[] | *risc-v_logo.svg*

RISC-V IOPMP Architecture Specification

RISC-V IOPMP Task Group

Version 1.0.0-draft3, September, 2023: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble . . . 1
Copyright and license information . . . 2
Contributors . . . 3
1. Introduction . . . 4
2. Terminology and Concepts . . . 5
  2.1. Source-ID and Transaction . . . 5
  2.2. Source-Enforcement . . . 5
  2.3. Initiator Port, Receiver Port and Control Port . . . 5
  2.4. Memory Domain . . . 5
  2.5. IOPMP Entry and IOPMP Entry Array . . . 6
  2.6. Priority and Matching Logic . . . 6
  2.7. Error Reactions . . . 7
  2.8. Prefetch Violation . . . 7
3. IOPMP Models . . . 9
  3.1. The Full Model . . . 9
  3.2. Configuration Protection . . . 9
    3.2.1. Protect the SRCMD Table . . . 9
    3.2.2. Protect the MDCFG Table . . . 10
  3.3. The Rapid-*k* Model . . . 10
  3.4. The Dynamic-*k* Model . . . 11
  3.5. The Isolation Model . . . 11
  3.6. The Compact-*k* Model . . . 11
  3.7. Model Detections . . . 12
4. Tables Reduction and Detection . . . 13
5. Registers . . . 14
  5.1. **INFO registers** . . . 15
  5.2. **Programming Protection Registers** . . . 17
  5.3. **Configuration Protection Registers** . . . 18
  5.4. **Error Capture Registers** . . . 19
  5.5. **SRCMD Table Registers** . . . 20
  5.6. **MDCFG Table** . . . 21
  5.7. **Entry Array Registers** . . . 21
6. Reset . . . 23
7. Programming IOPMPs . . . 24
  7.1. Atomicity Requirement . . . 24
  7.2. Programming Steps . . . 24
  7.3. Stall Transactions . . . 24
  7.4. Cherry Pick . . . 25

7.5. Resume Stall 25
7.6. The Order to Stall 25
7.7. Implementation-Dependecy 26
A1: Multi-Faulse Extension 27
A2: Run Out Memory Domains 28
A3: Sencondary Permission Setting 29
Bibliography 30

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.

