Intel® Resource Director Technology (Intel® RDT) Architecture Specification

September 2023

Revision 1.0

Document Number: 356688-001US

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel is a sponsor and member of the Benchmark XPRT Development Community and was the major developer of the XPRT family of benchmarks. Principled Technologies is the publisher of the XPRT family of benchmarks. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Contents

Figures

Tables

Revision History

Revision Number	Description	Date
001	• Initial release of the document.	September 2023

Glossary

Table 1-1 Glossary

Acronym	Term	Description
ACPI	Advanced Configuration and Power Interface	Advanced Configuration and Power Interface is an open standard that operating systems can use to discover and configure computer hardware components, to perform power management, auto configuration, and status monitoring.
CAT	Cache Allocation Technology	Software-guided redistribution of cache capacity is enabled by CAT, enabling important data center VMs, containers or applications to benefit from improved cache capacity and reduced cache contention. CAT may be used to enhance runtime determinism and prioritize important applications.
CDP	Code and Data Prioritization	As a specialized extension of CAT, Code and Data Prioritization (CDP) enables separate control over code and data placement in the L2 cache and the last-level (L3) cache. Certain specialized types of workloads may benefit with increased runtime determinism, enabling greater predictability in application performance.
CH	Channel	An I/O device channel, used to communicate between a device and an I/O Block and onto the coherent fabric.
CLOS	Class(es) of Service	A fundamental tag in RDT used for resource controls
CMT	Cache Monitoring Technology	Monitors the last-level cache (L3) utilization by individual threads, applications, or Virtual Machines, CMT improves workload characterization, enables advanced resource-aware scheduling decisions, aids “noisy neighbor” detection and improves performance debugging.
Intel® RDT	Intel® Resource Director Technology	Intel® RDT is the “umbrella” technology name for Intel’s Platform Quality of Service technologies, including CPU Agents and Non-CPU Agents.
I/O Intel® Resource Director Technology (Intel® RDT)	I/O Device Intel® Resource Director Technology	Intel RDT technologies specifically focusing on I/O devices including PCIe, CXL and integrated accelerators
MBA	Memory Bandwidth Allocation	MBA enables approximate and indirect control over memory bandwidth available to workloads, enabling new levels of interference mitigation and bandwidth shaping for “noisy neighbors” present on the system.
MBM	Memory Bandwidth Monitoring	Multiple VMs or applications can be tracked independently via Memory Bandwidth Monitoring (MBM), which provides memory bandwidth monitoring for each running thread simultaneously. Benefits include detection of noisy neighbors, characterization and debugging of performance for bandwidth-sensitive applications, and more effective non-uniform memory access (NUMA)-aware scheduling.

Acronym	Term	Description
MMIO	Memory Mapped I/O	I/O Intel RDT defines a series of MMIO-mapped interfaces to enable association of I/O devices to RMIDs and CLOS for monitoring and control.
PQR	PQR	A shorthand for the IA32_PQR_ASSOC MSR, which associates IA threads to RMID and CLOS tags.
RMD	Resource Management Domain	A set of features defined within a particular cache domain, such as an L3 cache supporting a number of logical processors.
RTD	Resource Telemetry Domain	A Resource Management Doman within which one or more resource monitoring (telemetry) controls are supported
RAD	Resource Allocation Domain	A Resource Management Doman within which one or more resource allocation controls are supported
RMID	Resource Monitoring ID(s)	A fundamental tag used for resource monitoring in Intel RDT.
SoC	System-on-Chip	An integrated chip composed of host processors, accelerators, memory, and I/O agents.
TC	Traffic Class	A PCI Express feature that allows differentiation of transactions to apply appropriate servicing policies.
VC	Virtual Channel	A PCI Express feature for differential bandwidth allocation. Virtual channels have dedicated physical resources (buffering, flow control management, and so on) across the hierarchy.
VMM	Virtual Machine Monitor	A software layer that controls virtualization.

1 Introduction

This document defines the architecture of the Intel® Resource Director Technology (Intel® RDT) feature set. The goal of Intel RDT is to bring new levels of monitoring and control over how shared platform resources such as last-level cache (L3) and main memory (typically DRAM) bandwidth are utilized by CPU Agents and non-CPU Agents. The monitoring and allocation are not necessarily applied across the entire system but are applied to a Resource Management Domain (RMD) which corresponds to a set of agents sharing a set of system resources, such as L2 cache capacity, L3 cache capacity, memory bandwidth, and I/O devices. A Resource Management Domain (RMD) consists of a collection of CPU agents or non-CPU agents. The set of CPU agents consist of one or more logical processors associating an RMID and/or CLOS tag with a software thread. Non-CPU agents include PCI Express* (PCIe*)/Compute Express Link (CXL)* devices and integrated accelerators, thus broadly encompassing the set of agents which read from and write to either caches or memory, excluding IA cores.

The Intel RDT feature set provides a series of monitoring and allocation capabilities such as Cache Monitoring Technology (CMT), Memory Bandwidth Monitoring (MBM), Cache Allocation Technology (CAT), Code and Data Prioritization (CDP), Memory Bandwidth Allocation (MBA) and others. These technologies enable monitoring and control of shared platform resources, such as the L3 cache capacity or main memory bandwidth, which may be in use by many applications, containers or VMs running on the platform concurrently. As described in subsequent chapters, these features enable deterministic behavior and fairness in communications, real-time and other usages, and are initially introduced in Section 1.3.

The Intel RDT features are based on a set of architectural tags, described in the following section, and fundamental capabilities for enabling monitoring and control over shared platform resources under the control of an operating system (OS) or virtual machine monitor (VMM), as described in the chapter on Reference Software Architecture.

1.1 High Level Usage Models

A wide variety of industry deployment models find value in either enhanced visibility into system resource utilization, or control over shared resources. As a result, a broad set of customer usage models are observed with Intel RDT, including but not limited to:

- **Cloud Hosting in the datacenter** – Prioritizing important Virtual Machines (VMs) and containing or mitigating "noisy neighbors".
- **Public/Private Cloud** – Isolating an important infrastructure VM which provides networking services such as a VPN to bridge the private cloud to the public cloud.

- **Datacenter Infrastructure** – Protecting virtual switches which provide local networking.
- **Communications** – Ensuring consistent performance and containing background tasks on a network appliance built atop an Intel® Xeon® Server Platform.
- **Content Delivery Network (CDN)** – Prioritizing key parts of the content serving application in order to improve throughput.
- **Networking** – Containing the impact of consolidated or co-located containers to help reduce jitter and reduce packet loss in noisy scenarios, and protecting high-performance applications based on the Dataplane Development Kit (DPDK).
- **Industrial Control** – Prioritizing important sections of code to help meet real-time requirements.

Varying usage models drive differing requirements. Datacenter usages may require control over relative container prioritization and management of tail latencies, for instance, while industrial control usages may require strict management of control loop cycle times, including the use of model-specific extended Intel RDT features. A number of examples use cases are described in more detail based on abstracted examples of real-world deployments in the chapter on Reference Software Architecture.

1.2 Scope

Broadly, this document discusses the following topics:

- An introduction to key Intel RDT architectural concepts and design philosophy.
- Details of architectural Intel RDT monitoring and allocation features for CPU agents and non-CPU agents.
- Details of model-specific Intel RDT monitoring and allocation features for CPU agents and non-CPU agents.
- Considerations for BIOS writers, and those consuming ACPI enumeration tables generated by BIOS.
- An overview of various real-world software usages of Intel RDT features that have been observed, and recommended software enabling strategies.

The following topics are not covered (or are covered in a limited context):

- Intel RDT for CPU Agents and non-CPU Agents architectural details - feature enumeration and interfaces using CPUID and configuration using MSRs. These details are provided in the Intel® 64 Architecture Software Developer's Manual (SDM), Volume 3B, Chapter Title: Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.

1.3 Audience

The intended audience for this specification includes Intel RDT consumers, users and implementers, across OS/VMM software, resource management driver and control loop developers, administrators, managers of datacenter infrastructure, workload owners and embedded and communications developers. Additionally, this specification may be of interest to those developing utilities, BIOS routines, administrative libraries and orchestration frameworks.

1.4 References

Table 1-1. References

Description
[1] Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3B, Chapters 18.18 and 18.19. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
[2] Intel® Architecture Instruction Set Extensions and Future Features. Instruction Set Architecture (intel.com)
[3] Intel® Virtualization Technology for Directed I/O Specification. http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html
[4] Unified Extensible Firmware Interface Forum – Links to ACPI-Related Documents (incudes IRDT table title and signature). https://uefi.org/acpi
[5] PCIe Express Specification, v5.0 or newer. https://pcisig.com/specifications
[6] Compute Express Link Specification, v1.0 or newer. https://www.computeexpresslink.org/download-the-specification
[7] User space software for Intel® Resource Director Technology https://github.com/intel/intel-cmt-cat

2 Intel® Resource Director Technology Overview

This chapter provides an overview of Intel® RDT features, including goals, key ingredients, and the architectural framework, which are discussed in more detail in the chapters that follow.

2.1 Common Tags

Intel RDT provides a layer of abstraction between applications and logical processors through the use of numeric tags. Both CPU agents and non-CPU agents use the following tags for resource monitoring and allocation, respectively:

- Resource Monitoring IDs (RMIDs) are used for monitoring of shared platform resource utilization.
- Classes of Service (CLOS) are used for control of shared platform resources, such as L3 cache occupancy or memory bandwidth.

The RMID and CLOS tags are described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. RMID and CLOS tags are independent. Usage of RMID tags does not affect CLOS, and vice versa (however, when CLOS tags are used to affect resource allocations, the effects may be observed with RMID-based monitoring features.) An RMID-based monitoring feature does not incur hardware overhead or affect a CLOS-based allocation feature. A product may be built to implement RMID-based monitoring features, CLOS-based control features, or both.

For CPU agents, RMIDs and CLOS tags are associated with the operation of a logical processor through the IA32_PQR_ASSOC MSR.

For non-CPU agents, a series of MMIO interfaces is used to associate upstream traffic from I/O devices with RMID and CLOS tags, and the numerical interpretation of the tags is the same as for processor traffic. (For example, the RMID value "5" used to track processor thread resource consumption means the same thing as when the RMID value "5" is used to track the cache fill behavior of a PCIe device.) These MMIO interfaces for tagging non-CPU agents are discovered using an ACPI structure called I/O Intel RDT, that is, IRDT. (see Chapter 5.)

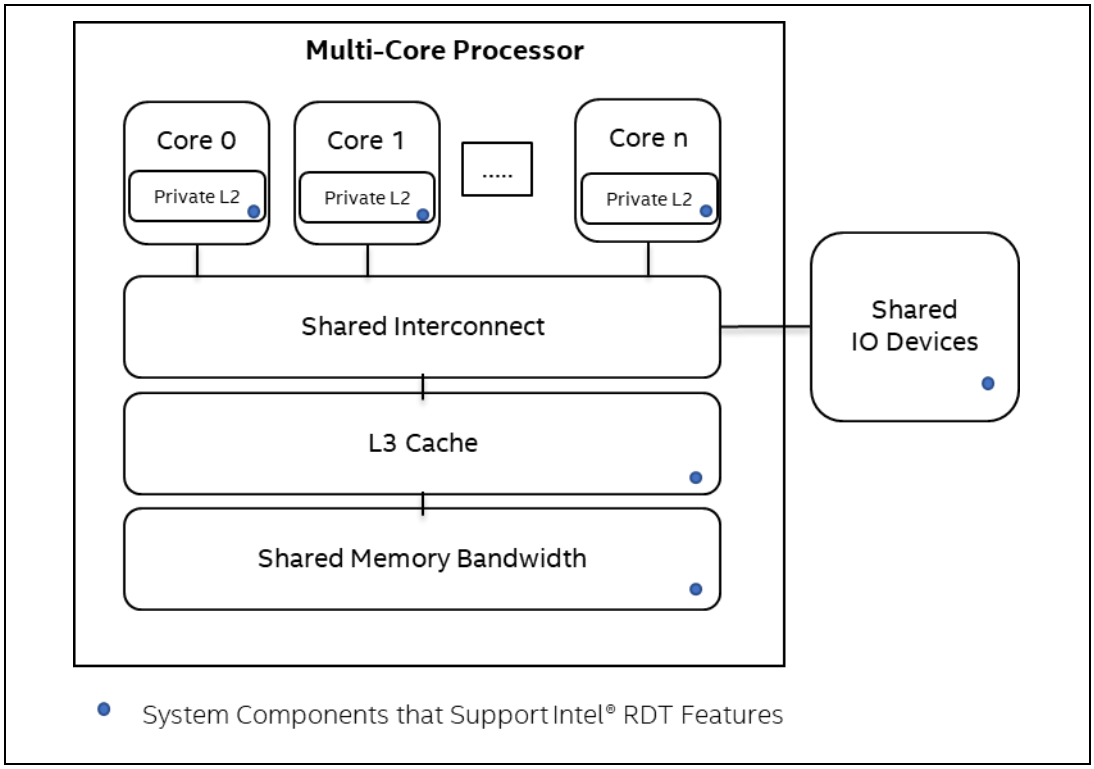
2.2 L3 Configurations

This specification describes two types of high level L3 configurations that may support Intel RDT features:

1. **Shared-L3 Configuration:** There is a common shared L3 cache for all the agents in the SoC, as shown in Figure 2-1. This SoC configuration supports interfaces for Intel RDT features based on the CPUID instruction

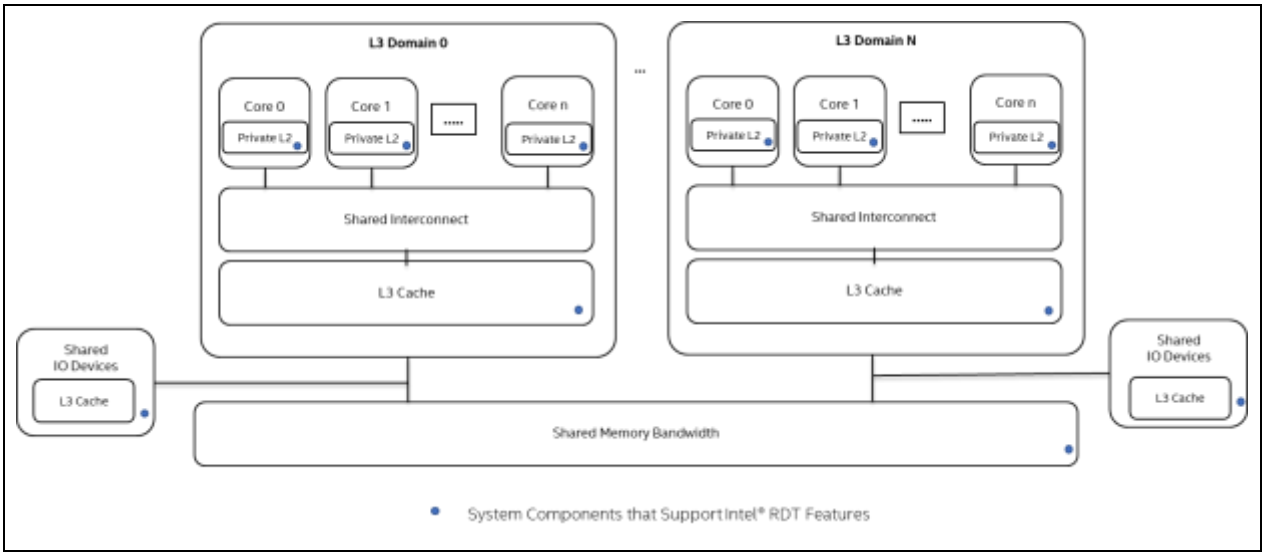
for feature enumeration and Model-Specific Registers (MSRs) for feature configuration and telemetry retrieval.

Figure 2-1. Shared-L3 Configuration System Model and Presence of Intel® RDT Features



2. **Multiple-L3 Configuration:** There may be more than one L3 cache instances that are local to CPU Agents or non-CPU Agents respectively, as shown in Figure 2-2.

Figure 2-2. Multiple-L3 Configuration System Model and Presence of Intel ® RDT Features



A set of features defined within a particular cache domain, such as an L3 cache supporting a number of logical processors, may be referred to as a Resource Telemetry Domain (RTD, for monitoring features) or a Resource Allocation Domain (RAD, for allocation features). More generally, a resource which supports Intel RDT monitoring features, allocation features or both may be referred to as a Resource Management Domain (RMD).Figure 2-2 shows example of multiple RMDs.

See Appendix A.4 for Intel RDT feature mapping for CPU agents and non-CPU agents in different SoC configurations.

