Official Release

AndeStar™ System Privilege Architecture Version 3 Manual

Document Number UM072-18

Date Issued 2018-07-02

ANDES TECHNOLOGY

Copyright Notice

The product described herein is subject to continuous development and improvement. Thus, all information herein is provided by Andes in good faith but without warranties. This document is intended only to assist the reader in the use of the product. Andes Technology Corporation shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

Contact Information

Should you have any problems with the information contained herein, you may contact Andes Technology Corporation through

- email – support@andestech.com
- Website – https://es.andestech.com/eservice/

Please include the following information in your inquiries:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem

General suggestions for improvements are welcome.

Revision History

Rev.	Revision Date	Revised Content
1.8	2018/7/2	1. Added MISC_CTL.HW_PREFETCH_EN bit. (section 10.10.2) 2. Clarified shared/non-shared attributes in MPU. (section 3.2.1)
1.7	2018/5/15	1. Added 8MB and 16MB configurations for local memory. (section 7.3, 10.4.7, 10.4.8)
1.6	2017/06/29	1. Added count events in the performance counters for ULM bank stall cycles (Sec. 10.7.3). 2. Added specification for the stack underflow detection feature (Sec. 4.5, 5, 5.1, 5.3, 5.4, 5.5, 5.6.2, 5.6.3, 5.6.4.1, 5.6.4.2, 10.3.7, 10.6.2, 10.6.3, 10.6.4, 10.6.5, 10.11). 3. Clarified the description for IVB.IVIC_VER (Sec. 10.3.4). 4. Adding a control bit (INT_MASK.IMPE) to control whether an imprecise exception should be blocked by PSW.GIE (Sec. 10.3.15). 5. Reformatted and updated this document to use the External Technical Document Template version 11.
1.5	2015/11/11	1. Added required system registers to support the parity/ECC feature. 2. Added Misc. Configuration Register 2 (cr7 or MSC_CFG2). 3. Added the definition and resources for the Interruption Architecture version 1 (MSC_CFG.INTV is 1) to improve processing of imprecise and multiple next-precise exceptions. 4. Clarified the description for Shadowed Privileged Stack Pointer (SP_PRIV or ir17). 5. Clarified the description for counting events of performance counters. 6. Clarified the requirement of optional system registers with explicit checking rules. 7. Changed the name “local memory error” to “local memory base setup error” to avoid confusion. 8. Clarified that machine errors will not affect the cacheability attribute sent out to the bus. 9. Clarified the description for the imprecise exception. 10. Updated the description for DMA_ISADDR and DMA_ESADDR. 11. Added an entry for “load/store instruction global stop” at ITYPE.ETYPE for the debug exception. 12. Clarified that Don’t Care (DC) values must not contain “reserved values”.

Rev.	Revision Date	Revised Content
1.4	2015/01/20	1. Added specification for the Hardware Stack Protection and Recording feature. 2. Added performance monitoring specification for V3m (MCU family). 3. Removed obsolete performance monitoring selection entry JAL9.JAT and J5.JAT. 4. Updated the bullet for PC to Machine Error exception entry point when entering the maximum Interrupt Stack Level. 5. Updated the listing of CPU IDs. 6. Added Coprocessor ISA extension version field CPV at cr6 (FUCOP_EXIST). 7. Added ITYPE.SUB_TYPE encoding for coprocessor imprecise exceptions and Andes Custom Extension (ACE) exceptions. 8. Clarified the description of ITYPE.SWID and removed its support for BREAK/BREAK16 instructions. 9. Added description for the optional feature to boot from ILM. Also changed the reset values of ILMB.IEN and ILMB.IBPA to “IM.” 10. Added configuration bit (CPU_VER.CFGID[6]) for Andes Custom Extension (ACE). 11. Clarified the description for CPU_VER.CFGID and CPU_VER.CPUID. 12. Clarified the description for CORE_ID (cr5). 13. Changed the implementation requirement of the Machine Error Log Register (ir8) from “optional” to “required.” 14. Clarified usages for Local Memory DMA trigger registers. 15. Clarified that any active DMA transfer will “freeze” (be suspended) if the CPU enters the debug mode. 16. Added extension for the total number of Misc. Configuration Registers (MSC_CFG.MSC_EXT). 17. Added configuration bit (MSC_CFG.SPP) for HW Stack protection/recording feature. 18. Removed obsolete descriptions for N1213 hardcore. 19. Added system registers needed to support the performance throttling (PowerBrake) feature. 20. Clarified the interface difference in lower 16 bits of INT_PEND and INT_PEND2. 21. Clarified the default settings and behavior for the debug mode (when PSW.DEX==1) and how to overwrite them with

Rev.	Revision Date	Revised Content
		EDM_CTL.DEX_USE_PSW. 22. Clarified the event selections available for performance counters are implementation-dependent and specified in the datasheet. 23. Clarified the usage of "software interrupt" bit (INT_PEND.SWI). 24. Added simple mnemonic name (racr0) for Privileged Resource User Access Control Register. 25. Corrected the updating rule for PSW.DEX in nested interrupt cases. 26. Clarified that P_P0 and P_P1 are not updated during interruption stack level transition from 0 to 0. 27. For "local memory base setup error" exception, clarified the write instruction causing the exception is canceled (forced to NOP) by HW to avoid further potential errors. 28. Clarified the vector entry point stores the beginning instructions of the interruption handlers. 29. Clarified the behavior of the interruption stack from level the maximum to the maximum. 30. Further clarified the meaning of ITYPE.INST. 31. Clarified that a "MSYNC" instruction must be added before switching the DLM between double-buffer and non-double-buffer mode. 32. Clarified the exact conditions for superuser mode. 33. Clarified the updating rule for "P_*" interruption stack registers. 34. Added control bits for ACE and Instruction Local Memory Cache at the Miscellaneous Control Register. 35. Changed the ITYPE.SWID from RO to RW (becomes writable). 36. Clarified that PSW.HSS is not modified by the debug exception. 37. Further clarified that the existence of Virtual Linear Page Table Index Register (mr5 or VLPT_IDX) is implementation-dependent. 38. Clarified that the EVA is not updated for imprecise exceptions due to lack of sufficient information. 39. Reminded that the PSW.IME and PSW.DME must be cleared by the exception handler to exit the machine error state and enable the cache. 40. Clarified the update rules for PSW.IME and PSW.DME. 41. Clarified that the ITYPE.ETYPE is set to 3 for all interrupts and 8 for SYSCALLs. 42. Removed an unsupported system register (EDM_PROBE).

Rev.	Revision Date	Revised Content
		43. Updated the implementation requirement to “required” for the ir18 (INT_PRI) and ir19 (INT_CTRL). 44. Updated the reset values of ITYPE.ETYPE, ITYPE.INST, ITYPE.VECTOR, and MISC_CTL.ILMC_EN. 45. Clarified the description of idr0 (SDZ_CTL). 46. Clarified the update rule of the ir11 (OIPC) for the configuration where the maximum interruption stack level is 2. 47. Removed an obsolete system register mr10 (HSMP_EADDR). 48. Simplified the description of the INT_PRI and INT_PRI2. 49. Clarified that the host debugger should not provide an unaligned returned address (OIPC); otherwise, unpredictable behavior may occur.
1.3	2014/01/13	1. Updated description for the local DMA to architecture 2.0. 2. Added new DMA system registers DMA_RCNT and DMA_HSTATUS. 3. Changed L1_PTE to Non-Leaf_PTE and L2_PTE to Leaf_PTE.
1.2	2013/11/27	1. Updated the definition for value 4 to 6 in PSW.CPL. 2. Corrected minor typos in the starting address of local memory base register for double buffer mode. 3. Clarified that either “isb” or “dsb” instruction must be added after “mtsr” if users want the written value to take effect immediately or read back this latest written value. 4. Removed obsolete section “TLB Implementation Constraint.” 5. Clarified instructions cannot be placed in the Data Local Memory (DLM). 6. Further clarified the usage of ITYPE.SUPRS_EXC and ITYPE.IMP_EXC flags. 7. Specified that ILMB.IEN and DLMB.IEN may become fixed to “0” for error prevention if the size of external local memory is set to zero or illegal values. 8. Clarified the description for “interruption handling in hardware.” 9. Updated the entry points for IVIC mode to 41 (9 exception + 32 interrupt). 10. Removed obsolete chapter for Optional Feature List. 11. Clarified that PSW.IT and PSW.DT become RAZWI if neither MMU nor MPU is supported. 12. Extended the maximal local memory size from 1MB to 4MB.

Rev.	Revision Date	Revised Content
		13. Fixed inconsistency in the bit position of TLB_DATA.PSB and redrew the diagram of TLB entry for clarity. 14. Fixed and clarified the sample code for TLB operations. 15. Clarified that the contents of the local memory never enter caches, and vice versa. 16. Clarified that the "Coprocessor Disabled Exception" is often used to save power. It detects the existence of coprocessor instructions and the exception handler only enables the coprocessor as needed. 17. Further clarified VA[23] or VA[23:22] are used for NTC and NTM in reduced 24-bit address space configuration (MSC_CFG.ADR24 is 1). 18. Fixed EDM_CTL to be "always exists" at the table in the Summary of System Register Existence. 19. Added description for Unified Local Memory (ULM) configuration. 20. Defined a next-precise exception: local memory base setup error. 21. Added new definition for ILM and DLM access in the performance counters.
1.1	2013/03/13	1. To support 32 interrupts (IVIC mode), added Interrupt Pending Register 2 (ir27) and Interrupt Priority Register 2 (ir28). 2. To support 32 interrupts, expanded the field width of IVB.NIVIC and added more configurations. 3. To support edge-triggered interrupts, added interrupt Trigger Type Register (ir29). 4. Described the hardware behavior for servicing edge-triggered interrupts. 5. Reserved ir20 to ir25 for AndeStar Security Extension at the summary tables for Interruption System Registers and System Register Existence. 6. Fixed the description of the default value for Interrupt Priority Register. 7. Updated the implementation requirement for Interrupt Priority Register (ir18) and Interrupt Control Register (ir19). 8. When address range overlaps, clarified that local memory is always picked first while other overlapping memories are masked out (ignored) completely. 9. Added definition for control bits MISC_CTL.SP_SHADOW_EN and MISC_CTL.LP_CACHE. 10. Further clarified the need for 8-byte stack pointer adjustment and

Rev.	Revision Date	Revised Content
		description of IPSW.SP_ADJ. 11. Clarified the description for MSC_CFG.DIV and MSC_CFG.MAC. 12. Corrected the existence criteria for High Speed Memory Port address (HSMP_SADDR and HSMP_EADDR). 13. Clarified description in the table for Summary of System Register Existence. 14. Corrected the total number of reserved bits for DMA Status Register. 15. Further clarified the definition of SUPRS_EXC and IMP_EXC flags at ITYPE. Clarified that they are set only for host-mode debugging. 16. Added a global definition for all implementation-dependent registers and removed definitions for individual cores. 17. Removed the description for obsolete configurations (ICM_CFG.BSAV==0) and (DCM_CFG.BSAV==0) for clarity. 18. Clarified the description for DCM_CFG.DLMB.
1.0	2012/11/12	1. Removed obsolete legacy_mode bit and replaced with V2_INT_MODE pin. 2. Specified that the default behavior at the maximum interruption level can be modified with EDM_CTL.DEX_USE_PSW when debugger is connected. 3. Clarified that IVBASE only defines the high 16 bits of the interruption vector table. 4. Clarified that Performance Monitoring Registers become USR when PFM_USR_EN is set to 1. 5. Clarified the description for MSC_CFG.HSMP. 6. To allow SW backup, changed all the fields in ITYPE from RO to RW. 7. Specified the stack pointer is cleared by cold reset but not by warm reset. 8. Explicitly listed all the operations performed by warm reset. 9. Clarified that warm reset does not clear the stack pointer adjust it to be 8-byte aligned. A warm reset also forces the CPU to wake up from standby mode immediately. 10. Clarified the triggering conditions for MERR.BUSERR are implementation dependent which are specified in the datasheet. 11. Clarified the description on DMA_ESADDR and DMA_SETUP.ESTR.

Rev.	Revision Date	Revised Content
		12. Added the definition for ITYPE.SUB_TYPE field. 13. Added out-of-range exception check in the 24-bit reduced address space configuration. 14. Changed "Nonexistent local memory address" exception to "Nonexistent memory address" to include 24-bit address out-of-range exception. 15. Directed to a common entry point for interruptions at the maximum interruption stack level. 16. Extended Vector ID to include both exceptions and interrupts for V3 (V3m only include exceptions). 17. Specified that Shadowed Privileged Stack Pointer (ir17) has a reset value of 0. 18. Specified EVA will record the misaligned target address (different from IPC) for branch target alignment exception. 19. Changed Exception Virtual Address Register (EVA or ir4) to be writable in order to record more information on suppressed exceptions. 20. To simplify context switch, moved the overflow flag (OV) for saturation arithmetic from USTAT to PSW. 21. In the basic counting events at Table 29, added selections for PUSH25, POP25, SYSCALL, JAL9.JAT and J5.JAT. 22. For performance monitoring, clarified that push25 and pop25 are not counted in the "load/store completed" category but are counted in the total "completed instructions". 23. For performance monitoring, clarified SYSCALL is included in the "completed instructions" category. Also, JRNEZ and JRALNEZ are included in the "conditional branch" category. 24. To help debugging, added a configuration bit (ICM_CFG.EXT) to indicate whether the instruction local memory is internal or external. 25. In CPU Version Register (CPU_VER), added a configuration bit to indicate whether Saturation Arithmetic Extension is supported or not. 26. Removed referral to specific chip IDs in the "summary of system register existence". 27. Changed the implementation requirement for SP_USR and SP_PRIV to "Shadow Register Optional (MSC_CFG.SHADOW == 1)". 28. Added MSC_CFG.EIT to indicate whether EX9.IT extension is

Rev.	Revision Date	Revised Content
		supported. 29. Specified that EVA is not updated for warm resets, debug related exceptions, and suppressed imprecise exceptions (i.e., imprecise bus error) during debug mode. 30. Specified accesses to shadow stack pointer through MTSR/MFSR must add DSB: “MTSR + DSB” and “modifying R31 + DSB + MFSR”. 31. Added exact equation for shadow stack pointer selection. 32. Clarified if a module does not exist, the associated events are set to 0 and the performance counters will keep their existing values unchanged. 33. Specified that debug exceptions are decoupled from the interruption stack and do not increase interruption stack level. 34. Described the behavior for suppress exception (special section for debug exception) and flags to record them. 35. Specified that debugging related information is no longer in ITYPE.ETYPE but moved to DETYPE and DIPC. 36. Added control for exception suppression with EDM_CTL.DEH_SEL. Only suppress exceptions in host-mode debugging but allow new exceptions in target-mode debugging. 37. Clarified the updating rule of system registers for debug exceptions. 38. Clarified that 8-byte stack pointer alignment will not be performed at the maximum interruption stack level for lack of IPSW as flag storage. 39. Clarified that 8-byte stack pointer alignment will not be performed for debug exceptions to avoid modifying interruption registers.

Table of Contents

List of Tables

List of Figures

Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal text	Georgia	Normal	12	Black
`Command line, source code or file paths`	`Lucida Console`	`Normal`	`11`	`Indigo`
`VARIABLES OR PARAMETERS IN COMMAND LINE, SOURCE CODE OR FILE PATHS`	**`LUCIDA CONSOLE`**	**`BOLD + ALL-CAPS`**	**`11`**	**`INDIGO`**
Note or warning	Georgia	Normal	12	Red
Hyperlink	**Georgia**	**Underlined**	**12**	**Blue**

1. Introduction

This document describes the privileged system architecture resources and features the AndesCore™ will provide to the system software for flexible and efficient management of user processes and system processes themselves.

It covers the following topics:

- Memory Management Unit (Chapter 2)
- Memory Protection Unit (Chapter 3)
- Interruption Architecture (Chapter 4)
- Hardware Stack Protection and Recording (Chapter 5)
- Performance Monitoring support (Chapter 6)
- Local Memory support (Chapter 7)
- Local Memory DMA support (Chapter 8)
- High Speed Memory Port support (Chapter 9)

All system registers controlling the behavior of above mentioned mechanisms are summarized in the System Register Definitions (Chapter 10) of this document.

1.1. Processor Operation Mode

A program running on an AndesCore can be operated under two different Processor Operation Modes (POM): the user mode, and the superuser mode. A program running in the user mode is called a user-mode program. A program running in the superuser mode is called a superuser-mode program.

A superuser-mode program has a higher privilege than a user-mode program. A user-mode program can only access user-mode resources, but cannot access superuser-mode privileged resources. A superuser-mode program can access both user-mode resources and superuser-mode resources.

User-mode resources include General Purpose Registers (GPRs), user special registers and user instructions. Superuser-mode resources include system registers and privileged instructions.

The Processor Operation Mode is located in the Processor Status Word (PSW) privileged system register. The PSW register controls the current states and statuses a running user-mode program or a running superuser-mode program can have.

User mode to Superuser mode transition can happen on one of the following conditions:

- When an exception happens.
- When an interrupt happens.
- Execution of a “SYSCALL” instruction.
- Execution of “TRAP”, “TEQZ”, and “TNEZ” instructions.

Superuser mode to User mode transition can happen on one of the following conditions:

- Update POM to user mode using MTSR (Move To System Register) instruction.
- Update POM to user mode from Interruption PSW (IPSW) register using an IRET (Interruption Return) instruction.

A CPU is in the superuser mode under the following conditions:

- When PSW.DEX==0 (non-debug exception):
  - PSW.POM==1, or
  - Maximum stack level is reached
- When PSW.DEX==1 (debug exception):
  - EDM_CTL.DEX_USE_PSW==0, or
  - EDM_CTL.DEX_USE_PSW==1 and PSW.POM==1

2. Memory Management Unit with Non-Fully-Associative TLB

This chapter describes memory management unit and TLB operations and contains the following sections:

2.1. Introduction

Andes Memory Management Unit (MMU) is responsible for managing physical memory resources using virtual page to physical page address translations. It provides address space protection among different processes (running programs) and provides memory page attributes such as cacheability and coherency for system performance enhancement.

The Andes MMU's main component is a Translation Look-aside Buffer (TLB) structure. It can be completely managed by software or a Hardware-assisted Page Table Walker (HPTWK) can be enabled to improve the TLB fill performance and/or the TLB capacity.

The exact organization of the TLB structure is implementation-dependent. A specific TLB implementation will be provided in Appendix A for reference. However, from the software's point of view, the Andes TLB structure consists of two parts: a software-visible part and a software-invisible part. The software-visible part of the TLB is a non-fully-associative cache memory with N sets and K ways, thus N*K=M entries. All the M entries can be managed by software using targeted TLB read/write, probe, and flush operations. Another TLB random write operation is also provided to insert a page translation into a hardware-determined entry to relieve software from the responsibility of managing TLB indexes. The software-invisible part of the TLB is implementation-dependent. And once it exists, software needs to manage it using probe and flush operations, but not the TLB read/write operations.

The relationship between the software-visible and software-invisible parts of the TLB is implementation-dependent. The software-invisible part of the TLB could contain a subset of translation information of the software-visible part of the TLB. Or the software-invisible part of the TLB could contain a completely different set of translation information from the software-visible part of the TLB.

The relationship between the HPTWK and the two parts of the TLB structure is implementation-dependent as well. The HPTWK can be implemented to insert translation information into only the software-invisible part of the TLB or it can be implemented to insert the information into the software-visible part of the TLB. The HPTWK can be enabled by setting a bit in the MMU L1_PPTB register. In order to use the HPTWK, the operating system has to prepare a physical page table (PPT) in a fixed format in order for the HPTWK to find the VA to PA translation (i.e., Page Table Entry) from the page table. In the current architecture, the page table format supports only a single small page size and a large page size which is an integer multiple of the small page size.

2.2. Address Spaces in Andes MMU

There are three addressing spaces in an Andes memory system: the virtual address (VA), the physical address (PA), and the system address, which correspond to the three major operating modes for the machine: user mode, superuser mode and hypervisor mode, respectively.

Since this document describes an MMU architecture without the hypervisor support, the physical address space is equal to the system address space. And the physical address space is the address that shows up on the system bus throughout this document.

The characteristics of the virtual and physical address spaces are as follows:

1. The virtual address space is the address space used by a process when fetching instructions (program counter value) and accessing memory using load/store instructions. The virtual address is 32-bit wide, and thus the virtual address space is a 4GB space. The user and superuser processes will both generate virtual addresses.
2. The physical address space is the address space managed/controlled by superuser mode processes. Any superuser mode process manages the physical address space by controlling the VA to PA translation policies for user mode processes and sometimes for itself. It can also disable the translation process for superuser and user mode processes respectively. If the translation process is disabled, the virtual address is equivalent to the physical address.
3. Typically, the physical address space is the universe the operating system operates. The superuser mode is a protected privilege mode designed for the operating system software to multiplex resources amongst applications. The operating system is responsible for managing the translation of virtual addresses into physical addresses.

2.2.1. Address Translation Process

A virtual address must be translated into a physical address before the data could be accessed if the translation process for user mode or superuser mode is enabled.

The virtual address translation starts when the pipeline sends a virtual address to the TLB

structure. If a translation is found in the TLB structure, a physical address is generated from the translation information and used to access memory. If the lookup fails, the virtual address is subsequently processed by the HPTWK or by the TLB fill exception handler.

Figure 1 illustrates the complete address translation process.

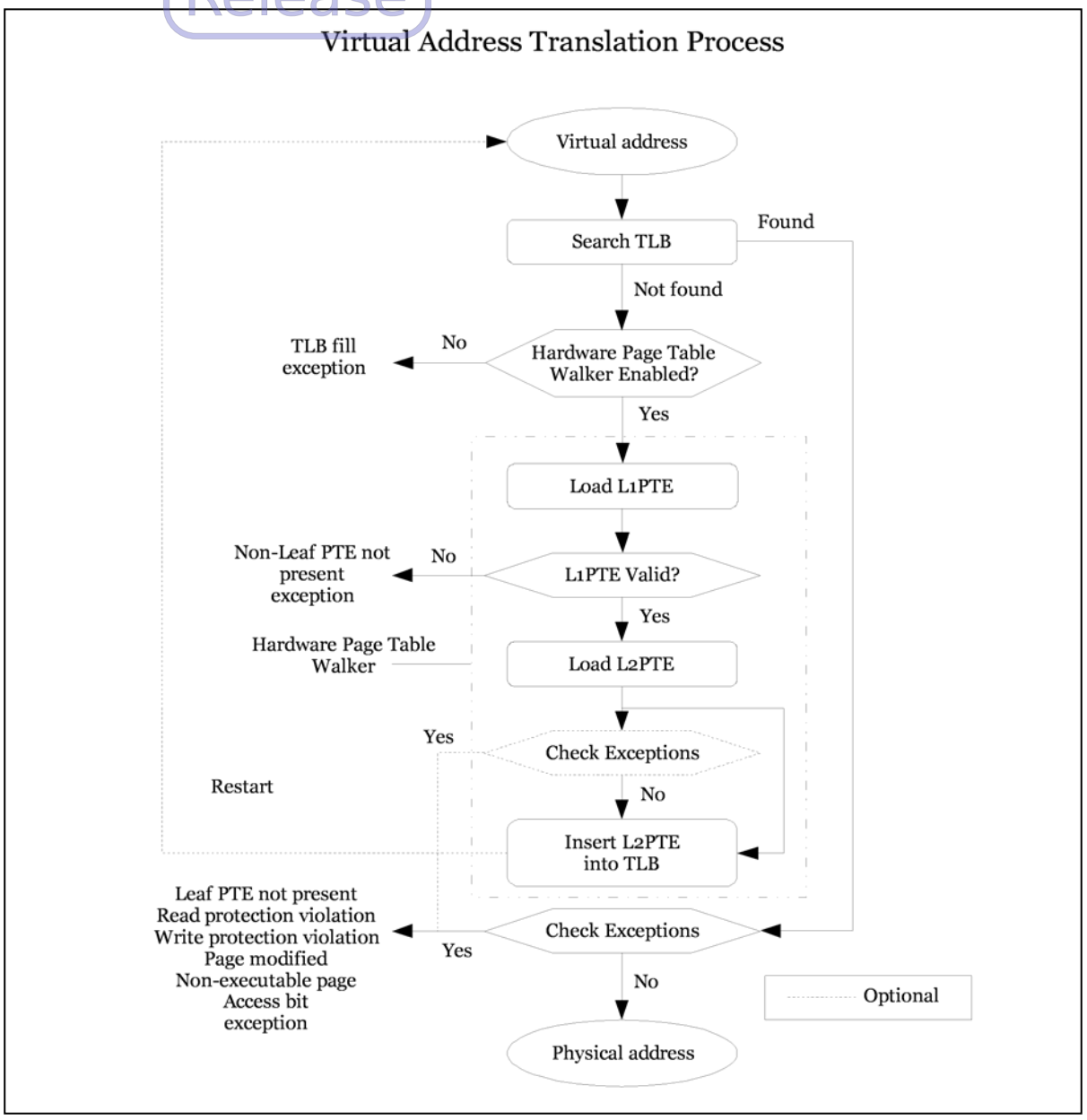


Figure 1. Complete virtual address translation process

2.2.2. Address Space Attributes

Attributes are defined for virtual/physical address spaces in order to achieve: system performance enhancement, access control, and sharing of program or data. These attributes are defined differently to reflect various use cases.

2.2.2.1 Attributes for Translated Virtual/Physical Address

To correctly translate a virtual address into a physical address, attributes are defined in the page table entry for each page. Since the translation is page-based, fine grained access control are added to each page. The page attributes are listed in the following tables along with their bit numbers in the 32-bit page table entry. Table 1 lists the attribute definitions for Andes MMU version 1 and Table 2 lists the attribute definitions for Andes MMU version 2. The differences between version 1 and version 2 are in the definitions of the X and the M[3:1] attribute fields. Version 2 allows earlier permission violation detection on the user code executing non-accessible superuser page than version 1.

Table 1. Translated address space attributes for MMU version 1

Field Name	Bits	Description ("-" means Reserved)
PPN	31:12	Physical Page Number of the physical memory page.
LP	11	Large Page hint for Hardware Page Table Walker (HPTWK). Setting this bit allows the HPTWK to fill a large page PTE into the TLB from searching a small page Page Table structure. The size of the large page is implementation-dependent based on the large page support of the TLB structure and the HPTWK's size choice for the large page insertion function.
C	10:8	Cacheability: 3'd0: device space 3'd1: device space, write bufferable/coalescable 3'd2: non-cacheable memory 3'd3: - (Reserved) 3'd4: cacheable, write-back, write-allocate memory (shared)

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description (“-“ means Reserved)</th></tr>
<tr><td></td><td></td><td>3’d5: cacheable, write-through, no-write-allocate memory (shared)
3’d6: cacheable, non-shared, write-back, write allocate memory
3’d7: cacheable, non-shared, write-through, no-write-allocate memory
Using of Reserved values (3) will generate Reserved PTE Attribute exception (Instruction or Data).
Note: For dual/multi cores sharing a second-level cache, an implementation may choose to cache “shared” data only in L2 while caching “non-shared” data in both L1/L2. Please see implementation documents for details.</td></tr>
<tr><td>G</td><td>7</td><td>Global bit. Setting this bit indicates that this translation is shared across contexts.</td></tr>
<tr><td>A</td><td>6</td><td>Accessed bit. If this bit is set, any access to this page will trigger an Access Bit exception.</td></tr>
<tr><td>X</td><td>5</td><td>Executable bit. It indicates whether this page can be fetched for execution. If this bit is not set, fetching from this page will generate an exception.</td></tr>
<tr><td>D</td><td>4</td><td>Dirty bit. If this bit is not set, any store to this page will generate a Page Modified Exception.</td></tr>
<tr><td>M</td><td>3:1</td><td>Read/Write Access mode:
<table>
<tr><th>M[3:1]</th><th>User mode</th><th>Superuser mode</th></tr>
<tr><td>3’b000</td><td>-</td><td>-</td></tr>
<tr><td>3’b001</td><td>Read only</td><td>Read only</td></tr>
<tr><td>3’b010</td><td>Read only</td><td>Read/Write</td></tr>
<tr><td>3’b011</td><td>Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b100</td><td>-</td><td>-</td></tr>
<tr><td>3’b101</td><td>No Read/Write access</td><td>Read only</td></tr>
<tr><td>3’b110</td><td>-</td><td>-</td></tr>
<tr><td>3’b111</td><td>No Read/Write access</td><td>Read/Write</td></tr>
</table>
Note that this access mode only governs the access permission of using load/store instructions to access the page. It does not govern the access permission of an instruction fetching and execution.</td></tr>
</table>

Field Name	Bits	Description ("-" means Reserved)
		Using of Reserved values (0, 4, 6) will generate a Reserved PTE Attribute exception (Data).
V	0	Valid bit. Setting this bit indicates that this Page Table Entry is valid and present.

Table 2. Translated address space attributes for MMU version 2

Field Name	Bits	Description ("-" means Reserved)
PPN	31:12	Physical Page Number of the physical memory page.
LP	11	Large Page hint for Hardware Page Table Walker (HPTWK). Setting of this bit allows the HPTWK to fill a large page PTE into the TLB from searching a small page Page Table structure. The size of the large page is implementation-dependent based on the large page support of the TLB structure and the HPTWK's size choice for the large page insertion function.
C	10:8	Cacheability: 3'd0: device space 3'd1: device space, write bufferable/coalescable 3'd2: non-cacheable memory 3'd3: - (Reserved) 3'd4: cacheable, write-back, write-allocate memory (shared) 3'd5: cacheable, write-through, no-write-allocate memory (shared) 3'd6: cacheable, non-shared, write-back, write allocate memory 3'd7: cacheable, non-shared, write-through, no-write-allocate memory Using of Reserved values (3) will generate Reserved PTE Attribute exception (Instruction or Data). Note: For dual/multi cores sharing a second-level cache, an implementation may choose to cache "shared" data only in L2 while caching "non-shared" data in both L1/L2. Please see implementation documents for details.
G	7	Global bit. Setting this bit indicates that this translation is shared across contexts.
A	6	Accessed bit. If this bit is set, any access to this page will trigger an Access Bit Exception.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description (“-“ means Reserved)</th></tr>
<tr><td>X</td><td>5</td><td>Executable bit.
■ In superuser mode, a non-executable page exception will be generated when fetching a page without this bit set.
■ In user mode and when M[3:1] != 3’b101 and M[3:1] != 3’b111, a non-executable page exception will be generated when fetching a page without this bit set.
■ In user mode and when M[3:1] == 3’b101 or M[3:1] == 3’b111, a non-executable page exception will always be generated. The X bit is don’t-care in this case.</td></tr>
<tr><td>D</td><td>4</td><td>Dirty bit. If this bit is not set, any store to this page will generate a Page Modified Exception.</td></tr>
<tr><td>M</td><td>3:1</td><td>Read/Write Access mode + user mode fetching restriction on superuser mode page.
<table>
<tr><th>M[3:1]</th><th>User mode</th><th>Superuser mode</th></tr>
<tr><td>3’b000 X==0</td><td>-</td><td>-</td></tr>
<tr><td>3’b000 X==1</td><td>No Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b001</td><td>Read only</td><td>Read only</td></tr>
<tr><td>3’b010</td><td>Read only</td><td>Read/Write</td></tr>
<tr><td>3’b011</td><td>Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b100</td><td>-</td><td>-</td></tr>
<tr><td>3’b101</td><td>No Read/Write/Fetch access</td><td>Read only</td></tr>
<tr><td>3’b110</td><td>-</td><td>-</td></tr>
<tr><td>3’b111</td><td>No Read/Write/Fetch access</td><td>Read/Write</td></tr>
</table>
Note that this access mode governs the access permission of using load/store instructions to access the page. And it governs the user mode instruction fetching/execution permission when M[3:1] == 3’b101 or M[3:1] == 3’b111.
Using of Reserved values 0 (when X is 0), 4, and 6 will generate a Reserved PTE Attribute exception for load/store instruction (Data) and instruction fetching (Instruction).</td></tr>
<tr><td>V</td><td>0</td><td>Valid bit. Setting this bit indicates that this Page Table Entry is valid and present.</td></tr>
</table>

2.2.2.2 Attributes for Non-translated Virtual/Physical Address

For virtual to physical address mapping that does not go through the page table translation process, the address space attributes are defined in either two or four fields of the non-translated attribute field of the MMU Control Register. The non-translated attribute field is indexed using either the top or the top two bits of the virtual address (i.e., VA(31) or VA(31,30)). Since this is a very coarse partition of the system address space (2 or 4 segments), only one cacheability attribute is defined for each segment. The cacheability attribute defined here is lumped into four different types (as shown in the following table) instead of seven types in the translated address space attributes.

Table 3. Non-translated address space attribute

Non-translated Cacheability (NTC)	Meaning
0	Non-cacheable/non-coalesable
1	Non-cacheable/coalesable
2	Cacheable/write-back
3	Cacheable/write-through

To improve memory access performance in a small system without MMU or MPU, another partition-level VA to PA mapping attribute (NTM) is defined such that different cacheability attributes can be mapped to the same PA partition through different VA partitions without frequently changing the content of the NTC field. This will improve the software efficiency of small systems.

An example usage of NTM partition mapping is illustrated in the following figure (Figure 2). In this example, VA partitions 0-2 are all mapped to the same PA partition 0 where the real memory is located. In this small system, a programmer can use the following C code sequences to define the cacheability attribute for each variable. Note that changing a variable from cacheable to non-cacheable requires a cache invalidation operation to guarantee correct behaviors.

```
#define CacheWT  (1 << 30)
#define NonCache (1 << 31)

int a, b, c, d;
int *p, *q, *r, *s;
p = &a | CacheWT; // make “a” cacheable/WT
q = &b | NonCache; // make “b” non-cacheable
```

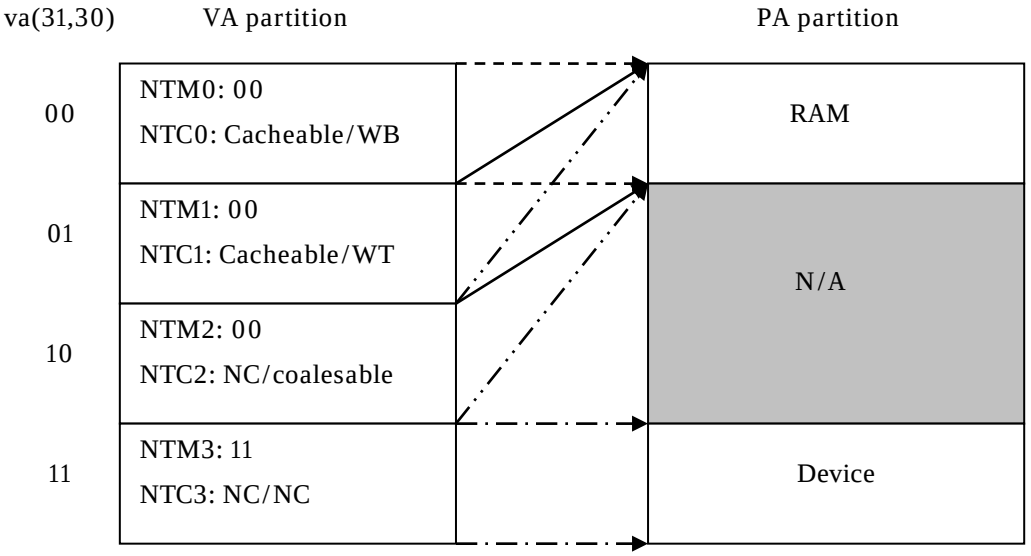



Figure 2. Example usage of NTM partition mapping

2.2.3. Address Space Operation Ordering Requirement

In Andes memory architecture, loads and stores for different addresses generated by an AndesCore™ are, in most part, not ordered except for regions characterized by certain device-related address space attributes. For un-ordered loads and stores, if there is a need to enforce certain ordering between these memory operations, MSYNC instruction must be used to achieve this goal.

The following table lists the address space operation ordering requirement based on the translated address space attributes. The symbols used in this table are defined as:

- < : This is a “BEFORE” relation between two operations in time or program sequence. For example, “A < B” in a program means instruction A comes earlier than instruction B in a program sequence.

- u : This is an "UN-ORDERED" relation between two operations in time of performing their operations.

Table 4. Address space operation ordering requirement

A < B in program order		B									
		Device		Bufferable device		Uncacheable memory		WT cacheable memory		WB cacheable memory	
A		read	write	read	write	read	write	read	write	read	write
Device	read	<	<	<	<	u	u	u	u	u	u
	write	<	<	<	<	u	u	u	u	u	u
Bufferable device	read	<	<	<	<	u	u	u	u	u	u
	write	<	<	<	u	u	u	u	u	u	u
Uncacheable memory	read	u	u	u	u	u	u	u	u	u	u
	write	u	u	u	u	u	u	u	u	u	u
WT cacheable memory	read	u	u	u	u	u	u	u	u	u	u
	write	u	u	u	u	u	u	u	u	u	u
WB cacheable memory	read	u	u	u	u	u	u	u	u	u	u
	write	u	u	u	u	u	u	u	u	u	u

For non-translated memory attributes, they can be mapped into the ordering requirement of the translated memory attributes using the following mapping:

Non-translated Attribute	Mapped Translated Attribute
Non-cacheable/non-coalesable	Device
Non-cacheable/coalesable	Bufferable Device
Cacheable/write-back	WB Cacheable Memory
Cacheable/write-through	WT Cacheable Memory

2.2.4. Reduced Address Space for Small Systems

To reduce cost for a small system, a reduced 24-bit address space configuration is provided by some AndesCore implementations. The existence of this configuration is recorded in the ADR24 field of the MSC_CFG system register.

In this reduced 24-bit address space configuration, the non-translated attribute fields (NTC and NTM) are indexed using either the top or the top two bits of the 24-bit virtual address (i.e., VA[23] or VA[23:22]).

The following memory address related registers will be reduced to 24-bit in this configuration. When moving the content of these registers into a 32-bit register, the top 8-bit (bit 31 to bit 24) will be filled with 0. When moving the content of a 32-bit register into these registers, only the lower 24-bit value (bit 23 to bit 0) will be moved into these registers.

Register Name	Note
Program Counter (PC)	
Interruption	
IVB	
EVA	
IPC	
OIPC	
P_EVA	
P_IPC	
Memory Management	
L1_PPTB	May not exist for ADR24
ILMB	
DLMB	
HSMP_SADDR	May not exist for ADR24

Register Name	Note
Local Memory DMA	
DMA_ESADDR	May not exist for ADR24
Embedded Debug Module	
BPAn	
BPAMn	
DIMBR	

Note that, regardless of 24-bit or 32-bit address space, all load/store (including multiple load/store) base register updates are still 32-bit operations.

For “branch and link” or “jump and link” instructions, the sequential return address calculated (PC+4 or PC+2) may be a 24-bit or 32-bit operation, depending on the implementation. In other words, if the calculated return address is greater than 0x00FFFFFF (the end of 24-bit address space), the top 8-bit of the content of the result register may be a value of zero or a value of one.

In V3 architecture (MSC_CFG.BASEV is 2), an out-of-range check is implemented to make sure an address does not go beyond the range of 24 bits. If a violation is detected, it generates a “Nonexistent Memory Address (+I/D bit)” exception which belongs to the general exception with ETYPE equal to 9. This feature improves the debugging of software.

2.3. Large Page and Multiple Page Size Support

The Andes MMU TLB structure provides multiple page size support in the software-visible part of the TLB. It may support ten different page sizes: 4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, and 256MB. The exact number of page size supported is implementation-dependent. And the HPTWK may support only one or two predetermined page sizes to reduce hardware cost in the HPTWK itself and the software-invisible part of the TLB structure. For page sizes not supported by the HPTWK, software is required to insert the page translation results into the Andes TLB structure.

2.4. TLB Locking Support

It is sometimes desirable to lock some particular entries inside the TLB to avoid triggering page faults that cause recursions, race conditions, or simply for performance concerns. The Andes MMU TLB structure provides locking support in the software-visible part of the TLB.

Recall that the software-visible part of the TLB is an N-set and K-way set-associative (or non-fully-associative) cache memory. To provide TLB locking support, a TLB "fill with lock" instruction will be provided to lock the TLB entry from TLB random replacement. However, since the TLB is not a fully-associative cache (i.e., only K-way set-associative), software has to make sure at most only K-1 ways can be locked in each set to allow at least one entry in a set to be used as the replacement entry for a TLB random insertion or HPTWK TLB insertion operation. If a violation occurs such that all K ways of a set are locked and a TLB insertion is performed on the set, then either a "Machine Error" exception or a hardware random replacement overwriting a locked entry will occur (depending on the setting in the TLBALCK field of the MMU Control Register). Please see section 10.4.1 MMU Control Register for more detail on the TLBACLK field.

2.5. MMU Bootstrapping

There are two working modes for MMU: "no translation", and "VA translation" modes. After a system reset, the memory management system enters into the "no translation" mode and the processor boots up in superuser mode. The operating system enables the "VA translation" mode after it initializes the page table to map itself and later transfers the control to a user mode application.

2.6. MMU Related Registers

Table 5 lists all the MMU related registers. All these registers require superuser mode privilege to access. Please see Section 10.4 MMU System Registers for their detailed definitions.

Table 5. MMU related system registers

Name	Type	Updater	Section
MMU control	RW	SW	10.4.1
L1 PPT Base	RW	SW	10.4.2
TLB Access Misc	RW	SW	10.4.5
TLB Access VPN	RW	SW/HW	10.4.3
TLB Access Data	RW	SW	10.4.4
Virtual Linear PT Index	RW	SW/HW	10.4.6
ILM (Instruction Local Memory) Base	RW	SW	10.4.7
DLM (Data Local Memory) Base	RW	SW	10.4.8
Cache Control	RW	SW	10.4.9
High Speed Memory Port Starting Address	RW	SW	10.4.10

The “Reserved” field in a register is reserved for future extension. It must be written with 0. The processor behavior is “UNPREDICTABLE” if 0 is not used to write the “Reserved” field. The reserved field is read as 0.

The “Reserved” value in a register field is reserved for future extension. If a reserved value is written into a register field, the processor behavior will be “UNPREDICTABLE”.

The “Ignored” field in a register is ignored by the processor when it is written. And it will be read as 0.

Implementation Note:

By defining that non-zero writes to "Reserved" field will cause "UNPREDICTABLE" behavior, it eliminates the need for hardware checking circuit against incoming write values because software is now responsible for not creating "UNPREDICTABLE" behavior. The only potential downside is that, if software indeed makes an error by writing a non-zero value, it will not be detected by hardware and may cause an error when this "Reserved" field is re-defined as a "meaningful" field in later implementations.

2.7. TLB Management Instructions

Nine types of TLB management instructions are defined to manage the Andes TLB structure. They are summarized in the following table and described in detail in the following sections.

Table 6. MMU management instructions

Instruction	Operation
TLBOP Rx, TargetRead (TRD)	Read targeted TLB entry
TLBOP Rx, TargetWrite (TWR)	Write targeted TLB entry
TLBOP Rx, RWrite (RWR)	Write PTE into a TLB entry
TLBOP Rx, RWriteLock (RWLK)	Write PTE into a TLB entry and lock
TLBOP Rx, Unlock (UNLK)	Unlock a TLB entry
TLBOP Rt, Rx, Probe (PB)	Probe TLB entry
TLBOP FlushAll (FLUA)	Flush all TLB entries except locked entries
TLBOP Rx, Invalidate (INV)	Invalidate TLB entries containing VA stored in Rx (except locked entries)
(Optional) LW_VLPT	Load VLPT page table which always goes through data TLB translation. On TLB miss, generate Double TLB miss exception

2.7.1. TLB Target Read

The TLB Target Read operation reads a specified entry in the software-visible portion of the TLB structure. The specified entry is in the Rx register. The read result is placed in the TLB _VPN, TLB _DATA, and TLB _MISC registers.

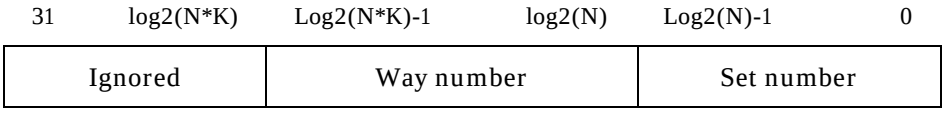
The TLB Target Read operation is performed using the following instruction:

"TLBOP Rx, TargetRead" or "TLBOP Rx, TRD"

The normal instruction sequence of performing the TLB Target Read operation is as follows:

```
movi   Rx, TLB_rd_entry   // prepare read entry number
tlbop  Rx, TRD            // read TLB
dsb                       // data serialization barrier
mfsr   Ry, TLB_VPN        // move read result to a general purpose reg.
```

The "TLB_rd_entry" in the code represents the entry number of a TLB entry you want to read out. The TLB entry number for the non-fully-associative N-set and K-way TLB cache is as follows:



2.7.2. TLB Target Write

The TLB Target Write operation writes a specified entry in the software-visible portion of the TLB structure. The specified entry is in the Rx register. The other write operands are in the TLB_VPN, TLB_DATA, and TLB_MISC registers.

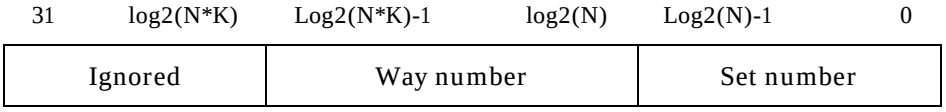
The TLB indexed write operation is performed using the following instruction:

"TLBOP Rx, TargetWrite" or "TLBOP Rx, TWR"

The normal instruction sequence of performing the TLB Target Write operation is as follows:

```
mtsr   Ra, TLB_VPN        // prepare VPN (may not be needed if HW
                          // already set up the correct value)
mtsr   Rb, TLB_DATA       // prepare PPN, etc.
mtsr   Rc, TLB_MISC       // prepare CID… (may not be needed if the
                          // correct value is already there)
dsb                       // data serialization barrier
movi   Rx, TLB_wr_entry   // prepare write entry
tlbop  Rx, TWR            // TLB target write
isb                       // inst. serialization barrier
```

The TLB entry number for the non-fully-associative N-set and K-way TLB cache is as follows:



If the selected target entry is locked, this instruction will overwrite the locked entry and clear the lock flag.

2.7.3. TLB Random Write (Hardware-determined Way in a Set)

The TLB_Random_Write operation writes a hardware-determined random TLB way in a set determined by the VA (in TLB_VPN) and page size (in TLB_MISC) in the software-visible portion of the TLB structure. The Rx operand contains the data to be written into the TLB_DATA portion of the TLB structure. The other write operands are in the TLB_VPN and TLB_MISC registers.

The TLB_Random_Write operation is performed using the following instruction:

"TLBOP Rx, RWrite" or "TLBOP Rx, RWR"

The normal instruction sequence of performing the TLB_random_insert operation is as follows:

```
mtsr   Rb, TLB_MISC   // prepare CID (may not be needed if the
                      // correct value is already there)
dsb                   // data serialization barrier
tlbop  Rx, RWR        // Rx contains PPN, etc.
isb                   // inst. serialization barrier
```

This instruction will generate an imprecise "Data Machine Error" exception if all the ways in the determined set are all locked.

2.7.4. TLB Random Write and Lock

TLB Random Write and Lock operation is similar to TLB Random Write Operation to write a PTE into a hardware determined TLB entry. In addition to the write operation, this instruction also locks the TLB entry.

The TLB Random Write and Lock operation is performed using the following instruction:

"TLBOP Rx, RWriteLock" or "TLBOP Rx, RWLK"

This instruction will generate an imprecise "Data Machine Error" exception if all the ways in the selected set are all locked before this instruction tries to lock an entry.

2.7.5. TLB Unlock

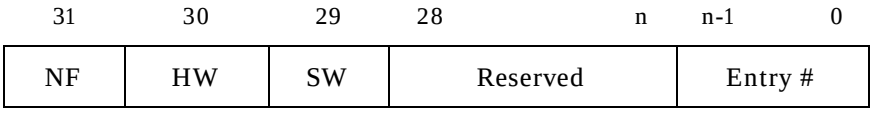
TLB Unlock operation unlocks a TLB entry if the VA in the Rx matches the VPN of an entry in a set determined by the VA (in Rx) and the page size (in TLB_MISC).

The TLB Unlock operation is performed using the following instruction:

"TLBOP Rx, Unlock" or "TLBOP Rx, UNLK"

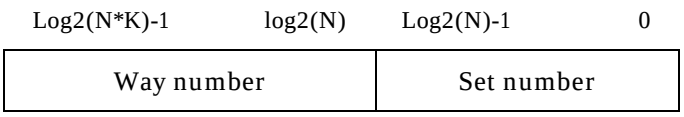
2.7.6. TLB Probe

The TLB_probe operation searches all TLB structures (software-visible and software-invisible) for a specified VA, and the search result is written into the general register Rt. The searched VA is specified in the Rx portion of the TLB probe instruction. The result stored in Rt has the following format:



It contains the entry number of the TLB entry (software-visible TLB) which contains the VA search key in the general source register of the TLB probe instruction. If the VA can be found in the software-visible part of the TLB, the "sw" bit will be set. If the VA can be found in the software-invisible part of the TLB, the "hw" bit will be set. And if the VA cannot be found in either the software-visible or software-invisible part of the TLB, the "nf" bit will be set.

The TLB entry number for the non-fully-associative N-set and K-way TLB cache is as follows:



<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>TLB entry number</td><td>n
(n-1,0)</td><td>Indicates the entry number of a PTE entry which contains the VA specified in the Rx of a TLB probe instruction. The TLB entry number for a non-fully-associative N-set and K-way TLB cache is as follows:
Log2(N*K)-1 log2(N) Log2(N)-1 0
<table><tr><td>Way number</td><td>Set number</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>Reserved</td><td>28-n+1
(28,n)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>SW</td><td>1
(29)</td><td>Search flag for the software-visible part of TLB.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The page containing VA is not found in the software-visible part of TLB.</td></tr><tr><td>1</td><td>The page containing VA is found in the software-visible part of TLB.</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>HW</td><td>1
(30)</td><td>Search flag for the software-invisible and non-subset part of TLB.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The page containing VA is not found in the software-invisible and non-subset part of TLB.</td></tr><tr><td>1</td><td>The page containing VA is found in the software-invisible and non-subset part of TLB.</td></tr></table></td><td>RO</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
NF	1 (31)	Search result. Value 0: The page containing VA is found in TLB. Value 1: The page containing VA is not found in TLB.	RO	0

The TLB_probe operation is performed using the following instruction:

"TLBOP Rt, Rx, Probe" or "TLBOP Rt, Rx, PB"

The normal instruction sequence of performing the TLB probe operation is as follows:

```
tlbop  Rt, Rx, PB            // VA is in Rx
<Examine> Rt
```

2.7.7. TLB Flush All

The TLB_flush_all operation flushes all TLB entries (software-visible and software-invisible) except the locked TLB entries.

The TLB_flush_all operation is performed by using the following instruction:

"TLBOP FlushAll" or "TLBOP FLUA"

The normal instruction sequence of doing the TLB_flush_all operation is as follows:

```
tlpop  FLUA        // move to system reg
isb               // Inst serialize barrier
```

2.7.8. TLB Invalidate VA

The TLB_Invalidate_VA operation flushes the TLB entry that contains the VA in the Rx register (software-visible and software-invisible) and the page size specified in the TLB_MISC register.

The TLB_Invalidate_VA operation is performed by using the following instruction:

"TLBOP Rx, Invalidate" or "TLBOP Rx, INV"

The normal instruction sequence of doing the TLB_invalidate_VA operation is as follows:

```
// prepare VA in Rx
…
tlbop  Rx, INV      // Invalidate TLB entries containing VA
isb                 // inst serialize barrier
```

Note that this TLB invalidate operation may flush more pages than the exact number of pages which contain this VA, up to flushing the entire TLB structure. And this is implementation-dependent.

