RISC-V IOPMP Specification Document

Paul Ku, Channing Tang, RISC-V IOPMP Task Group

Version 1.0, 09/2022: 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:

- Paul Shan-Chyun Ku <scku@andestech.com>
- Channing Tang <channingt@nvidia.com>

Chapter 1. Introduction

This document describes a mechanism to improve the security of a platform. In a platform, the bus masters on it can access the slave devices, just like a RISC-V hart. The introduction of IO peripherals 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 core-PMP and MPU which provide standard protection scheme for accesses from RISCV core to the physical address space, but there is not a likewise standard for safeguarding non-core masters. Here we propose the Physical Memory Protection Unit of Input/Output Devices, IOPMP for short, to regulate the accesses issued from the bus masters.

IOPMP is considered a hardware component in a bus fabric. But, why is a pure-software solution not enough? For software solution on a RISC-V-based platform, it's generally the security monitor, a program running on the M-mode takes care of security-related requests. However, it's impratical for the security monitor to check the legality of each request as the overhead and latency of trap-check-ret is non-negligible. And the latency becomes even worse when there are multipe inflight transactions in the system, for example, an DMA and a Crypto Accelerator with different privileges are making requests at the same time. A hardware component that can check accesses on the fly becomes a reasonable solution. That is the subject of this document, IOPMP.

Chapter 2. Concepts

This document uses the term security monitor to refer to the software in charge of security-related tasks. The security monitor also programs the IOPMPs. The security monitor is not confided to run on a core or a hart. It could be distributed on more than one core, virtualized on a virtual platform, or cascaded within a nested sub-platform.

For a register or a field X, X[n] represents the bit-n of X and X[n:m] for the bit-n to bit-m of X.

2.1. Source-ID and Transaction

Source-ID, SID for short, is a unique ID to identify a bus master or a group of bus masters with the same permission. When a bus master wants to access a piece of memory, it issues a transaction. A transaction should be tagged with an SID indicating which bus master issued it. We will discuss about the exception in the next section. Associating bus masters with SID could be implementation-dependent and is out of the scope of this document, but the IOPMP does have requirement on the uniqueness of SIDs which will be discussed when we talking about the cascading IOPMPs. The number of bits of an SID is implementation-dependent. However, there are some suggestions in this document if SID is programmable. SID could be a vulnerability because a malicious program can gain extra permission by forging SID. Hardwired SID can avoid such a risk. However, in the cases of requiring more flexibility, locking SID before entering REE can be a good choice. If a system provides programmable SIDs during the runtime, the write permission of SIDs should be controlled very carefully.

If a bus master has multiple channels and every channel is granted different access permissions, every channel should have its own SID. If a bus master runs in different privilege modes, such as a processor, every privilege mode should have a different SID if the system is designed to use IOPMPs to regulate its access. The usage of multiple SIDs is also applied to multiple virtual machines with different permissions.

2.2. Source-Enforcement

If the scope of an IOPMP contains only one bus master or a set of bus masters with the same permission, the Source-ID can be ignored in bus master side as well as the transactions going through the IOPMP. In the case, we denote the IOPMP performs source enforcement, IOPMP-SE for short. In the rest of the cases, we still need SID to distinct the transaction issuers.

2.3. Master Port and Slave Port

An IOPMP has at least a master port and at least a slave port. A slave port is where a transaction goes into the IOPMP, and a master port is where a transaction leaves it if the transaction passes all the checks.

2.4. Memory Domain

A memory domain, MD for short, is a concept inside an IOPMP. It is used to group a set of memory regions for a specific purpose and is 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 associate with this MD. However, associating the memory domain doesn't mean having full permissions on all memory regions. 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.

One thing should be noted: one SID may associate with more than one MD, and one MD may be associated with more than one SID. However, some models may limit the flexibility due to different requirements.

2.5. IOPMP Entry and IOPMP Entry Array

IOPMP entry array is the most fundamental structure of an IOPMP and is a list of ordered IOPMP entries. An IOPMP entry is indexed from zero and defines a rule when checking a transaction: including a memory region and the read/write permission. Each IOPMP entry belongs to exactly one memory domain, and a memory domain may have multiple IOPMP entries. An SID associating with an MD also means it associates with all IOPMP entries belonging to the MD.

IOPMP entries are statically prioritized. The lowest-numbered IOPMP entry that (1) matches any byte of the in-coming transaction and (2) is associated with the SID carried by the transaction determines whether the transaction is legal. The matching IOPMP entry must match all bytes of a transaction, or the transaction is illegal, irrespective of its permission.

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.

