[] | *risc-v_logo.svg*

RISC-V IOPMP Architecture Specification

RISC-V IOPMP Task Group

Version 1.0.0-draft5, December, 2023: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble

> *This document is in the Development state*
>
> Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

Many…

Chapter 1. Introduction

This document describes a mechanism to improve the security of a platform. In a platform, the bus initiators on it can access the target devices, just like a RISC-V hart. The introduction of I/O agents like the DMA (Direct Memory Access Unit) to systems improves performance but exposes the system to vulnerabilities such as DMA attacks. In the RISCV eco-system, there already exists the PMP/ePMP which provides standard protection scheme for accesses from a RISCV hart to the physical address space, but there is not a likewise standard for safeguarding non-CPU initiators. Here we propose the Physical Memory Protection Unit of Input/Output Devices, IOPMP for short, to control the accesses issued from the bus initiators.

IOPMP is considered a hardware component in a bus fabric. But why is a pure-software solution not enough? For a RISC-V-based platform, a software solution mainly refers to the security monitor, a program running on the M-mode in charge of handling security-related requests. Once a requirement from another mode asks for a DMA transfer, for example, the security monitor checks if the requirement satisfies all the security rules and then decides whether the requirement is legal. Only a legal requirement will be performed. However, the check could take a long time when the requirement is not as simple as a DMA transfer. A GPU, for example, can take a piece of program to run. Generally, examining whether a program violates access rules is an NP-hard problem. Even though we only consider the average execution time, the latency is not tolerable in most cases. A hardware component that can check accesses on the fly becomes a reasonable solution. That is the subject of this document, IOPMP.

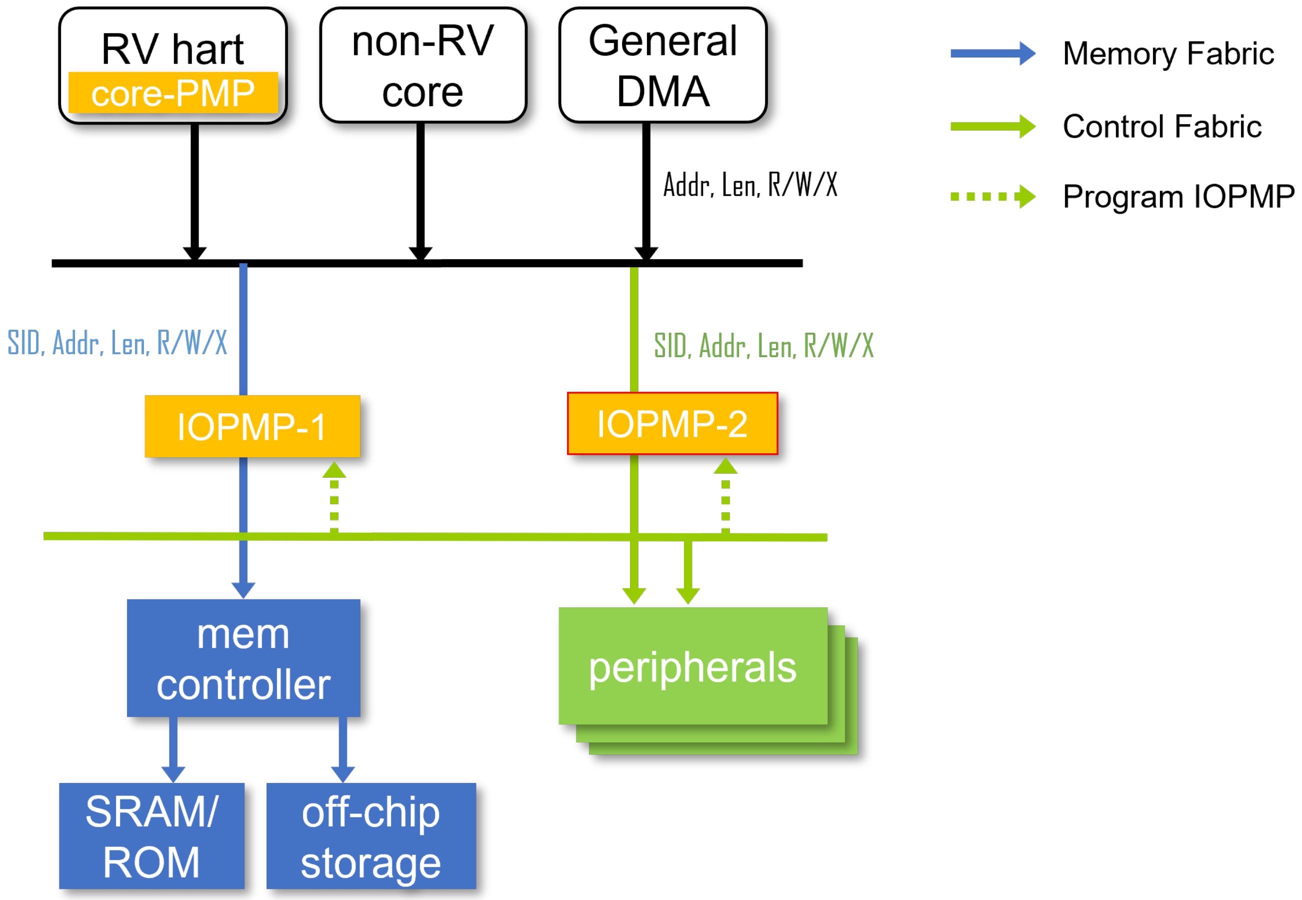

Figure 1. Examplary Integration of IOPMP(s) in System.

Chapter 2. Terminology and Concepts

This document refers to the term “secure monitor” as the software responsible for managing security-related tasks, including the programming of IOPMPs. The secure monitor is not restricted to operating on a single CPU or hart; instead, it has the flexibility to be distributed across multiple CPUs.

Glossary/ Acronyms	
Term	**Description**
DC	don’t care
IMP	implementation-dependent
MMIO	memory mapped input/output devices
NA4	naturally aligned four-byte region, one of the address matching mode used in RISC-V PMP and IOPMP
NAPOT	naturally aligned power-of-2 region, one of the address matching mode used in RISC-V PMP and IOPMP
N/A	not available
RX	receiver
TOR	top boundary of an arbitrary range, one of the address matching mode used in RISC-V PMP and IOPMP
TX	transmitter
WARL	write any read legal
W1C	write '1' clear
W1CS	write '1' clear and sticky to 0
W1S	write '1' set
W1SS	write '1' set and sticky to 1
X(*n*)	the *n*-th register in the register array X, which starts from 0.
X[*n*]	the *n*-th bit of a register X or register field X
X[*n* : *m*]	the *n*-th to *m*-th bits of a register X or register field X.