2.7.9. Load VLPT Page Table (Optional)

The LW_VLPT instruction will be used to do a memory load from the virtual linear page table. Its difference from the normal load instruction is that (1) it is a load that will always use translation without regarding to the DT bit in the PSW register, and (2) on TLB miss, instead of a TLB fill exception, it will generate a Double TLB miss exception.

Note that if we can get to this instruction in the TLB fill exception handler, the HPTWK is definitely not enabled to do TLB fill operations.

***AndesCore™* Info**: N12 does not have this feature implemented.

2.8. Page Table Formats

The Andes hardware page table format is implementation-dependent. As examples, this section describes 4KB and 8KB page tables which are the most popular implementations.

2.8.1. 4KB Page Table

The Andes 4KB page hardware page tables are organized as two level hierarchical tables. The entries in the first level of the page tables are indexed by VA(31,22). The entries in the second level of the page tables are indexed by VA(21,12). The entries in the first-level tables are pointers that point to the second-level tables. Both tables have 2^10 = 1024 entries, each entry is 32-bit (4-byte) wide and therefore each table is 4KB in size.

2.8.2. 8KB Page Table

The Andes 8KB page hardware page tables are organized as two level hierarchical tables. The entries in the first level of the page tables are indexed by VA(31,24). The entries in the second level of the page tables are indexed by VA(23,13). The entries in the first-level tables are pointers that point to the second-level tables. The first-level table has 256 4-byte page table entries, and thus occupies 1/8 of an 8KB page. And the second-level table has 2048 4-byte page table entries, thus occupy an 8KB page.

2.8.3. Page Table Coherence Requirements

The Hardware Page Table Walker (HPTWK) is logically a separate DMA agent of the memory system, so it does not have access to or information on the internal states of the AndesCore such as the first-level cache. Consequently, to maintain consistent and coherent data between the HPTWK and the AndesCore, software needs to use extra cache management and memory access ordering instructions to enforce the coherence of the PTE data after any PTE update operation. Software has to follow Rule A and Rule B after any PTE update operation; otherwise the HPTWK is not guaranteed to get the updated page table content.

- **Rule A**
  1. If the page table has a “non-cacheable” memory attribute, then no cache management instruction is needed.
  2. If the page table has a “cacheable/write-through” memory attribute, then no cache management instruction is needed.
  3. If the page table has a “cacheable/write-back” memory attribute, then a “CCTL Ra, L1D_VA_WB” instruction is needed after a PTE update instruction (i.e., store) to write back the data from the first-level Dcache to memory.

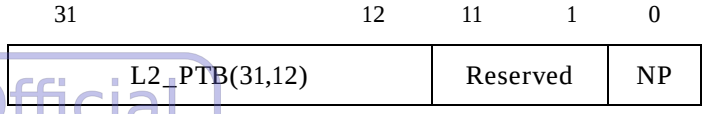
- **Rule B**
  1. Disregard of the cacheability attribute, a “MSYNC store” instruction is needed to force all PTE updates visible in memory before any load/store instruction which triggers HPTWK loading of the updated PTE. The “MSYNC” instruction has to be after any cache management instruction.

The following example code sequence illustrates the above requirements.

```
lw  R1, [R2]          // load PTE
...                   // PTE manupulation
sw  R1, [R2]          // store PTE
cctl R2, L1D_VA_WB    // assuming “cacheable/write-back” attribute
msync store           // drain L1D write buffer to memory
iret or isb           // serialize next fetch/HPTWK read
```

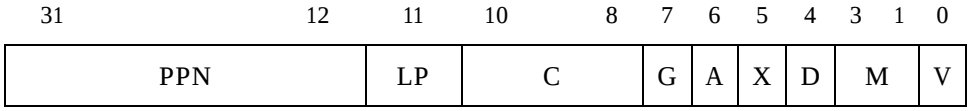
To reduce these extra overheads, software can consider preparing more PTEs than necessary at each PTE update occasion.

2.8.4. L1_PHYSICAL_PAGE_TABLE_ENTRY



L2_PTB is a 4KB-aligned physical address that points to the level-2 page table containing the page translation for the accessed VA. If the "NP" field is set, the level-2 page table does not exist and a Non-leaf_PTE_NOT_PRESENT exception will be signaled by hardware.

2.8.5. L2_PHYSICAL_PAGE_TABLE_ENTRY



PPN is the Physical Page Number the translated Virtual Page maps to. Once this entry is found by the HPTWK, it will insert this entry into the ANDES TLB structure, if no other exception is generated.

For detailed descriptions of the fields in this page table format, please see Table 1 in section 2.2.2.1 Attributes for Translated Virtual/Physical Address.

The "LP" field is added to indicate that this PTE is part of a large page whose size is determined by a processor core implementation. The Hardware Page Table Walker is required to use this bit to determine that if this PTE is for a large page size or not. If this PTE is for a large page size, then the HPTWK is required to insert the PTE into the TLB as a large page PTE entry. Note that this large page is built on the page table structure of a small page size. So an OS needs to prepare duplicated small page PTE entries to cover all the page table search space for the large page. Using a 4KB/1MB page size combination as an example, 256 entries of 4KB PTE needs to be prepared when the 1MB large page is being allocated.

2.8.6. PTE Modification and Insertion Requirement

When inserting a PTE into the TLB, the TLB update instructions (Random Write, Target Write) and the TLB update hardware (page table walker) do not search and invalidate the PTEs in the TLB that cover or overlap the address range of the new PTE. Software is required to remove (i.e., invalidate) all those PTEs that cover/overlap the address range of the new PTE from the TLB before inserting the new PTE.

2.8.7. Page Table Address Formation

To find the correct PTE from the page table in memory, the HPTWK needs to perform two memory load operations to get Non-leaf_PTE and then Leaf_PTE. The two physical load addresses for a 4KB page are formed by the HPTWK based on the following illustration in Figure 3.

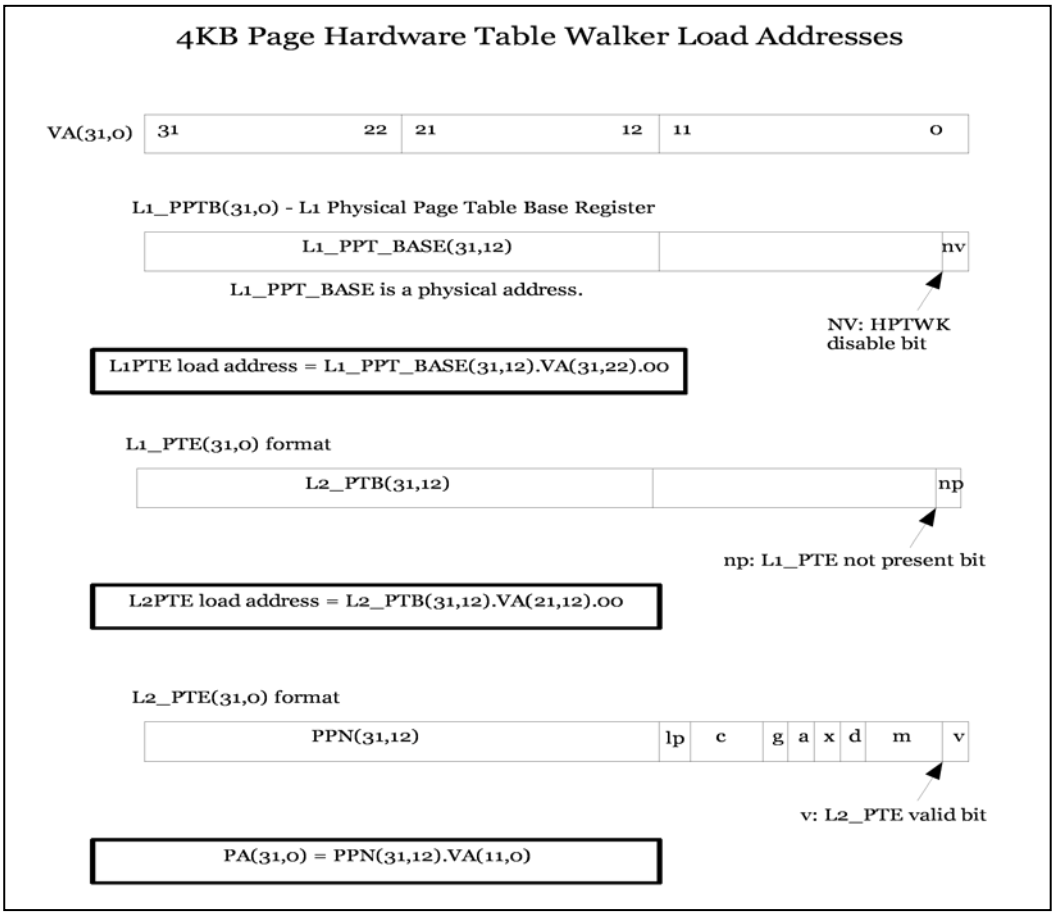


Figure 3. 4KB page Hardware Page Table Walker address formation

The two physical load addresses for an 8KB page are formed by the HPTWK based on the following illustration.

8KB page table format for HPTWK

VA(31,0)

31 24	23 13	12 0
8	11	13

L1PTE load address = L1_PPTB(31,12).00.VA(31,24).00

L2PTE load address = L2_PTB(31,13).VA(23,13).00

PA(31,0) = PPN(31,13).VA(12,0)

Figure 4. 8KB page Hardware Page Table Walker address formation

2.9. MMU Exception Handling

2.9.1. MMU Exceptions from VA to PA Translation

Nine possible exceptions could be generated during a VA to PA translation process. Their names and generating conditions are listed in the following table.

Table 7. MMU exceptions

Name	Generating condition	Entry vector
TLB fill	(No translation found in Andes TLB structure) and (HPTWK is not enabled)	TLB fill
Non-Leaf PTE not present	Non-Leaf_PTE(“np”) == 1	PTE not present
Leaf PTE not present	Leaf_PTE(“v”) == 0	PTE not present
Read protection violation	Based on Leaf_PTE(“m”) and current processor operating mode	TLB misc
Write protection violation	Based on Leaf_PTE(“m”) and current processor operating mode	TLB misc
Page modified	(Leaf_PTE(“D”) == 0) and this is a store access	TLB misc
Non-executable page	(Leaf_PTE(“X”) == 0) and this is an instruction fetch	TLB misc
Access bit	Leaf_PTE(“A”) == 1	TLB misc
Double TLB miss	TLB miss on LD_VLPT instruction	TLB VLPT miss

TLB fill exception has its own exception entry point to facilitate faster software TLB fill exception handling operations. PTE not present exceptions have their own exception entry point as well. All the other exceptions use the same MMU exception vector and the exact exception type is recorded at the Interruption Type Register (ITYPE).

On exceptions generated when a PTE is found and valid/present in the TLB, several of them could happen simultaneously. To resolve this situation, their handling priority is listed in the following table.

Table 8. Handling priority for simultaneous MMU exceptions

Name	Priority
Non-executable page, Read protection violation, Write protection violation	1
Page modified	2
Access bit	3

This priority list is defined as such based on the assumption that an Access bit exception is generated based on a legal memory access and has a less restrictive condition to generate than that of a Page modified exception which is under a legal access as well, while all the other access violation exceptions are all caused by illegal memory accesses. And the priority of reporting illegal accesses should be higher than that of reporting legal accesses.

2.9.2. Multiple Match on Software-visible Part of the TLB Sructure

Since the software-visible part of the Andes TLB structure is a set-associative cache memory, it is possible to generate multiple matches when searching for a VA translation due to software bugs or hardware errors (e.g., alpha particle hit, etc.). If multiple matches occur, it is not possible for the hardware to know which entry contains the correct information. Thus, a Machine Error exception will be generated to leave the handling of this situation to software.

The following operations could encounter this multiple match situation and generate a Machine Error exception:

- Instruction fetch – Instruction Machine Error exception.
- Load/Store type instructions – Data Machine Error exception.
- TLBOP Probe instruction – Imprecise Data Machine Error exception.

The following operation could encounter this multiple match situation but will not generate a Machine Error exception since it is very easy to resolve its problem in hardware.

- TLBOP Invalidate VA instruction.

2.9.3. All Entries Locked on a Set of the TLB Structure

The Andes MMU TLB structure provides locking support in the software-visible part of the TLB structure. However, since the TLB is not a fully-associative, but a K way set-associative cache, software has to make sure at most only K-1 ways can be locked in each set to allow at least one entry in a set to be used as the replacement entry for a TLB random insertion or HPTWK TLB insertion operation. If a violation occurs such that all K ways of a set are locked and a TLB insertion is performed on the set, then either a "Machine Error" exception or a hardware random replacement overwriting a locked entry will occur (depending on the setting in the TLBALCK field of the MMU Control Register). Please see section 10.4.1 MMU Control Register for more detail on the TLBACLK field. And see section 10.3.7 Interruption Type Register for the interruption type encoding of this exception in the Machine Error exception entry point.

2.10. Mixed Endian Support

Andes instruction set and memory architecture provide mechanisms to support a mixed data endian environment to speed up and facilitate the endian exchange process.

2.10.1. Basic Mixed Endian Support

The Processor Status Word (PSW) system register contains a BE bit to determine how load/store instructions move data between memory and registers: using little endian or big endian format. Software can change the endian format by executing a SETEND.B or SETEND.L instruction followed by a DSB instruction.

The MMU configuration (MMU_CFG) system register contains a DE (Default Endian) bit which will be copied into PSW.BE bit when an interruption has occurred. This makes sure that the interruption handler can have a consistent endian behavior no matter how user programs change the endian format before the interruption event.

2.10.2. Device Register Space Endian Control

To further enhance the flexibility of the OS and device driver pairing, an additional device register space endian control mechanism is provided on top of the basic Andes mixed endian support described in the previous section. This extra mechanism can provide an easy environment to operate an OS in one endian format while all device components in the opposite endian format.

The MMU control (MMU_CTL) system register contains a DREE (Device Register Endian Enable) bit that controls if this extra mechanism is turned on or not.

When this extra mechanism is turned on, a device register space access, i.e., memory page attribute C is equal to 0 or NTC in a partition is equal to 0, will use the DRBE bit contains in the Processor Status Word (PSW) system register to determine the access endian format while an access to all other memory attribute space will still use PSW.BE for the endian format. The PSW.DRBE bit can be updated using a privileged MTSR instruction followed by a DSB instruction.

The MMU configuration (MMU_CFG) system register contains a DRDE (Device Register Default Endian) bit that will be copied into PSW.DRBE bit when an interruption has occurred. This also makes sure that the interruption handler can have a consistent device register endian behavior no matter how kernel programs change the endian format before the interruption event.

3. Memory Protection Unit

This chapter describes a simple memory management unit (called Memory Protection Unit) and contains the following sections:

3.1. Introduction

This chapter describes a Memory Protection Unit scheme that uses less hardware and memory resources to manage memory compared to the Andes Memory Management Unit. This Memory Protection Unit scheme (called Andes MPU) provides memory remapping and bound checking on high boundaries. It re-uses most of the software interfaces defined by the Andes Memory Management Unit with some definition changes.

The Andes MPU contains an 8-entry TLB-like structure in hardware. The information inside the TLB structure is completely managed by software using TLBOP target write and target read instruction. The information in the MPU TLB includes the following items:

- Address remapping
- Address bound checking
- Cacheability attribute
- Accessibility attribute

It is similar to Andes MMU that when the PSW.IT is 1, the MPU TLB structure will be used for instruction fetch to generate memory remapping, various attributes, and protection information. When PSW.IT is 0, the memory remapping for instruction fetch is off and the NTC fields in the

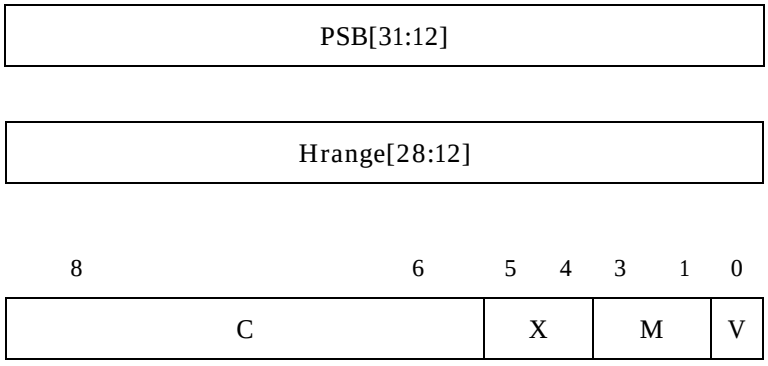
MMU control system register will be used to generate the cacheability attribute.
Similarly, when PSW.DT is 1, the MPU TLB structure will be used for load/store instruction to generate memory remapping, various attributes, and protection information. When PSW.DT is 0, the memory remapping for load/store instruction is off and the NTC fields in the MMU control system register will be used to generate cacheability attribute.

3.2. MPU TLB Structure

The MPU TLB structure is an eight-entry hardware table indexed by the top 3 bits of the virtual address [31:29]. An implementation may implement a merged MPU structure for both instruction fetching and data accesses or build separate MPU structures for instruction fetching and data accesses respectively. For separate instruction and data MPU structures, the MPU management instructions need to be able to manipulate each structure individually.

3.2.1. MPU Table Entry

Each MPU table entry has the following format:



It contains a 4KB-aligned Physical Section Base which re-maps the starting position of the 512MB-aligned Virtual Section to a physical memory. It also contains one 4KB-aligned upper virtual section limit for memory access range protection.

The detailed meanings of these fields are defined as follows:

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description (“-“ means Reserved)</th></tr>
<tr><td>PSB</td><td>20</td><td>Physical Section Base address of the physical memory section. The physical address is generated using the following formula:
PA(31,0) = {PSB(31,12) + VA(28,12)}.VA(11,0)</td></tr>
<tr><td>Hrange</td><td>17</td><td>High Range. The upper bound virtual address in the 512MB virtual address section. Read/Write/Execute protection exception will be generated if VA(28,12) >= Hrange(28,12) depending on load, store, or instruction fetch.</td></tr>
<tr><td>C</td><td>3
(8,6)</td><td>Cacheability:
<table>
<tr><th>Encoding</th><th>Meaning</th></tr>
<tr><td>0</td><td>Device space</td></tr>
<tr><td>1</td><td>Device space, write bufferable/coalescable</td></tr>
<tr><td>2</td><td>Non-cacheable memory</td></tr>
<tr><td>3</td><td>-</td></tr>
<tr><td>4</td><td>Cacheable, write-back, write-allocate memory (shared)</td></tr>
<tr><td>5</td><td>Cacheable, write-through, no-write-allocate memory (shared)</td></tr>
<tr><td>6</td><td>Cacheable, non-shared, write-back, write-allocate memory</td></tr>
<tr><td>7</td><td>Cacheable, non-shared, write-through, no-write-allocate memory</td></tr>
</table>
Using of Reserved values in this field will generate Reserved Attribute exception at the time of reading out the MPU TLB entry.
Note: For dual/multi cores sharing a second-level cache, an implementation may choose to cache “shared” data only in L2 while caching “non-shared” data in both L1/L2. Please see implementation documents for details.</td></tr>
<tr><td>X</td><td>2
(5,4)</td><td>Execution permission: exception when fetching a section without proper execution permission.
Bit-5: superuser mode execution is allowed when this bit is set.
Bit-4: user mode execution is allowed when when this bit is set.</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description (“-“ means Reserved)</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit-5 (S)</th><th>Bit-4 (U)</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>None executable page for all modes</td></tr>
<tr><td>0</td><td>1</td><td>User only code (Less useful)</td></tr>
<tr><td>1</td><td>0</td><td>Privileged code</td></tr>
<tr><td>1</td><td>1</td><td>User code</td></tr>
</table>
Non-executable code exception will be generated if an instruction fetch does not get execution permission based on the processor operating mode.
</td></tr>
<tr><td>M</td><td>3
(3,1)</td><td>
Read/Write Access mode:
<table>
<tr><th>M[3:1]</th><th>User mode</th><th>Superuser mode</th></tr>
<tr><td>3’b000</td><td>-</td><td>-</td></tr>
<tr><td>3’b001</td><td>Read only</td><td>Read only</td></tr>
<tr><td>3’b010</td><td>Read only</td><td>Read/Write</td></tr>
<tr><td>3’b011</td><td>Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b100</td><td>-</td><td>-</td></tr>
<tr><td>3’b101</td><td>No Read/Write access</td><td>Read only</td></tr>
<tr><td>3’b110</td><td>-</td><td>-</td></tr>
<tr><td>3’b111</td><td>No Read/Write access</td><td>Read/Write</td></tr>
</table>
Note that this Access mode only governs the access permission of using load/store instructions to access the page. It does not govern the access permission of an instruction fetching and execution. Using of Reserved values in this field will generate Reserved Attribute exception at the time of reading out the MPU TLB entry.
</td></tr>
<tr><td>V</td><td>1
(0)</td><td>This MPU Table Entry is valid and present.</td></tr>
</table>

3.3. MPU TLB Management Instructions

Two TLB management instructions are defined to manage the Andes MPU TLB structure. They are summarized in the following table and described in detail in the following sections.

Instruction	Operation
TLBOP Rx, TargetRead (TRD)	Read targeted MPU TLB entry
TLBOP Rx, TargetWrite (TWR)	Write targeted MPU TLB entry

3.3.1. TLB Target Read

The TLB target read operation reads a specific entry in the MPU TLB structure. The entry being read is specified in the Rx register. The read result is placed in the TLB_VPN and the TLB _DATA system registers. Note that these two system registers have different formats for the MPU and MMU schemes. Their MPU-defined formats are described in section 3.4.

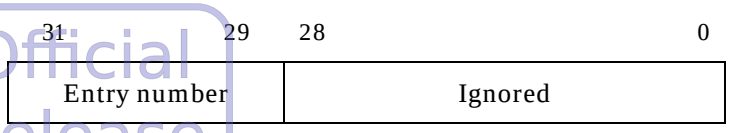
The TLB target read operation is performed using the following instruction:

"TLBOP Rx, TargetRead" or "TLBOP Rx, TRD"

The normal instruction sequence of performing the TLB Target Read operation is as follows:

```
movi   Rx, TLB_rd_entry<<29  // prepare read entry number
tlbop  Rx, TRD               // read TLB
dsb                          // data serialization barrier
mfsr   Ry, TLB_DATA          // move read out to reg
mfsr   Rz, TLB_VPN
```

The “TLB_rd_entry” in the code represents the entry of TLB you want to read out. The MPU TLB entry is defined as follows:



3.3.2. TLB Target Write

The TLB target write operation writes a specific entry in the MPU TLB structure. The entry being written is specified in the Rx register. A TLB target write gets its input operands from the TLB_VPN and TLB _DATA system registers. Note that these two system registers have different formats for the MPU and MMU schemes. Their MPU-scheme formats are described in section 3.4.

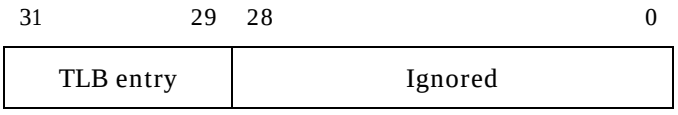
The TLB target write operation is performed using the following instruction:

“TLBOP Rx, TargetWrite” or “TLBOP Rx, TWR”

The normal instruction sequence of performing the TLB Target Write operation is as follows:

```
mtsr   Rb, TLB_DATA          // prepare PSB, etc.
mtsr   Rc, TLB_VPN           // prepare High range.
dsb                          // data serialization barrier
movi   Rx, TLB_wr_entry<<29  // prepare write entry number
tlbop  Rx, TWR               // TLB write
isb                          // inst. serialization barrier
```

The TLB entry is defined as follows:



3.4. MPU-related System Registers

3.4.1. MMU Control Register

Mnemonic Name: mr0 (MMU_CTL)

IM Requirement: required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 0, 0}

10 9	8 7	6 5	4 3	2 1	0
Reserved	NTC3	NTC2	NTC1	NTC0	Reserved

31 20	19	18 17	16 15	14 13	12 11
Reserved	DREE	NTM3	NTM2	NTM1	NTM0

When MMU_CFG.MMPS is 0 or 1, the D, TBALCK, and MPZIU fields at MMU_CTL may either have no functionality or become “reserved” (it is implementation-dependent). For the purpose of this specification, MMU_CFG.MMPS should be 1.

When MMU_CFG.NTME is 0, the NTM0-NTM3 fields should become “reserved” fields.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Reserved</td><td>1
(0)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>NTC0</td><td>2
(2,1)</td><td>Indicates Non-translated Cacheability memory attribute for partition 0.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Non-cacheable/Non-coalesable</td></tr>
<tr><td>1</td><td>Non-cacheable/Coalesable</td></tr>
<tr><td>2</td><td>Cacheable/Write-Back</td></tr>
<tr><td>3</td><td>Cacheable/Write-Through</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
NTC1	2 (4,3)	Indicates Non-translated Cacheability memory attribute for partition 1. See NTC0 description for its value definition.	RW	0
NTC2	2 (6,5)	Indicates Non-translated Cacheability memory attribute for partition 2. See NTC0 description for its value definition.	RW	0
NTC3	2 (8,7)	Indicates Non-translated Cacheability memory attribute for partition 3. See NTC0 description for its value definition.	RW	0
Reserved	2 (10,9)	Reserved	RAZWI	0
NTM0 (MMU_CFG.NTME == 1)	2 (12,11)	Indicates Non-translated VA to PA[31:30] mapping for partition 0.	RW	0
NTM1 (MMU_CFG.NTME == 1)	2 (14,13)	Indicates Non-translated VA to PA[31:30] mapping for partition 1.	RW	1
NTM2 (MMU_CFG.NTME == 1)	2 (16,15)	Indicates Non-translated VA to PA[31:30] mapping for partition 2.	RW	2
NTM3 (MMU_CFG.NTME == 1)	2 (18,17)	Indicates Non-translated VA to PA[31:30] mapping for partition 3.	RW	3
DREE	1 (19)	Device register endian control enable. Value: 0 — Meaning: Device register endian control function is disabled. A data memory access uses PSW.BE as the endian type.	RW	0

Field Name	Bits	Description	Type	Reset
		1 — Device register endian control function is enabled. A data memory access with the following attributes, (C==0 or NTC==0), uses PSW.DRBE as the endian type while a data memory access with the other attributes uses PSW.BE as the endian type.		
Reserved	12 (31,20)	Reserved	RAZWI	0

NTC0-NTC3 are the Non-translated Cacheability attributes used when the VA to PA translation process is turned off. The value definition of the NTC field is listed in Table 2 in section 2.2.2.1. When only VA[31] is used to access the NTC fields, only NTC0/NTC1 are used. When VA[31:30] is used to access the NTC fields, all NTC fields will be used. The NTC field assignment for each case is listed in Table 9 and Table 10. For reduced 24-bit address space configuration (MSC_CFG.ADR24 is 1), VA[23] or VA[23:22] is used.

NTM0-NTM3 are the Non-translated VA[31:30] to PA[31:30] Mapping fields when MMU_CFG.NTME is equal to 1. It is most often used to improve memory access performance on a small system without MMU or MPU by mapping different NTC attributes (e.g., cacheable/WB, cacheable/WT) to the same PA partition. Thus, the memory access type can be controlled using VA[31:30]. When only VA[31] is used to access the NTM fields, only NTM0/NTM1 are used. When VA[31:30] is used to access the NTM fields, all NTM fields will be used. The NTM field assignment for each case is listed in Table 9 and Table 10. For reduced 24-bit address space configuration (MSC_CFG.ADR24 is 1), VA[23] or VA[23:22] is used.

Table 9. NTC field assignment for indexing using VA[31]

VA[31]	NTC field	NTM field
1'b0	NTC0	NTM0
1'b1	NTC1	NTM1

Table 10. NTC field assignment for indexing using VA[31:30]

VA[31:30]	NTC field	NTM field
2'b00	NTC0	NTM0
2'b01	NTC1	NTM1
2'b10	NTC2	NTM2
2'b11	NTC3	NTM3

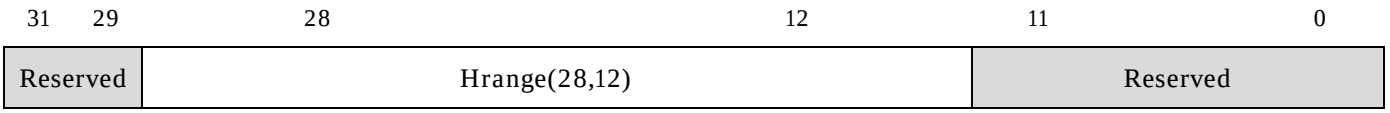
3.4.2. TLB Access VPN Register

Mnemonic Name: mr2 (TLB_VPN)

IM Requirement: MPU optional

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 2, 0}



For TLB target write operations, this register contains the high virtual address limit of the MPU TLB entry to be installed. For TLB target read operations, this register holds the same read out.

Field Name	Bits	Description	Type	Reset
Hrange	17 (28,12)	Contains the Hrange field to and from the MPU TLB structure.	RW	DC

Note that this register does not get updated with any exception VPN information as done in the MMU specification. This is due to the fact that it is Hrange, not exception VPN, which gets written into the MPU TLB structure. So there is no need for hardware to prepare the exception VPN in advance in the TLB_VPN register to reduce software effort.

3.4.3. TLB Access Data Register

Mnemonic Name: mr3 (TLB_DATA)

IM Requirement: MPU optional

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 3, 0}

31 12	11 9	8 6	5 4	3 1	0
PSB(31,12)	Reserved	C	X	M	V

For TLB target write operations, this register contains the physical section base, and various memory attribute information of the MPU TLB entry to be installed. For TLB target read operations, this register contains the same read out.

Field Name	Bits	Description	Type	Reset
V	1 (0)	This entry is valid and present.	RW	DC
M	3 (3,1)	Contains the M field to and from the MPU TLB structure.	RW	DC
X	2 (5,4)	Contains the X field to and from the MPU TLB structure.	RW	DC
C	3 (8,6)	Contains the C field to and from the MPU TLB structure.	RW	DC
Reserved	3 (11,9)	Reserved	RAZWI	0
PSB	20 (31,12)	Contains the PSB field to and from the MPU TLB structure.	RW	DC

3.5. MPU Exceptions

Five possible exceptions could be generated after accessing the MPU TLB entry. Their names and generating conditions are listed in the following table.

Name	Generating condition	Entry vector
MPU TLB Invalid	The MPU TLB entry is invalid, that is, MPU[VA(31,29)].V == 0	TLB fill (the same entry point of the MMU TLB fill exception)
Read protection violation	The load operation has no read permission. It can occur when there is a conflict between MPU[VA(31,29)].M and the processor operating mode.	TLB misc.
Write protection violation	The store operation has no write permission. It can occur when there is a conflict between MPU[VA(31,29)].M and the processor operating mode.	TLB misc.
Non-executable code	The fetched code has no permission to be executed. It can occur when there is a conflict between MPU[VA(31,29)].X and the processor operating mode.	TLB misc. (the same entry point of the MMU non-executable page exception)
Reserved attribute	The value read out from the MPU TLB entry MPU[VA(31,29)] contains a reserved value in C or M field.	TLB misc. (the same entry point of the MMU reserved PTE attribute exception)

4. Interruption Architecture

This chapter describes interrupt architecture and contains the following sections:

4.1. Introduction

In Andes architecture, an *Interruption* is a control flow change of normal instruction execution generated by an *Interrupt* or an *Exception*. When an interruption occurs, the processor stops processing current flow of instructions, saves enough states for later resuming of the interrupted current flow, disable interrupts, enters superuser Mode, and starts executing a software interruption handler.

An *interrupt* is a control flow change event generated by an asynchronous internal or external source. It includes hardware and software interrupts. A hardware interrupt is any interrupt event generated from external agents or AndesCore™. A software interrupt is an interrupt event generated by an instruction executing in the AndesCore.

An *exception* is a control flow change event generated as a by-product of instruction execution. For example, if an instruction with invalid format is encountered by an Andes processor core, a Reserved Instruction exception will be generated. An exception can also be classified as *precise*, *next-precise*, or *imprecise*.

An exception is said to be *precise* if when the exception handler is entered the instruction that causes the exception has not been completed. For example, the Reserved Instruction exception mentioned above is a precise exception.

An exception is said to be *next-precise* if when the exception handler is entered the instruction that causes the exception has been completed but the next instruction has not been completed. For example, a data watchpoint exception can be implemented as a next-precise exception.

An exception is said to be *imprecise* if when the exception handler is entered the instruction that causes the exception has been completed in the processor pipeline but the exact instruction complete/incomplete boundary following that instruction is unpredictable. For example, a Bus Error exception caused by a prefetch instruction is an imprecise exception. Most of the exceptions in the Andes Architecture are precise exceptions.

4.2. Interruption Handling in Hardware

In Andes architecture, an interruption is handled in hardware based on the "interruption stack level transition." Four interruption stack levels (0-3) are defined in the architecture. The interruption stack level 0 indicates that there is no interruption. The detailed behavior for interruption stack level transition is described in the following sections (4.2.1 and 4.2.2).

However, debug exceptions (and exceptions suppressed during debugging) do not invoke interruption stack level transition in V3 architecture (MSC_CFG.BASEV is 2). Instead, the debug exception has its own states: Debug IPC (DIPC) and Debug Event Type (DETYPE) defined at EDM Status Word (dr41 or EDMSW). This feature prevents corruption of interruption stack and enhances debugging capability. It may be disabled for backward compatibility using EDM_CTL.EDM_MODE. For more details, please refer to the Andes Embedded Debug Module V3 Specification.

4.2.1. Interruption Handling for Interruption Stack Level Transitions 0/1 and 1/2

For interruption stack level transitions 0/1 and 1/2, key interruption states are saved to and restored from hardware interruption stack registers. These states need to be saved and restored since they are updated during the transition to higher levels of interruption (0 → 1, and 1 → 2). More specifically, the following states are updated before fetching and executing the first instruction in the interruption handler:

- (PSW) Interruption Stack Level ← Interruption Stack Level ++
- (PSW) Global Interrupt Enable (GIE) ← 0
- (PSW) Privilege Mode ← 1 (Superuser mode)
- (PSW) IT/DT ← 0
- (PSW) BE ← default endian (MMU_CFG.DE)
- (PSW) DRBE ← MMU_CFG.DRDE
- (PSW) IME ← (instruction-related Machine Error) ? 1 : IME
- (PSW) DME ← (data-related Machine Error) ? 1 : DME
- (PSW) DEX ← (Debug Exception) ? 1 : 0
- (PSW) HSS ← (PSW.HSS) ? INT_MASK.DSSIM : 0

- (PSW) PFT_EN ← (PFT_CTL.FAST_INT) ? 0 : PFT_EN
- Interruption Type
- Exception VA (meaningful for only certain exceptions)
- Program Counter ← Interruption vector entry point

Before updating these values, the old content of these registers are saved to the lower level interruption stack registers.

Saving/restoring operations for interruption stack level transition

When new interruption occurs, an AndesCore automatically performs the following register saving:

- P_IPSW ← IPSW ← PSW
- P_IPC ← IPC ← PC
- P_ITYPE ← ITYPE
- P_EVA ← EVA
- P_P0 ← P0
- P_P1 ← P1

When exiting from an interruption, an AndesCore automatically restores the following registers:

- P_IPSW → IPSW → PSW
- P_IPC → IPC → PC
- P_ITYPE → ITYPE
- P_EVA → EVA
- P_P0 → P0
- P_P1 → P1

This register stacking design can handle nested error exceptions even if the software handler is not ready. More precisely, it allows exceptions to occur again even before executing any software handler.

4.2.2. Interruption Handling for Interruption Stack Level Transition 2/3

When the interruption stack level transitions from 2 to 3, only very limited interruption states are updated since the interruption stack level 3 is considered as a severe error state (e.g., nested kernel exceptions). More specifically, the following states are updated to permit minimum error handling:

- (PSW) Interruption Stack Level ← 3 (i.e., maximum stack level)
- (PSW) IME ← (instruction-related Machine Error) ? 1 : IME
- (PSW) DME ← (data-related Machine Error) ? 1 : DME
- (PSW) DEX ← Debug Exception ? 1 : 0
- Overflow_IPC ← Program Counter
- Program Counter ← Machine Error exception entry point

In addition, at the maximum interruption stack level or in the debugging mode, hardware continues to perform the following default behavior (without modifying the corresponding control bits):

- Disable interrupt.
- Turn off instruction/data address translation.
- Use “default endian (MMU_CFG.DE)” as the data access endian.
- Use “MMU_CFG.DRDE” as the device register access endian if MMU_CTL.DREE is asserted.
- For EDM_CFG.VER ≧ 0x0030, disable Hardware Single Stepping (HSS).
- For V3 architecture and above (MSC_CFG.BASEV >= 2), set to superuser mode.
- Disable the performance throttling.

Despite the above default behavior, the debugger can still overwrite it by setting EDM_CTL.DEX_USE_PSW and changing the contents of PSW. This gives the debugger maximum flexibility to further facilitate debugging. For example, a debugger may wish to turn on virtual translation or even enable interrupts in some particular situations.

When an AndesCore returns from interruption stack level 3 to interruption stack level 2 by executing a “return from interruption” instruction, the following states will be updated:

- Program Counter ← Overflow_IPC
- Interruption Stack Level ← 2 (i.e., maximum stack level minus 1)

Note that, to enhance debugging, all interrupts and exceptions (excluding reset and debug) at the maximum (overflow) interruption stack level are redirected to a single common entry point. By redirecting all but two interruptions to the common “machine error” entry point (number 5), exception handlers can effectively prevent this severe error condition from triggering more types of exceptions and thus make detection easier. This redirecting of all interruptions at the maximum level includes two types of transitions: “from Max-1 to Max” and “from Max to Max.” As for the reason excluding resets and debug exceptions, it is because they inherently have relatively different natures and purposes. This feature only exists in non-MCU CPUs (MSC_CFG.MCU == 0).

At the maximum level, there is no corresponding ITYPE register to store information such as ETYPE and VECTOR. The existing information in the ITYPE register still belongs to the lower interruption stack level so it cannot be overwritten. Similarly, there is no corresponding IPSW to store the SP_ADJ flag, so 8-byte stack pointer alignment will not be performed for exceptions at the maximum interruption stack level. As a result, the machine error exception handler first has to check the interruption level and, if it reaches the maximum level, perform extra handling. More specifically, if at the maximum interruption stack level, it needs to perform 8-byte alignment as needed and try to determine the cause without ITYPE information.

In V2 EDM compatibility mode, the behavior of interruptions remains the same because EDM compatibility mode only affects the behavior of EDM but not the CPU. Since the handlers in V3 architecture already assume interruptions at the maximum level will be redirected to a common entry point, this behavior will not change just because an old V2 ICE debugger is plugged in.

4.2.3. Interruption Handling for Interruption Stack Level Transition 3/3

The interruption stack level 3 is the maximum level and considered as a severe error state. Therefore, any further interruptions (excluding resets) neither increase the level (i.e., unchanged, from 3 to 3) nor update the interruption stack related system registers. The reason for not updating interruption stack registers is to preserve the states of the first event triggering this severe error (which should be debugged first) and not be overwritten by further interruptions.

Nevertheless, there is one special case to the rule of not updating interruption stack registers at the maximum level: a debug exception always sets the debug exception flag (PSW.DEX) to 1 regardless of the interruption level. Note that the PSW.DEX flag becomes sticky in this case which is different from the update rules in other interruption levels.

- (PSW) DEX ← Debug Exception ? 1 : DEX

4.2.4. Maximum Interruption Stack Level Option

To save hardware cost, Andes architecture provides the flexibility of configuring the maximum interruption stack level to be either 3 or 2, and this configuration is recorded at the INTLC field in the MISC_CFG system register.

The previous two sections (4.2.1 and 4.2.2) describe the operations performed at the interruption stack level transitions between 0/1, 1/2, and 2/3 when the INTLC field is set to zero (i.e., the maximum interruption stack level is 3).

When the INTLC field is set to 1 (i.e., the maximum interruption stack level is 2), all the P_* interruption stack system registers will be removed from an implementation. In this case, the operations performed at the interruption stack level transition between 1/2 will be changed to operations equivalent to those performed at the interruption stack level transition between 2/3 when the maximum interruption stack level is 3.

4.2.5. Software Lowering Interruption Stack Level

To support unlimited levels of nested interruption, software needs to lower the interruption stack level since the depth of the hardware interruption stack is limited. The operations to lower the interruption stack level include: saving lower-level interruption stack registers, reducing the interruption stack level, and later, after the nested interruptions complete, restoring lower-level interruption stack registers and increasing the interruption stack level. These procedures can be shown in the following figures.

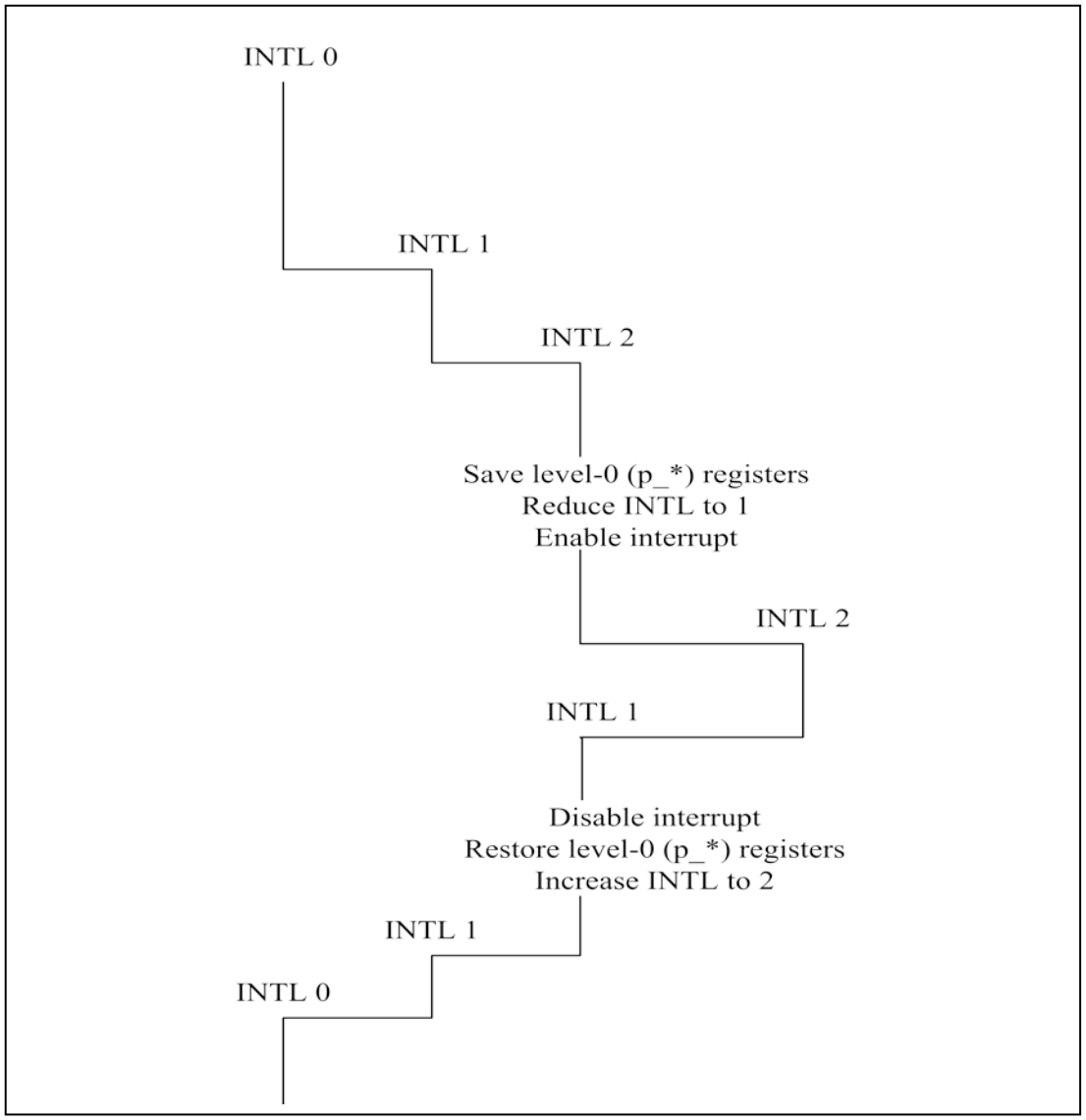


Figure 5. Operations of lowering the interruption stack level from 2 to 1

Lowering the interruption stack level from 2 to 1

As Figure 5 shows, at interruption stack level 2, the interruption stack contains level-0 (p_*), level-1 (i*), and level-2 information. Lowering interruption stack level from 2 to 1 requires changing the level-1 (i*) to level-0, and changing the level-2 to level-1. As a result, the original level-0 (p_*) states will be overwritten by level-1 and thus these level-0 states need to be saved before lowering the levels. The system registers needed to be saved in this case include P_IPSW, P_IPC, P_ITYPE, P_EVA, P_P0, and P_P1.

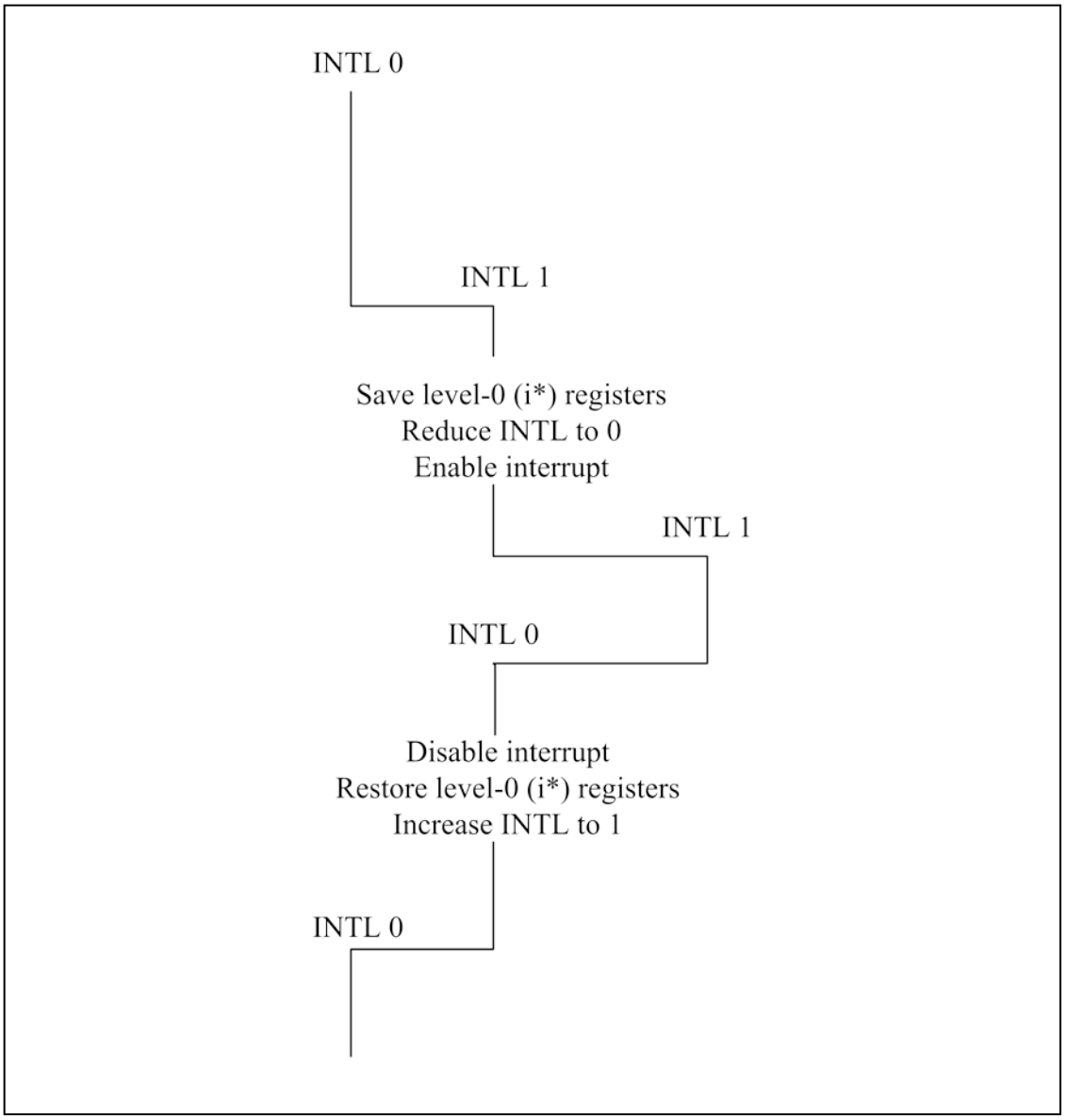


Figure 6. Operations of lowering the interruption stack level from 1 to 0

Lowering the interruption stack level from 1 to 0

As Figure 6 shows, at interruption stack level 1, the interruption stack contains level-0 (i*), and level-1 information. Lowering interruption stack level from 1 to 0 requires changing the level-1 to level-0. As a result, the level-0 (i*) states will be overwritten by level-1 and thus these level-0 (i*) states needs to be saved before lowering the levels. The system registers needed to be saved in this case include IPSW, IPC, ITYPE, EVA, P0, and P1.

Lowering interruption stack level from 3 to 2

Interruption stack level 3 is the overflow interruption level. At interruption stack level 3, the interruption stack contains level-0 (p_*), level-1 (i*), and level-2 information plus additional level-3 OIPC state. In this level, the level-3 interruption type and exception VA are not recorded in any register.

If there is a need to lower the interruption stack level from 3 to 2, more steps are required to re-arrange the interruption stack states to the correct stack level hierarchy. Basically, software is required to (1) save the level-0 (p_*) information; (2) emulate a stack push operation by moving IPSW, IPC, ITYPE, EVA, P0, and P1 to their corresponding P_* registers; (3) move PSW to IPSW, and OIPC to IPC; and (4) update PSW to the intended operating values along with the stack level change. Once the update of PSW completes, the CPU operates at the interruption stack level 2.

4.3. Types of Interruptions

This section lists the types of Interruptions which can be recognized by the combination of individual vector entry point and type information stored in the ITYPE system register. They are as follows:

- Reset/NMI
  - Cold Reset
  - Warm Reset
  - Non-Maskable Interrupt (NMI)
- Interrupt
  - External interrupt
  - Software interrupt
  - Performance counter interrupt
  - Local DMA interrupt
- Debug exception
  - Instruction breakpoint
  - Data address & value break
  - BREAK exception
  - Hardware single stepping
  - Load/store instruction global stop
- TLB fill exception (Instruction/Data)
- PTE not present exception (Instruction/Data)
  - Non-Leaf PTE not present
  - Leaf PTE not present
- TLB miscellaneous exception
  - Read protection violation (Data)
  - Write protection violation (Data)
  - Page modified (Data)
  - Non-executable page (Instruction)
  - Access bit (Instruction/Data)
  - Reserved PTE Attribute (Instruction/Data)
  - TLB VLPT miss (Instruction/Data)

- System Call exception
- General exception
  - Branch target alignment (Instruction) or alignment check exception (Data) exception
  - Reserved instruction exception
  - Trap exception
  - Arithmetic exception
  - Precise bus error (Instruction/Data)
  - Imprecise bus error (Instruction/Data)
  - Coprocessor N not-usable exception
  - Coprocessor N-related exception
  - Privileged instruction exception
  - Reserved value exception
  - Nonexistent memory address (Instruction/Data)
  - MPZIU control (Instruction/Data)
  - Next-precise stack overflow exception
- Machine error exception (Instruction/Data)
  - Cache locking error
  - TLB locking error
  - TLB multiple hit
  - Parity/ECC error
  - Unimplemented page size error
  - Illegal parallel memory accesses (Audio extension)
  - Local memory base setup error (for Unified Local Memory configuration)

4.4. Interruption Vector Entry Point

When an interruption occurs, instruction execution is directed to the interruption handler that can handle the corresponding interruption event. To speed up the interruption handling based on relative importance, dedicated entry points (in a vector table form) are assigned to different types of interrupts in order to promptly separate them and thus reduce the software efforts to dispatch them. The main design principle here is to quickly handle the TLB-fill and external interrupt events because they occur more frequently.