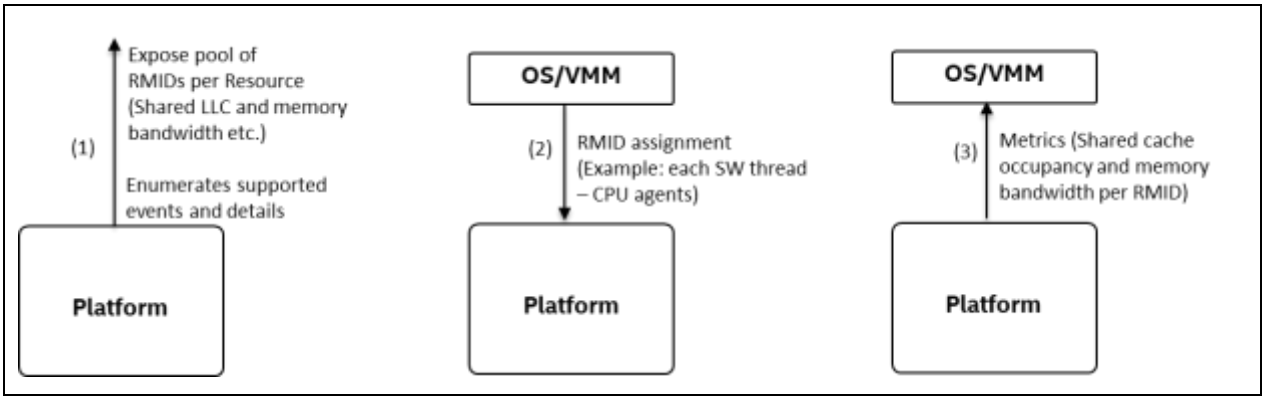
2.3 Intel® RDT Monitoring Technologies

2.3.1 Intel® RDT Monitoring Key Ingredients

Intel RDT Monitoring enables monitoring shared platform resources, such as L3 cache occupancy and memory bandwidth, based on software-defined Resource Monitoring IDs (RMIDs) that are tagged to applications or VMs on a per-thread basis (Figure 2-3). For CPU Agents, each logical processor exposes the IA32_PQR_ASSOC MSR to allow the OS/VMM to specify an RMID when an application, thread or VM is scheduled on a core.

Resource monitoring for the indicated application/thread/VM is then performed by hardware based on the RMID with which it is associated, and software can read back the L3 cache occupancy for a given RMID via counter registers (if the CMT feature is supported for instance). Each thread of an application may be tracked with a distinct RMID, or threads may be grouped into a single RMID, based on the granularity of monitoring required. Threads within a VM, apps within a VM, entire VMs or groups of VMs can similarly be tracked with RMIDs with variable granularity as needed.

Figure 2-3. Intel® RDT Monitoring – Enabling RMID-Based Monitoring for Shared Resources



The basic ingredients of Intel RDT Monitoring are as follows:

- CPUID and/or ACPI constructs to indicate support for Intel RDT Monitoring and sub-features (CMT, MBM, and so on) for Resource Telemetry Domains (RTD).

- Enumeration of the total number of RMIDs that can be tracked in the given RTD.
- Mechanisms to allow system software (OS/VMM) to specify the RMID of software threads and non-CPU agents.
- Mechanisms to allow system software to retrieve collected metrics on a per-RMID basis via architectural MSRs or MMIO interfaces.

The first ingredient to make use of Intel RDT Monitoring is to enumerate the set of monitoring capabilities provided on the given Resource Management domain via CPUID or ACPI and determine the number of RMIDs available for tracking on a particular Resource Telemetry Domain (RTD, that is, caching domain). This will allow the OS/VMM to determine how many unique IDs it may use. Given that certain processor topologies may include heterogenous capabilities which vary per-processor, it is recommended that software enumerate Intel RDT CPUID leaves from the perspective of each logical processor (LP) to construct the list of supported capabilities and which resources (such as L3 cache) may be shared among various LPs.

The second ingredient (Intel RDT Monitoring association) allows the OS/VMM to specify the RMID of the running software thread to the platform for CPU agents. The OS/VMM can also specify the RMID for upstream traffic and operation of non-CPU agents.

The third ingredient (Intel RDT marking and associated hardware support) enables each memory request from the CPU agents and non-CPU agents to be tagged with the RMID provided by the OS/VMM.

The fourth ingredient is Intel RDT Monitoring reporting. When the monitoring data retrieval register is programmed with the RMID and the specific event code of interest (L3 Cache Occupancy for example), this information is appropriately retrieved and provided back.

Multiple Intel RDT Monitoring features may exist within a platform, but the software should not assume that the presence of one Intel RDT Monitoring feature implies the existence of any others. Intel RDT features are independently enumerated in the sequence described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, Section 18.18.4, in order to avoid ambiguous situations.

2.3.2 Shared-L3 versus Multiple-L3 Configuration

Intel RDT Monitoring features may have different scope definitions depending on L3 configuration. With the shared-L3 configuration, CPU agents and non-CPU agents allocate into a shared L3 cache. Hence, all monitoring features have a consistent definition for CPU agents and non-CPU agents.

With the multiple-L3 configuration, non-CPU agents may have a separate nearby L3 cache which is distinct from CPU agents' L3 cache. Hence, monitoring features may have different definitions for CPU agents and non-CPU agents. For example, in certain implementations, non-CPU agents with a near

L3 cache implementation may report memory bandwidth monitoring data from the near cache only.

2.4 Intel® RDT Allocation Technologies

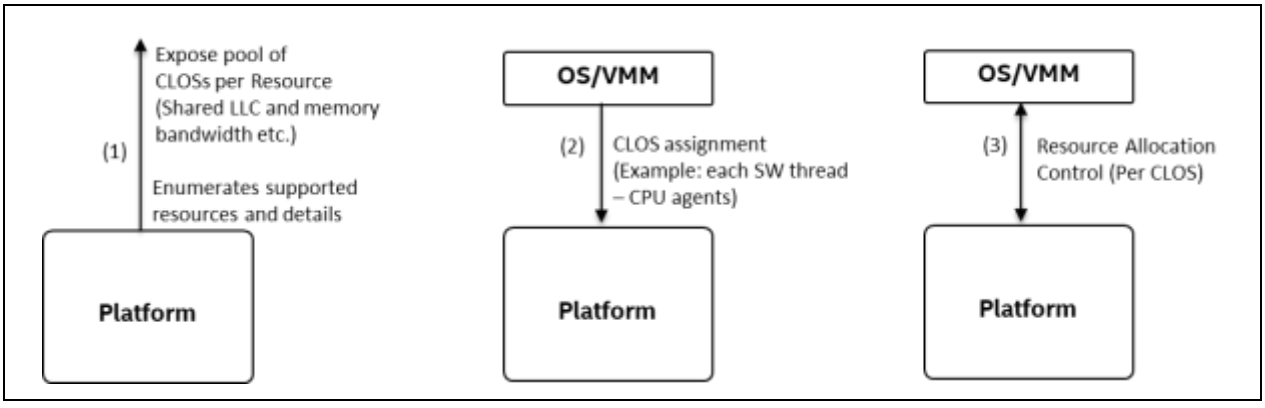
2.4.1 Intel® RDT Allocation Key Ingredients

Intel RDT Allocation enables resource allocation based on Class of Service (CLOS) tags. The processor exposes Classes of Services into which applications (or individual threads) and traffic from I/O devices may be assigned. A CLOS may have multiple associated resource allocation properties. For example, there may exist controls for each CLOS to specify L2 capacity available to that CLOS, L3 capacity available, memory bandwidth available, and other properties (Figure 2-4).

In the case of L3 capacity control features, for instance, such as Cache Allocation Technology (CAT), the cache allocation for a given is restricted based on the class with which they are associated. Similarly, in certain implementations supporting non-CPU agent controls, context-associated and upstream traffic from I/O devices may be controlled as it utilizes shared system resources. Each CLOS can be configured using bitmasks which represent capacity, and the degree of overlap and isolation between classes in allocation features which influence the SOC caches.

For CPU agents, each logical processor exposes the IA32_PQR_ASSOC MSR to allow the OS/VMM to specify a CLOS when an application, thread or VM is scheduled. Cache Allocation for the application/thread/VM is then controlled based on the CLOS and the associated bitmask.

Figure 2-4. Intel® RDT Allocation – Enabling CLOS-based Allocation for Shared Resources



The basic ingredients of Intel RDT Allocation are as follows:

- CPUID or ACPI constructs to indicate whether Intel RDT Allocation and sub-features (CAT, MBA, and so on) for Resource Allocation Domains (RADs) are supported and enumerate the total number of CLOS that may be associated to shared platform resources on the platform.

- Mechanisms to allow system software (OS/VMM) to specify the CLOS of software threads and non-CPU agents.
- Mechanisms to allow system software to configure the shared platform resource levels available to each CLOS via architectural MSRs or MMIO interfaces.

The first ingredient to make use of Intel RDT Allocation is to enumerate the level of allocation capability provided on the given Resource Allocation Domain via CPUID and/or ACPI and determine the number of CLOSs available for allocating shared platform resources on a particular RAD (that is, a certain L3 caching domain). This will allow the OS/VMM to determine how many unique IDs it may use. Given that certain processor topologies may include heterogenous capabilities which vary per-processor, it is recommended that software enumerate Intel RDT CPUID leaves from the perspective of each logical processor (LP) to construct the list of supported capabilities and which resources (such as L3 cache) may be shared among various LPs.

The second ingredient (Intel RDT Allocation association) allows the OS/VMM to specify the CLOS of the running software thread to the platform for CPU agents. The OS/VMM can also specify the CLOS for upstream traffic and operation of non-CPU agents.

The third ingredient (Intel RDT marking and associated hardware support) enables each memory request from CPU agents and non-CPU agents to be tagged with the CLOS provided by the OS/VMM.

The fourth ingredient is Intel RDT Allocation control, when the allocation register is programmed with the CLOS and allocation control is performed by the specific shared platform resource (L3 Cache capacity for example).

Multiple Intel RDT Allocation features may exist within a platform. The software should not assume that the presence of one RDT Allocation feature implies the existence of any others. Intel RDT features are independently enumerated in the sequence described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, in order to avoid ambiguous situations.

2.4.2 Shared-L3 versus Multiple-L3 Configuration

Intel RDT Allocation features may have different definitions depending on L3 configuration. With the shared-L3 configuration, CPU agents and non-CPU agents allocate into a shared-L3 cache. Hence, all allocation features have a consistent definition for CPU agents and non-CPU agents. With the multiple-L3 configuration, non-CPU agents may have a separate near L3 cache which is different from the CPU agents' L3 cache. Hence, allocation features may have different definitions for CPU agents and non-CPU agents. For example, non-CPU agents with a near L3 cache implementation provide separate interfaces for cache capacity allocation for the near L3 cache.

Chapter 3 and Chapter 4 provide details about each Intel RDT Monitoring and Allocation features for CPU agents and non-CPU agents.

3 Intel® Resource Director Technology for CPU Agents

This chapter contains an overview of the Intel RDT features for CPU agents. Chapter 4 describes details about features for non-CPU agents.

3.1 Intel® RDT Monitoring Features

The Intel RDT Monitoring architecture enables monitoring of the utilization level of critical shared platform resources and provides this data directly to the Hypervisor, Operating System or other privileged software. Intel RDT Monitoring supports three event codes: 1) L3 cache occupancy 2) L3 Total External bandwidth 3) L3 Local External bandwidth. This allows more efficient scheduling based on resource use, as well as application tuning and performance prediction based on resource use characterization, and optionally better reporting and billback. This functionality complements Intel RDT Allocation, which provides control over shared platform resources available to CPU agents.

3.1.1 Common Framework

The following mechanisms are shared by Intel RDT Monitoring features:

- CPUID feature bits to enumerate the presence of the Intel RDT Monitoring capabilities and the details of each sub feature.
- The IA32_PQR_ASSOC MSR, which the OS or Hypervisor uses to specify the RMID for each software thread scheduled to run on a logical processor. See Figure 3-2.
- The IA32_QM_EVTSEL and IA32_QM_CTR MSRs, to read cache occupancy and bandwidth statistics. See Figure 3-3.

Software may flexibly associate RMIDs with threads, applications, VMs, or containers. (See Figure 3-1). If multiple logical processors within a Resource Telemetry Domain (RTD) are assigned the same RMID, the total resource monitoring telemetry by these logical processors will be accumulated together and the total reported by hardware.

Monitoring data is retrieved using a window-based interface. Software writes an event ID and RMID to the IA32_QM_EVTSEL MSR and hardware provides the resulting data back in the IA32_QM_CTR MSR.

Refer to Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for details on CPUID and MSR usage.

Figure 3-1. Resource Monitoring IDs (RMIDs) Assignment Flow

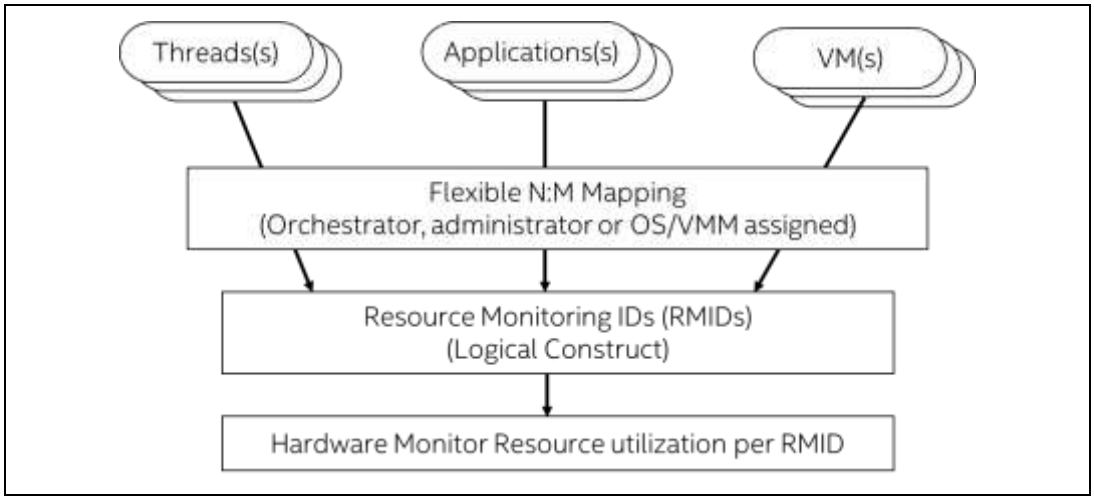


Figure 3-2. IA32_PQR_ASSOC MSR to Set RMID

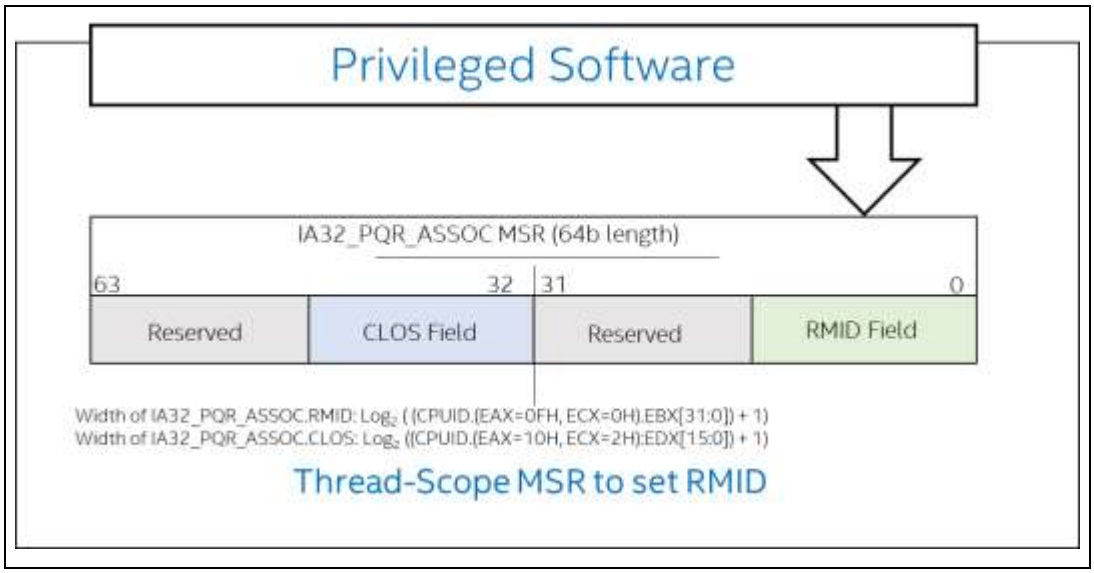
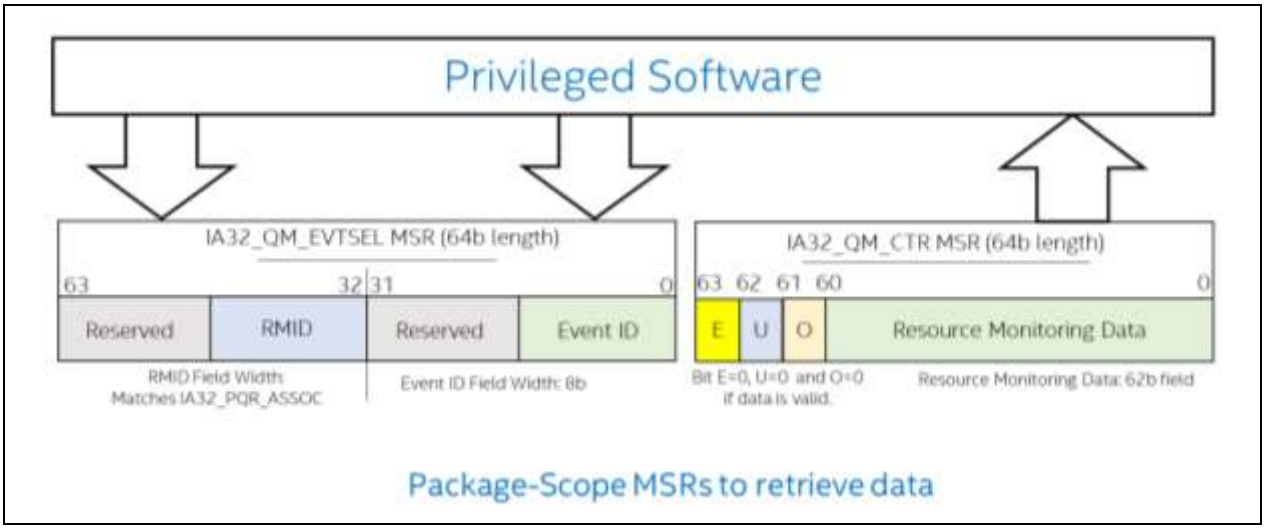


Figure 3-3. IA32_QM_EVTSEL and IA32_QM_CTR MSRs



3.1.2 Cache Occupancy Monitoring Technology

Intel RDT Cache Occupancy Monitoring Technologies provide visibility into cache utilization. Features such as Cache Monitoring Technology (CMT) provide occupancy counters on a per-RMID basis such that cache occupancy by each RMID may be tracked and read back in real-time during system operation.

