the ports used. On the SARAM and DARAM ports, the DMA controller can initiate one access per cycle, if the DMA controller is not competing with the DSP core for access to the same memory block. SARAM memory can support one access per cycle per memory block from either the DMA controller or the DSP core. A DSP core access to the same SARAM block used by the DMA controller (in the same cycle) will cause stalls on the DMA access. DARAM memory can support two accesses per cycle per memory block. If more than two DSP core accesses are pending to the same DARAM block used by the DMA controller, this causes stalls on the DMA access. The best-case transfer rate for channels using these ports would be one cycle to read at the source and one cycle to write at the destination. External memory accesses through the EMIF port are handled by the traffic controller; therefore, the minimum latency of the EMIF port is determined by factors such as pending traffic controller accesses, and EMIFF and EMIFS configurations. The minimum latency for the peripheral port is approximately 5 cycles per access.

7.3 DSP DMA Controller Registers

7.3.1 Overview

Table 59 lists the types of registers in the DMA controller. There are three global control registers (DMAGCR, DMAGSCR, and DMAGTCR) that affect all channel activity. In addition, for each of the DMA channels, there are individual channel configuration registers. For the I/O address of each register, see the data manual for your OMAP device.

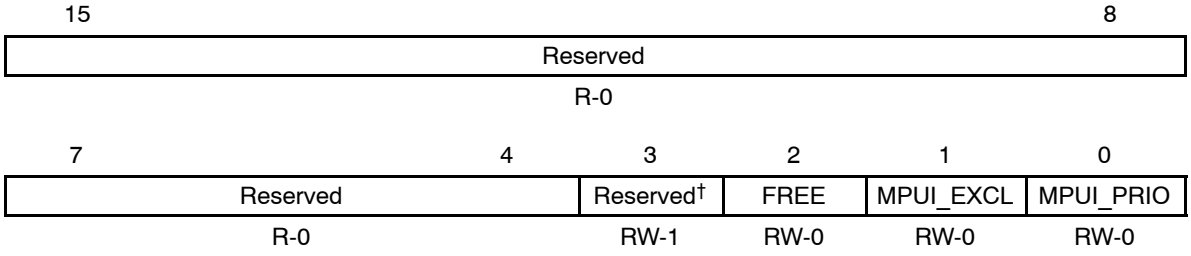
Table 59. Registers of the DMA Controller

Name	Description	See Section
DMAGCR	Global control register (only one)	7.3.2
DMAGSCR	Global software compatibility register (only one)	7.3.3
DMAGTCR	Global timeout control register (only one)	7.3.4
DMACCR	Channel control register (one for each channel)	7.3.5
DMACICR	Interrupt control register (one for each channel)	7.3.6
DMACSR	Status register (one for each channel)	7.3.6
DMACSDP	Source and destination parameters register (one for each channel)	7.3.7
DMACSSAL	Source start address (lower part) register (one for each channel)	7.3.8
DMACSSAU	Source start address (upper part) register (one for each channel)	7.3.8
DMACDSAL	Destination start address (lower part) register (one for each channel)	7.3.9
DMACDSAU	Destination start address (upper part) register (one for each channel)	7.3.9
DMACEN	Element number register (one for each channel)	7.3.10
DMACFN	Frame number register (one for each channel)	7.3.10
DMACEI/ DMACSEI	Element index register/source element index register (one for each channel)	7.3.11
DMACFI/ DMACSFI	Frame index register/source frame index register (one for each channel)	7.3.11
DMACDEI	Destination element index register (one for each channel)	7.3.11
DMACDFI	Destination frame index register (one for each channel)	7.3.11
DMACSAC	Source address counter register (one for each channel)	7.3.12
DMACDAC	Destination address counter register (one for each channel)	7.3.12

7.3.2 DMA Global Control Register (DMAGCR)

The global control register (see Figure 81 and Table 60) is a 16-bit read/write register. Use this I/O-mapped register to set the emulation mode of the DMA controller (FREE) and to define how the DMA controller treats the host port interface (MPUI_EXCL and MPUI_PRIO).

Figure 81. DMA Global Control Register (DMAGCR)



† Always write 1 to this reserved bit.

Note: R - Read, W = Write

Table 60. DMA Global Control Register (DMAGCR) Field Descriptions

Bits	Field	Value	Description
15–4	Reserved		These read-only bits return 0s when read.
3	Reserved		Always write 1 to this reserved bit.
2	FREE		Emulation mode bit. FREE controls the behavior of the DMA controller when an emulation breakpoint is encountered:
		0	A breakpoint suspends DMA transfers.
		1	DMA transfers continue uninterrupted when a breakpoint occurs.
1	MPUI_EXCL		MPUI exclusive access bit. MPUI_EXCL determines whether the Microprocessor Port Interface (MPUI) has exclusive access to the internal RAM of the DSP. Note: Regardless of the value of MPUI_EXCL, the MPUI cannot access the peripheral port.
		0	The MPUI shares the internal RAM with the DMA channels. The MPUI can access any internal and DSP external memory in its address reach.
		1	The MPUI has exclusive access to the internal RAM. If any channels must access the DARAM port or the SARAM port, activity in these channels is suspended. In this MPUI access configuration, the MPUI can only access the DARAM port and the SARAM port. It cannot access the DSP external memory port.

Table 60. DMA Global Control Register (DMAGCR) Field Descriptions (Continued)

Bits	Field	Value	Description
0	MPUI_PRIO		MPUI priority bit. MPUI_PRIO assigns the MPUI a high or low priority level in the service chain of the DMA controller. Note: When the MPUI has exclusive access to the DARAM and SARAM ports (MPUI_EXCL = 1), the MPUI priority is irrelevant at these ports because none of the DMA channels can access the DARAM and SARAM ports.
		0	Low priority level.
		1	High priority level.

7.3.3 DMA Global Software Compatibility Register (DMAGSCR)

The global software compatibility register is a 16-bit read/write register that controls how the DMA controller gets the destination element index and the destination frame index. The original DMA controller design used one element index register (DMACEI) for both source and destination, and one frame index register (DMACFI) for both source and destination. Later designs were enhanced to allow separate source and destination indexes. In the enhanced mode:

- DMACEI is DMACSEI, the source element index register. DMACFI is DMACSFI, the source frame index register.
- The destination element index is stored in a separate destination element index register (DMACDEI) and the destination frame index is stored in a separate destination frame index register (DMACDEI).

DMAGSCR provides the ability to choose either the original method of indexing (to maintain software compatibility with code written for the original design) or the enhanced method of indexing.

DMAGSCR is summarized by Figure 82 and Table 61.

Figure 82. DMA Global Software Compatibility Register (DMAGSCR)

15	1	0
Reserved		DINDXMD
R-0		RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 61. DMA Global Software Compatibility Register (DMAGSCR) Field Descriptions

Bits	Field	Value	Description
15–1	Reserved		These read-only bits return 0s when read.
0	DINDXMD		Destination element and frame index mode bit. This bit determines which registers will be used to indicate the destination element and frame indexes.
		0	Compatibility mode.
			One element index for both the source and destination is stored in the channel source element index register (DMACSEI).
			One frame index for both the source and destination is stored in the channel source frame index register (DMACSFI).
		1	Enhanced mode.
			The source element index is stored in the channel source element index register (DMACSEI).
			The destination element index is stored in the channel destination element index register (DMACDEI).
			The source frame index is stored in the channel source frame index register (DMACSFI).
			The destination frame index is stored in the channel destination frame index register (DMACDFI).

7.3.4 DMA Global Timeout Control Register (DMAGTCR)

The global timeout control register (see Figure 83 and Table 62) is a 16-bit read/write register that enables or disables timeout counters on the SARAM and DARAM ports. If the timeout counters are disabled, the DMA controller will never generate a timeout error for these ports. For details about the timeout error conditions, see section 7.2.15.3.

Figure 83. DMA Global Timeout Control Register (DMAGTCR)

15	2	1	0
Reserved		DTCE	STCE
R-0		RW-0	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 62. DMA Global Timeout Control Register (DMAGTCR) Field Descriptions

Bits	Field	Value	Description
15–2	Reserved		These read-only bits return 0s when read.
1	DTCE		DARAM timeout counter enable bit. This bit enables/disables the timeout counter used to monitor delays on DMA requests to the DARAM port.
		0	DARAM timeout counter disabled.
		1	DARAM timeout counter enabled.
0	STCE		SARAM timeout counter enable bit. This bit enables/disables the timeout counter used to monitor delays on DMA requests to the SARAM port.
		0	SARAM timeout counter disabled.
		1	SARAM timeout counter enabled.

7.3.5 DMA Channel Control Register (DMACCR)

Each channel has a channel control register as shown in Figure 84. This I/O-mapped register enables you to:

- ❑ Choose how the source and destination addresses are updated (SRCAMODE and DSTAMODE)
- ❑ Enable and control repeated DMA transfers (AUTOINIT, REPEAT, and ENDPROG)
- ❑ Enable or disable the channel (EN)
- ❑ Choose a low or high priority level for the channel (PRIO)
- ❑ Select element synchronization or frame synchronization (FS)
- ❑ Determine what synchronization event (if any) initiates a transfer in the channel (SYNC)

Table 63 describes the register fields.

Figure 84. DMA Channel Control Register (DMACCR)

15	14	13	12	11	10	9	8
DSTAMODE		SRCAMODE		ENDPROG	Reserved†	REPEAT	AUTOINIT
RW-0		RW-0		RW-0	RW-0	RW-0	RW-0

7	6	5	4	0
EN	PRIO	FS	SYNC	
RW-0	RW-0	RW-0	RW-0	

† Bit 10 must be kept 0 for proper operation of the DMA controller.

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 63. DMA Channel Control Register (DMACCR) Field Descriptions

Bits	Field	Value	Description
15–14	DSTAMODE		Destination addressing mode bits. DSTAMODE determines the addressing mode used by the DMA controller when it writes to the destination port of the channel. At the end of a transfer, but before any incrementing, the destination address is the address of the last byte that was modified at the destination.
		00b	Constant address. The same address is used for each element transfer.
		01b	Automatic post increment. After each element transfer, the address is incremented according to the selected data type: If data type is 8-bit Address = Address + 1 If data type is 16-bit Address = Address + 2 If data type is 32-bit Address = Address + 4
		10b	Single index. After each element transfer, the address is incremented by the programmed element index amount: Address = Address + element index See section 7.3.11 for more details.
		11b	Double index (sort). After each element transfer, the address is incremented by the appropriate index amount: If there are more elements to transfer in the current frame Address = Address + element index If the last element in the frame has been transferred Address = Address + frame index See section 7.3.11 for more details.

Table 63. DMA Channel Control Register (DMACCR) Field Descriptions (Continued)

Bits	Field	Value	Description
13–12	SRCAMODE		Source addressing mode bits. SRCAMODE determines the addressing mode used by the DMA controller when it reads from the source port of the channel. At the end of a transfer, but before any incrementing, the source address is the address of the last byte that was read from the source.
		00b	Constant address. The same address is used for each element transfer.
		01b	Automatic post increment. After each element transfer, the address is incremented according to the selected data type: If data type is 8-bit Address = Address + 1 If data type is 16-bit Address = Address + 2 If data type is 32-bit Address = Address + 4
		10b	Single index. After each element transfer, the address is incremented by the programmed element index amount: Address = Address + element index See section 7.3.11 for more details.
		11b	Double index (sort). After each element transfer, the address is incremented by the appropriate index amount: If there are more elements to transfer in the current frame: Address = Address + element index If the last element in the frame has been transferred: Address = Address + frame index See section 7.3.11 for more details.

Table 63. DMA Channel Control Register (DMACCR) Field Descriptions (Continued)

Bits	Field	Value	Description
11	ENDPROG		End-of-programming bit. Each DMA channel has two sets of registers: configuration registers and working registers. When block transfers occur repeatedly because of auto-initialization (AUTOINIT = 1), you can change the context for the next DMA transfer by writing to the configuration registers during the current block transfer. At the end of the current transfer, the contents of the configuration registers are copied into the working registers, and the DMA controller begins the next transfer using the new context. For proper auto-initialization, the DSP core must finish programming the configuration registers before the DMA controller copies their contents. The DMA controller automatically clears the ENDPROG bit after copying the configuration registers to the working registers. The DSP core can then program the DMA channel context for the next iteration of the transfer by programming the configuration registers. To make sure auto-initialization waits for the DSP core, follow this procedure: 1) Make auto-initialization wait for ENDPROG = 1 by clearing the REPEAT bit (REPEAT = 0). 2) Poll for ENDPROG = 0, which indicates that the DMA controller has finished copying the previous context. The configuration registers can now be programmed for the next iteration. 3) Program the configuration registers. 4) Set ENDPROG (ENDPROG = 1) to indicate the end of register programming.
		0	The configuration registers are ready for programming, or programming is in progress.
		1	End of programming. The programming of the configuration registers is complete.
10	Reserved	0	This reserved bit must be kept 0. Make sure that whenever your program modifies DMACCR, it writes a 0 to bit 10.

Table 63. DMA Channel Control Register (DMACCR) Field Descriptions (Continued)

Bits	Field	Value	Description
9	REPEAT		Repeat condition bit. If auto-initialization is selected for a channel (AUTOINIT = 1), REPEAT specifies one of two special repeat conditions:
		0	Repeat only if ENDPROG = 1. Once the current DMA transfer is complete, auto-initialization will wait for the end-of-programming bit (ENDPROG) bit to be set.
		1	Repeat regardless of ENDPROG. Once the current DMA transfer is complete, auto-initialization occurs regardless of whether ENDPROG is 0 or 1.
8	AUTOINIT		Auto-initialization bit. The DMA controller supports auto-initialization, which is the automatic re-initialization of the channel between DMA block transfers. Use AUTOINIT to enable or disable this feature.
		0	Auto-initialization is disabled. Activity in the channel stops at the end of the current block transfer. To stop a transfer immediately, clear the channel enable it (EN).
		1	Auto-initialization is enabled. Once the current block transfer is complete, the DMA controller reinitializes the channel and starts a new block transfer. There are two options to stop activity in the channel: ❑ To stop activity immediately, clear the channel enable bit (EN = 0). ❑ To stop activity after the current block transfer, clear AUTOINIT (AUTOINIT= 0).
7	EN		Channel enable bit. Use EN to enable or disable transfers in the channel. The DMA controller clears EN when a block transfer in the channel is complete. **Note**: If the DSP core attempts to write to EN at the same time that the DMA controller must clear EN, the DMA controller is given higher priority. EN is cleared, and the value from the DSP core is discarded.
		0	The channel is disabled. The channel cannot be serviced by the DMA controller. If a DMA transfer is already active in the channel, the DMA controller stops the transfer and resets the channel.
		1	The channel is enabled. The channel can be serviced by the DMA controller at the next available time slot.

Table 63. DMA Channel Control Register (DMACCR) Field Descriptions (Continued)

Bits	Field	Value	Description
6	PRIO		Channel priority bit. All six of the DMA channels are given a fixed position and programmable priority level on the service chain of the DMA controller. PRIO determines whether the associated channel has a high priority or a low priority. High-priority channels are serviced before low-priority channels.
		0	Low priority.
		1	High priority.
5	FS		Frame/element synchronization bit. You can use the SYNC bits of DMACCR to specify a synchronization event for the channel. The FS bit determines whether the synchronization event initiates the transfer of an element or an entire frame of data:
		0	Element synchronization. When the selected synchronization event occurs, one element is transferred in the channel. Each element transfer waits for the synchronization event.
		1	Frame synchronization. When the selected synchronization event occurs, an entire frame is transferred in the channel. Each frame transfer waits for the synchronization event.
4–0	SYNC	(see section 7.2.12.4)	Synchronization control bits. SYNC in DMACCR determines the peripheral request (for example, a serial port transmit request).
			A DSP subsystem reset selects SYNC = 00000b (no synchronization event). When SYNC = 00000b, the DMA controller does not wait for a synchronization event before beginning a DMA transfer in the channel; channel activity begins as soon as the channel is enabled (EN = 1).
			If the DMA is configured to recognize a synchronization event (SYNC is something other than 00000b) and the synchronization event occurs before the channel is enabled, the synchronization event will be latched and serviced as soon as the channel is enabled. If it is preferable to ignore the synchronization events that occur before the channel is enabled, then the SYNC field should be set to 00000b while the channel is disabled.
			Each peripheral request can be mapped to a specific DMA synchronization event through the GDMA Handler. However, at reset, a default mapping is implemented.