The Andes Interruption Vector Entry Points are defined for two different modes. The first mode is the Internal Vector Interrupt Controller (IVIC) mode where interrupts are prioritized inside an AndesCore using the internal interrupt controller logic. The second mode is the External Vector Interrupt Controller (EVIC) mode where interrupts are prioritized outside AndesCore using an external interrupt controller. These two modes can be configured using the "EVIC" field/bit of the Interruption Vector Base (IVB) system register. However, in some implementations (e.g., MCU family with MSC_CFG.MCU == 1), only one interrupt vector mode (internal only) is supported but not both (internal and external). In this case, the IVB.EVIC becomes read-only and not writable.

The Andes Interruption Vector Entry Points for the two modes are listed in the following subsections. The starting address of the vector table is 64KB-aligned and can be programmed using the "Interruption Vector Base" (IVB) system register. The size of the vector entry point can be configured to 4/16/64/256 bytes by setting the "ESZ" field in the IVB system register. Each vector entry point stores the beginning instructions of the corresponding interruption handlers. Therefore, if an interruption handler is short, the entire handler may even be stored in the vector entry point and directly executed from there (without further control transfer) for fast response time.

Note that, to enhance debugging, all interrupts and exceptions (excluding reset and debug) at the maximum interruption stack level are redirected to the "machine error" entry point (5). Please refer to section 4.2.2 for more details.

4.4.1. Entry Points for Internal VIC Mode

In V3 architecture (MSC_CFG.BASEV is 2), the maximum number of interrupt input sources for IVIC mode is extended from 6 to 32. The exact number of interrupt input sources is implementation-dependent and is recorded in IVB.NIVIC.

- 41 entry points (9 exceptions + 32 interrupts)
- 6-bit offset
- Address = IVB.IVBASE + (entry number) * IVB.ESZ

Entry number	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9	HW 0
10	HW 1
11	HW 2
12	HW 3
13	HW 4
14	HW 5
15	HW 6
16	HW 7
17	HW 8
18	HW 9
19	HW 10

Entry number	Entry point
20	HW 11
21	HW 12
22	HW 13
23	HW 14
24	HW 15
25	HW 16
26	HW 17
27	HW 18
28	HW 19
29	HW 20
30	HW 21
31	HW 22
32	HW 23
33	HW 24
34	HW 25
35	HW 26
36	HW 27
37	HW 28
38	HW 29
39	HW 30
40	HW 31

Note the entry points are different if compatibility mode for baseline version V2 is asserted with V2_INT_MODE pin. For more details, please refer to section 4.9 and the AndeStar System Privilege Architecture Version 2 Manual.

4.4.2. Entry Points for External VIC Mode

- 73 entry points (9 exceptions + 64 interrupts)
- 7-bit offset
- Address = IVB.IVBASE + (entry number) * IVB.ESZ

Entry number	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9-72	VEP 0-63

For each entry point, multiple interruptions can be mapped to it. Thus the interruption type (ITYPE) system register is needed to record the different types of interruptions. See "Interruption Type" system register description for detailed type information.

4.5. Priority of Interruptions

When multiple interruptions occur simultaneously, the interruption having the highest priority in the following table will be taken by the AndesCore as the selected interruption event for an interruption handler to process.

Table 11. Interruption priority table

Interruption Name (highest to lowest)	Vector Entry Point
cold reset/warm reset	Reset/NMI
Local memory base setup error – next-precise	Machine error
Stack overflow/underflow exception – next-precise	General exception
Debug data address watchpoint – next-precise	Debug
Debug data value watchpoint – next-precise	Debug
Hardware single-stepping	Debug
NMI	Reset/NMI
External debug request (debug interrupt)	Debug
ECC error - imprecise	Machine error
Data bus error – imprecise*	General exception
Instruction bus error - imprecise* (e.g., HW prefetch)	General exception
Interrupt	(Based on interrupt priority)
Debug instruction breakpoint	Debug
Instruction alignment check	General exception
ITLB multiple hit	Machine error
ITLB fill	TLB fill
ITLB VLPT miss	TLB VLPT miss
IPTE not present (non-leaf)	PTE not present
IPTE not present (leaf)	PTE not present
Reserved IPTE attribute	TLB misc
Instruction MPZIU control	General exception

Interruption Name (highest to lowest)	Vector Entry Point
ITLB non-execute page	TLB misc
IAccess bit	TLB misc
Instruction machine error	Machine error
Instruction nonexistent memory address	General exception
Instruction bus error (precise)	General exception
Reserved instruction	General exception
Privileged instruction	General exception
Reserved value	General exception
Unimplemented page size error	Machine error
Break	Debug
Coprocessor	General exception
Trap/syscall	General exception
Arithmetic	General exception
Branch target alignment	General exception
Debug data address watchpoint	Debug
Load/store instruction global stop	Debug
Data alignment check	General exception
DTLB multiple hit	Machine error
DTLB fill	TLB fill
DTLB VLPT miss	TLB VLPT miss
DPTE not present (non-leaf)	PTE not present
DPTE not present (leaf)	PTE not present
Reserved DPTE attribute	TLB misc
Data MPZIU control	General exception
DTLB permission (R/W)	TLB misc
Dpage modified	TLB misc
DAccess bit	TLB misc

Interruption Name (highest to lowest)	Vector Entry Point
Data machine error	Machine error
Data nonexistent memory address	General exception
Debug data value watchpoint - precise	Debug
Data bus error – precise	General exception

*Note: Imprecise instruction bus error and imprecise data bus error have the lowest and second-lowest priority respectively in the Interruption Architecture Version 0 (MSC_CFG.INTV is 0).

4.6. Interruption Related Registers

Interruption related registers can be classified into three different categories. One category (Interruption Control Register) contains registers which control interruption behaviors. The second category (Interruption Info Register) contains registers which are used to save certain information when an interruption happens. The third category (Interruption Shadow Register) contains registers which are used to shadow Interruption Info Registers for handling nested interruptions. A summary of these registers is listed in Table 12. Please see Section 10.3 Interruption System Registers for their detailed definitions.

Table 12. Interruption related registers

Name	Type	Updater	Section
Interruption Control Register			
Processor Status Word	RW	SW/HW	10.3.1
Interruption Vector Base	RW	SW	10.3.4
Interruption Masking	RW	SW	10.3.15 10.3.16
Interruption Pending	RW or W1C	SW/HW	10.3.17 10.3.18
Interrupt Priority	RW	SW	10.3.21 10.3.22
Interrupt Control	RO or RW	SW	10.3.23
Interruption Info Register			
Exception Virtual Address	RW	HW	10.3.5
Interruption Type	RW	SW/HW	10.3.7
Machine Error Log	W1C	HW	10.3.9
Interrupt Trigger Type	RO	HW	10.3.24
Interruption Shadow Register			
Interruption PSW	RW	SW/HW	10.3.2

Name	Type	Updater	Section
Previous IPSW	RW	SW/HW	10.3.3
Previous EVA	RW	SW/HW	10.3.6
Previous ITYPE	RW	SW/HW	10.3.8
Interruption Program Counter	RW	SW/HW	10.3.10
Previous IPC	RW	SW/HW	10.3.11
Overflow IPC	RW	SW/HW	10.3.12
Previous P0 / Previous P1	RW	SW/HW	10.3.13 10.3.14
Shadowed User Stack Pointer	RW	SW	10.3.19
Shadowed Privileged Stack Pointer	RW	SW	10.3.20

Basically, the following register stacks are built to handle potentially 2 levels + 1 error level nested interruptions in hardware.

Updates performed on interruption stack level transitions 0/1 and 1/2:

- PC ⇔ IPC ⇔ P_IPC
- PSW ⇔ IPSW ⇔ P_IPSW
- VA ⇒ EVA ⇔ P_EVA
- Interruption ⇒ ITYPE ⇔ P_ITYPE
- P0 ⇔ P_P0
- P1 ⇔ P_P1

Updates performed on interruption stack level transition 2/3

- PC ⇔ OIPC

The level 2 interruption is typically used to handle TLB VLPT miss exception, recoverable error condition, or hypervisor mode (if implemented) interception. The level 3 interruption is typically used to handle error events in the level 2 interruptions. If level 3 interruption is entered, the interruption handler may not be able to return to the interrupted level 2 interruption instruction flow.

4.7. Imprecise Exception

An imprecise exception is generated after the completion of the instruction that causes this exception. The delay between the reporting of an imprecise exception and the completion of its generating instruction is unpredictable. As a result, which instruction or even which process will be interrupted by the reporting of an imprecise exception is unknown. However, the interruption stack at high levels may be interfered by the occurrences of unexpected imprecise exceptions. To provide minimum protection for the interruption stack, an implementation may only allow an imprecise exception to be taken when the interrupt stack has enough resources to accommodate additional imprecise exceptions. This way, a group of successive imprecise exceptions will not overflow and corrupt the interruption stack.

4.8. Details of Individual Interruptions

4.8.1. Cold Reset

A cold reset exception occurs when the cold reset pin connected to an AndesCore™ is asserted. This exception is not maskable and forces the AndesCore to perform complete processor state initialization.

To identify a cold reset, the reset handler can use "BEQZ $SP" to check if the stack pointer equals to zero. This is because the stack pointer is only cleared by a cold reset but not by any other types of exceptions.

4.8.2. Warm Reset

A warm reset exception occurs when an optional warm reset pin connected to an AndesCore is asserted. Like a cold reset, this exception is not maskable but it only performs minimal processor state initialization. The purpose of the warm reset is to free up the CPU from any stall or lockup problems while preserving most processor states for debugging.

Note that the warm reset may not be implemented in some AndesCores (e.g., older versions or CPUs of the MCU family with MSC_CFG.MCU == 1), so please confirm its availability with the datasheet.

Listed below are the detailed operations performed by hardware on a warm reset exception. These operations are the same as those performed when the interruption stack level increases from 0 to 1. When a warm reset occurs, these operations effectively save the states for future debugging:

- P_IPSW ← IPSW ← PSW
- P_IPC ← IPC ← PC
- P_ITYPE ← ITYPE
- P_EVA ← EVA
- P_P0 ← P0
- P_P1 ← P1

After the above operations are done, an AndesCore further performs the following initialization operations:

- PC ← {IVB.IVBASE, 16'b0}
- PSW ← reset value (of PSW)
- ITYPE.ETYPE ← 1
- Reset all pipeline stall logic and internal state machines.

The exception handler or debugger can then retrieve IPC and IPSW to debug the status when the warm reset was originally triggered. Note that IPC/IPSW always contain the most recent states (at the warm reset), and this differs from the case where interruption stack level is at the maximum (in which OIPC contains the most recent state).

To preserve most information for debugging, a warm reset does not reset the stack pointer nor adjust it to be 8-byte aligned. In addition, it forces the CPU to wake up from standby mode immediately regardless of the status to allow debugging. However, if a warm reset should fail to free up the CPU, users need to apply a cold reset as the next resort.

To identify a warm reset exception, the reset handler needs to use the stack pointer to free up a register and then use this register to read out ITYPE.ETYPE (which should have the value 1). This can effectively distinguish a warm reset from a cold reset and Non-Maskable Interrupt (NMI) which all share the same vector entry.

4.8.3. Non-Maskable Interrupt (NMI)

A Non-Maskable Interrupt exception happens when the NMI signal is asserted to an AndesCore. This exception is not maskable including using PSW.GIE bit. Note that although the name has the word “interrupt” in it, this is really an exception and not an interrupt.

4.8.4. Machine Error

Due to the nature of handing machine errors for this exception handler, the instruction and data memory accesses within this exception handler will use un-cached space depending on if it is an instruction-related or data-related error.

The machine error exception includes several different types of errors. Please see Table 24 (page 199) for detailed error types.

4.8.5. Debug Exception

Debug exceptions are triggered when debug interrupts or debug events (e.g., breakpoints or watchpoints) occur. They are designed to help programmers finding potential errors and can be handled by the target CPU (target-mode debugging) or host CPU (host-mode debugging). For more details, please refer to the Andes Embedded Debug Module V3 Specification.

Since V3 architecture (MSC_CFG.BASEV is 2), debug exceptions are decoupled from the regular interruption stack to avoid potential interference and corruption. Instead, debug exceptions have their own dedicated registers to record information and do not share storage with normal hardware interruption stacks. As a result, debugging exceptions do not change interruption stack level so the interruption stack stays unchanged and all system register values are thus preserved.

Another advantage is it reduces confusion since the debugger's view of the debugged program's IPC, IPSW, and ITYPE are no longer skewed to P_* registers. Nevertheless, for backward compatibility, all V3 EDM features may be disabled and reverted back to the old V2 mode using EDM_CTL.EDM_MODE.

To record debugging states, new Debugging Interruption Program Counter (DIPC) and Debugging Event Type (DETYPE) registers are created. This DIPC reuses the same physical register of Overflow Interruption Program Counter (OIPC) to save hardware cost. This sharing is feasible because normal debugging only occurs at or below "the maximum interruption level – 1" where OIPC is not used in the normal execution. In case the maximum interruption level does occur, the debugger will lose its DIPC (now occupied by OIPC) and thus only limited debugging can be performed. The DETYPE is recorded in the EDM Status Word (dr41 or EDMSW).

When a debug exception occurs, hardware automatically sets the debug exception bit (PSW.DEX) to "1" to ensure correct default behavior for the debug mode. More specifically, when DEX is set to "1" (during debug mode), hardware will enforce the following behavior without modifying the corresponding control bit in PSW. In other words, the settings in PSW lose effects and are now controlled by the DEX default mode. This default behavior is necessary to guarantee correct operations of the debugger and to preserve the original content of PSW. PSW needs to be preserved because it is no longer backed up by the hardware interruption stack in a debug exception.

- (PSW) DEX ← 1

  Enforce the following default behavior:

  - Disable all interrupts.
  - Become superuser mode.
  - Turn off instruction/data address translation.
  - Use "default endian (MMU_CFG.DE)" as the data access endian.
  - Use "MMU_CFG.DRDE" as the device register access endian if MMU_CTL.DREE is asserted.
  - Disable audio special features.
  - Disable inline function call (IFC).

When debugging exception ends (executing an "IRET" instruction), hardware sets the DEX bit back to "0" and updates PC using DIPC:

- (PSW) DEX ← 0
- PC ← DIPC

Please note that, when DEX == 1 (during debug mode), an "IRET" instruction will update PC from DIPC and not from the regular IPC. In addition, the interruption stack level (INTL) is not affected by the debugging exception anymore in V3, so the interruption stack does not change at all.

To give debugger greater or finer control, a debugging environment control bit (EDM_CTL.DEX_USE_PSW) is added to select either the default behavior of (PSW.DEX == 1) or new behavior resulting from a customized PSW. This control bit allows the debugger to have new behavior such as turning on interrupts or turning on instruction/data address translation. Even if the debugger is connected at the maximum interruption stack level, the DEX_USE_PSW bit still takes effect and supersedes the default behavior of the maximum interruption stack level. This is to allow the debugger to have maximum flexibility.

In host-mode debugging (debugging through host CPU), exceptions are "suppressed" to decouple from target exception handlers and minimize disturbance. This allows the debugger to finish its task smoothly without concerns of losing control due to exceptions. Note that the "SYSCALL" is not suppressed during debugging mode because it is well controlled and invoked by the debugger itself.

In addition, most fields in the Interruption Type Register (ITYPE) become "writable" because the debugger needs to reuse them. More specifically, the debugger can no longer use the interruption stack to save information of suppressed exceptions, so the debugger needs to reuse fields in ITYPE to save this information. Therefore, the debugger must first save these ITYPE fields, reuse them by overwriting with new information of suppressed exception, and finally restore them before exit. These reused ITYPE fields that become "writable" include: SWID, INST, ETYPE, SUB_TYPE and CPID. This is because, when debugging system calls or exception handlers, these fields may contain active information which must be restored.

Note that the instructions causing suppressed exceptions will generate undefined results. Also, the host debugger should not provide an unaligned returned address (OIPC); otherwise, unpredictable behavior may occur.

Suppressed exceptions behave just like normal exceptions (updating all system registers/signals like MERR and trace unit) except the following:

- Next PC should use the sequential PC (instead of the corresponding exception entry point).
- Do not update interruption stack registers except the following:
  - PSW.DEX.
  - ITYPE register (which includes fields like ETYPE, INST, VECTOR, SUPRS_EXC, SUB_TYPE, and SWID). SUPRS_EXC and IMP_EXC are cleared initially when PSW.DEX becomes 1, and these flags are set when suppressed exceptions occur. VECTOR is not modified initially by the debug exception but may be updated during debug mode (should be backed up and restored by software).
  - OIPC (when it is not at the maximum interruption stack level).

The Exception Virtual Address Register (EVA) is not updated for debug exceptions since V3 architecture (MSC_CFG.BASEV is 2). In addition, ITYPE.VECTOR is not updated when the maximum interruption stack level is reached. This behavior follows the principle that the maximum level does not overwrite information from lower levels.

To record exceptions, new status flags are added to be referred by the debugger and host. More specifically, two flags are added internally at the system register for the CPU to access, and a third flag is added externally at EDM Misc. Registers for the host to access.

When entering host-mode debugging, the debugger needs to save the content of the ITYPE register (ETYPE and INST) to make room for recording future exceptions; similarly, the debugger also needs to save SWID if it plans to make system calls. However, an imprecise bus error exception caused by previous instructions can barge in and overwrite ITYPE before the debugger finishes saving it. Therefore, to avoid this potential information loss, a flag called IMP_EXC is added to record imprecise exceptions (e.g., imprecise bus error exceptions), while other ITYPE information (ETYPE and INST) is not modified for imprecise exceptions. This

IMP_EXC flag is set by hardware when PSW.DEX is 1 and an imprecise exception occurs; it is cleared by hardware when initially PSW.DEX becomes 1, or by the debugger when it finishes servicing an imprecise exception. The IMP_EXC flag is recorded at the Interruption Type Register (ITYPE).

In addition, one more flag called SUPRS_EXC is added to indicate the occurrence of a suppressed precise or next-precise exception during debugging mode (excluding imprecise exceptions). This flag indicates whether an exception actually occurs or not since ETYPE only records the reason but not the occurrence. It is set by hardware when PSW.DEX is 1 and an exception occurs (except for debug exceptions which are ignored silently); it is cleared by hardware when PSW.DEX initially becomes 1, or by the debugger when it initially saves ETYPE or finishes servicing a suppressed exception. Note that, for each particular exception, only one flag (either SUPRS_EXC or IMP_EXC) is set but never both together. The SUPRS_EXC flag is also recorded at ITYPE.

Note that a non-maskable interrupt (NMI) occurring in debug mode is treated as a precise/next-precise exception so it is recorded at SUPRS_EXC and updates VECTOR and ITYPE.ETYPE. NMI will not be treated as an imprecise exception even though it has similar non-maskable behavior.

Since the host (e.g., GDB) cannot access system registers directly, it is more efficient to create a flag in EDM rather than use instructions in the Debug Instruction Memory (DIM) to access system registers. Therefore, a new flag called ALL_SUPRS_EXC is added at Debug Event Register (DBGER) in EDM. This ALL_SUPRS_EXC flag represents the existence of any types of pending suppressed exception and is intended to be examined by the host in debugging mode. Consequently, its value is simply the logical-OR result of ITYPE.IMP_EXC and ITYPE.SUPRS_EXC; in other words, it means either an imprecise exception or a suppressed exception occurred. The ALL_SUPRS_EXC flag is recorded at Debug Event Register (DBGER) in the EDM Misc. Registers.

Furthermore, two operation modes are needed to give host-mode debugging maximum control while also allow target-mode debugging to be serviced by exception handlers. Therefore,

EDM_CTL.DEH_SEL is used to control whether or not exceptions are suppressed during debugging (0: do not suppress exceptions; 1: suppress exceptions).

When DEH_SEL is 0, the debugger runs from the interruption vector table and is considered as target-mode debugging. In this target mode, most debugging code (e.g., gdbserver for Linux) uses the virtual address so it requires the service of exception handlers to process page faults or TLB misses. As a result, exceptions should not be suppressed in this mode.

In contrast, when DEH_SEL is 1, the debugger runs from the Debug Instruction Memory and is considered as host-mode debugging. In this host mode, most debugging code uses the physical address and should not depend on exception handlers running on the CPU because they may not exist (e.g., in small MCU systems). In addition, the debugger needs full control and does not want to lose control due to some accidental exceptions. Therefore, exceptions are suppressed in this host-mode debugging.

4.8.6. Instruction Alignment Check

The Instruction Alignment Check exception is triggered for each instruction fetch that has a non-zero value at the bit 0 of the fetch address. When this exception happens, since the PC already has been updated before the instruction fetch, the IPC contains the mis-aligned instruction fetch address while the EVA has the address of the previous instruction that generates the mis-aligned instruction fetch address.

4.8.7. Branch Target Alignment Exception

In V3 architecture (MSC_CFG.BASEV is 2), a new “branch target alignment” exception is defined to improve debugging. This exception is issued when there is an unaligned branch (control flow change) target address. This exception belongs to the general exception vector entry point with the ITYPE.ETYPE set to 0 (same as the instruction alignment check exception) and the ITYPE.INST bit set to 1. This improved exception allows early detection at the branch instruction rather than late detection at the target instruction (where it is then difficult to identify which branch instruction has caused exception).

To facilitate debugging, the Exception Virtual Address Register (EVA) records the full 32-bit virtual address of the misaligned target when this exception occurs. This effectively eliminates the effort needed by the exception handler to calculate the illegal target address.

4.8.8. Reserved PTE Attribute

The Reserved PTE Attribute exception is generated when a valid PTE detected by hardware during load/store instructions or instruction fetching contains a reserved value in the “M” or “C” field of the PTE.

To clear/fix this exception, software is required to invalidate the incorrect, but valid, PTE in the TLB before inserting a new PTE that fixes the problem.

4.9. Internal Vector Interrupt Controller

The Internal Vector Interrupt Controller (IVIC) is a basic interrupt controller suitable for a simpler SoC platform not requiring a more sophisticated and bigger External Vector Interrupt Controller. In V3 architecture (MSC_CFG.BASEV is 2), the Internal Vector Interrupt Controller is enhanced with the following new features:

Extending the maximum hardware interrupt input sources to 32

For IVIC version 0 (IVB. IVIC_VER is 0), the maximum number of interrupt input sources for IVIC mode is extended from 6 to 16. The exact number of interrupt input sources is configurable and is recorded in IVB.NIVIC. To control these new interrupts, the Interruption Masking Register (ir14) and Interrupt Pending Register (ir15) are extended.

For IVIC version 1 and above (IVB.IVIC_VER >= 1), the maximum number of interrupt input sources for IVIC mode is further extended to 32. To control the new additional interrupt sources, several system registers are needed and described as follows. A 32-bit Interruption Masking Register 2 (ir26 or INT_MASK2) is added to control the mask bits of all interrupts. In addition, a 32-bit Interrupt Pending Register 2 (ir27 or INT_PEND2) with write-one-clear capability is added to control the pending bits of all interrupts. Lastly, a 32-bit Interrupt Priority Register 2 (ir28 or INT_PRI2) is added to control the priority level of interrupt sources 16 to 31. Although new full 32-bit registers (INT_MASK2 and INT_PEND2) are added for ease of programming, the lower 16 bits of these two registers are aliased and identical to the lower 16 bits of the older version (INT_MASK and INTE_PEND) to maintain backward compatibility. This allows old code to continue accessing old registers without changes; while new code to program entire 32 interrupts with just one single access to the new registers.

If more than 32 interrupts are needed, an external interrupt controller with EVIC mode should be used.

Removal of software interrupt entry point and its mask

For the software interrupt (i.e., interrupts triggered by SW using INT_PEND.SWI), the dedicated entry point and mask associated with it are removed in order to simplify the programming model and be consistent with EVIC mode. More specifically, just like EVIC mode,

the software interrupt pin must be routed outside of the CPU and be connected back to a hardware interrupt pin. In addition, the software interrupt mask bit (INIT_MASK.SIM) is removed and replaced by the mask bit of this hardware interrupt pin that connects to the software interrupt pin. The advantage of this approach is that the software interrupt can be assigned to any entry point and its priority can be arbitrarily adjusted using the priority level setting of the hardware interrupt.

Since the software entry point is removed, the hardware interrupt entry points become continuous and look like the following:

Table 13. Entry points for V3 architecture (MSC_CFG.BASEV is 2)

Entry number	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9	HW 0
10	HW 1
11	HW 2
12	HW 3
13	HW 4
14	HW 5
15	HW 6
16	HW 7
17	HW 8

Entry number	Entry point
18	HW 9
19	HW 10
20	HW 11
21	HW 12
22	HW 13
23	HW 14
24	HW 15
25	HW 16
26	HW 17
27	HW 18
28	HW 19
29	HW 20
30	HW 21
31	HW 22
32	HW 23
33	HW 24
34	HW 25
35	HW 26
36	HW 27
37	HW 28
38	HW 29
39	HW 30
40	HW 31

However, in the MCU family (e.g., V3m with MSC_CFG.MCU == 1), the software interrupt entry point and its mask are only removed in configurations with more than 6 interrupts (IVB.NIVIC > 6) but not in 2/6 interrupts. The reason is to be backward compatible and save gate count in the MCU family.

Adding backward compatible mode for interrupt controller

For backward compatibility with V2 architecture (MSC_CFG.BASEV is 1), a new hardware pin called V2_INT_MODE is added to configure interrupt controller to be exactly the same as V2 with software interrupt entry point. When this pin is set, it brings back the software interrupt entry point with its mask and reduces the total number of hardware interrupts to 6 (if it is more than 6); the software interrupt directly connects to the CPU internally and no longer needs to be routed outside of the CPU then connects back to a hardware interrupt pin. In other words, when V2_INT_MODE is set, the interrupt configuration and number in IVIC mode become exactly the same as V2 which is shown in the table below (the IVIC software interrupt entry point is fixed at 15). This pin is particularly useful when sharing the same IP instantiation among legacy and new projects.

Table 14. Entry points for V2 architecture compatible mode

Entry number	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9	HW 0
10	HW 1
11	HW 2
12	HW 3
13	HW 4
14	HW 5
15	SW 0

When V2_INT_MODE pin is set, hardware will change the settings of all interrupt related registers to be the same as V2 and make it indistinguishable to software. These changes include bringing back the software interrupt mask (INT_MASK.SIM) and changing the total interrupt number to 6 for the following registers: IVB.NIVIC, interrupt pending register, interrupt masking register, and interrupt priority register.

Please note that the MCU family (MSC_CFG.MCU is 1) does not implement this V2_INT_MODE pin for gate count saving purpose; instead, V3m only has fixed implementation per configuration. More specifically, V3m essentially implements “V2_INT_MODE pin always tie high” in 2/6 interrupts configurations and “V2_INT_MODE pin always tie low” in configurations with more than 6 interrupts (IVB.NIVIC > 6).
However, in order to be 100% compatible with V2, another mode bit called PPL2FIX_EN (at Interrupt Control Register or ir19) is added to allow execution of the old binary. When PPL2FIX_EN is set, it disables the comparison of the interrupt current priority level (PSW.CPL). This is because this CPL comparison only allows interrupts with priority higher than CPL to be serviced. In contrast, all interrupts essentially have the same priority in V2 architecture since V2 does not support programmable priority level. As a result, even with Global Interrupt Enable (GIE) set, the CPL comparison would still block all interrupts when an interrupt is being serviced. This makes it impossible to have nested interrupts. Therefore, this PPL2FIX_EN bit must be set to allow execution of V2 code with nested interrupts.

4.10. Support for Preemptive Interrupt with Programmable Priority

In V3 architecture (MSC_CFG.BASEV is 2), preemption for interrupts with programmable priority is supported. To support this feature, each hardware interrupt source is assigned a priority level. Based on this priority level, preemption ensures high priority interrupts get faster response. More specifically, if a high priority interrupt arrives, the CPU will automatically suspend the execution of the current low priority interrupt and begin executing this new high priority interrupt. The suspended low priority interrupt will resume execution only after the high priority interrupt is finished. In contrast, if an interrupt of same or lower priority arrives, preemption will not take place at all. Same or lower priority interrupts simply have to wait till the current interrupt is finished to take their turns.

The number of priority levels supported is different for Internal Vector Interrupt Controller (IVIC) mode and External Vector Interrupt Controller (EVIC) mode. For IVIC mode, each hardware interrupt source can be assigned 4 different priority levels ranging from 0 (highest) to 3 (lowest). The priority of each interrupt is specified at the Interrupt Priority Register (ir18). For EVIC mode, the CPU can support up to 7 priority levels ranging from 0 (highest) to 6 (lowest) depending on the setting of the interrupt controller.

To support preemptive interrupts, a new field called “Current Priority Level” (CPL) is added in the Program Status Word Register (PSW). This CPL is set automatically by hardware when an interrupt is accepted or completed. At reset or when there is no interrupt (user mode), the CPL is set to 7 (lowest priority level). When an interrupt is accepted, CPL is set differently depending on IVIC or EVIC mode. For IVIC mode, CPL is copied from the Interrupt Priority Register (ir18) according to the interrupt number. On the other hand, for EVIC mode, CPL is copied from the Interrupt Priority Level (IPL) interface when an interrupt is accepted. When an interrupt is completed (upon IRET), CPL is restored to its original value before this interrupt occurred. Even in the case of nested interrupts, this restoration of CPL value occurs automatically since CPL is stored at PSW and hence it is backed up by the interruption stack. Also note that the CPL field is writable so it can be modified by software if needed.

When the Global Interrupt Enable (PSW.GIE) bit is enabled, CPL is used to compare with the priority levels of all active interrupts. The active interrupt with the highest priority (must be higher than CPL) is picked and serviced by the CPU. In case there are multiple interrupts with same priority, the CPU will pick the interrupt with the smallest interrupt number (numerical value). In addition, when the software interrupt exists in V2 compatible mode (INT_CTRL.PPL2FIX_EN = 1), the software interrupt is always picked last (after all hardware interrupts).

Preemptive interrupts are also supported for the External Vector Interrupt Controller (EVIC) mode. The external interrupt controller must send an additional 3-bit Interrupt Priority Level (IPL) to the CPU along with each interrupt request. When the IPL of an interrupt is received, the CPU first compares this IPL with CPL of the current interrupt stored at PSW. If the IPL has higher priority than CPL, preemption occurs by suspending the current interrupt and starting to service this new high priority interrupt. Otherwise, if this interrupt has same or lower priority, it is ignored and left pending.

If an interrupt is accepted by the CPU, its IPL is copied into CPL to keep track of the current priority. In addition, this CPL is also sent out to the external interrupt controller for potential performance optimization (e.g., selecting the most appropriate interrupt).

This feature is both configurable statically and controllable at run-time. Accordingly, a configuration bit is added to indicate whether interrupts have programmable or a fixed priority. This bit is called PROG_PRI_LVL and is recorded at Interruption Vector Base Register (IVB). In addition, for run-time control, a mode bit called PPL2FIX_EN is added and it is recorded at Interrupt Control Register (ir19). This mode bit is used for backward compatibility to run interrupts in old V2 architecture (MSC_CFG.BASEV is 1) style which only has fixed priority. When this bit is set, it disables the comparison of the interrupt priority level and allows all interrupts (with any priority) to preempt as long as Global Interrupt Enable (GIE) is set.

Lastly, to best utilize this feature, an interrupt handler should enable interrupt (set GIE) as soon as it finishes critical work to allow preemption and hence let higher priority (more critical) interrupts to be serviced first.

4.11. Support for 8-byte Stack Pointer Alignment

In V3 architecture (MSC_CFG.BASEV is 2), a hardware feature is added to automatically perform 8-byte alignment for the stack pointer when interrupts and exceptions occur. For this 8-byte alignment, hardware only needs to adjust the stack pointer on entering and leaving interrupts/exceptions. More specifically, hardware performs the following steps:

- When an exception occurs, HW can easily access and use bit 2 of stack pointer ($sp[2]) as a flag to decide whether adjustment is needed for $sp.
- Also, HW saves this $sp[2] bit at IPSW to adjust $sp when later leaving exception.
- If ($sp[2] == 0): no adjustment is needed.
- Else ($sp[2] == 1): adjustment is needed by clearing $sp[2] to 0 (equivalently as performing $sp - 4).

The adjustment result (namely $sp[2] bit) is recorded at a flag called SP_ADJ which is stored at IPSW. This flag is needed upon the exit of the interrupt (i.e., IRET) to restore the stack pointer back to its original value.

Please note that this 8-byte stack pointer alignment will not be performed for exceptions at the maximum interruption stack level. This is because there is no corresponding IPSW to store the SP_ADJ flag in the maximum cases. To make up for this, all exceptions at the maximum level are redirected to a dedicated entry point (the machine error), where the corresponding software exception handler has to check the interruption stack level and then perform 8-byte alignment as needed.

Similarly, the 8-byte stack pointer alignment will not be performed for debug exceptions. This is because debug exceptions cannot modify interruption registers and thus there is no place to store the SP_ADJ flag.

4.12. Support for Shadow Stack Pointer in Privileged Mode

In V3 architecture (MSC_CFG.BASEV is 2), for safety reasons, a shadow stack pointer is added in privileged mode to prevent user code from corrupting the stack pointer of privileged code by accident. In addition, this shadow stack pointer improves the latency of interrupts and exceptions.

- The stack pointer is shadowed in privileged mode.
- Both copies of stack pointers are mapped to system registers: SP_USR (ir16) and SP_PRIV (ir17).
- The selection of stack pointers is determined by PSR.POM.

Please note that SP_USR (ir16) is the current GPR "R31", so it does not need storage but just extra decoding logic to map to ir16. On the other hand, SP_PRIV (ir17) is a new 32-bit register with storage.

A configuration field called "SHADOW" is added in the Miscellaneous Configuration Register (cr4 or MSC_CFG) to indicate whether the shadow register feature is supported and what types of shadow registers are supported.

For backward compatibility reasons, a control bit is added to turn off this feature and it is located in MISC_CTL.SP_SHADOW_EN.

Field Name	Bits	Description	Type	Reset
SP_SHADOW_EN	1 (4)	Enable control for shadow stack pointers. 0: Shadow stack pointers are not enabled (for backward compatibility). 1: Shadow stack pointers are enabled.	RW	0

The exact equation to select between the two stack pointers is summarized as below. Note that the privileged stack pointer is always used when the maximum interruption stack level is reached.

```
If (SHADOW_EN == 1'b0) {
   R31 = SP_USR
}
else {
          if (INTL = MAX_INTL) {//enforce superuser mode
              R31 = SP_PRIV
          }
          else {//depend on PSW.POM
              R31 = (PSW.POM == 1) ? SP_PRIV : SP_USR
          }
}
```

Also note that accesses to these shadowed stack pointers through MTSR and MFSR require adding DSB instructions to ensure correctness. More specifically, DSB must be added in the following two cases to guarantee latest data takes effect or is used:

- MTSR SP_USR/SP_PRIV + DSB
- After modifying R31 + DSB + MFSR SP_USR/SP_PRIV

As an aside, the debug exception does not change the mapping of R31 since it does not change PSW.POM.

4.13. Support for Edge-triggered Interrupt

Edge-triggered interrupts are supported in new versions of the Internal Vector Interrupt Controller (when IVB.IVIC_VER >= 1). Only positive edge-triggered interrupts are supported. If negative edge-triggered interrupts are needed, inverters should be added on the interrupt input externally. In addition, any edge-triggered interrupt must assert the request signal longer than one CPU-clock period to guarantee successful sampling by the CPU.

To support edge-triggered interrupts, several system registers are needed. An interrupt controller version field (IVIC_VER) is added at Interrupt Vector Base Register (ir3) to indicate whether edge-triggered interrupts are supported. In addition, an Interrupt Trigger Type Register (ir29) is added to specify whether each interrupt is level-triggered or edge-triggered. Lastly, an Interrupt Pending Register 2 (ir27) is added with write-one-clear capability to allow clearance of pending status.

To speed up interrupt response, hardware automatically clears the pending bit of an edge-triggered interrupt. More specifically, hardware clears the pending bit at the very beginning of service for this interrupt. With this hardware feature, software does not need to spend any time to clear pending bits in the interrupt service routine (ISR) so interrupt latency is improved. By automatically clearing the pending bit upon entering ISR, it has another advantage that the pending bit becomes available early to record the same interrupt even if it occurs again during ISR. Also note that this clearing of pending bits only applies to edge-triggered interrupts but not level-triggered ones. This is because the pending bits of level-triggered interrupts must be cleared from external sources (e.g., devices) instead of internal pending registers.

Although software does not need to clear pending bits during regular operation, it may need to clear pending bits when initializing or reprogramming the interrupt controller and interrupt source devices. This clearance is needed since pending bits can be accidentally set by glitches or noise before proper initialization. To ensure a clean start, software can explicitly clear pending bits using the write-one-clear capability in the Interrupt Pending Register 2 (ir27).

4.14. Handling of Imprecise and Next-Precise Exceptions in Interruption Architecture Version 1

Starting from the Interruption Architecture Version 1 (MSC_CFG.INTV is equal to or greater than 1), the handling of imprecise and multiple simultaneously-generated Next-Precise (NP) exceptions are improved to avoid unexpected corruption of the interruption stack and enforce orderly processing of each Next-Precise exception.

Starting from the version 1, an imprecise exception is treated like an interrupt with the highest priority and PSW.GIE is used to determine if an imprecise exception can be selected by the interruption mechanism or not. Thus, an asynchronous imprecise exception can only be taken when software has prepared enough interruption stack resources and indicated that information to hardware by setting the state of the PSW.GIE bit to 1. With this adjusted behavior of an imprecise exception and its lower-than-NP-exception priority in the interruption priority table, it is NOT possible to select an imprecise exception as the current interruption handler while leaving several Next-Precise exceptions pending for processing.

For Next-Precise exceptions, occasionally multiple NP exceptions may occur simultaneously and, in this case, we want to process them one-by-one in a controlled order. For example, a stack overflow NP exception and a watch-point (debug) NP exception can both occur at the same time. However, note that “Hardware Single-Stepping” (HSS) is not treated as an NP exception, even though it is triggered by an NP event of instruction commitment in hardware implementations.

4.14.1. Imprecise Exception

An imprecise exception is generated after the completion of the instruction triggering this exception, and the delay between the exception and generating instruction is unpredictable. Therefore, an imprecise exception can be treated as an asynchronous event since it is unknown which instruction or even which process will be interrupted by this imprecise exception. However, to provide proper protection for interruption stack at high levels from interfering imprecise exceptions, an imprecise exception is treated as an interrupt with the highest priority and its processing is controlled by the PSW.GIE state. Only when PSW.GIE is enabled, an

imprecise exception can then be selected for processing. In addition, at the maximum interruption level, interrupt is disabled by default and this behavior includes the imprecise exceptions. With the control of PSW.GIE, a group of successive imprecise exceptions will not interfere with any exception or overflow/corrupt the interruption stack unexpectedly.

4.14.2. Orderly Processing of Next-Precise Exceptions in 1st Level Code

The main goal is to process simultaneously-generated NP exceptions one after another in an orderly fashion only in the 1st level of code that generates this condition. In a very rare and unfortunate case, if another NP exception occurs again in the 2nd level of code (i.e., during the handler processing the 1st level NP exception), the processing of the latter exception will be either dropped or be deferred to the 1st level of code. This assumption allows implementations to use minimum hardware logic to handle multiple NP exceptions in a proper order.

A Pending Next-Precise (PNP) bit at the PSW register is allocated to record whether there are more NP exceptions pending after picking one of the multiple NP exceptions for processing. It serves as a global state to temporarily mask out the processing of the pending NP exceptions. This bit remains set upon interruption, so the pending NP exceptions will be masked out until the CPU returns back to the 1st level of code.

When the current NP exception handler is finished and executes an “IRET” instruction, the PNP bit will become 0 again (updated from IPSW.PNP) and the pending NP exceptions will start to participate in the interruption arbitration again.

4.14.3. Pending NP Exception Masking Behavior at the Maximum Interruption Level

To simplify design, at the maximum interruption level (which is a severe machine error state), PSW.PNP state will continue to determine the masking behavior of any pending NP exception, although PSW.PNP state will not be updated when a CPU enters the maximum interruption level. There is no default behavior defined for this extreme corner case at the maximum interruption level.

Note that this implies an irregular behavior if PSW.PNP happens to be 0 and multiple next-precise exceptions occur at the (max-1) interruption level causing the interruption level to raise to the maximum. In this case (without the setting and protection of PSW.PNP), the hardware selects the first NP exception for processing, but the remaining NP exceptions will continue to barge in and be selected for processing even before the execution of the first instruction in the first NP exception handler.

4.14.4. Terminating or Processing Pending NP Exceptions in an NP Exception Handler

Additional system register states are allocated to record the pending status of each NP exception. As an example, a state bit is defined to record NP watch-point exception (INT_PEND.NP_WPP) since NP watch-point exception can become pending if occurring at the same time with a stack overflow NP exception.

If a situation arises that the earlier picked NP exception handler wants to process or terminate other pending NP exceptions, it can do so by checking or clearing the pending NP exception states. A typical use case for terminating pending NP exceptions is when users want to kill all current processes and restart fresh.

5. Hardware Stack Protection and Recording

This chapter describes the specification of AndeStar™ Hardware Stack Protection and Recording Mechanism. There are two operating schemes in this mechanism. The hardware stack protection scheme helps customers protect against the stack overflow/underflow problem caused by insufficient allocation of stack memory or program errors. On the other hand, the hardware top of stack recording scheme helps customers find the required stack memory size for their applications during runtime.

To simplify the design, this hardware stack protection and recording mechanism adopts the following two assumptions.

- AndeStar™ programs should grow their stack space from high memory address to low memory address.
- Only one of the two schemes described above can be used at any time.

To implement this mechanism, three privileged system registers are required: one control register (HSP_CTL), one stack bound register (SP_BOUND), and one stack base register (SP_BASE). For an AndesCore with a shadowed privileged stack pointer (SP_PRIV), a shadowed stack bound register (SP_BOUND_PRIV) and a shadowed stack base register (SP_BASE_PRIV) are also required.

The registers in this mechanism are all privileged registers that can only be accessed by superuser mode with MTSR/MFSR instructions.

The availability of this mechanism is indicated at bit 27 of the Misc. Configuration Register (MSC_CFG).

5.1. Stack Overflow/Underflow Detection

The stack overflow protection scheme detects stack overflow by comparing the updated value to the stack pointer (SP) register (via any instruction) with the value of the stack bound register. If the updated value to the SP register is smaller than value of the stack bound register, a next-precise stack overflow exception is generated. The next-precise stack overflow exception belongs to the "General Exception" entry point with ETYPE equal to 11 and INST equal to 0.

The stack underflow protection scheme detects stack underflow by comparing the updated value to the stack pointer (SP) register (via any instruction) with the value of the stack base register. If the updated value to the SP register is larger than the value of the stack base register, a next-precise stack underflow exception is generated. The next-precise stack underflow exception belongs to the "General Exception" entry point with ETYPE equal to 11 and INST equal to 1.

When a next-precise stack overflow/underflow exception is taken, the hardware will first adjust the stack pointer to 8-byte alignment and the exception handler needs to save at least one register state to the memory stack before doing any processing. To prevent generating further overflow/underflow exceptions during the above stack pointer adjustment, the stack protection mechanism will be disabled automatically by turning off the enable bits (HSP_EN and UDF_EN) when the exception is taken. To re-enable the stack protection mechanism, software must turn on both enable bits again before executing the interruption return (IRET) instruction. Note that if a next-precise stack overflow/underflow exception is suppressed, the stack protection mechanism will not be disabled.

5.2. Top of Stack Recording

The top of stack recording scheme records the top of stack boundary by comparing the updated value to the SP register (via any instruction) with the value of the stack bound register. If the updated value to the SP register is smaller than the value of the stack bound register, the stack bound register is revised with the updated value in SP.

5.3. Control of the Mechanism

Before enabling this mechanism, one should first select the operating scheme and the usage mode of the mechanism. The "usage mode(s)" include a superuser mode bit and a user mode bit, so the mechanism can be used in user mode only, used in superuser mode only, or used in both modes.

To use the mechanism properly, one can initialize the stack pointer register, the SP_BOUND and the SP_BASE system registers before enabling the mechanism. When the mechanism is already in operation, if one wants to switch to another task stack, one should change or initialize the SP_BOUND and the SP_BASE registers first before changing the stack pointer register. Following this update sequence avoids an unexpected Stack Overflow/Underflow Exception in the stack range protection scheme and makes sure a proper setup in the top of stack recording scheme. This update sequence works because the SP_BOUND/SP_BASE registers are only compared on a stack pointer register update, rather than being compared all the time. Example codes are provided in Figure 7 and Figure 8 for the two schemes in this mechanism. Note that a DSB instruction is required after the SP_BOUND register update "MTSR" instruction to make sure that when the SP register is updated in the next instruction, the stack protection and recording mechanism will use the latest SP_BOUND register value to compare with the updated SP value.

```
lwi r3, [r4 + SP_BOUND_OFFSET];   // load new SP_BOUND value.
mtsr r3, SP_BOUND;                // update SP_BOUND register.
lwi r3, [r4 + SP_BASE_OFFSET];    // load new SP_BASE value.
mtsr r3, SP_BASE;                 // update SP_BASE register.
dsb;                              // make sure SP_BOUND is updated.
lwi sp, [r4 + SP_OFFSET];         // update SP with new value.
```

Figure 7. Example code for changing SP_BOUND/SP_BASE when the stack range protection is ON

```
sethi r3, 0xfffff;                  // set a large value.
mtsr r3, SP_BOUND;                  // initialize SP_BOUND register.
dsb;                                // make sure SP_BOUND is updated.
lwi sp, [r4 + SP_OFFSET];           // update SP with new value.
```


Figure 8. Example code for changing SP_BOUND when the top of stack recording is ON

When you want to read the SP_BOUND register value at the end of your program under the top of stack recording scheme to calculate the maximum stack size, you should add a "DSB" instruction in front of the "MFSR" instruction reading the SP_BOUND register value to make sure the "MFSR" instruction can get the latest update for the SP_BOUND register. Figure 9 shows the example code for this case.

```
addi10.sp #-12;             // some SP update.
dsb;                        // serialize SP update & SP_BOUND read.
mfsr r3, SP_BOUND;          // read SP_BOUND.
```

Figure 9. Example code for reading SP_BOUND under the top of stack recording scheme

5.4. Special Handling in 24-bit Reduced Address Space Configuration

In the 24-bit Reduced Address Space configuration, the SP_BOUND and the SP_BASE registers are reduced to 24 bits. When moving the content of these registers into a 32-bit register, the top 8 bits (bit 31 to bit 24) will be filled with 0. When moving the content of a 32-bit register into these registers, only the lower 24-bit value (bit 23 to bit 0) will be moved into them.

In this configuration, the stack protection comparison logic only uses the lower 24-bit of the stack pointer (SP) register to compare with the SP_BOUND and the SP_BASE registers, assuming the upper 8 bits of the SP register are all 0.

If one of the upper 8 bits of the SP register is not 0, then a next-precise stack overflow/underflow exception will also be generated and the SP_BOUND register will not be updated with the incorrect SP value under the Top of Stack Recording scheme.

5.5. Shadowed SP and SP_BOUND Register

When a CPU is configured with a shadowed SP, this mechanism should be configured with shadowed SP_BOUND/SP_BASE registers as well. Under this shadowed SP configuration, the hardware stack protection and recording mechanism should compare the updated value of the "current SP" with the value of the corresponding (matching) "current SP_BOUND/SP_BASE" registers. The "current SP" and "current SP_BOUND/SP_BASE" registers are determined as follows:

For "current SP_BOUND/SP_BASE":

- The "current SP_BOUND/SP_BASE" are entirely determined by the "current SP," and they should always be in identical mode. Specifically, if the "current SP" is the SP_USER register (user mode), then the SP_BOUND/SP_BASE registers (also user mode) should be used; if the "current SP" is the SP_PRIV/SP_BASE_PRIV register (superuser mode), then SP_BOUND_PRIV/SP_BASE_PRIV (also superuser mode) should be used.

For "current SP":

- When MISC_CTL.SP_SHADOW_EN==0, the "current SP" is always the SP_USER register.
- When MISC_CTL.SP_SHADOW_EN==1:
  - For host-mode debug exception (EDM_CTL.DEH_SEL==1 and PSW.DEX==1), if not overruled by the debugger, "current SP" is determined by the operating mode of the program being debugged. More specifically, if not overruled by the debugger (EDM_CTL.DEX_USE_PSW==1 and PSW.POM==0), the "current SP" is the SP_PRIV register when PSW.POM==1 or the maximum stack level is reached. Otherwise, the "current SP" is the SP_USER register (user mode).

  - For all other cases, the "current SP" is determined by the current operating mode. In other words, the "current SP" is the SP_USER register in user mode and is the SP_PRIV register in superuser mode.

The above definitions assume a host-mode debugger does not use the shadowed superuser stack to save or restore any register values and only cares about the register values of the debugged program. So the “current SP” and “current SP_BOUND/SP_BASE” register definitions are constructed to simplify the access of the SP register of the debugged program for a host-mode debugger.

Based on the above definition of the “current SP” and “SP_BOUND/SP_BASE” registers, updating all other shadow stack pointers that are not the “current SP” will NOT generate stack overflow/underflow exceptions or affect the SP_BOUND register. In other words, only updating the “current SP” (the stack pointer currently being used) to values smaller than the “current SP_BOUND” or to values larger than the “current SP_BASE” will generate stack overflow/underflow exceptions or update the SP_BOUND.

5.6. Next-precise Stack Overflow/Underflow Exception

5.6.1. Exception Entry Point

The next-precise Stack Overflow/Underflow Exception belongs to General Exception entry point with ETYPE equal to 11.

5.6.2. Nested Exceptions Caused by Hardware

In some rare cases, a nested stack overflow/underflow exception might be triggered by hardware while processing an older exception or interrupt. This occurs when hardware automatically aligns the stack pointer to 8-byte boundary while processing the older exception/interrupt. If the stack pointer happens to be within 8 bytes from the bound or base, the hardware alignment may trigger a stack overflow/underflow exception. In this case, even though the nested exception is caused by hardware (instead of a software instruction), the CPU will treat it just like a normal nested exception case. Namely, it will record the first instruction at the entry point of the older exception/interrupt (by pushing it to IPC or OIPC) and then redirect the PC to the stack overflow/underflow exception handler.

5.6.3. Disabling of Further Stack Overflow/Underflow Exceptions

When a next-precise stack overflow/underflow exception is taken, the hardware will first adjust the stack pointer to 8-byte alignment and the exception handler needs to save at least one register state to the memory stack before doing any processing. To prevent generating further overflow/underflow exceptions during the above stack pointer adjustment, the stack protection mechanism will be disabled automatically by turning off the enable bits (HSP_EN and UDF_EN) when an exception is taken. To re-enable the stack protection mechanism, software must turn on both enable bits again before executing the interruption return (IRET) instruction.

5.6.4. Implementation Notes

AndesCore users can skip this section as it contains implementation details.

5.6.4.1 Interaction Between an Interrupt and a Stack Overflow/Underflow Exception Detected on Load/Store Multiple Instruction

A load/store multiple instruction is interruptible. If a next-precise stack overflow/underflow exception is detected on any micro-operation of a load/store multiple instruction, the next-precise stack overflow/underflow exception will be taken after all of the remaining micro-operations of a load/store multiple instruction complete their operations. Then, when both events happen, what will be the interactions between them?

If an interrupt happens before the detection of any next-precise stack overflow/underflow exception during the execution of a load/store multiple instruction, the interrupt will be taken immediately.