More specific feature details about CMT are provided in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. Note that the MSRs are listed in Appendix A.5. See Appendix A.2 for CMT feature supported product details.

3.1.2.1 L3 Cache Monitoring Technology

L3 Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor or similar system management agent to determine the usage of L3 cache of the Resource Telemetry Domain (RTD) by applications running on the platform.

3.1.3 Memory Bandwidth Monitoring

Memory Bandwidth Monitoring (MBM) provides monitoring of bandwidth from one level of the cache or resource hierarchy to the next, allowing bandwidth-aware scheduling decisions, inter-RTD scheduling optimization, and enabling feedback to bandwidth allocation features which allow control over memory bandwidth.

More specific feature details about MBM are provided in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. Note that the MSRs are listed in Appendix A.5. See Appendix A.2 for MBM feature supported product details.

3.1.3.1 L3 Total and Local External Memory Bandwidth Monitoring

L3 Total and Local External Memory Bandwidth Monitoring allows system software to monitor the use of bandwidth between L3 cache and local or remote memory. In certain implementations, MBM is not guaranteed to track directory and Extended Prediction Table (XPT) prefetcher traffic.

3.2 Intel® RDT Allocation Features

The Intel RDT Allocation architecture enables control over utilization level of critically shared platform resources and provides this control directly to the Hypervisor or Operating System. This allows more efficient resource usage as well as application prioritization and determinism restoration based on resource repartitioning. The implementation of Intel RDT Allocation features may be product-specific or architectural. These capabilities compliment Intel RDT monitoring, which provides insight into shared platform resource utilization by CPU agents.

3.2.1 Common Framework

The following mechanisms are shared by Intel RDT allocation features:

- CPUID feature bits to enumerate the presence of Intel RDT Allocation capabilities and the details of each sub feature.
- The IA32_PQR_ASSOC MSR which software uses to specify the CLOS for each software thread. See Figure 3-5.
- Mechanisms in hardware to specify resource usage to apply to each Class of Service.

Software can flexibly associate Classes of Service with threads, applications, VMs, or containers (see Figure 3-4). CLOS values are shared across all allocation features. A particular numeric CLOS value has the same meaning from the viewpoint of all cores. Each CLOS has an associated set of mask registers as described later to associate that CLOS with a fraction of the shared platform resources. If multiple logical processors within a Resource Allocation Domain (RAD) are assigned the same CLOS, then resource allocations associated with that CLOS will be shared among that set of logical processors.

Figure 3-4. Classes of Service (CLOS) Association Flow

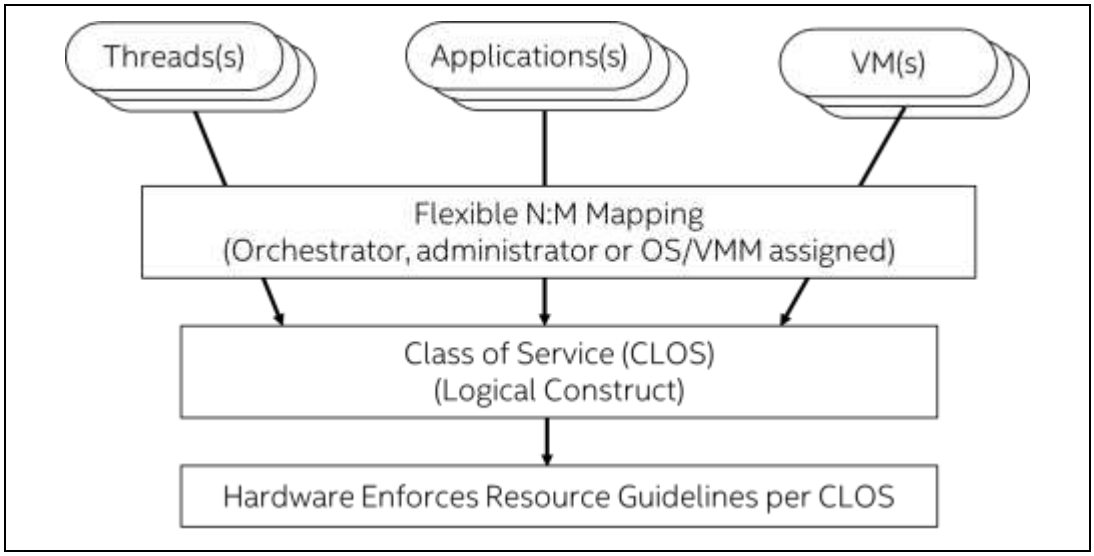
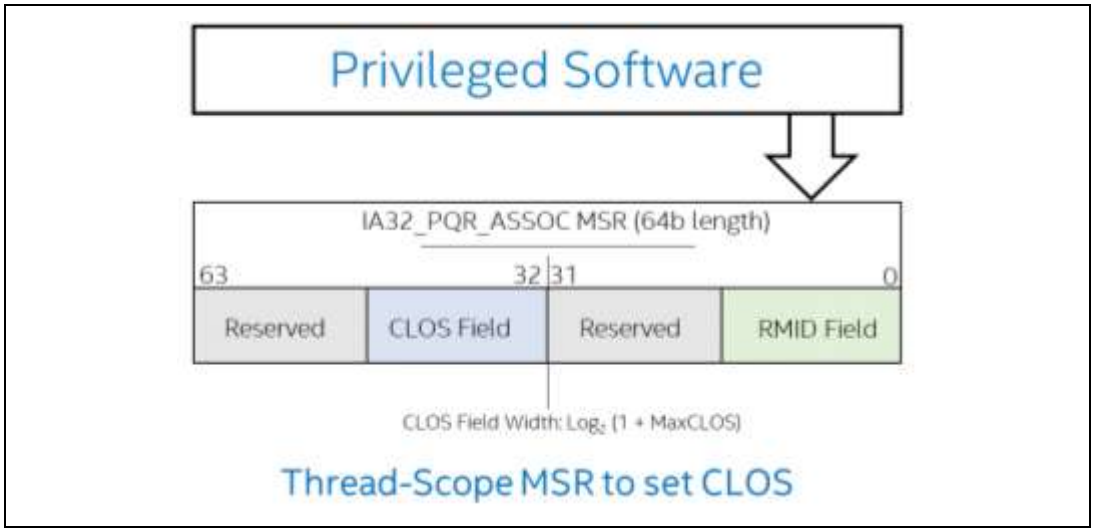


Figure 3-5. The IA32_PQR_ASSOC MSR to Set CLOS



For each resource, a block of registers is defined for software to configure the allocation values for each CLOS. The definition of the register fields depends on the type of resource being managed and is discussed in subsequent sections.

3.2.2 Cache Occupancy Allocation Technologies

A family of Cache Occupancy Allocation Technologies allows control over shared cache space on a per-CLOS basis, enabling both isolation and overlap for better throughput, fairness, determinism and differentiation. Typically, these features are known as Cache Allocation Technology (CAT), which is the term used in this document. Certain processors may support architectural or model-specific forms of CAT depending on the product generation. Model-specific implementations are discussed in Appendix B.1.4.

More specific feature details about CAT are provided in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. Note that the MSRs are listed in Appendix A.5. See Appendix A.2 for CAT feature supported product details.

3.2.2.1 L2 Cache Allocation Technology

L2 Cache Allocation Technology (L2 CAT) allows system software to specify the amount of L2 cache space of the Resource Allocation Domain into which an application can fill.

3.2.2.2 L2 Cache Code and Data Prioritization

L2 Code Data Prioritization (L2 CDP) provides differentiation between code and data for L2 cache usage by a single Class of Service. In a case where an application has a large code footprint which can overwhelm data in the cache, or vice versa, the ability to separately prioritize code and data is valuable.

L2 CDP provides a pair of allocation bitmasks for each Class of Service (rather than a single bitmask per CLOS as in L2 CAT), to allow system software to independently configure the amount of L2 cache available to code and data.

3.2.2.3 L3 Cache Allocation Technology

L3 Cache Allocation Technology (L3 CAT) allows an Operating System (OS), a Hypervisor, Virtual Machine Manager (VMM), or similar system service management agent to specify the amount of L3 cache space within a Resource Allocation Domain into which a CLOS may fill.

3.2.2.4 L3 Cache and Data Prioritization

L3 Code Data Prioritization (L3 CDP) provides differentiation between code and data for L3 usage by a single Class of Service. In a case where an application has a large code footprint which can overwhelm data in the cache, or vice versa, the ability to separately prioritize code and data is valuable.

L3 CDP provides a pair of allocation bitmasks for each Class of Service (rather than a single bitmask per CLOS as in L3 CAT), to allow system software to independently configure the amount of L3 cache available to code and data.

3.2.3 Memory Bandwidth Allocation

Memory Bandwidth Allocation (MBA) allows the system software to control access bandwidth to memory. It allows slowing "noisy neighbor" threads which may be overutilizing bandwidth and enables the creation of closed-loop control systems (monitoring and control combined) by exposing control over a credit-based throttling mechanism.

More specific feature details about MBA are provided in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. Note that the MSRs are

listed in Appendix A.5. See Appendix A.2 for MBA feature supported product details.

There are three different generations of MBA, each extending additional capabilities:

1. **First Generation MBA (Interface Scope)** – This is initial implementation of the MBA feature which provides indirect and approximate control over memory bandwidth available per-core. See Section 3.2.3.1 for implementation details and see Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for legacy enumeration, interface and per-CLOS delay value resolution details.
2. **Second Generation MBA (Interface Scope)** - This enhanced MBA capability provides improved efficiency and accuracy in throttling, along with providing increased system throughput. Rather than a strict bandwidth control mechanism, a dynamic hardware controller is implemented, which can react to changing bandwidth conditions at the microsecond level. Before using the second-generation MBA feature, the MBA hardware controller requires a BIOS-assisted calibration process that may include inputs such as the number of memory channels populated and other system parameters; this is a change from the first generation of MBA.
   Intel's BIOS reference code includes a default configuration that is recommended for general usage, and BIOS profiles may be created with alternate tuning values to optimize for certain usages (such as stricter throttling). See Section 3.2.3.2 for implementation details and Intel® 64 and the IA-32 Architectures Software Developer's Manual, Volume 3B, for legacy enumeration and interface details.
3. **Third Generation MBA (Agent Scope)** - The third generation MBA feature on future processors based on the codename Granite Rapids microarchitecture further enhances MBA with per-logical-processor control and a further improved controller design. Total memory bandwidth (all L3 miss traffic) is now managed by MBA 3.0. This implementation follows the past MBA precedent of delivering significant enhancements without a major software overhaul, and while preserving backward compatibility. See Section 3.2.3.3 for implementation details and Intel® 64 and the IA-32 Architectures Software Developer's Manual, Volume 3B, for legacy enumeration and interface details.

MBA performance properties change over time, for instance enhancing system-level efficiency. Software should not assume that performance properties or specific tunings of MBA remain identical across product generations. Third generation MBA shifts from interface-scope to agent-scope throttling support, and scheduler re-tuning to take advantage of this enhancement may be beneficial. Legacy architectural implementations of MBA are enumerated in the sequence described in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, in order to avoid ambiguous situations.

The MBA feature provides the following architectural components:

- A mechanism to enumerate the MBA capability to control the bandwidth from each level of the cache (for example, L2, L3) to the next level.

- A mechanism for the OS or Hypervisor to configure the amount of bandwidth available to a particular Class of Service via a throttling value (discussed later).
- Mechanisms for the OS or Hypervisor to specify the Class of Service to which a thread belongs.
- Hardware mechanisms to guide and enforce the delay value at each level of the cache hierarchy when an application has been designated to belong to a specific Class of Service.

Note that in some usages such as those seeking bandwidth control in MB/s, MBA may require either application-level performance feedback or complimentary Memory Bandwidth Monitoring (MBM) to use in the most optimal way. Backward compatibility of the software interfaces is preserved, and enhanced MBA generational changes manifest as enhancements atop the MBA feature baseline.

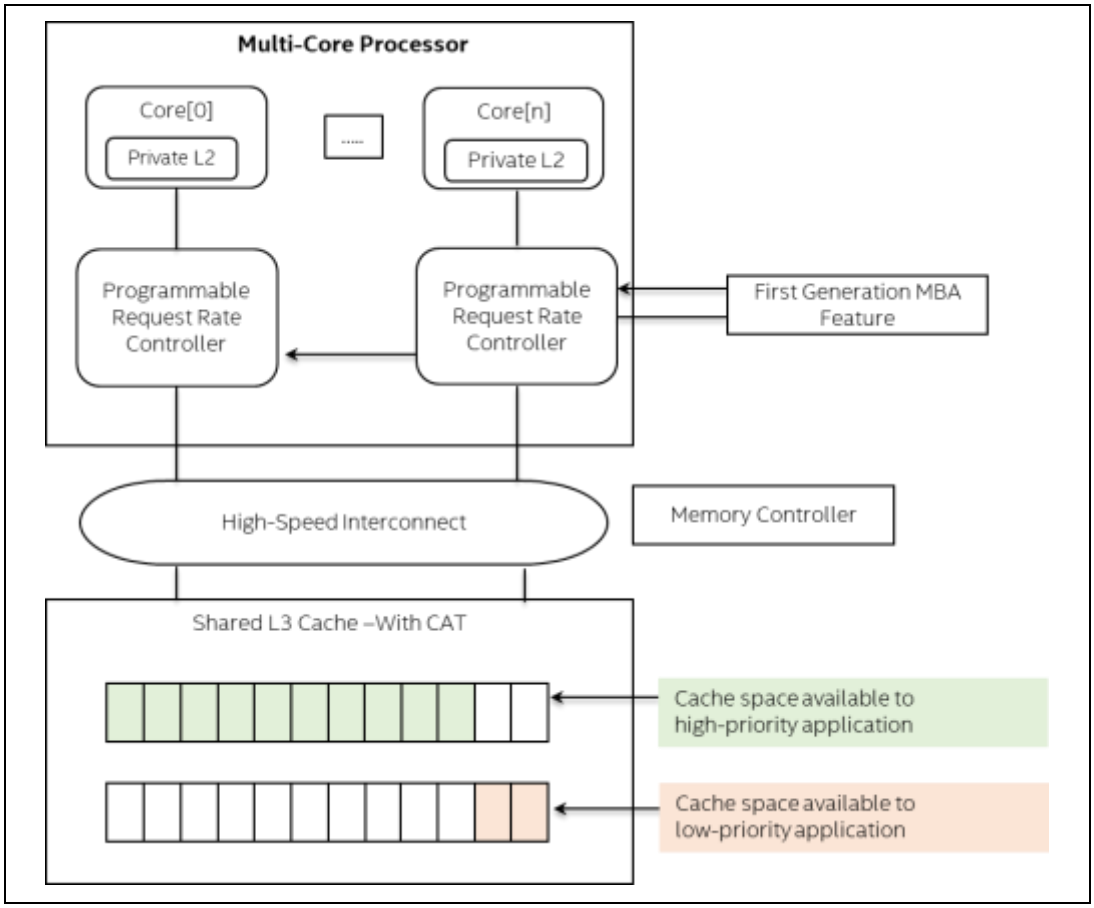
3.2.3.1 First Generation Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory band-width available per-core and was introduced on the Intel® Xeon® Scalable Processor Family. This feature provides a method to control applications which may be over-utilizing bandwidth relative to their priority in environments such as the datacenter.

The MBA feature uses existing constructs from the Intel RDT feature set including Classes of Service (CLOS). A given CLOS used for L3 CAT for instance means the same thing as a CLOS used for MBA. Infrastructure such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H [Cache Allocation Technology Enumeration Leaf]) are shared.

The high-level implementation of Memory Bandwidth Allocation is shown in Figure 3-6.

Figure 3-6. A High-Level Overview of the First-Generation MBA Feature



As shown here, the MBA feature introduces a programmable request rate controller between the cores and the high-speed interconnect, enabling indirect control over memory bandwidth for cores over-utilizing bandwidth relative to their priority. For instance, high-priority cores may be run un-throttled, but lower priority cores generating an excessive amount of traffic may be throttled to enable more bandwidth availability for the high-priority cores.

Because the MBA uses a programmable rate controller between the cores and the interconnect, higher-level shared caches and memory controller, bandwidth to these caches may also be reduced, so care should be taken to throttle only bandwidth-intense applications which do not use the off-core caches effectively.