7.3.6 DMA Interrupt Control Register (DMACICR) and Status Register (DMACSR)

Each channel has an interrupt control register (DMACICR) and a status register (DMACSR). DMACICR and DMACSR are I/O-mapped registers. Their bits are shown in Figure 85 and described in Table 64 and Table 65.

Use DMACICR to specify if one or more operational events in the DMA controller trigger an interrupt. If an operational event occurs and its interrupt enable (IE) bit is 1, an interrupt request is sent to the DSP core, where it can be serviced or ignored. Each channel has its own interrupt line to the DSP core and one set of flag and enable bits in the DSP core. In addition, the DMA controller can send a bus-error interrupt request to the DSP core in response to a timeout error. The bus-error interrupt also has a set of flag and enable bits in the DSP core.

To see which operational event or events have occurred in the DMA controller, your program can read DMACSR. The DMA controller only sets one of the interrupt flag bits (bits 5-0) if the operational event occurs and the associated interrupt enable bit is set in DMACICR. After your program reads DMACSR, all of its bits are cleared automatically.

The SYNC bit (bit 6) of DMACSR can be used to detect when a synchronization event has occurred (SYNC = 1) and when the resulting access request has been serviced (SYNC = 0).

Figure 85. DMA Interrupt Control Register (DMACICR) and Status Register (DMACSR)

DMACICR

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved		BLOCKIE	LASTIE	FRAMEIE	HALFIE	DROPIE	TIMEOUTIE
R-0		RW-0	RW-0	RW-0	RW-0	RW-1	RW-1

DMACSR

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	SYNC	BLOCK	LAST	FRAME	HALF	DROP	TIMEOUT
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 64. DMA Interrupt Control Register (DMACICR) Fields Descriptions

Bits	Field	Value	Description
15–6	Reserved		These read-only bits returns 0s when read.
5	BLOCKIE		Whole block interrupt enable bit. BLOCKIE determines how the DMA controller responds when all of the current block has been transferred from the source port to the destination port.
		0	Do not record the event.
		1	Set the BLOCK bit and send the channel interrupt request to the DSP core.
4	LASTIE		Last frame interrupt enable bit. LASTIE determines how the DMA controller responds when the DMA controller starts transferring the last frame from the source port to the destination port.
		0	Do not record the event.
		1	Set the LAST bit and send the channel interrupt request to the DSP core.

Table 64. DMA Interrupt Control Register (DMACICR) Fields Descriptions (Continued)

Bits	Field	Value	Description
3	FRAMEIE		Whole frame interrupt enable bit. FRAMEIE determines how the DMA controller responds when the all of the current frame has been transferred from the source port to the destination port.
		0	Do not record the event.
		1	Set the FRAME bit and send the channel interrupt request to the DSP core.
2	HALFIE		Half frame interrupt enable bit. HALFIE determines how the DMA controller responds when the first half of the current frame has been transferred from the source port to the destination port. For a frame with an odd number of elements, the half-frame event occurs as soon as the number of elements transferred is greater than the number that remains to be transferred. For example, for a frame of five elements, the half-frame event occurs when the DMA controller has transferred three of the elements.
		0	Do not record the event.
		1	Set the HALF bit and send the channel interrupt request to the DSP core.
1	DROPIE		Synchronization event drop interrupt enable bit. An error occurs if a DMA synchronization event occurs again before the DMA controller has finished servicing the previous DMA request. This error is called a synchronization event drop. DROPIE determines how the DMA controller responds when a synchronization event drop occurs in the channel.
		0	Do not record the drop.
		1	Set the DROP bit and send the channel interrupt request to the DSP core.
0	TIMEOUTIE		Timeout interrupt enable bit. TIMEOUTIE determines how the DMA controller responds to a timeout error at the source port or the destination port of the channel. Section 7.2.15.3 describes the timeout error conditions.
		0	Do not record the timeout error.
		1	Set the TIMEOUT bit and send the bus-error interrupt request to the DSP core.

Table 65. DMA Status Register (DMACSR) Field Descriptions

Bits	Field	Value	Description
15–7	Reserved		These read-only bits returns 0s when read.
6	SYNC		Synchronization event status bit. The DMA controller updates SYNC to indicate when the synchronization event for the channel has occurred or when the synchronized channel has been serviced. An error occurs if a DMA synchronization event occurs again before the DMA controller has finished servicing the previous DMA request. This error is called a synchronization event drop. You can track this type of error using the DROPIE bit and the DROP bit. To select a synchronization event for a channel, use the SYNC bits of DMACCR.
		0	The DMA controller has finished servicing the previous access request.
		1	A synchronization event has occurred. In response to the event, the synchronized channel submits an access request to its source port.
5	BLOCK		Whole block status bit. The DMA controller only sets BLOCK if BLOCKIE = 1 in DMACICR and all of the current block has been transferred from the source port to the destination port.
		0	The whole-block event has not occurred yet, or BLOCK has been cleared.
		1	The whole block has been transferred. A channel interrupt request has been sent to the DSP core.
4	LAST		Last frame status bit. The DMA controller sets LAST only if LASTIE = 1 in DMACICR and the DMA controller has started transferring the last frame from the source port to the destination port.
		0	The last-frame event has not occurred yet, or LAST has been cleared.
		1	The DMA controller has started transferring the last frame. A channel interrupt request has been sent to the DSP core.
3	FRAME		Whole frame status bit. The DMA controller sets FRAME only if FRAMEIE = 1 in DMACICR and all of the current frame has been transferred from the source port to the destination port.
		0	The whole-frame event has not occurred yet, or FRAME has been cleared.
		1	The whole frame has been transferred. A channel interrupt request has been sent to the DSP core.

Table 65. DMA Status Register (DMACSR) Field Descriptions (Continued)

Bits	Field	Value	Description
2	HALF		Half frame status bit. The DMA controller sets HALF only if HALFIE = 1 in DMACICR and the first half of the current frame has been transferred from the source port to the destination port. For a frame with an odd number of elements, the half-frame event occurs as soon as the number of elements transferred is greater than the number that remains to be transferred. For example, for a frame of five elements, the half-frame event occurs when the DMA controller has transferred three of the elements.
		0	The half-frame event has not occurred yet, or HALF has been cleared.
		1	The first half of the frame has been transferred. A channel interrupt request has been sent to the DSP core.
1	DROP		Synchronization event drop status bit. An error occurs if a DMA synchronization event occurs again before the DMA controller has finished servicing the previous DMA request. This error is called a synchronization event drop. The DMA controller sets DROP only if DROPIE = 1 in DMACICR and a synchronization event drop has occurred in the channel.
		0	A synchronization event drop has not occurred, or DROP has been cleared.
		1	A synchronization event drop has occurred. A channel interrupt request has been sent to the DSP core.
0	TIMEOUT		Timeout status bit. The DMA controller sets TIMEOUT only if TIMEOUTIE = 1 in DMACICR and a timeout error has occurred at the source port or the destination port of the channel. The timeout error conditions are described in section 7.2.15.3.
		0	A timeout error has not occurred, or TIMEOUT has been cleared.
		1	A timeout error has occurred. A bus-error interrupt request has been sent to the DSP core.

7.3.7 DMA Source and Destination Parameters Register (DMACSDP)

Each channel has a source and destination parameters register of the form shown in Figure 86. This I/O-mapped register enables you to choose a source port (SRC) and a destination port (DST), specify a data type (DATATYPE) for port accesses, enable or disable data packing (SRCPACK and DSTPACK), and enable or disable burst transfers (SRCBEN and DSTBEN). Table 66 describes the fields of this register.

Figure 86. DMA Source and Destination Parameters Register (DMACSDP)

15	14	13	12	9	8
DSTBEN		DSTPACK	DST		SRCBEN
RW-0		RW-0	RW-0		RW-0

7	6	5	2	1	0
SRCBEN	SRCPACK	SRC		DATATYPE	
RW-0	RW-0	RW-0		RW-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 66. DMA Source and Destination Parameters Register (DMACSDP) Field Descriptions

Bits	Field	Value	Description
15–14	DSTBEN		Destination burst enable bits. A burst in the DMA controller is four consecutive 32-bit accesses at a DMA port. DSTBEN determines whether the DMA controller performs a burst at the destination port of the channel. Bursting is not supported when both the source and destination are configured to be the EMIF port (SRC = DST = XX10b).
		00b	Bursting disabled (single access enabled) at the destination.
		01b	Bursting disabled (single access enabled) at the destination.
		10b	Bursting enabled at the destination. When writing to the destination, the DMA controller performs four consecutive 32-bit accesses.
		11b	Reserved (do not use).
13	DSTPACK		Destination packing enable bit. The DMA controller can perform data packing to double or quadruple the amount of data passed to the destination in a single transfer. For example, if an 8-bit data type is selected and the destination port has a 32-bit data bus, four 8-bit pieces of data can be packed into 32 bits before being sent to the destination. DSTPACK determines whether data packing is used at the destination port.
		0	Packing disabled at the destination.
		1	Packing enabled at the destination. Where possible, the DMA controller packs data before each write to the destination. Table 48 shows the instances where data packing is performed.

Table 66. DMA Source and Destination Parameters Register (DMACSDP) Field Descriptions (Continued)

Bits	Field	Value	Description
12–9	DST		Destination selection bits. DST selects which DMA port is the destination for data transfers in the channel.
		0000b	SARAM (single-access RAM inside the DSP subsystem).
		0001b	DARAM (dual-access RAM inside the DSP subsystem).
		0010b	External memory (via the external memory interface, EMIF). Internally, 8-bit data read requests from the DSP external memory are converted to 16-bit data read requests by the EMIF. The appropriate byte is fetched from this read request and placed in internal memory.
		0011b	Peripherals (via the shared TI peripheral bus bridge). Connection of a channel configured to have an 8-bit data type to the peripheral port is not supported.
		Others	Reserved.
8–7	SRCBEN		Source burst enable bits. A burst in the DMA controller is four consecutive 32-bit accesses at a DMA port. SRCBEN determines whether the DMA controller performs a burst at the source port of the channel. This field will be ignored if: ❑ The source port does not support burst capability, or ❑ Constant address mode is selected for the source port, or ❑ The channel is element synchronized. Bursting is not supported when both the source and destination are configured to be the EMIF port (SRC = DST = XX10b).
		00b	Bursting disabled (single access enabled) at the source.
		01b	Bursting disabled (single access enabled) at the source.
		10b	Bursting enabled at the source. When reading from the source, the DMA controller performs four consecutive 32-bit accesses.
		11b	Reserved (do not use).

Table 66. DMA Source and Destination Parameters Register (DMACSDP) Field Descriptions (Continued)

Bits	Field	Value	Description
6	SRCPACK		Source packing enable bit. The DMA controller can perform data packing to double or quadruple the amount of data gathered at the source before a transfer. For example, if an 8-bit data type is selected and the source port has a 32-bit data bus, four 8-bit pieces of data can be packed into 32 bits before being sent through the channel. SRCPACK determines whether data packing is used at the source port.
		0	Packing disabled at the source.
		1	Packing enabled at the source. Where possible, the DMA controller packs data from the source before beginning a data transfer in the channel. Table 48 shows the instances where data packing is performed.
5–2	SRC		Source selection bits. SRC selects which DMA port is the source for data transfers in the channel.
		0000b	SARAM (single-access RAM inside the DSP subsystem).
		0001b	DARAM (dual-access RAM inside the DSP subsystem).
		0010b	External memory (via the external memory interface, EMIF). Internally, 8-bit data read requests from the DSP external memory are converted to 16-bit data read requests by the EMIF. The appropriate byte is fetched from this read request and placed in internal memory.
		0011b	Peripherals (via the shared TI peripheral bus bridge). Connection of a channel configured to have an 8-bit data type to the peripheral port is not supported.
		Others	Reserved.

Table 66. DMA Source and Destination Parameters Register (DMACSDP) Field Descriptions (Continued)

Bits	Field	Value	Description
1–0	DATATYPE		Data type bits. DATATYPE indicates how data is to be accessed at the source and at the destination of the channel. Note that the DMA controller uses *byte addresses* for its accesses; each byte in data space or I/O space has its own address. For information on how addresses are updated between element transfers, see the descriptions for the DSTAMODE bits and the SRCAMODE bits of DMACCR, section 7.3.5.
		00b	8-bit. The DMA controller makes 8-bit accesses at the source and at the destination of the channel. The source and destination start addresses have no alignment constraint: Start address: XXXX XXXX XXXX XXXXb (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the corresponding address is updated by an increment of 1 after each element transfer. Connection of a channel configured as 8-bit data type to the peripheral port is not supported. Internally, 8-bit data read requests from the DSP external memory are converted to 16-bit data read requests by the EMIF. The appropriate byte is fetched from this read request and placed in internal memory.
		01b	16-bit. The DMA controller makes 16-bit accesses at the source and at the destination. The source and destination start addresses must each be on an even 2-byte boundary; the least significant bit (LSB) must be 0: Start address: XXXX XXXX XXXX XXX0b (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the address is updated by an increment of 2 after each element transfer.
		10b	32-bit. The DMA controller makes 32-bit accesses at the source and at the destination. The source and destination start addresses must be on an even 4-byte boundary; the 2 LSBs must be 0: Start address: XXXX XXXX XXXX XX00b (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the address is updated by an increment of 4 after each element transfer.
		11b	Reserved (do not use).

7.3.8 DMA Source Start Address Registers (DMACSSAU and DMACSSAL)

Each channel has two source start address registers, which are shown in Figure 87 and described in Table 68 and Table 67. For the first access to the source port of the channel, the DMA controller generates a byte address by concatenating the contents of these two I/O-mapped registers. DMACSSAU supplies the upper bits, and DMACSSAL supplies the lower bits:

Source start address = DMACSSAU:DMACSSAL

Notes:

1) You must load the source start address registers with a byte address. If you have a word address, shift it left by 1 before loading the registers.
2) If you have a 16-bit or 32-bit data type, the start address must be aligned properly. See the description for the DATATYPE bits of DMACSDP in section 7.3.7.
3) Ensure that the start address, element index, and frame index will produce valid addresses within the range of the port. If an invalid address is generated, a timeout error will occur.

The destination start address is supplied by DMACDSAL and DMACDSAU, which are described in section 7.3.9.

Figure 87. DMA Source Start Address Registers (DMACSSAU and DMACSSAL)

DMACSSAU

15	0
SSAU	
RW-x	

DMACSSAL

15	0
SSAL	
RW-x	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after DSP reset is not defined.

Table 67. DMA Source Start Address Register − Upper Part (DMACSSAU) Field Descriptions

Bits	Field	Value	Description
15−0	SSAU	0000h−00FFh	Upper part of source start address (byte address).
		0100h−FFFFh	Reserved (do not use).

Table 68. DMA Source Start Address Register − Lower Part (DMACSSAL) Field Descriptions

Bits	Field	Value	Description
15−0	SSAL	0000h−FFFFh	Lower part of source start address (byte address).