However, if a next-precise stack overflow/underflow exception is detected on any micro-operation of a load/store multiple instruction before any incoming interrupt, then the incoming interrupt will not be taken until the load/store multiple instruction has completed and the CPU has entered the exception handler of the stack overflow/underflow exception.

5.6.4.2 Stack Overflow/Underflow Exception and Next-Precise Data Address/Value Watchpoint on Load/Store Multiple Instruction

When a next-precise stack overflow/underflow exception and a next-precise data address/value watchpoint exception happen simultaneously on a load/store multiple instruction, the stack overflow/underflow exception will be taken first due to its higher exception priority. The simultaneously-happened lower priority exceptions should be taken later after the stack overflow/underflow exception processing is complete.

6. Performance Monitoring

This chapter describes performance monitoring and contains the following section:

The AndesCore™ provides performance monitoring feature for programmers to obtain program running statistics and performance data. These performance data will help the programmers to understand the behaviors of their programs so that they can easily pin-point the performance bottlenecks in their code and continuously tune the code to remove those bottlenecks.

The performance monitoring feature consists of several 32-bit performance monitoring counters and their associated control registers. The exact number of counters is implementation-dependent and specified in the bit 31 of the Performance Counter Control Register (pfr3). However, at least one cycle/instruction counter is always available to provide basic performance monitoring functionalities.

The performance counters and their controls provide the following performance monitoring features:

- One cycle/instruction counter.
- Two optional event counters.
- Individual controls:
  - Counter enable.
  - Interrupt enable.
  - Overflow status/clear bit.
  - Counting event selection among the maximum of 64 events.
  - Superuser and user mode event counting filters.
- Readable/Writable in privileged mode.
- Continuing counting after overflow.

The performance monitoring related registers are all privileged resources and are described in detail in Section 10.7. A summary of these registers is listed in Table 15.

Table 15. Performance monitoring related registers

Name	Type	Updater	Section
Performance Counters	RW	SW/HW	10.7.1 10.7.2
Performance Monitoring Control	RW	SW/HW	10.7.3
Performance Throttling Control	RW	SW	10.7.4

6.1. Interrupt Interface

When any of the performance monitoring counters overflows and the interrupt enable control is set, an interrupt is generated. This interrupt is exported out of the core through an interface signal, PM_INT. This signal can then be connected with the internal or external interrupt controller. This signal can be synchronous or asynchronous, depending on the implementation.

Note that if software wants to know which counter generated an interrupt, it has to examine individual overflow bit to get this information.

7. Local Memory

In addition to caches, Andes architecture provides support for fast-accessed low latency local memories in the AndesCore™. The local memories are divided into instruction local memory (ILM) and data local memory (DLM) for increased performance and design simplicity. This chapter describes the detail of local memory and contains the following sections:

7.1. Local Memory Base Address

There are various versions and behavior for the base physical address alignment of the Andes local memory. They are indicated in the BSAV field of the ICM_CFG and DCM_CFG system registers respectively.

When the BSAV field is greater than or equal to 1, the base physical address of the Andes local memory is aligned to a power-of-2 byte boundary that is a multiple of the local memory size. In this configuration, there is no “nonexistent memory address” exception regardless of whether the *Double-Buffer* mode is disabled or enabled. Any access outside the local memory address range (physical DLM size if *Double-Buffer* mode is disabled or half of the physical DLM size if *Double-Buffer* mode is enabled) will access the other parts of the system memory.

If the instruction local memory base address is programmed to be the same as the data local memory base address, UPREDICTABLE behavior will happen to the registers and the local memory content.

To speed up the access time, the local memory is given the first priority and thus is always selected when there is a hit. As a result, accesses falling inside the address range of the local memory only go to the local memory (and not go out to caches, buses or devices). In other words, if the address range of the local memory overlaps with other memories, only the local memory can be accessed while others are ignored (shadowed) completely.

7.2. Data Local Memory Access Modes

To provide different data processing models for different applications, the AndesCore data local memory has two different access modes for the processor pipeline and the DLM DMA engine. The first mode is the normal access mode where the DLM address range, seen by the processor pipeline and the DMA engine, covers the whole DLM size. In this mode at any moment, the processor core and the DMA engine will see the same DLM address space and it is possible for them to access the same DLM location at the same time using the same address. Please note that the performance of the DLM data access may be degraded if there are frequent simultaneous DLM accesses from the core pipeline and the DMA engine. This performance characteristic of DLM access is implementation-dependent.

The second mode is the *Double-Buffer* access mode where the DLM is divided into two banks and the processor pipeline is directed to access one bank while the DMA engine is directed to access the other bank. In this mode, the address range is only half of the physical DLM size when accessed from the processor pipeline and the DMA engine; meanwhile, the actual memory location accessed by this address range is different for the processor pipeline and the DMA engine. If they access DLM using the same address at the same time, they will access different memory locations simultaneously without any contention. This arrangement is good for the processor pipeline and DMA engine doing their own work without interfering with each other. However, to facilitate sharing and coarse-grained task-level pipelining, the accessed bank should be switched between the processor pipeline and the DMA engine. As a result, the data prepared by the DMA engine can be used by the processor pipeline after the bank switch; and vice versa. The bank switch in this mode is controlled by the DBB field in the DLMB register.

Before switching DLM configuration between the double-buffer mode and non-double-buffer mode, a “MSYNC” instruction must be added to make sure all previous store instructions have completed their accesses to the DLM. Otherwise, a transition between the double buffer mode and the normal mode may cause UNPREDICTABLE data lost in the data local memory content. In addition, software should re-initialize the data local memory after changing the access mode.

7.3. Local Memory Address Range

The local memory address ranges are listed in the following tables. LMB_BPA represents the based address field of the ILM and DLM local memory base address system registers (ILMB.IBPA in Section 10.4.7 and DLMB.DBPA in Section 10.4.8). The "." serves as a bit concatenation operator. 0(m,n) and 1(m,n) represent a replication of bit 0 or bit 1 in the bit range specified. The 1-bit range representations of 0(m,m) and 1(m,m) can be simplified to 0(m) and 1(m). For example, to set up an instruction local memory of size 4KB starting at address 12KB (0x3000), we simply program ILMB.IBPA to 0xC (take the upper 22 bits of 0x3000).

If the base address programmed is not a multiple of local memory size (an illegal address), the hardware will automatically mask out the illegal least-significant bits and only use the valid portion to access local memory to prevent errors. Nevertheless, the address in the base register will be processed differently depending on whether internal or external local memory is used. For internal local memory, the base register will get a new address which has the illegal least-significant bits masked out by hardware. In contrast, for external local memory, the base register preserves the original illegal base address programmed without masking.

When xCM_CFG.BSAV == 1 (x = I or D), the starting address must be multiples of 1 KB:

- Non-Double-buffer mode (for ILM and DLM):

LM Size	Start	End
1KB	LMB_BPA(31,10).0(9,0)	LMB_BPA(31,10).1(9,0)
2KB	LMB_BPA(31,11).0(10,0)	LMB_BPA(31,11).1(10,0)
4KB	LMB_BPA(31,12).0(11,0)	LMB_BPA(31,12).1(11,0)
8KB	LMB_BPA(31,13).0(12,0)	LMB_BPA(31,13).1(12,0)
16KB	LMB_BPA(31,14).0(13,0)	LMB_BPA(31,14).1(13,0)
32KB	LMB_BPA(31,15).0(14,0)	LMB_BPA(31,15).1(14,0)
64KB	LMB_BPA(31,16).0(15,0)	LMB_BPA(31,16).1(15,0)
128KB	LMB_BPA(31,17).0(16,0)	LMB_BPA(31,17).1(16,0)
256KB	LMB_BPA(31,18).0(17,0)	LMB_BPA(31,18).1(17,0)
512KB	LMB_BPA(31,19).0(18,0)	LMB_BPA(31,19).1(18,0)

LM Size	Start	End
1024KB	LMB_BPA(31,20).0(19,0)	LMB_BPA(31,20).1(19,0)
2048KB	LMB_BPA(31,21).0(20,0)	LMB_BPA(31,21).1(20,0)
4096KB	LMB_BPA(31,22).0(21,0)	LMB_BPA(31,22).1(21,0)
8192KB	LMB_BPA(31,23).0(22,0)	LMB_BPA(31,23).1(22,0)
16384KB	LMB_BPA(31,24).0(23,0)	LMB_BPA(31,24).1(23,0)

- Double-buffer mode (for DLM only):

LM Size	Start	End
1KB	LMB_BPA(31,10).0(9,0)	LMB_BPA(31,10).0(9).1(8,0)
2KB	LMB_BPA(31,11).0(10,0)	LMB_BPA(31,11).0(10).1(9,0)
4KB	LMB_BPA(31,12).0(11,0)	LMB_BPA(31,12).0(11).1(10,0)
8KB	LMB_BPA(31,13).0(12,0)	LMB_BPA(31,13).0(12).1(11,0)
16KB	LMB_BPA(31,14).0(13,0)	LMB_BPA(31,14).0(13).1(12,0)
32KB	LMB_BPA(31,15).0(14,0)	LMB_BPA(31,15).0(14).1(13,0)
64KB	LMB_BPA(31,16).0(15,0)	LMB_BPA(31,16).0(15).1(14,0)
128KB	LMB_BPA(31,17).0(16,0)	LMB_BPA(31,17).0(16).1(15,0)
256KB	LMB_BPA(31,18).0(17,0)	LMB_BPA(31,18).0(17).1(16,0)
512KB	LMB_BPA(31,19).0(18,0)	LMB_BPA(31,19).0(18).1(17,0)
1024KB	LMB_BPA(31,20).0(19,0)	LMB_BPA(31,20).0(19).1(18,0)
2048KB	LMB_BPA(31,21).0(20,0)	LMB_BPA(31,21).0(20).1(19,0)
4096KB	LMB_BPA(31,22).0(21,0)	LMB_BPA(31,22).0(21).1(20,0)
8192KB	LMB_BPA(31,23).0(22,0)	LMB_BPA(31,23).0(22).1(21,0)
16384KB	LMB_BPA(31,24).0(23,0)	LMB_BPA(31,24).0(23).1(22,0)

7.4. Optional Feature to Boot from ILM

An optional feature exists in some CPUs to allow booting directly from Instruction Local Memory (ILM). As for the availability of this feature, please check the datasheet.

This "boot from ILM" feature is turned on by asserting the "ilm_boot" pin to 1 before reset. If this feature is on, the CPU automatically performs the following 3 tasks:

- Enables the ILM by setting the reset value of ILMB.IEN to 1.
- Initializes the base address of ILM by setting its reset value to the base address of the interruption vector table (i.e., ILMB.IBPA[31:16] = IVB.IVBASE[31:16]).
- Boot (fetch the first instruction) from ILM directly instead of booting from system memory bus.

This feature is very useful for users who allocate the reset code in ILM (containing some flash/ROM) and want to boot from there.

7.5. Unified Local Memory (ULM)

Unified Local Memory (ULM) configuration adds the flexibility to freely mix and place instruction and data in either Instruction Local Memory (ILM) or Data Local Memory (DLM). In particular, this configuration allows instructions to be placed in DLM (which is not allowed in non-ULM configuration) and offers the same fixed minimal access time to both ILM/DLM. It provides the capability for users to balance the storage of instruction and data in different local memories. Consequently, both Instruction and Data Local Memories can be viewed as one "unified" local memory and hence derives its name. The support for Unified Local Memory is indicated with a configuration bit recorded at MSC_CFG.ULM. When MSC_CFG.ULM is 1, the behavior of both ILM and DLM changes according to the specification of ULM described in this section.

In Unified Local Memory configuration, either local memory (ILM or DLM) can be optionally split into two banks to double the number of ports. This two-bank configuration allows instruction and data be placed in the same local memory while still accessed by the CPU in parallel. More specifically, if instruction and data are placed in different banks, the CPU can fetch instruction from one bank while access data from the other bank simultaneously. Another typical usage of two banks is to allow simultaneous access by the CPU and DMA, provided that each accesses a different bank in local memory. Note that, for fast access time, the two banks must be the same size which is half the size of the local memory. The two-bank configuration is specified in ICM_CFG.ULM_2BANK for ILM and DCM_CFG.ULM_2BANK for DLM respectively.

To speed up access time, a special mapping rule is imposed on the base addresses of ILM (ILMB.IBPA) and DLM (DLMB.DBPA). This rule allows hardware to fast detect whether a request belongs to ILM or DLM using only one single address bit. The principle for this rule is to select the larger size between ILM and DLM. Then use this size as a basic unit to divide the entire address space into even and odd units. The base addresses of ILM and DLM must be mapped to units with opposite parity (one must be even while the other must be odd). In other words, if ILM is mapped to even units then DLM must be mapped to odd units, or vice versa.

To illustrate this mapping rule, an example is shown here. If the ILM is 1KB and DLM is 2KB in size, then this rule can be described as the following steps:

1. First, select the size of the larger local memory (between ILM and DLM).
   - In this example, select the larger between 1KB and 2KB. So the selected size is 2KB.

2. Since the size must always be power of two, only one single bit (most significant bit) is "one" for the selected size. The bit position of this "one" is denoted as "p".
   - 2KB is 2^11, or equals to "1000,0000,0000" in binary form. Only the bit at position 11 is non-zero, so "p" is 11.

3. The base addresses of ILM and DLM must be assigned opposite binary values at the bit position "p".
   - At bit position 11 ("p"), if ILM base address is assigned to "0" (even), then DLM must be assigned to "1" (odd). In other words, ILM can be assigned to even multiples of 2KB, which are: 0 (0*2KB), 4KB (2*2KB), 8KB (4*2KB), 12KB (6*2KB), 16KB (8*2KB),... While DLM can be assigned to odd multiples of 2KB, which are: 2KB (1*2KB), 6KB (3*2KB), 10KB (5*2KB), 14KB (7*2KB),...

To summarize, this mapping rule is simply: assign different binary values to the base addresses of ILM and DLM at the bit position "p", where "p" equals to log2(max(ILM_size,DLM_size)). Using the numbers in the example above, "p" equals to log2(max(1KB, 2KB)), which is 11.

If the above mentioned mapping rule is violated, a request will erroneously map to both ILM and DLM. When this violation is detected, a Machine Error Exception called "local memory base setup error" (with ETYPE equal to 6) is generated. This local memory base setup error is a next-precise exception and is detected by hardware when programming the local memory attributes. More specifically, when programming ILMB or DLMB using the MTSR instruction, this exception is detected and issued if both of the following conditions are true: (1) IBPA and DBPA violates the base address rule; (2) Both ILM and DLM are enabled (i.e., both ILMB.IEN and DLMB.DEN are set). In addition, the write instruction causing the exception is canceled (forced to NOP) by HW to avoid further potential errors. To fix this violation, the physical base addresses of ILM and DLM must be adjusted to meet to the rule.

ULM provides an SOC interface to support external accesses (e.g., from an external DMA engine). This SOC interface is simplified for speed and hence is more restricted than that for the internal CPU. It only provides one set of address for both ILM/DLM and uses the most significant bit (MSB) to select between the two: "0" for ILM and "1" for DLM. In addition, it assumes that both ILM and DLM start at address 0. It is up to SOC decoders to handle (by controlling requests to ULM) more complicated cases like: ILM and DLM are not adjacent, or ILM and DLM have different sizes. If any access generates an error, the result is UPREDICTABLE and this error is notified through an optional status signal (please refer to the Integration Guide for the availability of this status signal).

Errors can also occur at the local memory interface to external SRAMs. For instance, errors may occur if the actual physical memory (1KB) is smaller than the specified local memory size (2KB). Accesses to addresses without physical memory generate errors with UNPREDICTABLE results. These errors may be fed back to the CPU via an optional status signal at the local memory interface (please check the Integration Guide for its availability). If this status signal is implemented, the errors also trigger a "bus error" exception (general exception with ETYPE equal to 4) for reads/loads or an "imprecise bus error" exception (general exception with ETYPE equal to 5) for writes/stores.

Since ULM does not support double-buffer mode, there is no need to allocate actual flip-flops for these 2 control bits. Therefore, both DLMB.DBB and DLMB.DBM should be hardwired to 0 in ULM configuration. In other words, these 2 bits still exist in DLMB but become Read-Only (RO) with value of 0.

To support large applications, the maximum size for local memory is increased from 1 MB to 4 MB. It is specified in ILMB.ILMSZ for ILM and DLMB.DLMSZ for DLM respectively.

7.6. Local Memory Usage Constraints

Local memories are specially built memories for fast access. Consequently, to reduce hardware overhead, local memories have additional usage constraints that may not otherwise exist in general memory. Some typical constraints are listed below. For more complete and accurate constraints, please consult the implementation documents.

- The local memory needs to be mapped onto an uncacheable region; otherwise, UPREDICTABLE behavior may happen to the local memory content.
- The contents of the local memory never enter caches, and vice versa.
- If the address range of the local memory overlaps with other memories (e.g., buses or devices), only the local memory can be accessed while others are ignored (shadowed) completely.
- The local memory needs to be mapped onto a page whose size is larger than or equal to the local memory size.
- The data local memory content will be changed UNPERDICTABLY when the data local memory access mode is transitioned between the double buffer mode and the normal (non-double buffer) mode.
- Instructions cannot be placed in the Data Local Memory (DLM). Note that this constraint does not apply to the Unified Local Memory (ULM) configuration.

8. Local Memory DMA

The local memory DMA transfers blocks of data between the local memory of AndesCore™ processors and external (off-core) memories. This transfer occurs in parallel with the AndesCore execution pipeline to improve performance. To transfer large amounts of data to or from the local memory, it is more efficient to use the DMA engine instead of the AndesCore load/store instructions. This chapter describes the DMA detail and contains the following sections.

8.1. Features

The AndesCore Local Memory DMA engine has the following features:

- Two channels.
- Programming using physical addressing.
- Address translation and permission check performed by software.
- Support for accessing in Superuser mode and User mode.
- Support for both instruction and data local memories.
- Support for incrementing strides on external address.
- Aligned internal address.
- Aligned external address for the basic configuration.
- Sub-act commands for the implementation of the DMA command queue.

- Optional action-command mode and fast-start mode.
- Optional Un-aligned External Address (UNEA) feature allowing un-aligned external address for element transfer.
- Optional 2-D Element Transfer (2DET) feature providing an easy way to transfer two-dimensional blocks from external memory.

The DMA architecture version 2.0 provides the following additional features:

- A channel can be activated from the “Complete” state, in addition to the “Idle” and “Stop/Error” states.
- When a DMA channel transfer is activated, if the DMA Transfer Element Count Register (DMA_TCNT) is equal to zero, the value of the DMA Refill Element Count Register will be copied to the DMA_TCNT register automatically.
- If a new fast-start mode is enabled, a DMA channel transfer can be started (i.e., activating a DMA channel) by writing to a trigger register instead of writing to the DMA Action Register. A trigger register (selected by DMA_GCSW.FSM) can be DMA Transfer Element Count Register, DMA Internal Start Address Register, or DMA External Start Address Register.
- For efficient management, new sub-action commands and a head pointer (DMA_GCSW.HCHAN) are added to support the DMA Channel Queue.
- A DSB instruction is no longer required between a write to the DMA_CHNSEL register and a read to any register of a channel. The read is guaranteed to get the value of the register of the newly selected channel immediately.
- The maximum supported size of the local memory is extended from 1MB to 4MB.

8.2. Basic Rules of Operation

The detailed operation of the DMA engine can be described using the state transition diagram of one channel as shown in Figure 10. In this diagram, the solid lines represent the transition paths common to all versions of DMA architecture, while the dotted lines represent new transition paths only available in DMA version 2 and later.

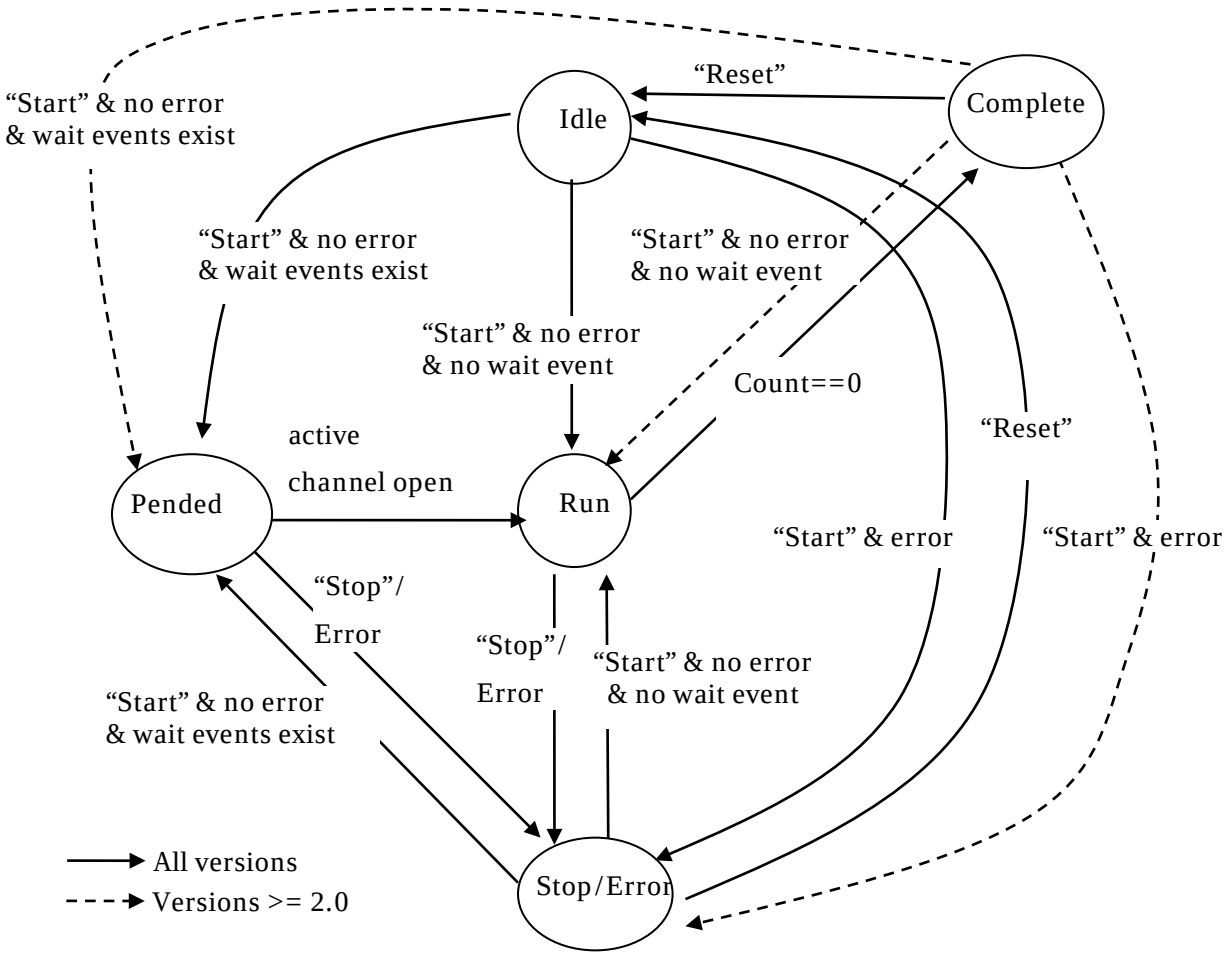


Figure 10. DMA channel state transition diagram

The following basic rules apply to all versions of the DMA architecture.

1. The current architecture defines a maximum of two DMA channels, yet future extension to more DMA channels is possible.

2. If more than one DMA channels are activated, only one DMA channel can be in the “Run” state, while the other activated channel must be in the "Pended" state.
3. Each DMA channel has its own set of control and status registers, and its own operating state.
4. A superuser mode program is able to access DMA-related System Registers with MTSR/MFSR instructions. In contrast, a user mode program is allowed to access DMA-related User Special Registers with MTUSR/MFUSR instructions only if PRUSR_ACC_CTL.DMA_USR_EN is set.
5. As shown in Figure 10, a DMA channel has the following states: Idle, Run, Pended, Stop/Error, and Complete. This diagram illustrates the relationship of these states and state transitions among them. State transitions are usually triggered by DMA action commands written to the DMA Action Register such as: “Start”, “Stop”, and “Reset.”
   - The state transition diagram in Figure 10 only describes the major state transitions. For more details, please refer to Section 10.8.4 "DMA Action Register” for all acceptable commands and corresponding transitions.
6. The “Stop” and “Error” share one common state, but they can be distinguished by checking if there is any error recorded in the DMA Status Register.
7. In version 1 of the DMA architecture, a channel can be activated from the "Idle" or "Stop/Error" state by a “Start” command. Starting from version 2, in addition to these two states, a channel can also be activated from the "Complete" state by a “Start” command.
8. Starting from DMA architecture version 2, writing any value into a trigger register activates the channel in the fast-start mode.
9. When a DMA channel is activated, it enters one of the following states:
   - Run: if there are no errors detected and no wait events required.
   - Pended: if there are no errors detected but some wait event is required.
   - Stop/Error: if an error is detected.
10. A wait event is a condition that a DMA channel must wait before it can enter the “Run” state. There are two kinds of wait events, (a) Channel Ready Event and (b) DLM Bank Switching Event. For more detailed information, please refer to Section 10.8.4 "DMA Action Register.”

11. The issue order is the ordering that channels are activated by "Start" commands. The channels must follow their issue order to enter the "Run" state.
    - The channels are expected to complete and generate interrupts in the issue order if no error occurs.
    - However, a channel can immediately generate an interrupt without following the issue order if it is stopped due to errors or by a "Stop" command.
12. If an error occurs, software needs to perform the following operations to restart the DMA transfer:
    - Fix this error based on error conditions specified in the DMA Status Register.
    - Write the DMA Action Register with a "Reset" command to clear the error conditions and interrupt generated, and then set the DMA channel into the "Idle" state.
    - Check the DMA Status Register to make sure the DMA channel is in the "Idle" state.
    - Write the DMA Action Register with a "Start" command to re-start the DMA transfer.
13. When a channel is stopped due to an error or a "Stop" command, the other channel will not be affected.
14. When a channel is in the "Run" or "Pended" state, writing to any one of the following registers may cause a DMA transfer disruption error:
    - The DMA_SETUP, DMA_ISADDR, DMA_ESADDR, DMA_TCNT, DMA_2DSET, or DMA_2DSCTL System Registers.
    - Starting from version 2, writing a "Start" or "Reset" command into the DMA_ACT register will cause a disruption error when the channel is in the "Run" or "Pended" state.
    - If a DMA transfer disruption error occurs due to writing any channel register mentioned above, the new value of this channel register is unpredictable.
15. To select an available DMA channel to use, software can check the DMA Global Control and Status Word Register to see if any channel is in the "Idle" state. Starting from version 2, any channel in the "Complete" state is also available for use.
16. A DMA channel data transfer region is programmed using physical addresses. Moreover, software is responsible for virtual address to physical address translation

and all access permission checks.

17. The baseline DMA engine supports only transfer element size aligned internal and external memory addresses. However, optional "Unaligned External Address" feature may be implemented to support transferring data between unaligned external memory addresses and aligned internal memory addresses.
18. There is no coherency support between the level-one caches and the DMA accesses. Thus the DMA accesses must be either from un-cached space or if from cached space, the transfer region should have already been properly written back by software instructions.
19. While a DMA channel is running, there is no ordering requirement of memory accesses between those generated by the DMA and those generated by the reads and writes of an AndesCore. When a DMA channel has completed running, all its transfers are visible to all other readers in the system. All memory accesses of the DMA transfers are ordered by the channel running sequence and within a running channel, all memory accesses are ordered from the starting address to the end address of the transfer.
20. An interrupt can be generated when a DMA channel transfer is complete, explicitly stopped, or terminated due to an error. There are individual control bits to specify the interrupt generation condition to best fit the need.
21. Once a DMA-related interrupt is generated, the interrupt bit and error flags can be cleared by the following conditions:
    - Receiving a "Reset" command in the "Complete" or "Stop/Error" states.
    - Receiving a "Start" command in the "Stop/Error" state.
    - Receiving a "Start" command in the "Complete" state when the DMA architecture version is equal to or larger than 2.0.
22. To program DMA, Superuser mode supports either polling-style or interrupt-style programming model. In contrast, User mode only supports polling-style programming model.
23. When the CPU enters the debug mode, any active DMA transfer will "freeze" (be suspended) to avoid potential problems due to security permission discrepancy. This DMA transfer will only "unfreeze" (resume) and complete the remaining transfer when the CPU exists the debug mode.

8.2.1. Simplified View of State Transition Diagram If No Error Occurs

The state transition diagram can be simplified to Figure 11 if no error occurs. Typical state transitions in channel programming are described below:

1. A channel is activated by a "Start" command from the "Idle" state.
2. If there are wait events, the channel moves to the "Pended" state; otherwise, the channel moves to the "Run" state.
3. When the channel completes, it moves to the "Complete" state.
4. For DMA architecture versions smaller than 2.0, the channel must be reset to the "Idle" state (using a "Reset" command) before it can be activated again. For version 2.0 and above, the channel can be re-activated directly from the "Complete" state.

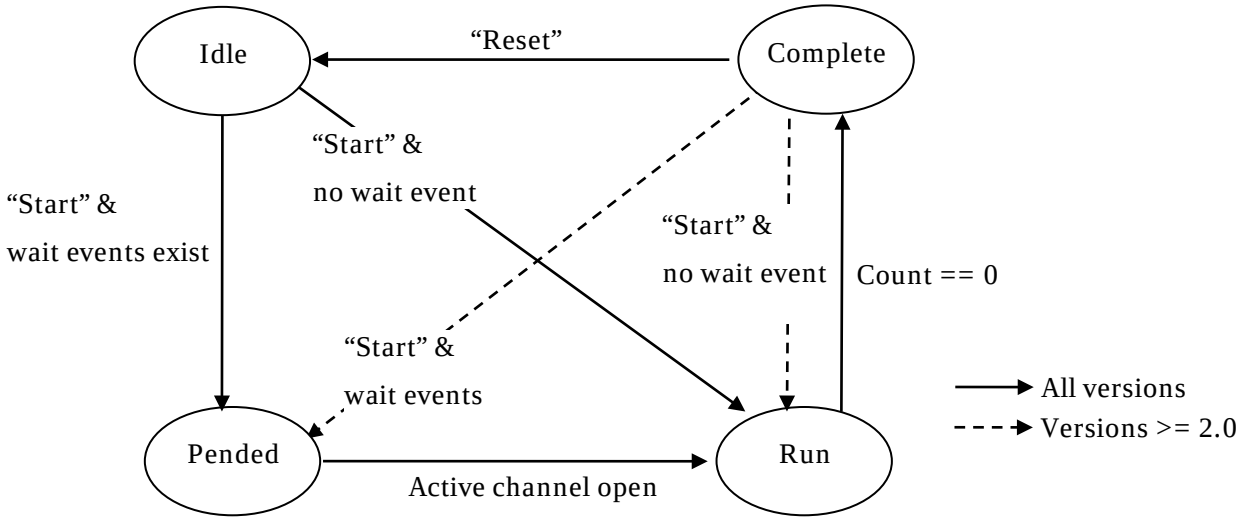


Figure 11. Simplified view of state transition diagram if no error occurs

8.3. Related Registers

The local memory DMA is controlled and initiated by accessing the related DMA system registers.

Local memory DMA related registers can be classified into two different categories. One category (DMA Global Register) contains registers which represent or control overall DMA status and behaviors. The second category (DMA per Channel Register) contains registers which are used to reflect the status and control the behaviors of a selected DMA channel. Table 16 lists all the DMA related registers. All these registers require superuser mode privilege to access. Please see Section 10.8 Local Memory DMA Registers for their detailed definitions.

Table 16. Local memory DMA related registers

Name	Type	Updater	Section
DMA Global Registers			
DMA Configuration	RO	-	10.8.1
DMA Global Control and Status Word	RW	SW/HW	10.8.2
DMA Channel Selection	RW	SW	10.8.3
DMA Head Channel Control and Status	RW	SW/HW	10.8.13
DMA per Channel Registers			
DMA Action	WO	SW	10.8.4
DMA Setup	RW	SW	10.8.5
DMA Internal Starting Address	RW	SW/HW	10.8.6
DMA External Starting Address	RW	SW/HW	10.8.7
DMA Transfer Element Count	RW	SW/HW	10.8.8
DMA Status	RO	HW	10.8.9
DMA 2D Setup	RW	SW	10.8.10
DMA 2D Startup Control	RW/WS	SW/HW	10.8.11
DMA Refill Element Count	RW	SW	10.8.12

8.4. Interrupt Interface

An interrupt is generated when a DMA channel transfer is complete, manually stopped, or stopped due to errors and the corresponding interrupt enable control is set. This interrupt is exported out of the core through an interface signal, DMA_INT. This signal can then be connected with the internal or external interrupt controller. In addition, the signal can be synchronous or asynchronous depending on the implementation.

Note that in order to know which DMA channel or what state of a DMA channel generated the interrupt, software has to examine the corresponding status bits to get this information.

Once a DMA-related interrupt is generated, the status for this interrupt can be cleared with the following events:

- Receiving a “Reset” command in the “Complete” or “Stop/Error” state.
- Receiving a “Start” command in the “Stop/Error” state.
- Receiving a “Start” command in the “Complete” state when the DMA architecture version is equal to or larger than 2.0.

8.5. DMA Channel Queue

As shown in Figure 12, all DMA channels are organized as one "DMA Channel Queue" to ease programming and achieve high efficiency. When organized as a queue, a channel can easily be activated by program, while another channel is being setup for the next transfer. Furthermore, this DMA Channel Queue accepts input commands sequentially and completes them in order (if no error occurs). This sequential property allows a DMA program to expect DMA results in the exact ordering as commands were programmed.

To speed up management, the DMA Channel Queue consists of two pointers (a head and a tail) and logically forms a circular queue. These two pointers plus action commands are used to perform the "enqueue" and "dequeue" operations. More detailed operations are described in the following sections.

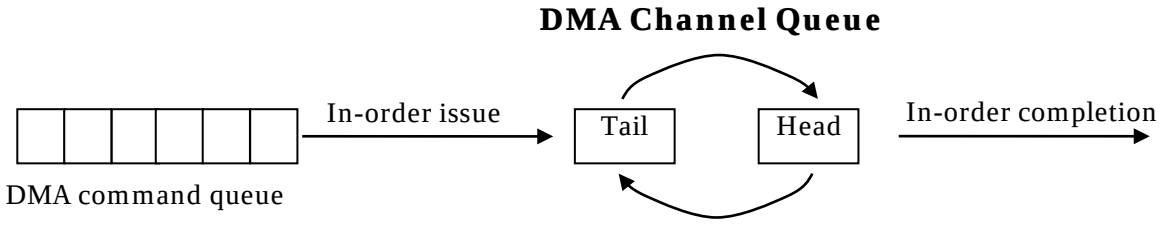


Figure 12. Implementation with a DMA command queue

The tail pointer always points to a channel that is ready to accept a new command. When the tail channel is activated with a new command, this is referred as the "enqueue" operation. An enqueue operation is performed using a "Start" action command with an "EnQ" or "EnQWait" sub-action command. After an enqueue operation, the tail pointer is updated to point to the next available idle channel. Furthermore, the tail pointer can be efficiently implemented using the DMA_CHSEL.CHAN which is enhanced with hardware acceleration.

The head pointer always points to the oldest channel that has performed the DMA transfer and waits for the final status check. When the transfer status is checked and the data is processed, this is referred as the "dequeue" operation which makes the head channel available again and

also updates the head pointer. A dequeue operation is performed by writing an arbitrary value to the DMA_HSTATUS register. Moreover, the head pointer can be easily implemented using the DMA_GCSW.HCHAN which has extra hardware support.

Although the DMA channels are expected to complete and generate interrupts in the issue order, a channel could immediately generate an interrupt without following the issue order when the channel is stopped due to any error or by a “Stop” command. If such situation occurs, the program could still follow the issue order to check the result of the DMA commands.

Please refer to the following sections for more information.

- Section 8.8.6 "Implementing a DMA Command Queue”
- Section 10.8.2 "DMA Global Control and Status Word Register”
- Section 10.8.4 "DMA Action Register”
- Section 10.8.13 "DMA Head Channel Control and Status Register”

8.6. Action-command Mode and Fast-start Mode

A DMA channel can be activated in two modes: action-command mode and fast-start mode. These activation modes are selected using DMA_GCSW.FSM. The detailed activation methods for each mode are shown in Table 17.

Table 17. Activation modes for a channel

Operation mode	Activation Method
Action-command Mode	■ Write a "Start" command into the DMA_ACT register
Fast-start Mode	■ Write a "Start" command into the DMA_ACT register ■ Write any value into a trigger register

If the channel is in the fast-start mode, writing any value into the trigger register will activate the channel. The trigger register is selected with the DMA_GCSW.FSM field and can be one of the following registers: DMA_TCNT, DMA_ISADDR, or DMA_ESADDR.

Writing any value to the trigger register is equivalent to performing the following operations:

1. Update the value of the trigger register with the new value.
2. Write a "Start" command into the DMA_ACT register. The DMA_GCSW.SCMD field will be used as the sub-action command for the "Start" command.

The program should follow the steps to activate a channel in the fast-start mode.

1. Enable the fast-start mode and select the appropriate trigger register with the DMA_GCSW register.
2. Set up the control registers for the channel.
3. Activate the channel by writing a value into the trigger register.

Code samples for activating a channel are shown in Section 8.8.1 "DMA Setup in the Action-command Mode" and Section 8.8.2 "DMA Setup in the Fast-start Mode". Please refer to these sections form more information.

8.7. Essential Data Serialization Barrier Instructions

A “Data Serialization Barrier” (DSB) instruction may be required between a write and read operations on DMA registers. The following rules define the pairs of instructions that require DSB instructions:

- To read the status of a channel after a write to the DMA_ACT or the trigger register of the same channel.
  - The status of a channel can be from: the DMA_GCSW register, the DMA_STATUS register or the DMA_HSTATUS register.
  - The channel can be selected by the DMA_CHNSEL register or the DMA_GCSW.HCHAN field.
- To read any channel-related register after a write to the DMA_CHNSEL register for DMA versions smaller than 2.
  - However, this restriction is removed starting from the DMA architecture version 2. In other words, a DSB instruction is no longer required between the read from any channel-related registers after a write to the DMA_CHNSEL register.
- To read any channel-related register after a write to the DMA_GCSW register.
  - Nevertheless, to read the DMA_HSTATUS register after a write to the DMA_GCSW register does not require a DSB instruction between them. This is because a read from the DMA_HSTATUS is guaranteed to get the status of the most recently selected channel.

To write any channel-related register after a previous write does not require a DSB instruction in between them.

The trigger register is determined by the DMA_GCSW.FSM field and can be one of the registers: DMA_TCNT, DMA_ESADDR or DMA_ISADDR. Please refer to Section 8.6 "Action-command Mode and Fast-start Mode” for more information.

8.8. Sample Code

The "`li reg, imm32`" instruction in the example codes is a software pseudo assembly instruction of "loading a 32-bit immediate value into a register". It will be translated into the following two hardware instructions by the NDS32 assembler:

- `sethi reg, hi20(imm32)`
- `ori reg, lo12(imm32)`

where hi20(imm32) is the "higher 20 bits" of the imm32 value and lo12(imm32) is the "lower 12 bits" of the imm32 value.

8.8.1. DMA Setup in the Action-command Mode

The following code sequence illustrates a typical DMA setup flow.

- For DMA architecture versions smaller than 2.0, a "Reset" command is needed to make sure a channel is activated from the "Idle" state.

```
li r1, 0x80000000    ! Enable DMA
mtsr r1, DMA_GCSW    ! Ignored if DMA is already enabled
li r1, 0x00000000    ! Channel number
mtsr r1, DMA_CHNSEL ! Select a channel
                        ! No DSB instruction is required here. Refer to
                     Section
                    ! 8.7 for more information.
li r1, SETUP_CONST
mtsr r1, DMA_SETUP   ! DMA transfer setup
li r1, ISADDR_CONST
mtsr r1, DMA_ISADDR  ! Setup internal starting address
li r1, ESADDR_CONST
mtsr r1, DMA_ESADDR  ! Setup external starting address
li r1, TCNT_CONST
mtsr r1, DMA_TCNT    ! Setup total transfer count
li r1, 0x00000003
mtsr r1, DMA_ACT     ! Reset action
li r1, 0x00000001
mtsr r1, DMA_ACT     ! Start action
```

```
dsb                  ! A DSB instruction is required here. Refer to Section
                     ! 8.7 for more information.
! example code ends here ...
mfsr r1, DMA_STATUS  ! Polling DMA channel status
.............
```


- For DMA architecture versions greater or equal to 2.0, no “Reset” command is needed to make sure the channel is in the “Idle” state before the channel is activated.

```
li r1, 0x80000000    ! Enable DMA
mtsr r1, DMA_GCSW    ! Ignored if DMA is already enabled
li r1, 0x00000000    ! Channel number
mtsr r1, DMA_CHNSEL ! Select a channel
                     ! No DSB instruction is required here. Refer to Section
                     ! 8.7 for more information.
li r1, SETUP_CONST
mtsr r1, DMA_SETUP   ! DMA transfer setup
li r1, ISADDR_CONST
mtsr r1, DMA_ISADDR  ! Setup internal starting address
li r1, ESADDR_CONST
mtsr r1, DMA_ESADDR  ! Setup external starting address
li r1, TCNT_CONST
mtsr r1, DMA_TCNT    ! Setup total transfer count
li r1, 0x00000001
                     ! No “Reset” action is required here. Refer to Section
                     ! 8.2 for more information.
mtsr r1, DMA_ACT     ! Start action
dsb                  ! A DSB instruction is required here. Refer to Section
                     ! 8.7 for more information.
! example code ends here ...
mfsr r1, DMA_STATUS  ! Polling DMA channel status
.............
```

8.8.2. DMA Setup in the Fast-start Mode

The following code sequence illustrates a typical DMA setup flow in the fast-start mode. Several assumptions are made:

- The DMA_TCNT register is used as the trigger register to activate the channel.
- "NoCond" (No Condition) sub-action command is used in this example (specified in the DMA_GCSW.SCMD field).


```
li r1, 0x80040000    ! Enable DMA in the fast-start mode.
                     ! Setting the DMA_TCNT register to activate
                     ! the channel with the "Start.NoCond" command.
mtsr r1, DMA_GCSW
li r1, 0x00000000    ! Channel number
mtsr r1, DMA_CHNSEL  ! Select a channel
                     ! No DSB instruction is required here. Refer to Section
                     ! 8.7 for more information.
li r1, SETUP_CONST
mtsr r1, DMA_SETUP   ! DMA transfer setup
li r1, ISADDR_CONST
mtsr r1, DMA_ISADDR  ! Setup internal starting address
li r1, ESADDR_CONST
mtsr r1, DMA_ESADDR  ! Setup external starting address
li r1, TCNT_CONST
mtsr r1, DMA_TCNT    ! Start the channel by writing the transfer count into
                     ! the trigger register.
dsb                  ! A DSB instruction is required here. Refer to Section
                     ! 8.7 for more information.
! example code ends here ...
mfsr r1, DMA_STATUS  ! Polling DMA channel status
.............
```

8.8.3. DMA Channel Selection

If you do not know which DMA channel is idle for use, you can use the following code sequence to find an idle DMA channel.


```
Find_Idle_Channel:
    mfsr r1, DMA_GCSW
    andi r2, r1, 0x07        ! Extract channel 0 status
    beqz r2, Channel1_Free
    andi r2, r1, 0x38        ! Extract channel 1 status
    beqz r2, Channel2_Free
    j No_Free_Channel
Channel1_Free:
    ............
Channel2_Free:
    ............
No_Free_Channel:
    ............
```

8.8.4. Polling DMA Channel Status

The following code sequence polls the “complete” status of a selected DMA channel.

```
......... ! DMA channel setup and action command
... dsb   !  make sure channel has started
! make sure a “dsb” is between DMA channel setup and this code
.........
! example code begins here ...
Check_Status:
    mfsr r1, DMA_STATUS
    andi r0, r1, 0x4
    bnez r0, Transfer_Complete
    j Check_Status
Transfer_Complete:
    ............
```

8.8.5. Displaying the Details of a Newly Selected Channel

The following C-like code sequence displays the details of a newly selected channel. For DMA architecture versions smaller than 2.0, a Data Serialization Barrier (DSB) instruction is required between a write to the DMA_CHNSEL register and a read from the DMA_STATUS register. In contrast, for DMA versions greater or equal to 2.0, no DSB instruction is needed because the hardware guarantees to use the most updated value for the DMA_CHNSEL register. Please refer to Section 8.7 “Essential Data Serialization Barrier Instructions” for more information.


```
void Channel_Info(unsigned int ch_id)
{
  $DMA_CHNSEL = ch_id;

  dsb();  // this DSB instruction is required only if DMA version < 2
          // No DSB instruction is needed if DMA version >= 2

  printf(“DMA_STATUS:%08x\n”, $DMA_STATUS);
  printf(“DMA_ISADDR:%08x\n”, $DMA_ISADDR);
  printf(“DMA_DSADDR:%08x\n”, $DMA_DSADDR);
  printf(“DMA_TCNT:%08x\n”, $DMA_TCNT);
.............
}
```

8.8.6. Implementing a DMA Command Queue

To efficiently manage multiple requests, a DMA command queue (*cmd_queue*) is often implemented to store commands that are ready to be issued. This queue usually consists of two subroutines (“IssueCmd” and “DMA_ISR”) to manage DMA commands. These two subroutines and their purposes are shown in Table 18. Please refer to Section 8.5 "DMA Channel Queue” for a system view of the DMA command queue.

Table 18. Subroutines for management of a DMA command queue

Subroutine	Purpose
void IssueCmd(Cmd* cmd)	Called by a program to issue a new DMA command. A command from the command queue is issued immediately if the tail channel is in the “Idle” state.

Subroutine	Purpose
void DMA_ISR()	Called by the interrupt generated from the head channel. If a command has completed normally, this command is dequeued and the head channel is reset. Furthermore, a new command is enqueued into the tail channel if there are commands ready to be issued.

The following C-like pseudo code illustrates a typical implementation of “IssueCmd” and “DMA_ISR” for DMA architecture versions greater than 2.0.

This sample code makes the following assumptions:

- The DMA engine operates in the “Fast-start” mode.
- The “EnQ” (enqueue) sub-action command is used for DMA_GCSW.SCMD.
- The DMA_ESADDR register is used as the trigger register (selected using DMA_GCSW.FSM).

```
Queue cmd_queue; // The command queue for the commands ready to be issued.

void IssueCmd(Cmd* cmd)
{
  setgie.d; // Disable the interrupt of all channels.
  enqueue( &cmd_queue, cmd );
  do
  {
    if( $DMA_STATUS == Idle ) // Check the status of the tail channel.
    {
      Cmd* cmd = dequeue( &cmd_queue );
      $DMA_TCNT = cmd->size;
      $DMA_ISADDR = cmd->isaddr;

      // Setting the DMA_ESADDR register will use the “Start.EnQ”
      // command to activate the channel and change the DMA_CHNSEL register
      // to point at the next channel.
      $DMA_ESADDR = cmd->esaddr;
    }
    else
      break;
```

```
    } while (queue_size( &cmd_queue ) > 0 );
    setgie.e; // Enable the interrupt of all channels.
  }

void DMA_ISR()
{
  while ( 1 )
  {
    // If the command has not completed normally, perform the error handler.
    if ( $DMA_HSTATUS != Complete )
    {
      error_handler();
      return;
    }

    $DMA_HSTATUS = $R0; // Reset and dequeue the head channel.

    // Enqueue a new command into the tail channel.
    if ( queue_size( &cmd_queue ) > 0 )
    {
      Cmd* cmd = dequeue( &cmd_queue );
      $DMA_TCNT = cmd->size;
      $DMA_ISADDR = cmd->isaddr;

      // Setting the DMA_ESADDR register will use the "Start.EnQ"
      // command to activate the channel and change the DMA_CHNSEL register
      // to point at the next channel.
      $DMA_ESADDR = cmd->esaddr;
    }
  }
}
```

9. High Speed Memory Port

This chapter describes high speed memory port.

In addition to the AMBA AHB-style system interface, to reduce memory access delays and thus increase system performance, an AndesCore may provide a high speed memory port interface which has higher bus protocol efficiency and can run at a higher frequency when connecting to a memory controller. It is called High Speed Memory Port.

This high speed memory port uses a protocol that has higher speed than the AMBA AHB bus protocol. Please refer to AndesCore implementation documents for detailed protocol definitions and signals.

The High Speed Memory Port is defined within a power-of-2 range of aligned memory region from 1MB to 4GB. The beginning and range of this region are defined in one system register.

Table 19. High Speed Memory Port related register

Name	Type	Updater	Section
High Speed Memory Port Starting Address	RW	SW	10.4.10

Configuration Note:

The High Speed Memory Port and the defined registers only have their proper usages and meanings when a main bus port exists. The main bus port can use any possible protocol such as AHB, OCP, AXI, etc. Please do not confuse the “High Speed Memory Port” with this “main bus protocol”. They are related here, but are actually different concepts.

10. System Register Definitions

This chapter describes system registers and contains the following sections:

Note : when writing to a system register, either “ISB” or “DSB” instruction must be added after “MTSR” if users want the written value to take effect immediately or read back this latest written value.

10.1. Terminology

- **IM**: Implementation dependent/determined.
- **DC**: The reset value is don't care (but it must not contain "reserved values").
- **RO**: Read Only register/field by software. Any software write to a RO register/field will be silently ignored by hardware, so it can also be denoted as ROWI.
- **WO**: Write Only register/field by software. Any software read from a WO register/field will get a value of 0, so it can also be denoted as WORAZ.
- **RW**: Read/Write register/field by software.
- **WS**: Write shadow of another register. More specifically, writing to another register will update this field as well.
- **W1C**: Write one to clear the state of the field.
- **RAZWI**: Read As Zero Write Ignored register/field by software. This behavior is for reserved register fields.
- **SR Encoding**: System Register instruction encoding index. It is 10 bits. It is partitioned into 3 parts, Major, Minor, and Extension. The Major index is 3 bits. The Minor index is 4 bits. The Extension index is 3 bits.
- **Reserved** value in a **RW** field: Writing a reserved value into a RW field will cause a "Reserved Value" exception.