Chapter 3. IOPMP Models

To fit in various needs of the different platforms, we provide several IOPMP configuration models. We will begin with the full model which comes with practically all of the features listed in this document. Next, we'll discuss optional features and moving to the other models. These models assist users in choosing and refining their designs to meet various needs, 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*, it first lookups the SRCMD table to find out all the memory domains associated to *s*. The size of the table is implementation-dependent. In the table, the register SRC_sMD is defined per SID *s*, occupies a 64-bit space, and has two fields, $SRC_sMD.L$ and $SRC_sMD.MD$. $SRC_sMD.L$ is a sticky lock to this register. In the model, $SRC_sMD.MD$ is a bitmapped field and has up to 63 bits. Each bit here indicates if a MD is associated with the SID *s*. Not all bits should be implemented, but implemented bits should be right justified. For unimplemented memory domains, the corresponding bits in $SRC_sMD.MD$ should be WARZ. A full model IOPMP supports up to 63 memory domains. For a system requiring more memory domains than 63, one could cascade IOPMPs. Cascading IOPMPs is described in the Appendix.

Once an IOPMP retrieves all associated MDs for a transaction with SID *s*, it looks up the corresponding IOPMP entries belonging to these MDs. The MDCFG table defines the relation between MDs and IOPMP entries. The MDCFG table has an array of registers where the register MD_mCFG is for the memory domain *m*. One field, $MD_mCFG.T$, 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 $MD_{m-1}CFG.T_j < MD_mCFG.T$, where $m>0$. The MD *0* owns the IOPMP entries with index $j<MD_0CFG.T$. Each MD_mCFG register occupies a 32-bit space and the field $MD_mCFG.T$ 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.

Appendix provides a reference implementation for a full model IOPMP.

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. To lock the programmability of an IOPMP completely, we may not really need a new mechanism. If you have a platform similar to the above example, you could create a memory domain to stop any future IOPMP control operations. We will go through the idea first. If you want a fine-grained method that reserves partial programmability, the subsequent optional features are designed for it.

3.2.1. Protect the SRCMD Table

in order to enforce that all SIDs associate MD 0, you should lock the whole the SRCMD table due to the granularity. You are not able to lock all $SRC_sMD[0]=1$ (for all s) but leave the rest of the bits programmable. Thus, you lose all programmability of the mapping from SID to MD.

Here, there are two optional mechanisms to lock the SRCMD table partially. The register MDMSK can lock certain bits for each SRC_sMD.MD. MDMSK has two fields: a 63-bit MDMSK.MD and a 1-bit MDMSK.L. On MDMSK.MD[*m*]=1, all SRC_sMD.MD[*m*] are not programmable for all SID s. MDMSK.L is the lock bit for the MDMSK. In above example, when you want to enforce every SID to associate MD 0, you can first set all SRC_sMD.MD[0]=1, and then let MDMSK.MD[0]=1 and MDMSK.L=1. The rest mappings are still programmable. If MDMSK is not implemented, it should be WARZ.

For unimplemented memory domains, the corresponding bits of MDMSK.MD should be WARZ. The bits for implemented memory domains in MDMSK.MD can be also hardwired. However, in this case, the corresponding bits in SRC_sMD.MD should be well initialized during reset process. If whole MDMSK.MD is hardwired, MDMSK.L should be wired to 1. To figure out which memory domains are implemented, you can do the following steps: (1) set all ones (0x7fffffff_ffffffff) to SRC_0MD.MD, (2) read back the field, and (3) OR it with MDMSK.MD. The corresponding bits for implemented memory domains in the result will be 1's.

Besides, every SRC_sMD 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 SRC_sMD.L is implemented, SRC_sMD.L should be initialized to zero after reset. If SRC_sMD.L is not implemented, it can be hardwired to 0 or 1. If it is wired to 1, SRC_sMD.MD should be hardwired properly.

3.2.2. Protect the MDCFG Table

Subsequently, the belonging of IOPMP entries, that is MDCFG table, can 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 MD_{m-1}CFG.T. By introduction, if MD m is locked, MD n should also be locked, where $n < m$. Thus, we define all MD_mCFG.T are nonprogrammable where $m <$ 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. To enforce it, we use MD_mCFG.F, a 15-bit field in MD_mCFG, to define the number of nonprogrammable entries in the memory domain m; that is, the first MD_mCFG.F IOPMP entries belonging to MD m is not programmable. MD_mCFG.F is initialized to 0 and can be increased only when written. Besides, MD_mCFG.L is the lock to MD_mCFG.F and itself. If MDmCFG.F is not implemented, MD_mCFG.L and MD_mCFG.F should be wired to 1 and 0, respectively. Please note that MD_mCFG.L does not control whether or not MD_mCFG.T is programmable. MDCFGLCK.F does.

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 SRC_sMD.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. MD_mCFG.T should be *(m+1)*k*. That is, one can read MD_0CFG.T to retrieve the value *k*.

MD_mCFG.F and MD_mCFG.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.

MD_0CFG.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. MD_mCFG.T is read-only and equals to *(m+1)*k* when it is read. By updating MD_0CFG.T and then examining MD_1CFG.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 MDMSK and all SRC_sMD should be read-only, to ensure the semantic compatibility to the full model, we have the following rules. For reading SRC_sMD, SRC_sMD.MD should be 1<<s, and SRC_sMD.L should be 1. As to MDMSK.MD, all implemented MDs should be hardwired to 1. MDMSK.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, MDMSK, all SRC_sMD, MDCLK, and all MD_mCFG.T are read-only. When read, MDMSK and all SRC_sMD should be the same as the isolation model. MDCFGLCK and all MD_mCFG.T should be the same as the rapid-k model. MD_mCFG.L and MD_mCFG.F can still be programable or hardwired.