The throttling values exposed by MBA are approximate and are calibrated to specific traffic patterns. As workload characteristics vary, the throttling values provided may affect each workload differently. In cases where precise control is needed, the Memory Bandwidth Monitoring (MBM) feature can be used as input to a software controller which makes decisions about the MBA throttling level to apply.

Legacy enumeration and configuration details are discussed in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

3.2.3.1.1 Usage Considerations

As the memory bandwidth control that MBA provides is indirect and approximate, using the feature with a closed-loop controller to also monitor memory bandwidth and how effectively the applications use the cache (via the Cache Monitoring Technology feature) may provide additional value. This approach also allows administrators to provide a bandwidth target or set point which a controller could use to guide MBA throttling values applied, and this allows bandwidth control independent of the execution characteristics of the application.

As control is provided per processor core (the max of the delay values of the per-thread CLOS applied to the core), the user should take care in scheduling threads so as to not inadvertently place a high-priority thread (with zero intended MBA throttling) next to a low-priority thread (with MBA throttling intended), which would lead to inadvertent throttling of the high-priority thread, as the maximum resolved throttling value is applied.

3.2.3.2 Second Generation Memory Bandwidth Allocation

The second generation of Memory Bandwidth Allocation (MBA) is implemented in the 3rd Gen Intel® Xeon® Scalable Processor Family, and related Intel Atom® processors such as the P5000 Series. This enhanced MBA capability provides improved efficiency and accuracy in throttling, along with providing increased system throughput. Rather than a strict bandwidth control mechanism, a dynamic hardware controller is implemented, which can react to changing bandwidth conditions at the microsecond level.

Before using the second-generation MBA feature, the MBA hardware controller requires a BIOS-assisted calibration process that may include inputs such as the number of memory channels populated and other system parameters; this is a change from the first generation of MBA. Intel BIOS reference code includes a default configuration that is recommended for general usage, and BIOS profiles may be created with alternate tuning values to optimize for certain usages (such as stricter throttling) as described in the subsequent BIOS Considerations chapter.

Second generation MBA moves from static throttling at the core/uncore interface, to a more dynamic control method based on a hardware controller that tracks actual main memory bandwidth. This allows software that uses primarily the L3 cache to observe increased throughput for a given throttling level, or fine-grained throughput benefits for software that exhibits L3-bound phases. Due to the closer consideration of memory bandwidth loading, this enhancement may lead to an increase in system efficiency when using second generation MBA relative to prior implementations of the feature. Backward compatibility of the software interfaces is preserved, and second-generation MBA changes manifest as enhancements atop the MBA feature baseline.

As with the prior generation feature, the second generation MBA uses CPUID for enumeration and throttling is performed using a mapping created from software thread-to-CLOS (in the IA32_PQR_ASSOC MSR), which is then mapped per-CLOS to delay values via the IA32_L2_QoS_Ext_BW_Thrtl_n

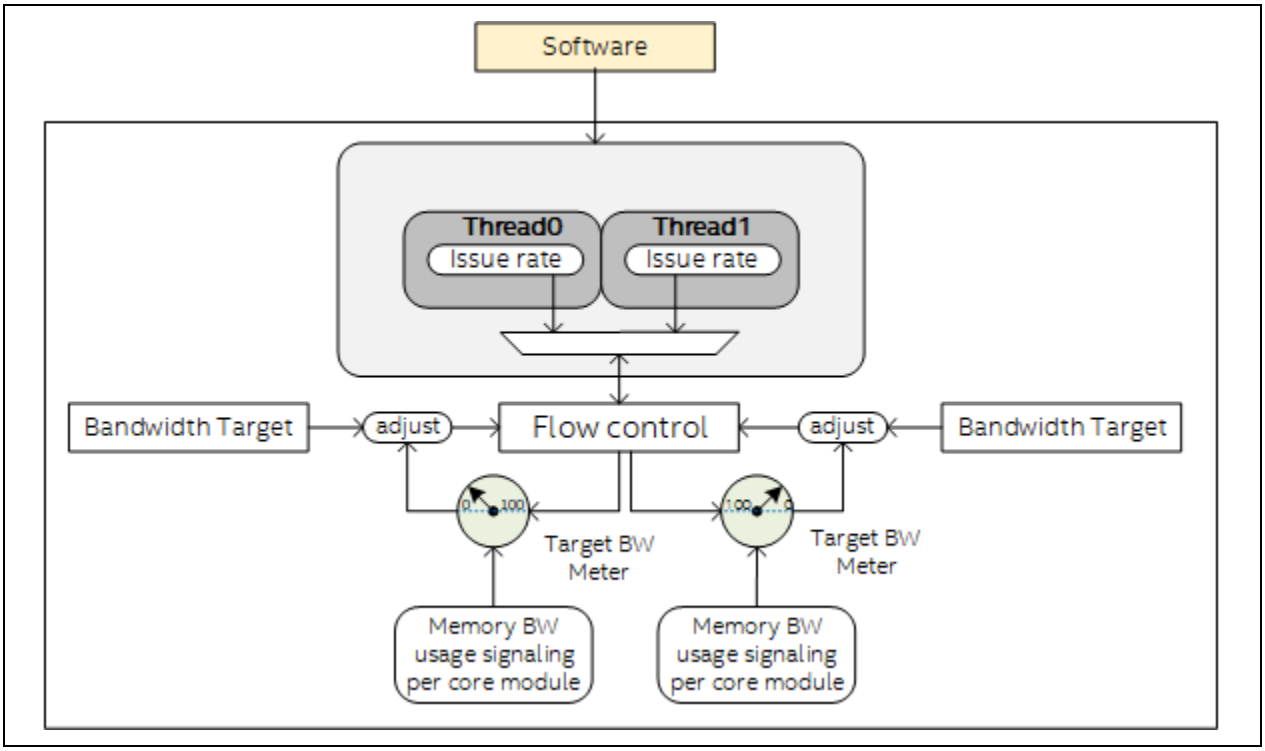
MSRs. A privileged operating system or virtual machine manager software may specify a per-CLOS delay value, 0-90% bandwidth throttling for instance, though the max and granularity values are platform dependent and enumerated in CPUID.

3.2.3.2.1 Second Generation MBA Advantages

Additional features added over first generation MBA are described next:

1. Previously, only the maximum delay value across two CLOS on a physical core could be selected in MBA. Second generation MBA allows a minimum delay value to be selected instead, which may enhance usage with Intel® Hyper-Threading Technology.
2. Only a single preprogrammed calibration table was possible in first generation MBA, meaning different memory configurations had the potential for different linearity and percent delay value error values depending on the configuration. This is addressed by the BIOS support in the second generation of MBA, and certain BIOS implementations may program a different calibration table per memory configuration, for instance.
3. The second-generation MBA controller provides the ability to more closely monitor the memory bandwidth loading and deliver more optimal results.
4. The new MBA hardware controller reduces the need for a fine-grained software controller to manage application phases for optimal efficiency. Note that a software controller may still be valuable to translate MBA throttling values to bandwidths in GB/s or application Service Level Objectives (SLOs), such as performance targets.

Figure 3-7. Second Generation MBA, Including a Fast-Responding Hardware Controller



The second-generation MBA implementation is shown in Figure 3-7. The feature operates through the use of an advanced hardware controller and feedback mechanism, which allows automated hardware monitoring and control around the user-provided delay value set point. This set point and associated throttling value infrastructure remains unchanged from prior generation MBA, preserving software compatibility.

MBA enhancements, in addition to the new hardware controller, include:

1. Configurable delay selection across threads.
   - MBA 1.0 implementation statically picks the max MBA Throttling Level (MBAThrotLvl) across the threads running on a core (by calculating value = max(MBAThrotLvl(CLOS[thread0]), MBAThrotLvl(CLOS[thread1]))).
   - Software may have the option to pick either maximum or minimum delay to be resolved and applied across the threads; maximum value remains the default.
2. Increasing CLOSIDs from 8 to 15 in certain implementations (product-specific, see CPUID)
   - Previous certain implementations of the feature provided 8 CLOS tags for MBA.
   - The 3rd Gen Intel® Xeon® Scalable Processor Family and related Intel Atom® processors, such as the P5000 Series, increase this value to 15 (also consistent with L3 CAT).

3.2.3.2.2 Software-Visible Changes

A new model specific MSR is introduced with second generation MBA to allow software to select from the maximum (default) or minimum of resolved throttling values (see the previous formula). This capability is controlled via a bit in the new MBA_CFG MSR, shown in Table 3-1.

Table 3-1. MBA_CFG MSR Definition

Register Address		Architectural MSR Name / Bit Fields	Description
Hex	Decimal		
C84H	3204	MBA_CFG	MBA Configuration Register
		0	Min (1) or max (0) of per-thread MBA delays.
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).

Note that bit[0] for min/max configuration is supported in second generation MBA but is removed in the third generation MBA when the controller logic becomes capable of managing throttling values on a per-logical-processor or per-agent basis. The transient nature of this enhancement is why the min/max control remains model specific.

To enumerate and manage support for the model-specific min/max feature, software may use processor family/model/stepping to match supported products, then CPUID to later detect enhanced third generation MBA support.

3.2.3.3 Third Generation Memory Bandwidth Allocation

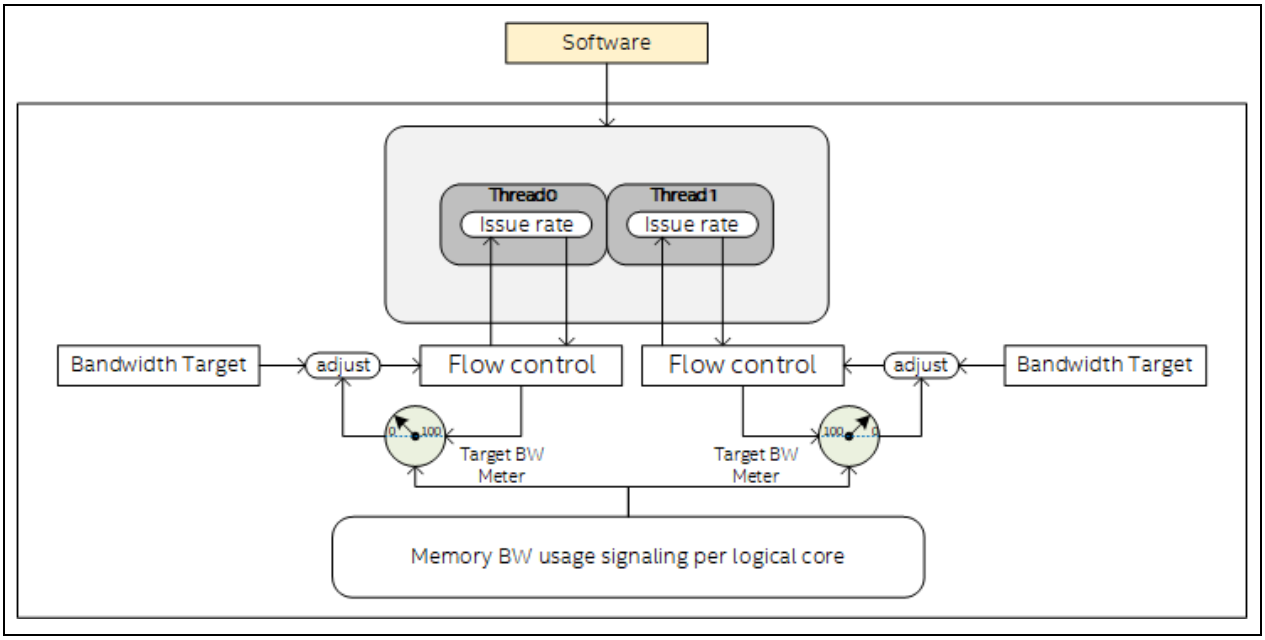
The third-generation MBA feature on future processors based on the codename Granite Rapids microarchitecture further enhances the feature with per-logical-processor control and a further improved controller design. Total memory bandwidth (all L3 miss traffic) is now managed by MBA.

This implementation follows the past MBA precedent of delivering significant enhancements without a major software overhaul, and while preserving backward compatibility.

3.2.3.3.1 Hardware Changes

The third generation of MBA builds upon the hardware controller introduced in the previous generation, which enabled significant system-level benefits, while providing the new capability to independently throttle logical processors, rather than more coarse-grained per-core throttling in prior generations. Throttling values are no longer selected as the “min” or “max” of the two throttling values for the threads running on the core; instead, throttling values are independently and directly applied to each logical processor. The third generation MBA implementation is shown in Figure 3-8.

Figure 3-8. High-Level Overview of the Third Generation MBA Feature



While this enhancement means that more direct throttling of threads is possible, re-tuning of software may be helpful to comprehend the effects of Intel® Hyper-Threading Technology contention versus cache and memory contention, and the effects on software performance.

3.2.3.3.2 Software-Visible Changes

In order to allow software to change its tuning behavior and detect that per-logical-processor throttling is supported on a particular product generation, a CPUID bit is added to the MBA CPUID leaf to indicate support. See "CPUID—CPU Identification" in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B for details.

Despite another significant improvement of the hardware controller infrastructure architecture and improved capabilities, controller responsiveness, new internal microarchitecture, and transient-arresting capabilities, no new software interface changes are required to make use of the third generation of MBA relative to prior generations. Software previously using the second-generation MBA min/max selection capability should discontinue the use of the MBA_CFG MSR. The third-generation MBA capabilities are the default mode of operation on the codename Granite Rapids server microarchitecture.

Note that the MBA MSRs are listed in Appendix A.5 for completeness, but details of these legacy MSRs are available in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. See Appendix A.2 for MBA feature supported product details.

3.2.4 Cache Bandwidth Allocation

Cache Bandwidth Allocation (CBA) allows an Operating System, Hypervisor, or similar system management agent to control internal core and downstream memory bandwidth for each of the logical processors. This feature is complimentary to MBA and provides OS/VMMs with the ability to throttle threads within the core.

The CBA feature along with the existing MBA feature provides a system-wide mechanism to throttle the bandwidth across different caches in the system including external memory, as well as control within a processor core or module. In combination, CBA and MBA provide both deterministic control and dynamic management of bandwidth resources to meet system Service Level Objectives (SLOs). The CBA feature reuses and extends existing constructs from the Intel RDT feature set including Classes of Service (CLOS).

A given CLOS used for L3 CAT for instance means the same thing as a CLOS used for CBA. Infrastructure such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H (Cache Allocation Technology Enumeration Leaf)) are shared.

The Cache Bandwidth Allocation (CBA) feature provides control over bandwidth available between Level 1 (L1) caches, Level 2 (L2) Caches, and Level 3 (L3) Caches (as applicable) for each of the logical processors. Since reducing upstream bandwidth can also reduce bandwidth to external memory, this also provides an indirect control of memory bandwidth. This indirect control of external memory bandwidth can also reduce memory bandwidth. The CBA feature along with the MBA provides a mechanism to control the bandwidth of different applications.

Software should understand that the effective throttling of an application may be whichever of CBA or MBA specifies more throttling.

Similar to Intel RDT features, CBA includes the following key ingredients:

- A mechanism to enumerate the CBA capability to control the bandwidth from each level of the cache (for example, L1, L2, L3) to the next level (CPUID).
- A mechanism for the OS or Hypervisor to configure the amount of bandwidth available to a logical processor with a particular Class of Service via a throttle Level (MSRs, discussed later).
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs (the PQR MSR).
- Hardware mechanisms to guide and enforce the bandwidth throttle level across the cache hierarchy.

In some usages, the software may measure the memory bandwidth consumed by a given thread, application, VM or container at different Levels of cache hierarchy and external memory using performance monitor events and Memory Bandwidth Monitoring (MBM). Once the memory bandwidth is measured software can dynamically adjust the bandwidth throttling level for the class of

service (CLOS) used by that application. In other usages, software control loops may monitor application performance and adjust throttling levels dynamically to achieve certain performance targets.

More specific feature details about CBA are provided in the Intel® Architecture Instruction Set Extensions and Future Features. Note that the MSRs are listed in Appendix A.5. See Appendix A.2 for CBA feature supported product details.