10.2. Configuration System Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
cr0	CPU_VER	0	0	0	136
cr1	ICM_CFG	0	1	0	138
cr2	DCM_CFG	0	2	0	142
cr3	MMU_CFG	0	3	0	146
cr4	MSC_CFG	0	4	0	151
cr7	MSC_CFG2	0	4	1	161
cr5	CORE_ID	0	0	1	161
cr6	FUCOP_EXIST	0	5	0	164

10.2.1. CPU Version Register

Mnemonic Name: cr0 (CPU_VER)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {0, 0, 0}

31 24	23 16	15 0
CPUID	REV	CFGID

Field name	Bits	Description	Type	Reset
Config ID (CFGID)	16 (15,0)	Configuration Identifier for instruction extension. It indicates which instruction extensions are implemented in this CPU. It is defined as follows: Bit (0): Performance extension exists? Bit (1): 16-bit extension exists? Bit (2): Performance extension 2 exists? Bit (3): COP/FPU extension exists? Bit (4): String extension exists? Bit (5): Saturation arithmetic extension exists? Bit (6): Andes Custom Extension (ACE) exists?	RO	IM
Revision (REV)	8 (23,16)	Revision number for this CPU product.	RO	IM

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CPU ID
(CPUID)</td><td>8
(31,24)</td><td>CPU Identification number. It is used to distinguish CPU products. Please refer to the datasheet for the exact CPU ID. As examples, the following table shows the CPUID for some CPUs:
<table>
<tr><th>CPU</th><th>Value</th></tr>
<tr><td>N7</td><td>0x07</td></tr>
<tr><td>N8</td><td>0x08</td></tr>
<tr><td>N13</td><td>0x0D</td></tr>
<tr><td>N968</td><td>0x19</td></tr>
<tr><td>N1068</td><td>0x1A</td></tr>
<tr><td>D10</td><td>0xDA</td></tr>
<tr><td>N15</td><td>0xoF</td></tr>
<tr><td>D15</td><td>0xDF</td></tr>
</table>
</td><td>RO</td><td>IM</td></tr>
</table>

10.2.2. Instruction Cache/Memory Configuration Register

Mnemonic Name: cr1 (ICM_CFG)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {0, 1, 0}

31 21	20 19	18 17	16	15	14 13	12 10	9	8 6	5 3	2 0
Reserved	ILM_ECC	IC_ECC	ULM_2BANK	EXT	BSAV	ILMB	ILCK	ISZ	IWAY	ISET

Field name	Bits	Description	Type	Reset
Icache sets per way (ISET)	3 (2,0)	Icache sets (# of cache lines) per way: Encoding: Meaning 0: 64 1: 128 2: 256 3: 512 4: 1024 5: 2048 6: 4096 7: Reserved	RO	IM
Icache ways (IWAY)	3 (5,3)	Icache ways: Encoding: Meaning 0: Direct-mapped 1: 2 way 2: 3 way 3: 4 way 4: 5 way 5: 6 way	RO	IM

Field name	Bits	Description	Type	Reset
		6 — 7 way 7 — 8 way		
Icache line size (ISZ)	3 (8,6)	Icache block (line) size: 0 — No Icache 1 — 8 bytes 2 — 16 bytes 3 — 32 bytes 4 — 64 bytes 5 — 128 bytes 6-7 — Reserved	RO	IM
Icache locking support (ILCK)	1 (9)	Icache locking support: 0 — No locking support 1 — With locking support	RO	IM
On-chip Instruction Local Memory Base (ILMB)	3 (12,10)	Indicates how many On-chip Instruction Local Memory Base Registers exist. 0 — No ILMB exists 1 — One ILMB exists 2-7 — Reserved	RO	IM
BSAV	2 (14,13)	ILM base register alignment version indication: 0 — ILM base address is 1MB-aligned.	RO	IM

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>ILM base address is aligned to local memory size that has to be power of 2.</td></tr><tr><td>2-3</td><td>Reserved</td></tr></table></td><td></td><td></td></tr>
<tr><td>EXT</td><td>1
(15)</td><td>Indicates whether the instruction local memory is external or not.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Internal ILM</td></tr><tr><td>1</td><td>External ILM</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>ULM_2BANK</td><td>1
(16)</td><td>In Unified Local Memory (ULM) configuration, indicates whether the Instruction Local Memory has two banks or not.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Only 1 memory bank</td></tr><tr><td>1</td><td>2 memory banks</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>IC_ECC</td><td>2
(18,17)</td><td>Parity/ECC support for the instruction cache.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>No parity/ECC support</td></tr><tr><td>1</td><td>Has parity support</td></tr><tr><td>2</td><td>Has ECC support</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>ILM_ECC</td><td>2
(20,19)</td><td>Parity/ECC support for the instruction local memory.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>No parity/ECC support</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

Field name	Bits	Description	Type	Reset
		1 Has parity support 2 Has ECC support 3 Reserved		
Reserved	11 (31,21)	Reserved	RAZWI	0

10.2.3. Data Cache/Memory Configuration Register

Mnemonic Name: cr2 (DCM_CFG)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {0, 2, 0}

31 21	20	19	18	17 16	15	14 13	12 10	9	8 6	5 3	2 0
Reserved	DLM_ECC		DC_ECC	ULM_2BANK	EXT	BSAV	DLMB	DLCK	DSZ	DWAY	DSET

Field name	Bits	Description	Type	Reset
Dcache sets per way (DSET)	3 (2,0)	Dcache sets (# of cache lines) per way: Encoding: Meaning 0: 64 1: 128 2: 256 3: 512 4: 1024 5: 2048 6: 4096 7: Reserved	RO	IM
Dcache ways (DWAY)	3 (5,3)	Dcache ways: Encoding: Meaning 0: Direct-mapped 1: 2 way 2: 3 way 3: 4 way 4: 5 way	RO	IM

Field name	Bits	Description	Type	Reset
		5 — 6 way 6 — 7 way 7 — 8 way		
Dcache line size (DSZ)	3 (8,6)	Dcache block (line) size: Encoding / Meaning 0 — No Dcache 1 — 8 bytes 2 — 16 bytes 3 — 32 bytes 4 — 64 bytes 5 — 128 bytes 6-7 — Reserved	RO	IM
Dcache locking support (DLCK)	1 (9)	Dcache locking support: Encoding / Meaning 0 — No locking support 1 — With locking support	RO	IM
On-chip Data Local Memory Base (DLMB)	3 (12,10)	Indicates how many On-chip Data Local Memory Base Registers exist. It also indicates if double buffer mode is supported or not. Encoding / Meaning 0 — No DLMB exists. 1 — One DLMB exists and no double buffer mode support. 2 — One DLMB exists and with double buffer mode support. 3-7 — Reserved	RO	IM

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>BSAV</td><td>2
(14,13)</td><td>DLM base register alignment version indication:<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>DLM base address is 1MB-aligned.</td></tr><tr><td>1</td><td>DLM base address is aligned to local memory size that has to be power of 2.</td></tr><tr><td>2-3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>EXT</td><td>1
(15)</td><td>Indicates whether the data local memory is external or not.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Internal DLM</td></tr><tr><td>1</td><td>External DLM</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>ULM_2BANK</td><td>1
(16)</td><td>In Unified Local Memory (ULM) configuration, indicates whether the Data Local Memory has two banks or not.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Only 1 memory bank</td></tr><tr><td>1</td><td>2 memory banks</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>DC_ECC</td><td>2
(18,17)</td><td>Parity/ECC support for the data cache.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>No parity/ECC support</td></tr><tr><td>1</td><td>Has parity support</td></tr><tr><td>2</td><td>Has ECC support</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DLM_ECC</td><td>2
(20,19)</td><td>Parity/ECC support for the data local memory.<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>No parity/ECC support</td></tr><tr><td>1</td><td>Has parity support</td></tr><tr><td>2</td><td>Has ECC support</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Reserved</td><td>11
(31,21)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

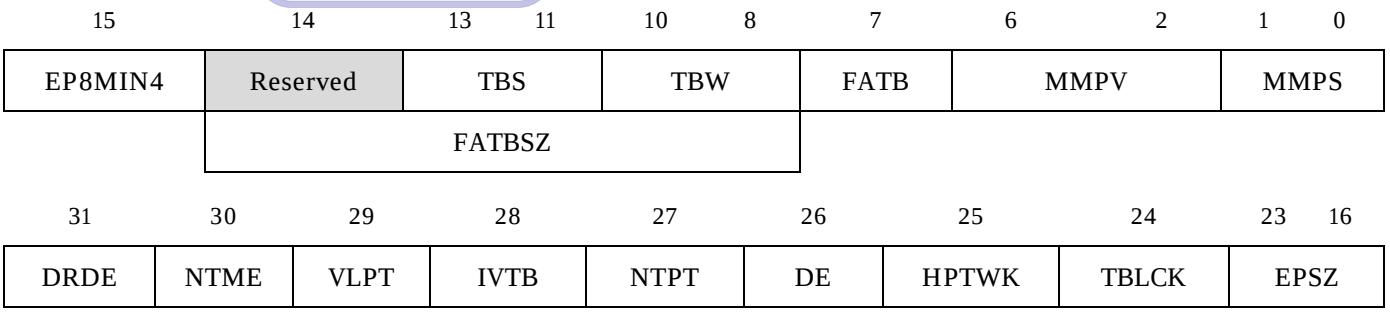
10.2.4. MMU Configuration Register

Mnemonic Name: cr3 (MMU_CFG)

IM Requirement: Required

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {0, 3, 0}



Field name	Bits	Description	Type	Reset
Memory Management Protection Scheme (MMPS)	2 (1,0)	The major category for the Memory Management Protection Scheme: Value: 0 — No memory management; 1 — Protection Unit; 2 — TLB MMU; 3 — Reserved	RO	IM
Memory Management Protection Version Number (MMPV)	5 (6,2)	Indicates the version number of the memory management protection scheme.	RO	IM
Fully-associative TLB (FATB)	1 (7)	Under the TLB MMU memory management scheme, is the TLB a fully-associative structure: Value: 0 — Non-fully-associative	RO	IM

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Fully-associative</td></tr></table></td><td></td><td></td></tr>
<tr><td>TLB size
(fully-associative)
(FATBSZ)</td><td>7
(14,8)</td><td>If the TLB is fully-associative, this field indicates the number of TLB entries. This 7 bits field will be reused in the following fields if the TLB is non-fully-associative.</td><td>RO</td><td>IM</td></tr>
<tr><td>TLB ways
(non-fully-associative)
(TBW)</td><td>3
(10,8)</td><td>The number of ways in a TLB cache:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Direct-mapped</td></tr><tr><td>1</td><td>2 ways</td></tr><tr><td>2</td><td>3 ways</td></tr><tr><td>3</td><td>4 ways</td></tr><tr><td>4</td><td>5 ways</td></tr><tr><td>5</td><td>6 ways</td></tr><tr><td>6</td><td>7 ways</td></tr><tr><td>7</td><td>8 ways</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>TLB sets per way
(non-fully-associative)
(TBS)</td><td>3
(13,11)</td><td>The sets per way of the TLB cache:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>4</td></tr><tr><td>1</td><td>8</td></tr><tr><td>2</td><td>16</td></tr><tr><td>3</td><td>32</td></tr><tr><td>4</td><td>64</td></tr><tr><td>5</td><td>128</td></tr><tr><td>6</td><td>256</td></tr><tr><td>7</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Reserved
(non-fully-associative)</td><td>1
(14)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>Is 8KB page supported while minimum page is 4KB?
(EP8MIN4)</td><td>1
(15)</td><td>Indicates if 8KB page is supported under the minimum 4KB page configuration. Usually, most AndesCore support (4KB, 1MB) or (8KB, 1MB), but not (4KB, 8KB, 1MB).
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>8KB page is not supported under the 4KB page configuration.</td></tr><tr><td>1</td><td>8KB page is supported under the 4KB page configuration.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Extra page size supported
(EPSZ)</td><td>8
(23,16)</td><td>Bit enable to indicate additional supported page sizes.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>8’bxxxxxxx1</td><td>16KB</td></tr><tr><td>8’bxxxxxx1x</td><td>64KB</td></tr><tr><td>8’bxxxxx1xx</td><td>256KB</td></tr><tr><td>8’bxxxx1xxx</td><td>1MB</td></tr><tr><td>8’bxxx1xxxx</td><td>4MB</td></tr><tr><td>8’bxx1xxxxx</td><td>16MB</td></tr><tr><td>8’bx1xxxxxx</td><td>64MB</td></tr><tr><td>8’b1xxxxxxx</td><td>256MB</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>TLB locking support
(TBLCK)</td><td>1
(24)</td><td>Indicates if TLB has locking support:
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No Locking support</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

Field name	Bits	Description	Type	Reset
		1 — With Locking support		
Hardware Page Table Walker implemented (HPTWK)	1 (25)	Indicates if a Hardware Page Table Walker is implemented or not: 0 — No HPTWK 1 — With HPTWK	RO	IM
Default Endian (DE)	1 (26)	The default endian bit. This bit will be copied to PSW.BE on reset and on interruption stack level transitions 0->1 and 1->2. 0 — Default is Little Endian. 1 — Default is Big Endian.	RO	IM
Partitions for non-translated attributes (NTPT)	1 (27)	Indicates how many address space partitions for non-translated attributes. 0 — 2 partitions from VA(31) 1 — 4 partitions from VA(31,30)	RO	IM
Invisible TLB (IVTB)	1 (28)	Indicates if a software-invisible TLB that includes additional PTEs from software-visible TLB exists. 0 — No such invisible TLB exists. 1 — Such invisible TLB exists.	RO	IM

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>VLPT</td><td>1
(29)</td><td>Indicates if the feature of VLPT for fast TLB fill handling is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The VLPT feature is not implemented.</td></tr><tr><td>1</td><td>The VLPT feature is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>NTME</td><td>1
(30)</td><td>Indicates if the non-translated VA to PA mapping function is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The non-translated VA to PA mapping function is not implemented.</td></tr><tr><td>1</td><td>The non-translated VA to PA mapping function is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>DRDE</td><td>1
(31)</td><td>Device register default endian. This bit will be copied to PSW.DRBE on reset and on interruption stack level transitions 0->1 and 1->2.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Default is Little Endian.</td></tr><tr><td>1</td><td>Default is Big Endian.</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

10.2.5. Misc. Configuration Register

Mnemonic Name: cr4 (MSC_CFG)

IM Requirement: Required

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {0, 4, 0}

15 13	12	11	10	9	8 7	6	5	4	3	2	1	0
BASEV	INTLC	ADR24	RDREG (GPR16)	L2C	AUDIO	MAC	DIV	TRACE	HSMP	PFM	LMDMA	EDM

31 30	29 28	27	26	25	24	23 21	20	19	18	17	16
MSC_EXT	INTV	HSP	PFT	ULM	EIT	SHADOW	MCU	IFC	IMR	IMV	NOD

Field Name	Bits	Description	Type	Reset
EDM	1 (0)	Indicates if an EDM exists or not. Value: 0 — EDM is not implemented. Value: 1 — EDM is implemented.	RO	IM
LMDMA	1 (1)	Indicates if a local memory DMA engine exists or not. Value: 0 — DMA is not implemented. Value: 1 — DMA is implemented.	RO	IM
PFM	1 (2)	Indicates if the performance monitoring feature (counters) exists or not. Value: 0 — Performance monitoring feature	RO	IM

Field Name	Bits	Description	Type	Reset
		0 is not implemented. 1 Performance monitoring feature is implemented.		
HSMP	1 (3)	Indicates if the High Speed Memory Port optional HSMP_SADDR (V1 and V2 of HSMP) register exists or not. 0 The HSMP_SADDR register is not implemented. 1 The HSMP_SADDR register is implemented.	RO	IM
TRACE	1 (4)	Indicates if a Debug Tracer Unit exists or not. 0 A Debug Tracer Unit is not implemented. 1 A Debug Tracer Unit is implemented.	RO	IM
DIV	1 (5)	Indicates if Divide instructions are supported in the core or not. 0 Divide instructions are not implemented. 1 Divide instructions are implemented. This field will be removed for BASEV >= 3 (architecture V4 and above).	RO	IM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MAC</td><td>1
(6)</td><td>Indicates if Multiply instructions are supported in the core or not. The instructions affected are listed in the following table.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Multiply instructions are not implemented.</td></tr><tr><td>1</td><td>Multiply instructions are implemented.</td></tr></table>This field will be removed for BASEV >= 3 (architecture V4 and above).</td><td>RO</td><td>IM</td></tr>
<tr><td>AUDIO</td><td>2
(8,7)</td><td>Indicates if AUDIO ISA extension support exists or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Audio extension is not supported.</td></tr><tr><td>1</td><td>32-bit Audio extension is supported.</td></tr><tr><td>2</td><td>24-bit Audio extension is supported.</td></tr><tr><td>3</td><td>Both 24-bit and 32-bit Audio extensions are supported.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>L2C</td><td>1
(9)</td><td>Indicates if a second-level I/D unified cache exists or not. This configuration affects the behaviors of the CCTL 1level and CCTL alevel operations.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>L2 unified cache is not implemented.</td></tr><tr><td>1</td><td>L2 unified cache is</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		implemented.		
RDREG or (GPR16)	1 (10)	Indicates if Reduced Register configuration option is selected or not. If it is selected, then the number of general purpose registers in this implementation is reduced to 16. 0: Reduced Register configuration is not implemented. 1: Reduced Register configuration is implemented.	RO	IM
ADR24	1 (11)	Indicates if a Reduced Address Space to 24-bit configuration is implemented or not. If it is implemented, the VA and PA address spaces are all reduced from 32 bits to 24 bits. 0: Reduced Address Space to 24-bit configuration is not implemented. 1: Reduced Address Space to 24-bit configuration is implemented.	RO	IM
INTLC	1 (12)	Indicates the interruption level configuration. 0: Interruption level 1, 2, and 3 are implemented. Level 3 is the overflow level. 1: Interruption level 1 and 2 are	RO	IM

Field Name	Bits	Description	Type	Reset
		implemented. Level 2 is the overflow level.		
BASEV	3 (15,13)	Indicates the version of Baseline instructions. Value: 0 — Baseline version 1 1 — Baseline version 2 (include version 1 + new instructions for version 2) 2 — Baseline version 3 3-7 — Reserved	RO	IM
NOD	1 (16)	Indicates if the Dx registers exist or not. If the Dx registers do not exist, the instructions using Dx registers should generate a Reserved Instruction exception. It is meaningful after and including baseline version 2. Value: 0 — Dx registers and the instructions involving Dx registers exist. 1 — Dx registers do not exist and the instructions involving Dx registers do not exist. When MISC.AUDIO is not 0 (i.e., Audio ISA extension is supported), this bit can never be set to 1. The instructions involved are: DIVS/DIV	RO	IM

Field Name	Bits	Description	Type	Reset
		MULT32/MULTS64/MULT64 MADD32/MADDS64/MADD64 MSUB32/MSUBS64/MSUB64 MFUSR Dx / MTUSR Dx		
IMV	1 (17)	Implementation-dependent register detection scheme version. Value: 0 — Version 0. Non-scalable scheme. Value: 1 — Version 1. Scalable scheme.	RO	IM
IMR	1 (18)	If (IMV == 0), this bit is always 0. If (IMV == 1), this bit indicates if the first implementation-dependent register of SR encoding {2, 15, 0} exists or not.	RO	IM
IFC	1 (19)	Indicates whether Inline Function Call (IFC) instructions are supported. Value: 0 — IFC is not supported. Value: 1 — IFC is supported.	RO	IM
MCU	1 (20)	Indicates whether this processor is a member of MCU family. In other word, this bit is set to one if the architecture version has a suffix “m” (e.g., V3m). Value: 0 — This processor does not belong to MCU family. Value: 1 — This processor belongs to MCU	RO	IM

Field Name	Bits	Description	Type	Reset
		family.		
SHADOW	3 (23,21)	Indicates whether shadow registers and what types of shadow registers are supported. 0: Shadow registers are not implemented. 1: One shadow stack pointer for privileged mode. 2-7: Reserved.	RO	IM
EIT	1 (24)	Indicates whether EX9.IT extension is supported or not. 0: EX9.IT extension is not supported. 1: EX9.IT extension is supported.	RO	IM
ULM	1 (25)	Indicates whether the Unified Local Memory (ULM) configuration is supported or not. 0: Original Instruction and Data Local Memory configuration is supported. 1: Unified Local Memory (ULM) configuration is supported.	RO	IM
PFT	1 (26)	Indicates if the Performance Throttling feature exists or not. Value | Meaning	RO	IM

Field Name	Bits	Description	Type	Reset
		0 — Performance throttling feature is not implemented. 1 — Performance throttling feature is implemented.		
HSP	1 (27)	Indicates if the HW Stack protection/recording feature exists or not. 0 — The HW stack protection feature is not implemented. 1 — The HW stack protection feature is implemented.	RO	IM
INTV	2 (29,28)	INTerrupt Version. It indicates the version number of the Interruption Architecture. 0 — Interruption architecture with the handling of imprecise and multiple next-precise exceptions defined as “Implementation dependent.” 1 — Interruption architecture with the handling of imprecise and multiple next-precise exceptions defined as controlled by PSW.GIE and PSW.PNP states. 2–3 — Reserved	RO	IM

Field Name	Bits	Description	Type	Reset
MSC_EXT	2 (31,30)	Miscellaneous Configuration Extension. This field indicates how many Misc. Configuration Registers exist. Value / Meaning: 0 — There is only 1 Misc. Configuration Register: 1. Misc. Configuration Register (MSC_CFG), SR Value: {0, 4, 0} 1 — There are 2 Misc. Configuration Registers: 1. Misc. Configuration Register (MSC_CFG), SR Value: {0, 4, 0} 2. Misc. Configuration Register 2 (MSC_CFG2), SR Value: {0, 4, 1} 2 — There are 3 Misc. Configuration Registers: 1. Misc. Configuration Register (MSC_CFG), SR Value: {0, 4, 0} 2. Misc. Configuration Register 2 (MSC_CFG2), SR Value: {0, 4, 1} 3. Misc. Configuration Register 3 (MSC_CFG3), SR Value: {0, 4, 2} 3 — There are 4 Misc. Configuration	RO	IM

Field Name	Bits	Description	Type	Reset
		Registers: 1. Misc. Configuration Register (MSC_CFG), SR Value: {0, 4, 0} 2. Misc. Configuration Register 2 (MSC_CFG2), SR Value: {0, 4, 1} 3. Misc. Configuration Register 3 (MSC_CFG3), SR Value: {0, 4, 2} 4. Misc. Configuration Register 4 (MSC_CFG4), SR Value: {0, 4, 3}		

10.2.6. Misc. Configuration Register 2

Mnemonic Name: cr7 (MSC_CFG2)

IM Requirement: Misc. configuration optional (MSC_CFG.MSC_EXT >= 1)

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {0, 4, 1}

The Misc. Configuration Register 2 exits if the configuration value at the field MSC_CFG. MSC_EXT is greater than or equal to 1.

31 ... 3	2	1	0
Reserved	ECC	TLB_ECC	

Field name	Bits	Description	Type	Reset
TLB_ECC	2 (1,0)	Configuration for parity/ECC support for TLB. Encoding: 0 = No parity/ECC support; 1 = Has parity support; 2 = Has ECC support; 3 = Reserved	RO	IM
ECC	1 (2)	Configuration for parity/ECC support. Encoding: 0 = No parity/ECC support; 1 = Has parity/ECC support The specific parity/ECC scheme used for each protected RAM is specified in the following fields: ICM_CFG.IC_ECC ICM_CFG.ILM_ECC	RO	IM

Field name	Bits	Description	Type	Reset
		DCM_CFG.DC_ECC DCM_CFG.DLM_ECC MSC_CFG2.TLB_ECC		
Reserved	29 (31,3)	Reserved	RAZWI	0

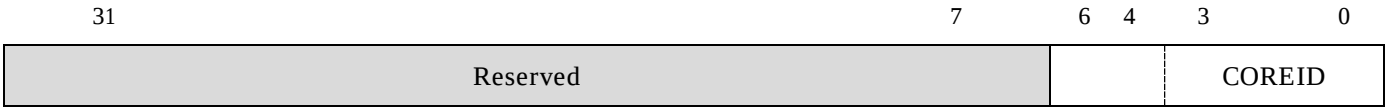
10.2.7. Core Identification Register

Mnemonic Name: cr5 (CORE_ID)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {0, 0, 1}



This register specifies the Core Identification Number which is commonly used in a multi-core system to distinguish different cores.

Field name	Bits	Description	Type	Reset
COREID	4 ~ 7 (6~3,0)	Core Identification Number. This is used to distinguish different processor cores in a multi-core system. An implementation should find ways to hardwire this field to different values for each core in a multi-core system. The minimum number of bits for this field is 4 bits while the maximum number of bits for this field can be up to 7 bits.	RO	IM
Reserved	25 (31,7)	Reserved	RAZWI	0

10.2.8. FPU and Coprocessor Existence Configuration Register

Mnemonic Name: cr6 (FUCOP_EXIST)

IM Requirement: COP/FPU extension (CPU_VER[3] == 1)

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {0, 5, 0}

This system register indicates the existence status of the coprocessors and the floating-point unit. Note that the floating-point unit existence is a combination of "CP0EX" and "CP0ISFPU" (i.e., CR6.CP0EX is 1 and CR6.CP0ISFPU is 1).

31	30 ... 18	17	16	15 ... 4	3	2	1	0
CP0ISFPU	Reserved	CPV		Reserved	CP3EX	CP2EX	CP1EX	CP0EX

Field Name	Bits	Description	Type	Reset
CP0EX	1 (0)	Coprocessor #0 existence status bit. Value: 0 — Coprocessor #0 does not exist. Any encountering of Coprocessor #0 instruction will cause a "reserved instruction" exception. Value: 1 — Coprocessor #0 exists.	RO	IM
CP1EX	1 (1)	Coprocessor #1 existence status bit. Value: 0 — Coprocessor #1 does not exist. Any encountering of coprocessor #1 instruction will cause a "reserved instruction" exception. Value: 1 — Coprocessor #1 exists.	RO	IM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CP2EX</td><td>1
(2)</td><td>Coprocessor #2 existence status bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #2 does not exist. Any encountering of coprocessor #2 instruction will cause a “reserved instruction” exception.</td></tr><tr><td>1</td><td>Coprocessor #2 exists.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>CP3EX</td><td>1
(3)</td><td>Coprocessor #3 existence status bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #3 does not exist. Any encountering of coprocessor #3 instruction will cause a “reserved instruction” exception.</td></tr><tr><td>1</td><td>Coprocessor #3 exists.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Reserved</td><td>12
(15,4)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>CPV</td><td>2
(17,16)</td><td>Coprocessor ISA extension version.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Version 1</td></tr><tr><td>1</td><td>Version 2</td></tr><tr><td>2-3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Reserved</td><td>13
(30,18)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>CP0ISFPU</td><td>1
(31)</td><td>Indicates if Coprocessor #0 is FPU or not when CP0EX is 1.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #0 is not FPU when</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		0 CP0EX is 1. 1 Coprocessor #0 is FPU when CP0EX is 1.		

10.3. Interruption System Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
ir0	PSW	1	0	0	169
ir1	IPSW	1	0	1	180
ir2	P_IPSW	1	0	2	182
ir3	IVB	1	1	1	183
ir4	EVA	1	2	1	186
ir5	P_EVA	1	2	2	188
ir6	ITYPE	1	3	1	190
ir7	P_ITYPE	1	3	2	201
ir8	MERR	1	4	1	203
ir9	IPC	1	5	1	204
ir10	P_IPC	1	5	2	205
ir11	OIPC	1	5	3	206
ir12	P_P0	1	6	2	207
ir13	P_P1	1	7	2	208
ir14	INT_MASK	1	8	0	209
ir15	INT_PEND	1	9	0	214
ir16	SP_USR	1	10	0	219
ir17	SP_PRIV	1	10	1	220
ir18	INT_PRI	1	11	0	221
ir19	INT_CTRL	1	1	2	224
ir20	Used by AndeStar Security Extension				
ir21					
ir22					

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
ir23					
ir24					
ir25					
ir26	INT_MASK2	1	8	1	212
ir27	INT_PEND2	1	9	1	217
ir28	INT_PRI2	1	11	1	223
ir29	INT_TRIGGER	1	9	4	226

10.3.1. Processor Status Word Register

Mnemonic Name: ir0 (PSW)

IM Requirement: Required

Access Mode: Superuser (few bits for user)

SR Encoding {Major, Minor, Extension}: {1, 0, 0}

The Processor Status Word register contains several fields which control the global masking behavior for interrupts, the MMU translation modes, the current processor operation mode, data endianness control, and indicate the current interruption stack level, and machine error status.

13	12	11	10	9	8	7	6	5	4 3	2 1	0
AEN	DRBE	HSS	DEX	DME	IME	DT	IT	BE	POM	INTL	GIE

31 23	22	21	20	19	18 16	15	14
Reserved	PNP	PFT_EN	OV	Reserved	CPL	IFCON	WBNA

Field Name	Bits	Description	Type	Reset
GIE (Global Interrupt Enable)	1 (0)	Controls if Interrupt is enabled or not. Note that the Non-Maskable Interrupt (NMI) cannot be disabled by this bit. Imprecise exception processing is also controlled by this bit starting from the Interruption Architecture Version 1 (MSC_CFG.INTV >= 1). Value / Meaning 0 – Interrupt is disabled. 1 – Interrupt is enabled.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>INTL (Interruption Stack Level)</td><td>2
(2,1)</td><td>Controls interruption behaviors. When interruption happens in interruption stack level 2, all the interruption resources are not updated except that the interruption stack level is being updated to 3.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No interruption</td></tr><tr><td>1</td><td>Interruption stack level 1</td></tr><tr><td>2</td><td>Interruption stack level 2</td></tr><tr><td>3</td><td>Interruption stack level 3</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>POM</td><td>2
(4,3)</td><td>Processor Operation Mode. This bit specifies the operation mode of the program. User mode can only access to GPRs, user special registers and user instructions. In contrast, superuser mode has access to all resources including system registers and privileged instructions.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>User</td></tr><tr><td>1</td><td>Superuser</td></tr><tr><td>2</td><td>Reserved</td></tr><tr><td>3</td><td>Reserved</td></tr></table>Note that writing a reserved value into this POM field causes a “Reserved Value” exception.</td><td>RW</td><td>1</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>BE</td><td>1
(5)</td><td>Endian mode for data memory access. This bit is user modifiable using a SETEND instruction.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Little Endian</td></tr><tr><td>1</td><td>Big Endian</td></tr></table></td><td>RW</td><td>cr3.DE (MMU Config)</td></tr>
<tr><td>IT</td><td>1
(6)</td><td>Controls instruction address translation. On interruption stack level transitions from 0 to 1 and from 1 to 2, it gets changed to 0.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Instruction address translation is disabled.</td></tr><tr><td>1</td><td>Instruction address translation is enabled.</td></tr></table>Software Note: When changing the state of this bit using a MTSR instruction, the instruction pages which contain any instruction following the MTSR instruction until the ISB instruction need to be identity-mapped (VPN == PPN) in the page table.

This bit becomes Read-As-Zero-Write-Ignored (RAZWI) if neither MMU nor MPU is supported.</td><td>RW/ RAZWI</td><td>0</td></tr>
<tr><td>DT</td><td>1
(7)</td><td>Controls data address translation. On interruption stack level transitions from 0 to 1 and from 1 to 2, it gets changed to 0.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Data address translation is disabled.</td></tr><tr><td>1</td><td>Data address translation is</td></tr></table></td><td>RW/ RAZWI</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		<table><tr><td></td><td>enabled.</td></tr></table> Software Note: When changing the state of this bit using a MTSR instruction, the data pages which are referenced by any load/store instruction following the MTSR instruction until the ISB instruction need to be identity- mapped (VPN == PPN) in the page table. This bit becomes Read-As-Zero-Write-Ignored (RAZWI) if neither MMU nor MPU is supported.		
IME	1 (8)	Instruction Machine Error flag. It indicates an exception occurs at the instruction cache or instruction local memory (ILM). <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No Instruction Machine Error exception.</td></tr><tr><td>1</td><td>An Instruction Machine Error occurred. When it is set, all instruction accesses bypass all instruction caches inside the CPU and are directed out to the bus. These accesses going out to the bus still carry the original cacheability attributes without modifications. Note that CCTL instructions can still access caches in this case.</td></tr></table> It will be set by the following Instruction Machine Error exceptions:	RW	0

Field Name	Bits	Description	Type	Reset
		Instruction cache locking error Instruction parity/ECC error Instruction TLB locking error Instruction TLB multiple hit Note that this bit must be cleared by the exception handler to exit the machine error state and enable the instruction cache.		
DME	1 (9)	Data Machine Error flag. It indicates an exception occurs at the data cache or data local memory (DLM). 0: No Data Machine Error exception. 1: A Data Machine Error occurred. When it is set, all data accesses bypass all data caches inside the CPU and are directed out to the bus. These accesses going out to the bus still carry the original cacheability attributes without modifications. Note that CCTL instructions can still access caches in this case. It will be set by the following Data Machine Error exceptions: Data cache locking error Data parity/ECC error Data TLB locking error	RW	0

Field Name	Bits	Description	Type	Reset
		Data TLB multiple hit Note that this bit must be cleared by the exception handler to exit the machine error state and enable the data cache.		
DEX	1 (10)	Debug Exception. This bit will be set when the processor enters the debug exception entry point.	RW	0
HSS	1 (11)	Hardware Single Stepping. If hardware single stepping is enabled, the processor will enter the debug exception after an instruction is complete. If this bit is set, it will be changed to the value of DSSIM of the Interrupt Masking Register on interruption entrance. Value: 0 — Hardware single stepping is OFF. On interruption entrance, this bit remains at 0. Value: 1 — Hardware single stepping is ON. On interruption entrance, this bit is set to the value of DSSIM of the Interrupt Masking Register.	RW	0
DRBE	1 (12)	Device Register Endian mode for device register space access when MMU_CTL.DREE is asserted. This Device Register Endian mode allows uncacheable/un-coalesable device registers to have a different endian type than that of the other attribute space. Value: 0 — Little Endian	RW	cr3.DRDE

Field Name	Bits	Description	Type	Reset
		1 — Big Endian		
AEN	1 (13)	Audio ISA special features enable control. This bit exists only if audio ISA extension is supported. The audio special features include: zero-overhead loop mechanism and BLW24. If audio ISA extension is supported, system software needs to turn on this bit in order for user programs to use these features. This bit will be set to 0 automatically on interruption entry to a non-highest level. For interruption entry to the highest level, this bit will not be updated, but will have the same AEN == 0 behavior. 0 — Audio special features are disabled. 1 — Audio special features are enabled.	RW	0
WBNA	1 (14)	Changing write-back, write-allocation memory to write-back, no-write-allocation memory. 0 — Write-back memory with write-allocation policy. 1 — Write-back memory with no-write-allocation policy.	RW	0
IFCON	1 (15)	Indicates if the program is executed in an IFC sequence. This bit is user modifiable using IFC related instructions.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CPL</td><td>3
(18,16)</td><td>Current Priority Level (CPL) indicates the priority level of the interrupt currently being serviced. Only interrupts with a priority level higher (numerically smaller) than this CPL are allowed to replace the current interrupt. This field is valid only when programmable priority is supported, so it becomes RAZWI in fixed priority configuration.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No interrupts are allowed.</td></tr>
<tr><td>1</td><td>Only allows interrupts with priority 0.</td></tr>
<tr><td>2</td><td>Only allows interrupts with priority 0 or 1.</td></tr>
<tr><td>3</td><td>Only allows interrupts with priority 0, 1, or 2.</td></tr>
<tr><td>4</td><td>Only allows interrupts with priority 0 to 3.</td></tr>
<tr><td>5</td><td>Only allows interrupts with priority 0 to 4.</td></tr>
<tr><td>6</td><td>Only allows interrupts with priority 0 to 5.</td></tr>
<tr><td>7</td><td>Allows interrupts with any priority (0, 1, 2, 3, 4, 5 or 6).</td></tr>
</table></td><td>RW</td><td>7</td></tr>
<tr><td>Reserved</td><td>1
(19)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
OV	1 (20)	Overflow flag for saturation arithmetic instructions. This flag is set if an overflow occurs. Encoding: 0 — Meaning: No overflow Encoding: 1 — Meaning: Saturation arithmetic overflow occurred.	RW	0
PFT_EN	1 (21)	Enable performance throttling. When throttling is enabled, the processor executes instructions at the performance level specified in PFT_CTL.T_LEVEL. This filed is RAZWI if the performance throttling feature is not supported.	RW/ RAZWI	0
PNP	1 (22)	Pending Next-Precise exceptions. This bit only exists starting from the Interruption Architecture Version 1 (MSC_CFG.INTV >= 1). Typically, it is set by hardware automatically to temporarily block additional pending Next-Precise (NP) exceptions. When this bit is set, it implies that multiple NP exceptions have occurred simultaneously but only the highest priority one is serviced. The rest of NP exceptions are blocked (become pending) until this bit is cleared and, at that time, the next highest priority one will be serviced. When this bit is set and additional interruption occurs, it is sticky and remains set. When this bit is cleared, normally through an update from IPSW on IRET instruction, the pending NP exceptions become active for	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>processing.

This bit is normally cleared through an update from IPSW upon executing an IRET instruction. Clearing this bit will immediately trigger the next highest pending NP exception. To cancel all pending NP exceptions, all the pending bits at INT_PEND (e.g., NP_WPP) should be cleared.

The above descriptions can be summarized in more concise rules as shown below:
On interruption entry to non-maximum level:
PNP ← (Pending NP exception exists) ? 1 : PNP
On IRET instruction:
PSW.PNP ← IPSW.PNP
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>No next-precise exceptions are pending for processing.</td></tr><tr><td>1</td><td>At least one next-precise exception is pending for processing.</td></tr></table>Although this bit is not updated when the processor enters the maximum interruption level, it is still used to determine the masking behavior of any pending NP exceptions. In contrast, at the maximum level, other bits in PSW (such as GIE or IT) no longer have control and are replaced by fixed default behaviors instead (as described in section 4.2.2).</td><td></td><td></td></tr>
</table>

Field Name	Bits	Description	Type	Reset
Reserved	9 (31,23)	Reserved	RAZWI	0

10.3.2. Interruption PSW Register

Mnemonic Name: ir1 (IPSW)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 0, 1}

This register is the shadow stack register of the PSW register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It has the same format as PSW.

It is a RW type register with the reset value defined as DC (Don't Care). And writing a reserved value into the corresponding POM field causes a "Reserved Value" exception.

In V3 architecture (MSC_CFG.BASEV is 2), a new bit called SP_ADJ is added in IPSW but this bit does not exist in PSW. This bit is used to adjust stack pointer to 8-byte alignment. Please refer to section 4.11 for more details.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SP_ADJ</td><td>1
(19)</td><td>Stack Pointer Adjustment. This bit indicates whether the stack pointer is adjusted to be 8-byte aligned or not. This bit is added since V3 architecture (MSC_CFG.BASEV is 2) and does not exist in PSW.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No stack adjustment is performed since it is already 8-byte aligned.</td></tr>
<tr><td>1</td><td>Stack adjustment is performed since it is not 8-byte aligned. The stack pointer is deducted by 4 when entering interrupts and exceptions; it is later added by 4</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		when leaving interrupts and exceptions.		

10.3.3. Previous IPSW Register

Mnemonic Name: ir2 (P_IPSW)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 0, 2}

This register is the shadow stack register of the IPSW register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It has the same format as IPSW.

It is a RW type register with the reset value defined as DC (Don't Care). And writing a reserved value into the corresponding POM field causes a "Reserved Value" exception.

10.3.4. Interruption Vector Base Register

Mnemonic Name: ir3 (IVB)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 1, 1}

The IVB register is the base physical address of the interruption vector table which anchors the interruption entry points for the exception and interruption handlers. This register also records various configurations and properties for interrupts. Its format is as follows:

31 16	15 14	13	12 11	10 4	3 1	0
IVBASE(31,16)	ESZ	EVIC	IVIC_VER	Reserved	NIVIC	PROG_PRI_LVL

Field Name	Bits	Description	Type	Reset
PROG_PRI_LVL	1 (0)	Programmable Priority Level. This bit indicates whether interrupts have programmable priority or fixed priority. Value: 0 — Fixed priority; 1 — Programmable priority	RO	IM
NIVIC	3 (3,1)	Number of input for Internal Vector Interrupt Controller. Indicates the number of interrupt input sources implemented at IVIC mode. Value: 0 — 6 interrupt input sources; 1 — 2 interrupt input sources; 2 — 10 interrupt input sources; 3 — 16 interrupt input sources	RO	IM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><td>4</td><td>24 interrupt input sources</td></tr><tr><td>5</td><td>32 interrupt input sources</td></tr><tr><td>6-7</td><td>Reserved</td></tr></table></td><td></td><td></td></tr>
<tr><td>Reserved</td><td>7
(10,4)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>IVIC_VER</td><td>2
(12,11)</td><td>Internal Vector Interrupt Controller (IVIC) Version. These bits indicate which version of internal interrupt controller is supported. They determine if related system registers exist and can be referred for software programming.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>■ Only support level-triggered interrupts.
■ Only support up to 16 interrupt sources.</td></tr><tr><td>1</td><td>■ Support both level-triggered and edge-triggered interrupts.
■ Support up to 32 interrupt sources.
■ The following system registers always exist:
INT_MASK2 (ir26)
INT_PEND2 (ir27)
INT_PRI (ir18)
INT_CTRL (ir19)
INT_PRI2 (ir28)
INT_TRIGGER (ir29)</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><td>2</td><td>Reserved</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td></td><td></td></tr>
<tr><td>EVIC</td><td>1
(13)</td><td>Configures the proper Vector Interrupt Controller mode used in the AndesCore.<table><tr><td>Value</td><td>Meaning</td></tr><tr><td>0</td><td>Internal Vector Interrupt Controller mode.</td></tr><tr><td>1</td><td>External Vector Interrupt Controller mode.</td></tr></table></td><td>RW/ RO</td><td>IM</td></tr>
<tr><td>ESZ</td><td>2
(15,14)</td><td>Defines the size of each vector entry.<table><tr><td>Value</td><td>Meaning</td></tr><tr><td>0</td><td>4 Byte</td></tr><tr><td>1</td><td>16 Byte</td></tr><tr><td>2</td><td>64 Byte</td></tr><tr><td>3</td><td>256 Byte</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>IVBASE</td><td>16
(31,16)</td><td>Defines the high 16 bits of the base physical address of the interruption vector table. The table has to be 64KB aligned.

Note that this field becomes read-only for MCU family (MSC_CFG.MCU == 1) to reduce gate count. It can still be set through pins from input interface.</td><td>RW/ RO</td><td>IM</td></tr>
</table>

10.3.5. Exception Virtual Address Register

Mnemonic Name: ir4 (EVA)

IM Requirement: Required for non-MCU family (MSC_CFG.MCU == 0)

Access Mode: Superuser

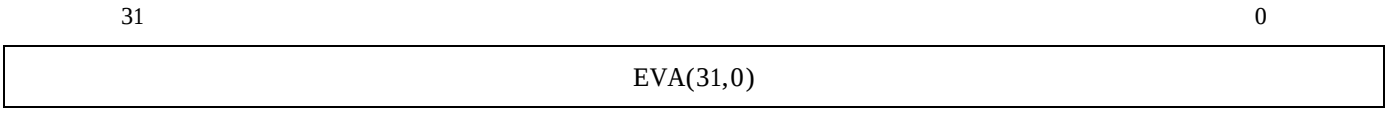
SR Encoding {Major, Minor, Extension}: {1, 2, 1}

The EVA register will be updated with the exception virtual address information needed by the interruption handler when certain exception happens during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It will also be updated from the P_EVA register when executing a "return from interruption" (IRET) instruction within interruption stack level 2. It is used by the exception handler routine to fix the exception problem, or to debug and log the problem if the exception cannot be solved immediately by the handler.

There are some special cases where EVA is not updated: warm resets, debug related exceptions, and suppressed imprecise exceptions (e.g., imprecise bus error) during debug mode.

When debugging, the debugger is responsible for backing up EVA at the beginning and restoring EVA before it exits at the end. In the meantime, EVA is updated by all suppressed exceptions other than imprecise ones. The debugger should also back up EVA whenever it sees an exception flag to make room for further potential exceptions.

Its register format is as follows:



Field Name	Bits	Description	Type	Reset
EVA	32 (31,0)	The virtual address which causes the exception. For instruction fetch related exceptions, this register records part of the same virtual address captured in	RW	DC

Field Name	Bits	Description	Type	Reset
		the IPC register. For data access related exceptions, this register records the data access virtual address which is different from the virtual address in the IPC register. Detailed exceptions and their EVA values are listed in Table 20. For other exceptions not listed in the table, the EVA update behavior and its update value will be implementation-dependent.		

The EVA register will be updated by hardware for the following exceptions that is related to memory access or memory access hardware structures during interruption stack level transitions 0 → 1 and 1 → 2:

Table 20. Exceptions and EVA values

Exceptions updating EVA	EVA value
ITLB fill	4KB-aligned instruction fetch virtual address (i.e., lower 12 bits 0)
ITLB VLPT miss	4KB-aligned instruction fetch virtual address (i.e., lower 12 bits 0)
ITLB misc. (all)	4KB-aligned instruction fetch virtual address (i.e., lower 12 bits 0)
I access PTE not present (all)	4KB-aligned instruction fetch virtual address (i.e., lower 12 bits 0)
DTLB fill	Full 32-bit data access virtual address
DTLB VLPT miss	Full 32-bit data access virtual address
DTLB misc. (all)	Full 32-bit data access virtual address
D access PTE not present (all)	Full 32-bit data access virtual address
Branch target alignment	Full 32-bit virtual address of misaligned target

Exceptions updating EVA	EVA value
Alignment check	Full 32-bit virtual address of the instruction which generates the misaligned address
Precise bus error	Full 32-bit data access virtual address
TLB locking error generated by a TLBOP instruction (in machine error entry point)	4KB-aligned virtual page number (i.e., lower 12-bit is 0)
Machine error (fetch/load/store related, thus excluding unimplemented page size error)	Full 32-bit instruction fetch or data access virtual address
Illegal parallel memory accesses	Full 32-bit data access virtual address of the first address operand
Precise parity/ECC error on instruction cache, data cache, or TLB	Full 32-bit data access virtual address
Precise data address watchpoint	Full 32-bit watched virtual address
Precise data value watchpoint	Full 32-bit watched virtual address

The EVA is not updated for imprecise exceptions due to lack of sufficient information.

10.3.6. Previous EVA Register

Mnemonic Name: ir5 (P_EVA)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 2, 2}

This register is the shadow stack register of the EVA register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It has the same format as EVA register.

It is a RW type register with the reset value defined as DC (Don't Care).

10.3.7. Interruption Type Register

Mnemonic Name: ir6 (ITYPE)

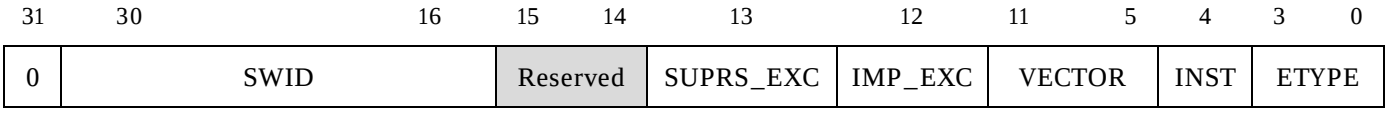
IM Requirement: Required

Access Mode: Superuser

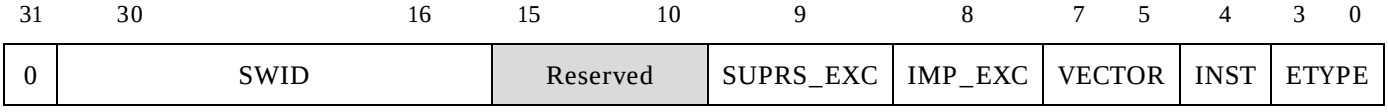
SR Encoding {Major, Minor, Extension}: {1, 3, 1}

This register is updated when exceptions or interrupts occur. It records detailed information needed by the interruption handler when interruptions occur at interruption stack level 0 and level 1. It is also restored from the P_ITYPE register when executing a "return from interruption" (IRET) instruction at the interruption stack level 1 and level 2. Its register format is as follows:

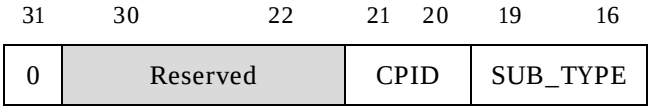
For non-MCU family (MSC_CFG.MCU == 0):



For MCU family (MSC_CFG.MCU == 1):



For coprocessor exceptions and arithmetic exceptions (in the general exception vector entry point), the upper half is redefined as follows:



Field Name	Bits	Description	Type	Reset
ETYPE	4 (3,0)	Event type information. Its definition is based on the exception entry point. Detailed definition will be listed in the following tables. **Entry Point Definition**	RW	0

Field Name	Bits	Description	Type	Reset
		Reset/NMI Table 21 PTE not present Table 22 TLB misc Table 23 Machine error Table 24 Debug Table 25 General exception Table 26 Syscall Table 27 Interrupts Table 28		
INST	1 (4)	Indicates if an exception is caused by an instruction fetch or a data memory access for the following exceptions: TLB fill TLB VLPT miss TLB read protection TLB write protection TLB non-executable page TLB page modified TLB Access bit PTE not present (all) Reserved PTE Attribute Alignment check Branch target alignment Machine error Precise bus error Imprecise bus error Nonexistent memory address MPZIU Control Cache locking error	RW	0

Field Name	Bits	Description	Type	Reset
		TLB locking error TLB multiple hit Parity/ECC error All other exceptions not in the above table should have the INST field of the ITYPE register set to 0. Value: 0 — Meaning: Exception is not caused by an instruction fetch access (e.g., a data memory access). Value: 1 — Meaning: Exception is caused by an instruction fetch access.		
VECTOR	7 (11,5) or 3 (7,5) for MCU family	Vector indicates the entry point location of exceptions and interrupts. This vector is used for two occasions: debugger to identify suppressed exceptions and code sharing of handlers (for non-MCU family). This field is automatically updated by hardware when an exception or interrupt occurs. In addition, it is also updated even if an exception is suppressed during debug mode. The debugger can use this vector to identify which exceptions were suppressed during debug mode and perform proper handling afterwards. For non-MCU family (MSC_CFG.MCU == 0), the entry points of interrupts are also recorded in the vector to help reduce code size. Since the vector records the entry point, no software is needed to	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>record this information and thus the size of each vector entry can be reduced to the minimum of 4 bytes. Handlers can jump directly from the vector entry to a common code section to eliminate duplication. At the end of the common code section, this vector can then be used to branch out to individual handler.

For non-MCU family (MSC_CFG.MCU == 0):
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Reset/NMI</td></tr>
<tr><td>1</td><td>TLB fill</td></tr>
<tr><td>2</td><td>PTE not present</td></tr>
<tr><td>3</td><td>TLB misc</td></tr>
<tr><td>4</td><td>TLB VLPT miss</td></tr>
<tr><td>5</td><td>Machine Error</td></tr>
<tr><td>6</td><td>Debug related</td></tr>
<tr><td>7</td><td>General exception</td></tr>
<tr><td>8</td><td>Syscall</td></tr>
<tr><td>9-72</td><td>HW Interrupt 0-63</td></tr>
</table>
For MCU family (MSC_CFG.MCU == 1):
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Reset/NMI</td></tr>
<tr><td>1</td><td>TLB fill</td></tr>
<tr><td>2</td><td>PTE not present</td></tr>
<tr><td>3</td><td>TLB misc</td></tr>
<tr><td>4</td><td>TLB VLPT miss</td></tr>
<tr><td>5</td><td>Machine Error</td></tr>
</table></td><td></td><td></td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>6</td><td>Debug related</td></tr>
<tr><td>7</td><td>General exception</td></tr>
</table>
</td><td></td><td></td></tr>
<tr><td>IMP_EXC</td><td>1
(12)
or
1
(8)
for MCU
family</td><td>Imprecise Exception flag for suppression during host-mode debugging. This flag indicates if an imprecise exception occurs during the debug mode. More specifically, this flag is set by hardware if an imprecise exception occurs under the condition of both PSW.DEX and EDM_CTL.DEH_SEL (host-mode) are 1; this flag is cleared by hardware when initially PSW.DEX becomes 1, or by the debugger when it finishes servicing an imprecise exception. In particular, the IMP_EXC is still set to 1 if PSW.DEX and imprecise exception both occur at the same cycle.

Note that, for each particular exception, only one flag (either SUPRS_EXC or IMP_EXC) is but never both together.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No suppressed imprecise exception.</td></tr>
<tr><td>1</td><td>A suppressed imprecise exception occurred during host-mode debugging and needs service. Note that ETYPE and INST fields are not modified in this case.</td></tr>
</table>
</td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SUPRS_EXC</td><td>1
(13)
or
1
(9)
for MCU family</td><td>Suppressed Exception flag for precise and next-precise exceptions during host-mode debugging. This flag indicates if a suppressed precise or next-precise exception occurs during host-mode debugging. This flag is set by hardware if a precise/next-precise exception occurs under the condition of both PSW.DEX and EDM_CTL.DEH_SEL (host-mode) are 1; this flag is cleared by hardware when PSW.DEX initially becomes 1, or by the debugger when it initially saves ETYPE or finishes servicing a suppressed exception.