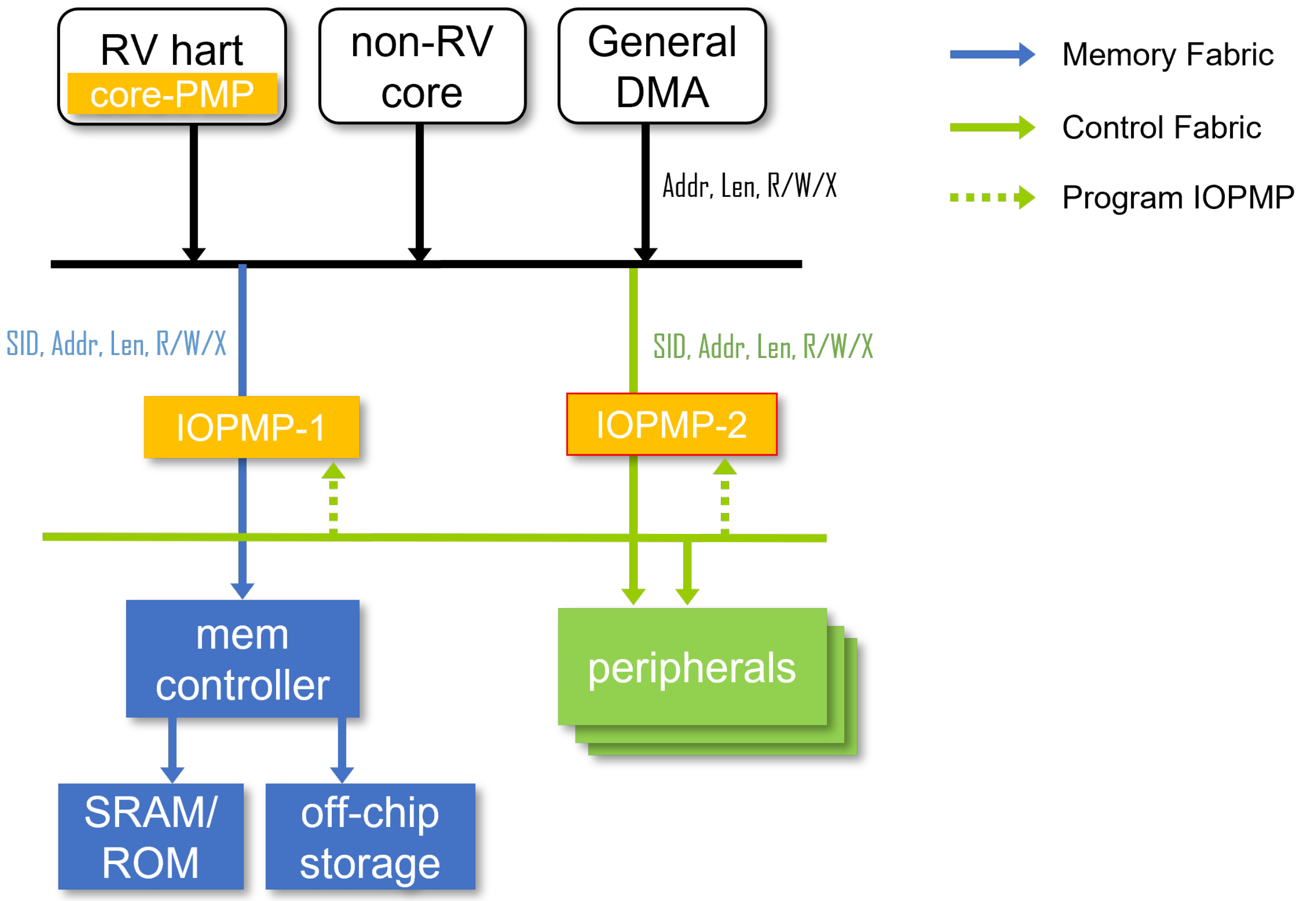

Figure 1. Examplary Integration of IOPMP(s) in System.

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.

Use X(*n*) represents the *n*-th register in the register array X, which starts from 0.

Use X[*n*] represents the *n*-th bit of a register X or register field X, and X[*n*:*m*] represent 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 a 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, is a list of 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.

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 (MD), 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.

As to an IOPMP-SE, the only structure of it is the IOPMP entry array. Due to no SID, when selecting the matching IOPMP entry, an IOPMP-SE ignores the SID comparison.

2.6. Priority and Matching Logic

IOPMP entries exhibit partial prioritization. Entries with indices below *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.