2.1. Source-ID and Transaction

Source-ID, SID for short, is a unique ID to identify a bus initiator or a group of bus initiators with the same permission. When a bus initiator wants to access a piece of memory, it issues a transaction. A transaction should be tagged with an SID to identify the issuing bus initiator. We will discuss about the exception in the next section. Tagging bus initiators with SID could be implementation-dependent. The number of bits of an SID is implementation-dependent as well. If different channels of a bus initiator could be granted different access permissions, they should have its own SID.

2.2. Source-Enforcement

If all transactions going through the IOPMP are issued by the same bus initiator or a set of bus initiators with the same permission, the Source-ID can be ignored on the bus initiator side and the above transactions. In the case, we denote the IOPMP performs source enforcement, IOPMP-SE for short.

2.3. Initiator Port, Receiver Port and Control Port

An IOPMP has at least an initiator port, at least a receiver port and one control port. A receiver port is where a transaction goes into the IOPMP, and a initiator port is where a transaction leaves it if the transaction passes all the checks. The control port is used to program the IOPMP.

2.4. Memory Domain

An SID is an abstract representation of a transaction source. It encompasses one or more transaction initiators that are granted identical permissions. On the other hand, a Memory Domain, MD for short, is an abstract representation of a transaction destination that groups a set of memory regions for a specific purpose. MDs are indexed from zero. For example, a network interface controller, NIC, may have three memory regions: an RX region, a TX region, and a region of control registers. We could group them into one MD. If a processor can fully control the NIC, it can be associated with the MD. An SID associated with a MD doesn't mean having full permissions on all memory regions of the MD. The permission of each region is defined in the corresponding IOPMP entry. However, there is an extension to adhere the permission to the MD that will be introduced in the Appendix A3.

It's important to note that, generally speaking, a single SID can be associated with multiple Memory Domains (MDs), and vice versa. However, certain models may impose restrictions on this flexibility, which will be discussed in the following chapter.

2.5. IOPMP Entry and IOPMP Entry Array

The IOPMP entry array, a fundamental structure of an IOPMP, contains IOPMP entries. Each entry, starting from an index of zero, defines how to check a transaction. An entry includes a specified memory region and the corresponding read/write permissions.

IOPMP entry encodes the memory region in the same way as the RISC-V PMP, which are OFF, NA4, NAPOT, and TOR. Please refer to the RISC-V unprivileged spec for the details of the encoding schemes.

> Since the address encoding scheme of TOR refers to the previous entry's memory region, which is not in the same memory domain, it would cause two kinds of unexpected results. If the first entry of a memory domain selects TOR, the entry refers to the previous memory domain. When the previous memory domain may not change unexpectedly, the region of this entry will be altered. To prevent the unexpected change of memory region, one should avoid adopting TOR for the first entry of a memory domain. The second issue is that the memory region of the last

entry is referred by the next memory domain. To avoid it, one can set an OFF for the last entry of a memory domain with the maximal address.

Memory domains are a partition of the entry array. Each entry is tied to exactly one memory domain, while a single memory domain could have multiple entries.

When an SID is associated with a Memory Domain, it is also inherently associated with all the entries that belong to that MD. An SID could be associated with multiple Memory Domains, and one Memory Domain could be associated with multiple SIDs.

2.6. Priority and Matching Logic

IOPMP entries exhibit partial prioritization. Entries with indices below HWCFG2.*prio_entry* are prioritized according to their index, with lower indices having higher priority. These entries are referred to as priority entries. Conversely, entries with indices greater than or equal to *prio_entry* are treated equally and assigned the lowest priority. These entries are referred to as non-priority entries. The value of *prio_entry* is implement-dependent.

The specification incorporates both priority and non-priority entries due to considerations of security, latency, and area. Priority entries, which are locked, safeguard the most sensitive data, even in the event of secure software being compromised. However, implementing a large number of these priority entries results in higher latency and increased area usage. On the other hand, non-priority entries are treated equally and can be cached in smaller numbers. This approach reduces the amortized latency, power consumption, and area when the locality is sufficiently high. Thus, the mix of entry types in the specification allows for a balance between security and performance.

The entry with the highest priority that (1) matches any byte of the incoming transaction and (2) is associated with the SID carried by the transaction determines whether the transaction is legal. If the matching entry is priority entry, the matching entry must match all bytes of a transaction, or the transaction is illegal, irrespective of its permission. If one of non-priority matching entries matches all bytes of a transaction and grants enough permission, the transaction is legal. A transaction matching no entry is illegal.

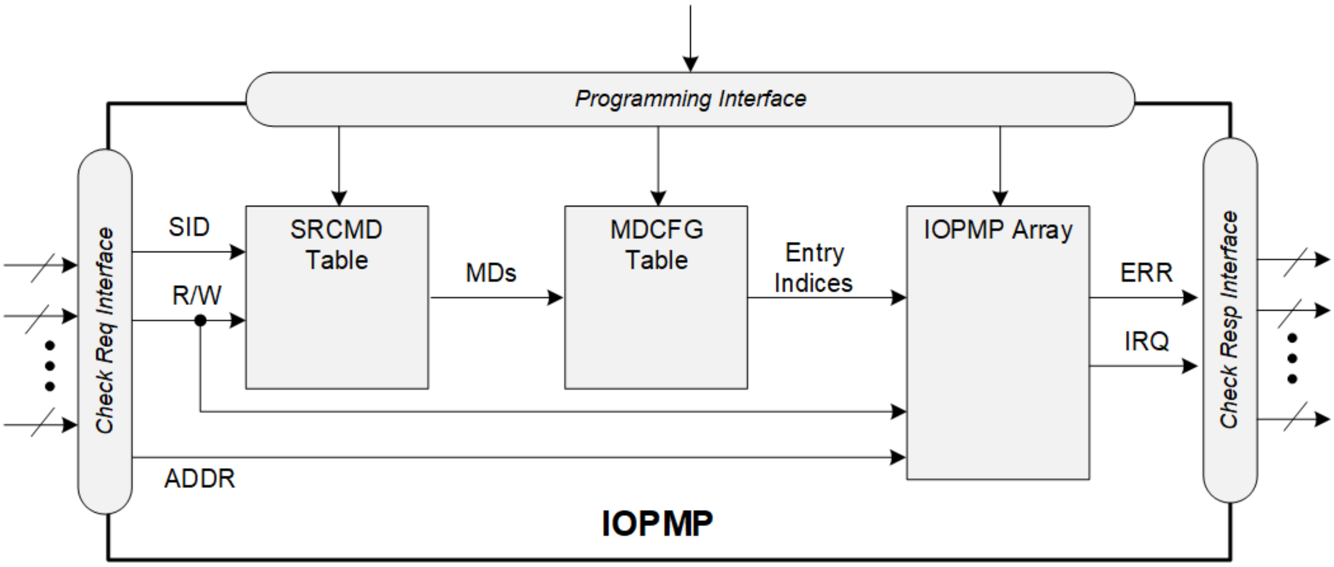

Figure 2. IOPMP Block Diagram.