Note that, for each particular exception, only one flag (either SUPRS_EXC or IMP_EXC) is but never both together.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No suppressed precise or next-precise exception.</td></tr>
<tr><td>1</td><td>A suppressed precise or next-precise exception occurred during host-mode debugging and needs service.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>2
(15,14)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>SWID</td><td>15
(30,16)</td><td>Software ID. It records a SW (user) specified ID to be used by SYSCALL and TRAP instructions for processing the corresponding exceptions.</td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>0</td><td>1
(31)</td><td>Zero</td><td>RO</td><td>0</td></tr>
<tr><td>SUB_TYPE</td><td>4
(19,16)</td><td>1. Indicates arithmetic exception type when an arithmetic exception occurs. This exception sub-type encoding exists only if (MSC_CFG.BASEV >= 2). The encoding is as follows:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>INT Divide by Zero (for DIV, DIVS)</td></tr>
<tr><td>2</td><td>Integer Overflow (for DIVS)</td></tr>
<tr><td>3-15</td><td>Reserved</td></tr>
</table>
2. Indicates the coprocessor exception type when a coprocessor exception occurs. The encoding is as follows:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>Coprocessor Disabled Exception. The corresponding coprocessor number is shown in the CPID field.
This exception is often used to save power. Specifically, it is used by the exception handler to detect the existence of coprocessor instructions (e.g., floating-point instructions) and the handler only enables the</td></tr>
</table></td><td>RW</td><td>IM</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		coprocessor as needed to save power.		
		2 — Coprocessor exception. Please check coprocessor exception status for details. The corresponding coprocessor number is shown in the CPID field.		
		3 — Audio extension disabled exception.		
		4 — Audio extension zero-overhead loop error.		
		5 — Coprocessor imprecise exception. Please check coprocessor imprecise exception status for details. The corresponding coprocessor number is shown in the CPID field.		
		6 — Reserved		
		7 — Andes Custom Extension (ACE) disabled exception.		
		8-15 — Andes Custom Extension (ACE) exceptions. It is formed by concatenating (1, status[3:1]).		
CPID	2 (21,20)	Indicates the ID number of the coprocessor which generates the exception.	RW	IM

Detailed Definitions for Event Type (ETYPE)

For Reset/NMI exception entry point, the ETYPE is defined as follows:

Table 21. Reset/NMI exception ETYPE definition

Value	Event (Qualified with INST field)
0	Cold reset
1	Warm reset
2	NMI
3-15	Reserved

For PTE not present exception entry point, the ETYPE is defined as follows:

Table 22. PTE not present exception ETYPE definition

Value	Event (Qualified with INST field)
0	Non-leaf PTE not present (+I/D bit)
1	Leaf PTE not present (+I/D bit)
2-15	Reserved

For TLB misc. exception entry point, the ETYPE is defined as follows:

Table 23. TLB misc. exception ETYPE definition

Value	Event (Qualified with INST field)
0	TLB read protection (data)
1	TLB write protection (data)
2	TLB non-executable page (Inst)
3	TLB page modified (data)
4	TLB Access bit (+I/D bit)
5	Reserved PTE Attribute (+I/D bit)

Value	Event (Qualified with INST field)
6-15	Reserved

For Machine Error exception entry point, the ETYPE is defined as follows:

Table 24. Machine error exception ETYPE definition

Value	Event (Qualified with INST field)
0	Cache locking error (+I/D bit)
1	TLB locking error (+I/D bit)
2	TLB multiple hit (+I/D bit)
3	Parity/ECC error (+I/D bit)
4	Unimplemented page size error
5	Illegal parallel memory accesses (Audio extension)
6	Local memory base setup error (next-precise)
7-15	Reserved

For Debug exception entry point, the ETYPE is defined as follows:

Table 25. Debug exception ETYPE definition

Value	Event
0	BREAK
1	BREAK16
2	Instruction breakpoint
3	Data address watchpoint (precise)
4	Data value watchpoint (precise)
5	Data value watchpoint (imprecise)
6	Debug interrupt
7	Hardware single stepping

Value	Event
8	Data address watchpoint (next-precise)
9	Data value watchpoint (next-precise)
10	Load/store instruction global stop
11-15	Reserved

For General Exception entry point, the ETYPE is defined as follows:

Table 26. General exception ETYPE definition

Value	Event (Qualified with INST field)
0	Alignment check (+I/D bit) or branch target alignment (when MSC_CFG.BASEV == 2, baseline version 3)
1	Reserved instruction
2	Trap
3	Arithmetic
4	Precise bus error (+I/D bit)
5	Imprecise bus error (+I/D bit)
6	Coprocessor
7	Privileged instruction
8	Reserved value
9	Nonexistent memory address (+I/D bit)
10	MPZIU Control (+I/D bit)
11	Next-precise stack overflow: INST=0: stack overflow, INST=1: stack underflow
12-15	Reserved

For Syscall entry point, the ETYPE is defined as follows:

Table 27. Syscall exception ETYPE definition

Value	Event
0-7	Reserved
8	Syscall
9-15	Reserved

For Interrupt entry points, the ETYPE is defined as follows:

Table 28. Interrupt ETYPE definition

Value	Event
0-2	Reserved
3	Interrupt
4-15	Reserved

10.3.8. Previous ITYPE Register

Mnemonic Name: ir7 (P_ITYPE)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 3, 2}

This register is the shadow stack register of the ITYPE register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It has the same format as ITYPE register.

It is a RW type register with the reset value defined as DC (Don't Care).

10.3.9. Machine Error Log Register

Mnemonic Name: ir8 (MERR)

IM Requirement: Required

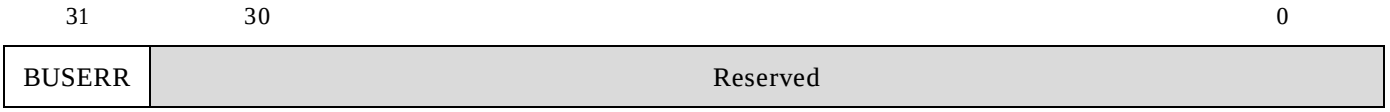
Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 4, 1}

This register records error information needed by the exception handlers when exceptions occur (e.g., General Exception or Machine Error Exception).

No shadow stack register is specified for this register since these errors are infrequent and we do not expect nested errors to occur.

The format and encoding of this register is as follows:



Field Name	Bits	Description	Type	Reset
Reserved	31 (30,0)	Reserved	RAZWI	0
BUSERR	1 (31)	Indicates that a bus error occurred. The exact triggering conditions are implementation dependent which are specified in the datasheet.	W1C	0

10.3.10. Interruption Program Counter Register

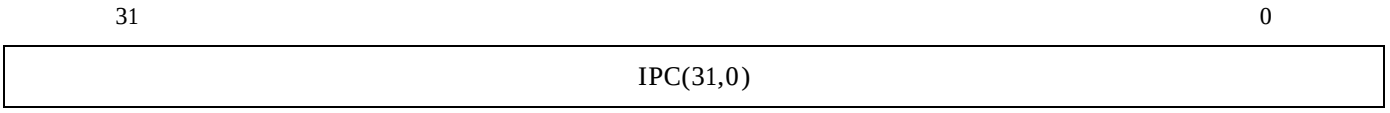
Mnemonic Name: ir9 (IPC)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 5, 1}

This register is the shadow stack register of the Program Counter. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. And its format is the same as the Program Counter as follows:



It is a RW type register with the reset value defined as DC (Don't Care).

10.3.11. Previous IPC Register

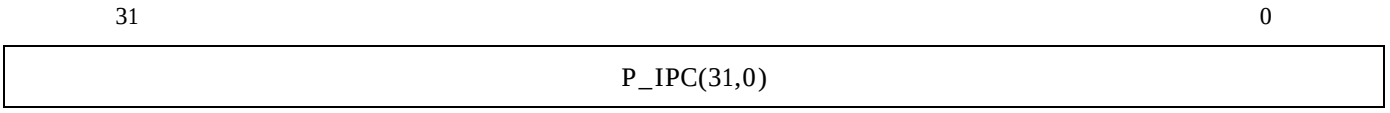
Mnemonic Name: ir10 (P_IPC)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 5, 2}

This register is the shadow stack register of the IPC register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. It has the same format as IPC as follows:



It is a RW type register with the reset value defined as DC (Don't Care).

10.3.12. Overflow Interruption Program Counter Register

Mnemonic Name: ir11 (OIPC)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 5, 3}

This register is the special shadow stack register of the Program Counter. It is updated by hardware during interruption stack level transition 2 → 3 (1 → 2 for the configuration where the maximum interruption stack level is 2, or MSC_CFG.INTLC is 1). It has the same format as the Program Counter.

It is a RW type register with the reset value defined as DC (Don't Care).

10.3.13. Previous P0 Register

Mnemonic Name: ir12 (P_P0)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 6, 2}

This register is the shadow stack register of the P0 (R26) register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. However, it is not updated for interruption stack level transition 0 → 0. It has the same format as a General Purpose Register.

It is a RW type register with the reset value defined as DC (Don't Care).

10.3.14. Previous P1 Register

Mnemonic Name: ir13 (P_P1)

IM Requirement: Interruption level configuration optional (MSC_CFG.INTLC != 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 7, 2}

This register is the shadow stack register of the P1 (R27) register. It is updated during interruption stack level transitions 0 → 1 and 1 → 2 by hardware. However, it is not updated for interruption stack level transition 0 → 0. It has the same format as a General Purpose Register.

It is a RW type register with the reset value defined as DC (Don't Care).

10.3.15. Interruption Masking Register

Mnemonic Name: ir14 (INT_MASK)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 8, 0}

The Interruption Masking Register is used to mask the software interrupt and hardware interrupts in the Internal Vector Interrupt Controller mode. It also controls if an exception is generated on Divide-by-Zero condition for integer divide instructions.

31	30	29	28	27 – 17	16	15 – 6	5 – 0
DSSIM	IDIVZE	ALZ	IMPE	Reserved	SIM	H15IM – H6IM (IVIC)	H5IM – H0IM (IVIC)

Field Name	Bits	Description	Type	Reset
H5IM – H0IM	6 (5,0)	Hardware Interrupt X Mask bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows: (H5,H4,H3,H2,H1,H0) == (5,4,3,2,1,0). Each bit is defined below. Value: 0 – The corresponding hardware interrupt is disabled. Value: 1 – The corresponding hardware interrupt is enabled.	RW	0 for all bits
H15IM – H6IM	10 (15,6)	Hardware Interrupt X Mask bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows:	RW	0 for all bits

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>(H15,H14,H13,H12,H11,H10,H9,H8,H7,H6) == (15,14,13,12,11,10,9,8,7,6). Each bit is defined below.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The corresponding hardware interrupt is disabled.</td></tr>
<tr><td>1</td><td>The corresponding hardware interrupt is enabled.</td></tr>
</table></td><td></td><td></td></tr>
<tr><td>SIM</td><td>1
(16)</td><td>Software Interrupt Mask bit. This bit is used in the Internal Vector Interrupt Controller mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Software interrupt is disabled.</td></tr>
<tr><td>1</td><td>Software interrupt is enabled.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>11
(27,17)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>IMPE</td><td>1
(28)</td><td>An imprecise exception enable control. When this bit is asserted, an imprecise exception can happen without checking the state of PSW.GIE. When this bit is deasserted, an imprecise exception can happen only when PSW.GIE is asserted.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>An imprecise exception is blocked by PSW.GIE.</td></tr>
<tr><td>1</td><td>An imprecise exception is not blocked by PSW.GIE.</td></tr>
</table></td><td>RW</td><td>1</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ALZ</td><td>1
(29)</td><td>All zero opcode (LBI R0,[R0+0]) reserved instruction exception masking control. This is a baseline version 2 instruction set special behavior and is mainly used for the debugging purpose.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>All zero opcode reserved instruction exception is disabled.</td></tr><tr><td>1</td><td>All zero opcode reserved instruction exception is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>IDIVZE</td><td>1
(30)</td><td>Controls if the Arithmetic exception is enabled or not for “Divide-By-Zero” condition.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The Arithmetic exception is disabled.</td></tr><tr><td>1</td><td>The Arithmetic exception is enabled.</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>DSSIM</td><td>1
(31)</td><td>Default Single Stepping Interruption Mask. This bit determines the value of PSW.HSS on interruption entrance when PSW.HSS is set (i.e., hardware single-stepping is on).
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>If (PSW.HSS == 1) then PSW.HSS ← 0 (OFF) on interruption entrance.</td></tr><tr><td>1</td><td>If (PSW.HSS == 1) then PSW.HSS ← 1 (ON) on interruption entrance.</td></tr></table></td><td>RW</td><td>1</td></tr>
</table>

10.3.16. Interruption Masking Register 2

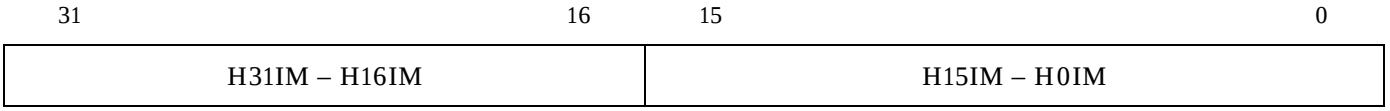
Mnemonic Name: ir26 (INT_MASK2)

IM Requirement: Internal vector interrupt controller optional (IVB.IVIC_VER >= 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 8, 1}

The Interruption Masking Register 2 is used to mask the hardware interrupts in the Internal Vector Interrupt Controller mode. Note that, for backward compatibility and ease of programming, its lower 16 bits (15, 0) are aliased and identical to the lower 16 bits of Interruption Masking Register (ir14 or INT_MASK).



Field Name	Bits	Description	Type	Reset
H15IM – H0IM	16 (15,0)	Hardware Interrupt X Mask bits. Note that, for backward compatibility and ease of programming, these 16 bits are aliased and identical to the lower 16 bits of Interruption Masking Register (ir14 or INT_MASK). They are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows: (H15IM, H14IM, H13IM, H12IM, H11IM, H10IM, H9IM, H8IM, H7IM, H6IM, H5IM, H4IM, H3IM, H2IM, H1IM, H0IM) == (15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0). Each bit is defined below. Value: 0 — Meaning: The corresponding hardware interrupt is disabled	RW	0 for all bits

Field Name	Bits	Description	Type	Reset
		1 — The corresponding hardware interrupt is enabled		
H31IM – H16IM	16 (31,16)	Hardware Interrupt X Mask bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows: (H31IM, H30IM, H29IM, H28IM, H27IM, H26IM, H25IM, H24IM, H23IM, H22IM, H21IM, H20IM, H19IM, H18IM, H17IM, H16IM) == (31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16). Each bit is defined below.	RW	0 for all bits

Value	Meaning
0	The corresponding hardware interrupt is disabled.
1	The corresponding hardware interrupt is enabled.

10.3.17. Interrupt Pending Register

Mnemonic Name: ir15 (INT_PEND)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 9, 0}

The Interrupt Pending Register is used to generate the software interrupt or record the status of the hardware interrupt. Software generates a software interrupt by writing to the SWI field of this register. The software interrupt bit is fed into the internal interrupt controller logic or exported through a pin to an external interrupt controller.

31 18	17	16	15 6	5 0
Reserved	NP_WPP	SWI	H15I – H6I (IVIC)	H5I – H0I (IVIC) or CIPL(5,0) (EVIC)

Field Name	Bits	Description	Type	Reset
H5I – H0I	6 (5,0)	Hardware Interrupt X pending bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows: (H5,H4,H3,H2,H1,H0) == (5,4,3,2,1,0). Each bit is defined below. Value: 0 – The corresponding hardware interrupt is not pending. Value: 1 – The corresponding hardware interrupt is pending.	RO	0
CIPL	6 (5,0)	The CIPL field is used in External Vector Interrupt Controller (EVIC) mode. It contains the Current Interrupt Priority Level requested by the external interrupt controller.	RO	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>H15I – H6I</td><td>10
(15,6)</td><td>Hardware Interrupt X pending bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows:
(H15,H14,H13,H12,H11,H10,H9,H8,H7,H6) == (15,14,13,12,11,10,9,8,7,6). Each bit is defined below.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The corresponding hardware interrupt is not pending.</td></tr><tr><td>1</td><td>The corresponding hardware interrupt is pending.</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>SWI</td><td>1
(16)</td><td>Software Interrupt. This bit is used to trigger a software interrupt (by setting it to “1”) and it also indicates whether a software interrupt is pending. It is used in both Internal and External Vector Interrupt Controller modes.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No software interrupt is pending.</td></tr><tr><td>1</td><td>A software interrupt is pending.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>NP_WPP</td><td>1
(17)</td><td>Next-Precise Watch-Point exception Pending. This bit indicates/remembers that there is still a next-precise watch-point exception pending for processing. It is cleared by either hardware selecting the watch-point exception as the current interruption handler or software writing a 0 to this bit.<table><tr><th>Value</th><th>Meaning</th></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		0: No next-precise watch-point exception is pending. 1: A next-precise watch-point exception is pending.		
Reserved	14 (31,18)	Reserved	RAZWI	0

10.3.18. Interrupt Pending Register 2

Mnemonic Name: ir27 (INT_PEND2)

IM Requirement: Internal vector interrupt controller optional (IVB.IVIC_VER >= 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 9, 1}

The Interrupt Pending Register 2 is used to record the status of the hardware interrupts in Internal Vector Interrupt Controller (IVIC) mode. Note that, for backward compatibility and ease of programming, its lower 16 bits (15, 0) are aliased and identical to the lower 16 bits of Interrupt Pending Register (ir15 or INT_PEND). Furthermore, these lower 16 bits can be cleared from this register interface (INT_PEND2) but not from the older interface (INT_PEND).

31 16	15 0
H31I – H16I	H15I – H0I

Field Name	Bits	Description	Type	Reset
H15I – H0I	16 (15,0)	Hardware Interrupt X pending bits. Note that, for backward compatibility and ease of programming, these 16 bits are aliased and identical to the lower 16 bits of Interrupt Pending Register (ir15 or INT_PEND). They are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows: (H15I, H14I, H13I, H12I, H11I, H10I, H9I, H8I, H7I, H6I, H5I, H4I, H3I, H2I, H1I, H0I) == (15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0). For level-triggered interrupts, software should not try to clear these pending bits since the pending status of level-triggered interrupts must be cleared from external sources (e.g., devices).	W1C	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>Otherwise, the pending bits will be triggered and set again if external interrupt request signals remain high.

Each bit is defined below.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The corresponding hardware interrupt is not pending.</td></tr><tr><td>1</td><td>The corresponding hardware interrupt is pending.</td></tr></table></td><td></td><td></td></tr>
<tr><td>H31I – H16I</td><td>16
(31,16)</td><td>Hardware Interrupt X pending bits. These bits are used in the Internal Vector Interrupt Controller mode. The hardware interrupt to bit assignment is as follows:
(H31I, H30I, H29I, H28I, H27I, H26I, H25I, H24I, H23I, H22I, H21I, H20I, H19I, H18I, H17I, H16I) ==
(31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16).

For level-triggered interrupts, software should not try to clear these pending bits since the pending status of level-triggered interrupts must be cleared from external sources (e.g., devices). Otherwise, the pending bits will be triggered and set again if external interrupt request signals remain high.

Each bit is defined below.
<table><tr><th>Value</th><th>Meaning</th></tr></table></td><td>W1C</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		0 — The corresponding hardware interrupt is not pending. 1 — The corresponding hardware interrupt is pending.		

10.3.19. Shadowed User Stack Pointer

Mnemonic Name: ir16 (SP_USR)

IM Requirement: Shadow register optional (MSC_CFG.SHADOW == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 10, 0}

This is simply an alias to user GPR R31 (stack pointer) and no extra storage is needed.

10.3.20. Shadowed Privileged Stack Pointer

Mnemonic Name: ir17 (SP_PRIV)

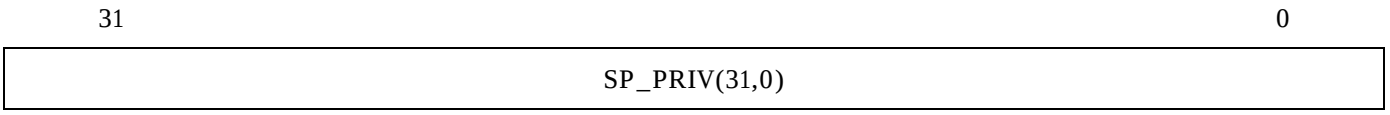
IM Requirement: Shadow register optional (MSC_CFG.SHADOW == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 10, 1}

This register is activated and used only when MISC_CTL.SP_SHADOW_EN = 1. Except for one special case described below, this register is automatically mapped to GPR “R31” (the stack pointer) when operating in superuser mode. The exact operating conditions for superuser mode are described in Section 1.1 Processor Operation Mode. However, there is one special case where it does not follow the rule mentioned previously. Namely, for host-mode debug exception (EDM_CTL.DEH_SEL==1 and PSW.DEX==1), if not overruled by the debugger (EDM_CTL.DEX_USE_PSW==1 and PSW.POM==0), the SP_PRIV register is mapped to GPR “R31” when PSW.POM==1 or the maximum stack level is reached. More details can be found in Section 5.5 Shadowed SP and SP_BOUND Register.

Its register format is as follows:



It is a RW type register with the reset value defined as 0.

10.3.21. Interrupt Priority Register

Mnemonic Name: ir18 (INT_PRI)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 11, 0}

The Interrupt Priority Register stores the priority level of the first 16 hardware interrupt sources. Each interrupt is assigned 2 bits to represent 4 possible priority levels ranging from 0 (highest) to 3 (lowest).

As asides, note that this programmable priority effect can also be achieved in previous V2 architecture with software by adjusting the Interruption Masking Register.

Its register format is as follows:

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
H7PRI	H6PRI	H5PRI	H4PRI	H3PRI	H2PRI	H1PRI	H0PRI

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
H15PRI	H14PRI	H13PRI	H12PRI	H11PRI	H10PRI	H9PRI	H8PRI

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>HnPRI</td><td>2
(2n+1,2n)</td><td>Hardware interrupt n priority level, where n is 0 to 15. This priority level can range from 0 (highest priority) to 3 (lowest priority).

This field exists only when the corresponding interrupt pin exists.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Highest priority level</td></tr>
<tr><td>1</td><td>Second highest priority level</td></tr>
<tr><td>2</td><td>Second lowest priority level</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		3 Lowest priority level		

10.3.22. Interrupt Priority Register 2

Mnemonic Name: ir28 (INT_PRI2)

IM Requirement: Internal vector interrupt controller optional (IVB.IVIC_VER >= 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 11, 1}

The Interrupt Priority Register 2 stores the priority level of hardware interrupt source number 16 to 31. Each interrupt is assigned 2 bits to represent 4 possible priority levels ranging from 0 (highest) to 3 (lowest).

Its register format is as follows:

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
H23PRI	H22PRI	H21PRI	H20PRI	H19PRI	H18PRI	H17PRI	H16PRI

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
H31PRI	H30PRI	H29PRI	H28PRI	H27PRI	H26PRI	H25PRI	H24PRI

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>HnPRI</td><td>2
(2*(n-16)+1,
2*(n-16))</td><td>Hardware interrupt n priority level, where n is 16 to 31. This priority level can range from 0 (highest priority) to 3 (lowest priority).

This field exists only when the corresponding interrupt pin exists.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Highest priority level</td></tr>
<tr><td>1</td><td>Second highest priority level</td></tr>
<tr><td>2</td><td>Second lowest priority level</td></tr>
<tr><td>3</td><td>Lowest priority level</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

10.3.23. Interrupt Control Register

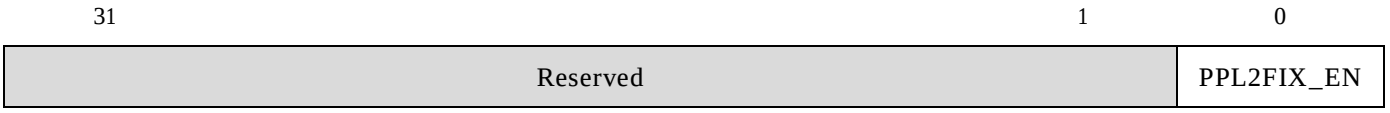
Mnemonic Name: ir19 (INT_CTRL)

IM Requirement: Required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 1, 2}

This register controls miscellaneous behavior for the interrupt controller. Its register format is as follows:



Field Name	Bits	Description	Type	Reset
PPL2FIX_EN	1 (0)	Programmable Priority Level (PPL) to Fixed Priority Level (FIX) Enable. This mode bit is used to make the interrupt controller backward compatible (as fixed priority level). When this mode bit is set, it changes all interrupts from programmable priority level to fixed priority level, and all interrupts become the same level. More specifically, all interrupts are treated as the same level, so they can preempt one another (when the Global Interrupt Enable is on). Please note that the PPL2FIX_EN feature applies to both IVIC and EVIC modes. This bit becomes Read-Only (RO) in CPUs configured with fixed priority level only. Value / Meaning	RW/ RO	IM

Field Name	Bits	Description	Type	Reset
		0 — Interrupts still keep Programmable Priority Level. 1 — Programmable Priority Level is changed into Fixed Priority Level for backward compatibility reason. The comparison of the interrupt current priority level (CPL) is disabled, so all interrupt priority level become the same but they can preempt one another.		

10.3.24. Interrupt Trigger Type Register

Mnemonic Name: ir29 (INT_TRIGGER)

IM Requirement: Internal vector interrupt controller optional (IVB.IVIC_VER >= 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {1, 9, 4}

The Interrupt Trigger Type Register is used to record the trigger type of hardware interrupts in Internal Vector Interrupt Controller (IVIC) mode. These bits indicate interrupts are level-triggered or edge-triggered.

31	0
H31TRIG – H0TRIG	

Field Name	Bits	Description	Type	Reset
H31TRIG – H0TRIG	32 (31,0)	Hardware interrupt X trigger type bits. These bits are used in the Internal Vector Interrupt Controller mode. They indicate each interrupt source belong to type level-triggered or edge-triggered. The hardware interrupt to bit assignment is as follows: (H31TRIG, H30TRIG, H29TRIG, H28TRIG, H27TRIG, H26TRIG, H25TRIG, H24TRIG, H23TRIG, H22TRIG, H21TRIG, H20TRIG, H19TRIG, H18TRIG, H17TRIG, H16TRIG, H15TRIG, H14TRIG, H13TRIG, H12TRIG, H11TRIG, H10TRIG, H9TRIG, H8TRIG, H7TRIG, H6TRIG, H5TRIG, H4TRIG, H3TRIG, H2TRIG, H1TRIG, H0TRIG) == (31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19,	RO	IM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0).
Each bit is defined below.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The corresponding hardware interrupt is level-triggered.</td></tr>
<tr><td>1</td><td>The corresponding hardware interrupt is edge-triggered.</td></tr>
</table></td><td></td><td></td></tr>
</table>

10.4. MMU System Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
mr0	MMU_CTL	2	0	0	229
mr1	L1_PPTB	2	1	0	235
mr2	TLB_VPN	2	2	0	236
mr3	TLB_DATA	2	3	0	238
mr4	TLB_MISC	2	4	0	244
mr5	VLPT_IDX	2	5	0	246
mr6	ILMB	2	6	0	248
mr7	DLMB	2	7	0	252
mr8	CACHE_CTL	2	8	0	256
mr9	HSMP_SADDR	2	9	0	259

10.4.1. MMU Control Register

Mnemonic Name: mr0 (MMU_CTL)

IM Requirement: required

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 0, 0}

For the Memory Protection Unit (MPU) configuration:

Please refer to section 3.4.1.

For the Memory Management Unit (MMU) configuration:

10	9	8 7	6 5	4 3	2 1	0
MPZIU	TBALCK	NTC3	NTC2	NTC1	NTC0	D

31 24	23	22 21	20	19	18 17	16 15	14 13	12 11
Reserved	UNA	ECCEN	Reserved	DREE	NTM3	NTM2	NTM1	NTM0

When MMU_CFG.MMPS is 0 or 1, the D, TBALCK, and MPZIU fields at MMU_CTL may either have no functionality or become “reserved” (it is implementation-dependent).

When MMU_CFG.NTME is 0, the NTM0-NTM3 fields should become “reserved” fields.

Field Name	Bits	Description	Type	Reset
Default minimum page size (D)	1 (0)	Indicates the default minimum page size: 0: 4KB 1: 8KB	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>NTC0</td><td>2
(2,1)</td><td>Indicates Non-translated Cacheability memory attribute for partition 0.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Non-cacheable/Non-coalesable</td></tr><tr><td>1</td><td>Non-cacheable/Coalesable</td></tr><tr><td>2</td><td>Cacheable/Write-Back</td></tr><tr><td>3</td><td>Cacheable/Write-Through</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>NTC1</td><td>2
(4,3)</td><td>Indicates Non-translated Cacheability memory attribute for partition 1. See NTC0 description for its value definition.</td><td>RW</td><td>0</td></tr>
<tr><td>NTC2</td><td>2
(6,5)</td><td>Indicates Non-translated Cacheability memory attribute for partition 2. See NTC0 description for its value definition.</td><td>RW</td><td>0</td></tr>
<tr><td>NTC3</td><td>2
(8,7)</td><td>Indicates Non-translated Cacheability memory attribute for partition 3. See NTC0 description for its value definition.</td><td>RW</td><td>0</td></tr>
<tr><td>TBALCK</td><td>1
(9)</td><td>Controls TLB all-lock resolution scheme:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Generating “Machine Error” exception.</td></tr><tr><td>1</td><td>Hardware random/LRU replacement.</td></tr></table>Operation exception behavior affected by this bit is listed as follows:<table><tr><td>HPTWK TLB fill operation</td></tr><tr><td>TLB Random Write</td></tr><tr><td>TLB Random Write and Lock</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MPZIU</td><td>1
(10)</td><td>Multiple Page Size In Use bit. This bit is for OS to indicate to hardware that it is currently using multiple page sizes at the same time. If this bit is not set properly while OS tries to use multiple page sizes, the TLB search for the PTE of large page size may always miss. If this bit is not set while the hardware page table walker is instructed by a PTE to insert the PTE as a large page size, then a MPZIU Control exception will be generated and the PTE will not be inserted into TLB.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No multiple page size in use. Hardware TLB search for the PTE of large page size may always fail.</td></tr>
<tr><td>1</td><td>Multiple page size in use. Hardware TLB will perform TLB search for the PTE of large page size. But this may take more cycles.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>NTM0
(MMU_CFG.NTME == 1)</td><td>2
(12,11)</td><td>Indicates Non-translated VA to PA[31:30] mapping for partition 0.</td><td>RW</td><td>0</td></tr>
<tr><td>NTM1
(MMU_CFG.NTME == 1)</td><td>2
(14,13)</td><td>Indicates Non-translated VA to PA[31:30] mapping for partition 1.</td><td>RW</td><td>1</td></tr>
<tr><td>NTM2
(MMU_CFG.NTME == 1)</td><td>2
(16,15)</td><td>Indicates Non-translated VA to PA[31:30] mapping for partition 2.</td><td>RW</td><td>2</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>NTM3
(MMU_CFG.NTME
== 1)</td><td>2
(18,17)</td><td>Indicates Non-translated VA to PA[31:30] mapping for partition 3.</td><td>RW</td><td>3</td></tr>
<tr><td>DREE</td><td>1
(19)</td><td>Device register endian control enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Device register endian control function is disabled. A data memory access uses PSW.BE as the endian type.</td></tr><tr><td>1</td><td>Device register endian control function is enabled. A data memory access with the following attributes, (C==0 or NTC==0), uses PSW.DRBE as the endian type while a data memory access with the other attributes uses PSW.BE as the endian type.</td></tr></table></td><td>RW</td><td>IM</td></tr>
<tr><td>Reserved</td><td>1
(20)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>ECCEN</td><td>2
(22,21)</td><td>TLB parity/ECC enable control.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC.</td></tr><tr><td>1</td><td>Reserved.</td></tr><tr><td>2</td><td>Generate exceptions only on uncorrectable parity/ECC errors.</td></tr><tr><td>3</td><td>Generate exceptions on any type</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		of parity/ECC errors.		
UNA	1 (23)	This field controls whether load/store world/half-word instructions can access unaligned memory locations without generating the Data Alignment Check exceptions. Value: 0 — Unaligned access generates the Data Alignment Check exception. Value: 1 — Unaligned access is allowed.	RW	0
Reserved	8 (31,24)	Reserved	RAZWI	0

NTC0-NTC3 is the Non-translated Cacheability attributes used when the VA to PA translation process is turned off. The value definition of the NTC field is listed in Table 2 in section 2.2.2.1. When only VA[31] is used to access the NTC fields, only NTC0/NTC1 are used. When VA[31:30] is used to access the NTC fields, all NTC fields will be used. The NTC field assignment for each case is listed in Table 29 and Table 30. For reduced 24-bit address space configuration (MSC_CFG.ADR24 is 1), VA[23] or VA[23:22] are used.

NTM0-NTM3 is the Non-translated VA[31:30] to PA[31:30] Mapping fields when MMU_CFG.NTME is equal to 1. It is most often used to improve memory access performance on a small system without MMU or MPU by mapping different NTC attributes (e.g., cacheable/WB, cacheable/WT) to the same PA partition. Thus, the memory access type can be controlled using VA[31:30]. When only VA[31] is used to access the NTM fields, only NTM0/NTM1 are used. When VA[31:30] is used to access the NTM fields, all NTM fields will be used. The NTM field assignment for each case is listed in Table 29 and Table 30. For reduced 24-bit address space configuration (MSC_CFG.ADR24 is 1), VA[23] or VA[23:22] are used.

Table 29. NTC field assignment for indexing using VA[31]

VA[31]	NTC field	NTM field
1'b0	NTC0	NTM0
1'b1	NTC1	NTM1

Table 30. NTC field assignment for indexing using VA[31:30]

VA[31:30]	NTC field	NTM field
2'b00	NTC0	NTM0
2'b01	NTC1	NTM1
2'b10	NTC2	NTM2
2'b11	NTC3	NTM3

10.4.2. L1 Physical Page Table Base Register

Mnemonic Name: mr1 (L1_PPTB)

IM Requirement: MMU hardware page table walker optional

((MMU_CFG.MMPS == 2) & (MMU_CFG.HPTWK == 1))

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 1, 0}

31 12	11 1	0
L1_PPT_BASE	Reserved	NV

This is a pointer to the current level 1 physical page table. The value of the pointer should be in Physical address space. This pointer is used by the HPTWK to construct the load address to locate the level 2 physical page table base.

Field Name	Bits	Description	Type	Reset
NV	1 (0)	Used to enable/disable Hardware Page Table Walker (HPTWK). Value: 0 — HPTWK is enabled. Value: 1 — HPTWK is disabled.	RW	1
Reserved	11 (11,1)	Reserved	RAZWI	0
L1 PPT Base Address (L1_PPT_BASE)	20 (31,12)	Indicates the first-level physical page table base address. It is 4KB-aligned.	RW	0

10.4.3. TLB Access VPN Register

Mnemonic Name: mr2 (TLB_VPN)

IM Requirement: MPU or MMU optional (MMU_CFG.MMPS > 0)

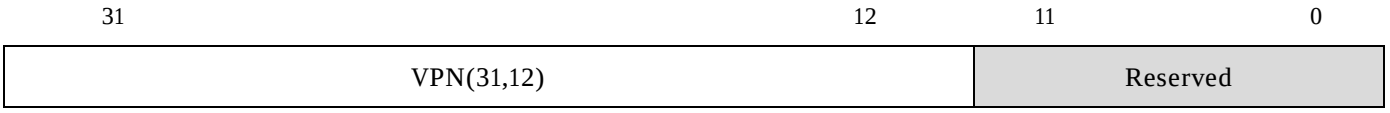
Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 2, 0}

For the Memory Protection Unit (MPU) configuration:

Please refer to section 3.4.2.

For the Memory Management Unit (MMU) configuration:



For TLB writes (target or random), this is the virtual page number (VPN) of the page translation entry to be installed.

For TLB target reads, this register holds the VPN of the read result.

When the following TLB exceptions happen, this register will be updated to the exception VPN.

TLB fill
PTE not present
Page modified
TLB permission

So, for example, after a TLB fill exception, this register will contain the correct VPN content for the TLB fill operation needed to be performed in the exception handler.

No shadow stack register is specified for the TLB_VPN.VPN field since we assume that nested exceptions specified above cannot happen.

Field Name	Bits	Description	Type	Reset
Reserved	12 (11,0)	Reserved	RAZWI	0
Virtual Page Number (VPN)	20 (31,12)	Virtual Page Number. It is updated by hardware on the following TLB exceptions: TLB Fill, PTE not present, Page Modified, and TLB permission, when interruption stack level transition 0 → 1 or 1 → 2 happens. For all minimum page configurations, it is updated with VA(31,12).	RW	DC

10.4.4. TLB Access Data Register

Mnemonic Name: mr3 (TLB_DATA)

IM Requirement: MPU or MMU optional (MMU_CFG.MMPS > 0)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 3, 0}

For the Memory Protection Unit (MPU) configuration:

Please refer to section 3.4.3.

For the Memory Management Unit (MMU) configuration:

31 12	11	10 8	7	6	5	4	3 1	0
PPN	Reserved	C	G	A	X	D	M	V

For TLB target write, this is the page translation entry (PTE) to be written into TLB. For TLB reads, this is the page translation entry read out. The page translation entry translates virtual addresses into physical addresses.

Note that for TLB random write, the PTE data is coming from a general register instead of this register.

Field Name	Bits	Description	Type	Reset
V	1 (0)	Valid bit. Setting this bit indicates that this Page Table Entry is valid and present.	RW	DC
M (For MMU version 1)	3 (3,1)	This field indicates the page read/write access privilege. If a load instruction accesses a page which has no read privilege, a Read Protection Violation exception is generated. If a store instruction accesses a page which has no write privilege, a Write Protection Violation exception is	RW	DC

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>generated.
<table>
<tr><th>Value</th><th>User mode</th><th>Superuser mode</th></tr>
<tr><td>0</td><td>-</td><td>-</td></tr>
<tr><td>1</td><td>Read only</td><td>Read only</td></tr>
<tr><td>2</td><td>Read only</td><td>Read/Write</td></tr>
<tr><td>3</td><td>Read/write</td><td>Read/write</td></tr>
<tr><td>4</td><td>-</td><td>-</td></tr>
<tr><td>5</td><td>No access</td><td>Read only</td></tr>
<tr><td>6</td><td>-</td><td>-</td></tr>
<tr><td>7</td><td>No access</td><td>Read/Write</td></tr>
</table>
Note that this Access mode only governs the access permission of using load/store instructions to access the page. It does not govern the access permission of an instruction fetching and execution.
Using of Reserved values (0, 4, 6) will generate Reserved PTE Attribute exception (Data).</td><td></td><td></td></tr>
<tr><td>M
(For MMU version 2)</td><td>3
(3,1)</td><td>Read/Write Access mode + user mode fetching restriction on superuser mode page.
<table>
<tr><th>M[3:1]</th><th>User mode</th><th>Superuser mode</th></tr>
<tr><td>3’b000
X==0</td><td>-</td><td>-</td></tr>
<tr><td>3’b000
X==1</td><td>No Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b001</td><td>Read only</td><td>Read only</td></tr>
<tr><td>3’b010</td><td>Read only</td><td>Read/Write</td></tr>
<tr><td>3’b011</td><td>Read/Write</td><td>Read/Write</td></tr>
<tr><td>3’b100</td><td>-</td><td>-</td></tr>
</table></td><td>RW</td><td>DC</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		3'b101 — No Read/Write/Fetch access — Read only 3'b110 — - — - 3'b111 — No Read/Write/Fetch access — Read/Write Note that this Access mode governs the access permission of using load/store instructions to access the page. And it governs the user mode instruction fetching/execution permission when M[3:1] == 3'b101 or M[3:1] == 3'b111. Using of Reserved values 0 (when X is 0), 4, and 6 will generate Reserved PTE Attribute exception for load/store instructions and instruction fetching.		
D	1 (4)	This is the Dirty bit. Value 0 — If this bit is not set, a store to this page generates a Page Modified Exception. Value 1 — No Page Modified Exception will be generated.	RW	DC
X (For MMU version 1)	1 (5)	Indicates if this page is executable or not. Value 0 — This page is not executable. An instruction fetch to this page generates a Non-Executable	RW	DC

Field Name	Bits	Description	Type	Reset
		<table><tr><td></td><td>Page exception.</td></tr><tr><td>1</td><td>This page is executable.</td></tr></table>		
X (For MMU version 2)	1 (5)	Executable bit. ■ In superuser mode, a non-executable page exception will be generated when fetching a page without this bit set. ■ In user mode and when M[3:1] != 3'b101 and M[3:1] != 3'b111, a non-executable page exception will be generated when fetching a page without this bit set. ■ In user mode and when M[3:1] == 3'b101 or M[3:1] == 3'b111, a non-executable page exception will always be generated. The X bit is don't care in this case.	RW	DC
A	1 (6)	This bit controls if an Access Bit exception will be generated or not. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No Access Bit exception will be generated.</td></tr><tr><td>1</td><td>Any access to this page generates Access Bit exception.</td></tr></table>	RW	DC
G	1 (7)	This bit indicates if this page is shared across contexts. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>This page is not shared with other context.</td></tr><tr><td>1</td><td>This page is shared across contexts.</td></tr></table>	RW	DC

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>C</td><td>3
(10,8)</td><td>Indicates Cacheability attributes.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>device space</td></tr><tr><td>1</td><td>device space, write bufferable / coalescable</td></tr><tr><td>2</td><td>non-cacheable memory</td></tr><tr><td>3</td><td>Reserved</td></tr><tr><td>4</td><td>cacheable, write-back, write-allocate memory (shared)</td></tr><tr><td>5</td><td>cacheable, write-through, no-write-allocate memory (shared)</td></tr><tr><td>6</td><td>cacheable, non-shared, write-back, write allocate memory</td></tr><tr><td>7</td><td>cacheable, non-shared, write-through, no-write-allocate memory</td></tr></table>Using of the reserved value (3) will generate Reserved PTE Attribute exception (Instruction/Data).
Note: For dual/multi cores sharing a second-level cache, an implementation may choose to cache “shared” data only in L2 while caching “non-shared” data in both L1/L2. Please see implementation documents for details.</td><td>RW</td><td>DC</td></tr>
<tr><td>Reserved</td><td>1
(11)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
PPN	20 (31,12)	This is the Physical Page Number of a page.	RW	DC

10.4.5. TLB Access Misc. Register

Mnemonic Name: mr4 (TLB_MISC)

IM Requirement: MMU optional (MMU_CFG.MMPS == 2)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 4, 0}

31 13	12 4	3 0
Reserved	CID	ACC_PSZ

For TLB reads, this register holds the context ID and page size field of the read result.

For TLB writes (target or random), this is the context ID and page size of the page translation entry to be installed.

The CONTEXT_ID field of this register is used as the current context ID to tag virtual addresses to avoid the TLB flushing penalty between context switches. The operating system is responsible for assigning a new context ID for each context switch. Since changing this field affects how PTE is searched in the TLB, care must be taken when changing this field. The code that changes this field must be either executed with TLB translation off or contained in a global page PTE where changing CID will not cause any TLB translation exceptions.

The ACC_PSZ field of this register is the page size of the page translation entry installed in the software-visible part of the TLB. Note that not all page sizes are implemented in an implementation for the value range between 0 and 9. When this field is written with an unimplemented page size in the value range between 0 and 9, a “Machine Error” exception will be generated.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Access page size
(ACC_PSZ)</td><td>4
(3,0)</td><td>Indicates the page size of a PTE entry. Not all page sizes are implemented in an implementation. The MMU_CFG configuration register lists the page sizes supported in an implementation. Writing an unimplemented page size into this field will generate a “Machine Error” exception.
<table>
<tr><th>Value</th><th>Page Size</th></tr>
<tr><td>0</td><td>4KB</td></tr>
<tr><td>1</td><td>8KB</td></tr>
<tr><td>2</td><td>16KB</td></tr>
<tr><td>3</td><td>64KB</td></tr>
<tr><td>4</td><td>256KB</td></tr>
<tr><td>5</td><td>1MB</td></tr>
<tr><td>6</td><td>4MB</td></tr>
<tr><td>7</td><td>16MB</td></tr>
<tr><td>8</td><td>64MB</td></tr>
<tr><td>9</td><td>256MB</td></tr>
<tr><td>10-15</td><td>Reserved</td></tr>
</table></td><td>RW</td><td>DC</td></tr>
<tr><td>CID</td><td>9
(12,4)</td><td>Current Context ID. It will be installed into TLB along with a PTE entry.</td><td>RW</td><td>DC</td></tr>
<tr><td>Reserved</td><td>19
(31,13)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

10.4.6. Virtual Linear Page Table Index Register

Mnemonic Name: mr5 (VLPT_IDX)

IM Requirement: MMU optional (MMU_CFG.MMPS == 2) and Implementation-dependent

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 5, 0}

31 22	21 2	1 0
VLPTB	EVPN	0

Optionally, to speed up software page table walk operations in the TLB fill exception handler, an operating system can provide a Virtual Linear Page Table which maps the physical page tables into a linear table in a process's virtual address space. If this VLPT mapping exists in the TLB structure, then a two-level physical page table walk using two load instructions will be simplified to a one-level virtual memory load access in the TLB fill exception handler. This is beneficial if there are spare TLB entries which can be used to map VLPT pages. And a 4KB VLPT page can map 4MB of user address space. The VLPT is aligned on a 4MB boundary in the virtual address space. If the VLPT page mapping does not exist in the TLB structure, a nested TLB miss (called double TLB miss exception) will be raised.

The "EVPN" field of this register will be updated to the exception VPN when the following TLB-related exceptions occur.

TLB fill
PTE not present
Page modified

So when a TLB fill exception happens, this register will already contain the correct VLPT index where the PTE translation is located for the exception VA in the exception handler.

No shadow stack register is specified for the VLPT_IDX.EVPN field since we assume that nested exceptions specified above cannot happen.

Field Name	Bits	Description	Type	Reset
Zero	2 (1,0)	Always 0.	RO	0
EVPN	20 (21,2)	Exception Virtual Page Number. It is updated by hardware on the following TLB exceptions: TLB Fill, PTE not present, and Page Modified, when interruption stack level transition 0 → 1 or 1 → 2 happens. For a 4KB minimum page configuration, it is updated with VA(31,12). For a 8KB minimum page configuration, it is updated with 0.VA(31,13).	RO	0
VLPTB	10 (31,22)	The base virtual address of a process's Virtual Linear Page Table. It is 4MB-aligned.	RW	0

10.4.7. ILM (Instruction Local Memory) Base Register

Mnemonic Name: mr6 (ILMB)

IM Requirement: ILM optional (ICM_CFG.ILMB != 0)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 6, 0}

For ICM_CFG.BSAV == 0:

Obsolete and no longer used.

For ICM_CFG.BSAV == 1:

31 — 10	9	8 — 7	6 — 5	4 — 1	0
Base Physical Address (IBPA)	Reserved	ECCEN	Reserved	ILMSZ	IEN

Field Name	Bits	Description	Type	Reset
Enable (IEN)	1 (0)	Instruction Local Memory enable bit. This controls AndesCore pipeline access only, including instruction fetch, and instruction execution. It does not affect DMA engine operations. In some implementations, this bit may become fixed to “0” for error prevention if the size of external local memory is set to zero or illegal values. Its reset value is 0 unless the optional feature of booting from ILM is turned on. If this feature is on (by asserting pin “ilm_boot”), the CPU automatically sets the reset value to 1 and boots from ILM directly.	RW/ RO	IM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>ILM is disabled.</td></tr><tr><td>1</td><td>ILM is enabled.</td></tr></table></td><td></td><td></td></tr>
<tr><td>ILM_Size (ILMSZ)</td><td>4
(4,1)</td><td>Indicates the size of ILM. The size has to be power of 2.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>4KB</td></tr><tr><td>1</td><td>8KB</td></tr><tr><td>2</td><td>16KB</td></tr><tr><td>3</td><td>32KB</td></tr><tr><td>4</td><td>64KB</td></tr><tr><td>5</td><td>128KB</td></tr><tr><td>6</td><td>256KB</td></tr><tr><td>7</td><td>512KB</td></tr><tr><td>8</td><td>1024KB</td></tr><tr><td>9</td><td>1KB</td></tr><tr><td>10</td><td>2KB</td></tr><tr><td>11</td><td>2048KB</td></tr><tr><td>12</td><td>4096KB</td></tr><tr><td>13</td><td>8192KB</td></tr><tr><td>14</td><td>16384KB</td></tr><tr><td>15</td><td>0 Byte
(used for unconnected external ILM)</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Reserved</td><td>2
(6,5)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
ECCEN	2 (8,7)	Parity/ECC enable control. 0: Disable parity/ECC. 1: Reserved. 2: Generate exceptions only on uncorrectable parity/ECC errors. 3: Generate exceptions on any type of parity/ECC errors.	RW	0
Reserved	1 (9)	Reserved	RAZWI	0
ILM Base Physical Address (IBPA)	22 (31,10)	The base physical address of ILM. It has to be aligned to multiple of ILM size. For example, to set up an instruction local memory of size 4KB starting at address 12KB (0x3000), we simply program ILMB.IBPA to 0xC (take the upper 22 bits of 0x3000). If the base address programmed is not a multiple of the local memory size (an illegal address), the hardware will automatically mask out the illegal least-significant bits and only use the valid portion to access local memory to prevent errors. Nevertheless, the value of this register will be processed differently depending on whether internal or external local memory is used. For internal local memory, this register will get a new address which has the illegal least-significant bits masked out by hardware. In contrast, for external	RW	IM

Field Name	Bits	Description	Type	Reset
		local memory, this register preserves the original illegal base address programmed without masking. Its reset value is “don’t care” unless the optional feature of booting from ILM is turned on. If this feature is on (by asserting pin “ilm_boot”), the CPU automatically sets the reset value of IBPA to the default address of IVB (i.e., IBPA[31:16] = IVB.IVBASE[31:16]). This allows the CPU to boot from ILM directly.		

10.4.8. DLM (Data Local Memory) Base Register

Mnemonic Name: mr7 (DLMB)

IM Requirement: DLM optional (DCM_CFG.DLMB != 0)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 7, 0}

For DCM_CFG.BSAV == 0:

Obsolete and no longer used.

For DCM_CFG.BSAV == 1:

31 — 10	9	8	7	6	5	4 — 1	0
Base Physical Address (DBPA)	Reserved	ECCEN		DBB	DBM	DLMSZ	DEN

Field Name	Bits	Description	Type	Reset
Enable (DEN)	1 (0)	Data Local Memory enable bit. This controls AndesCore pipeline access only. It does not affect DMA engine operations. In some implementations, this bit may become fixed to "0" for error prevention if the size of external local memory is set to zero or illegal values. Value: 0 — DLM is disabled.; 1 — DLM is enabled.	RW/ RO	0
DLM_Size (DLMSZ)	4 (4,1)	Indicates the size of DLM. The size has to be power of 2. Value: 0 — 4KB	RO	IM

Field Name	Bits	Description	Type	Reset
		1 — 8KB 2 — 16KB 3 — 32KB 4 — 64KB 5 — 128KB 6 — 256KB 7 — 512KB 8 — 1024KB 9 — 1KB 10 — 2KB 11 — 2048KB 12 — 4096KB 13 — 8192KB 14 — 16384KB 15 — 0 Byte (used for unconnected external DLM)		
DBM	1 (5)	Enable Double-Buffer Mode for data local memory. When in Double-Buffer mode, the data local memory is divided into two address overlapping regions where one can only be accessed by the processor and the other can only be accessed by the DMA engine. The next field, DBB, determines which half of the local memory can be accessed by the processor. 0 — Double-buffer mode is disabled. 1 — Double-buffer mode is enabled.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>Toggling/Changing this bit will cause UNPREDICTABLE change in the data local memory content. Software should re-initialize the data local memory content after changing this bit.