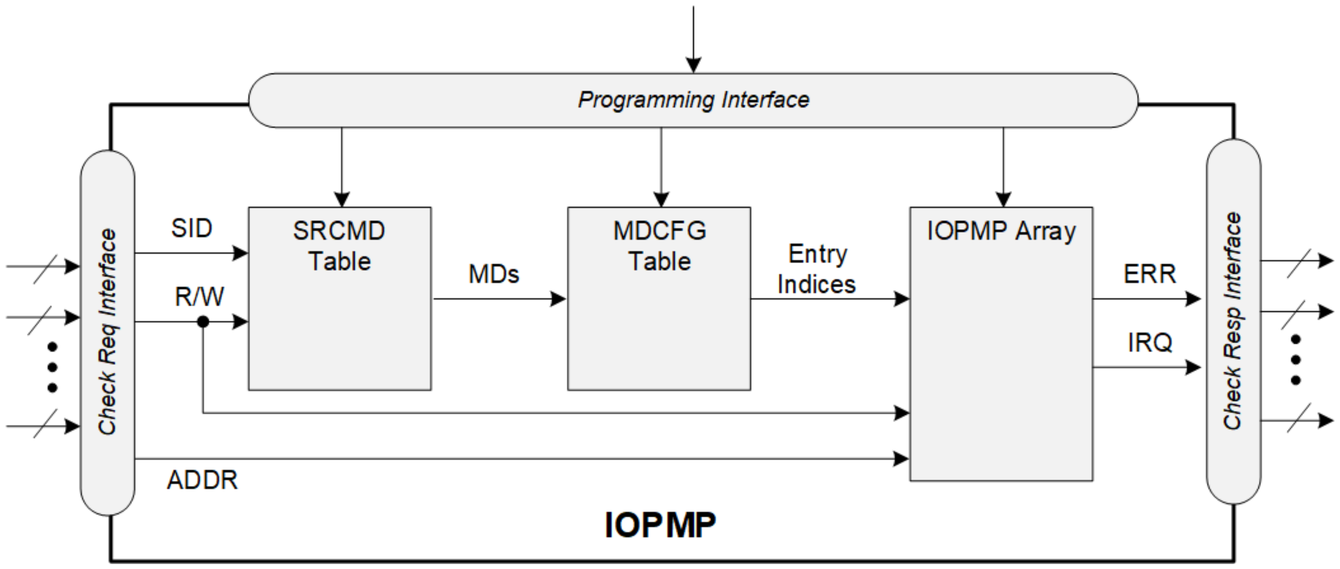

Figure 2. IOPMP Block Diagram.

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, 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. Similarly, if the ERRREACT.ie is enabled and ERRREACT.iwe is set to 1, an interrupt is triggered for an illegal write access. Regardless of whether ERRREACT.ie is set to 1, ERRREACT.ip will be set to 1 for an illegal read with ERRREACT.ire = 1 or an illegal write with ERRREACT.iwe = 1. When ERRREACT.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 bit is write-1-clean and not affected by the lock bit of ERRREACT.l.

The error capture record maintains the specifics of the most recent illegal access detected. This capture only occurs when ERRREACT.ip is set to 0. If ERRREACT.ip is set to 1, the record will not be updated, even if a new illegal access is detected. In other words, ERRREACT.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 read latency. 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 ERRREACT.pee = 1, the ERRREACT.rpe is the response for an illegal prefetch access. In this case, no interrupt is triggered, and the ERRREACT.ip is not updated.

Chapter 3. IOPMP Models

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 256 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, and is optionally to implement SRCMD_R(*s*) and SRCMD_W(*s*) if SPS extension is supported, see details in Appendix A3. Register SRCMD_EN(*s*) is occupies a 64-bit space, and has two fields, SRCMD_EN(*s*).l and SRCMD_EN(*s*).md. SRCMD_EN(*s*).l is the sticky lock to register SRCMD_EN(*s*). In the full model, SRCMD_EN(*s*).md is a bitmapped field and has up to 63 bits. Each bit in SRCMD_EN(*s*).md indicates if the corresponding MD is associated with the SID *s*. For unimplemented memory domains, the corresponding bits in SRCMD(*s*).md should be WARZ. A full model IOPMP supports up to 63 memory domains. For a system requiring more memory domains than 63, please refer to Appendix A2.

Once IOPMP retrieves all associated MDs for a transaction with SID *s* from the SRCMD table, it looks up the corresponding IOPMP entries belonging to these MDs. The MDCFG table has an array of registers where the register MDCFG(*m*) is for the memory domain *m*. One 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. Each MDCFG register occupies a 32-bit space and the field MDCFG(*m*).t occupies the lowest 16 bits. The number of implemented bits is implement-dependent.

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

3.2. Configuration Protection

A hardwire behavior that makes an IOPMP fully or partially nonprogrammable unless resetting the IOPMP is the so-called “lock.” A lock in an IOPMP is similar to that in a PMP. It can ensure critical settings are unchanged even though the security monitor is compromised.

3.2.1. Protect the SRCMD Table

In certain use cases, the user may want to ensure a specific MD is enabled with (associated to) all sources. For example, one may want to set MD 0 as a denylist region where all sources has no access to memories in this region. A register MDLCK is defined to set lock on specific MDs for all sources. MDLCK has two fields: a 63-bit MDLCK.md and a 1-bit MDLCK.l. If MDLCK.md[*m*] == 0x1, for all SIDs, SRCMD.md[*m*] is not programmable. MDLCK.l is the lock bit for the MDLCK. In the above example, when you want to enforce every SID to associate MD 0, you can first set all

SRCMD(*s*).md[0]=1, and then let MDLCK.md[0]=1. The rest mappings are still programmable. If MDLCK is not implemented, it should be WARZ.

For unimplemented memory domains, the corresponding bits of MDLCK.md should be WARZ. The bits for implemented memory domains in MDLCK.md can be also hardwired. However, in this case, the corresponding bits in SRCMD(*s*).md should be well initialized during reset process. If whole MDLCK.md is hardwired, MDLCK.l should be wired to 1. The user can detect the number of implemented memory domain through HWCFG0 register.