2.7. Error Reactions

When an IOPMP detects an illegal transaction, it initiates three actions. Firstly, it should respond to the bus. Secondly, it could trigger an interrupt. Lastly, it could generate an error report. They are defined in the ERRREACT register. In the event of an illegal read access or instruction fetch, an IOPMP can respond in one of three ways: it can indicate a bus error, a decode error, or it can respond a success with fabricated data. Similarly, for an illegal write access, an IOPMP can respond with either a bus error, a decode error, or a success. The response options mentioned are dependent on the specific implementation and are WARL. In cases where an implementation only supports one option for the aforementioned selections, these can be hardwired.

In addition, an IOPMP has the capability to trigger an interrupt when it detects an illegal access. Specifically, if the ERRREACT.ie is enabled and ERRREACT.ire is set to 1, an interrupt is triggered for an illegal read access or instruction fetch. Similarly, if the ie is enabled and ERRREACT.iwe is set to 1, an interrupt is triggered for an illegal write access. Regardless of whether ie is set to 1, ERR_REQINFO.ip will be set to 1 for an illegal read with ire = 1 or an illegal write with iwe = 1. When ip is set, no new interrupt will be triggered and the error capture record (registers ERR_XXX) is valid until the bit is cleared.

The error capture record maintains the specifics of the first illegal access detected. This capture only occurs when ERR_REQINFO.ip is set to 0. If ip is set to 1, the record will not be updated, even if a new illegal access is detected. In other words, ip indicates whether the content of the capture record is valid and should be intentionally cleared in order to capture subsequent illegal accesses. All fields in the error capture record are optional. If a field is not implemented, it should be wired to zero.

2.8. Prefetch Violation

Prefetching is a common technique used to minimize the latency of reading data or instructions. It does this by predicting the next or subsequent addresses and preloading them. However, there's a chance that a predicted address could fall into an illegal region, which would be detected by IOPMP. Such illegal access might be seen as a false security alarm because it's the result of the prefetcher's speculation, not an actual security attack. Responding to such an event would unnecessarily burden the security software. Therefore, IOPMP can react differently to a violation caused by a prefetch access, as opposed to a typical illegal access. IOPMP could respond to the prefetcher, which issues the access, with a bus error or decode error. This would alert the prefetcher to its error and stop further speculation, at least within this speculation stream. All of this happens without the need for software intervention, allowing IOPMP to work more seamlessly with prefetchers.

The implementation-dependent flag, ERRREACT.pee, instructs the IOPMP to differentiate between a prefetch access and a normal access. This is applicable if the bus protocol of the receiver port has a corresponding signal to identify a prefetch access. When pee = 1, the ERRREACT.rpe is the response for an illegal prefetch access. In this case, no interrupt is triggered, and the ip is not updated.

Chapter 3. IOPMP Models and Configuration Protection

The spec offers several IOPMP configuration models to accommodate varied platforms. Users can choose one of the models that best fits the use cases, including those for low area, low power, low latency, high throughput, high portability, and other criteria.

3.1. The Full Model

When a full model IOPMP receives a transaction with SID *s*, IOPMP first lookups the SRCMD table to find out all the memory domains associated to source *s*. An IOPMP instance can support up to 65,535 sources, the actual number of sources can be implementation-defined and is indicated in the HWCFG0 register. Each entry in the SRCMD table defines the mapping of MDs to a specific source with SID *s*. An SRCMD entry must impelment an SRCMD_EN(*s*) register. If SPS extension described in Appendix A3 is supported, SRCMD_R(*s*) and SRCMD_W(*s*) must be implemented. If the number of MDs is over 31, SRCMD_ENH(*s*) must be implemented and SRCMD_RH(*s*) and SRCMD_WH(*s*) are for the SPS extension.

For easier description, SRCMD(*s*) is a 64-bit register representing the concatenation of SRCMD_ENH(*s*) for the higher word and SRCMD_EN(*s*) for the lower word. The field SRCMD(*s*).md is the concatenation of SRCMD_ENH(*s*).mdh and SRCMD_EN(*s*).md, and the bit SRCMD(*s*).l is the bit SRCMD_EN(*s*).l.

The field SRCMD(*s*).md is a bitmapped field and has up to 63 bits. The bit md[*j*] in SRCMD(*s*) indicates if the MD *j* is associated with the SID *s*. For unimplemented memory domains, the corresponding bits should be zero. A full model IOPMP supports up to 63 memory domains. For a system requiring more memory domains than 63, please refer to Appendix A2.

When a transaction with SID *s* arrives at an IOPMP, the IOPMP retrieves all associated MDs with SID *s* by looking up the SRCMD table. Then, by using the MDCFG table, the IOPMP can obtain all entries for a MD. The MDCFG table, viewed as a partition of the entries in the IOPMP, contains an array of registers. Each register in this array, denoted as MDCFG(*m*), corresponds to a specific memory domain *m*. The field MDCFG(*m*).t indicates the top index of the IOPMP entry belonging to the memory domain *m*. An IOPMP entry with index *j* belongs to MD *m* if MDCFG(*m*-1).t ≤ *j* < MDCFG(*m*).t, where *m* > 0. The MD 0 owns the IOPMP entries with index j < MDCFG(0).t.

After retrieving all associated IOPMP entries, a full model IOPMP checks the transaction according to these entries.

3.2. Configuration Protection

A hardware behavior that makes one or more fields or registers nonprogrammable unless resetting the IOPMP is the so-called “lock.” It can ensure critical settings are unchanged even when secure software is compromised. If a lock bit is programmable, it should be 0 after reset and sticky to 1 on write 1.

3.2.1. SRCMD Table Protection

The associations between a specific MD j and all SIDs can be prevented from further changes, which is used to enforce some SIDs associated with this MD and the reset of SIDs not. That is, it makes SRCMD(s).md[j] nonprogrammable for all s. MDLCK and MDLCKH are designed to fix the above associations. To fix MD j, one can set MDLCK.md[j] for j<32 or set MDLCKH.mdh[j-32] for j>31.

The bit MDLCK.l is a stickly to 1 and indicates if MDLCK is locked.

The MDLCK.md is optional, if not implemented, MDLCK.md should be wired to 0 and MDLCK.l should be wired to 1.

Besides, every SRCMD_EN(s) register has a bit l, which is used to lock the registers SRCMD_EN(s), SRCMD_ENH(s), SRCMD_R(s), SRCMD_RH(s), SRCMD_W(s), and SRCMD_WH(s) if any.

> Locking the SRCMD table in either way can prevent the table from being altered accidentally or maliciously. By locking the association of the MD containing the configuration regions of a component, one can prevent the component from being configured by unwanted SIDs. To make it more secure, one can use another high-priority MD containing the same regions but no permission, let it be associated with all unwanted SIDs, and then lock the two MDs' associations by MDLCK/MDLCKH. By adopting this approach, it is possible to safeguard the configuration from direct access by potentially compromised security software.