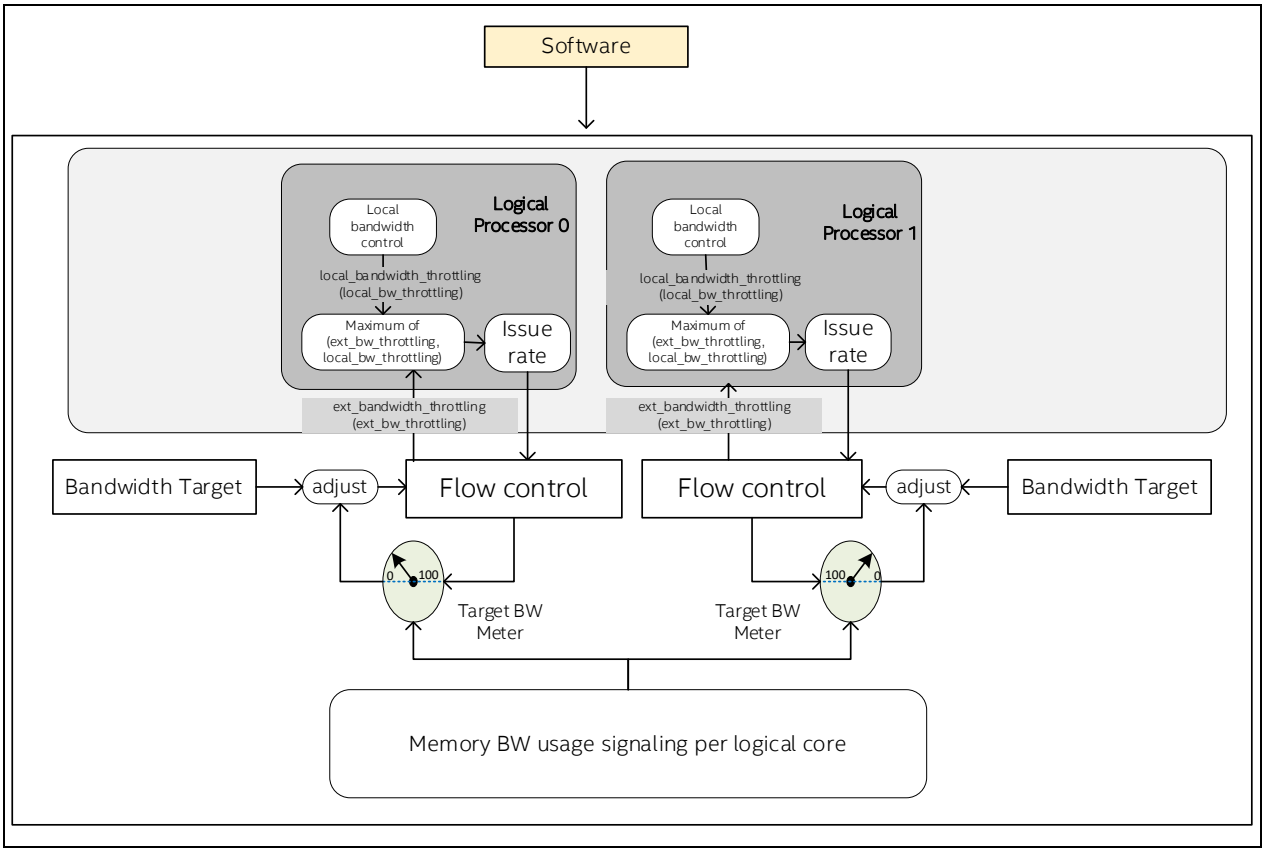
3.2.4.1 CBA Overview

The CBA feature implements a local hardware controller which when enabled provides the capability to independently throttle memory bandwidth of the logical processors across cache hierarchy and complements the MBA controller which throttles the external memory bandwidth.

3.2.4.2 Example of CBA Throttling Mechanism

An example of the bandwidth throttling enforced between L2 cache and L3 cache is the maximum of the bandwidth throttling from the local CBA controller within the logical processor and the MBA hardware controller. An example of CBA implementation is shown in Figure 3-9.

Figure 3-9. Example of CBA Throttling between L2 and L3 caches



3.2.4.3 Software Interface

In order to allow software to adapt its tuning behavior and detect that cache bandwidth throttling is supported on a particular product generation, a CPUID bit is added to the Intel RDT A CPUID leaf to indicate support (details are provided in the Intel® Architecture Instruction Set Extensions Manual).

The IA32_PQR_ASSOC MSR specifies the Class of Service associated with each logical processor. The CBA feature defines a set of MSRs known as IA32_QoS_Core_BW_Thrtl_n which provide a byte-encoded field for each CLOS to program the memory bandwidth throttle level. A higher value of throttling level means more bandwidth throttling and lower number indicates lesser throttling. The CPUID of the CBA feature enumerates the number of levels and maximum level supported by the logical processor. The reset value of each of the CLOS throttle values of the logical processor is 0 which indicates unthrottled bandwidth.

Each of the fields in the CBA IA32_QoS_Core_BW_Thrtl_n MSRs may be programmed from 0 to the maximum throttle level provided in the CPUID. If a value beyond the range from 0 to maximum throttle level is programmed, it will cause a #GP fault. The Resource Management Domain (RMD) for CBA is per logical processor and thus the IA32_QoS_Core_BW_Thrl_n MSRs are logical processor scope. Further details are provided in the Intel® Architecture Instruction Set Extensions and Future Features Programming reference manual.

3.2.4.4 Software Usage

The next sequence of steps provides a typical software usage of CBA feature:

1. System is setup with the desired workloads.
2. The software can use the performance counters along with MBM counters when available to profile and understand the bandwidth characteristics of the application.
3. The system administrator sets up the bandwidth throttling level field in the IA32_QoS_Core_BW_Thrtl_n MSR (for example, in the VMM) to enforce the desired limits and the CLOS for each application. They can monitors the bandwidth to confirm the setting is appropriate and adjust when needed.

In some cases, a specialized application software such as in embedded or communications usages will be able to communicate the memory bandwidth and latency requirements. This information may be used be performance management software to program the RDT features including CBA to meet the software memory bandwidth and latency requirements.

4 Intel® Resource Director Technology for Non-CPU Agents

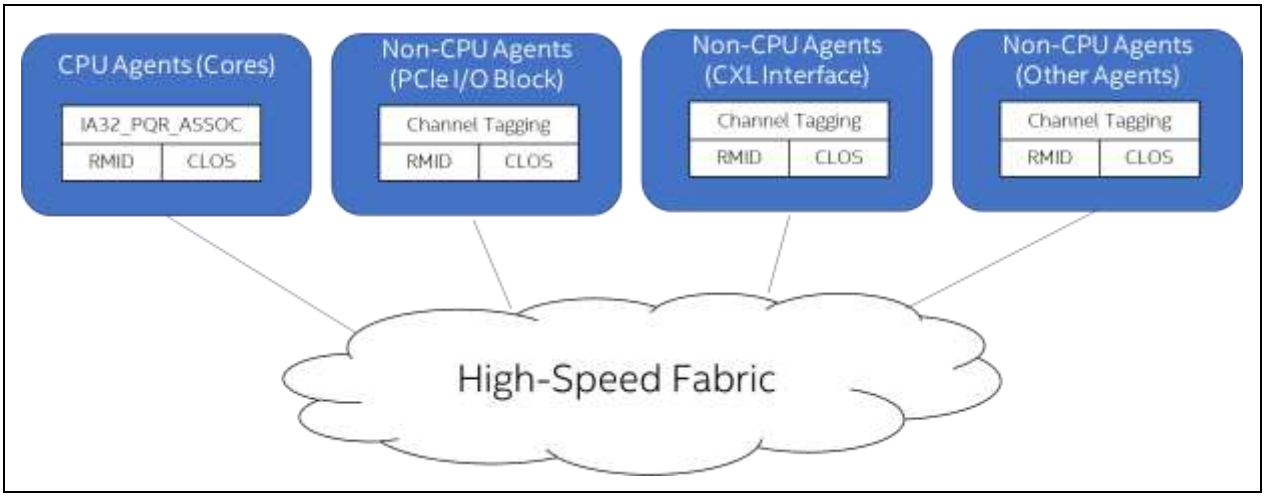
This chapter details Intel RDT features for non-CPU agents. Discussion is included on use cases and how Intel RDT monitoring, and controls are provided for non-CPU agents through extensions to the foundational CPU Agent Intel RDT features. Chapter 3 describes the components of the Intel RDT feature set which are common.

4.1 Introduction

Intel RDT for non-CPU agents is a set of features to monitor and control the resource utilization of non-CPU agents including PCI Express* (PCIe*) [5] and Compute Express Link (CXL)* [6] devices and integrated accelerators. The feature set enables monitoring usage of shared cache and memory bandwidth and control of cache usage by non-CPU agents. This feature set provides the equivalent CPU agent Intel RDT capabilities of CMT, MBM, and CAT for I/O devices.

The non-CPU agent Intel RDT includes controls at the device level and channel-level granularity in some cases. However, this granularity is fundamentally coarser than for software threads. CPU cores may execute hundreds of threads, all of which are tagged with RMIDs and CLOS, whereas an I/O device such as a NIC may serve hundreds of software threads, but it may only be monitored and controlled at a device level or channel level (see subsequent sections for details on channel-level monitoring and controls.)

Figure 4-1. Non-CPU Agent Building Atop CPU Agent Intel® RDT Features



4.2 Features

Cache Monitoring Technology (CMT) provides visibility into the cache (typically L3 cache). CMT provides occupancy counters on a per-RMID basis for non-CPU agents so cache occupancy (for example, capacity used by a particular RMID for I/O agents) can be tracked and read back dynamically during system operation. See Appendix A.2 for L3 CMT feature supported product details.

L3 Total and Local External Memory Bandwidth Monitoring (MBM) allows system software to monitor the usage of bandwidth between L3 cache and local or remote memory by non-CPU agents on a per-RMID basis. See Appendix A.2 for L3 Total and Local External MBM feature supported product details.

Cache Allocation Technology (CAT) allows control over shared cache capacity on a per-CLOS basis for non-CPU agents, enabling both isolation and overlap for better throughput, fairness, determinism and differentiation. See Appendix A.2 for L3 CAT feature supported product details.

4.3 Enumeration

Intel RDT uses the CPUID instruction to enumerate supported features and uses architectural Model-Specific Registers (MSRs) as interfaces to the monitoring and allocation features as described in Chapter 3 and in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

There are no CPUID leaves or sub-leaves that are created for non-CPU agent Intel RDT; rather, existing CPUID leaves are augmented. CPUID.0xF(Shared Resource Monitoring Enumeration leaf).[ResID=1]:EAX[bit 9,10] enumerates presence of CMT and MBM features for non-CPU agents. CPUID.0x10(Cache Allocation Technology Enumeration Leaf).[ResID=1(L3 CAT)]:ECX[bit 1] enumerates the presence of the L3 CAT feature for non-CPU agents. Refer to Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for CPUID details.

Additional enumeration information for Intel RDT for non-CPU agents is provided in the I/O Intel RDT table (IRDT), a vendor-specific extension to Advanced Configuration and Power Interface (ACPI) [4]. The IRDT table provides information on supported features, the structure of devices attached to particular links behind I/O blocks, the forms of tagging and controls supported on each link, and the specific MMIO interfaces used to control a given device. Details of IRDT are described in Chapter 5.

Confirming the presence of Intel RDT for CPU agents is a prerequisite for using the equivalent non-CPU agent Intel RDT feature. A compatibility matrix is provided in Appendix A.4. If a particular CPU agent Intel RDT feature is not present, any attempt to use non-CPU agent Intel RDT equivalents will result in a general protection fault in the MSR interface. Attempts to enable unsupported features in the I/O complex will result in writes to the corresponding MMIO enable or configuration interfaces being ignored.

Software may use the existing CPUID leaves to gather the maximum number of RMID and CLOS tags for each resource level (for example, L3 cache), and non-CPU agent Intel RDT is also subject to these limits.

Some platforms may support a mix of features, for instance supporting L3 CAT and the non-CPU agent Intel RDT equivalent, but no CMT or MBM monitoring.

4.4 Interface

Before configuring non-CPU agent Intel RDT (through MMIO), the feature should be enabled. The presence of one or more CPUID bits indicating support for one or more non-CPU agent Intel RDT features implies the presence of the IA32_L3_IO_RDT_CFG architectural MSR. This MSR is used to enable the non-CPU agent Intel RDT features.

Two bits are defined in this MSR. IRAE (Bit[0]) enables non-CPU agent RDT resource allocation features. IRME (Bit[1]) enables non-CPU agent RDT monitoring features.

The non-CPU agent Intel RDT Monitoring bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource monitoring features are present.

The non-CPU agent Intel RDT Allocation bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource allocation features are present.

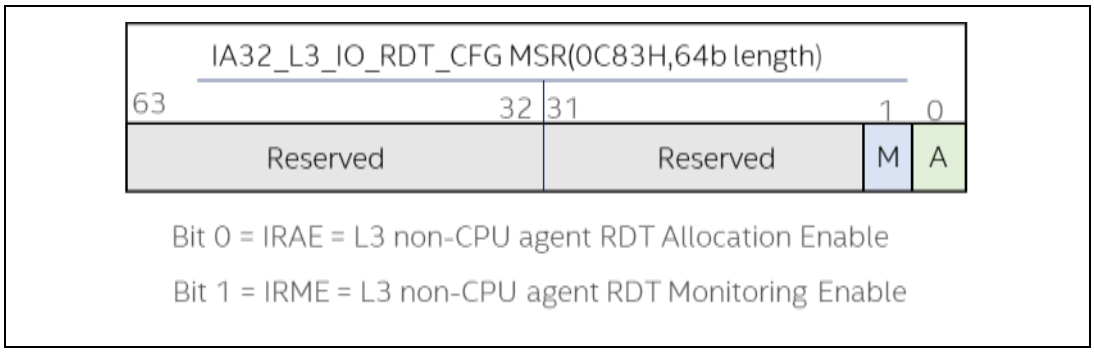
The default value is 0x0 (both the monitoring and allocation features are disabled by default). All bits not defined are reserved. Any writes to reserved bits will generate a General Protection Fault (#GP(0)).

This MSR is die-scoped and is cleared on system reset. It is expected that software will configure this MSR consistently across all L3 caches that may be present on a particular SOC die.

The definition of the IA32_L3_IO_RDT_CFG MSR is shown in Figure 4-2, and its MSR address is 0C83h.

Non-CPU agent RDT uses the RMID and CLOS tags in the same way that they are used for CPU agents.

Figure 4-2. The IA32_L3_IO_QOS_CFG MSR for Enabling Non-CPU Agent Intel® RDT



MMIO interfaces, discussed in subsequent sections, are defined by non-CPU agent Intel RDT to enable devices and/or channels to be tagged with RMIDs and CLOS, as applicable.

An example of device tagging with RMIDs, and CLOS is shown in Figure 4-3, where a PCIe device and a CXL device are tagged for monitoring and control of upstream resources in the L3 cache (shown within the fabric). Note that CPU cores are also shown, and as defined in the CPU agent Intel RDT feature set, their bandwidths may be controlled with the Memory Bandwidth Allocation (MBA) feature set.

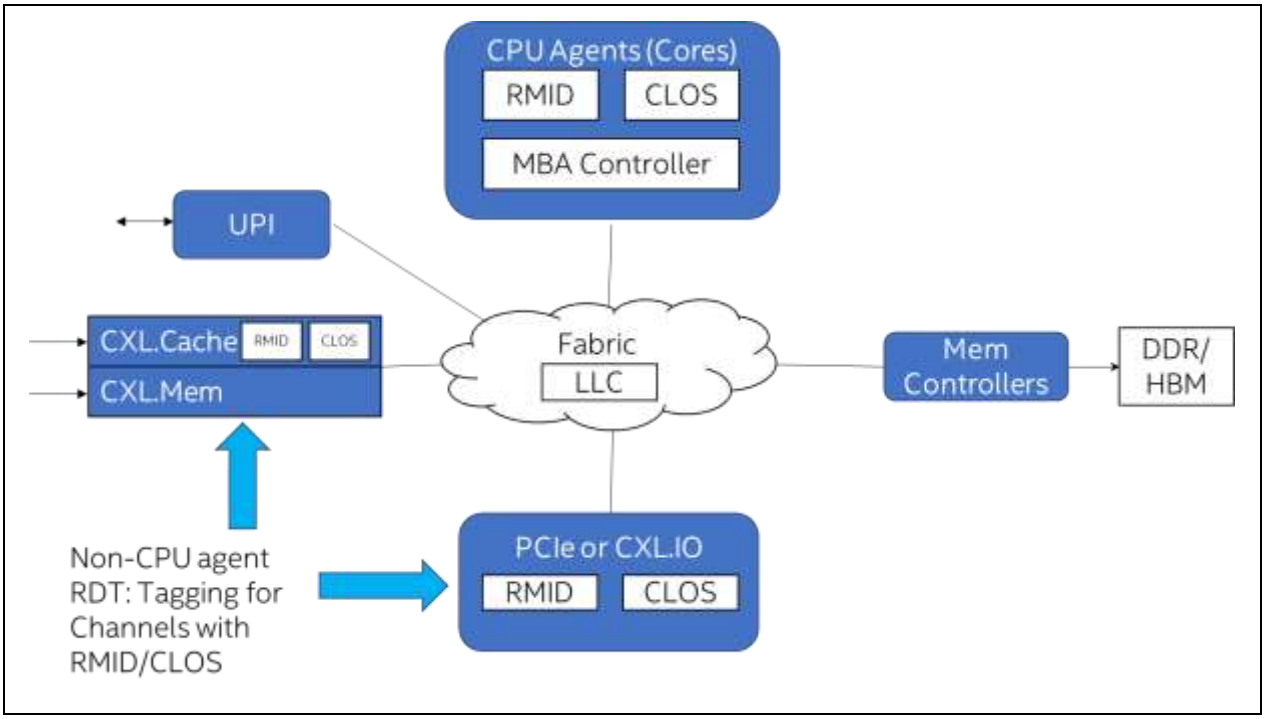
In the model of Figure 4-3, cores, PCIe devices and CXL devices are symmetrically arranged about the fabric and are symmetric in their ability to use RMIDs and CLOS.

The Intel RDT monitoring data retrieval MSRs IA32_QM_EVTSEL and IA32_QM_CTR are used for monitoring usage by non-CPU agents in the same way that they are used for Intel RDT for CPU agents.