Besides, every SRCMD register has an optional bit, L, which is used to lock this register. It is a convenient way to lock the MD mapping of an SID without consuming any IOPMP entry. If a programmable SRCMD(*s*).l is implemented, SRCMD(*s*).l should be initialized to zero after reset. If SRCMD(*s*).l is not implemented, it can be hardwired to 0 or 1. If it is wired to 1, SRCMD(*s*).md should be hardwired properly.

3.2.2. Protect the MDCFG Table

Subsequently, the MDCFG table can also be locked. The register MDCFGLCK is designed for the purpose, which has two fields: MDCFGLCK.l and MDCFGLCK.f. Please note that if the top index of MD *m* is locked, that of DM *m-1* should be locked, too. Otherwise, the IOPMP entries of MD m can be added or removed by modifying MDCFG(*m-1*).t. By introduction, if MD m is locked, MD n should also be locked, where $n < m$. MDCFG(*m*) are nonprogrammable if *m* < MDCFGLCK.f. MDCFGLCK.f is initialized to 0 after reset, and can be increased only when written. MDCFGLCK.l is the lock of MDCFGLCK. If MDCFGLCK is hardwired, MDCFGLCK.l should be wired to 1.

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 security monitor is compromised, one unlocked entry in higher priority can overwrite all the other 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* ≤ 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.

3.3. The Rapid-*k* Model

To shorten the latency, the rapid-*k* model replaces the lookup of the MDCFG table by simple logics. Every memory domain has exactly *k* IOPMP entries where *k* is implementation-dependent and hardwired. Since *k* is a fixed number, once MDs are retrieved for a transaction, these indexes of selected MDs can quickly transform into the signals to pick up the IOPMP entries. An extreme case is *k*=1 in which every non-zero bit in SRCMD(*s*).md directly maps to a selected IOPMP entry for SID=*s*.

To make it semantically compatible with the full model, the realated fields should be read with their original meanings. MDCFGLCK.f should be the same as the number of implemented MDs and MDCFGLCK.l should be 1. MDCFG(*m*).t should be (*m*)+1)**k*. That is, one can read MDCFG(0).t to retrieve the value *k*.

MDCFG(*m*).f and MDCFG(*m*).l can still be programmable or hardwired. The two fields typically do not affect the latency of checking a transaction. They are usually related to writing to IOPMP registers, and writing latency is not a concern in this model.

3.4. The Dynamic-*k* Model

The dynamic-*k* model is similar to the rapid-*k* model, except the *k* value is programmable. If you have a fixed number of IOPMP entries, you probably don't need this model. You can simply divide all IOPMP entries evenly to every memory domain and obtain a fixed k. However, if the IOPMP array is not in dedicated storage and could be shared for other purposes, the dynamic-*k* model helps partition these IOPMP entries.

The IOPMP entry reassignment is not suggested during the run time. The boot time is a better choice.

MDCFG(0).t stores the value *k* and is WARL. That is, an implementation may accept limited *k*. However, zero should not be a legal value. One should make sure if a written value is legal by reading it back. The *k* is usually considered as a power of 2 for easier hardware implementation. MDCFG(*m*).t is read-only and equals to (*m*+1)**k* when it is read. By updating MDCFG(0).t and then examining MDCFG(1).t's change, one can know the IOPMP is the dynamic-*k* model.

MDCFGLCK.f should be zero right after the IOPMP resets. MDCFGLCK.f and MDCFGLCK.l can be programmable or hardwired. If MDCFGLCK.f is programmable, it can only accept two values: 0 and the number of MDs.

3.5. The Isolation Model

One of the benefits of the full model is to share common memory regions (by memory domains) among different SIDs. The isolation model can be implemented for the case of no or a few shared regions. In this model, each SID is exactly associated with one MD. Thus, no table-lookup is needed to retrieve the associated MD. It benefits the area as well as the latency. The penalty is to duplicate IOPMP entries when two SIDs do share regions. Besides, even though MDLCK and all SRCMD(*s*) should be read-only, to ensure the semantic compatibility to the full model, following rules are defined: for reading SRCMD(*s*), SRCMD(*s*).md should be 1<<s, and SRCMD(*s*).l should be 1. As to MDLCK.md, all implemented MDs should be hardwired to 1. MDLCK.l should also be wired to 1. There is no constrain on the MDCFG table and the MDCFGLCK register.