3.2.2. MDCFG Table Protection

The register MDCFGLCK is designed to partially or fully lock the MDCFG table. MDCFGLCK is consisted of two fields: MDCFGLCK.l and MDCFGLCK.f. MDCFG(j) is locked if j<MDCFGLCK.f. MDCFGLCK.f is incremental-only. Any smaller value can not be written into it. The bit MDCFGLCK.l is used to lock MDCFGLCK.

> If a MD is locked, while its preceding MD is not locked, it could lead to the potential addition or removal of unexpected entries within the locked MD. This can occur by manipulating the top index of the preceding unlocked MD. Thus, the specification asks that one MD is locked, all its preceding MDs should be locked.

3.2.3. Entry Protection

IOPMP entry protection is also related to the other IOPMP entries belonging to the same memory domain. For a MD, locked entries should be placed in the higher priority. Otherwise, when the secure monitor is compromised, one unlocked entry in higher priority can overwrite all the other locked or non-locked entries in lower priority. A register ENTRYLCK is define to indicate the number of nonprogrammable entries. The ENTRYLCK register has two fields: ENTRYLCK.l and ENTRYLCK.f. Any IOPMP entry with index $i \le$ ENTRYLCK.f is not programmable. ENTRYLCK.f is initialized to 0 and can be increased only when written. Besides, ENTRYLCK.l is the lock to ENTRYLCK.f and itself. If ENTRYLCK is hardwired, ENTRYLCK.l should be wired to 1.

Chapter 4. Other IOPMP Models

4.1. Tables Reduction

The full model comprises two tables and an array, offering substantial flexibility for configuring an IOPMP. However, this comes at the cost of increased latency and area usage. The chapter presents the other models designed to simplify these tables, thereby catering to diverse design requirements.

Regarding the IOPMP array, it serves as the primary storage for IOPMP entries and is indispensable. Its size, however, can be minimized. Memory domains can be shared among SIDs, leading to shared entries between these SIDs. This sharing mechanism may contribute to reducing the overall size of the IOPMP array. Nevertheless, if a design doesn't encompass many shared regions, simplifying the SRCMD table, as done in the isolation and compact-*k* models, could be a viable consideration.

As to the MDCFG table, it mainly plays a role of a partation of the entries in the IOPMP. Besides programming each MDCFG(*m*).t for every MD *m*, we could also consider evenly distributing entries across each MD. The rapid-*k*, dynamic-*k*, and compact-*k* models do so.

4.2. The Rapid-*k* Model

The rapid-*k* model is based on the full model, and to shorten the latency, it omits the lookup of the MDCFG table. Every memory domain has exactly *k* entries where *k* is implementation-dependent and non-programmable. The value *k* is stored in MDCFG(0).t. Implementing MDCFG(*j*) is not required when *j*>0. MDCFGLCK.f is wired to the number of MDs and MDCFGLCK.l should be 1.

4.3. The Dynamic-*k* Model

The dynamic-*k* model is based on the rapid-*k* model, except the *k* value is programmable. That is, MDCFG(0).t is WARL and accepts a limited set of values. MDCFGLCK.f is wired to the number of MDs, and MDCFGLCK.l indicates if MDCFG(0).t is still programmable or locked.

4.4. The Isolation Model

The bitmap implementation of the SRCMD table facilitates the sharing of regions between SIDs. The isolation model is designed for the case of no or a few shared regions. In this model, each SID is exactly associated with one MD. Thus, SRCMD table lookup is not needed. SID *i* is associated with MD *i* exactly. It benefits the area, the latency, and complexity. The penalty is to duplicate the same entries once some shared regions are needed. In this model, the SRCMD table and the MDLCK(H) registers are omitted.

The number of SIDs to support is bounded by the maximal number of MDs, 63.

There is no constraint imposed on the MDCFG table and the MDCFGLCK register.

4.5. The Compact-*k* Model

The compact-*k* model is the smallest model. Based on the isolation model, it requres that every MD has exactly *k* entries and *k* is not programmable. MDCFG(0).t holds the value *k*, MDCFGLCK.f is wired to the number of MDs and MDCFGLCK.l is 1.

4.6. Model Detections

To distinguish the above models, the user can read register HWCFG0.model to determine the current implemented IOPMP model.

Chapter 5. Registers

OFFSET	Register	Description
0x0000	INFO	
	VERSION	Indicates the specification and the IP vendor.
	IMPLEMENTATION	Indicates the implementation version.
	HWCFG0~2	Indicate the configurations of current IOPMP instance
	ENTRYOFFSET	Indicates the internal address offsets of each table.
	ERRREACT	Indicates the reactions for the violaions
	Programming Protection	
	MDSTALL/MDSTALLH	(Optional) Stall and resume the transaction checks when programming the IOPMP.
	SIDSCP	
	Configuration Protection	
	MDMSK/MDMSKH	Lock Register for SRCMD table.
	MDCFGLCK	Lock register for MDCFG table
	ENTRYLCK	Lock register for IOPMP entry array.
	Error Reporting	
	ERR_REQINFO	Indicate the information regarding the first captured violation
	ERR_REQID	
	ERR_REQADDR/ERR_REQADDRH	
	ERR_USER(0~7)	(Optional) User-defined violation information
0x0800	MDCFG Table, *m* =0…HWCFG0.md_num -1	
	MDCFG(*m*)	MD config register, which is to specify the indices of IOPMP entries belonging to a MD.
0x1000	SRCMD Table, *s* = 0…HWCFG1.sid_num-1	
	SRCMD_EN(*s*)/SRCMD_ENH(*s*)	The bitmapped MD enabling register of the source '*s*' that SRCMD_EN(*s*)[*j*] indicates if the source is associated with MD *j* and SRCMD_ENH(*s*)[*j*] indicates if the source is associated with MD (*j*+31).
	SRCMD_R(*s*)/SRCMD_RH(*s*)	(Optional) bitmapped MD read eanble register, '*s*' corresponding to number of sources, it indicates source *s* read permission on MDs.
	SRCMD_W(*s*)/SRCMD_WH(*s*)	(Optional) bitmapped MD write eanble register, '*s*' corresponding to number of sources, it indicates source *s* write permission on MDs.

OFFSET	Register	Description
ENTRYOFFSET	Entry Array, *i* =0…HWCFG1.entry_num-1	
	ENTRY_ADDR(*i*)	The address (region) for entry *i*
	ENTRY_ADDRH(*i*)	(Optional for 32-bit system)
	ENTRY_CFG(*i*)	The configuration of entry *i*
	ENTRY_USER_CFG(*i*)	(Optional) extension to support user customized attributes

5.1. INFO registers

The INFO registers are use to indicate the IOPMP instance configuration info.