7.3.9 DMA Destination Start Address Registers (DMACDSAU and DMACDSAL)

Each channel has two destination start address registers, which are shown in Figure 88 and described in Table 70 and Table 69. For the first access to the destination port of the channel, the DMA controller generates a byte address by concatenating the contents of the two I/O-mapped registers. DMACDSAU supplies the upper bits, and DMACDSAL supplies the lower bits:

Destination start address = DMACDSAU:DMACDSAL

Notes:

1) You must load the source start address registers with a byte address. If you have a word address, shift it left by 1 before loading the registers.
2) If you have a 16-bit or 32-bit data type, the start address must be aligned properly. See the description for the DATATYPE bits of DMACSDP in section 7.3.7.
3) Ensure that the start address, element index, and frame index will produce valid addresses within the range of the port. If an invalid address is generated, a timeout error will occur.

The source start address is supplied by DMACSSAU and DMACSSAL, described in section 7.3.8.

Figure 88. DMA Destination Start Address Registers (DMACDSAU and DMACDSAL)

DMACDSAU

15	0
DSAU	
RW-x	

DMACDSAL

15	0
DSAL	
RW-x	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after DSP reset is not defined.

Table 69. DMA Destination Start Address Register – Upper Part (DMACDSAU) Field Descriptions

Bits	Field	Value	Description
15–0	DSAU	0000h–00FFh	Upper part of source start address (byte address).
		0100h–FFFFh	Reserved (do not use).

Table 70. DMA Destination Start Address Register – Lower Part (DMACDSAL) Field Descriptions

Bits	Field	Value	Description
15–0	DSAL	0000h–FFFFh	Lower part of source start address (byte address).

7.3.10 DMA Element Number Register (DMACEN) and Frame Number Register (DMACFN)

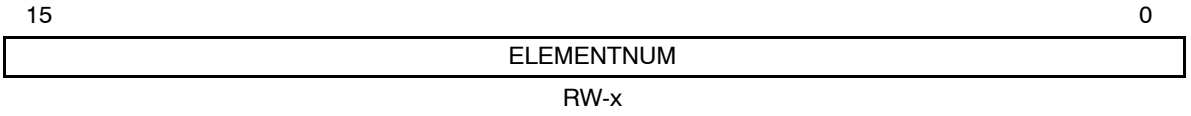
Each channel has an element number register and a frame number register (see Figure 89, Table 71, and Table 72). Load DMACFN with the number of frames you want in each block. Load DMACEN with the number of elements you want in each frame. You must have at least one frame and one element, and you can have as many as 65535 of each:

$1 \leq$ frame number ≤ 65535
$1 \leq$ element number ≤ 65535

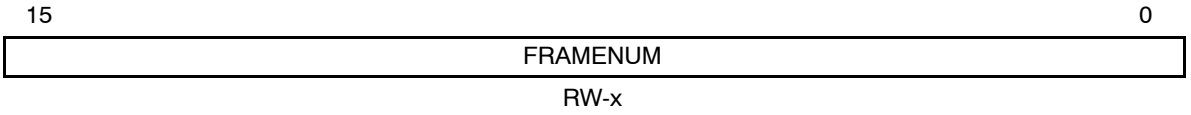
DMACEN and DMACFN are uninitialized after a DSP subsystem reset.

Figure 89. DMA Element Number Register (DMACEN) and Frame Number Register (DMACFN)

DMACEN



DMACFN



Note: R = Read; W = Write; –n = Value after reset; –x = Value after DSP reset is not defined.

Table 71. DMA Element Number Register (DMACEN) Field Descriptions

Bits	Field	Value	Description
15–0	ELEMENTNUM	0000h	Reserved (do not use).
		0001h–FFFFh	Number of elements per frame (1–65535).

Table 72. Frame Number Register (DMACFN) Field Descriptions

Bits	Field	Value	Description
15–0	FRAMENUM	0000h	Reserved (do not use).
		0001h–FFFFh	Number of frames per block (1–65535).

7.3.11 DMA Element Index Registers (DMACSEI, DMACDEI) and Frame Index Registers (DMACSFI, DMACDFI)

The single- or double-index addressing mode can be selected separately for the source and destination ports by using the SRCAMODE bits and the DSTAMODE bits, respectively, of DMACCR (see section 7.3.5). To support the index addressing modes, there are four index registers: two source index registers (DMACSEI and DMACSFI) and two destination index registers (DMACDEI and DMACDFI). The way these registers are used depends on the destination index mode chosen with the DINDXMD bit of DMAGSCR.

When DINDXMD = 0 (the default forced by a DSP subsystem reset), a compatibility mode is selected. In the original DMA controller design, the source and the destination shared one element index register called DMACEI and one frame index register called DMACFI. When DINDXMD = 0, compatible behavior is enabled; DMACSEI is used as DMACEI, and DMACSFI is used as DMACFI. The destination index registers are not used.

When DINDXMD=1, an enhanced mode is selected. In this mode, the source index registers are used only for the source, and the destination index registers are used for the destination.

The element and frame indexes are 16-bit signed numbers, providing the following range:

−32768 bytes ≤ frame index ≤ 32767 bytes
−32768 bytes ≤ element index ≤ 32767 bytes

After each transfer, the source and destination address registers contain the address for the last byte of the transferred element. For example, if the DMA channel is reading a 32-bit element at byte address 0x2000, the source address will be 0x2003 after the element is read because the DMA channel will read a total of four bytes. If the DMA channel reads a 16-bit element, the source address would be 0x2001 after the element read because only two bytes are read. For a byte read, the source address would stay at 0x2000 after the byte read.

When the single index mode is used, the element index is added to the source or destination address at the end of each element transfer. The modified address will then be used at the beginning of the next element transfer.

When the double index mode is used for the source or the destination address, the element index is added to the source or destination address at the end of each element transferred as described above, except for the last element in the frame. For the last element in the frame, the frame index is added to the source or destination address instead of the element index. For example, if the last element in the frame starts at byte address 0x801E where the data type is 16-bit and the frame index is set to 0x0003, the DMA controller will move the first byte (0x801E), then the second byte (0x801F) of the element. The frame index will then be added to 0x801F to create the address for the first byte of the next element to be moved (0x801F + 0x0003 = 0x8022).

The element index that is added to the source or destination address must produce an aligned address according to the data type selected in the DATATYPE field of DMACSDP. Therefore, only certain values are valid for the element index.

Valid values for the element index are:

- [4 x N] + 1 (where N = ...–2, –1, 0, 1, 2...) if the data type is 32-bit
- [2 x N] + 1 (where N = ...–2, –1, 0, 1, 2...) if the data type is 16-bit
- Any value if the data type is 8-bit

As with the element index, the frame index must produce an aligned address according to the data type selected in the DATATYPE field of DMACSDP. Valid values for the frame index are:

- [4 x N] + 1 (where N = ...–2, –1, 0, 1, 2...) if the data type is 32-bit
- [2 x N] + 1 (where N = ...–2, –1, 0, 1, 2...) if the data type is 16-bit
- Any value if the data type is 8-bit

The start address, element index, and frame index must produce valid addresses within the range of the port. If an invalid address is generated, a timeout error will occur.

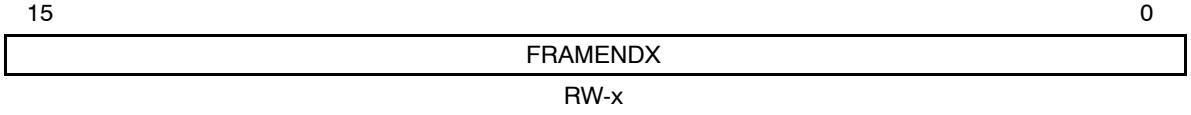
If the DSP core attempts to load an index register with an element or a frame index that would cause an unaligned address, the DMA controller will send a bus error interrupt (BERRINT) request to the DSP core. This occurs even if address indexing is not used.

The index registers are summarized by Figure 90 and Table 73–Table 76. All of the element index and frame index registers are uninitialized after a DSP subsystem reset.

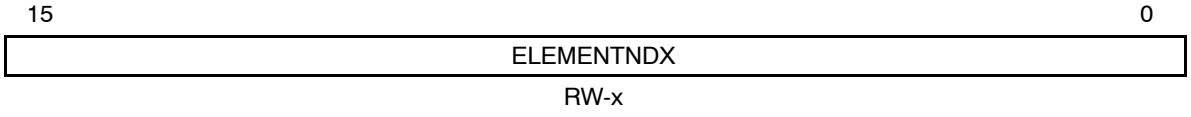
Figure 90. DMA Source Element Index Registers (DMACSEI, DMACDEI) and Frame Index Registers (DMACSFI, DMACDFI)

DMACEI/DMACSEI

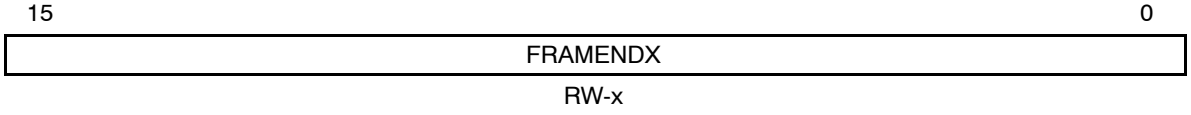
DMACFI/DMACSFI



DMACDEI



DMACDFI



Note: R = Read; W = Write; –n = Value after reset; –x = Value after DSP reset is not defined.

Table 73. DMA Source Element Index Register (DMACSEI/DMACEI) Field Descriptions

Bits	Field	Value	Description
15–0	ELEMENTNDX	–32768 to 32767	When DINDXMD = 0, DMACSEI is used as DMACEI; it contains the element index (in bytes) for both the source and the destination. When DINDXMD = 1, DMACSEI contains the source element index (in bytes).

Table 74. DMA Source Frame Index Register (DMACSFI / DMACFI) Field Descriptions

Bits	Field	Value	Description
15–0	FRAMENDX	–32768 to 32767	When DINDXMD = 0, DMACSFI is used as DMACFI; it contains the frame index (in bytes) for both the source and the destination. When DINDXMD = 1, DMACSFI contains the source frame index (in bytes).

Table 75. DMA Destination Element Index Register (DMACDEI) Field Descriptions

Bits	Field	Value	Description
15–0	ELEMENTNDX	–32768 to 32767	When DINDXMD = 1, DMACDEI contains the destination element index (in bytes).

Table 76. DMA Destination Frame Index Register (DMACDFI) Field Descriptions

Bits	Field	Value	Description
15–0	FRAMENDX	–32768 to 32767	When DINDXMD = 1, DMACDFI contains the destination frame index (in bytes).

7.3.12 DMA Source Address Counter (DMACSAC) and Destination Address Counter (DMACDAC)

The progress of each DMA channel can be monitored by reading the source and destination address counters (DMACSAC and DMACDAC). DMACSAC shows the low 16 bits of the current source address. DMACDAC shows the low 16 bits of the current destination address. The address counters are summarized by Figure 91, Table 77, and Table 78. DMACSAC and DMACDAC are uninitialized after a DSP subsystem reset.

Figure 91. DMA Source Address Counter (DMACSAC) and Destination Address Counter (DMACDAC)

DMACSAC

15	0
SAC	
RW-x	

DMACDAC

15	0
DAC	
RW-x	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after DSP reset is not defined.

Table 77. DMA Source Address Counter (DMACSAC) Field Descriptions

Bits	Field	Value	Description
15–0	SAC	0000h–FFFFh	Low 16 bits of current source address.

Table 78. DMA Destination Address Counter (DMACDAC) Field Descriptions

Bits	Field	Value	Description
15–0	DAC	0000h–FFFFh	Low 16 bits of current destination address.

8 TI Peripheral Bus Bridges

8.1 Introduction

The TI peripheral bus (TIPB) bridges manage accesses to peripheral control and data registers by the DSP core, DSP DMA controller, and MPU Interface (MPUI) via two peripheral buses (see the DSP subsystem block diagram in section 1.4):

- DSP private peripheral bus. Peripherals connected here cannot be accessed by the MPU via the MPUI port.
- DSP public peripheral bus. Peripherals connected here can be accessed by the MPU via the MPUI port.

There are two TIPB bridges in the DSP subsystem:

- The private TIPB bridge provides a preconfigured bus interface to peripherals residing on the DSP private peripherals bus.
- The public TIPB bridge provides a user-configurable interface to peripherals on the DSP public peripheral bus. It includes functions to configure the interface timing to the complement of peripherals operating at a given time.

All peripheral control and data registers are located in the I/O space. To read from or write to these registers, you must access the DSP subsystem I/O space either through C language constructs or by using the assembly language peripheral port register access qualifier. See the *TMS320C55x DSP Mnemonic Instruction Set Reference Guide* (SPRU374) for more details.

Note:

Byte access to I/O space is not supported.

8.2 DSP Private Peripherals

Peripherals on the DSP private peripheral bus are considered private peripherals. The MPU cannot access these peripherals. DSP private peripherals on OMAP5910 and OMAP5912 include:

- Three timers
- Watchdog timer
- Interrupt handlers

8.3 DSP Public Peripherals

Peripherals on the DSP public peripheral bus are considered public peripherals. These peripherals are directly accessible by the DSP core and DSP DMA. The MPU core can also access these peripherals through the MPUI port (see section 9). DSP public peripherals on OMAP5910 and OMAP5912 include:

- Two Multichannel Buffered Serial Ports (McBSP1 and McBSP3)
- Two Multichannel Serial Interfaces (MCSI1 and MCSI2)

8.4 DSP/MPU Shared Peripherals

The shared peripherals are connected to both the MPU public peripheral bus and the DSP public peripheral bus. Connections are achieved via a TI peripheral bus switch, which must be configured to allow MPU or DSP access. The other shared peripherals have permanent connections to both public peripheral buses, although read and write accesses to each peripheral register may differ.

DSP/MPU shared peripherals on OMAP5910 and OMAP5912 include:

- Mailbox registers
- Three Universal Asynchronous Receiver/Transmitter modules (UART1, UART2, and UART3)
- General-purpose I/O (GPIO)

OMAP5912 also includes the following DSP/MPU shared peripherals:

- Eight general purpose timers
- Serial Port Interface (SPI)
- I^2C master/slave interface
- Multichannel Buffered Serial Port 2 (McBSP2)
- Multimedia Card/Secure Digital Interface (MMC/SDIO2)
- 32-KHz synchronization timer

8.5 Peripheral Access Rate

The TIPB bridges control the access rate to peripherals by inserting a configurable amount of wait states into the strobe cycle of a peripheral register access. The access factor bits of the TIPB control mode register (CMR) specify the number of wait states of the strobe signal. The access rate to a peripheral depends on the number of wait states used. Table 79 lists the equivalent access rate for each of the possible number of wait states.

Table 79. TIPB Access Rates

Number of Wait States	Total Strobe Period (in DSP core clock cycles)	Equivalent Access Rate
0	2	DSP clock divided by 2
1	3	DSP clock divided by 3
2	4	DSP clock divided by 4
3	5	DSP clock divided by 5
4	6	DSP clock divided by 6
5	7	DSP clock divided by 7
6	8	DSP clock divided by 8
7	9	DSP clock divided by 9

Each access factor field in the control mode register controls the access rate for a group of peripherals. Table 80 and Table 81 show the peripherals that are affected by the ACCESS_FACTOR0 and ACCESS_FACTOR1 bits for OMAP5912 and OMAP5910, respectively.