In the Unified Local Memory configuration, this bit is hardwired to 0 since Double-Buffer Mode is not supported.</td><td></td><td></td></tr>
<tr><td>DBB</td><td>1
(6)</td><td>Double-buffer bank which can be accessed by the processor. This field is used only when the previous field, DBM, is set to 1.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Double-buffer bank 0 is accessed by the processor.</td></tr><tr><td>1</td><td>Double-buffer bank 1 is accessed by the processor.</td></tr></table>
In the Unified Local Memory configuration, this bit is hardwired to 0 since Double-Buffer Mode is not supported.</td><td>RW</td><td>0</td></tr>
<tr><td>ECCEN</td><td>2
(8,7)</td><td>Parity/ECC enable control.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC.</td></tr><tr><td>1</td><td>Reserved.</td></tr><tr><td>2</td><td>Generate exceptions only on uncorrectable parity/ECC errors.</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
Reserved	1 (9)	Reserved	RAZWI	0
DLM Base Physical Address (DBPA)	22 (31,10)	The base physical address of DLM. It has to be aligned to multiple of DLM size. For example, to set up a data local memory of size 4KB starting at address 12KB (0x3000), we simply program DLMB.DBPA to 0xC (take the upper 22 bits of 0x3000). If the base address programmed is not a multiple of the local memory size (an illegal address), the hardware will automatically mask out the illegal least-significant bits and only use the valid portion to access local memory to prevent errors. Nevertheless, the address in this register will be processed differently depending on whether internal or external local memory is used. For internal local memory, this register will get a new address which has the illegal least-significant bits masked out by hardware. In contrast, for external local memory, this register preserves the original illegal base address programmed without masking.	RW	DC

10.4.9. Cache Control Register

Mnemonic Name: mr8 (CACHE_CTL)

IM Requirement: Cache optional ((ICM_CFG.ISZ != 0) | (DCM_CFG.DSZ != 0))

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 8, 0}

31 11	10	9 8	7 6	5	4	3	2	1	0
Reserved	FWTM (opt)	DC_ECCEN	IC_ECCEN	DCPMW	DCCWF	DCALCK	ICALCK	DC_EN	IC_EN

Field Name	Bits	Description	Type	Reset
Icache Enable (IC_EN)	1 (0)	Controls if the first-level instruction cache is enabled or not. Value: 0 — The instruction cache is disabled. Value: 1 — The instruction cache is enabled.	RW	0
Dcache Enable (DC_EN)	1 (1)	Controls if the first-level data cache is enabled or not. Value: 0 — The data cache is disabled. Value: 1 — The data cache is enabled.	RW	0
ICALCK	1 (2)	Controls I-cache all-lock resolution scheme. Value: 0 — Generates a “Machine Error” exception. Value: 1 — Changes a cacheable request to	RW	0

Field Name	Bits	Description	Type	Reset
		<table><tr><td></td><td>the all-locked cache set into an un-cacheable request.</td></tr></table>		
DCALCK	1 (3)	Controls D-cache all-lock resolution scheme. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Generates a “Machine Error” exception.</td></tr><tr><td>1</td><td>Changes a cacheable request to the all-locked cache set into an un-cacheable request.</td></tr></table>	RW	0
DCCWF	1 (4)	Controls if D-cache Critical Word Forwarding functionality is enabled or not. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>D-cache Critical Word Forwarding is disabled.</td></tr><tr><td>1</td><td>D-cache Critical Word Forwarding is enabled.</td></tr></table>	RW	1
DCPMW	1 (5)	Controls if D-cache concurrent (parallel) miss and write-back processing is enabled or not. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>D-cache concurrent miss and write-back processing is disabled.</td></tr><tr><td>1</td><td>D-cache concurrent miss and write-back processing is enabled.</td></tr></table>	RW	1

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>IC_ECCEN</td><td>2
(7,6)</td><td>Parity/ECC enable control for the instruction cache.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC.</td></tr><tr><td>1</td><td>Reserved.</td></tr><tr><td>2</td><td>Generate exceptions only on uncorrectable parity/ECC errors.</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>DC_ECCEN</td><td>2
(9,8)</td><td>Parity/ECC enable control for the data cache.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC.</td></tr><tr><td>1</td><td>Reserved.</td></tr><tr><td>2</td><td>Generate exceptions only on uncorrectable parity/ECC errors.</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>FWTM
(optional)</td><td>1
(10)</td><td>Force to the write-through mode. The existence of this bit should be checked by writing then reading it back.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Enable</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>26
(31,6)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

10.4.10. High Speed Memory Port Starting Address

Mnemonic Name: mr9 (HSMP_SADDR)

IM Requirement: High Speed Memory Port optional (MSC_CFG.HSMP == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 9, 0}

31 20	19 13	12 1	0
SADDR	Reserved	RANGE	EN

This register defines the starting base physical address of the High Speed Memory Port region and the power-of-2 range of the region. The region has to be power-of-2 aligned. Any value of SADDR(31,20) outside of the power of 2 range defined in RANGE(12,1) will be ignored (i.e., treated as zero). This register is removed if the MSC_CFG.HSMP is 0.

Field Name	Bits	Description	Type	Reset
EN	1 (0)	Enable control bit for the High Speed Memory port. Value: 0 — The High Speed Memory Port is disabled. Value: 1 — The High Speed Memory Port is enabled.	RW	0
RANGE	12 (12,1)	The address range is denoted using the following pattern from 1MB to 4GB. If these patterns are not followed, UNPREDICTABLE memory access behavior will be generated. Range: 1MB — Pattern: 12'b111111111111	RW	DC

Field Name	Bits	Description	Type	Reset
		2MB: 12'b111111111110 4MB: 12'b111111111100 8MB: 12'b111111111000 16MB: 12'b111111110000 32MB: 12'b111111100000 64MB: 12'b111111000000 128MB: 12'b111110000000 256MB: 12'b111100000000 512MB: 12'b111000000000 1GB: 12'b110000000000 2GB: 12'b100000000000 4GB: 12'b000000000000		
Reserved	7 (19,13)	Reserved	RAZWI	0
SADDR	12 (31,20)	Starting base physical address of the High Speed Memory Port region. It is a power-of-2 range of memory blocks. The value outside of the power-of-2 range defined in RANGE(12,1) will be ignored.	RW	DC

10.5. EDM System Registers

Note that EDM system registers will have unpredictable values if the CPU and ICE both write at the same time.

Please refer to specification on "*Andes Embedded Debug Module*" for detailed description.

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension
dr0+(n*5)	BPCn (n=0..7)	3	0	n
dr1+(n*5)	BPAn (n=0..7)	3	1	n
dr2+(n*5)	BPAMn (n=0..7)	3	2	n
dr3+(n*5)	BPVn (n=0..7)	3	3	n
dr4+(n*5)	BPCIDn (n=0..7)	3	4	n
dr40	EDM_CFG	3	5	0
dr41	EDMSW	3	6	0
dr42	EDM_CTL	3	7	0
dr43	EDM_DTR	3	8	0
dr44	BPMTC	3	9	0
dr45	DIMBR	3	10	0
dr46	TECR0	3	14	0
dr47	TECR1	3	14	1

10.6. Hardware Stack Protection Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
hspr0	HSP_CTL	4	6	0	263
hspr1	SP_BOUND	4	6	1	266
hspr2	SP_BOUND_PRIV	4	6	2	267
hspr3	SP_BASE	4	6	3	268
hspr4	SP_BASE_PRIV	4	6	4	269

10.6.1. HW Stack Protection Control Register

Mnemonic Name: hspr0 (HSP_CTL)

IM Requirement: HW stack protection optional (MSC_CFG.HSP == 1)

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 6, 0}

This register enables or disables the stack range protection and the top of stack recording mechanism and selects the operating scheme between the stack range protection and the top of stack recording. This register also specifies the "usage mode" of the mechanism. The enabled mechanism can be used in user mode only, superuser mode only, or both.

31 9	8	7	6 4	3	2	1	0
Reserved	UDF_EN	UDF	SPL	U	SU	SCHM	HSP_EN

Field Name	Bits	Description	Type	Reset
HSP_EN	1 (0)	Enable bit for the stack overflow protection and recording mechanism. This bit will be set to 0 automatically by hardware when a next-precise stack overflow/underflow exception is taken. Value: 0 — Meaning: The stack overflow protection and recording mechanism is disabled. Value: 1 — Meaning: The stack overflow protection and recording mechanism is enabled.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SCHM</td><td>1
(1)</td><td>Indicates the operating scheme of the stack protection and recording mechanism.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The stack overflow/underflow detection is selected as the operating scheme.</td></tr><tr><td>1</td><td>The top of stack recording is selected as the operating scheme.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SU</td><td>1
(2)</td><td>Enables the SP protection and recording mechanism in the superuser mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The mechanism is disabled in the superuser mode.</td></tr><tr><td>1</td><td>The mechanism is enabled in the superuser mode.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>U</td><td>1
(3)</td><td>Enables the SP protection and recording mechanism in the user mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The mechanism is disabled in the user mode.</td></tr><tr><td>1</td><td>The mechanism is enabled in the user mode.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SPL</td><td>3
(6,4)</td><td>(Secure Core Only) the enabled Security Privilege Level for the SP protection and recording mechanism.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Enabled for SPL 0.</td></tr><tr><td>1</td><td>Enabled for SPL 1.</td></tr><tr><td>2</td><td>Enabled for SPL 2.</td></tr><tr><td>3</td><td>Enabled for SPL 3.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UDF</td><td>1
(7)</td><td>Indicates if the stack underflow protection mechanism is supported or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The stack underflow protection mechanism is not supported.</td></tr><tr><td>1</td><td>The stack underflow protection mechanism is supported.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>UDF_EN</td><td>1
(8)</td><td>Enable bit for the stack underflow protection mechanism. This bit will be set to 0 automatically by hardware when a next precise stack overflow/underflow exception is taken.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The stack underflow protection mechanism is disabled.</td></tr><tr><td>1</td><td>The stack underflow protection mechanism is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>23
(31,9)</td><td>RAZWI</td><td>RAZWI</td><td>0</td></tr>
</table>

10.6.2. SP Bound Register

Mnemonic Name: hspr1 (SP_BOUND)

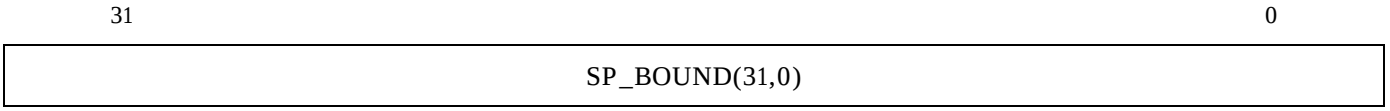
IM Requirement: HW stack protection optional (MSC_CFG.HSP == 1)

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 6, 1}

When the SP overflow detection mechanism is properly selected and enabled, any updated value to the SP register (via any instruction) is compared with the SP_BOUND register. If the updated value to the SP register is smaller than the SP_BOUND register, a next-precise stack overflow exception is generated. The next-precise stack overflow exception belongs to the “General Exception” entry point with ETYPE equal to 11 and INST equals to 0.

When the top of stack recording mechanism is properly selected and enabled, any updated value to the SP register on any instruction is compared with the SP_BOUND register. If the updated value to the SP register is smaller than the SP_BOUND register, the SP_BOUND register is updated with this updated value. This register is 32-bit (or 24-bit in Reduced Address Space configuration.)



It is a RW type register with the reset value defined as 0xFFFFFFFF.

10.6.3. Shadowed Privileged SP Bound Register

Mnemonic Name: hspr2 (SP_BOUND_PRIV)

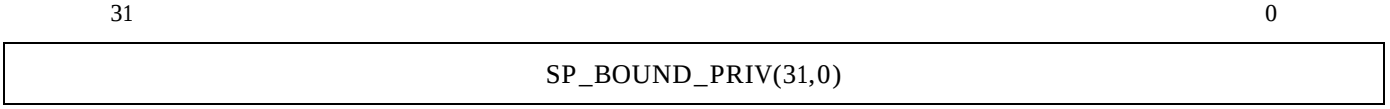
IM Requirement: HW stack protection & shadow register optional

((MSC_CFG.HSP == 1) & (MSC_CFG.SHADOW != 0))

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 6, 2}

This register will be activated and used in superuser mode when MISC_CTL.SP_SHADOW_EN is set to 1. When a CPU is in superuser mode (please see section 5.5), the enabled stack pointer overflow detection and recording mechanism will use this register, instead of the SP_BOUND register, to compare with any updated value to the SP register. This register is 32-bit (or 24-bit in Reduced Address Space configuration.)



It is a RW type register with the reset value defined as 0xFFFFFFFF.

10.6.4. SP Base Register

Mnemonic Name: hspr3 (SP_BASE)

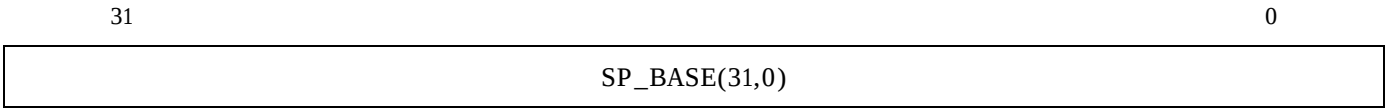
IM Requirement: HW stack protection optional

(MSC_CFG.HSP == 1) & (HSP_CTL.UDF == 1)

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 6, 3}

When the SP protection mechanism is properly selected and enabled, any updated value to the SP register (via any instruction) is compared with the SP_BASE register. If the updated value to the SP register is greater than the SP_BASE register, a next-precise stack underflow exception is generated. The next precise stack underflow exception belongs to the “General Exception” entry point with ETYPE equal to 11 and INST equals to 1. This register is 32-bit (or 24-bit in Reduced Address Space configuration.)



It is a RW type register with the reset value defined as 0xFFFFFFFF.

10.6.5. Shadowed Privileged SP Base Register

Mnemonic Name: hspr4 (SP_BASE_PRIV)

IM Requirement: HW stack protection & shadow register optional

((MSC_CFG.HSP == 1) & (HSP_CTL.UDF == 1) & (MSC_CFG.SHADOW != 0))

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 6, 4}

This register will be activated and used in superuser mode when MISC_CTL.SP_SHADOW_EN is set to 1. When a CPU is in superuser mode (please see section 5.5), the enabled stack pointer underflow protection mechanism will use this register, instead of the SP_BASE register, to compare with any updated value to the SP register. This register is 32-bit (or 24-bit in Reduced Address Space configuration.)

It is a RW type register with the reset value defined as 0xFFFFFFFF.

10.7. Performance Monitoring Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
pfr0	PFMC0	4	0	0	271
pfr1-pfr2	PFMCn (n=1..2)	4	0	n	272
pfr3	PFM_CTL	4	1	0	272
pfr4	PFT_CTL	4	2	0	281

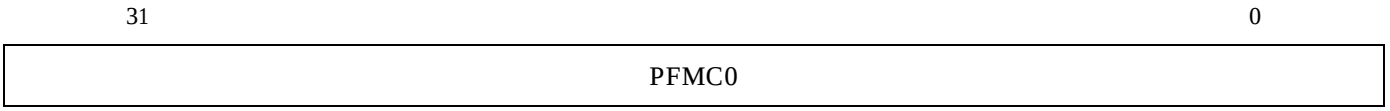
10.7.1. Performance Counter Register 0

Mnemonic Name: pfr0 (PFMC0)

IM Requirement: Performance Monitoring Optional (MSC_CFG.PFM == 1)

Access Mode: Superuser and User (when PFM_USR_EN == 1)

SR Encoding {Major, Minor, Extension}: {4, 0, 0}



This Performance Counter counts the events selected in the Performance Counter Control Register (PFM_CTL). It is readable and writable by software in superuser mode. In addition, when PFM_USR_EN is set to 1, it also becomes readable and writable by software in User mode. The counter overflows when its value wraps around from 0xFFFFFFFF to 0x0. In this case, the corresponding Overflow bit in the PFM_CTL register is also set. If the corresponding Interrupt Enable bit is set, then a pending hardware interrupt is generated to signal the interrupt controller. The software can clear the pending interrupt by writing a one to the Overflow bit in the PFM_CTL register. A counter will continue to count after the overflow event. The event counting can be filtered out individually for Superuser and User mode on each counter. So to stop a counter from counting completely, software can filter out counting from both Superuser and User mode.

When a counter is disabled, an implementation may put the counter into a low power state to preserve energy. When a counter is disabled and then re-enabled again, the counter value is whatever value existed before the re-enabled point.

Field Name	Bits	Description	Type	Reset
PFMC0	32 (31,0)	Performance event counter	RW	DC

10.7.2. Performance Counter Register 1-2

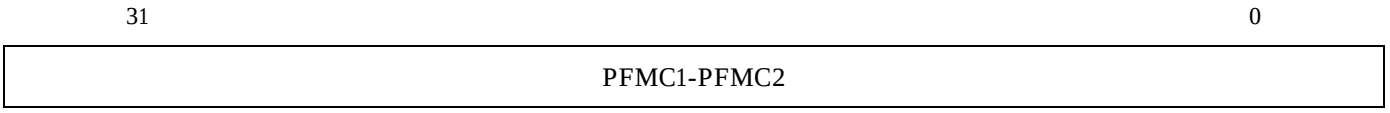
Mnemonic Name: pfr1-pfr2 (PFMC1-PFMC2)

IM Requirement: Performance Monitoring Optional (MSC_CFG.PFM == 1)

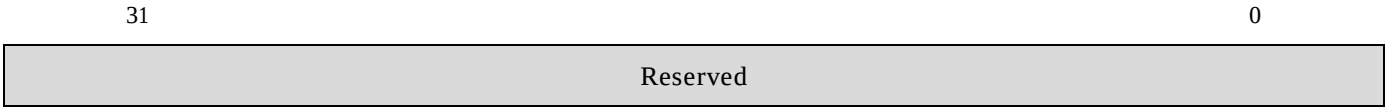
Access Mode: Superuser and User (when PFM_USR_EN == 1)

SR Encoding {Major, Minor, Extension}: {4, 0, 1-2}

For regular configuration (PFM_CTL.MIN_CFG == 0):



For minimal configuration (PFM_CTL.MIN_CFG == 1):



These Performance Counters are not available in the minimal configuration (PFM_CTL.MIN_CFG == 1) where they become “read as zero write ignored” (RAZWI). In regular configuration (PFM_CTL.MIN_CFG == 0), they count the events selected in the Performance Counter Control Register (PFM_CTL). They are readable and writable by software in Superuser mode. In addition, when PFM_USR_EN is set to 1, they also become readable and writable by software in User mode. A counter overflows when its value wraps around from 0xFFFFFFFF to 0x0. In this case, the corresponding Overflow bit in the PFM_CTL register is set. If the corresponding Interrupt Enable bit is set, then a pending hardware interrupt is generated to signal the interrupt controller. The software can clear the pending interrupt by writing a one to the Overflow bit in the PFM_CTL register. A counter will continue to count after the overflow event. The event counting can be filtered out individually for Superuser and User mode on each counter. So to stop a counter from counting completely, software can filter out counting from both Superuser and User mode.

When a counter is disabled, an implementation may put the counter into a low power state to preserve energy. When a counter is disabled and then re-enabled again, the counter value is whatever value existed before the re-enabled point.

Field Name	Bits	Description	Type	Reset
PFMC1-PFMC2	32 (31,0)	Performance event counter	RW/ RAZWI	DC

10.7.3. Performance Counter Control Register

Mnemonic Name: pfr3 (PFM_CTL)

IM Requirement: Performance Monitoring Optional (MSC_CFG.PFM == 1)

Access Mode: Superuser and User (when PFM_USR_EN == 1)

SR Encoding {Major, Minor, Extension}: {4, 1, 0}

For regular configuration (PFM_CTL.MIN_CFG == 0):

31	30 28	27 22	21 16	15	14 12	11 9	8 6	5 3	2 0
MIN_CFG	Reserved	SEL2	SEL1	SEL0	KU2-0	KS2-0	OVF2-0	IE2-0	EN2-0

For minimal configuration (PFM_CTL.MIN_CFG == 1):

31	30 16	15	14 13	12	11 10	9	8 7	6	5 4	3	2 1	0
MIN_CFG	Resv.	SEL0	Resv.	KU0	Resv.	KS0	Resv.	OVF0	Resv.	IE0	Resv.	EN0

This register is readable and writable by software in Superuser mode. In addition, when PFM_USR_EN is set to 1, it also becomes readable and writable by software in User mode.

Field name	Bits	Description	Type	Reset
EN2-0	3 (2,0)	The enable bit for each performance counter register. 0: The corresponding performance counter is turned off. 1: The corresponding performance counter is turned on and starts to count.	RW	0
IE2-0	3 (5,3)	The interrupt enable bit for each performance counter.	RW	0

Field name	Bits	Description	Type	Reset
		Value / Meaning 0: The corresponding performance counter interrupt is being masked off. 1: The corresponding performance counter interrupt is enabled for reporting.		
OVF2-0	3 (8,6)	The overflow bit for each performance counter. An interrupt request will be generated if this bit is set and the interrupt enable bit is set as well. Value / Meaning 0: The corresponding performance counter has not overflowed. 1: The corresponding performance counter has overflowed.	W1C	0
KS2-0	3 (11,9)	The Superuser mode event counting control for each performance counter. Value / Meaning 0: The event counting is on in Superuser mode for the corresponding performance counter. 1: The event counting is off in Superuser mode for the corresponding performance counter.	RW	0

<table>
<tr><th>Field name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>KU2-0</td><td>3
(14,12)</td><td>The User mode event counting control for each performance counter.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The event counting is on in User mode for the corresponding performance counter.</td></tr><tr><td>1</td><td>The event counting is off in User mode for the corresponding performance counter.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SEL0</td><td>1
(15)</td><td>The event selection for performance counter 0.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Cycles</td></tr><tr><td>1</td><td>Completed instructions</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SEL1</td><td>6
(21,16)</td><td>The event selection for the performance counter 1. The actual available events are implementation-dependent and specified in the datasheet. The definition of each event is listed in Table 31.</td><td>RW</td><td>0</td></tr>
<tr><td>SEL2</td><td>6
(27,22)</td><td>The event selection for the performance counter 2. The actual available events are implementation-dependent and specified in the datasheet. The definition of each event is listed in Table 31.</td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>3
(30,28)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

Field name	Bits	Description	Type	Reset
MIN_CFG	1 (31)	Minimum Configuration. Indicates the number of Performance Counters implemented. Value 0: Full configuration: all Performance Counters are implemented. Value 1: Minimum configuration: only Performance Counter 0 is implemented.	RO	IM

Table 31. Defined counting events for counter 1 and counter 2

SEL	C1	C2
0	Total cycles	
1	Completed instructions	
2	Conditional branches	Conditional branch mispredictions
3	Taken conditional branches	Taken conditional branch mispredictions
4	Prefetch instructions	Prefetch instructions with cache hit
5	RET inst	RET mispredict
6	JR (non-RET) instructions	Immeidate J instructions (exclude JAL)
7	JAL/JRAL instructions	Multiply instructions
8	NOP instructions	16-bit instructions
9	SCW instructions	Failed SCW instructions
10	ISB/DSB instructions	ld-after-st conflict replays
11	CCTL instructions	-
12	Taken interrupts	Exceptions taken
13	Loads completed (LMW count as 1)	Stores completed (SMW count as 1)

SEL	C1	C2
14	uITLB accesses	uITLB misses
15	uDTLB accesses	uDTLB misses
16	MTLB accesses	MTLB misses
17	Instruction cache accesses	Instruction cache misses
18	Stall cycles due to data dependency (when it is the sole cause of stall)	Stall cycles due to empty instruction queue
19	Data cache miss stall cycles	Data cache writebacks
20	Data cache accesses (LMW/SMW count as many, not including CCTL)	
21	Data cache misses	
22	Load data cache accesses (LMW count as many, not including CCTL)	Load data cache misses (LMW count as many, not including CCTL)
23	Store data cache accesses (SMW count as many, not including CCTL)	Store data cache misses (SMW count as many, not including CCTL)
24	ILM access	DLM access
25	LSU BIU cycles (data cache fill, noncacheable, write-back, write-through)	LSU BIU requests (data cache fill, noncachable, write-back, write-through)
26	HPTWK BIU cycles	HPTWK BIU requests
27	DMA BIU cycles	DMA BIU requests
28	Instruction cache fill BIU cycles	Instruction cache fill BIU requests
29	Legal unaligned data cache accesses	External events (implementation-dependent)
30	PUSH25 instructions	POP25 instructions
31	SYSCALL instructions	-
32	Parity/ECC checks	Parity/ECC errors
33	ULM bank 0 total stall cycles[1]	ULM bank 1 total stall cycles[1]
34	ULM bank 0 IFETCH stall cycles[1]	ULM bank 1 IFETCH stall cycles[1]

SEL	C1	C2
35	ULM bank 0 load/store stall cycles[1]	ULM bank 1 load/store stall cycles[1]
36	ULM bank 0 bus slave stall cycles[1]	ULM bank 1 bus slave stall cycles[1]
37	ULM bank 0 LMDMA stall cycles[1]	ULM bank 1 LMDMA stall cycles[1]
38	ULM bank 2 total stall cycles[1]	ULM bank 3 total stall cycles[1]
39	ULM bank 2 IFETCH stall cycles[1]	ULM bank 3 IFETCH stall cycles[1]
40	ULM bank 2 load/store stall cycles[1]	ULM bank 3 load/store stall cycles[1]
41	ULM bank 2 bus slave stall cycles[1]	ULM bank 3 bus slave stall cycles[1]
42	ULM bank 2 LMDMA stall cycles[1]	ULM bank 3 LMDMA stall cycles[1]
43-63	-	-

- "-" in the above table means "un-assigned value". Writing un-assigned value will result in UNPREDICTABLE event collection which is implementation-dependent.
- Note 1: The ULM bank stall cycle events (from 33~42) count both conflict stall cycles and ULM bank access wait cycles. When the ULM bank access wait cycle is 0, the stall cycle events equivalently count the stall cycles caused by ULM bank access conflicts.
- Please check with datasheet for the actual available events.

Note that the push25 and pop25 instructions have their dedicate selections, so they are not counted in the "load/store instructions completed" category. Nevertheless, push25 and pop25 are counted in the total "completed instructions" (selection 0).

Since V3 architecture (MSC_CFG.BASEV is 2), the SYSCALL instruction is included in the "completed instructions" category. In addition, JRNEZ and JRALNEZ are included in the "conditional branch" category.

If a module does not exist, there is no need to count events associated with that particular module so these events are set to 0. For example, there is no need to count miss events if cache modules do not exist. Another example is that there is no need to count TLB related accesses if MMU module does not exist. As an aside, TLB events are also set to 0 for MPU configuration (they are counted only when MMU exists). This is because all TLB accesses must go through

MPU so they are trivial and do no affect performance. In case an event is set to 0, the corresponding performance counter will keep its existing value unchanged since there is no event to trigger counting.

In Unified Local Memory (ULM) configuration, "ILM access" refers to all instruction fetches from either ILM or DLM. Similarly, "DLM access" refers to all data accesses (loads/stores) from either ILM or DLM. In addition, the sum of these two event counts is the total accesses to ULM. This change of definition is because ULM allows both instruction and data to be freely placed in either ILM or DLM (no longer limited to one particular local memory). Therefore, the new definition provides more intuitive meaning and useful measurement than the original one.

Note: The following event counts in Table 31 should not include counts during stall cycles, and should exclude, with reasonable efforts, extra counts due to replay cycles.

uITLB access	uITLB miss
uDTLB access	uDTLB miss
MTLB access	MTLB miss
I$ access	I$ miss
D$ access	D$ miss
ILM access	DLM access

For all protected RAMs, the event "parity/ECC checks" counts the number of hardware checks performed; while the event "parity/ECC errors" counts the number of errors found. Both events include speculative and non-speculative accesses. If two or more events occur simultaneously, only one event is counted in the performance counter.

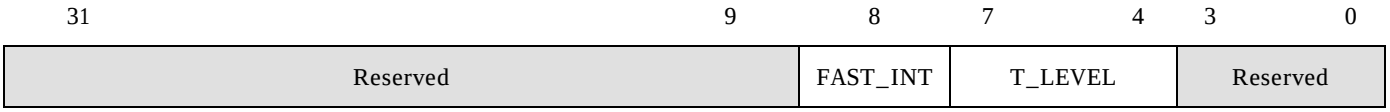
10.7.4. Performance Throttling Control Register

Mnemonic Name: pfr4 (PFT_CTL)

IM Requirement: Performance Throttling Optional (MSC_CFG.PFT == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {4, 2, 0}



This register is available only when the performance throttling, or PowerBrake, feature is supported. It controls the setting of the performance throttling feature, which balances performance and power consumption of the processor.

Field Name	Bits	Description	Type	Reset
Reserved	4 (3,0)	Reserved	RAZWI	0
T_LEVEL	4 (7, 4)	Throttling Level. The processor has the highest performance at throttling level 0 and the lowest performance at throttling level 15. Value: 0 — Meaning: Throttling Level 0 (the highest performance) Value: 1-14 — Meaning: Throttling Level 1-14 Value: 15 — Meaning: Throttling Level 15 (the lowest performance)	RW	0
FAST_INT	1 (8)	Fast interrupt response. When this field is set, the PSW.PFT_EN is automatically clear when the processor enters an interrupt handler.	RW	0

Field Name	Bits	Description	Type	Reset
Reserved	23 (31,9)	Reserved	RAZWI	0

10.8. Local Memory DMA Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
dmar0	DMA_CFG	5	0	0	284
dmar1	DMA_GCSW	5	1	0	286
dmar2	DMA_CHNSEL	5	2	0	292
dmar3	DMA_ACT	5	3	0	294
dmar4	DMA_SETUP	5	4	0	303
dmar5	DMA_ISADDR	5	5	0	308
dmar6	DMA_ESADDR	5	6	0	311
dmar7	DMA_TCNT	5	7	0	313
dmar8	DMA_STATUS	5	8	0	316
dmar9	DMA_2DSET	5	9	0	321
dmar10	DMA_2DSCTL	5	9	1	324
dmar11	DMA_RCNT	5	7	1	326
dmar12	DMA_HSTATUS	5	8	1	328

10.8.1. DMA Configuration Register

Mnemonic Name: dmar0 (DMA_CFG)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 0, 0}

Per DMA Channel: No

31 16	15 4	3	2	1 0
VER	Reserved	2DET	UNEA	NCHN

The DMA Configuration Register specifies the implemented DMA features in an implementation.

Field Name	Bits	Description	Type	Reset
NCHN	2 (1,0)	The number of DMA channels implemented. Value: 0 — 1 DMA channel; 1 — 2 DMA channels; 2 — Reserved; 3 — Reserved	RO	IM
UNEA	1 (2)	Indicates if the Un-aligned External Address transfer feature is implemented. Value: 0 — The feature is not implemented.; 1 — The feature is implemented.	RO	IM
2DET	1 (3)	Indicates if the 2-D Element Transfer feature is implemented. Value: Meaning	RO	IM

Field Name	Bits	Description	Type	Reset
		0 The feature is not implemented. 1 The feature is implemented.		
Reserved	12 (15,4)	Reserved	RAZWI	0
VER	16 (31,16)	Indicates the DMA architecture and implementation version. It is partitioned into a high 12-bit part and a low 4-bit part. The 12-bit part is for major architecture changes and the 4-bit part is for minor implementation updates. The DMA sub-action command is added in DMA architecture version 2 (0x0200) and above.	RO	IM

10.8.2. DMA Global Control and Status Word Register

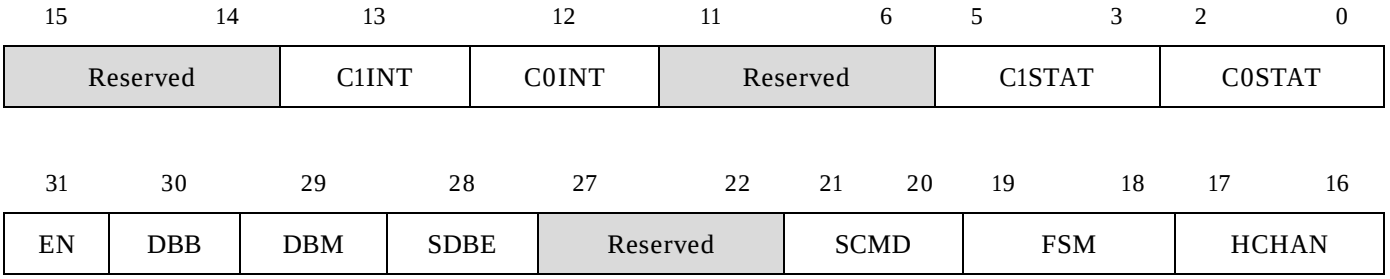
Mnemonic Name: dmar1 (DMA_GCSW)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 1, 0}

Per DMA Channel: No



The DMA Global Control and Status Word Register enables DMA engine logic. The enable state will be exported out of the core as information for power management. When the DMA engine is enabled, it also collects status information from the two DMA channels. Software can use this register to determine if there is any “Idle” DMA channel to use.

Disabling the DMA global enable bit will immediately put the two DMA channels in the “Idle” state and stop all ongoing DMA transactions.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>C0STAT</td><td>3
(2,0)</td><td>The current state for the DMA channel 0.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Idle (value after a “reset” command)</td></tr><tr><td>1</td><td>Run</td></tr><tr><td>2</td><td>Pended</td></tr><tr><td>3</td><td>Stop/Error</td></tr><tr><td>4</td><td>Complete</td></tr></table></td><td>RO</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		5-7 | -		
C1STAT	3 (5,3)	The current state for the DMA channel 1. Please see the above field (C0STAT) for value definitions.	RO	0
Reserved	6 (11,6)	Reserved	RAZWI	0
C0INT	1 (12)	Interrupt flag for the DMA channel 0. Indicates if an interrupt has been generated from DMA channel 0. Refer to Section 8.2 "Basic Rules of Operation" for more information. Value | Meaning 0 | No interrupt 1 | An interrupt has been generated.	RO	0
C1INT	1 (13)	Interrupt flag for the DMA channel 1. Indicates if an interrupt has been generated from DMA channel 1. Please see the above field (C0INT) for value definitions.	RO	0
Reserved	2 (15,14)	Reserved	RAZWI	0
HCHAN (exists only in DMA version >= 2.0)	2 (17,16)	Head channel number. If the head channel is in the "Complete" state, writing the DMA_HSTATUS register makes this field automatically point to the next channel. Otherwise, the field is not changed. Refer to Section 8.5 "DMA Channel Queue" for more information. Value | Meaning 0 | Channel 0 is the head channel. 1 | Channel 1 is the head channel.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>2</td><td>Reserved</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table>
</td><td></td><td></td></tr>
<tr><td>FSM
(exists only in DMA version >= 2.0)</td><td>2
(19,18)</td><td>Fast-start mode field. Please see Section 8.6 “Action-command Mode and Fast-start Mode” for more information.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>DMA engine is in the action-command mode.</td></tr>
<tr><td>1</td><td>DMA engine is in the fast-start mode and uses the DMA_TCNT register as the trigger register.</td></tr>
<tr><td>2</td><td>DMA engine is in the fast-start mode and uses the DMA_ISADDR register as the trigger register.</td></tr>
<tr><td>3</td><td>DMA engine is in the fast-start mode and uses the DMA_ESADDR register as the trigger register.</td></tr>
</table>
</td><td>RW</td><td>0</td></tr>
<tr><td>SCMD
(exists only in DMA version >= 2.0)</td><td>2
(21,20)</td><td>DMA Sub-Action Command used in the fast-start mode. Writing any value into the trigger register will use this field as the sub-action command to activate the channel. Please see Section 8.6 “Action-command Mode and Fast-start Mode” for more information.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>NoCond</td></tr>
</table>
</td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		1 EnQ 2 WaitDBS 3 EnQWait		
Reserved	6 (27,22)	Reserved	RAZWI	0
SDBE (exists only in DMA version >= 2.0)	1 (28)	Shadow DBM and DBB bits Enable control. 0: The DBM and DBB bits are not shadowed. 1: The DBM and DBB bits are shadowed. **Note:** If the Double-Buffer Mode is not supported, this bit is tied to zero.	RW	0
DBM (exists only in DMA version >= 2.0)	1 (29)	Double-Buffer Mode enable control for the data local memory. This bit is a shadow bit of the DBM bit in the DLMB register. When this bit is read, it will get the value of the DBM bit in the DLMB register if the SDBE bit is set. When this bit written, it will update the DBM bit in the DLMB register if the SDBE bit is set. 0: Double-buffer mode is disabled. 1: Double-buffer mode is enabled. Note: If the Double-Buffer Mode is not supported, this bit is tied to zero.	RW	0
DBB (exists only in DMA version	1 (30)	Double-Buffer Bank which can be accessed by the processor. This bit is a shadow bit of the DBB bit in the DLMB register. When this bit is read, it will	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>>= 2.0)</td><td></td><td>get the value of the DBB bit in the DLMB register if the SDBE bit is set. When this bit written, it will update the DBB bit in the DLMB register if the SDBE bit is set.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Double-buffer bank 0 is accessed by the processor.</td></tr><tr><td>1</td><td>Double-buffer bank 1 is accessed by the processor.</td></tr></table>Note: If the Double-Buffer Mode is not supported, this bit is tied to zero.</td><td></td><td></td></tr>
<tr><td>EN</td><td>1
(31)</td><td>DMA engine global enable bit. This state will be exported out of AndesCore as one of the information for external power and frequency control.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>DMA engine is disabled. No DMA transfer is possible. Any write to DMA Action Register will be ignored. Before disabling this bit, users should make sure all channels are in the “Idle” state; Otherwise, the behavior of the DMA engine is implementation-dependent. In such case, the software should use the following steps to reset the DMA engine:
■ Enable this bit.
■ Issue “Stop” commands for</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		all channels. ■ Make sure all channels really stop. ■ Issue “Reset” commands to all channels. 1: DMA engine is enabled. DMA Action Register can be programmed to perform DMA operations.		

10.8.3. DMA Channel Selection Register

Mnemonic Name: dmar2 (DMA_CHNSEL)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 2, 0}

Per DMA Channel: No

31 ... 2	1 ... 0
Reserved	CHAN

The DMA Channel Selection Register is used to control the accessibility of the DMA per channel registers. Once the CHAN field is set to a particular DMA channel, all later per channel register accesses will occur on registers belong to that particular channel.

After a write to this register, a Data Serialization Barrier (DSB) instruction may be required before reading any register in the selected channel. This limitation is removed for DMA architecture version greater or equal 2.0 as shown in the following table.

DMA Architecture	Requirement for the DSB instruction
Version < 2.0	■ The DSB instruction is needed by the read to any register of the newly selected channel to get the latest value.
Version >= 2.0	■ The read is guaranteed to get the value of any register of the newly selected channel.

Please refer to Section 8.7 "Essential Data Serialization Barrier Instructions" for more information.

To write a register of the selected channel after a write to this register, the DSB instruction does not need to be present in between these two write operations. All implementations will guarantee that after a channel selection operation the newly selected channel can be written immediately.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CHAN</td><td>2
(1,0)</td><td>Selected channel number.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Channel 0 is selected.</td></tr><tr><td>1</td><td>Channel 1 is selected.</td></tr><tr><td>2</td><td>Reserved</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>30
(31,2)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

10.8.4. DMA Action Register

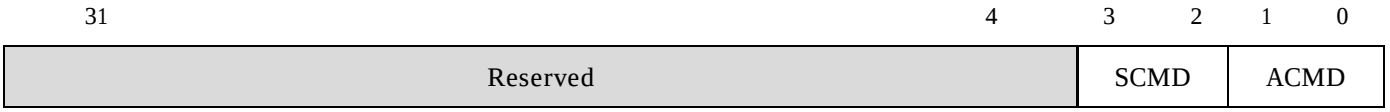
Mnemonic Name: dmar3 (DMA_ACT)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 3, 0}

Per DMA Channel: Yes



The DMA Action Register activates/de-activates the transfer operation of the DMA engine based on the value written into the register.

Field Name	Bits	Description	Type	Reset
ACMD	2 (1,0)	DMA Main Action Command. 0: No operation 1: Start the DMA transfer 2: Stop the DMA transfer 3: Reset DMA status/error	WO	0
SCMD (exists only in DMA version >= 2.0)	2 (3,2)	DMA Sub-action Command. This is used when the DMA architecture version is equal to or larger than 2.0. 0: NoCond 1: EnQ 2: WaitDBS 3: EnQWait	WO	0

Field Name	Bits	Description	Type	Reset
Reserved	28 (31,4)	Reserved	RAZWI	0

DMA Main Action Command

Details of DMA Main Action Command are described below:

- NOP
  - The “NOP” (No Operation) main command orders the DMA channel to perform no operation for the main action command.
  - The "Nop" command with the associated SCMD sub-action command is always performed regardless of the channel state.
- Start
  - The “Start” main command uses with the sub-action command field (SCMD) to activate the DMA transfer operation from the “Idle” state or the “Stop/Error” state. Starting from DMA version 2, the “Complete” state can also be used as a starting state for this command.
  - The interrupt flag and error flags of the channel are cleared when this command is issued in the “Stop/Error” state. Starting from DMA version 2, these flags are also cleared for the command issued in the “Complete” state.
  - Starting from DMA version 2, the “Start” command will cause a DMA Transfer Disruption error when the DMA channel is in the “Run” state or the “Pended” state.
  - A “Channel Ready” event may be required depends on the implementations while the DMA channel is activated.
  - After the DMA channel is activated, the channel will be transitioned into one of the following states:
    - The “Run” state: if no errors are detected and no wait events are required.
    - The “Pended” state: if no errors are detected but still waiting for some required events.
    - The “Stop/Error” state: if any errors are detected.

- Stop
  - The “Stop” main command uses with the sub-action command field (SCMD) to deactivate the DMA transfer operation from the “Run” or “Pended” states.
  - The “Stop” main command will perform no operation when the DMA channel is in the “Idle”, “Complete” or “Stop/Error” states, but the sub-action command still takes the effect.
  - When the DMA transfer operation finally stops, the DMA channel will be put into the “STOP” state. Software needs to check the status of the DMA channel to make sure it really stops after it executes this command.
  - When the channel finishes the transfer (i.e., DMA_TCNT is zero) and receives a “Stop” command at the same time, the next state of the channel should be the “Complete” state.
  - When the DMA channel stops, the following registers contain the values that can restart the remaining DMA transfer transactions.
    - DMA Internal Address Start Register (DMA_ISADDR).
    - DMA External Address Start Register (DMA_ESADDR).
    - DMA Transfer Element Count Register (DMA_TCNT).
    - DMA 2D Startup Control Register (DMA_2DSCTL).
- Reset
  - The “Reset” main command uses with the sub-action command field (SCMD) to clear out the channel’s status, interrupt, and error information and put the channel into the “Idle” state when the DMA channel is in the “Complete” or “Stop/Error” states.
  - The “Reset” main command will perform no operation when the DMA channel is in the “Idle” state, but the sub-action command still takes the effect.
  - Starting from version 2, the “Reset” command will cause a DMA Transfer Disruption error when the DMA channel is in the “Run” state or the “Pended” state.

DMA Sub Action Command

The sub-action commands assist the main action commands to perform extra operations. The details of DMA sub-action command are defined below:

- NoCond:
  - It stands for “No Condition.” This sub-action command indicates that the DMA channel

can execute the main action command with no extra condition (wait event).

- The combination of “NOP” main command and “NoCond” sub command performs no operation.

- EnQ:
  - The "EnQ" sub-action command, used along with any ACMD command, operates the channels in a channel queue fashion. The ACMD command operates on the current channel specified by the DMA_CHNSEL.CHAN field, and the DMA_CHNSEL.CHAN field switches to the next channel such that all subsequent DMA programming will operate on the next channel. Essentially, the ACMD command is enqueued to the DMA channels.
  - All subsequent read to the DMA_STATUS register get the status of the newly selected channel, which is pointed to by the new DMA_CHNSEL.CHAN value, immediately without the need for any barrier instructions (the DSB/ISB instructions).
- WaitDBS:
  - It stands for “Wait for DLM Bank Switch.” This sub-action command indicates that the DMA channel must wait for a DLM bank switching event before it can move to the “Run” state.
  - The "WaitDBS" sub-action command can only be used along with the "Start" command. Otherwise, a Reserved Value exception will be asserted.
- EnQWait:
  - The "EnQWait" sub-action command is the merge of the EnQ sub-action command and the Wait-DBS sub-action command. When it is used along with the "Start" command, the current channel transition to the "Pended" state to wait for the "DLM bank switching" event before it enters the "Run" state. Meanwhile, the DMA_CHNSEL.CHAN field switches to the next channel such that all subsequent DMA programming will operate on the next channel.
  - The "EnQWait" sub-action command can only be used along with the "Start" command. Otherwise, a "Reserved Value" exception will be asserted.

The wait events are the events waited by the DMA channel before the channel enters the “Run” state. The details of event are defined below:

- Channel Ready Event:

- The channel ready event is an event that the channel is ready to be opened.
- The event must be waited if any one of the following conditions is true:
  - Any other channel is in the “Run” state or the “Pended” state.
  - The channel is pended due to any implementation reason.

- DLM Bank Switching Event:
  - This event is only relevant when the SCMD sub-action command is the "WaitDBS" or the "En-QWait" sub-action command and the ACMD command is the "Start" command. It is ignored for all other command/sub-action command combinations.
  - The DBS Event is triggered when the DLMB.DBB (or the shadowed DMA_GCSW.DBB) bit is toggled when DLM is in the double buffer mode (when the DLMB.DBM (or the shadowed DMA_GCSW.DBM) bit is one)
  - When multiple channels wait for the DBS event, they must follow their issue order to wait for (to receive) the event.
  - When the local memory is in the ULM mode or when the DLM is not in the double-buffer mode, there will be no DBS events and the DMA channels will hang when they are asked to wait for the events with the WaitDBS and EnQWait sub-action commands. The two sub-action commands should be avoided under these scenarios.

The programmer should know what event the channel is waiting for according to the DMA_STATUS.WE bit if the channel is in the “Pended” state.

The mapping of commands and operations is shown in Table 32. Some sub commands are used to support the implementation of a DMA command queue. Please refer to Section 8.5 "DMA Channel Queue” for more information.

Table 32. Mapping of DMA commands and operations

Command		Operation after the command	
ACMD	**SCMD**	**Wait event**	**DMA_CHNSEL**
NOP	NoCond	-	No change
	EnQ	-	The next channel
	-	-	-
	-	-	-

Command		Operation after the command	
ACMD	**SCMD**	**Wait event**	**DMA_CHNSEL**
Start	NoCond	▪ A channel ready event	No change
	EnQ	▪ A channel ready event	The next channel
	WaitDBS	▪ A channel ready event ▪ A DLM bank switching event	No change
	EnQWait	▪ A channel ready event ▪ A DLM bank switching event	The next channel
Stop	NoCond	-	No change
	EnQ	-	The next channel
	-	-	-
	-	-	-
Reset	NoCond	-	No change
	EnQ	-	The next channel
	-	-	-
	-	-	-

"-" : Reserved value.

- ACMD:

  The DMA action command written to the "ACMD" field of the DMA_ACT register.

- SCMD:

  The DMA sub-action command written to the "SCMD" field of the DMA_ACT register.

- Wait event:

  The events a just activated DMA channel transfer transaction has to wait for before a real memory transfer starts.

- DMA_CHNSEL:

  The new value of The DMA_CHNSEL register after a DMA channel transfer is activated.

Writing a reserved value into the SCMD field will cause a “Reserved Value” exception. If the sub command is a reserved value, the main and sub commands will not take effect except the “Reserved Value” exception.

If the data local memory is not in the double buffer mode, the “WaitDBS” or “EnQWait” sub-action commands are still accepted to activate a channel. However, this way would cause the channel hangs on in the “Pended” state. To detect and fix this error, the programmer can check the DLMB and DMA_STATUS registers.

The commands accepted by the DMA for each channel state are summarized in Table 33. If the command is not shown in this table, writing the command into the DMA_ACT register will cause a “Reserved Value” exception.

Table 33. Valid commands for each state

Input				Output			
Current State	ACMD	SCMD	Version	Possible Next State	DERR	Effect of SCMD	
						DBS Event	DMA_CHNSEL
Idle	NOP	NoCond, EnQ	All	Idle	-	-	Depend
	Start	NoCond, EnQ, WaitDBS, EnQWait	All	Pended, Run	-	Depend	
			All	Stop/Error	No	Ignore	
	Stop	NoCond, EnQ	All	Idle	-	-	
	Reset	NoCond, EnQ	All	Idle	-	-	
Pended	NOP	NoCond, EnQ	All	Pended	-	-	
	Start	NoCond, EnQ, WaitDBS, EnQWait	< 2.0	Pended	-	Ignore	
			>= 2.0	Stop/Error	Yes	Ignore	
	Stop	NoCond, EnQ	All	Stop/Error	-	-	
	Reset	NoCond, EnQ	< 2.0	Pended	-	-	
			>= 2.0	Stop/Error	Yes	-	
Run	NOP	NoCond, EnQ	All	Run	-	-	

Input				Output			
Current State	ACMD	SCMD	Version	Possible Next State	DERR	Effect of SCMD	
						DBS Event	DMA_ CHNSEL
	Start	NoCond, EnQ, WaitDBS, EnQWait	< 2.0	Run	-	Ignore	
			>= 2.0	Stop/Error	Yes	Ignore	
	Stop	NoCond, EnQ	All	Stop/Error	-	-	
	Reset	NoCond, EnQ	< 2.0	Run	-	-	
			>= 2.0	Stop/Error	Yes	-	
Stop /Error	NOP	NoCond, EnQ	All	Stop/Error	No	-	
	Start	NoCond, EnQ, WaitDBS, EnQWait	All	Pended, Run	-	Depend	
			All	Stop/Error	No	Ignore	
	Stop	NoCond, EnQ	All	Stop/Error	No	-	
	Reset	NoCond, EnQ	All	Idle	-	-	
Complete	NOP	NoCond, EnQ	All	Complete	-	-	
	Start	NoCond, EnQ, WaitDBS, EnQWait	< 2.0	Complete	-	Ignore	
			>= 2.0	Pended, Run	-	Depend	
			All	Stop/Error	No	Ignore	
	Stop	NoCond, EnQ	All	Complete	-	-	
	Reset	NoCond, EnQ	All	Idle	-	-	
Cases are not in the above				"Reserved Value" Exception			

- Current State:

  The current state of the channel before the command is issued.

- ACMD:

  The DMA action command written to the “ACMD” field of the DMA_ACT register.

- SCMD:

  The DMA sub-action command written to the “SCMD” field of the DMA_ACT register. When the architecture version of the DMA is less than 2.0, the SCMD field is always the “NoCond” sub command.

- Version:

  The architecture version of the DMA.

- Possible Next State:

  The possible nest state of the channel after the command is issued.

- DERR:

  This field indicates whether the command will cause the disruption error for the state.

- DBS Event:

  Wait for a DLM bank switching event for the command after the command is issued.

  - Depend: The channel must wait for the event depending on the sub command.
  - Ignore: The channel ignore the indication of the sub command for the event.
  - -: The field is empty.

- DMA_CHNSEL:

  The new value of The DMA_CHNSEL register after the command is issued. The new value depends on the sub command. If the sub command indicates that the DMA_CHNSEL register should point at the next channel after the command is issued, the DMA_CHNSEL register will point at the next channel even the disruption occurs.