The CPU cache capacity control MSR interfaces are also used for controlling I/O device access to the L3 cache. The CLOS assigned to the device and the corresponding capacity bitmask in the IA32_L3_QOS_MASK_n MSR governs the fraction of the L3 cache into which the data may be filled, as described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

The CLOS tag retains the same meaning with regard to L3 fills for both CPU agents and non-CPU agents. Other cache levels may also be applicable depending on model-specific data flow patterns, which are governed by how I/O device data is filled into the cache in a model-specific fashion as governed by a given product generation's implementation of the DDIO (the Data Direct I/O feature).

Figure 4-3. Tagging for PCIe and CXL Devices



4.5 Common Tags

Non-CPU agent Intel RDT allows the traffic and operation of non-CPU agents to be associated with RMIDs and CLOS. In CPU agent Intel RDT, RMIDs and CLOS are numeric tags which may be associated with the operation of a thread through the IA32_PQR_ASSOC MSR. In non-CPU agent Intel RDT, a series of MMIO interfaces may be used to associate I/O devices with RMID and CLOS tags, and the numerical interpretation of the tags remains the same.

To wit, a particular CLOS tag, such as CLOS[5], means the same thing from the perspective of an CPU core or a non-CPU agent, and the same holds for RMIDs. In this fashion, RMIDs and CLOS used for non-CPU agents are said to be drawn from a "common pool" of RMID or CLOS tags, defined at the common L3 configuration level. Often these tags have specific meanings at a particular level of resource such as the L3 cache.

With non-CPU agent Intel RDT, specific devices may be selected for monitoring and control, and software enumeration and control are added to (1) enable non-CPU agent Intel RDT to build atop CPU agent Intel RDT, and (2) to comprehend the topology of devices behind I/O links, such as PCIe or CXL, and (3) to enable association of devices with RMID and CLOS tags.

4.6 I/O Blocks and Channels

I/O interfacing blocks are used to bridge from the ordered, non-coherent domain (such as PCIe) to the unordered, coherent domain (for example, the

shared interconnect fabric hosting the L3 cache). The non-CPU agent Intel RDT interface describes the devices connected behind each I/O complex (which may contain downstream PCIe root ports or CXL links) and enables configuration RMID/CLOS tagging for the same.

The I/O architecture is formalized as shown next. Channel mapping may occur anywhere between the device and the I/O block.

Figure 4-4. Mapping of Channels in the I/O Domain (PCIe Example)

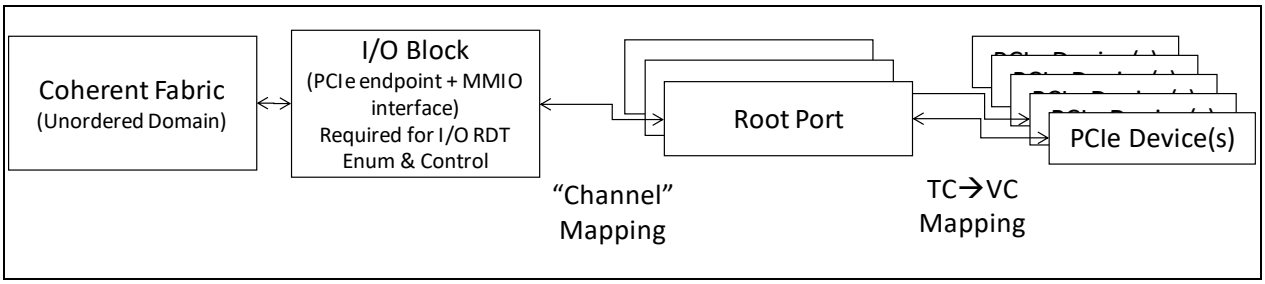
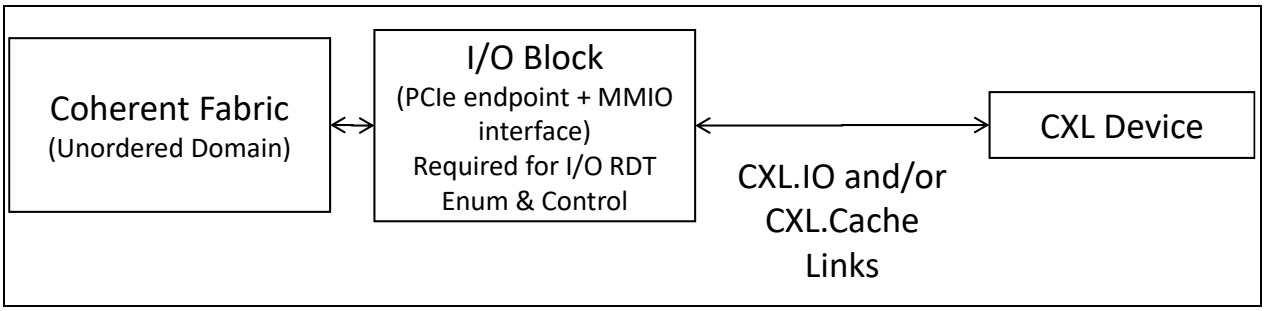


Figure 4-5. Mapping of Channels in the I/O Domain (CXL Example)



As shown in Figure 4-4, PCIe devices connected through a root port are routed through an I/O block, which applies non-CPU agent Intel RDT tagging (RMID and CLOS tagging) before traffic reaches the coherent fabric. Device traffic which is routed on various TCs and mapped to VCs, as defined in the PCIe specification [5], may be mapped to internal "Channels" between the root port and the I/O block. The non-CPU agent Intel RDT enumeration structures define the mapping between PCIe VCs and the non-CPU agent Intel RDT Channels so that software may perform tagging configuration based on Channels for platforms which support this capability (see the following sections for more detail).

An example with CXL [6] is shown in Figure 4-5. In this case a CXL.IO and CXL.Cache link may be in use, and the I/O block is again responsible for tagging, if supported. The links (CXL.IO and CXL.Cache) are controlled separately, through separate software interfaces. (See Chapter 7 for MMIO control interfaces.)

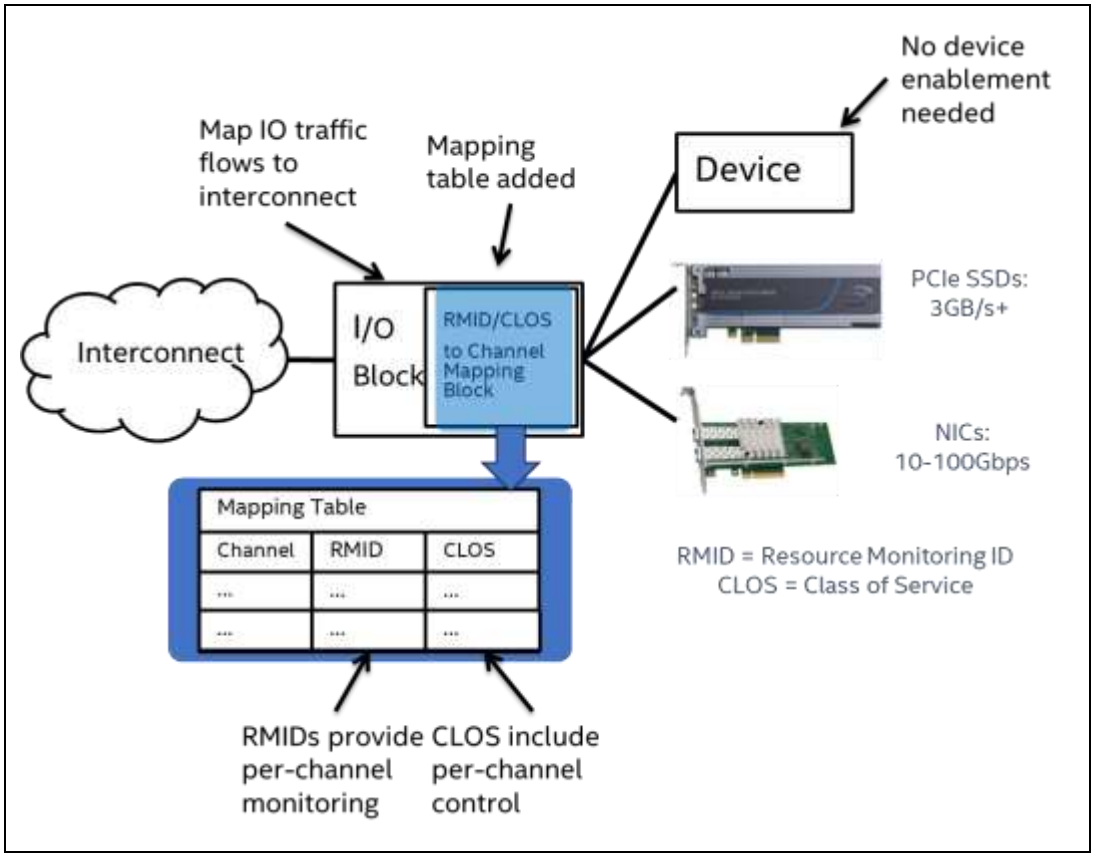
4.7 I/O Block Configuration

As described in the preceding section, PCIe devices mapped through their VCs to "Channels" may be configured on a per-Channel basis in the I/O Block. CXL

is a subset example of this, with the same configuration format, but only one configuration entry (the equivalent of a single Channel).

An enumerated number of Channels are supported in IRDT ACPI and configured through an MMIO interface to a "Mapping Table", as shown in Figure 4-6. A number of downstream PCIe devices may be mapped to various channels, and their traffic streams may be tagged, as applicable, through configuration of the I/O block.

Figure 4-6. Resource Monitoring and Control for PCIe and CXL Endpoints



4.8 Shared-L3 Configuration

The following sub-sections describe shared-L3 configuration and non-CPU agent Intel RDT features interplay.

4.8.1 Software Flow

Key software actions required to utilize non-CPU agent Intel RDT include (1) enumeration of the supported capabilities and details of that support, and (2) usage of the features through architectural platform interfaces.

- The software may enumerate the presence of non-CPU agent Intel RDT through a combination of parsing bit fields from CPUID and the IRDT ACPI table. The CPUID infrastructure provides basic information on the level of

CPU agent Intel RDT and non-CPU agent Intel RDT support present, and details of the common CLOS/RMID tags shared with CPU agent Intel RDT. The IRDT ACPI extensions provide many more details on non-CPU agent RDT specifically, such as which I/O blocks support non-CPU agent Intel RDT and where the control interfaces to configure the I/O blocks are located in MMIO space.

- Once software has enumerated the presence of non-CPU agent Intel RDT, configuration changes may be made through selecting a subset of RMID/CLOS tags to use with non-CPU agent Intel RDT, and configuring resource limits for those tags through MSRs for shared platform resources such as L3 cache (for example, for I/O use of L3 CAT) may be configured through the I/O block MMIO interfaces (the location of which is enumerated via IRDT ACPI).
- After resource limits are associated, RMID/CLOS tagging may be applied to the I/O device upstream traffic by assigning each I/O device into RMID/CLOS tags through its mapping to channels (and corresponding configuration through the MMIO interfaces for each I/O block, the location of which is enumerated via IRDT ACPI).
- It should be noted that while upstream shared SoC resources like L3 cache are monitored and controlled via shared RMID/CLOS tags, certain resources which are closer to the I/O may be controlled locally within each I/O block. In this view, RMIDs and CLOS are used for upstream resources which may be shared with CPU cores, but capabilities unique to the I/O device domain are controlled through I/O block-specific interfaces.
- Once tags are assigned and resource limits are applied, upstream traffic from I/O devices, though I/O blocks are tagged with the corresponding RMIDs/CLOS and such traffic is monitored and controlled within the shared resources of the SoC, much as CPU agent resources are controlled against these tags in CPU agent Intel RDT.
- As the IRDT ACPI tables used to enumerate non-CPU agent Intel RDT are generated by the BIOS, in the event of a hot-plug operation the OS or VMM software should update its internal tracking of device mappings based on newly added or removed device.
- In the case of bifurcation of a set of PCIe lanes, downstream devices which may be mapped to individual Channels may still be separately tagged and controlled, but devices sharing Channels will be mapped together against the same RMID/CLOS tags. As CXL devices have no notion of Channels, in the case of a bifurcated CXL link all downstream devices will be subject to the same RMID/CLOS.

4.8.2 Monitoring: Data Flows for RMIDs

As previously described, once RMID tags are applied to non-CPU agent traffic, all RMID-driven counter infrastructure in the platform may be used with non-CPU agent Intel RDT. In the case of the features in Appendix A.2 for instance, RMID-based cache occupancy and memory bandwidth overflow data is collected for non-CPU agents and may be retrieved by software. For each supported Cache Monitoring resource type, hardware supports only a finite

number of RMIDs. CPUID.(EAX=0FH(Shared Resource Monitoring Enumeration leaf), ECX=1H). ECX enumerates the highest RMID value that can be monitored with this resource type, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B for details.

As the interfaces for CPU agent Intel RDT data retrieval for RMID-based counters area already defined, the same interfaces are used, including MSR-based data retrieval for the corresponding set of three Event IDs (EvtIDs) defined for CPU agent Intel RDT's CMT and MBM features (See Chapter 3).

RMIDs are allocated to devices by software from the pool of RMIDs defined at the L3 cache level, and the IA32_QM_EVTSEL / IA32_QM_CTR MSRs can be used to specify RMIDs and Event IDs and retrieve data.

The MSR pair used to retrieve event data is shown in Figure 3-3, however as all properties are inherited from CPU agent RDT (See Chapter 3 for details) . All of access rules and usage sequence, reserved bit properties, initial values, and virtualization properties are inherited from CPU agent Intel RDT.

4.8.3 Allocation: CLOS-based Control Interfaces

The Intel RDT Allocation features for non-CPU agent use CLOS-based tagging for control of cache at a given level, subject to where data fills from I/O devices in a particular cache and SoC implementation. In common cases this will be the last-level cache (L3) as described in the ACPI – specifically in the IRDT sub-table known as RCS and its flags. Software may adjust the levels of cache that it controls based on the expected level(s) of cache into which I/O data may fill subject to flags in the RCS. This in turn may affect which CPU agent CAT control masks software programs to control the data fills of non-CPU agents and may vary depending on how a particular RCS is connected to shared resources on a platform.

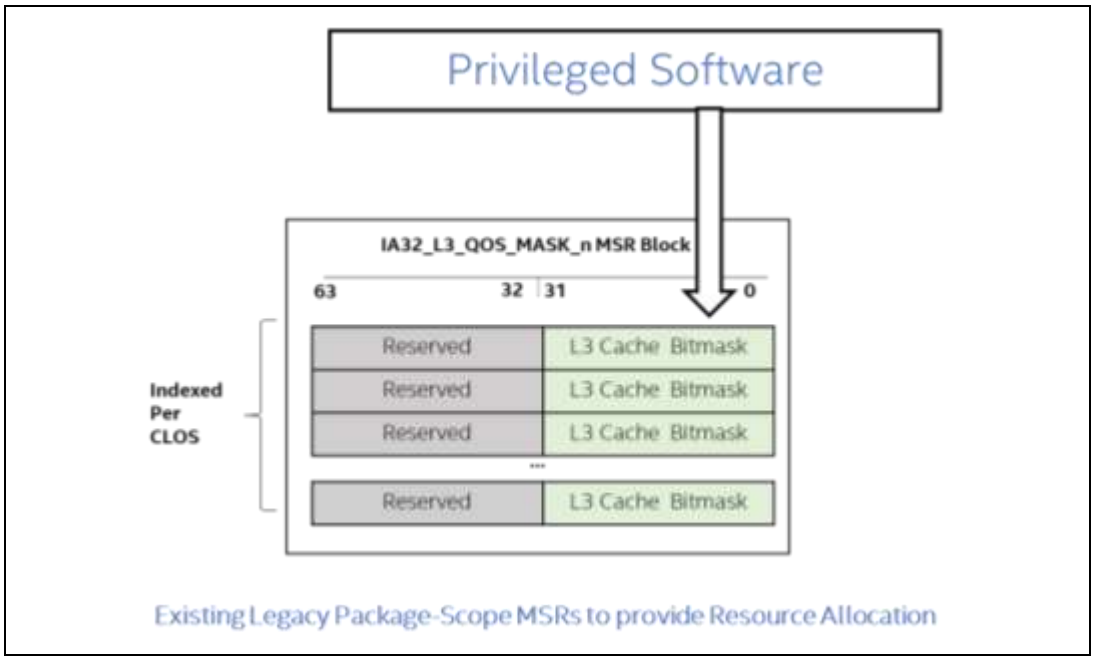
For each supported Cache Allocation resource type, the hardware supports only a finite number of CLOS. CPUID.(EAX=10H(Cache Allocation Technology Enumeration Leaf), ECX=2):EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of "15" would indicate 16 total supported CLOS). Bits 31:16 are reserved, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B for details.

In a typical example, with a non-CPU agent (for example, a PCIe device) filling data into an L3 cache, the RCS structure's "Cache Level Bit Vector" would have bit 17 set to indicate the L3 cache, and software may control the CPU agent Intel RDT L3 CAT masks (in IA32_L3_QoS_MASK_n MSRs) to define the amount of cache into which non-CPU agents may fill. As with RMID management, the CLOS used in this context are drawn from the pool at the applicable resource (L3 cache in this context).

If other cache levels are introduced or used in the future, incremental software enabling may be required to comprehend fills into other cache levels.

As the masks used for control are drawn from the existing definitions of such cache controls in the CPU agent Intel RDT definitions, details such as reserved fields, initialization values, and so on, are defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. Figure 4-7 shows an example of the CPU agent Intel RDT L3 CAT control MSRs.