Table 80. Peripherals Affected by Access Factor Bits (OMAP5912)

Access Factor Bits	Peripherals Affected
ACCESS_FACTOR0	TIPB registers
	CLKM2 registers
	DSP interrupt handler 2.0
	DSP interrupt handler 2.1
ACCESS_FACTOR1	UART3
	McBSP1
	McBSP3
	MCSI1
	MCSI2
	GPIO
	Mailbox
	DSP MPUI register
	TIPB switch
	GP timers (x8)
	32-KHz synchronization timer
	SPI
	I2C
	MMCSDIO2
	GPIO (x4)

Table 81. Peripherals Affected by Access Factor Bits (OMAP5910)

Access Factor Bits	Peripherals Affected
ACCESS_FACTOR0	TIPB registers
	CLKM2 registers
ACCESS_FACTOR1	UART3
	McBSP1
	McBSP3
	MCSI1
	MCSI2
	GPIO
	Mailbox
	DSP MPUI register

8.6 Peripheral Access Timeout

A timeout counter in TIPB bridge module counts how many cycles have passed since a request was made to access a peripheral. When the counter reaches 127 (default) DSP subsystem clock cycles, the TIPB bridge module sends a timeout signal the original requestor. The default timeout value can be changed with the TIMEOUT bits in the control mode register.

In response to a timeout, the TIPB bridge module sends the timeout signal to the requestor and sets the bus error bit of the CMR register. In the case of the DSP core access, the timeout signal sets the bus-error interrupt flag bit in the DSP core. The DSP core can respond to the interrupt request or ignore the interrupt request.

8.7 TIPB Register

8.7.1 Overview

The DSP TIPB bridge module is configured through a single register accessible through the DSP I/O space (see Table 82). The MPU core can also read the contents of this register.

Table 82. Register of the TIPB Bridge

Name	Description	DSP I/O Address†	MPU Byte Address†
CMR	Control mode register. This register is used to control the TIPB bridge module.	0x0000	0xE100 0000

† DSP I/O and MPU byte addresses apply to both OMAP5910 and OMAP5912.

8.7.2 TIPB Control Mode Register (CMR)

The control mode register (CMR) indicates the access mode of the MPUI and the bus error condition status for accesses to the TIPB bridge module. It also controls DSP core priority versus the MPUI and DSP DMA controller for accesses to peripherals on the DSP public peripheral bus. This register is accessible through both the MPU side and the DSP side; however, the MPU core has read access only. Figure 92 and Table 83 summarize CMR.

Figure 92. TIPB Control Mode Register (CMR)

DSP Side

15	9	8
TIMEOUT		ACCESS_ FACTOR1
RW-0x7F		RW-1

7	6	5	3	2	1	0
ACCESS_FACTOR1		ACCESS_FACTOR0		CPU_ PRIORITY	BUS_ ERROR	MPUI_ MODE
RW-1		RW-1		RW-1	R-0	R-1

MPU Side

31	16
Reserved	
R-0	

15	9	8
TIMEOUT		ACCESS_ FACTOR1
R-0x7F		R-0

7	6	5	3	2	1	0
ACCESS_FACTOR1		ACCESS_FACTOR0		CPU_ PRIORITY	BUS_ ERROR	MPUI_ MODE
R-0		R-1		R-1	R-0†	R-1

† The MPU core always reads this bit as a 0 when host-only mode is selected (see section 9.2.1).

Note: R = Read; W = Write; –n = Value after reset; -x = Value after reset is not defined.

Table 83. TIPB Control Mode Register (CMR) Field Descriptions

Bits	Field	Value	Description
15–9	TIMEOUT		This field determines the number of DSP subsystem clock cycles that can elapse before the TIPB bridge module returns a bus-error condition. The timeout period is determined as: Timeout Period = TIMEOUT + 2
8–6	ACCESS_ FACTOR1	0–7	These bits set the number of wait states inserted when communicating with peripherals as listed in Table 80 and Table 81.

Table 83. TIPB Control Mode Register (CMR) Field Descriptions (Continued)

Bits	Field	Value	Description
5–3	ACCESS_ FACTOR0	0–7	These bits set the number of wait states inserted when communicating with peripherals as listed in Table 80 and Table 81.
2	CPU_PRIORITY		This bit determines the priority of the DSP core, MPUI, and DSP DMA controller in the case of simultaneous accesses to the TIPB bridge.
		0	Accesses to the TIPB bridge are arbitrated in a rotating priority fashion: DSP core, MPUI, DSP DMA, DSP core, and so on.
		1	Accesses to the TIPB bridge have the following priorities in arbitration: 1) DSP core 2) MPUI 3) DSP DMA
1	BUS_ERROR		This bit is set when the TIPB generates a bus error to the DSP core because of a timeout condition or a host-only mode (HOM) change error or shared-access mode (SAM) change error. The BUS_ERROR bit is cleared when this register is read by the DSP core. This bit cannot be read when the MPUI is operating in host-only mode (register always reads as zero).
		0	No bus error has been generated by the TIPB.
		1	TIPB has generated a bus error to the DSP core.
0	MPUI_MODE		This bit indicates whether the MPUI is in host-only mode (HOM) or in shared-access mode (SAM) for peripherals. Section 9 describes these modes of the MPUI.
		0	MPUI is in shared-access mode.
		1	MPUI is in host-only mode.

9 MPU Interface Port

9.1 Introduction

The MPU interface (MPUI) port is a 16-bit parallel port that allows the MPU core and the system DMA controller to communicate with the DSP subsystem internal memory and its peripherals, facilitating software downloads and data transfers. The MPUI port is part of the DSP subsystem domain.

Note that there is also a module called MPUI in the MPU subsystem domain that connects to the MPUI port (see system block diagrams in section 1.4). MPU controllers, like the MPU core and system DMA, access DSP subsystem resources (internal memory and public peripherals) through the MPUI.

This section is intended to give a brief introduction of the communication between the MPUI port and the MPUI. For additional information, see the *OMAP5910 Dual-Core Processor MPU Subsystems Reference Guide* (SPRU671) or the *OMAP5912 Multimedia Processor OMAP3.2 Subsystem Reference Guide* (SPRU749).

9.2 MPUI and MPUI Port Overview

The MPUI port provides MPU controllers with access to the full internal memory space of the DSP subsystem. In addition, the MPUI port allows MPU controllers to access devices on the DSP public peripheral bus through duplicate memory-mapped peripheral registers in the MPU address space. The MPU controllers can also access the control registers of the TIPB bridge module and the CLKM2 configuration registers. The DSP private peripherals are not accessible via the MPUI port.

MPUI port transfers are facilitated by an auxiliary channel of the DSP subsystem DMA controller; however, this dedicated DMA channel is preconfigured and does not need to be configured for MPUI support.

The MPUI is always the master in the transfer operation. It initiates the reading or writing of DSP subsystem memory or peripherals. The MPU core also controls the parameters of the MPUI by configuring the MPUI_CTRL_REG and the MPUI_DSP_MPUI_CONFIG registers. There are five additional registers the MPU can read to observe the state of the MPUI:

- MPUI_DEBUG_ADDR
- MPUI_DEBUG_DATA
- MPUI_DEBUG_FLAG
- MPUI_STATUS_REG
- MPUI_DSP_STATUS_REG

For complete information on these registers, see the *OMAP5910 Dual-Core Processor MPU Subsystems Reference Guide* (SPRU671) and the *OMAP5912 Multimedia Processor OMAP3.2 Subsystem Reference Guide* (SPRU749).

9.2.1 MPUI Port Modes

The MPUI port supports two access modes for the DSP subsystem internal memory:

- Shared-access mode for internal memory (SAM_M). The DSP subsystem SARAM and DARAM are shared between the DSP domain and the MPU domain.
- Host-only mode for internal memory (HOM_M). In this mode, the MPU domain can only access SARAM. However, it can completely lock out the DSP controllers from a portion of the SARAM. The API_SIZE bits in the MPUI_DSP_CONFIG register specify the amount of SARAM that is dedicated to MPU domain accesses.

Similarly, the MPUI port supports two access modes for DSP subsystem peripherals on the DSP public peripheral bus:

- Shared-access mode for peripherals (SAM_P). The peripherals on the DSP public peripheral bus are shared between the DSP controllers and the MPU domain.
- Host-only mode for peripherals (HOM_P). The MPU domain has exclusive access to the peripherals on the DSP public peripheral bus.

SAM_M and SAM_P are the normal operating modes in which all the DSP subsystem internal memory and the public peripherals are accessible by the MPUI controllers, as well as the DSP controllers. When the DSP subsystem is taken out of reset, SAM_M and SAM_P are invoked automatically. In SAM_M, if both the DSP controllers and the MPU controllers access the same memory at the same time, priority is given to the DSP controllers. An MPU controller access in SAM_M is synchronized to the internal DSP core clock, which can add access latency to transfers.

HOM_M and HOM_P provide the MPU controllers with exclusive access to the DSP subsystem SARAM or public peripherals, primarily to support high-speed transfers from or to the DSP domain during DSP subsystem reset or IDLE conditions. When the DSP subsystem is placed in reset, HOM_M and HOM_P are invoked automatically. In HOM_M, the MPUI interface does not have access to the DARAM, but it has access to all of SARAM. Additionally, only the MPU controllers can access the DSP public peripheral bus in HOM_P.

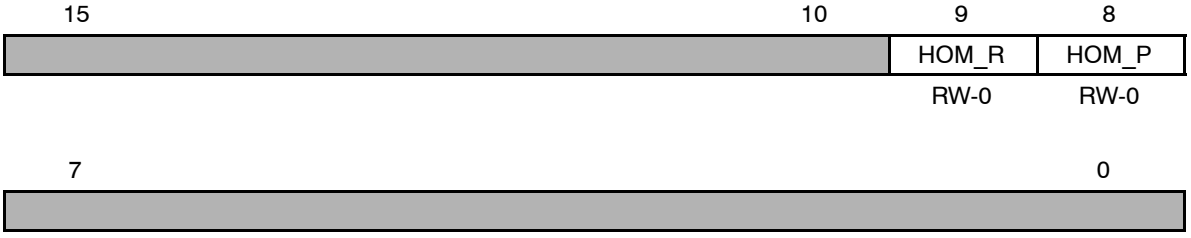
9.2.2 HOM/SAM Change Outside of Reset

Only the DSP core can invoke a change between the host-only and shared-access modes outside of reset. OMAP devices use two bits in the DSP core's ST3_55 register to change between SAM_M and HOM_M and between SAM_P and HOM_P: the HOM_R bit (bit 9) and the HOM_P bit (bit 8), respectively (see Figure 93 and Table 84).

The mode change is initiated by a DSP core write to the HOM_P bit and/or the HOM_R bit. When the appropriate bit is written to request the change, the mode change is not reflected on the HOM_P and HOM_R bits until an internal controller completes the mode switch. Therefore, the DSP core should poll the HOM_P and HOM_R bits after requesting a mode change to determine if the mode change is complete.

The mode status can be observed by the MPU core by reading the DSP Status Register (DSP_STATUS_REG).

Figure 93. MPUI Mode Change Bits in ST3_55



Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 84. HOM_R and HOM_P Bits in DSP Core Register ST3_55

Bits	Field	Value	Description
9	HOM_R		Memory access mode bit. This bit determines whether the MPUI port operates in SAM_M or HOM_M.
		0	SAM_M. The DSP SARAM and DARAM are shared by the MPUI and the DSP controllers.
		1	HOM_M. The DSP SARAM can be configured for exclusive access by the MPUI. The API_SIZE bits of the MPUI_DSP_CONFIG register specify the amount of SARAM that is dedicated to MPUI accesses.
8	HOM_P		DSP public peripherals access mode bit. This bit determines whether the MPUI port operates in SAM_P or HOM_P.
		0	SAM_P. DSP public peripherals are shared by the MPU and DSP domains.
		1	HOM_P. DSP public peripherals are owned only by the MPU domain.

10 DSP Subsystem Endianess

10.1 Endianess Within OMAP

The DSP subsystem modules operate in Big-Endian mode, contrary to the rest of the OMAP modules (including the MPU core and system DMA), which operate in Little-Endian mode. Shared data between the DSP subsystem and the rest of the OMAP modules must be converted to their respective formats before any processing. This process is called endianess conversion.

Table 85 details the Little- and Big-Endian data formats, assuming a byte-addressable architecture.

Table 85. Little-Endian versus Big-Endian Data Format

	32-bit Value: 0x1234 5678		16-bit Value: 0x1234		8-bit Value: 0x12	
4 LSBs of Byte Address	Little Endian	Big Endian	Little Endian	Big Endian	Little Endian	Big Endian
XX00b	0x78	0x12	0x34	0x12	0x12	0x12
XX01b	0x56	0x34	0x12	0x34		
XX10b	0x34	0x56				
XX11b	0x12	0x78				

Note: X = Don't care.

10.2 Endianess Conversion

Endianess conversion is not always necessary. Conversion is required when the following two conditions are met:

- Shared data is accessed using different Endian formats. For example, when the MPU core stores data in memory using Little-Endian format and the DSP core reads the same data using Big-Endian data format.
- Shared data is accessed using different access sizes. For example, when the MPU core stores data using 32-bit accesses while the DSP core reads the same data with 16-bit accesses.

The following statements can be made about endianess data formats and data access size.

- For shared data stored as a 32-bit element:
  - If this 32-bit element is accessed with one 32-bit access, then no endianess conversion is required; this is independent of the endianess of the accessing processor.
  - If this 32-bit element is accessed with two 16-bit accesses using a different Endian format, then the two 16-bit halves within this 32-bit element must be swapped.
  - If this 32-bit element is accessed with four 8-bit accesses using a different Endian format, then the order of the four bytes within this 32-bit word must be reversed.
- For shared data stored as a 16-bit element:
  - If this 16-bit element is accessed with one 16-bit access, then no endianess conversion is required; this is independent of the endianess of the accessing processor.
  - If this16-bit element is accessed with two 8-bit accesses using a different Endian format, then the two bytes within this 16-bit element must be swapped.

Table 86 shows the results of different accesses from a Big-Endian processor to Little-Endian byte-addressable memory storing the 32-bit data element 0x1234 5678.

Table 86. Big-Endian Access of Little-Endian Data

4 LSBs of Byte Address	32-bit Value (0x1234 5678) stored as Little Endian	Single 32-Bit Read Access	Two 16-Bit Read Accesses	Four 8-Bit Read Accesses
XX00b	0x78	0x1234 5678	0x5678	0x78
XX01b	0x56			0x56
XX10b	0x34		0x1234	0x34
XX11b	0x12			0x12

Note: X = Don't care.

10.3 Endianess Conversion Modules

There are three distinct scenarios where the DSP subsystem modules and the rest of the OMAP modules access the same resources, and endianess conversion might be required.

- MPU core or system DMA accesses to the DSP subsystem internal memory via the MPU Interface (MPUI). Endianess conversion is performed in the MPUI.
- DSP core or DSP DMA accesses to DSP external memory via the DSP memory management unit (MMU) and the traffic controller. Endianess conversion is performed in the DSP MMU.
- DSP core and MPU core accesses to shared peripherals. No endianess conversion is performed, as both the DSP core and the MPU core see the same data.

Because the endianess conversion is performed in hardware, data swapping is transparent to software. This reduces the software overhead needed to format the data. The swapping logic can be disabled and the conversion handled by software, as desired.

10.3.1 Endianess Conversion by the DSP MMU

Endianess conversion is performed at the boundary between the DSP subsystem and the DSP MMU. The endianess conversion unit of the DSP MMU splits data accesses into individual bytes and reorders them according to the access type and chosen configuration.

The DSP MMU endianess conversion logic is configured by the MPU core through the DSP MMU endianess configuration register (see section 10.3.1.1). This register contains two configuration bits that control whether, and how, endianess conversion is performed:

- The first bit, EN, enables or disables endianess conversion.
- The second bit, SWAP, selects the type of endianess conversion to be performed– either swapping the 16-bit words only or swapping both the 16-bit words and the bytes within the words. Note that the 16-bit word swapping applies only to 32-bit accesses.

Table 87 lists the effects of the different configuration settings of the DSP MMU endianess conversion logic.