VERSION				
0x0000				
Field	Bits	R/W	Default	Description
vendor	23:0	R	IMP	the vendor ID
specver	31:24	R	IMP	the specification version

IMPLEMENTATION				
0x0004				
Field	Bits	R/W	Default	Description
impid	31:0	R	IMP	the implementation ID

HWCFG0				
0x0008				
Field	Bits	R/W	Default	Description
model	3:0	R	IMP	Indicate the iopmp instance model • 0x0: Full model: the number of MDCFG registers is equal to HWCFG0.md_num, all MDCFG registers are readable and writable. • 0x1: Rapid-k model: a single MDCFG register to indicate the k value, read only. • 0x2: Dynamic-k model: a single MDCFG register to indicate the k value, readable and writable. • 0x3: Isolation model: the number of MDCFG registers is equal to HWCFG0.md_num, all MDCFG registers are readable and writable. • 0x4: Compact-k model: a single MDCFG register to indicate the k value, read only.

tor_en	4:4	R	IMP	Indicate if TOR is supported
sps_en	5:5	R	IMP	Indicate the secondary permission settings is supported
user_cfg_en	6:6	R	IMP	Indicate the if user customized attributes is supported
prient_prog	7:7	W1CS	IMP	A write-1-clear bit is sticky to 0 and indicates if prio_entry is programmable. Reset to 1 if the implementation supports programmable prio_entry, otherwise, wired to 0.
sid_transl_en	8:8	R	IMP	Indicate the if tagging a new SID on the initiator port is supported
sid_transl_prog	9:9	W1CS	IMP	A write-1-set bit is sticky to 0 and indicate if the field sid_transl is programmable. Support only for sid_transl_en=1, otherwise, wired to 0.
chk_x	10:10	R	IMP	Indicate if the IOPMP checks execution violations
no_x	11:11	R	IMP	For chk_x=1, the IOPMP with no_x=1 always fails execution transactions; otherwise, it should depend on the per-entry x-bit. For chk_x=0, no_x has no effect.
no_w	12:12	R	IMP	Indicate if the IOPMP always fails write transactions
stall_en	13:13	R	IMP	Indicate if the IOPMP implements stall-related features, which are MDSTALL, MDSTALLH, and SIDSCP registers.
rsv	23:14	ZERO	0	Must be zero on write, reserved for future
md_num	30:24	R	IMP	Indicate the supported number of MD in the instance
enable	31:31	W1SS	0	Indicate if the IOPMP checks transactions by default. If it is implemented, it should be initial to 0 and sticky to 1. If it is not implemented, it should be wired to 1.

HWCFG1				
0x000C				
Field	Bits	R/W	Default	Description
sid_num	15:0	R	IMP	Indicate the supported number of SID in the instance
entry_num	31:16	R	IMP	Indicate the supported number of entries in the instance

HWCFG2				
0x0010				
Field	Bits	R/W	Default	Description

prio_entry	15:0	WARL	IMP	Indicate the number of entries matched with priority. These rules should be placed in the lowest order. Within these rules, the lower order has a higher priority.
sid_transl	31:16	WARL	IMP	The SID tagged to outgoing transactions. Support only for sid_transl_en=1.

ENTRYOFFSET				
0x0014				
Field	Bits	R/W	Default	Description
offset	31:0	R	IMP	Indicate the offset address of the IOPMP array from the base of an IOPMP instance, a.k.a. the address of VERSION. Note: the offset is a signed number. That is, the IOPMP array can be placed in front of VERSION.

ERRREACT				
0x0018				
Field	Bits	R/W	Default	Description
l	0:0	W1SS	0	Lock fields to ERRREACT register.
ie	1:1	RW	0	Enable the interrupt of the IOPMP
rsv1	3:2	ZERO	0	Must be zero on write, reserved for future
ire	4:4	WARL	0	To triggle the interrupt on illegal read if ie = 1
rre	7:5	WARL	0	Response on read illegal access • 0x0: respond a bus error • 0x1: respond a decode error • 0x2: respond a success with data, all of which are zeros. • 0x3: respond a success with data, all of which are ones. • 0x4~0x7: user defined
iwe	8:8	WARL	0	To triggle the interrupt on illegal write if ie = 1
rwe	11:9	WARL	0	Response on write illegal access • 0x0: respond a bus error • 0x1: respond a decode error • 0x2: respond a success • 0x3~0x7: user defined
rsv2	27:12	ZERO	0	Must be zero on write, reserved for future

pee	28:28	WARL	0	Enable to differentiate between a prefetch access and an illegal access. If the feature is not implemented, it should be wired to 0.
rpe	31:29	WARL	0	Response on prefetch error • 0x0: respond a bus error • 0x1: respond a decode error • 0x2~0x7: user defined

An implementation can optionally support the full and partial functions defined in the fields ree, rwe, and rpe.

5.2. Programming Protection Registers

The MDSTALL(H) and SIDSCP registers are all optional and used to support atomicity issue while programming the IOPMP, as the IOPMP rule may not be updated in a single transaction.

MDSTALL				
0x0030				
Field	Bits	R/W	Default	Description
exempt	0:0	W	N/A	Stall transactions with exempt selected MDs, or Stall selected MDs.
is_stalled	0:0	R	0	Indicate if the requested stalls have occured
md	31:1	WARL	0	Writting md[*i*]=1 selects MD *i*; reading md[*i*] = 1 means MD *i* selected.

MDSTALLH				
0x0034				
Field	Bits	R/W	Default	Description
mdh	31:0	WARL	0	Writting mdh[*i*]=1 selects MD (*i*+31); reading mdh[*i*] = 1 means MD (*i*+31) selected.

SIDSCP				
0x0038				
Field	Bits	R/W	Default	Description
sid	15:0	WARL	DC	SID to select
rsv	29:16	ZERO	0	Must be zero on write, reserved for future
op	31:30	W	N/A	0: query, 1: stall transactions associated with selected SID, 2: don’t stall transactions associated with selected SID, and 3: reserved

stat	31:30	R	0	0: SIDSCP not implemented, 1: transactions associated with selected SID are stalled, 2: transactions associated with selected SID not are stalled, and 3: unimplemented or unselectable SID

5.3. Configuration Protection Registers

MDLCK and **MDLCKH** are optional registers with a bitmap field to indicate which MDs are locked in the SRCMD table.

MDLCK				
0x0040				
Field	Bits	R/W	Default	Description
l	0:0	W1SS	0	Lock bit to MDLCK and MDLCKH register.
md	31:1	WARL	0	md[*j*] is stickly to 1 and indicates if SRCMD_EN(*i*).md[*j*], SRCMD_R(*i*).md[*j*] and SRCMD_W(*i*).md[*j*] are locked for all *i*.