Figure 4-7. Reuse of the IA32_L3_QOS_MASK_n MSRs for L3 CAT Control



4.9 CXL-Specific Considerations

This section describes CXL-specific device considerations including management of traffic on multiple links and CXL device types.

4.9.1 CXL block Interfacing Fundamentals

CXL devices may connect to an RMUD via multiple RCSes, and independent control of each RCS may be required. See Chapter 5 for RMUD and RCS details.

Non-CPU agent Intel RDT features provide monitoring and controls for CXL.IO and CXL.Cache link types. CXL.mem is not subject to controls in the I/O block as it is viewed as a resource rather than an agent in Intel RDT terms. Instead bandwidth to CXL.mem is controlled at the agent source (for example, using MBA) as previously described and where supported.

4.9.2 Integrated Accelerators

Integrated accelerators, including those using integrated CXL links, may be monitored and controlled using the semantics described in preceding sections.

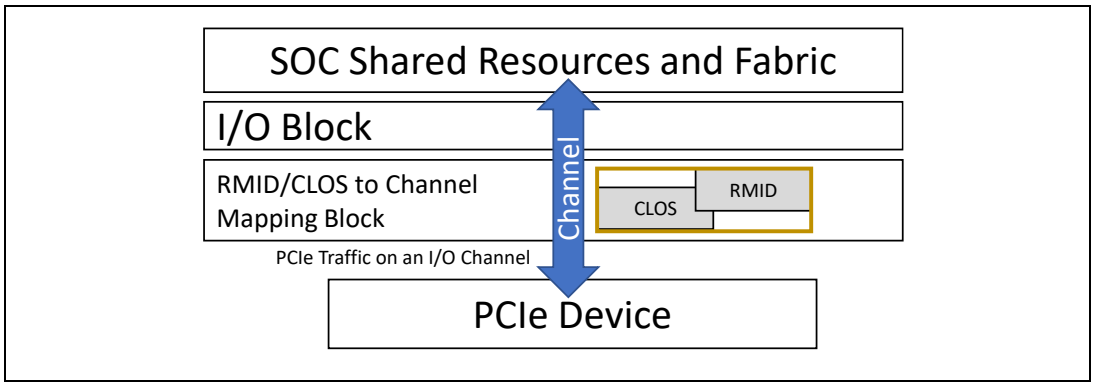
4.10 Use Cases

A number of non-CPU agent Intel RDT use cases are described in this section involving PCIe, CXL, and integrated accelerators.

As an implementation of the architectural model shown in Figure 4-4 and Figure 4-5, I/O block tags upstream DMA traffic (such as PCIe writes) as shown in Figure 4-8, enabling the device's resource utilization in the shared resources of the fabric, such as L3 cache, to be monitored and controlled through Intel RDT RMIDs and CLOS.

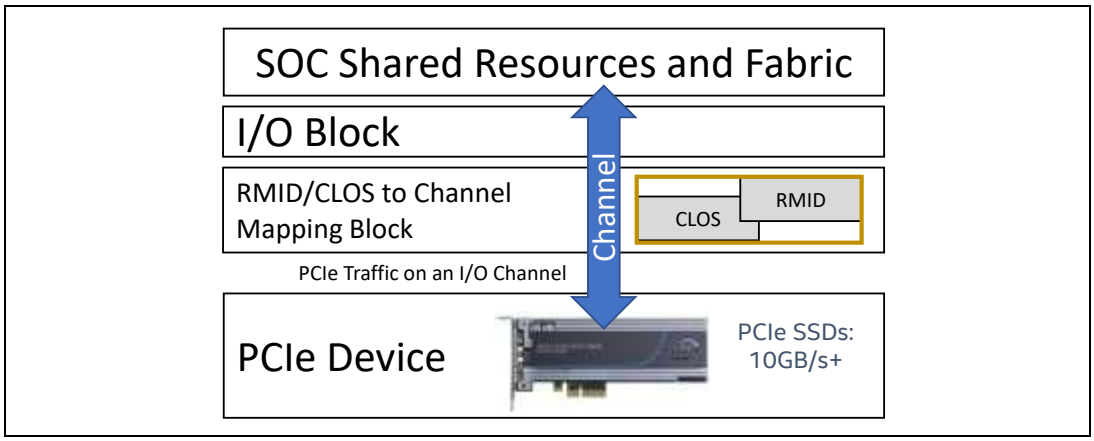
The applicable features for each tag are described in Appendix A.2, and software may configure these tags as described in Chapter 5, which describes the ACPI; see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for CPUID enumeration, and Section 4.8 and Chapter 7 for how the software may actuate these controls.

Figure 4-8. Device Traffic Tagging Model with PCIe as the Sole Traffic Path



As a concrete example, Figure 4-9 shows a high-performance PCIe SSD, subject to tagging with CLOS (so that its L3 cache footprint may be controlled), and RMIDs (so that its L3 cache occupancy and overflow bandwidth to memory may be monitored).

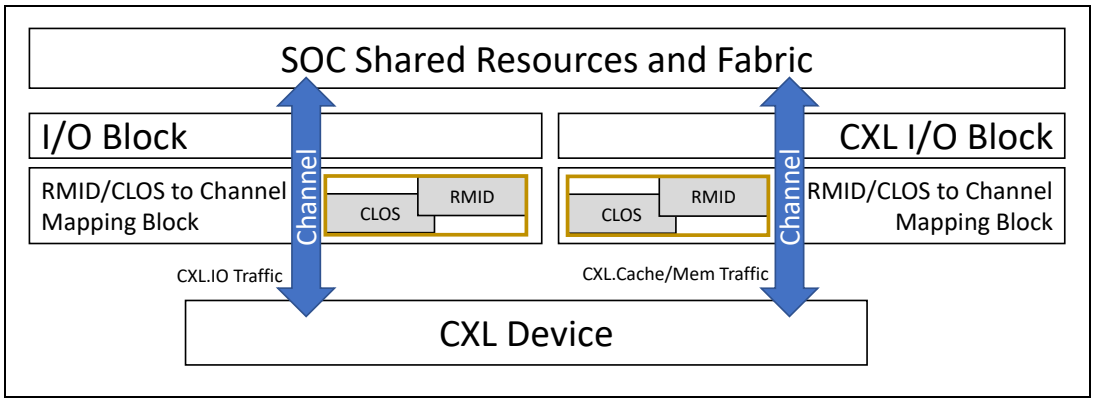
Figure 4-9. PCIe Device Example, with Traffic on a Channel Tagged with an RMID and CLOS



An example with a CXL device is shown in Figure 4-10, in which two paths are used for the device's traffic, one over CXL.IO, and one over CXL.Cache, through two separate I/O blocks, and note that the CXL.Cache link defines only one channel. In such a case, the software may configure RMID and CLOS tagging separately for the links. The links operate independently.

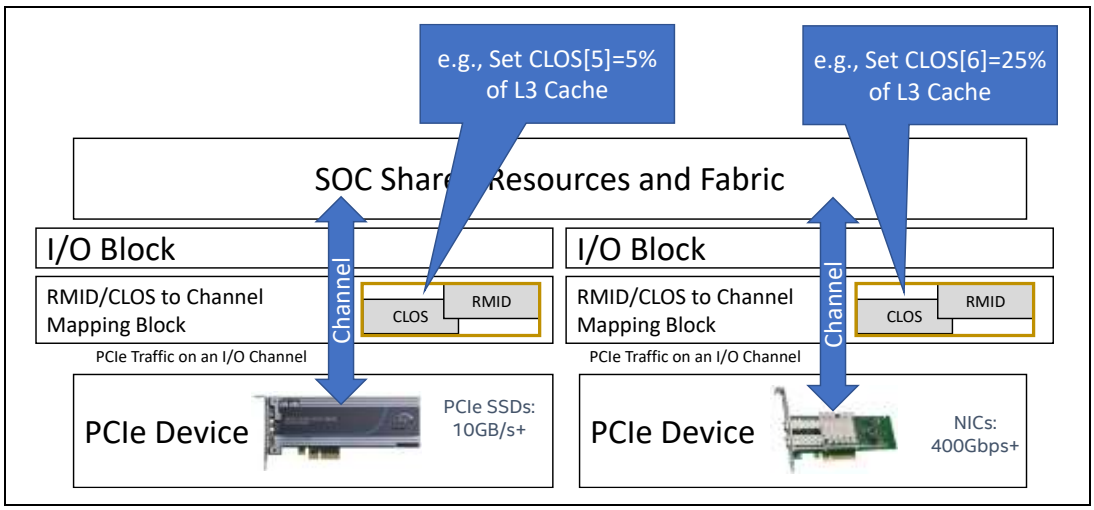
Note that no controls are provided for CXL.Mem, as the use of CXL.Mem resolves around accessing memory on a target device, and bandwidths from logical processors may be controlled with Intel RDT's Memory Bandwidth Allocation (MBA) feature. A more detailed discussion of this case surrounds Figure 4-14.

Figure 4-10. CXL Example of Device Tagging Model with CXL.IO and CXL.Cache Traffic Paths



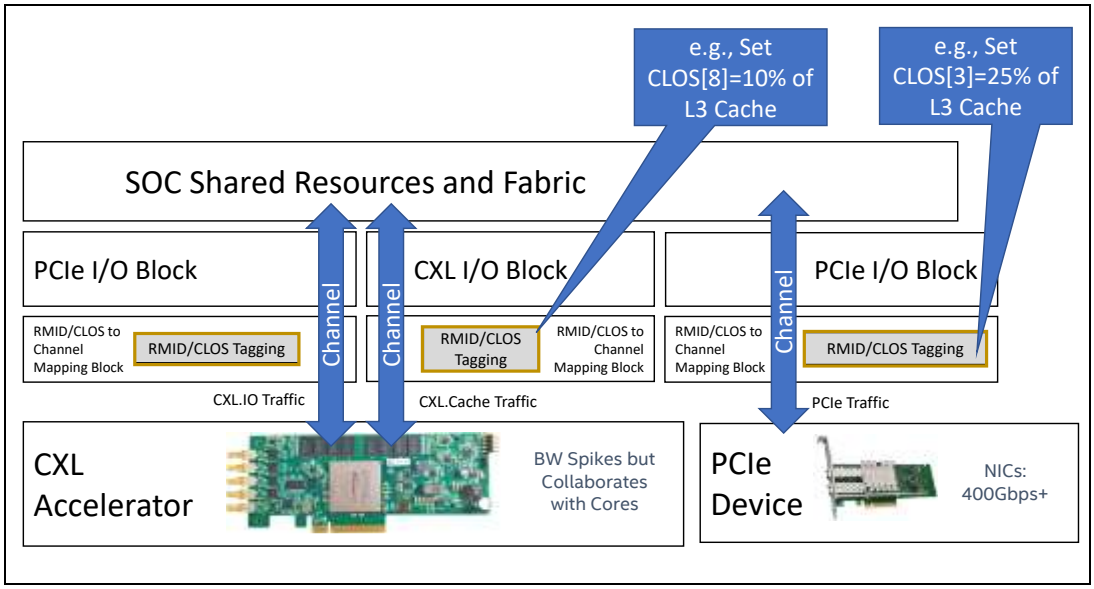
An example with multiple devices with different properties is shown in Figure 4-11, where a pair of PCIe devices on separate I/O blocks may be controlled independently, with separate RMID and CLOS tags. In this case a PCIe SSD which does not utilize the cache effectively may be limited, but a NIC which fills into the cache for data to be consumed by CPU cores may be prioritized.

Figure 4-11. Example of Controlling Two Different PCIe Devices



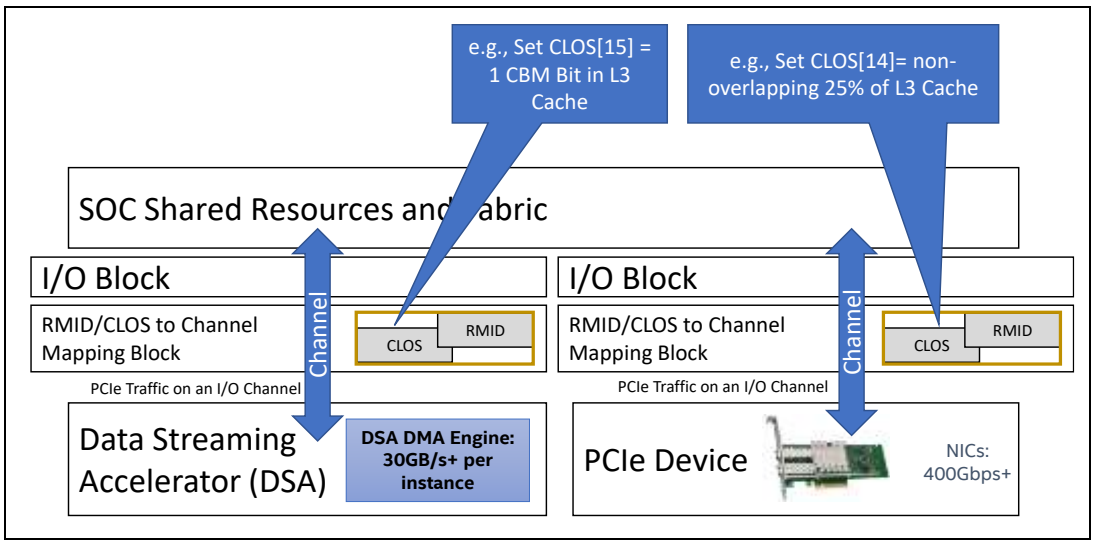
The following image shows an example with one CXL accelerator, perhaps a CXL-enabled FPGA card, utilizing CXL.IO and CXL.Cache, controlled independently from an I/O block with a PCIe device attached.

Figure 4-12. Example of Controlling a CXL Accelerator



An example of tagging and controlling an integrated accelerator, the Data Streaming Accelerator (DSA) alongside a PCIe device is shown in Figure 4-13. Depending on system load conditions and the DSA usage case, software may choose to allocate non-overlapping portions of the cache to minimize cache contention effects.

Figure 4-13. Example of Controlling a High-Bandwidth Integrated Accelerator

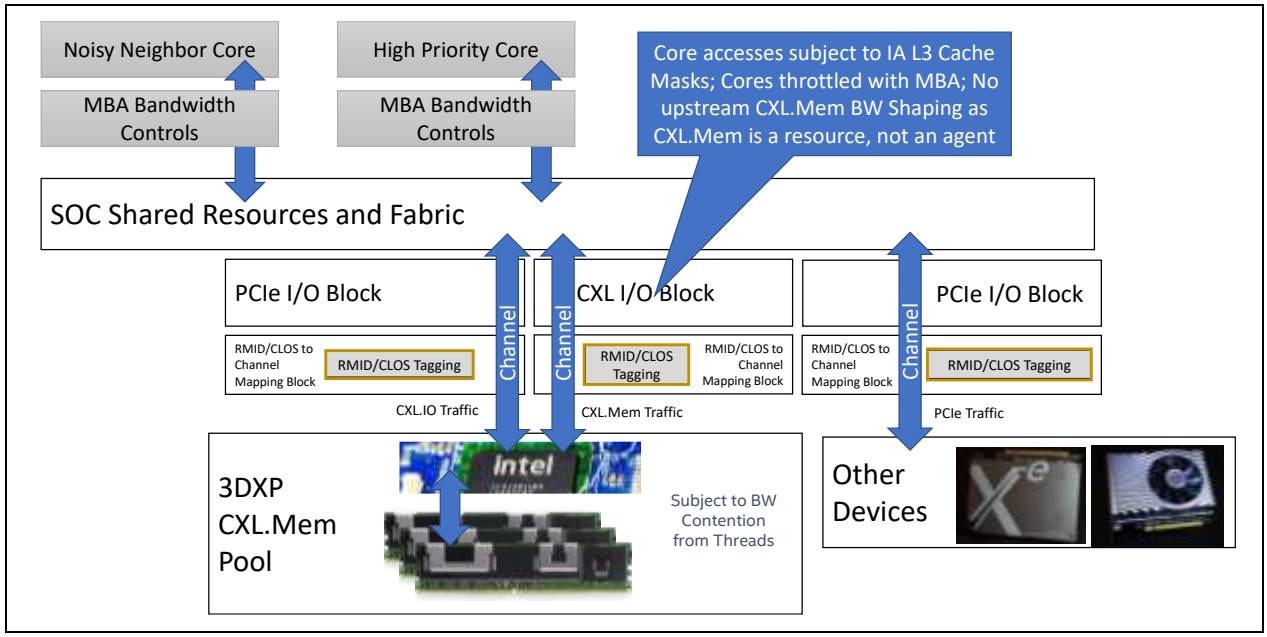


A complex example with multiple features in use is shown in Figure 4-14, where various PCIe devices are controlled with non-CPU agent Intel RDT, but a CXL device is also present, using CXL.IO and CXL.Mem links. The CXL device may be tagged and controlled on its CXL.IO interface.

As the main purpose of CXL.Mem is for host accesses to device memory, however, traffic responses up through the CXL.mem path are not subject to MBA bandwidth shaping, though they are sent with RMID and CLOS tags. If bandwidth is constrained on this link and software seeks to redistribute bandwidth across different priorities of accessing agents, such as CPU cores, the MBA feature may be used to redistribute bandwidth and throttle at the source of the requests (the agent's traffic injection point).