3.7. Model Detections

To distinguish the above models, one can follow the below approach.

First, we figure out how many MDs are implemented by (1) writing all ones to SRC_0MD.MD and (2) OR-ing the values and MDMSK.MD. Denote the result as *IMD*. The ones in *IMD* mean the implemented MDs.

Then, we test if the SRCMD table is programmable by reading MDMSK. Suppose MDMSK.L=1 and MDMSK.MD = *IMD*, the SRCMD table is read-only, and the IOPMP is either the isolation or the compact-k models. Subsequently, if MD_0CFG.T can accept zero, that is, writing zero and reading back a zero, the MDCFG table is programmable, and the IOPMP is the isolation model. Otherwise, it is the compact-k model because you can never have the compact-0 model.

If the SRCMD table is programmable, the IOPMP should be the rapid-*k* model, the dynamic-*k* model, or the full model. If MDCFGLCK.L is 1 and MDCFGLCK.F is non-zero, it is the rapid-*k* model. Then, if MD_0CFG.T accepts zero, it is the full model; otherwise, the dynamic-*k* 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 SRC_sMD.MD should have the same programmable bits, so one can just detect SRC_tMD.MD if SRC_tMD.L is not wired to 1. If all SRC_sMD.L are wired to 1, there is no reason to implement MDMSK. If all bits in SRC_sMD.MD are hardwired, SRC_sMD.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 SRC_sMD.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_mCFG.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_mCFG.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_mCFG.F is hardwired, MD_mCFG.L should be wired to 1.

A special reduction makes MD_mCFG.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_mCFG 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

TBD

Chapter 6. Reset

TBD

Chapter 7. Program IOPMPs

<TBD: pls feel free to raise comment or if you have better ideas> Programming the IOPMP usually consists of a series operation. There exists a potential race condition while one entity is programming the IOPMP, some transactions are checked or under checking but has not been completed yet. Such race condition may lead to vulnerabilities, for example, another entity starts to use the re-allocated region for confidential information but the entity who had the access is still able to retrieve the data with those transactions which already past security check. To mitigate such vulnerabilities, the series of IOPMP updates should be atomic in the view of other transactions that go through the IOPMP.

A straightforward thinking is: all transactions should be checked by either (1) the previous setting before any IOPMP updates or (2) the new setting after all IOPMP updates take effect. However, for some systems, stalling/blocking all transactions on the bus may impose a large impact on the performance; and in certain use cases, it is even not allowed to stall transaction from some specific sources. As an alternative, instead of stalling all transaction, one may implement the IOPMP atomic update in a way that only stalls transactions which are impacted.

Two register arrays are introduced to enforce an IOPMP atomic update:

- SIDBLKMODE: a register with one field SIDBLKMODE.FAULT to indicate if the blocked transactions should bebackpressred or faulted .
  - 0x0, the blocked transactions should be backpressred
  - 0x1, the blocked transactions should be faulted
- SIDBLKEN: Register array with one field SIDBLKEN.SID, SIDBLKEN.SID is a bitmapped field with each bit corresponds to one source. The number of registers required per each array is implementation-dependent. If a SIDBLKEN.SID bit is set, transaction from the corresponding source shall be blocked.
- SIDBLKSTAT: egister array with one field SIDBLKEN.SID, SIDBLKEN.SID is a bitmapped field with each bit corresponds to one source. When a bit is set, it indicates all transactions from that source are blocked.

A1: Reference Data Path

TBD

A2: Cascading IOPMPs

TBD

A3: Permission on Memory Domains

In the models mentioned so far, an IOPMP entry is a pair of a memory region and its corresponding permission setting. The bits used to store the memory region are much more than its permission setting. If different memory domains want to share these regions but not their permission settings, the IOPMP/PMD described in this appendix can help.

IOPMP/PMD extends every association bit in the SRCMD table to 4 bits of a second permission setting. We denote SPS[*s*, *m*] as the second permission setting for SID=*s* and MID=*m*. When a transaction arrives at an IOPMP/PMD, it looks up the corresponding memory domains as usual. Then, it also follows the original way to find the IOPMP entry matching the transaction. When checking the permission, IOPMP/PMD has two sets of permission settings: one from IOPMP entry and the other from the second permission setting that is retrieved from SPS. For either read or write operation, only if both permission settings allow, the transaction can do such operation.

Besides, 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.

As the programmability of every second permission setting, it is the same as the programmability of the bit in the same location in the SRCMD table.

SPS also supports up to 63 memory domains. The LSB of SPS[*s*, 63] is reserved for locking all second permission settings for SID = *s*.

Index

Bibliography