MDLCKH				
0x0044				
Field	Bits	R/W	Default	Description
mdh	31:0	WARL	0	mdh[*j*] is stickly to 1 and indicates if SRCMD_ENH(*i*).mdh[*j*], SRCMD_RH(*i*).mdh[*j*] and SRCMD_WH(*i*).mdh[*j*] are locked for all *i*.

MDCFGLCK is the lock register to MDCFG table.

MDCFGLCK				
0x0048				
Field	Bits	R/W	Default	Description
l	0:0	W1SS	0	Lock bit to MDCFGLCK register.
f	7:1	RW	IMP	Indicate the number of locked MDCFG entries, MDCFG entry[*f*-1:0] is locked. SW shall write a value that is no smaller than current number.
rsv	31:8	ZERO	0	

ENTRYLCK is the lock register to entry array.

ENTRYLCK				
0x004C				
Field	Bits	R/W	Default	Description

l	0:0	W1SS	0	Lock bit to ENTRYLCK register.
f	16:1	WARL	IMP	Indicate the number of locked IOPMP entries – IOPMP_ENTRY(0) ~ IOPMP_ENTRY(*f*-1) are locked. SW shall write a value that is no smaller than current number.
rsv	31:17	ZERO	0	Must be zero on write, reserved for future

5.4. Error Capture Registers

ERR_REQINFO Captures more detailed error infomation.

ERR_REQINFO				
0x0060				
Field	Bits	R/W	Default	Description
ip	0:0	R	0	Indicate if an interrupt is pending on read. for 1, the illegal capture recorder (ERR_REQID, ERR_REQADDR, ERR_REQADDRH, and fields in this register) has valid content and won't be updated even on subsequent violations.
ip	0:0	W1C	N/A	Write 1 clears the bit and the illegal recorder reactivates. Write 0 causes no effect on the bit.
ttype	2:1	R	0	Indicated the transaction type • 0x00 = reserved • 0x01 = read • 0x02 = write • 0x03 = execution
rsv1	3:3	ZERO	0	Must be zero on write, reserved for future
etype	6:4	R	0	Indicated the type of violation • 0x00 = no error • 0x01 = read error • 0x02 = write error • 0x03 = execution error • 0x04 = partial hit on a priority rule • 0x05 = not hit any rule • 0x06 = unknown SID • 0x07 = user-defined error
rsv2	30:7	ZERO	0	Must be zero on write, reserved for future

When the bus matrix doesn't have a signal to indicate an instruction fetch, the ttype and etype can never return "execution" (0x03) and "execution error" (0x03), respectively.

ERR_REQID Indicates the errored SID and entry index.

ERR_REQID				
0x0064				
Field	Bits	R/W	Default	Description
sid	15:0	R	DC	Indicate the errored SID.
eid	31:16	R	DC	Indicated the errored entry index.

ERR_REQADDR and **ERR_REQADDRH** indicate the errored request address.

ERR_REQADDR				
0x0068				
Field	Bits	R/W	Default	Description
addr	31:0	R	DC	Indicate the errored address[33:2]

ERR_REQADDRH				
0x006C				
Field	Bits	R/W	Default	Description
addrh	31:0	R	DC	Indicate the errored address[65:34]

ERR_USER(0..7) are optional registers to provide users to define their own error capture information.

ERR_USER(*i*)				
0x0080 + **0x04** * *i*, *i* = **0...**7				
Field	Bits	R/W	Default	Description
user	31:0	IMP	IMP	(Optional) user-defined registers

5.5. MDCFG Table

The MDCFG table is a lookup to specify the number of IOPMP entries that is associated with each MD. For different models:

1. Full model: the number of MDCFG registers is equal to HWCFG0.md_num, all MDCFG registers are readable and writable.
2. Rapid-*k* model: a single MDCFG register to indicate the *k* value, read only. Only MDCFG(0) is implemented.
3. Dynamic-*k* model: a single MDCFG register to indicate the *k* value, readable and writable. Only MDCFG(0) is implemented.

4. Isolation model: the number of MDCFG registers is equal to HWCFG0.md_num, all MDCFG registers are readable and writable.
5. Compact-*k* model: a single MDCFG register to indicate the *k* value, read only. Only MDCFG(0) is implemented.

MDCFG(*m*), *m* = 0…HWCFG0.md_num-1, support up to 63 MDs				
0x0800 + (*m*)*4				
Field	Bits	R/W	Default	Description
t	15:0	WARL	DC/IMP	Indicate the top range of memory domain m. An IOPMP entry with index j belongs to MD m - If MDCFG(*m*-1).t ≤ j < MDCFG(*m*).t, where m>0. The MD0 owns the IOPMP entries with index j<MDCFG(0).t. - If MDCFG(*m*-1).t >= MDCFG(*m*).t, then MD *m* is empty. - For rapid-*k*, dynamic-*k* and compact-*k* models, MDCFG(0).t indicates the number of IOPMP entries belongs to each MD, that is, the *k* value. The MDCFG(*i*) can be omitted for *i*>0.
rsv	31:16	ZERO	0	Must be zero on write, reserved for future

5.6. SRCMD Table Registers

Only the full model, the rapid-*k* model and the dynamic-*k* model implement the SRCMD table.

0x1000 + (*s*)*32				
SRCMD_EN(*s*), *s* = 0…HWCFG1.sid_num-1				
Field	Bits	R/W	Default	Description
l	0:0	W1SS	0	A sticky lock bit. When set, locks SRCMD_EN(*s*), SRCMD_ENH(*s*), SRCMD_R(*s*), SRCMD_RH(*s*), SRCMD_W(*s*), and SRCMD_WH(*s*) if any.
md	31:1	WARL	DC	md[*j*] = 1 indicates MD *j* is associated with SID *s*.

0x1004 + (*s*)*32				
SRCMD_ENH(*s*), *s* = 0…HWCFG1.sid_num-1				
Field	Bits	R/W	Default	Description
mdh	31:0	WARL	DC	mdh[*j*] = 1 indicates MD (*j*+31) is associated with SID *s*.

SRCMD_R, **SRCMD_RH**, **SRCMD_W** and **SRCMD_WH** are optional registers; When SPS extension is enabled, the IOPMP checks both the R/W and the ENTRY_CFG.r/w permission and follows a fail-first rule.

SRCMD_R(*s*), *s* = 0...HWCFG1.sid_num-1				
0x1008 + (*s*)*32				
Field	Bits	R/W	Default	Description
rsv	0:0	ZERO	0	Must be zero on write, reserved for future
md	31:1	WARL	DC	md[*j*] = 1 indicates SID *s* has read permission to the corresponding MD *j*.

SRCMD_RH(s), *s* = 0...HWCFG1.sid_num-1				
0x100C + (*s*)*32				
Field	Bits	R/W	Default	Description
mdh	31:0	WARL	DC	mdh[*j*] = 1 indicates SID *s* has read permission to MD (*j*+31).