3.6. The Compact-*k* Model

The compact-*k* model can achieve even lower latency and smaller area than the isolation model. Besides having each SID exactly associated with one MD, every MD should have exactly k IOPMP entries. Once SID is known, the IOPMP entries can be selected efficiently. In the model, MDLCK, all SRCMD, MDCLK, and all MDCFG.t are read-only. When read, MDLCK and all SRCMD should be the same as the isolation model. MDCFGLCK and all MDCFG.t should be the same as the rapid-k model. MDCFG.l and MDCFG.f can still be programable or hardwired.

3.7. Model Detections

To distinguish the above models, the user can read register HWCFG1.model to determine the current implemented IOPMP model.

Chapter 4. Tables Reduction and Detection

The full model has two tables and one array. It provides good flexibility to configure an IOPMP but sacrifices the latency and the area. In this section, we introduce the methods to reduce these tables in order to reach different design requirements. Some of the bits can be hardwired. However, for the sake of software detection and portability, the values read from these hardwired bits should maintain the same sematic as that of the full model.

The latency consideration here is about checking a transaction instead of accessing the IOPMP control registers because programming IOPMPs is considered less frequent. That is, we will not address the latency of updating the tables or the IOPMP entries.

As to the IOPMP array, it is the body of storing the IOPMP entries, so it cannot be omitted. We can only reduce its size. Memory domains can be shared among different SIDs, so the entries belonging to these shared MDs are shared among SIDs. Sharing entries may help to reduce the size of the IOPMP array.

The SRCMD table can be hardwired fully or partially to save area. For some cases, we can farther save the latency. Every SRCMD(*s*).MD should have the same programmable bits, so one can just detect SRCMD(*s*).MD if SRCMD(*s*).L is not wired to 1. If all SRCMD(*s*).L are wired to 1, there is no reason to implement MDMSK. If all bits in SRCMD(*s*).MD are hardwired, SRCMD(*s*).L should be wired to 1, too.

The SRCMD table can be replaced by simple circuits in order to save area or latency. A special reduction makes SRCMD(*s*).MD=(1 << *s*) for all *s*. It replaces a table look by a binary decoder, which shortens the latency to retrieve the corresponding MD. In the case, there is no any shared MD and it supports up to 63 SIDs.

As to the MDCFG table, the MD$_m$CFG.T can also be hardwired to save area and/or shorten the latency in some cases. It means that the ownership of every IOPMP entry is fixed all the time. Hardwiring MD$_m$CFG.F to a non-zero value is not a usual case because it makes some highest priority IOPMP entries nonprogrammable by software at any time. If MD$_m$CFG.F is hardwired, MD$_m$CFG.L should be wired to 1.

A special reduction makes MD$_m$CFG.T=$(m+1)*k$ for all m. It replaces a table lookup by a simple circuit, e.g., k is a power of 2. It can save some area and shorten the latency as well.

Every MD$_m$CFG should have the same programmable bits in an IOPMP except the dynamic-*k* model. We will describe it in the next section.

IOPMP-SE is a special case since it only needs the IOPMP entry array. The two tables are not needed, not even hardwired, and do not occupy any address space.

Chapter 5. Registers

OFFSET	**Register**	**Description**
0x0000	INFO	
	VERSION	Indicates the IP version etc.
	HWCFG	Indicates the configurations of current IOPMP instance
	ENTRYOFFSET	Indicates the internal address offsets of each table.
	ERRREACT	
	Programming Protection	
	MDSTALL	
	SIDSCP	
	Configuration Protection	
	MDMSK	Lock Register for SRCMD table.
	MDCFGLCK	Lock register for MDCFG table
	ENTRYLCK	Lock register for IOPMP entry array.
	Error Reporting	
	ERR_REQADDR	
	ERR_REQID	
	ERR_REQINFO	
	ERR_IRQSTAT	
	ERR_IRQMASK	
0x0800	MDCFG Table, *m* =0…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…sid_num-1	
	SRCMD_EN(s)	Bitmapped MD enable register, 's' corresponding to number of sources, it indicate source s associated MDs.
	SRCMD_R(s)	(Optional)Bitmapped MD read eanble register, 's' corresponding to number of sources, it indicate source s read permission on MDs.
	SRCMD_W (s)	(Optional)Bitmapped MD write eanble register, 's' corresponding to number of sources, it indicate source s write permission on MDs.