Table 87. Effect of DSP MMU Endianess Conversion Settings

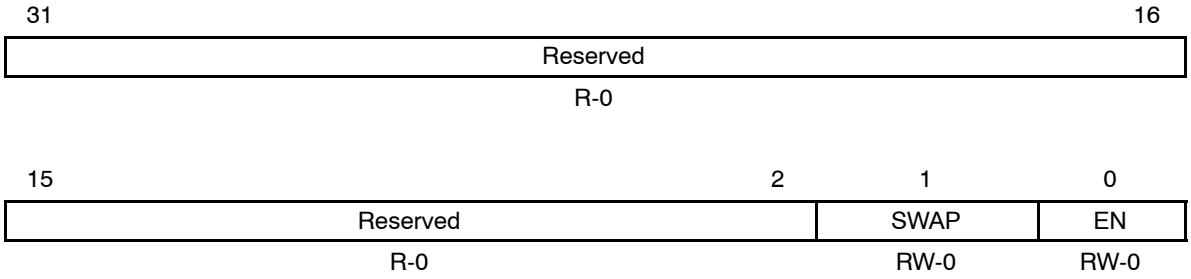
DSP-Side Read Access to MPU-Side Data Value 0x1234 5678			
Conversion Enable	**Word/Byte Swap**	**16-Bit Access**	**32-Bit Access**
Disabled (EN = 0)	Don't care (SWAP = X)	0x5678	0x1234 5678
Enabled (EN = 1)	Byte and word swap (SWAP = 0)	0x7856	0x7856 3412
Enabled (EN = 1)	Word swap only (SWAP = 1)	0x5678	0x5678 1234

Typically, both the 16-bit words and the bytes within each 32-bit word are swapped if the MPU has written data using four 8-bit accesses and is read as one 32-bit word by the DSP. In contrast, only the 16-bit words (but not the bytes within them) are swapped when the MPU has written data in two 16-bit accesses and the DSP reads them using one 32-bit access. No endianess conversion is required if both the MPU and the DSP access the data as 32-bit.

10.3.1.1 DSP MMU Endianess Control Register (DSP_ENDIAN_CONV)

The DSP MMU endianess conversion unit is configured by the MPU core using the DSP MMU endianess control register (Figure 94 and Table 88) located at address 0xFFFE CC34.

Figure 94. DSP MMU Endianess Control Register (DSP_ENDIAN_CONV)



Note: R = Read, W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 88. DSP MMU Endianess Control Register (DSP_ENDIAN_CONV) Field Descriptions

Bits	Field	Value	Description
31–2	Reserved		These bits are not used.
1	SWAP		When the DSP MMU endianess conversion unit is enabled, this bit selects the swapping mode.
		0	Swap bytes and 16-bit words.
		1	Swap 16-bit words only.
0	EN		This bit is used to enable the endianess configuration unit of the DSP MMU.
		0	Endianess conversion unit is disabled.
		1	Endianess conversion unit is enabled. Endianess conversion is carried out as dictated by the SWAP bit.

10.3.2 Endianess Conversion by the MPUI

The MPU Interface (MPUI) enables accesses from MPU core and system DMA to the DSP subsystem internal resources. Internal resources include the DSP subsystem internal memory (through the MPUI port, section 9) and the DSP public peripherals (through the public TIPB Bridge, section 8).

The endianess conversion unit within the MPUI offers the options of controlling byte swapping and word swapping depending on the access type (memory or peripheral). Its architecture is similar to that used within the DSP MMU.

The endianess conversion unit of the MPUI is configured by the MPU core through the MPUI control register (see section 10.3.2.1). This register contains two configuration bit fields that control the way the endianess conversion is handled:

- The WORD_SWAP bit field determines whether word swapping is performed on accesses to DSP subsystem internal memory, shared peripherals, or both.
- The BYTE_SWAP bit field determines whether byte swapping is performed on accesses to DSP subsystem internal memory, shared peripherals, or both.

The results of the different endianess settings for MPUI memory accesses are shown in Table 89.

Table 89. Effect of MPUI Endianess Conversion Settings

MPU-Side Read Access to DSP-Side Data Value 0x1234 5678			
Word Swap	**Byte Swap**	**16-Bit Access**	**32-Bit Access**
Off	Off	0x1234	0x12345678
Off	On	0x3412	0x34127856
On	Off	0x1234	0x56781234
On	On	0x3412	0x78563412

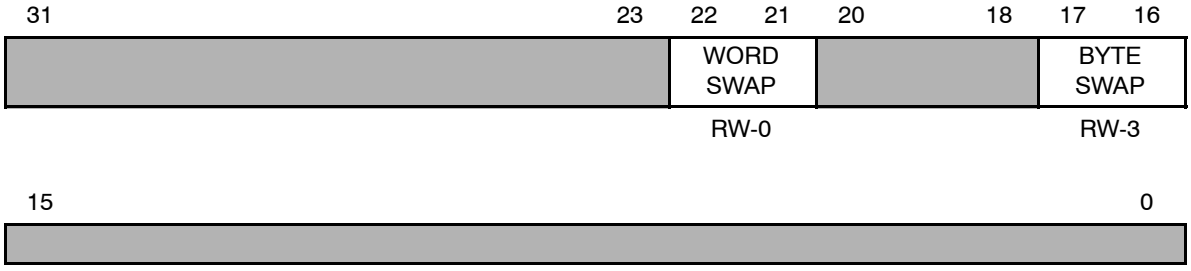
When using the same access size on the MPU and the DSP, no endianess conversion is required. In this case, both word swapping and byte swapping should be disabled. Otherwise, word and byte swapping must be used as specified by your application.

Typically, neither byte nor word swapping are needed when accessing the DSP peripheral registers, as the register size is predefined.

10.3.2.1 MPUI Control Register (CTRL_REG)

The endianess conversion unit of the MPUI is configured by the MPU core using the MPUI control register (CTRL_REG). The MPUI control register is located at address 0xFFFE C900 in the memory space of the MPU core. Figure 95 and Table 90 describe the bits in the MPUI control register that affect the endianess of DSP subsystem accesses through the MPUI.

Figure 95. MPUI Control Register (CTRL_REG)



Note: R = Read, W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 90. MPUI Control Register (CTRL_REG) Field Descriptions

Bits	Field	Value	Description
31–23	Reserved		These bits are not used.
22–21	WORD SWAP		These bits enable/disable word-swapping in the endianess conversion logic of the MPUI.
		00	Word-swap all accesses.
		01	Word-swap only peripheral accesses.
		10	Word-swap only memory accesses.
		11	Disable word-swapping.
20–18	Reserved		These bits are not used.
17–16	BYTE SWAP		These bits enable/disable byte-swapping in the endianess conversion logic of the MPUI.
		00	Disable byte-swapping.
		01	Byte-swap only peripheral accesses.
		10	Byte-swap all accesses.
		11	Byte-swap only memory accesses.
15–0	Reserved		These bits are not used.

11 DSP Subsystem Interrupts

A number of interrupts can be generated inside and outside the DSP subsystem. The DSP core is responsible for servicing these interrupts. This section gives a brief overview of interrupt setup and handling for the DSP subsystem. For more details on DSP subsystem interrupts, see the *OMAP5912 Multimedia Processor Interrupts Reference Guide* (SPRU757) or the *OMAP5910 Multimedia Processor DSP Subsystem Interrupts Reference Guide* (SPRU923).

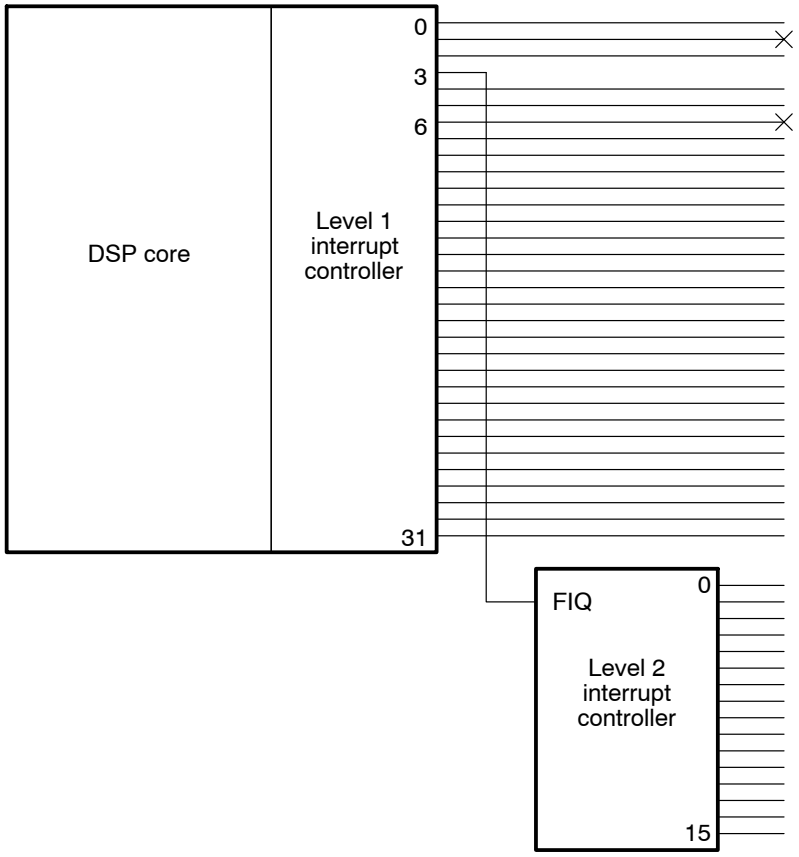
11.1 Overview

DSP core interrupts on OMAP devices are cascaded through a two levels of interrupt handlers. A second-level interrupt controller(s) takes a number of interrupts and generates a single interrupt to a first-level interrupt controller. The first-level interrupt controller manages the interrupt(s) from the second-level interrupt controller(s) and a number of other interrupts. The first-level interrupt controller interfaces directly to the DSP core. Figure 96 and Figure 97 show this process. The number of interrupt controllers varies across OMAP devices, but the concept of cascading interrupts is the same.

OMAP5910 contains two interrupt controllers for the DSP subsystem:

- One DSP level 2 interrupt controller (referred to as DSP interrupt level 2) that can handle 16 interrupts.
- One DSP level 1 interrupt controller (referred to as DSP interrupt level 1) that can handle 32 interrupts.

Figure 96. OMAP5910 DSP Subsystem Interrupts

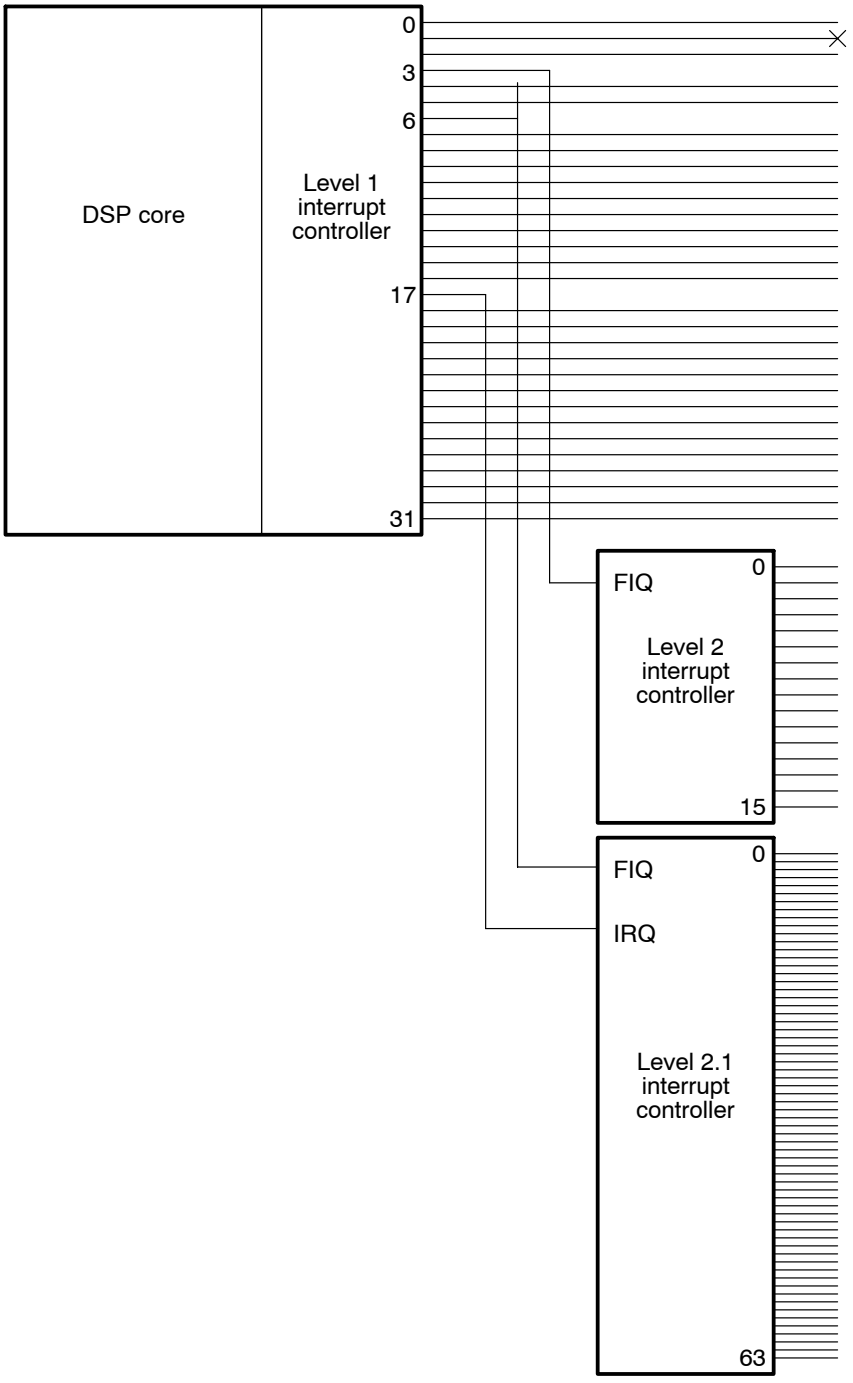


Note: × – No connection.

OMAP5912 contains three interrupt controllers for the DSP subsystem, see Figure 97 on the next page:

- One DSP level 2.1 interrupt controller (referred to as DSP interrupt level 2.1) that can handle 64 interrupts.
- One DSP level 2.0 interrupt controller (referred to as DSP interrupt level 2.0) that can handle 16 interrupts.
- One DSP level 1 interrupt controller that can handle 32 interrupts (DSP).

Figure 97. OMAP5912 DSP Subsystem Interrupts



Note: × – No connection.

11.2 First Level Interrupts

The C55x DSP core supports 32 level 1 interrupts. After receiving and acknowledging an interrupt request, the DSP core generates an interrupt vector address. At the vector address, the core fetches the vector that points to the corresponding interrupt service routine (ISR). When multiple hardware interrupts occur simultaneously, the DSP core services them one at a time, according to their predefined hardware interrupt priorities.

Vector pointers IVPD and IVPH point to up to 32 interrupt vectors in program space. IVPD points to the 256-byte program page for interrupt vectors 0-15 and 24-31. IVPH points to the 256-byte program page for interrupt vectors 16-23.

The DSP core supports two types of interrupts: maskable interrupts and nonmaskable interrupts. Maskable interrupts can be blocked (masked) or enabled (unmasked) through software. Each maskable interrupt has a corresponding bit in an interrupt enable register (IER0 or IER1) and an interrupt flag register (IFR0 or IFR1). Maskable interrupts include the interrupts associated with vectors 2-23, a bus error interrupt, a data log interrupt, and a real-time operating system interrupt.

Whenever a maskable interrupt is requested by hardware, its corresponding interrupt flag is set in one of the interrupt flag registers. Once the flag is set, the interrupt is not serviced unless it is enabled through the interrupt enable registers. The ISRs for the maskable interrupts can also be executed by software with the use of the INTR and TRAP assembly instructions.