SRCMD_W(*s*), *s* = 0...HWCFG1.sid_num-1				
0x1010 + (*s*)*32				
Field	Bits	R/W	Default	Description
rsv	0:0	ZERO	0	Must be zero on write, reserved for future
md	31:1	WARL	DC	md[*j*] = 1 indicates SID *s* has write permission to the corresponding MD *j*.

SRCMD_WH(*s*), *s* = 0...HWCFG1.sid_num-1				
0x1014 + (*s*)*32				
Field	Bits	R/W	Default	Description
mdh	31:0	WARL	DC	mdh[*j*] = 1 indicates SID *s* has write permission to MD (*j*+31).

5.7. Entry Array Registers

ENTRY_ADDR(*i*), *i* = 0...HWCFG1.entry_num-1				
ENTRYOFFSET + (*i*)*16				
Field	Bits	R/W	Default	Description
addr	31:0	WARL	DC	The physical address[33:2] of protected memory region.

ENTRY_ADDRH(*i*), *i* = 0...HWCFG1.entry_num-1				
ENTRYOFFSET + 0x4 + (*i*)*16				
Field	Bits	R/W	Default	Description

addrh	31:0	WARL	DC	The physical address[65:34] of protected memory region.

A complete 64-bit address consists of these two registers, ENTRY_ADDR and ENTRY_ADDRH. However, an IOPMP can only manage a segment of space, so an implementation would have a certain number of the most significant bits that are the same among all entries. These bits are allowed to be hardwired.

ENTRY_CFG(*i*), *i* = 0…HWCFG1.entry_num-1				
ENTRYOFFSET + 0x8 + (*i*)*16				
Field	Bits	R/W	Default	Description
r	0:0	WARL	DC	The read permission to protected memory region
w	1:1			The write permission to the protected memory region
x	2:2			The executable permission to the protected memory region. Optional field, if unimplemented, write any read the same value as r field.
a	4:3	WARL	DC	The address mode of the IOPMP entry • 0x0: OFF • 0x1: TOR • 0x2: NA4 • 0x3: NAPOT
rsv	31:5	ZERO	0	Must be zero on write, reserved for future

The bits, r, w, and x, grant the read, write, or execution permission, respectively. Not each bit should be programmable. Some or all of them could be wired. Besides, an implementation can optionally impose constraints on their combinations. For example, x and w can't be 1 simultaneously.

The **ENTRY_USER_CFG** implementation defined registers that allows the users to define their own additional IOPMP check rules beside the rules defined in **ENTRY_CFG**.

ENTRY_USER_CFG(*i*), *i* =0…HWCFG1.entry_num-1				
ENTRYOFFSET + 0xC + (*i*)*16				
Field	Bits	R/W	Default	Description
im	31:0	IMP	DC	User customized permission field

Chapter 6. Static Rules

TBD

Chapter 7. Programming IOPMPs

At times, it can be difficult or even impossible to configure all IOPMP settings when the system starts, especially before I/O agents are active or connected to the system. As a result, it is necessary to update IOPMP settings during runtime. This may occur when a device is enabled, disabled, added, removed, or when the accessible area of a device changes. When updating, it is important to avoid putting IOPMP in a transient metastable state due to incomplete settings. However, updating IOPMP settings often involves a series of control accesses, and if a transaction check occurs during the update, it can potentially create a vulnerability. It can be difficult for the security software to guarantee that no transactions are in progress from all related initiators. A false alarm could result in significant performance issues. This chapter describes an optional method for updating IOPMP's settings without intervening transaction initiators.

7.1. Atomicity Requirement

The term here "stable" refers to meeting the atomicity requirement. This implies that when updating an IOPMP, all transactions from input ports must be checked either before any changes or after completing all changes. Essentially, using partial settings in an IOPMP should be avoided. The succeeding sections will describe the mechanism to satisfy this requirement.

7.2. Programming Steps

The general approach to the atomicity requirement has three major steps, conceptually described as follows:

- Step 1: Stall related transactions. Before proceeding with any updates, delay checking the transactions that may be impacted.
- Step 2: Update IOPMP's settings.
- Step 3: Resume stalled transactions.

For step 1, it's important to verify if the necessary stalling transactions have taken place since they might not be instantaneous in certain implementations. Following this, execute the IOPMP update as step 2, and finally, resume all stalled transactions in step 3.

> In some cases, Step 1 and Step 3 may be skipped as long as no transaction check can interrupt Step 2. Updating MDs associated with a specific SID to other MDs is an example.

7.3. Stall Transactions

For Step 1, it's possible to postpone all transactions until all updates are finished. However, this could cause unrelated transactions to experience unnecessary delays. This might not be tolerable for devices that require low latency, like a display controller that periodically retrieves a frame from its video buffer. This section explains the mechanism that only stalls specific transactions to prevent the aforementioned scenario and ensure the atomicity requirement. All the features mentioned below are optional.

Since the stalls occur when updating is in progress, determining wheater a transaction's check should wait cannot be based on any IOPMP's configuration about to change. Therefore, the only information that can be relied upon for this decision is the SID carried by the transaction. To simplify the following description, we use a conceptual signal called sid_stall[*i*] to indicate whether the transaction with SID=*i* must wait. Please note that it may not be an actual signal in practice and is not accessible directly for software.

A conceptual internal signal sid_stall has the same number of bits as the SIDs in the IOPMP. sid_stall is generated by the bit MDSTALL.exempt. stall_by_md is the concatenation of MDSTALL.mdh and MDSTALL.md, that is, stall_by_md[30:0] is MDSTALL.md[31:1] and stall_by_md[62:31] is MDSTALLH.mdh[31:0] if any. When MDSTALL.exempt is zero, any non-zero value in stall_by_md[*j*] will cause transactions with SID=*i* to be stalled for all the SID *i* associated with the MD *j*. On the contrary, on MDSTALL.exempt=1, checks of all transactions must wait except those with SID=*i* associated with any MD *j* and stall_by_md[*j*] = 1. This relation can be more precisely described as follows:

```
sid_stall[i] <= MDSTALL.exempt ^ ( Reduction_OR (SRCMD(i) & stall_by_md));
```

For any unimplemented memory domain, the corresponding bit in MDSTALL.md or MDSTALLh.mdh should be wired to 0.

sid_stall should be captured only when MDSTALL.exempt is written, that is, when MDSTALL is written. When MDSTALLH is written, the only action is to hold the value.

> Although sid_stall is related to the SRCMD table, but should be captured only when MDSTALL.exempt is written. The behavior of writing MDSTALL is used to capture a momentary snapshot of the table because the table may not be stable during the updating.