OFFSET	**Register**	**Description**
ENTRYOFFSET	Entry Array, *i* =0…entry_num-1	
	ENTRY_ADDR(*i*)	
	ENTRY_ADDRH(*i*)	(Optional for 32-bit system)
	ENTRY_CFG(*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	Description
vendor	23:0	R	the vendor ID
specver	31:24	R	the specification version

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

HWCFG0			
0x0008			
Field	Bits	R/W	Description
md_num	6:0	R	Indicate the supported number of MD in the instance
sid_num	15:7	R	Indicate the supported number of SID in the instance
entry_num	30:16	R	Indicate the supported number of entries in the instance
enable	31	W1	Indicate if the IOPMP checks transactions. 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	Description
tor_en	0:0	R	Indicate if TOR is supported
sps_en	1:1	R	Indicate the secondary permission settings is supported
user_cfg_en	2:2	R	Indicate the if user customized attributes is supported

prog_prient	3:3	W0	A sticiky bit to indicate if prio_entry is programmable. Reset to 1 if the implementation supports programmable prio_entry, otherwise, wired to 0.
model	7:4	R	Indicate the iopmp instance model • 0x0: Full model: the number of MDCFG registers is equal to HWCFG.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: Dyanmic-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 HWCFG.md_num, all MDCFG registers are readable and writable. • 0x4 Compact-k model: a single MDCFG register to indicate the k value, read only.
prio_entry	30:16	WARL	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.

ENTRYOFFSET			
0x0010			
Field	Bits	R/W	Description
offset	31:0	R	Indicate the offset address of the IOPMP array from the base of an IOPMP instance, a.k.a. the address of HWCFG. Note: the offset is a signed number. That is, the IOPMP array can be placed in front of HWCFG.

ERRREACT			
0x0018			
Field	Bits	R/W	Description
l	0:0	W1	Lock fields to ERRREACT register except ip
ie	1:1	RW	Enable the interrupt of the IOPMP
ip	2:2	RW1C	Indicate if an interrupt is pending on read. for 1, the illegal capture recorder (ERR_XXXX) won’t be updated even on subsequent violations. Write 1 clears the bit and the illegal recorder reactivates. Write 0 causes no effect on the bit.
ire	4:4	WARL	To triggle the interrupt on illegal read if ie = 1

rre	5:7	WARL	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
ire	8:8	WARL	To triggle the interrupt on illegal write if ie = 1
rwe	9:11	WARL	Response on write illegal access • 0x0: respond a bus error • 0x1: respond a decode error • 0x2: respond a success • 0x3~0x7: user defined
rsv	12:15	ZERO	must be zero, reserved for future
pee	28:28	WARL	Enable to differentiate between a prefetch access and an illegal access
rpe	29:31	WARL	Response on prefetch error • 0x0: respond a bus error • 0x1: respond a decode error • 0x2~0x7: user defined

5.2. Programming Protection Registers

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

MDSTALL			
0x0020			
Field	Bits	R/W	Description
exempt	0:0	W	Stall transactions with exempt selected MDs, or Stall selected MDs.
is_stalled	0:0	R	Indicate if the requested stalls have occured
md	30:0	W	setting MD[*i*]=1 selects MD *i*.
md	30:0	R	MD[*i*]=1 means MD *i* selected.

MDSTALLH

0x0024			
Field	Bits	R/W	Description
md	31:0	W	setting MD[*i*]=1 selects MD (*i*+31)
md	31:0	R	MD[*i*]=1 means MD (*i*+31) selected

SIDSCP			
0x0028			
Field	Bits	R/W	Description
op	31:30	W	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: 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
sid	15:0	WARL	SID to select

5.3. Configuration Protection Registers

MDLCK is a register with a bitmap field to indicate which MDs are locked.

MDLCK			
0x0040			
Field	Bits	R/W	Description
l	0:0	W1	Lock bit to MDLCK and MDLCKH register.
md	31:1	WARL	md[*j*] indicates if MD *j* in SRC*i*MD is locked for all *i*.

MDLCKH			
0x0044			
Field	Bits	R/W	Description
md	31:0	WARL	md[*j*] indicates if MD (*j*+31) in SRC*i*MD is locked for all *i*.

MDCFGLCK is the lock register to MDCFG table.