When the DSP core receives a nonmaskable interrupt request, the DSP core acknowledges it unconditionally and immediately branches to the corresponding interrupt service routine (ISR). The nonmaskable interrupts include RESET and any software interrupts initiated through the use of the INTR and TRAP instructions.

Note:

$\overline{\text{NMI}}$ is not available on OMAP devices.

For more details on C55x DSP core interrupts, see the *TMS320C55x DSP CPU Reference Guide* (SPRU371).

11.2.1 OMAP5910 First Level Interrupt Mapping and Interrupt Registers

Table 91 shows the level 1 interrupts sorted by interrupt vector number for OMAP5910. Figure 98 and Figure 99 show the bit layout for the IFR0/IER0 and IFR1/IER1 registers, respectively.

Table 91. OMAP5910 Level 1 Interrupt Mapping

Hardware Interrupt Priority	C55x DSP Core Vector Name	Vector Address (Byte Address)	Name	Interrupt Source
1	RESETIV (IV0)	IVPD:00h	RESET	DSP reset (hardware or software) interrupt
2	NMIIV (IV1)	IVPD:08h	NMI[†]/SINT1	Hardware nonmaskable interrupt (or software interrupt #1)
3	IV2	IVPD:10h	EMUINT	DSP emulator/test interrupt
5	IV3	IVPD:18h	L2FIQ	DSP level 2 interrupt handler FIQ
6	IV4	IVPD:20h	TCABORT	Traffic controller abort interrupt
7	IV5	IVPD:28h	MBX1	MPU-to-DSP mailbox 1 interrupt
9	IV6	IVPD:30h	SINT6	Software interrupt #6
10	IV7	IVPD:38h	GPIO	Interrupt for DSP-owned shared GPIO
11	IV8	IVPD:40h	TIMER3	DSP private timer #3 interrupt
13	IV9	IVPD:48h	DMAC1	DSP DMA channel #1 interrupt
14	IV10	IVPD:50h	MPU	MPU interrupt to DSP
15	IV11	IVPD:58h	SINT11	Software interrupt #11
17	IV12	IVPD:60h	UART3	UART #3 interrupt
18	IV13	IVPD:68h	WDT	DSP watchdog timer interrupt
21	IV14	IVPD:70h	DMAC4	DSP DMA channel #4 interrupt
22	IV15	IVPD:78h	DMAC5	DSP DMA channel #5 interrupt
4	IV16	IVPH:80h	EMIF	Interrupt for DMA EMIF interface to traffic controller
8	IV17	IVPH:88h	LCLBUS	Local bus interrupt

[†] $\overline{\text{NMI}}$ is not physically connected on OMAP devices, it is included here for compatibility with other C55x documentation.

Table 91. OMAP5910 Level 1 Interrupt Mapping (Continued)

Hardware Interrupt Priority	C55x DSP Core Vector Name	Vector Address (Byte Address)	Name	Interrupt Source
12	IV18	IVPH:90h	DMAC0	DSP DMA channel #0 interrupt
16	IV19	IVPH:98h	MBX2	MPU-to-DSP mailbox #2 interrupt
19	IV20	IVPH:A0h	DMAC2	DSP DMA channel #2 interrupt
20	IV21	IVPH:A8h	DMAC3	DSP DMA channel #3 interrupt
23	IV22	IVPH:B0h	TIMER2	DSP private timer #2 interrupt
24	IV23	IVPH:B8h	TIMER1	DSP private timer #1 interrupt
25	BERRIV (IV24)	IVPD:C0h	BERR	Bus error interrupt
26	DLOGIV (IV25)	IVPD:C8h	DLOG	Data log interrupt
27	RTOSIV (IV26)	IVPD:D0h	RTOS	Real-time operating system interrupt
28	SIV27	IVPD:D8h	SINT27	Software interrupt #27
29	SIV28	IVPD:E0h	SINT28	Software interrupt #28
30	SIV29	IVPD:E8h	SINT29	Software interrupt #29
31	SIV30	IVPD:F0h	SINT30	Software interrupt #30
32	SIV31	IVPD:F8h	SINT31	Software interrupt #31

† $\overline{\text{NMI}}$ is not physically connected on OMAP devices, it is included here for compatibility with other C55x documentation.

Figure 98. IFR0 and IER0 Bit Locations (OMAP5910)

15	14	13	12	11	10	9	8
DMAC5	DMAC4	WDT	UART3	SINT11	MPU	DMAC1	TIMER3
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
GPIO	SINT6	MBX1	TCABORT	L2FIQ	EMUINT	Reserved	
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Figure 99. IFR1 and IER1 Bit Locations (OMAP5910)

15					11	10	9	8
Reserved						RTOS	DLOG	BERR
R-0						RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
TIMER1	TIMER2	DMAC3	DMAC2	MBX2	DMAC0	LCLBUS	EMIF
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

11.2.2 OMAP5912 First Level Interrupt Mapping and Interrupt Registers

Table 92 shows the level 1 interrupt sorted by interrupt number for OMAP5912. The bit layout for the IER0/IFR0 and IER1/IFR1 registers is shown in Figure 100 and Figure 101, respectively.

Table 92. OMAP5912 Level 1 Interrupt Mapping

Hardware Interrupt Priority	C55x DSP Core Vector Name	Vector Address (Byte Address)	Name	Interrupt Source
1	RESETIV (IV0)	IVPD:00h	RESET	DSP reset (hardware or software) interrupt
2	NMIIV (IV1)	IVPD:08h	$\overline{\text{NMI}}$†/SINT1	Hardware nonmaskable interrupt (or software interrupt #1)
3	IV2	IVPD:10h	EMUINT	DSP emulator/test interrupt
5	IV3	IVPD:18h	L20IRQ	DSP Level 2.0 interrupt handler IRQ
6	IV4	IVPD:20h	TCABORT	Traffic controller abort interrupt
7	IV5	IVPD:28h	MBX1	MPU-to-DSP mailbox 1 interrupt
9	IV6	IVPD:30h	L21FIQ	DSP level 2.1 interrupt handler FIQ
10	IV7	IVPD:38h	IRQ2_GPIO1	GPIO #1 interrupt

† $\overline{\text{NMI}}$ is not physically connected on OMAP devices, it is included here for compatibility with other C55x documentation.

Table 92. OMAP5912 Level 1 Interrupt Mapping (Continued)

Hardware Interrupt Priority	C55x DSP Core Vector Name	Vector Address (Byte Address)	Name	Interrupt Source
11	IV8	IVPD:40h	TIMER3	DSP private timer #3 interrupt
13	IV9	IVPD:48h	DMAC1	DSP DMA channel #1 interrupt
14	IV10	IVPD:50h	MPU	MPU interrupt to DSP
15	IV11	IVPD:58h	SINT11	Software interrupt #11
17	IV12	IVPD:60h	UART3	UART #3 interrupt
18	IV13	IVPD:68h	WDT	DSP watchdog timer interrupt
21	IV14	IVPD:70h	DMAC4	DSP DMA channel #4 interrupt
22	IV15	IVPD:78h	DMAC5	DSP DMA channel #5 interrupt
4	IV16	IVPH:80h	EMIF	Interrupt for DMA EMIF interface to traffic controller
8	IV17	IVPH:88h	L21IRQ	Level 2.1 interrupt Handler IRQ
12	IV18	IVPH:90h	DMAC0	DSP DMA channel #0 interrupt
16	IV19	IVPH:98h	MBX2	MPU-to-DSP mailbox #2 interrupt
19	IV20	IVPH:A0h	DMAC2	DSP DMA channel #2 interrupt
20	IV21	IVPH:A8h	DMAC3	DSP DMA channel #3 interrupt
23	IV22	IVPH:B0h	TIMER2	DSP private timer #2 interrupt
24	IV23	IVPH:B8h	TIMER1	DSP private timer #1 interrupt
25	BERRIV (IV24)	IVPD:C0h	BERR	Bus error interrupt
26	DLOGIV (IV25)	IVPD:C8h	DLOG	Data log interrupt

† $\overline{\text{NMI}}$ is not physically connected on OMAP devices, it is included here for compatibility with other C55x documentation.

Table 92. OMAP5912 Level 1 Interrupt Mapping (Continued)

Hardware Interrupt Priority	C55x DSP Core Vector Name	Vector Address (Byte Address)	Name	Interrupt Source
27	RTOSIV (IV26)	IVPD:D0h	RTOS	Real-time operating system interrupt
28	SIV27	IVPD:D8h	SINT27	Software interrupt #27
29	SIV28	IVPD:E0h	SINT28	Software interrupt #28
30	SIV29	IVPD:E8h	SINT29	Software interrupt #29
31	SIV30	IVPD:F0h	SINT30	Software interrupt #30
32	SIV31	IVPD:F8h	SINT31	Software interrupt #31

† $\overline{\text{NMI}}$ is not physically connected on OMAP devices, it is included here for compatibility with other C55x documentation.

Figure 100. IFR0 and IER0 Bit Locations (OMAP5912)

15	14	13	12	11	10	9	8
DMAC5	DMAC4	WDT	UART3	SINT11	MPU	DMAC1	TIMER3
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IRQ2_ GPIO1	L21FIQ	MBX1	TCABORT	L20IRQ	EMUINT	Reserved	
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Figure 101. IFR1 and IER1 Bit Locations (OMAP5912)

15	11	10	9	8
Reserved		RTOS	DLOG	BERR
R-0		RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
TIMER1	TIMER2	DMAC3	DMAC2	MBX2	DMAC0	L21IRQ	EMIF
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

11.3 Second Level Interrupts

OMAP devices include a second level of interrupt handlers which take a number of peripheral interrupts and generate one or two interrupts to the first level interrupt handler. For each interrupt, you must specify whether the interrupt is edge or level sensitive.

On OMAP5910, the level 2 interrupt handler generates a single interrupt to INT3 on the level 1 interrupt handler. The level 2.0 interrupt handler on OMAP5912 generates a single interrupt to INT3 on the level 1 interrupt handler, while level 2.1 generates a single interrupt to INT17. Table 93, Table 94, and Table 95 detail the interrupt mapping for each device.

Table 93. OMAP5910 Level 2 Interrupt Mapping

Level 2 Mapping	Name	Required Sensitivity Setup	Function
IRQ_0	MCBSP3_TX	Edge	McBSP #3 transmit interrupt
IRQ_1	MCBSP3_RX	Edge	McBSP #3 receive interrupt
IRQ_2	MCBSP1_TX	Edge	McBSP #1 transmit interrupt
IRQ_3	MCBSP1_RX	Edge	McBSP #1 receive interrupt
IRQ_4	UART2	Level	UART #2 interrupt
IRQ_5	UART1	Level	UART #1 interrupt
IRQ_6	MCSI1_TX	Level	MCSI #1 transmit interrupt
IRQ_7	MCSI1_RX	Level	MCSI #1 receive interrupt
IRQ_8	MCSI2_TX	Level	MCSI #2 transmit interrupt
IRQ_9	MCSI2_RX	Level	MCSI #2 receive interrupt
IRQ_10	MCSI1_FRAME_ERROR_INT	Level	MCSI #1 frame error interrupt
IRQ_11	MCSI2_FRAME_ERROR_INT	Level	MCSI #2 frame error interrupt
IRQ_12	–	–	Reserved, keep masked
IRQ_13	–	–	Reserved, keep masked
IRQ_14	–	–	Reserved, keep masked
IRQ_15	–	–	Reserved, keep masked

Table 94. OMAP5912 Level 2.0 Interrupt Mapping

Level 2 Mapping	Name	Required Sensitivity Setup	Function
IRQ_0	MCBSP3_TX	Level	McBSP #3 transmit interrupt
IRQ_1	MCBSP3_RX	Level	McBSP #3 receive interrupt
IRQ_2	MCBSP1_TX	Level	McBSP #1 transmit interrupt
IRQ_3	MCBSP1_RX	Level	McBSP #1 receive interrupt
IRQ_4	UART2	Level	UART #2 interrupt
IRQ_5	UART1	Level	UART #1 interrupt
IRQ_6	MCSI1_TX	Level	MCSI #1 transmit interrupt
IRQ_7	MCSI1_RX	Level	MCSI #1 receive interrupt
IRQ_8	MCSI2_TX	Level	MCSI #2 transmit interrupt
IRQ_9	MCSI2_RX	Level	MCSI #2 receive interrupt
IRQ_10	MCSI1_FRAME_ERROR_INT	Level	MCSI #1 frame error interrupt
IRQ_11	MCSI2_FRAME_ERROR_INT	Level	MCSI #2 frame error interrupt
IRQ_12	IRQ2_GPIO2	Level	GPIO #2 interrupt
IRQ_13	IRQ2_GPIO3	Level	GPIO #3 interrupt
IRQ_14	IRQ2_GPIO4	Level	GPIO #4 interrupt
IRQ_15	I^2C	Level	I^2C interrupt

Table 95. OMAP5912 Level 2.1 Interrupt Mapping

Level 2 Mapping	Name	Required Sensitivity Setup	Function
IRQ_0	NAND	Level	NAND flash interrupt
IRQ_1	GPTIMER1	Level	General purpose timer #1 interrupt
IRQ_2	GPTIMER2	Level	General purpose timer #2 interrupt
IRQ_3	GPTIMER3	Level	General purpose timer #3 interrupt
IRQ_4	GPTIMER4	Level	General purpose timer #4 interrupt

Table 95. OMAP5912 Level 2.1 Interrupt Mapping (Continued)

Level 2 Mapping	Name	Required Sensitivity Setup	Function
IRQ_5	GPTIMER5	Level	General purpose timer #5 interrupt
IRQ_6	GPTIMER6	Level	General purpose timer #6 interrupt
IRQ_7	GPTIMER7	Level	General purpose timer #7 interrupt
IRQ_8	GPTIMER8	Level	General purpose timer #8 interrupt
IRQ_9	–	–	Reserved, keep masked
IRQ_10	MCBSP2_TX	Level	McBSP #2 transmit interrupt
IRQ_11	MCBSP2_RX	Level	McBSP #2 transmit interrupt
IRQ_12	–	–	Reserved, keep masked
IRQ_13	–	–	Reserved, keep masked
IRQ_14	MMC_SDIO2	Level	MMC/SDIO #2 interrupt
IRQ_15	SPI	Level	SPI interrupt
IRQ_16 through IRQ_63	–	–	Reserved, keep masked

12 DSP Subsystem Reset, Clocking, Idle Control, and Boot

12.1 Reset Control

The DSP subsystem can be reset through hardware or software. The following sections provide a brief overview of the different sources which generate hardware and software reset signals for the DSP subsystem.

Complete information on OMAP hardware and software reset control can be found in the *OMAP5910 Dual-Core Processor Clock Generation and System Reset Management Reference Guide* (SPRU678) or the *OMAP5912 Multimedia Processor Initialization Reference Guide* (SPRU752).

12.1.1 Hardware (Cold) Resets

OMAP5912 and OMAP5910 devices have external pins which can be used to generate hardware resets. On the OMAP5910/5912 devices, these pins include $\overline{\text{PWRON\_RESET}}$ and $\overline{\text{MPU\_RST}}$, The OMAP5912 device also has the RTC_ON_NOFF pin. Activating any of these pins will reset the DSP subsystem modules, including the DSP MMU, TIPB, and MPUI.

12.1.2 Software (Warm) Resets

The OMAP clock generation and system reset module manages the software reset signals of the DSP subsystem. The MPU-Reset-Control-1 register (ARM_RSTCT1) can generate three types of software resets that affect the DSP subsystem. These software resets are:

- Global software reset.

  The DSP, the MPU, and all internal modules are reset when the SW_RST bit is set.

- DSP interface modules reset.