7.4. Cherry Pick

If MDSTALL doesn't stall all the desired transactions, there is an optional method to pick the transaction with specific SIDs. The SIDSCP register comprises two fields: a 2-bit SIDSCP.op and a field for SIDSCP.sid. By setting SIDSCP.op=1, the sid_stall[*i*] is activated for *i*=SIDSCP.sid. Conversely, by setting SIDSCP.op=2, the sid_stall[*i*] is deactivated for *i*=SIDSCP.sid. This register can be considered as the fine-tuning sid_stall after MDSTALL. The value of SIDSCP.op=0 is to query the sid_stall indirectly, and the value of 3 is reserved.

7.5. Resume Stall

In order to resume all stalled transactions, the IOPMP can be prompted by writing 0 to MDSTALL. This corresponds to Step 3 of the "Programming Steps" section.

7.6. The Order to Stall

In Step 1 of programming IOPMP, MDSTALL can be written at most once and before any SIDSCP is written. After a resume, writing a non-zero value to MDSTALL multiple times leads to an undefined situation.

SIDSCP can be written multiple times or not at all. To determine whether all requested stalls take effect, one can read back the bit MDSTALL.is_stalled, which is in the same location as MDSTALL.exempt on a write. MDSTALL.is_stalled=1 indicates all requested stalls taking effect.

To query if all transactions associated with a specific SID are stalled, do the following. First, write 0 to SIDSCP.op and the SID you want to query to SIDSCP.sid. Then, read back SIDSCP. The readback of SIDSCP.stat = 1 means that transactions with the queried SID have stalled, that is, the corresponding bit in sid_stall is 1. If the value is 2, it means they are not stalled. A value of 3 indicates an unimplemented or unselectable SID in SIDSCP.sid. SIDSCP.stat is in the same location as SIDSCP.op on a write. SIDSCP.sid should keep the last written legal SID and SIDSCP.stat reflects the current state of this SID. This method is considered an indirect way to read sid_stall.

7.7. Implementation-Dependent

All registers described in this chapter are optional. Moreover, these features could be partially implemented. In MDSTALL.md and MDSTALLH.mdh, not every bit should be implemented even though the corresponding MD is implemented. An unimplemented bit means unselectable and should be wired to zero. To test which bits are implemented, one can write all 1's to MDSTALL.md and MDSTALLH.mdh and then read them back. An implemented bit returns 1.

If an IOPMP implementation has fewer than 32 memory domains, MDSTALLH should be wired to zero.

> An example of partial implementation of MDSTALL.md/MDSTALLH.mdh is a system with a display controller, which is a latency-sensitive device. On updating the IOPMP, the transactions initiated from the display controller should not be stalled. Thus, one can always use MDSTALL.exempt=1 and MDSTALL.md[*j*]=1, where MD *j* is the memory domain for the frame buffer that the display controller keeps accessing. Thus, the system only needs to implement MDSTALL.md[*j*].

If the whole MDSTALL is not implemented, MDSTALL and MDSTALLH should always return zero.

If SIDSCP is not implemented, it always returns zero. One can test if it is implemented by writing a zero and then reading it back. Any IOPMP implementing SIDSCP should not return a zero in SIDSCP.stat in this case.

It is unnecessary to allow every implemented SID to be selectable by SIDSCP.sid. If an unimplemented or unselectable SID is written into SIDSCP.sid, it returns SIDSCP.stat = 3.

A1: Multi-Faults Extension

TBD

Chapter 8. A2: Run Out Memory Domains

In this specification, the support is capped at 63 memory domains. However, this chapter provides pertinent recommendations for situations that necessitate a larger number of memory domains.

8.1. A2.1 Parallel IOPMP

Multiple IOPMPs can be placed in parallel. A transaction should be directed to one of these IOPMPs for its check. The chosen IOPMP then determines its legality. There are two potential methods for routing the transaction: by address or by SID. Address-based routing divides the address space into multiple disjoint sets, and a transaction is directed to the IOPMP based on its starting address. Similarly, SID-based routing divides all possible SIDs, and a transaction is directed to the IOPMP based on its SID.

> Placing IOPMPs in parallel can seamlessly enhance the support for an increased number of memory domains since all the IOPMPs are located in the same position. This arrangement may also concurrently increase the checking throughput.

8.2. A2.2 Cascading IOPMP

Cascading multiple IOPMPs allows a transaction to traverse through more than one IOPMP. Each time a transaction goes through an IOPMP, it is tagged a new SID until it reaches the final IOPMP. This new SID represents that the transaction has been checked by a specific IOPMP. Subsequent IOPMPs could deem the transaction trustworthy and forward it to their initiator port without further checks, or check it in a higher level view, e.g., a subsystem view. An IOPMP with the above feature of tagging a new SID is referred to as an IOPMP gateway. Its HWCFG0.sid_transl_en should be set to 1, and HWCFG2.sid_transl is used to store the SID. HWCFG0.sid_transl_prog indicates whether HWCFG2.sid_transl is programmable or not. To lock sid_transl, write 1 to sid_transl_prog, which clears sid_transl_prog and is sticky to 0.

> The integration of several independently developed smaller Systems on a Chip (SoCs) to construct a larger SoC reduces the chip count in a device. This approach also decreases costs by enabling the use of larger and shared memory devices. In such a system, each subsystem upholds its governance through its own secure software, SID assignment, and security configuration. The cascading approach facilitates this: the secure software manages the IOPMP in the boundary of the subsystem. The boundary IOPMP assigns a new SID to each outgoing transaction, representing that it has been checked by the IOPMP. The outer IOPMPs are tasked with controlling the transactions from a subsystem perspective by the new subsystem-level SID. That is, the IOPMP only considers the legality of the transactions initiated from a specific subsystem instead of individual transaction initiators. The boundary IOPMP hides some details of the subsystem good for protecting intellect properties. The development flow becomes more abstract, reusable, and modularized.

A3: Secondary Permission Setting

IOPMP/SPS (Secondary Permission Setting) is an extension to support different sources to share memory domain while allowing each sources to have different R/W/X permission to a single memory domain.

If the IOPMP/SPS extension is implemented, each SRCMD table entry shall additionally define read and write permission registers: SRCMD_R(*s*) and SRCMD_W(*s*), and SRCMD_RH(*s*) and SRCMD_WH(*s*) if applicable. Register SRCMD_R(*s*) and SRCMD_W(*s*) each has a single fields, SRCMD_R(*s*).md and SRCMD_W(*s*).md respectively representing the read and write permission for each memory domain for source *s*. Setting lock to SRCMD_EN(*s*).l also locks SRCMD_R(*s*), SRCMD_RH(*s*), SRCMD_W(*s*), and SRCMD_WH(*s*).

IOPMP/SPS has two sets of permission settings: one from IOPMP entry and the other from SRCMD_R/SRCMD_W. IOPMP/SPS shall check read and write permission on both the SRCMD table and entries, a transaction fail the IOPMP/SPS check if it violates either of the permission settings.

The IOPMP/SPS register for setting executable permission on each memory domain is [TBD].

Bibliography