MDCFGLCK			
0x0048			
Field	Bits	R/W	Description
l	0:0	W1	Lock bit to MDLCK and MDLCKH register.

f	7:1	RW	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.

ENTRYLCK is the lock register to entry array.

ENTRYLCK			
0x004C			
Field	Bits	R/W	Description
l	0:0	W1	Lock bit to ENTRYLCK register.
f	15:1	WARL	Indicate the number of locked IOPMP entries – IOPMP_ENTRY[*f*-1:0] is locked. SW shall write a value that is no smaller than current number.

5.4. Error Capture Registers

ERR_REQADDR indicate the errored request address.

ERR_REQADDR			
0x0060			
Field	Bits	R/W	Description
addr	31:0	R	Indicate the errored address

ERR_REQADDRH			
0x0064			
Field	Bits	R/W	Description
addrh	31:0	R	Indicate the errored address

ERR_REQSID Indicate the errored SID.

ERR_REQSID			
0x0068			
Field	Bits	R/W	Description
sid	15:0	R	Indicate the errored SID.

ERR_REQINFO Captures more detailed error infomation.

ERR_REQINFO			
0x006C			
Field	Bits	R/W	Description
no_hit	0:0	R	Indicate the request hit no entry.

par_hit	1:1	R	Indicate the request failed due to partial hit.
type	10:8	R	• Indicated if it’s a read, write or user field violation. • 0x0 = read error • 0x1 = write error • 0x3 = user_attr error
eid	31:16	R	Indicated the errored entry index.

5.5. SRCMD Table Registers

0x1000 + (s)*32			
SRCMD_EN(s), s = 0...sid_num-1			
Field	Bits	R/W	Description
l	0:0	W1	A sticky lock bit. When set, locks SRCMD_EN(*i*), SRCMD_R(*i*) and SRCMD_W(*i*)
md	31:1	WARL	md[*j*] = 1 indicates md *j* is associated with SID *s*.

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

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

SRCMD_R(s), s = 0...sid_num-1			
0x1008 + (s)*32			
Field	Bits	R/W	Description
md	31:1	WARL	md[*j*] = 1 indicates SID *s* has read permission to the corresponding MD[*j*].

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

SRCMD_W(s), s = 0...sid_num-1			
0x1010 + (s)*32			
Field	Bits	R/W	Description

md	31:1	WARL	md[*j*] = 1 indicates SID *s* has write permission to the corresponding MD[*j*].

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

5.6. 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 HWCFG.md_num, all MDCFG registers are readable and writable.
2. Rapid-*k* model: a single MDCFG register to indicate the *k* value, read only.
3. Dyanmic-*k* model: a single MDCFG register to indicate the *k* value, readable and writable.
4. isolation model: the number of MDCFG registers is equal to HWCFG.md_num, all MDCFG registers are readable and writable.
5. Compact-*k* model: a single MDCFG register to indicate the *k* value, read only.

MDCFG(*m*), *m* = 0…HWCFG.md_num-1, support up to 63 MDs			
0x0800 + (*m*)*4			
Field	Bits	R/W	Description
t	16	WARL	• 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<MD0CFG.t. • If MDCFG(*m*-1).t >= MDCFG(*m*).t, then MD *m* is empty. • For rapid-*k*, dynamic-*k* and compact-*k* models, t indicates the number of IOPMP entries belongs to each MD.

5.7. Entry Array Registers

ENTRY_ADDR(*i*), *i* = 0…HWCFG0.entry_num-1			
ENTRYOFFSET + (*i*)*16			
Field	Bits	R/W	Description
addr	31:0	WARL	The physical address of protected memory region.

ENTRY_ADDR(*i*), *i* = 0…HWCFG0.entry_num-1			
ENTRYOFFSET + 0x4 + (*i*)*16			
Field	Bits	R/W	Description
addrh	31:0	WARL	The physical address of protected memory region.