  The priority registers (TIPB), EMIF configuration registers, and MPUI control logic (partially) of the DSP subsystem are reset when the DSP_RST bit is cleared.

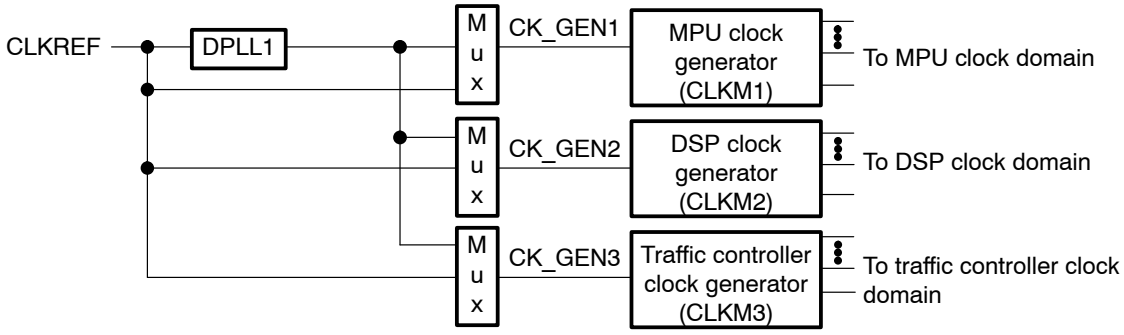
- DSP subsystem (excluding interface modules) reset.

  The modules within the DSP subsystem (excluding the interface modules) are reset when the DSP_EN bit is cleared. The DSP_EN bit must be cleared after a cold reset to take the DSP subsystem out of reset.

12.2 Clock Source

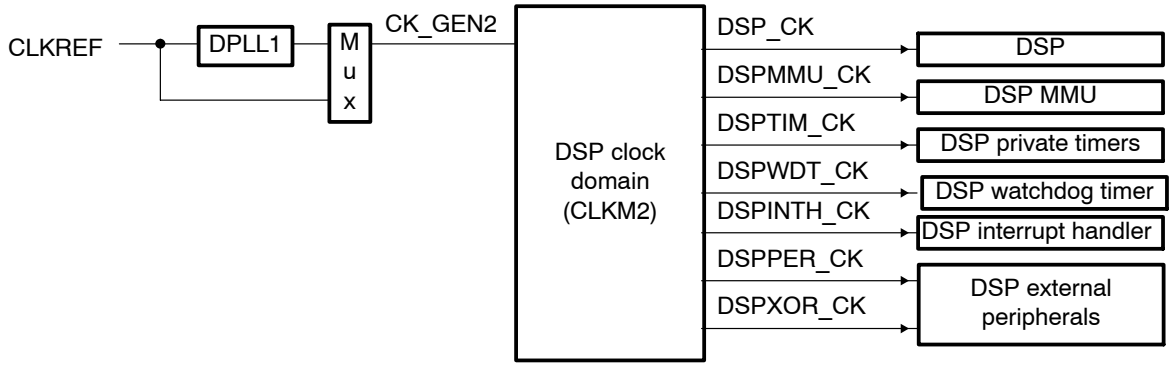
The OMAP clock system is organized around three main clock domains: the MPU, DSP, and traffic controller clock domains. There is a clock generator associated with each clock domain (see Figure 102). The input clock to the three clock generators can be taken from two sources: the output of a digital phase locked loop module called DPLL1, or the input clock source to the DPLL1 module. The input clock source to DPLL1, called CLKREF, can be generated through the use of an on-chip oscillator or supplied by an external device.

Figure 102. OMAP Clock Generation



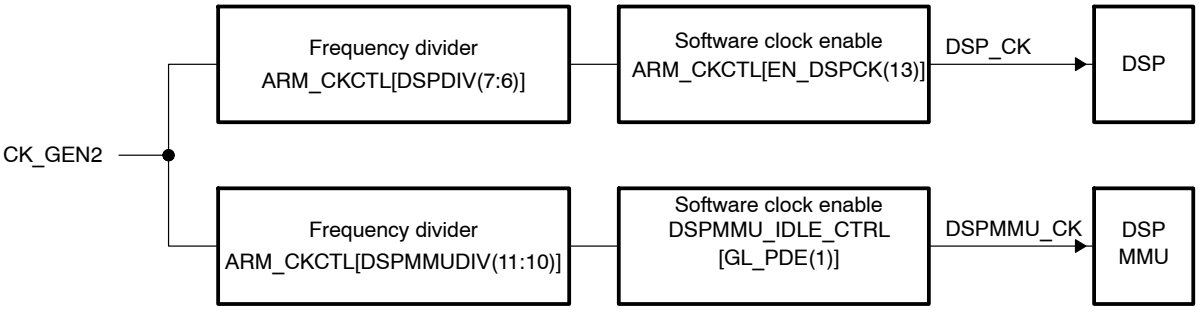
The DSP clock domain output clock, CK_GEN2, is further distributed to different modules within the DSP subsystem (Figure 103). The frequencies of these individual clock signals can be programmed through the use of various clock dividers within the DSP clock generator.

Figure 103. DSP Clock Domain



The DSP clock generator on both OMAP5910 and OMAP5912 provides two clocks which affect most of the modules described in this reference guide: the DSP subsystem master clock and the DSP MMU clock (see Figure 104).

Figure 104. Generation of DSP Subsystem Master Clock and DSP MMU Clock



The DSP subsystem master clock, DSP_CK, feeds all the modules included in the DSP module (section 1.2 lists these modules). The DSPDIV bits in the clock control register (ARM_CKCTL) specify the divider value used to generate DSP_CK from the DSP domain clock. By default, the DSPDIV bits select the divide-by-one mode. The EN_DSPCK bit in the same register enables the DSP_CK (by default DSP_CK is enabled).

The DSP MMU clock, DSPMMU_CK, is generated from the DSP domain clock similarly to DSP_CK. The DSPMMUDIV bits in the ARM_CKCTL register specify the divider value used to generate the DSPMMU_CK from the DSP domain clock. By default, the DSPMMUDIV bits select the divide-by-one mode. DSPMMU_CK can be shut off by setting the GL_PDE bit of the DSPMMU_IDLE_CTRL register (section 6.5.17).

The OMAP clock generation and system reset module manages operations such as the reset sequences, the clock generation function, the power-saving modes, idle controls, and setup for the OMAP5912. The MPU manages the master clock configuration for the OMAP5912 device.

For more information on OMAP clock architecture and control, see the following documents: *OMAP5912 Multimedia Processor Clocks Reference Guide* (SPRU751), the O*MAP5912 Multimedia Processor OMAP3.2 Subsystem Reference Guide* (SPRU749), and the *OMAP5910 Dual-Core Processor Clock Generation and System Reset Management Reference Guide* (SPRU678).

12.3 Idle Control

The DSP subsystem can be idled at two levels: at the DSP subsystem level and at the DSP module level. Idling at the subsystem level is quick and simple; however, it requires that the DSP subsystem be reset, thereby eliminating any currently executing application. To preserve the state of the DSP subsystem, the subsystem must be idled at the DSP module level. The following sections describe both approaches.

12.3.1 Idle Control at the DSP Subsystem Level

As mentioned in section 12.2, the main clock that feeds the DSP subsystem is the DSP subsystem master clock (DSP_CK). DSP_CK feeds all the modules included in the DSP module (a list of these modules is included in section 1.2). The EN_DSPCK bit in the MPU Clock Control Prescaler Selection Register (ARM_CKCTL) enables the DSP_CK (by default DSP_CK is enabled).

To idle the DSP subsystem, follow these steps:

1) Place the DSP subsystem in reset by clearing the DSP_EN bit in the Master Software Reset Register (ARM_RSTCT1).

2) Disable the DSP subsystem clock by clearing the EN_DSPCK bit in the MPU Clock Control Prescaler Selection Register to (ARM_CKCTL).

For more information on the ARM_RSTCT1 and ARM_CKCTL registers, see the *OMAP5912 Multimedia Processor OMAP3.2 Subsystem Reference Guide* (SPRU749), or the *OMAP5910 Dual-Core Processor Clock Generation and System Reset Management Reference Guide* (SPRU678).

12.3.2 Idle Control at the DSP Module Level

The DSP module is divided into the idle domains described in this section. To minimize power consumption, you can choose which domains are active and which domains are idle at any given time. The current state of all domains is collectively called the idle configuration.

12.3.2.1 Idle Domains

The DSP is divided into the idle domains described in Table 96. You can control which of these idle domains are active and which are idle at any given time, as described in section 12.3.2.2.

Table 96. Idle Domains in the DSP

Domain	Contents of the Domain	Configurability
CPU	DSP core and buses	When the IDLE instruction is executed, the DSP core remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP subsystem reset, it is active after a DSP subsystem reset.
DMA	DSP DMA controller and DMA buses	When the IDLE instruction is executed, the DMA controller remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP subsystem reset, it is active after a DSP subsystem reset.

Table 96. Idle Domains in the DSP (Continued)

Domain	Contents of the Domain	Configurability
CACHE	Instruction cache	When the IDLE instruction is executed, the instruction cache remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP subsystem reset, it is active after a DSP subsystem reset.
EMIF	External memory interface (EMIF)	When the IDLE instruction is executed, the EMIF is disabled or enabled, depending on the chosen idle configuration. Regardless of this domain's state before a DSP subsystem reset, it is active after a DSP subsystem reset.

12.3.2.2 Idle Configuration Process

The idle configuration indicates which idle domains will be idle and which idle domains will be active the next time the IDLE instruction is executed. The basic steps to the idle configuration process are:

1) Define a new idle configuration by writing to the bits in the idle configuration register (ICR). Make sure that you use a valid idle configuration (see section 12.3.2.3).

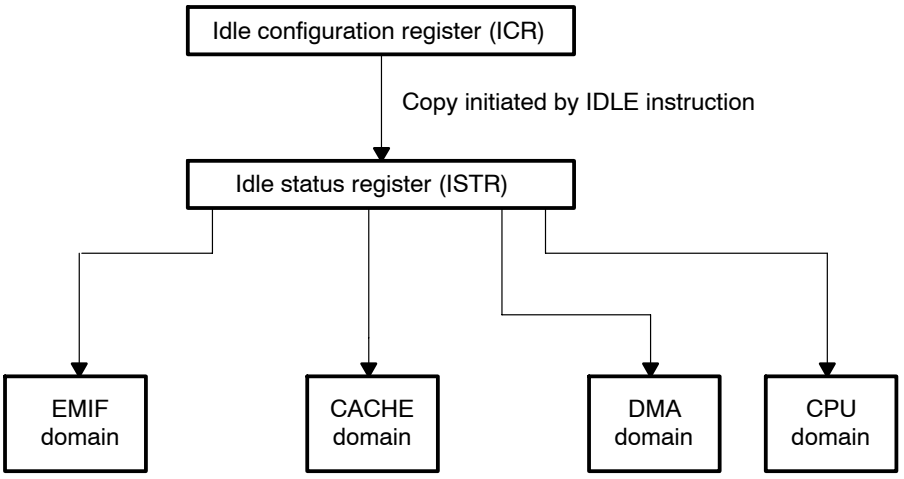
2) Apply the new idle configuration by executing the IDLE instruction. The effects are shown in Figure 105. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the system to enable or disable each of the chosen domains.

The IDLE instruction cannot be executed in parallel with another instruction.

Note:

If you intend to switch among multiple idle configurations, ensure that your system has the means to change from one idle configuration to the next. For important considerations, see section 12.3.2.4.

Figure 105. Idle Configuration Process



12.3.2.3 Valid Idle Configurations

Not all of the values that you can write to the idle configuration register (ICR) provide valid idle configurations. The valid configurations are limited by dependencies within the system. For example, the EMIF domain should not be idled when the DSP core is executing instructions from DSP external memory.

12.3.2.4 Key Conditions to Change Idle Configurations

Before you use the IDLE instruction, ensure that there is a method for the DSP to change the idle configuration afterward. Table 97 summarizes the methods available under two key conditions. For each condition there are two methods for changing the idle configuration, A and B. The effects of each method on the idle idle status register (ISTR) and the idle configuration register (ICR) are also detailed. For more details about these idle registers, see section 12.3.2.8.

Table 97. Changing Idle Configurations

Condition	Available Methods For Changing Idle Configuration	ISTR After Change	ICR After Change
1. CPU domain is active	A. Write a new configuration to the idle configuration register (ICR), and then execute the IDLE instruction.	A. Modified by the IDLE instruction; contains a copy of the new ICR value.	A. Contains the new value loaded by the program.
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s).	B. Cleared (all 0s).
2. CPU domain is idle	A. Use an unmasked hardware interrupt.	A. CPUIS bit is 0. No other bits were modified.	A. Not modified.
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s).	B. Cleared (all 0s).

Condition 1: CPU Domain Active

When the CPU domain is active (the DSP core is running), program flow continues. In this case, there are two methods of changing idle configurations:

- Write a new idle configuration to the idle configuration register (ICR), and then execute the IDLE instruction. The IDLE instruction copies the content of the ICR to the idle status register (ISTR), and the ISTR bit values are propagated to the idle domains. After the domains change states, the value in ISTR matches the value in ICR.
- Initiate a DSP subsystem reset. When the DSP subsystem resets, all domains are made active.

Condition 2: CPU Domain Idle

When the CPU domain is idle, program flow is halted. It is not possible to write a new value to the idle configuration register (ICR) or to execute the IDLE instruction. Two methods are available for changing the idle configuration:

- Use an unmasked interrupt. The interrupt clears the CPUIS bit of the idle status register (ISTR). The change to CPUIS reactivates the CPU domain. The content of the idle configuration register (ICR) is not modified. To learn how the DSP core responds to the interrupt, see section 12.3.2.5.
- Initiate a DSP subsystem reset. When the DSP subsystem resets, all domains are made active.

Once program flow has begun again, you can reactivate or deactivate other domains by writing a new idle configuration to ICR and then executing the IDLE instruction.

12.3.2.5 Interrupt Handling When the DSP Core Is Reactivated

If the DSP core has been halted by an idle configuration, it can be reactivated by a DSP subsystem reset or by a maskable interrupt that is enabled in an interrupt enable register (IER0 or IER1). A maskable interrupt request will also set the corresponding interrupt flag bit in an interrupt flag register (IFR0 or IFR1). Table 98 summarizes how the DSP core responds after being reactivated by maskable and nonmaskable interrupts. INTM is the global interrupt mask bit in status register ST1_55.

Table 98. DSP Core Response After Reactivation

Interrupt	Response After Reactivation
A maskable interrupt	If INTM = 0: The DSP core executes the interrupt service routine, executes the instruction that follows the IDLE instruction, and continues from there. The interrupt flag bit associated with the maskable interrupt will be cleared automatically when the DSP core branches to the interrupt service routine. If INTM = 1: The DSP core executes the instruction that follows the IDLE instruction and then continues from there. The interrupt service routine cannot be executed until interrupts have been globally enabled through INTM. The interrupt flag bit associated with the maskable interrupt will be set.
DSP subsystem reset	The DSP subsystem is reset. During a DSP subsystem reset, all idle domains are made active.

After the DSP core is brought out of the idle state, the flag bit that was set in the interrupt flag register (IFR0 or IFR1) has to be cleared before the DSP core can be placed back in the idle state.

12.3.2.6 Effect of a DSP Reset on the Idle Domains

During a DSP subsystem reset, all idle domains are made active.

12.3.2.7 DSP Module Idle Configuration Examples

To put one or more domains in idle mode, set the corresponding bits in the ICR register and then execute the DSP IDLE instruction. The idle status register reflects the state of the DSP when the IDLE instruction is executed.

Placing the DSP DMA in Idle

To set the DSP DMA domain to idle, follow these steps:

1) Set the DMA domain bit in the idle control register (ICR) by writing 0x0002 to ICR.