10.8.5. DMA Setup Register

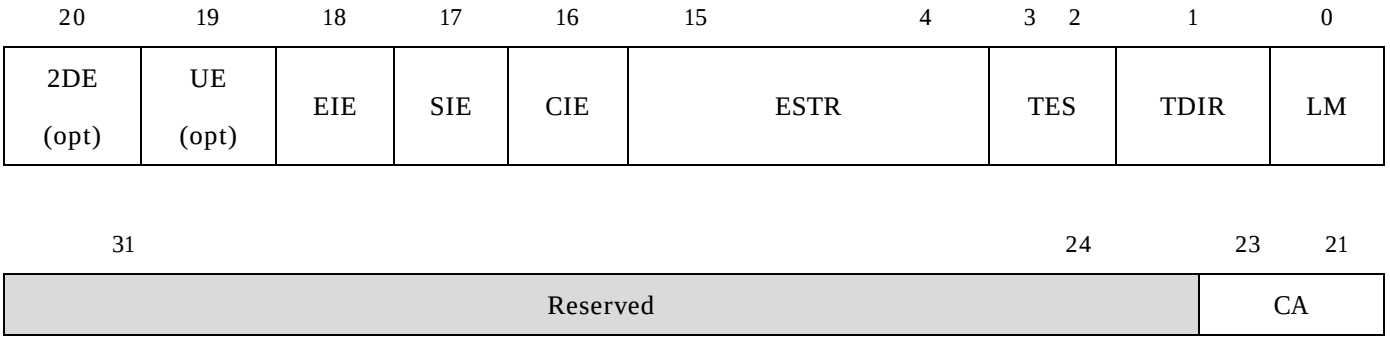
Mnemonic Name: dmar4 (DMA_SETUP)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 4, 0}

Per DMA Channel: Yes



The DMA Setup Register specifies all the essential DMA transfer parameters for the DMA engine to perform its transfer work. When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

Field Name	Bits	Description	Type	Reset
LM	1 (0)	Local Memory Selection. Value: 0 — ILM is selected for DMA transfer. Value: 1 — DLM is selected for DMA transfer.	RW	0
TDIR	1 (1)	Transfer Direction. Value: 0 — From external memory to LM.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>From LM to external memory.</td></tr></table></td><td></td><td></td></tr>
<tr><td>TES</td><td>2
(3,2)</td><td>Transfer Element Size.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Byte</td></tr><tr><td>1</td><td>2 Bytes (Half-word)</td></tr><tr><td>2</td><td>4 Bytes (Word)</td></tr><tr><td>3</td><td>8 Bytes (Double-word) optional</td></tr></table>When the 8 bytes transfer element size is not supported, writing this value to this field will generate a reserved value exception, or unpredictable results for some implementations.</td><td>RW</td><td>0</td></tr>
<tr><td>ESTR</td><td>12
(15,4)</td><td>External memory transfer Stride. This is the byte value increments on the external memory address between successive DMA transfers. A value of 0 will keep the external memory address to be the same for all transfers. This stride value has to be aligned to the transfer element size if the optional Unaligned External Address feature is not implemented; otherwise an Un-aligned error will be reported in the DMA Status Register (i.e., DMA_STATUS.EUNA will be set to 1) when the DMA transfer starts. If the optional Unaligned External Address feature is implemented and the ESTR value is not aligned to the transfer element size, an Un-aligned error will be reported in the DMA Status Register (i.e., DMA_STATUS.EUNA will be set to 1) when DMA_SETUP.UE is zero; when DMA_SETUP.UE is one, this condition is not considered to be an error, so no error will be</td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>reported. In the 2-D Element transfer mode, this stride is the “width stride” of a transfer. And an additional “height stride” is needed and is defined in the DMA 2D Setup Register. Please see section 10.8.10 for more detail.</td><td></td><td></td></tr>
<tr><td>CIE</td><td>1
(16)</td><td>Interrupt Enable on Completion.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No interrupt on DMA transfer completion.</td></tr><tr><td>1</td><td>Generate interrupt on DMA transfer completion.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SIE</td><td>1
(17)</td><td>Interrupt Enable on explicit Stop. This may be necessary since after an explicit Stop command is being issued to the DMA engine, the DMA engine may not stop immediately until completing its current outstanding bus transactions.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No interrupt when the DMA transfer is explicitly stopped by a STOP command.</td></tr><tr><td>1</td><td>Generate interrupt when the DMA transfer is explicitly stopped by a STOP command.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>EIE</td><td>1
(18)</td><td>Interrupt Enable on Error.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No interrupt on DMA errors</td></tr><tr><td>1</td><td>Generate interrupt on DMA</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		errors		
UE (optional)	1 (19)	This is used when the Un-aligned External Address (UEA) Feature is implemented. This bit is used to enable the Un-aligned External Address feature. 0: The UEA feature is disabled. 1: The UEA feature is enabled.	RW	0
2DE (optional)	1 (20)	This is used when the 2-D External Transfer (2DET) feature is implemented. This bit is used to enable the 2-D External Transfer feature. 0: The 2DET feature is disabled. 1: The 2DET feature is enabled.	RW	0
CA	3 (23,21)	Cacheability attribute of external memory access. This is used for external cache memory management of the DMA transferred data. The cacheability attribute is defined as follows: 0: Device space 1: Device space, write bufferable/coalescable 2: Non-cacheable memory 3: - 4: cacheable, write-back, write-allocate memory (shared)	RW	0

Field Name	Bits	Description	Type	Reset
		5 — cacheable, write-through, no-write-allocate memory (shared) 6 — cacheable, non-shared, write-back, write allocate memory 7 — cacheable, non-shared, write-through, no-write-allocate		
Reserved	8 (31,24)	Reserved	RAZWI	0

10.8.6. DMA Internal Start Address Register

Mnemonic Name: dmar5 (DMA_ISADDR)

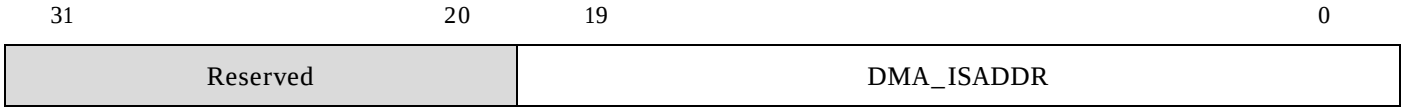
IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

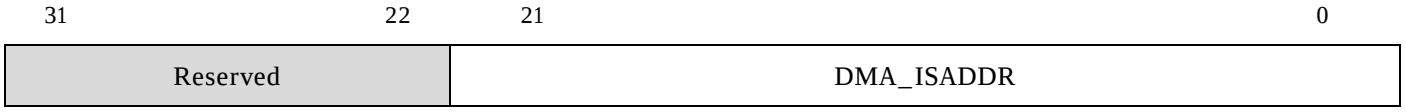
SR Encoding {Major, Minor, Extension}: {5, 5, 0}

Per DMA Channel: Yes

For the DMA architecture (version < 2.0):



For the DMA architecture (version >= 2.0):



The DMA External Start Address Register specifies the starting physical address offset from the local memory base address the data transfer will happen in local memory. Based on the real local memory size and the double buffer mode control, some of the upper bits of this address offset from Bit-21 to Bit-8 will be ignored by an AndesCore. Writing any value into these upper bits will update these bits, but these upper bits will be ignored when an AndesCore uses the register to index the local memory. The effective bit range is listed in the following table.

LM Size	Non-double-buffer range	Double buffer range
1KB	(9,0)	(8,0)
2KB	(10,0)	(9,0)
4KB	(11,0)	(10,0)
8KB	(12,0)	(11,0)
16KB	(13,0)	(12,0)

LM Size	Non-double-buffer range	Double buffer range
32KB	(14,0)	(13,0)
64KB	(15,0)	(14,0)
128KB	(16,0)	(15,0)
256KB	(17,0)	(16,0)
512KB	(18,0)	(17,0)
1024KB	(19,0)	(18,0)
2048KB	(20,0)	(19,0)
4096KB	(21,0)	(20,0)

The physical address offset specified must be aligned to the transfer element size set in the DMA Setup Register; otherwise, an Un-aligned error will be reported in the DMA Status Register.

An Out-Of-Range error should be reported if any of the following conditions is true.

- The value of the register is not in the valid range while the channel is activated.
- The address offset increment overflows during a DMA transfer.

Either of the instruction/data local memory is not supported, the size of the local memory should be considered as zero. In this case, an Out-Of-Range error should be reported while the channel is activated to accesses the corresponding local memory.

During a DMA transfer, if the address offset increment overflows, an Out-Of-Range error will be generated. The behavior of moving data into local memory should meet the following rules.

- For the DMA version (< 2.0), the data of local memory around the zero offset may be corrupted due to this error. Software should be careful to avoid this problem.
- For the DMA version (>= 2.0), the data of local memory around the zero offset must not be accessed.

When the DMA channel is in the RUN state, the content of this register allows to be changed by the DMA channel. When the DMA channel is stopped, by a STOP command or an error, this register will contain the address that is required to restart the transaction. When the DMA

channel completes its transfer, this register will contain the next transfer address offset of the last transfer of the transaction.

When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

This is a RW type register. The reset value of this register is DC.

The DMA_ISADDR register can be used as the trigger register in the fast-start mode. In the fast-start mode, writing any value into the trigger register will induce the DMA to issue a “Start” command to activate the channel. Please refer to Section 8.6 ”Action-command Mode and Fast-start Mode” for more information.

Implementation Note: If the DMA_ISADDR register is used as the trigger register, a Data Serialization Barrier (DSB) instruction is required between a write instruction (MTSR) to the DMA_ISADDR register and a following read instruction (MFSR). Please refer to Section 8.7 "Essential Data Serialization Barrier Instructions” for more information.

10.8.7. DMA External Start Address Register

Mnemonic Name: dmar6 (DMA_ESADDR)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 6, 0}

Per DMA Channel: Yes

31	0
DMA_ESADDR	

The DMA External Start Address Register specifies the starting physical address in the external memory outside the core the data transfer will happen. And the address specified must be aligned to the transfer element size set in the DMA Setup Register if the optional Unaligned External Address feature is not implemented; otherwise, an Un-aligned error will be reported in the DMA Status Register (i.e., DMA_STATUS.EUNA will be set to 1). If the optional Unaligned External Address feature is implemented and the address specified is not aligned to the transfer element size, an Un-aligned error will be reported in the DMA Status Register (i.e., DMA_STATUS.EUNA will be set to 1) when DMA_SETUP.UE is zero; when DMA_SETUP.UE is one, this condition is not considered to be an error, so no error will be reported.

When the DMA channel is in the RUN state, the content of this register allows to be changed by the DMA channel. When the DMA channel is stopped, by a STOP command or an error, this register will contain the address that is required to restart the transaction. When the DMA channel completes its transfer, this register will contain the next transfer address of the last transfer of the transaction. To be more specific, for 1-D transfer, the next transfer address is the last transfer address plus DMA_SETUP.ESTR. For 2-D transfer, the next transfer address is the last transfer address plus DMA_2DSET.HTSTR or, if the last transfer address is not the last element of a row, the last transfer address plus DMA_SETUP.ESTR.

When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

This is a RW type register. The reset value of this register is DC.

The DMA_ESADDR register can be used as the trigger register in the fast-start mode. If the DMA is in the fast-start mode, writing any value into the trigger register will cause that the DMA issues a “Start” command to activate the channel. Please refer to Section 8.6 ”Action-command Mode and Fast-start Mode” for more information.

Implementation Note: If the DMA_ESADDR register is used as the trigger register, a Data Serialization Barrier (DSB) instruction is required between a write instruction (MTSR) to the DMA_ESADDR register and a following read instruction (MFSR). Please refer to Section 8.7 "Essential Data Serialization Barrier Instructions” for more information.

10.8.8. DMA Transfer Element Count Register

Mnemonic Name: dmar7 (DMA_TCNT)

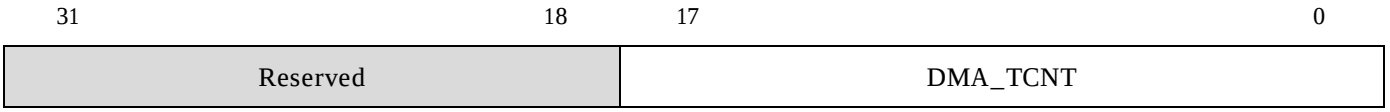
IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

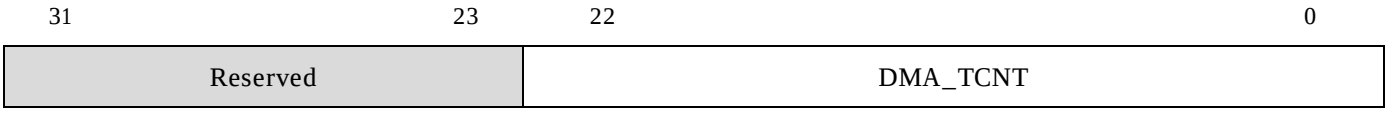
SR Encoding {Major, Minor, Extension}: {5, 7, 0}

Per DMA Channel: Yes

For the DMA architecture (version < 2.0):



For the DMA architecture (version >= 2.0):



The DMA Transfer Element Count Register specifies the total number of element transfers the DMA engine will perform. During a DMA transfer transaction, at any time if the internal transfer address offset overflows, an Out-Of-Range error will be generated.

When the DMA channel is in the RUN state, the content of this register allows to be changed by the DMA channel. When the DMA channel is stopped, by a STOP command or an error, this register will contain the remaining transfer count that is required to complete the transaction. When the DMA channel completes its transfer, this register will contain a value of 0.

When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

No matter what the next state of the channel is, the DMA_TCNT register will be automatically refilled by the DMA Refill Element Count Register (DMA_RCNT) while the channel is activated by a "Start" command and the following conditions are true.

- The DMA architecture version is equal to or larger than 2.0.
- The DMA_TCNT register is equal to zero.

If the refilled DMA_TCNT register is zero, no transfer will happen after the channel is activated. If the refilled DMA_TCNT register is not zero, the number of element defined in the refill DMA_TCNT register will be transferred.

When the DMA architecture version is smaller than 2.0, the channel will directly start with the DMA_TCNT register after the channel is activated. If the DMA_RCNT register is zero, no transfer will happen. Otherwise, the number of element defined in the DMA_TCNT register will be transferred.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DMA_TCNT</td><td>23
(22,0)</td><td>The DMA transfer element count.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>See the above description.</td></tr><tr><td>> 0</td><td>The number of element transfers the DMA will perform.</td></tr></table>Note: The bits (18~22) are reserved in the DMA architecture (version < 2.0).</td><td>RW</td><td>DC</td></tr>
<tr><td>Reserved</td><td>9
(31,23)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

The DMA_TCNT register can be used as the trigger register in the fast-start mode. In the fast-start mode, writing any value into the trigger register will induce the DMA to issue a "Start" command to activate the channel. Please refer to Section 8.6 "Action-command Mode and Fast-start Mode" for more information.

Implementation Note: A Data Serialization Barrier (DSB) instruction is required between a write instruction (MTSR) to the register and a following read instruction (MFSR) to the status of the same channel if the register is used as the trigger register. Refer to Section 8.7 "Essential Data Serialization Barrier Instructions" for more information.

10.8.9. DMA Status Register

Mnemonic Name: dmar8 (DMA_STATUS)

IM Requirement: DMA optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 8, 0}

Per DMA Channel: Yes

31	30 ... 12	11	10	9	8	7	6	5	4	3	2 ... 0
WE	Reserved	EECC	ELM	ESUP	EBUS	IOOR	IUNA	EUNA	DERR	STUNA	STAT

The DMA Status Register is a read-only register. It shows the current status, state and error, in the selected DMA channel. A “Reset” action command written to the DMA Action Register will change the DMA state to “Idle” and clears all error indications.

Implementation Note: If the write instruction (MTSR) of the DMA_ACT register and the read instruction (MFSR) of this register access on the same channel, a “data serialization barrier” (DSB) instruction may need to be present in between these two instructions. The requirement for the DSB instruction is listed in the following table.

DMA Architecture	Requirement for the DSB instruction
Version < 2.0	■ The DSB instruction is needed in between the write and the read.
Version >= 2.0	■ The DSB instruction is not needed in between the write and the read. The read is guaranteed to get the status of the newly selected channel immediately.

Refer to Section 8.7 "Essential Data Serialization Barrier Instructions” for more information.

The various error flags/indications in this register are only valid when the DMA channel state is in the Stop/Error state.

Field Name	Bits	Description	Type	Reset
STAT	3 (2,0)	The current state of the DMA channel. Value: 0 — Idle (value after a "reset" command); 1 — Run; 2 — Pended; 3 — Stop/Error; 4 — Complete; 5-7 — -	RO	0
STUNA	1 (3)	Indicates if an Un-aligned error has occurred on the External Stride value. Value: 0 — No error (value after a "reset" command).; 1 — Errors have occurred.	RO	0
DERR	1 (4)	DMA Transfer Disruption Error. Value: 0 — No error (value after a "reset" command).; 1 — Errors have occurred.	RO	0
EUNA	1 (5)	Indicates if an Un-aligned error has occurred on the address to the External bus (not local memory bus). Value: 0 — No error (value after a "reset" command).	RO	0

Field Name	Bits	Description	Type	Reset
		1 — Errors have occurred.		
IUNA	1 (6)	Indicates if an Un-aligned error has occurred on the address for local memory. 0 — No error (value after a "reset" command). 1 — Errors have occurred.	RO	0
IOOR	1 (7)	Indicates if an Out-Of-Range error has occurred on the address for local memory. Refer to Section 10.8.6 "DMA Internal Start Address Register" for more information. 0 — No error (value after a "reset" command). 1 — Errors have occurred.	RO	0
EBUS	1 (8)	Indicates if a Bus Error has occurred on a DMA transfer. 0 — No error (value after a "reset" command). 1 — Errors have occurred.	RO	0
ESUP	1 (9)	Indicates if a DMA setup error has occurred. This error typically occurs when a DMA transfer starts with a value of 0 in DMA_2DSET.WECNT or DMA_2DSCTL.STWECNT.	RO	0

Field Name	Bits	Description	Type	Reset
		<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No error (value after a “reset” command).</td></tr><tr><td>1</td><td>Errors have occurred.</td></tr></table>		
ELM	1 (10)	Indicates if a Local Memory error has occurred during the DMA transfer. This type of error does not include parity/ECC errors as they are indicated separately at “EECC” field below (bit 11). <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No error (value after a “reset” command).</td></tr><tr><td>1</td><td>Errors have occurred.</td></tr></table>	RO	0
EECC	1 (11)	Indicates if an uncorrectable parity/ECC error has occurred during the DMA transfer. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No error (value after a “reset” command).</td></tr><tr><td>1</td><td>Errors have occurred.</td></tr></table>	RO	0
Reserved	19 (30,12)	Reserved	RAZWI	0
WE	1 (31)	Wait event. Indicates which event the channel is waiting for in the “Pended” state. Refer to Section 10.8.4 “DMA Action Register” for more information. If the channel is waiting for both events, the value of this field is set to one. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Waiting for a channel ready</td></tr></table>	RO	0

Field Name	Bits	Description	Type	Reset
		0 event. 1 Waiting for a DLM bank switching event.		

10.8.10. DMA 2D Setup Register

Mnemonic Name: dmar9 (DMA_2DSET)

IM Requirement: DMA 2-D Element Transfer feature optional (MSC_CFG.LMDMA == 1)

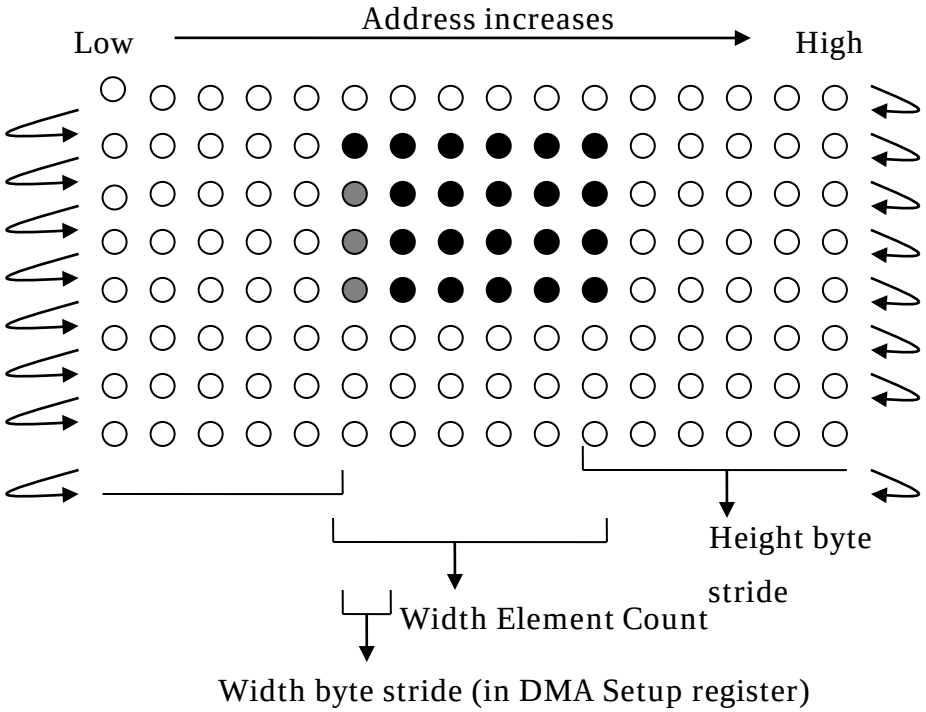
Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 9, 0}

Per DMA Channel: Yes

31 16	15 0
HTSTR	WECNT

The DMA 2D Setup Register specifies two values which, along with the stride value defined in the DMA Setup Register, define a 2-dimensional region in the external memory that needs to be transferred into a linear array in the local memory. The 2-dimensional region in the external memory can be visualized using the following figure.



The Height Byte Stride (HTSTR) shown in the above picture represents the byte value between the starting address of the rightmost transfer element of the current line of the 2-D region to the starting address of the leftmost transfer element of the next line of the 2-D region regardless of the element transfer size.

If the 2-D Element Transfer function is enabled, the DMA engine will

1. Use the External Starting address to transfer the first data element, (at this point, only one element has transferred and the transfer address still points to the External starting address).
2. Use the “Width stride” defined in the DMA Setup Register as address increment to transfer “Startup Width element count” (defined in DMA 2D Startup Control Register) minus one numbers of data elements from external memory (at this point, all the elements of the first line of the 2-D region has been transferred, and the transfer address points at the starting address of the rightmost element of the first line), and then.
3. Use the “Height stride” defined in this register as address increment to transfer the leftmost element of the next line (shown as a gray dot in the above 2-D transfer picture), when this is done the transfer address points at the starting address of the leftmost element, and the
4. Use the “Width stride” defined in the DMA Setup Register as address increment to transfer “Width element count” (defined in this register) minus one numbers of data elements from external memory.
5. Repeat 3 and 4 until the “Transfer element count” numbers of data element have been transferred.

The gray dots shown in the figure are the data elements whose transfer addresses are calculated by adding the starting address of the rightmost transfer element of the previous line and the “Height stride” (HTSTR) in step 3.

When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

Field Name	Bits	Description	Type	Reset
WECNT	16 (15,0)	The Width Element Count for a 2-D transfer region. A value of 1 degenerates the 2D transfer feature into a linear stride transfer and is not recommended for its use. A value of 0 is illegal and will generate a DMA setup error when a DMA transfer starts.	RW	DC
HTSTR	16 (31,16)	The Height Stride for a 2-D transfer region. This is the byte value increment on the external memory address between successive "Width Element Count" numbers of DMA transfers. This stride value has to be aligned to the transfer element size if the optional Unaligned External Address feature is not implemented; otherwise an Un-aligned error will be reported in the DMA Status Register when the DMA transfer starts. If the optional Unaligned External Address feature is implemented, the error reporting condition will depend on if the feature is enabled, not reporting error, or not, reporting error. Implementation Constraints: Due to implementation complexity, an implementation may add the following constraints to the legal value of this field. ■ For byte transfer, HTSTR cannot be 0. ■ For halfwod transfer, HTSTR cannot be 0 or 1. ■ For word transfer, HTSTR cannot be 0, 1, 2, or 3.	RW	DC

10.8.11. DMA 2D Startup Control Register

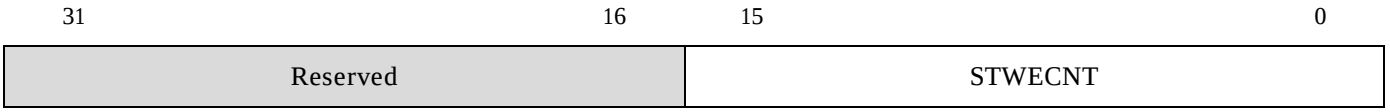
Mnemonic Name: dmar10 (DMA_2DSCTL)

IM Requirement: DMA 2-D Element Transfer feature optional (MSC_CFG.LMDMA == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 9, 1}

Per DMA Channel: Yes



The 2D Startup Control Register specifies the first width element count used in a DMA 2D block transfer operation. This register is a read/write register, and a write-shadow of the WECNT field of the 2D Setup Register.

When the DMA channel is in the RUN state, the content of this register allows to be changed by the DMA channel. When the DMA channel is stopped, by a STOP command or an error, this register will contain the “starting width element count” that is required to restart the transaction. When the DMA channel completes its transfer, the STWECNT field of this register will contain a value that is implementation-dependent.

When the DMA channel is in the RUN or the PENDED state, writing to this register will cause a DMA Transfer Disruption error.

Field Name	Bits	Description	Type	Reset
STWECNT	16 (15,0)	The Startup (first) Width Element Count for a 2-D transfer region. This is a write shadow of the WECNT field of the 2D Setup Register. So a write to the WECNT field of the 2D Setup Register will also update this field. But a write to this field will not change the value in the WECNT field of the 2D Setup Register. A value of 0 is illegal and will	RW/WS	DC

Field Name	Bits	Description	Type	Reset
		generate a DMA setup error when a DMA transfer starts.		
Reserved	16 (31,16)	Reserved	RAZWI	0

10.8.12. DMA Refill Element Count Register

Mnemonic Name: dmar11 (DMA_RCNT)

IM Requirement: DMA optional and DMA version 2.0 or above

(MSC_CFG.LMDMA == 1) & (DMA_CFG.VER >= 0x0200)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 7, 1}

Per DMA Channel: Yes

31 ... 23	22 ... 0
Reserved	DMA_RCNT

The DMA Refill Element Count Register refills the DMA Transfer Count Register (DMA_TCNT) with its content when all of the following conditions are true.

- The channel is activated by a "Start" command.
- The DMA_TCNT register is equal to zero.

When the DMA channel is in the RUN or PENDED state, writing to this register will not cause a DMA Transfer Disruption error.

Implementation Note: If the DMA is in the fast-start mode, writing any value into the trigger register will induce the DMA to issue a "Start" command to the channel. In this case, the DMA Refill Element Count Register will refill the DMA Transfer Count Register if the current value of DMA_TCNT register is zero. Please refer to Section 8.6 "Action-command Mode and Fast-start Mode" for more information.

Field Name	Bits	Description	Type	Reset
DMA_RCNT	23 (22,0)	The DMA Refill Transfer Element Count that will be copied to the DMA_TCNT register when ■ a DMA channel transfer is started by a "Start" command and ■ DMA_TCNT == 0	RW	DC
Reserved	9 (31,23)	Reserved	RAZWI	0

10.8.13. DMA Head Channel Control and Status Register

Mnemonic Name: dmar12 (DMA_HSTATUS)

IM Requirement: DMA optional and DMA version 2.0 or above

(MSC_CFG.LMDMA == 1) & (DMA_CFG.VER >= 0x0200)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {5, 8, 1}

Per DMA Channel: No

31	30 10	9	8	7	6	5	4	3	2 0
WE	Reserved	ESUP	EBUS	IOOR	IUNA	EUNA	DERR	STUNA	STAT

The DMA Head Channel Control and Status Register is a writable register with the same format as the DMA_STATUS register (in Section 10.8.9 "DMA Status Register"). This register is used to implement the dequeue operation for the DMA Channel Queue. Please refer to Section 8.5 "DMA Channel Queue" for more information.

This register has two functions:

- Reading it will return the current status (state and error) of the head channel selected by the DMA_GCSW.HCHAN field.
- Writing any value into it will dequeue the head channel from the DMA Channel Queue. More specifically, it will perform the following operations:
  - Issue a "Reset" action command to the head channel.
  - Change the DMA_GCSW.HCHAN field to point to the next channel.

Note that, to dequeue the head channel, the status of the head channel must be in the "Complete" state before any value can be written into DMA_HSTATUS. Otherwise, if the head channel is not in the "Complete" state, writing to this register will generate a DMA Transfer Disruption Error. The detailed states and conditions generating DMA Transfer Disruption Errors are shown in Table 34.

Table 34. The state after the DMA_HSTATUS register is written

Input		Output	
Current State	Next State	DMA Disruption Error	DMA_GCSW. HCHAN
Idle	Idle	-	The next channel
Pended	Stop/Error	Yes	
Run	Stop/Error	Yes	
Stop/Error	Idle	-	
Complete	Idle	-	

Although writing any value into this register will dequeue the DMA Channel Queue, we recommend you to use the register ($r0) as the source register. Any of the following commands will dequeue the DMA Channel Queue regardless of the value in the register.

- mtsr $r0, $DMA_HSTATUS
- mtusr $r0, $DMA_HSTATUS

Implementation Note: A Data Serialization Barrier (DSB) instruction is required between a write instruction (MTSR) to the DMA_HSTATUS register and a following read instruction (MFSR). Refer to Section 8.7 "Essential Data Serialization Barrier Instructions" for more information.

10.9. Resource Access Control Registers

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
racr0	PRUSR_ACC_CTL	4	4	0	331
fucpr	FUCOP_CTL	4	5	0	333

PRUSR_ACC_CTL register exists when either MSC_CFG.PFM is 1 or MSC_CFG.LMDMA is 1.

10.9.1. Privileged Resource User Access Control Register

Mnemonic Name: racr0 (PRUSR_ACC_CTL)

IM Requirement: DMA or performance monitoring optional

((MSC_CFG.LMDMA == 1) | (MSC_CFG.PFM == 1))

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {4, 4, 0}

31 ... 2	1	0
Reserved	PFM_USR_EN	DMA_USR_EN

This register controls whether a user mode program is allowed to directly access a group of related system registers (or user special registers) and the corresponding functionality for a particular privileged feature.

This register exists when either MSC_CFG.PFM is 1 or MSC_CFG.LMDMA is 1.

Field Name	Bits	Description	Type	Reset
DMA_USR_EN	1 (0)	The DMA registers user mode access enable control. Value: 0 — User mode access of DMA registers and feature is not allowed. Value: 1 — User mode access of DMA registers and feature is allowed.	RW	0
PFM_USR_EN	1 (1)	The Performance Monitoring registers user mode access enable control. Value — Meaning	RW	0

Field Name	Bits	Description	Type	Reset
		0: User mode access of Performance Monitoring registers and feature is not allowed. 1: User mode access of Performance Monitoring registers and feature is allowed.		
Reserved	30 (31,2)	Reserved	RAZWI	0

10.9.2. FPU and Coprocessor Enable Control Register

Mnemonic Name: fucpr (FUCOP_CTL)

IM Requirement: COP/FPU or Audio extension ((CPU_VER[3] == 1) | (MSC_CFG[8:7] != 0))

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 5, 0}

This system register controls enabling/disabling of audio extension, coprocessors, and the floating-point unit. Note that the floating-point unit enable is "CP0EN" when the floating-point unit is supported (i.e., CR6.CP0EX is 1 and CR6.CP0ISFPU is 1).

For power saving purpose, the coprocessor can be enabled by the exception handler only upon occurrences of the "Coprocessor Disabled Exception." This exception detects the existence of coprocessor instructions (e.g., floating-point instructions) so the handler only enables the coprocessor as needed to save power.

31	30 ... 4	3	2	1	0
AUEN	Reserved	CP3EN	CP2EN	CP1EN	CP0EN

Field Name	Bits	Description	Type	Reset
CP0EN	1 (0)	Coprocessor #0 enable bit. 0: Coprocessor #0 is disabled. Any encountering of Coprocessor #0 instruction when Coprocessor #0 exists will cause a Coprocessor Disabled Exception. 1: Coprocessor #0 is enabled.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CP1EN</td><td>1
(1)</td><td>Coprocessor #1 enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #1 is disabled. Any encountering of coprocessor #1 instruction when coprocessor #1 exists will cause a Coprocessor Disabled Exception.</td></tr><tr><td>1</td><td>Coprocessor #1 is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>CP2EN</td><td>1
(2)</td><td>Coprocessor #2 enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #2 is disabled. Any encountering of coprocessor #2 instruction when coprocessor #2 exists will cause a Coprocessor Disabled Exception.</td></tr><tr><td>1</td><td>Coprocessor #2 is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>CP3EN</td><td>1
(3)</td><td>Coprocessor #3 enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #3 is disabled. Any encountering of coprocessor #3 instruction when coprocessor #3 exists will cause a Coprocessor Disabled Exception.</td></tr><tr><td>1</td><td>Coprocessor #3 is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>27
(30,4)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>AUEN</td><td>1
(31)</td><td>Audio extension enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Audio extension is disabled. Any encountering of Audio extension instruction when Audio extension exists will cause “Audio extension disabled” exception.</td></tr><tr><td>1</td><td>Audio extension is enabled.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

10.10. Implementation-Dependent Registers

Implementation-dependent registers are specified in the following system register index range. These registers may or may not exist in a particular CPU so users must refer to the datasheet to confirm their existence. Similarly, the exact contents of these registers are also CPU-dependent and are specified in the corresponding datasheet.

Major	Minor	Extension
2	15	0-7

Brief Summary

Simple Mnemonics	Symbolic Mnemonics	Major	Minor	Extension	Page
idr0	SDZ_CTL	2	15	0	337
idr1	MISC_CTL	2	15	1	342
idr2	ECC_ MISC	2	15	2	346

10.10.1. Structure Downsizing Control Register

Mnemonic Name: idr0 (SDZ_CTL)

IM Requirement: Specified by datasheet

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 15, 0}

31 12	11 9	8 6	5 3	2 0
Reserved	BTBDZ	MTBDZ	DCDZ	ICDZ

This control register allows customers to emulate smaller cache structures on Andes hardcore. This control feature is intended to help customers to figure out the best cache configurations for their final products. Writing an un-assigned value into one of the defined fields of this register will result in UNPREDICTABLE structure size which is implementation-dependent.

Note that not all control fields are implemented so please refer to the datasheet for the actual availability. If a control field is not implemented, it becomes “Reserved” (or “Read As Zero Write Ignored”).

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ICDZ</td><td>3
(2,0)</td><td>Instruction Cache downsizing control.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Size = original size, consisting of:
■ full sets
■ full ways</td></tr><tr><td>1</td><td>Size = (original size)/2, consisting of:
■ full sets
■ (full ways)/2</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>2</td><td>Size = (original size)/2, consisting of:
■ (full sets)/2
■ full ways</td></tr>
<tr><td>3</td><td>Size = (original size)/4, consisting of:
■ (full sets)/2
■ (full ways)/2</td></tr>
<tr><td>4-7</td><td>Unassigned/UNPREDICTABLE</td></tr>
</table>
As an example, this control field works on the following configurations when both Icache and Dcache are configured to 4-way:
<table>
<tr><th>Maximum size</th><th>Supported non-default values</th></tr>
<tr><td>16KB</td><td>1</td></tr>
<tr><td>32KB</td><td>1, 2, 3</td></tr>
<tr><td>64KB</td><td>1, 2, 3</td></tr>
</table>
The limited 16KB downsizing support is due to the minimum size-per-way constraint of 4KB.
</td><td></td><td></td></tr>
<tr><td>DCDZ</td><td>3
(5,3)</td><td>Data Cache downsizing control.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Size = original size, consisting of:
■ full sets
■ full ways</td></tr>
</table>
</td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>1</td><td>Size = (original size)/2, consisting of:
■ full sets
■ (full ways)/2</td></tr>
<tr><td>2</td><td>Size = (original size)/2, consisting of:
■ (full sets)/2
■ full ways</td></tr>
<tr><td>3</td><td>Size = (original size)/4, consisting of:
■ (full sets)/2
■ (full ways)/2</td></tr>
<tr><td>4-7</td><td>Unassigned/UNPREDICTABLE</td></tr>
</table>
As an example, this control field works on the following configurations when both Icache and Dcache are configured to 4-way:
<table>
<tr><th>Maximum size</th><th>Supported non-default values</th></tr>
<tr><td>16KB</td><td>1</td></tr>
<tr><td>32KB</td><td>1, 2, 3</td></tr>
<tr><td>64KB</td><td>1, 2, 3</td></tr>
</table>
The limited 16KB downsizing support is due to the minimum size-per-way constraint of 4KB.
</td><td></td><td></td></tr>
<tr><td>MTBDZ</td><td>3
(8,6)</td><td>MTLB downsizing control.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
</table>
</td><td>RW</td><td>0</td></tr>
</table>

Field Name	Bits	Description	Type	Reset
		0: Size = original size, consisting of: ■ full sets ■ full ways 1: Size = (original size)/2, consisting of: ■ full sets ■ (full ways)/2 2: Size = (original size)/2, consisting of: ■ (full sets)/2 ■ full ways 3: Size = (original size)/4, consisting of: ■ (full sets)/2 ■ (full ways)/2 4-7: Unassigned/UNPREDICTABLE This control field works on the maximum size of 128, 64, and 32 entries. When changing the value of this field, it is required that (1) the change can only be done when PSW.IT == 0 and PSW.DT == 0 and (2) after the change all MTLB entries need to be invalidated before turning-on instruction/data address translation. If these rules are not followed, UNPREDICTABLE memory address may be generated for instruction fetch and data access.		

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>BTBDZ</td><td>3
(11,9)</td><td>Branch Target Table downsizing control.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Size = original size, consisting of:
■ full sets
■ full ways</td></tr><tr><td>1</td><td>Size = (original size)/2, consisting of:
■ full sets
■ (full ways)/2</td></tr><tr><td>2</td><td>Size = (original size)/2, consisting of:
■ (full sets)/2
■ full ways</td></tr><tr><td>3</td><td>Size = (original size)/4, consisting of:
■ (full sets)/2
■ (full ways)/2</td></tr><tr><td>4-7</td><td>Unassigned/UNPREDICTABLE</td></tr></table>This control field works on the maximum size of 256, 128, 64, and 32 entries.</td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>20
(31,12)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

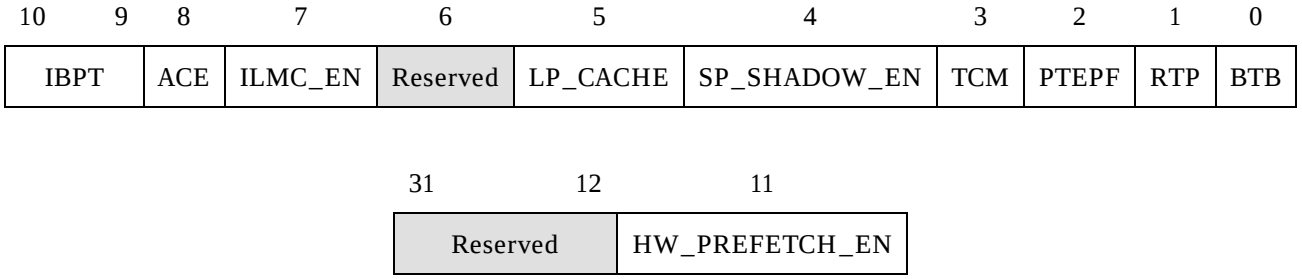
10.10.2. Miscellaneous Control Register

Mnemonic Name: idr1 (MISC_CTL)

IM Requirement: Specified by datasheet

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 15, 1}



This register contains the disabling control and enabling control for various features. Please note that each control bit may have different polarity so special attention is needed to use them.

Note that not all control fields are implemented so please refer to the datasheet for the actual availability. If a control field is not implemented, it becomes “Reserved” (or “Read As Zero Write Ignored”).

Field Name	Bits	Description	Type	Reset
BTB	1 (0)	Disable Branch Target Buffer. Value: 0 — Branch Target Buffer functionality is enabled. Value: 1 — Branch Target Buffer functionality is disabled. All branches will be predicted not taken in the early N12 pipeline stage.	RW	1

Field Name	Bits	Description	Type	Reset
RTP	1 (1)	Disable Return Target Predictor. 0: Return Target Predictor functionality is enabled. 1: Return Target Predictor functionality is disabled.	RW	1
PTEPF	1 (2)	Disable Hardware Page Table Walker L2 PTE prefetch function. 0: HPTWK L2 PTE prefetch functionality is enabled. 1: HPTWK L2 PTE prefetch functionality is disabled.	RW	0
TCM	1 (3)	Disable Two Cycle Multiplication. 0: Two cycle multiplication functionality is enabled. 1: Two cycle multiplication functionality is disabled.	RW	0
SP_SHADOW_EN	1 (4)	Enable control for shadow stack pointers. 0: Shadow stack pointers are disabled (for backward compatibility). 1: Shadow stack pointers are enabled.	RW	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>LP_CACHE</td><td>1
(5)</td><td>Disable the Loop Cache. This field is RAZWI if the Loop Cache is not configured.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The Loop Cache is enabled.</td></tr><tr><td>1</td><td>The Loop Cache is disabled.</td></tr></table></td><td>RW/
RAZWI</td><td>0</td></tr>
<tr><td>Reserved</td><td>1
(6)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>ILMC_EN</td><td>1
(7)</td><td>Enable Instruction Local Memory Cache. This field is RAZWI if EILM cache interface is not configured.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Instruction Local Memory Cache is disabled. The output port eilm_cache_en is driven LOW.</td></tr><tr><td>1</td><td>Instruction Local Memory Cache is enabled. The output port eilm_cache_en is driven HIGH.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>ACE</td><td>1
(8)</td><td>Disable Andes Custom Extension (ACE).<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Andes Custom Extension (ACE) is enabled.</td></tr><tr><td>1</td><td>Andes Custom Extension (ACE) is disabled.</td></tr></table></td><td>RW</td><td>1</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>IBPT</td><td>2
(10,9)</td><td>ULM external port access I-bank protection.
This field controls the accessibility of ULM external port to ULM I-banks. When ULM is not configured, these bits are hardwired to 2'b0.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No access protection.</td></tr>
<tr><td>1</td><td>Enable access protection for I-bank0.</td></tr>
<tr><td>2</td><td>Enable access protection for I-bank1.</td></tr>
<tr><td>3</td><td>Enable access protection for both I-bank0 and I-bank1.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>HW_PREFETCH_EN</td><td>1
(11)</td><td>Hardware prefetch enable control bit.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Hardware prefetch is disabled.</td></tr>
<tr><td>1</td><td>Hardware prefetch is enabled.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>20
(31,12)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
</table>

10.10.3. Parity/ECC Misc. Register

Mnemonic Name: idr2 (ECC_ MISC)

IM Requirement: ECC optional (MSC_CFG2.ECC == 1)

Access Mode: Superuser

SR Encoding {Major, Minor, Extension}: {2, 15, 2}

31	30 27	26 13	12 6	5 2	1	0
EI_TYP	EI_EN	Reserved	Code	RAMID	P	C

Field Name	Bits	Description	Type	Reset
C	1 (0)	Correctable error. This bit is updated when a parity/ECC error exception occurs. Value: 0 = Uncorrectable; 1 = Correctable	RO	0
P	1 (1)	Precise error. This bit is updated when a parity/ECC error exception occurs. Value: 0 = Imprecise error; 1 = Precise error	RO	0
RAMID	4 (5, 2)	The ID of RAM that has parity/ECC errors. This bit is updated when a parity/ECC error exception occurs. Value: 0–1 = Reserved; 2 = Tag RAM of the instruction cache; 3 = Data RAM of the instruction cache	RO	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>4</td><td>Tag RAM of the data cache</td></tr>
<tr><td>5</td><td>Data RAM of the data cache</td></tr>
<tr><td>6</td><td>Tag RAM of the TLB</td></tr>
<tr><td>7</td><td>Data RAM of the TLB</td></tr>
<tr><td>8</td><td>ILM</td></tr>
<tr><td>9</td><td>DLM</td></tr>
<tr><td>10–15</td><td>Reserved</td></tr>
</table>
</td><td></td><td></td></tr>
<tr><td>CODE</td><td>7
(12, 6)</td><td>Raw parity/ECC code. This field is updated when executing a CCTL instruction (L1I_IX_RTAG, L1I_IX_RWD, L1D_IX_RTAG, L1D_IX_RWD) or a TLBOP instruction (RWD). After the instruction is executed, the parity/ECC code is recorded at this field without correction.
<table>
<tr><td rowspan="2">ECC_MISC Bits</td><td colspan="2">Meaning</td></tr>
<tr><td>Parity Bits</td><td>ECC Bits</td></tr>
<tr><td>6</td><td>0</td><td>0</td></tr>
<tr><td>7</td><td>1</td><td>1</td></tr>
<tr><td>8</td><td>2</td><td>2</td></tr>
<tr><td>9</td><td>3</td><td>3</td></tr>
<tr><td>10</td><td>-</td><td>4</td></tr>
<tr><td>11</td><td>-</td><td>5</td></tr>
<tr><td>12</td><td>-</td><td>6</td></tr>
</table>
</td><td>RO</td><td>0</td></tr>
<tr><td>Reserved</td><td>14
(26, 13)</td><td>Reserved</td><td>RAZWI</td><td>0</td></tr>
<tr><td>EI_EN</td><td>4
(30, 27)</td><td>Enable control for the error injection. For the parity scheme, a byte is the basic unit for each injection; for the ECC scheme, a word is the basic</td><td>RW</td><td>0</td></tr>
</table>

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td></td><td></td><td>unit for each injection.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>None</td></tr><tr><td>1</td><td>All</td></tr><tr><td>2</td><td>Tag RAM of the instruction cache</td></tr><tr><td>3</td><td>Data RAM of the instruction cache</td></tr><tr><td>4</td><td>Tag RAM of the data cache</td></tr><tr><td>5</td><td>Data RAM of the data cache</td></tr><tr><td>6</td><td>Tag RAM of the TLB</td></tr><tr><td>7</td><td>Data RAM of the TLB</td></tr><tr><td>8</td><td>ILM</td></tr><tr><td>9</td><td>DLM</td></tr><tr><td>10–15</td><td>Reserved</td></tr></table></td><td></td><td></td></tr>
<tr><td>EI_TYP</td><td>1
(31)</td><td>Error injection type. This field selects which type of errors to be injected into RAMs with an ECC scheme. However, this selection bit has no effect on RAMs with a parity scheme because a single-bit error is always injected for the parity scheme.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Single-bit error</td></tr><tr><td>1</td><td>Two-bit error</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Implementation Note: After a read operation using CCTL or TLBOP instruction, a “data serialization barrier” (DSB) instruction must be inserted in order to read out the latest value at the “code” field.

10.11. Summary of System Register Existence

The following table summarizes the configuration condition(s) that determines the existence of each system register. For any implementation-dependent condition, please see the implementation data sheets for more details.

Table 35. Summary of system register existence

Simple Mnemonics	Symbolic Mnemonics	Existence configuration condition(s)
Configuration		
cr0	CPU_VER	Always exist
cr1	ICM_CFG	
cr2	DCM_CFG	
cr3	MMU_CFG	
cr4	MSC_CFG	
cr5	CORE_ID	
cr6	FUCOP_EXIST	CPU_VER[3] == 1
cr7	MSC_CFG2	MSC_CFG.MSC_EXT >= 1
Interruption		
ir0	PSW	Always exist
ir1	IPSW	
ir3	IVB	
ir6	ITYPE	
ir8	MERR	
ir9	IPC	
ir11	OIPC	
ir14	INT_MASK	
ir15	INT_PEND	
ir18	INT_PRI	

Simple Mnemonics	Symbolic Mnemonics	Existence configuration condition(s)
ir19	INT_CTRL	
ir4	EVA	MSC_CFG.MCU == 0
ir2	P_IPSW	MSC_CFG.INTLC != 1
ir5	P_EVA	
ir7	P_ITYPE	
ir10	P_IPC	
ir12	P_P0	
ir13	P_P1	
ir16	SP_USR	MSC_CFG.SHADOW == 1
ir17	SP_PRIV	
ir20	-	Exist only in implementations with the Security Extension
ir21	-	
ir22	-	
ir23	-	
ir24	-	
ir25	-	
ir26	INT_MASK2	IVB.IVIC_VER >= 1
ir27	INT_PEND2	
ir28	INT_PRI2	
ir29	INT_TRIGGER	
Memory Management		
mr0	MMU_CTL	Always exists
mr1	L1_PPTB	(MMU_CFG.MMPS == 2) & (MMU_CFG.HPTWK == 1)
mr4	TLB_MISC	MMU_CFG.MMPS == 2
mr2	TLB_VPN	MMU_CFG.MMPS > 0

Simple Mnemonics	Symbolic Mnemonics	Existence configuration condition(s)
mr3	TLB_DATA	
mr5	VLPT_IDX	(MMU_CFG.MMPS == 2) & Implementation-dependent
mr6	ILMB	ICM_CFG.ILMB != 0
mr7	DLMB	DCM_CFG.DLMB != 0
mr8	CACHE_CTL	(ICM_CFG.ISZ != 0) | (DCM_CFG.DSZ != 0)
mr9	HSMP_SADDR	MSC_CFG.HSMP == 1
Hardware Stack Protection		
hspr0	HSP_CTL	MSC_CFG.HSP == 1
hspr1	SP_BOUND	
hspr2	SP_BOUND_PRIV	(MSC_CFG.HSP == 1) & (MSC_CFG.SHADOW != 0)
hspr3	SP_BASE	(MSC_CFG.HSP == 1) & (HSP_CTL.UDF == 1)
hspr4	SP_BASE_PRIV	(MSC_CFG.HSP == 1) & (HSP_CTL.UDF == 1) & (MSC_CFG.SHADOW != 0)
Performance Monitoring		
pfr0-pfr2	PFMCn	MSC_CFG.PFM == 1
pfr3	PFM_CTL	
pfr4	PFT_CTL	MSC_CFG.PFT == 1
Local Memory DMA		
dmar0	DMA_CFG	MSC_CFG.LMDMA == 1
dmar1	DMA_GCSW	
dmar2	DMA_CHNSEL	
dmar3	DMA_ACT	

Simple Mnemonics	Symbolic Mnemonics	Existence configuration condition(s)
dmar4	DMA_SETUP	
dmar5	DMA_ISADDR	
dmar6	DMA_ESADDR	
dmar7	DMA_TCNT	
dmar8	DMA_STATUS	
dmar9	DMA_2DSET	
dmar10	DMA_2DSCTL	
dmar11	DMA_RCNT	(MSC_CFG.LMDMA == 1) & (DMA_CFG.VER >= 0x0200)
dmar12	DMA_HSTATUS	
Resource Access Control		
racr0	PRUSR_ACC_CTL	(MSC_CFG.LMDMA == 1) | (MSC_CFG.PFM == 1)
fucpr	FUCOP_CTL	(CPU_VER[3] == 1) | (MSC_CFG[8:7] != 0)
Embedded Debug Module		
dr42	EDM_CTL	Always exists
dr0+(*n**5)	BPCn	MSC_CFG.EDM == 1 (The range of *n* is defined in EDM_CFG.BC)
dr1+(*n**5)	BPAn	
dr2+(*n**5)	BPAMn	
dr3+(*n**5)	BPVn	
dr4+(*n**5)	BPCIDn	
dr40	EDM_CFG	
dr41	EDMSW	
dr43	EDM_DTR	
dr44	BPMTC	
dr45	DIMBR	

Simple Mnemonics	Symbolic Mnemonics	Existence configuration condition(s)
dr46	TECR0	MSC_CFG.TRACE == 1
dr47	TECR1	
Implementation-dependent		
idr0	SDZ_CTL	Implementation-dependent
idr1	MISC_CTL	Implementation-dependent
idr2	ECC_ MISC	MSC_CFG2.ECC == 1