ENTRY_CFG(*i*), *i* = 0…HWCFG0.entry_num-1			
ENTRYOFFSET + 0x8 + (*i*)*16			
Field	Bits	R/W	Description
r	0:0	RW	The read permission to protected memory region
w	1:1	WARL	The write permission to the protected memory region
x	2:2	WARL	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	The address mode of the IOPMP entry • 0x0: OFF • 0x1: TOR • 0x2: NA4 • 0x3: NAPOT

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…HWCFG0.entry_num-1			
ENTRYOFFSET + 0xC + (*i*)*16			
Field	Bits	R/W	Description
im	31:0	RW	User customized permission field

Chapter 6. Reset

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 STALL_EXEMPT and the field STALL_BY_MD having the same number of bits as the memory domains in the IOPMP. When STALL_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 STALL_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] <= STALL_EXEMPT ^ ( | (SRCiMD & STALL_BY_MD));
```

There are two 32-bit registers, MD_STALL and MD_STALLH, to store STALL_EXEMPT and STALL_BY_MD. STALL_EXEMPT is in the LSB of MD_STALL, STALL_BY_MD[30:0] is in MD_STALL[31:1] and STALL_BY_MD[62:31] is in MD_STALLH. If any MD*j* is not implemented, STALL_BY_MD[*j*] is wired to 0.

sid_stall should be captured only when STALL_EXEMPT is written, that is, MD_STALL is written. When MD_STALLH is written, the only action is to hold the value of STALL_BY_MD[62:31].

Although sid_stall is related to the SRCMD table, but should be captured only when STALL_EXEMPT is written. The behavior of writing MD_STALL 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 MD_STALL doesn't stall all the desired transactions, there is an optional method to pick the transaction with specific SIDs. The SID_SCP register comprises two fields: a 2-bit SID_SCP.OP and a field for SID_SCP.SID. By setting SID_SCP.OP=1, the sid_stall[*i*] is activated for *i*=SID_SCP.SID. Conversely, by setting SID_SCP.OP=2, the sid_stall[*i*] is deactivated for *i*=SID_SCP.SID. This register can be considered as the fine-tuning sid_stall after MD_STALL. The value of SID_SCP.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 MD_STALL. This corresponds to Step 3 of the "Programming Steps" section.

7.6. The Order to Stall

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

SID_SCP can be written multiple times or not at all. To determine whether all requested stalls take effect, one can read back the bit MD_STALL.IS_STALLED, which is in the same location as MD_STALL.EXEMPT on a write. MD_STALL.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 SID_SCP.OP and the SID you want to query to SIC_SCP.SID. Then, read back SID_SCP. The readback of SID_SCP.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 SID_SCP.SID. SID_SCP.STAT is in the same location as SID_SCP.OP on a write. SID_SCP.SID should keep the last written legal SID and SID_SCP.STAT reflects the current state of this SID. This method is considered an indirect way to read sid_stall.

7.7. Implementation-Dependecy

All registers described in this chapter are optional. Moreover, these features could be partially implemented. In STALL_BY_MD, 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 MD_STALL and MD_STALLH and then read them back. An implemented bit returns 1.

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

An example of partial implementation of STALL_BY_MD 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 STALL_EXEMPT=1 and STALL_BY_MD[*j*]=1, where MD*j* is the memory domain for the frame buffer of the display controller. Thus, the system only needs to implement STALL_BY_MD[*j*].

If the whole MD_STALL is not implemented, MD_STALL and MD_STALLH should always return zero, which means no bit implemented in STALL_BY_MD.

If SID_SCP 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 SID_SCP should not return a zero in SID_SCP.STAT in this case.

It is unnecessary to allow every implemented SID to be selectable by SID_SCP.SID. If an unimplemented or unselectable SID is written into SID_SCP.SID, it will return SID_SCP.STAT = 3 to respond to any defined SID_SCP.OP.

A1: Multi-Faulse Extension

TBD

A2: Run Out Memory Domains

TBD

A3: Sencondary Permission Setting

IOPMP/SPS is an extension to support different sources to share memory domain while allowing each sources to have different R/W/X permission to a signle memory domain.

If the IOPMP/SPS extension is implemented, each SRCMD table entry shall additionally define two registers: SRCMD_R(*s*) and SRCMD_R(*s*). Register SRCMD_R(*s*) and SRCMD_R(*s*) each occupies a 64-bit space, and has a fields, SRCMD_R(*s*).md and SRCMD_W(*s*).md respectively. Setting lock to SRCMD_EN(*s*).l also locks SRCMD_R(*s*) and SRCMD_W(*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.

Besides, IOPMP/SPS can offer the control of execution permission. If the signal indicating an instruction fetch is carried by a transaction, the second permission setting can control instruction fetches.

Bibliography