2) Use the DSP core to execute the IDLE instruction. The contents of ICR are copied to the idle status register (ISTR). This places the DMA domain into its idle state.

To wake up the DMA domain, follow these steps:

1) Clear the DMA domain bit in ICR by writing 0x0000 to ICR.

2) Use the DSP core to execute the IDLE instruction.

Note:

When the DMA domain is idle, there is one case when it can be temporarily reactivated without a change in the idle configuration. If one of the multichannel buffered serial ports (McBSPs) needs the DMA controller for a data transfer, the DMA controller will leave its idle state to perform the data transfer and then enter its idle state again.

Placing the Entire DSP Module Domain in Idle

To idle all the domains in the DSP module, follow these steps:

1) Set all the idle domain bits in the idle control register (ICR) by writing 0x0027 to ICR.

2) Use the DSP core to execute the IDLE instruction. The contents of ICR are copied to the idle status register (ISTR). This places all the domains into their idle states.

To wake up all the domains from this idle configuration, follow these steps:

1) Interrupt the DSP core using a maskable interrupt. The CPU and CPUIS bits of ICR and ISTR, respectively, are automatically cleared. This takes the CPU domain out of the idle state.

2) Clear all the idle domain bits in ICR by writing 0x0000 to ICR.

3) Use the DSP core to execute the IDLE instruction. This takes the rest of the domains out of idle.

Note:

The EMIF domain is not taken out of its idle state when the CPU domain is woken up by a maskable interrupt. This means that if both the EMIF and CPU domains are idle, the DSP core must not branch to the interrupt service routine (ISR) immediately after waking up if the ISR is located in DSP external memory. The DSP core can be kept from branching to the ISR by keeping the INTM bit until the EMIF has been manually awakened (see section 12.3.2.5).

12.3.2.8 Idle Registers

Two registers provide the means to individually configure and monitor each of the idle domains: the idle configuration register (ICR) and the idle status register (ISTR). These registers are accessible to the DSP core only.

Table 99. Registers for DSP Module Idle Control

Name	Description	DSP I/O Address†
ICR	Idle control register. Use this register to specify which domain should be idled when the IDLE instruction is executed.	0x0001
ISTR	Idle status register. Use this register to check the status of the idle domains.	0x0002

† DSP I/O addresses apply to both OMAP5910 and OMAP5912.

ICR lets you configure how each idle domain will respond upon IDLE instruction execution. When you execute the IDLE instruction, the content of ICR is copied to ISTR. The ISTR values are then propagated to the idle domains.

Figure 106. Idle Control Register (ICR)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved		EMIFI	Reserved†		CACHEI	DMAI	CPUI
R-0		RW-0	RW-0		RW-0	RW-0	RW-0

† These bits must always be 0.

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 100. Idle Control Register (ICR) Field Descriptions

Bits	Field	Value	Description
15–6	Reserved	–	These bits are read-only and return 0s when read.
5	EMIFI		EMIF-domain idle configuration bit. EMIFI determines whether the external memory interface (EMIF) will be idle after the next execution of the IDLE instruction:
		0	EMIF will be active.
		1	EMIF will be idle.
4-3	Reserved	–	Must always be kept as 0.
2	CACHEI		CACHE-domain idle configuration bit. CACHEI determines whether the cache will be idle after the next execution of the IDLE instruction:
		0	Cache will be active.
		1	Cache will be idle.
1	DMAI		DMA-domain idle configuration bit. DMAI determines whether the DMA controller will be idle after the next execution of the IDLE instruction:
		0	DMA controller will be active.
		1	DMA controller will be idle.
0	CPUI		CPU-domain idle configuration bit. CPUI determines whether the DSP core will be idle after the next execution of the IDLE instruction:
		0	DSP core will be active.
		1	DSP core will be idle.

Figure 107. Idle Status Register (ISTR)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved		EMIFIS	Reserved		CACHEIS	DMAIS	CPUIS
R-0		R-0	R-0		R-0	R-0	R-0

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 101. Idle Status Register (ISTR) Field Descriptions

Bits	Field	Value	Description
15–6	Reserved	–	These bits are not available for your use. They are read-only bits and return 0s when read.
5	EMIFIS		EMIF-domain idle status bit. EMIFIS is a copy of EMIFI made during the execution of the IDLE instruction. EMIFIS reflects the current idle status of the external memory interface (EMIF):
		0	EMIF is active.
		1	EMIF is idle.
4-3	Reserved	–	These bits are not available for your use. They are read-only bits and return 0s when read.
2	CACHEIS		CACHE-domain idle status bit. CACHEIS is a copy of CACHEI made during the execution of the IDLE instruction. CACHEIS reflects the current idle status of the cache:
		0	Cache is active.
		1	Cache is idle.
1	DMAIS		DMA-domain idle status bit. DMAIS is a copy of DMAI made during the execution of the IDLE instruction. DMAIS reflects the current idle status of the DMA controller:
		0	DMA controller is active.
		1	DMA controller is idle.
0	CPUIS		CPU-domain idle status bit. CPUIS is a copy of CPUI made during the execution of the IDLE instruction. CPUIS reflects the current idle status of the DSP core:
		0	DSP core is active.
		1	DSP core is idle.

12.4 DSP Bootloader

12.4.1 Introduction

This section provides a description of the features of the on-chip DSP bootloader provided with the OMAP5910 and OMAP5912 devices.

12.4.1.1 Bootloader Features

OMAP devices contain program code residing in the DSP subsystem PDROM, called a bootloader. The bootloader is executed by the DSP core when it is taken out of reset. Depending on the boot mode selected, the DSP core will branch to an internal or DSP external memory address or go into idle. The following is a list of the boot modes supported by the bootloader and a summary of their functional operation:

- DSP idle mode. The DSP subsystem is brought out of reset with all of the modules within the DSP module placed in their idle mode.
- Direct execution from DSP external memory. The on-chip PDROM is disabled and the interrupt vector table is fetched directly from DSP external memory.
- Branch directly to a starting address. The bootloader directs code execution to an entry point address which is specified by the selected boot mode. The entry point may be located in either internal or DSP external memory. Note that the memory at the destination address must be initialized with valid code before the bootloader is executed. With an internal memory entry point, the MPU core or system DMA can initialize the memory through the MPUI port.

The bootloader also offers the following features:

- Register controlled boot mode selection. The MPU core can select the boot mode by writing to a bootloader configuration register (described in section 12.4.2.2).
- Multiple entry points. Each boot mode uses a different entry point address.

12.4.1.2 DSP Subsystem On-Chip PDROM

The DSP subsystem PDROM contains the bootloader program. The PDROM may be enabled or disabled based on the settings of the BOOT_MODE bits in the DSP_BOOT_CONFIG register (section 12.4.2.2). When the PDROM is disabled, the PDROM memory space is mapped to memory that is external to the DSP subsystem. Table 102 lists the contents of the DSP PDROM.

Table 102. DSP PDROM Contents

Starting Byte Address	Contents
0xFF_8000	DSP Bootloader
0xFF_8200	Reserved
0xFF_FF00	Interrupt Vector Table

12.4.2 Bootloader Operation

The following sections describe the structure and operation of the bootloader.

12.4.2.1 Bootloader Initialization

When the bootloader begins execution, it performs some initialization of DSP resources prior to loading code. Table 103 describes the DSP resources that are configured by the bootloader.

Table 103. Bootloader Initialization

Resource	Initialization Value
Stack registers	The data stack register (SP) is initialized with address 0000A0h, and the system stack register (SSP) is initialized with address 0000C0h.
Stack configuration	The stack configuration is set to the dual 16-bit stack mode with fast return.
Interrupts	The INTM bit of Status Register 1 (ST1_55) is set to the default value of 1, to disable interrupts.
Sign extension	The SXMD bit of Status Register 1 (ST1_55) is cleared, to disable the sign extension mode.
Compatibility mode	The C54CM bit of Status Register 1 (ST1_55) is cleared, to disable the C54x compatibility mode.

To avoid corruption of default stack locations, the sections loaded into internal memory should not be between word addresses 0000h and 0100h (byte address 0000h-0200h).

After the initialization is performed, the bootloader branches to the starting point address specified through the selected boot mode. At that point, the boot load process is complete. Whenever the DSP subsystem is reset, the DSP core starts the execution of the bootloader again, and the entire boot process is repeated.

12.4.2.2 Boot Mode Selection

The bootloader mode (or boot mode) used by the bootloader is specified by the MPU core using the DSP Boot Configuration Register (DSP_BOOT_CONFIG). This register is read-only for the DSP and is mapped to address 0x000F in the DSP I/O space (within the DSP TIPB address space). The MPU can read and write to this register at address 0xFFFE C918 (within the MPUI address space). The MPU controls the boot process by programming the BOOT_MODE bits while the DSP subsystem is held in its reset state. Figure 108 and Table 104 show the details of the DSP boot configuration register.

Figure 108. DSP Boot Configuration Register (DSP_BOOT_CONFIG)

DSP Side

16	4	3	0
Reserved		BOOT_MODE	
R-0		R-0	

MPU Side

31	4	3	0
Reserved		BOOT_MODE	
R-0		RW-0	

Note: R = Read; W = Write; –n = Value after reset; –x = Value after reset is not defined.

Table 104. DSP Boot Configuration Register (DSP_BOOT_CONFIG) Field Descriptions

Bits	Field	Value	Description
31–4	Reserved		These bits are not used.
3–0	BOOT_MODE		DSP subsystem bootloader mode bits. These bits specify the boot mode used by the DSP bootloader.

12.4.3 Boot Modes

There are five boot modes for the DSP bootloader. The MPU core can select any of these boot modes by writing to the DSP_BOOT_CONFIG register.

When the DSP subsystem is released from reset (boot), the DSP core always fetches the interrupt vector table starting at byte address 0xFF FF00. The physical location of this address depends on the boot mode selected through the BOOT_MOD bits. If BOOT_MOD bits are equal to 0000b, the on-chip PDROM is disabled and the boot address is located off-chip. If BOOT_MOD is not equal to 0000b, the on-chip PDROM is enabled and the boot address is located on the on-chip PDROM.

Note:

The DSP MMU will determine the actual physical location of the PDROM memory space when the on-chip PDROM is disabled.

Table 105 lists the supported boot modes.

Table 105. Registers for DSP Module Idle Control

BOOT_MODE[3:0]	Boot Process	Description
0000	Direct boot	The on-chip PDROM is disabled and the PDROM memory space is mapped to DSP external memory. The DSP core fetches the interrupt vector table from 0xFF FF00 (located in DSP external memory).
0001	External memory boot	The bootloader simply branches to byte address 0x08 0000 in DSP external memory. Note that the MPU core may need to set up the DSP MMU such that the DSP core executes from valid DSP external memory.
0010	DSP idle boot	The bootloader places the DSP into idle mode.
0011	Reserved	Do not use this option.
0100	Reserved	Do not use this option.
0101	Internal memory boot	The bootloader branches to internal byte address 0x01 0000.
Other	Internal memory boot	The bootloader branches to internal byte address 0x02 4000.

The following sections describe each boot mode in detail.

12.4.3.1 Direct Boot Mode

When BOOT_MODE[3:0] = 0000b, the direct boot mode option is selected. In this mode, the bootloader program does not execute because the on-chip PDROM is not mapped into the internal memory map. The PDROM memory map addresses are treated as DSP external memory addresses, and thus are handled by the DSP EMIF and DSP MMU. When this boot option is selected, the DSP core branches to the interrupt vector table in DSP external memory space. For more information on the OMAP memory map, see section 3.4.

12.4.3.2 External Memory Boot Mode

When BOOT_MODE[3:0] = 0001b, the External Memory Boot mode is selected. In this mode, the bootloader does execute. After initializing the resources described in section 12.4.2.1, the bootloader simply branches to byte address 0x08 0000 in DSP external memory.

Note:

The MPU core may need to set up the DSP MMU such that the DSP core executes from valid DSP external memory.

12.4.3.3 DSP Idle Boot Mode

When BOOT_MODE[3:0] = 0010b at reset, the DSP Idle Boot mode is selected. In this mode, the bootloader will place the DSP subsystem into its idle state using the following sequence:

- Disable the DSP watchdog timer.
- Set the DSP RAM and peripherals to shared-access mode (SAM) with the HOM_R and HOM_P bits of ST3_55.
- Set all the DSP domains to idle through the Idle Control Register (ICR).
- Set the DSP RAM and peripherals to host-only mode (HOM) with the HOM_R and HOM_P bits of ST3_55.
- Execute the IDLE instruction.

A hardware or software reset can wake up the DSP subsystem from its idle state.

12.4.3.4 Internal Memory Boot Mode

In this mode, code and data sections must be initialized before the bootloader is executed. Internal memory can be directly initialized by the MPU core or system DMA via the MPUI port while the DSP core is in reset. When the DSP subsystem is released from reset, the bootloader will branch to the byte address specified by the selected boot mode.

The general procedure for an internal memory boot is:

- The code and data sections are loaded into DSP subsystem internal memory by the MPU core or system DMA via the MPUI port.
- The MPU core takes the DSP subsystem out of reset with an internal memory boot mode selected by the BOOT_MODE[3:0] bits.
- The bootloader executes and transfers execution to the appropriate byte address, and the loaded application begins running.

If the application has been previously loaded and another external reset is necessary (warm boot), the MPU core can reset the DSP subsystem without reloading the application code, and the application execution will begin.

12.4.4 Bootloader Sequence

The next two sections describe the entire bootloader sequence for all of the boot mode options.

12.4.4.1 Direct Boot Mode

In the Direct Boot Mode (BOOT_MOD[3:0] = 0000b) the on-chip PDROM is disabled and all PDROM memory map addresses are located in DSP external memory. When the MPU core releases the DSP subsystem from reset, the DSP core fetches the interrupt vector table starting at byte address 0xFF FF00. The physical location of the interrupt vector table in OMAP system memory is dependent on the use of the DSP MMU.

12.4.4.2 Other Valid Boot Modes

When BOOT_MOD contains a value other than 0000b, the on-chip PDROM is enabled and all PDROM memory map addresses are located in internal memory. The DSP core fetches the interrupt vector table starting at byte address 0xFF FF00 after the DSP subsystem is released from reset. The interrupt vector table then directs program execution to the DSP bootloader starting at byte address 0xFF 8000. At this point, the bootloader starts to execute.

The bootloader checks to see if the MPUI RAM is in shared-access mode (SAM) by checking the HOM_R bit of ST3_55. If not, the bootloader keeps checking indefinitely; it does not request an access mode change for the MPUI RAM. As soon as the MPUI RAM is in SAM, it does the following initialization:

- Sets up the stack pointers. The data stack register (SP) is initialized to address 00 00A0h, and the system stack register (SSP) is initialized to address 00 00C0h.

  The stack configuration is set to the dual 16-bit stack mode with fast return.
- Disables interrupts globally. The INTM bit of ST1_55 is set.
- Disables sign extension. The SXMD bit of ST1_55 is cleared.
- Disables the C54x compatibility mode. The C54CM bit of ST1_55 is cleared.

After this is done, the bootloader checks the BOOT_MOD[3:0] bits of the DSP_BOOT_CONFIG register and takes action as described in section 12.4.3.

Revision History

Table 106 lists the changes made since the previous version of this document.

Table 106. Document Revision History

Page	Additions/Modifications/Deletions
28	Updated the byte address for the external memory space to 0x02 8000 in section 3.2.
29	Deleted reserved row, and changed the byte address for the external memory space to 0x02 8000–0xFF 7FFF, and the word address to 0x01 4000–0x7F BFFF.
30	Updated the byte address for the external memory space to 0x02 8000 in section 4.1.
83	Updated the byte address for the external memory space to 0x02 8000 in section 6.2.5**.**