This example shows that for comprehensive management of cache and bandwidth resources on the platform, a combination of CPU agent Intel RDT and non-CPU agent Intel RDT controls may be necessary.

Figure 4-14. MBA to Control a CXL.Mem Pooling Device



5 BIOS Considerations

Software may query processor support of shared resource monitoring and allocation capabilities by executing CPUID for the CPU Agents Intel RDT features. An ACPI structure named IRDT may be consulted for further details on the enhanced Intel RDT feature support for non-CPU Agents. These ACPI structures also provide the locations of specific MMIO interfaces used to allocate or monitor shared resources.

5.1 Architectural Intel® RDT Features for Non-CPU Agents

This section describes ACPI enumeration for architectural Intel RDT features for non-CPU agents.

5.1.1 RMID/CLOS tagging - ACPI Enumeration

5.1.1.1 ACPI Definitional Goals

A number of goals are accomplished through the IRDT ACPI enumeration definition in this chapter, including:

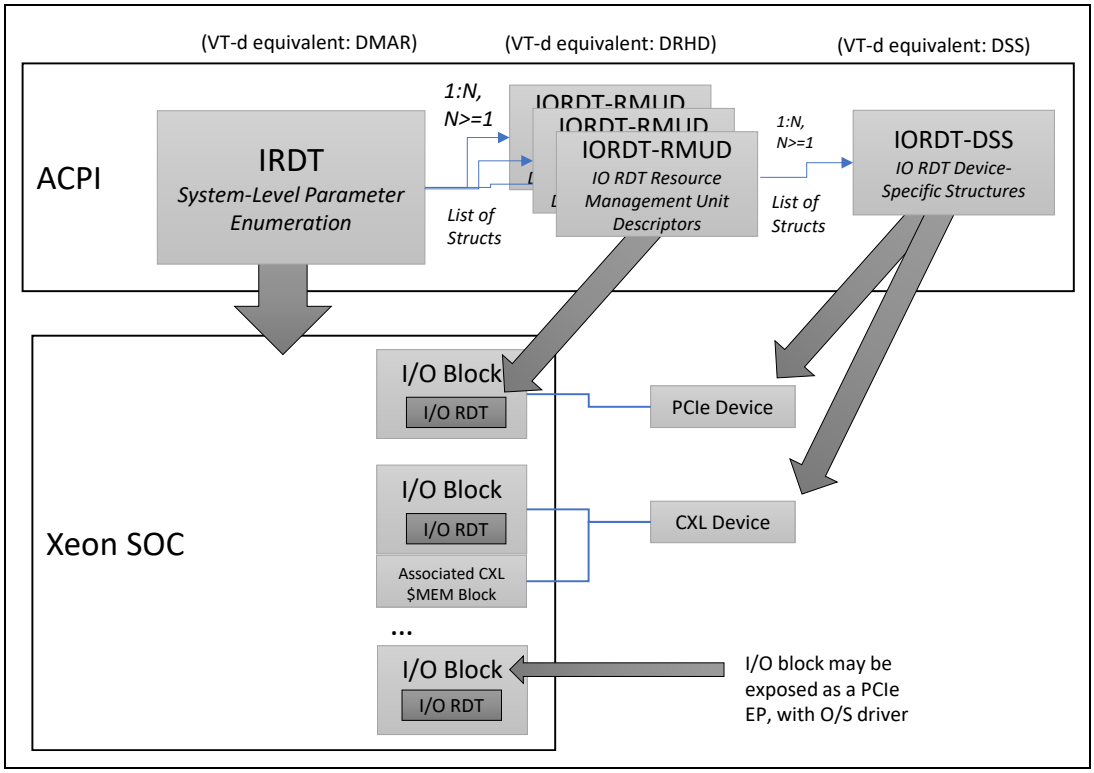
1. Providing top-level configuration information for the SoC, such as how many RMID/CLOS tags non-CPU agent Intel RDT supports relative to CPU agent Intel RDT (as enumerated by CPUID, see Chapter 3).
2. Providing a logical description of the control hierarchy – meaning which MMIO address to use to configure a link's RMID/CLOS tagging.
3. Provide flexibility in the implementation topology of devices behind I/O blocks, and cover cases with discrete or integrated PCIe and CXL links, and integrated accelerators.
4. Provide enhanced ease-of-use information for software, including device topologies, TC/VC/Channel mapping information for advanced QoS usages for forward-compatibility.

5.1.1.2 IRDT ACPI Enumeration Overview

This section provides a number of diagrams introducing key I/O Intel RDT structures and their mapping to Intel SoC components. Section 5.1.1.4 provides table specifics.

The top-level ACPI structure defined to support I/O Intel RDT is the "IRDT" structure. This is a vendor-specific extension to the ACPI table space [4]. The named IRDT structure is generated by BIOS and contains all other non-CPU agent Intel RDT ACPI enumeration structures and fields as described in this chapter.

Figure 5-1. Non-CPU Agent Intel® RDT ACPI Enumeration



Note that all Reserved fields in IRDT structures should be initialized to 0 by BIOS.

Under the IRDT structure in the hierarchy (embedded within the IRDT structure) are the I/O Intel RDT Resource Management Unit Descriptors (RMUDs.). The RMUDs typically map to I/O blocks within the system, though it is possible that one RMUD may be defined at other levels (such as one RMUD per SoC).

An example mapping is shown in Figure 5-1, showing ACPI details at the top, and Intel® Xeon® SoC mappings to hardware blocks at the bottom. The IRDT and RMUD relationships are shown for a typical implementation, in which RMUDs describe the properties of an I/O block. The IRDT table defines zero or more RMUDs, and an RMUD contains one of more RPs.

The RMUD structures contain two embedded structures, the Device Specific Structures (DSSes) and Resource Control Structures (RCSes) which map to devices and links and help describe the relationships regarding which I/O devices are connected to particular links, and which I/O links are in use by which devices. Each RMUD defines one or more DSS and RCS structures.

In the example of Figure 5-1, one DSS exists per PCIe, CXL or other non-CPU agent device (including accelerators), subservient to an RMUD. A CXL device may be expected to have multiple links (for example, CXL.Cache and CXL.IO) and this topology is described by the associated DSS structure and multiple RCS structures for the device and its links. Note that Figure 5-1 shows the DSS structure downstream of the RMUD but does not show the RCS for simplicity.

Figure 5-2 shows an example of the RMUD mapping to DSS and RCS structures. Each device attached to an I/O block is described by a DSS, and has one or more links, with properties described in the RCS structures. The RCS structures contain pointers to MMIO locations (in absolute address form, not BAR-relative) to allow software to configure the RMID/CLOS tags and bandwidth shaping properties, if supported, in an I/O Block.

Figure 5-2. ACPI Enumeration – Detail of DSS and RCS Structures Downstream from an RMUD

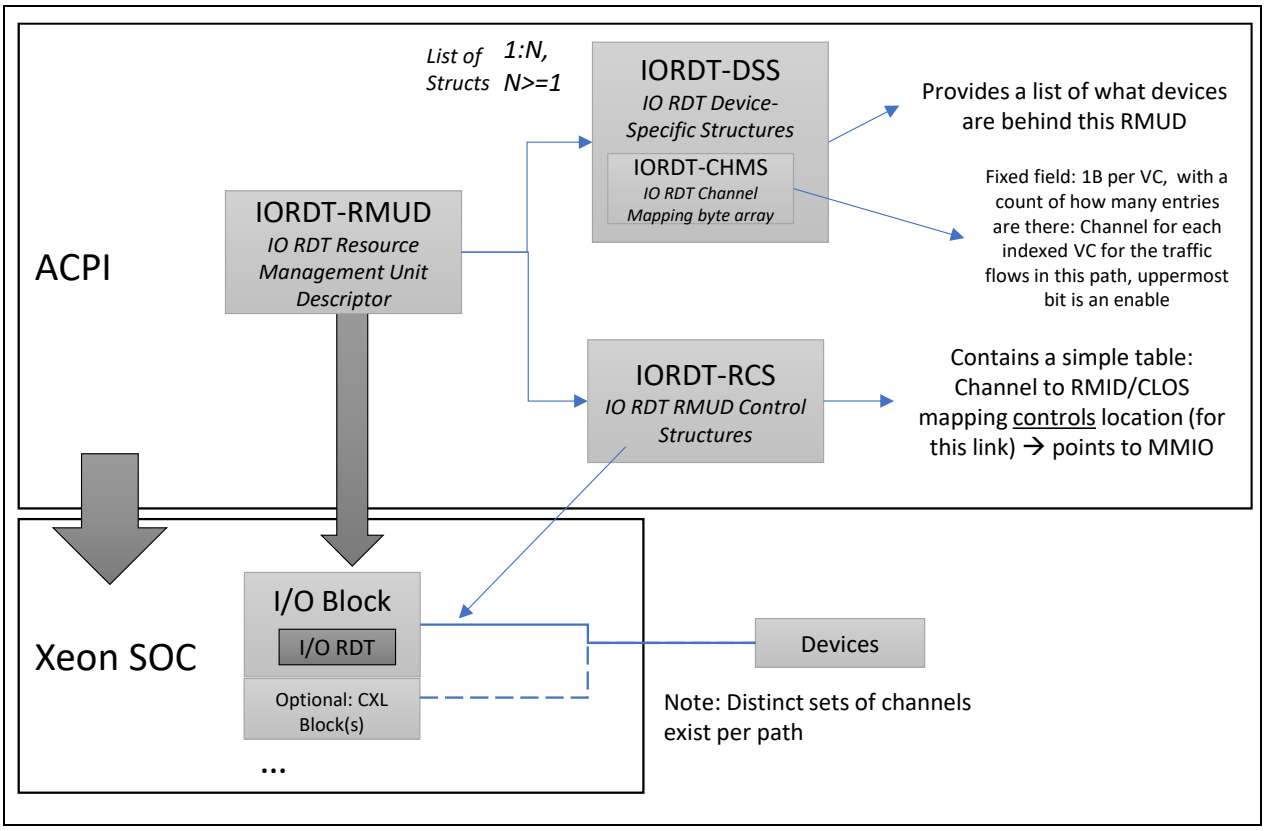
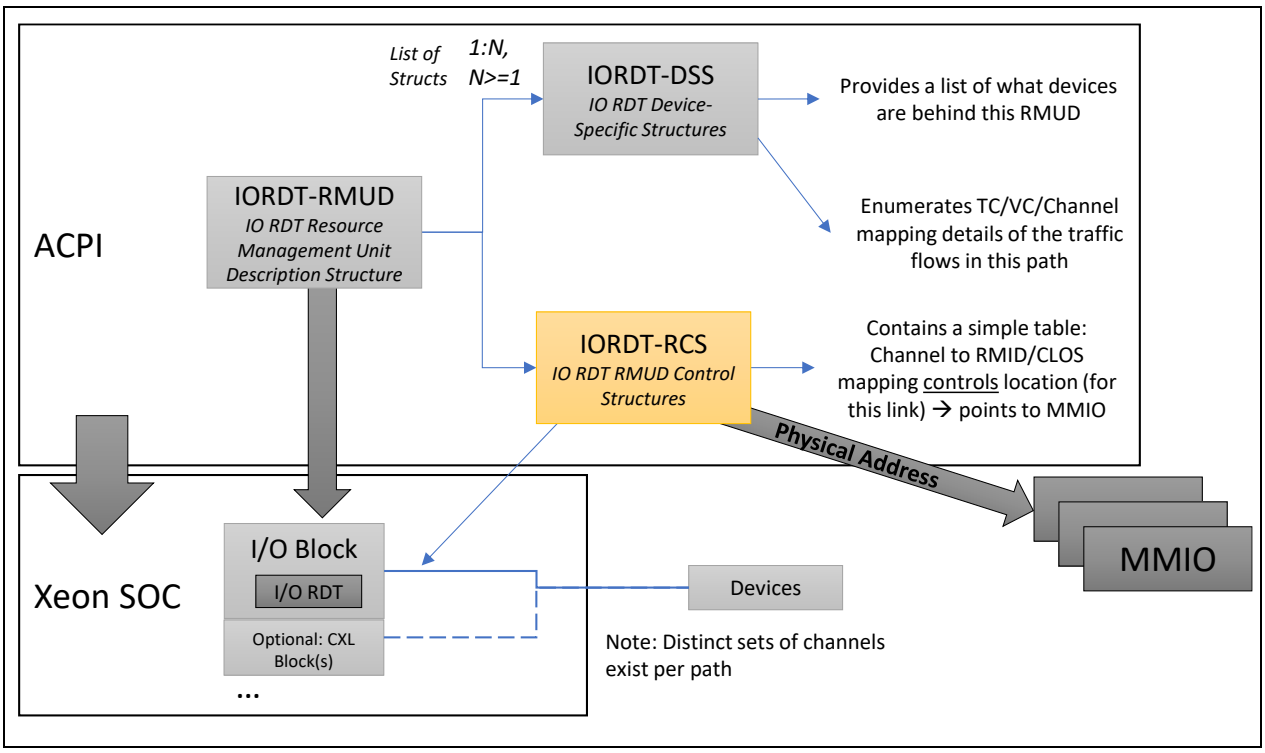


Figure 5-3 shows a further layer of detail where devices mapped through I/O blocks are described by the RMUDs, the DSS describes the properties of the device, and the RCS provides a pointer to the MMIO locations used for configuring the tagging and bandwidth shaping for a particular link.

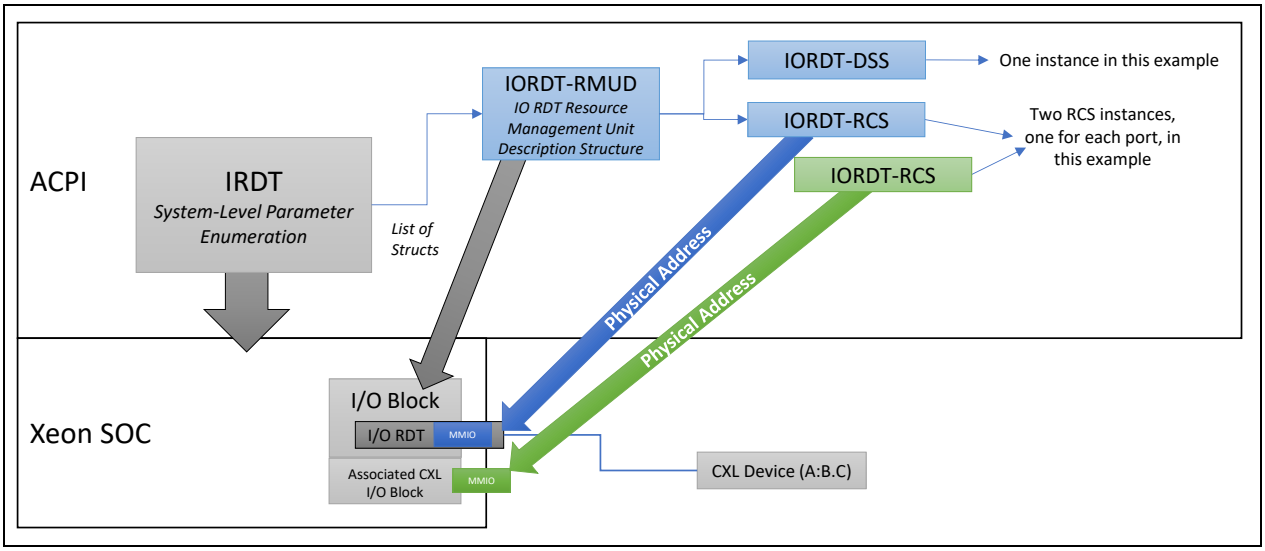
Figure 5-3. Mapping from RCS Structures to MMIO Addresses for Per-link Control



5.1.1.3 Example ACPI Enumeration Cases

Given the table hierarchy described in the preceding section, an example CXL Type 1 (CXL.IO + CXL.Cache) device mapping is shown in Figure 5-4. The device is described by one DSS behind an RMUD, while two RCSes are used, one for each link type (CXL.IO and CXL.Cache).

Figure 5-4. CXL Enumeration Example with CXL.IO and CXL.Cache Links



5.1.1.4 ACPI Feature Enumeration – Table Structure Details

5.1.1.4.1 Introduction and Notation

Given the previously described relationships of RMUD, DSS and RCS structures, table format details are described in this section.

Using the ACPI table hierarchy shown earlier in this chapter, following are the details of each table type and constituent fields. Field definitions are detailed in the table, and the text covers interpretation, corner cases, and interactions between fields.

5.1.1.4.2 IRDT Table Format and Field Descriptions

The top-level ACPI table, the I/O Resource Director Technology table (IRDT) is shown in Table 5-1, and one instance of this table is defined at the system level, generated by the system BIOS. This table includes a unique signature, and length including all sub-structures, including embedded RMUDs. The length of the IRDT table is variable.

Table 5-1. IRDT Table Format (Variable Length)

Field	Byte Length	Byte Offset	Description
Signature	4	0	"IRDT". Signature for the top-level I/O Intel RDT Description Table.
Length	4	4	Length, in bytes, of the description table including the length of the associated remapping structures.
Revision	1	8	1
Checksum	1	9	Checksum: Entire table must sum to zero.
OEMID	6	10	OEM ID.
OEM Table ID	8	16	For IORDT description table, the Table ID is the manufacturer model ID.
OEM Revision	4	24	OEM Revision of IRDT Table for OEM Table ID.
Creator ID	4	28	Vendor ID of utility that created the table.
Creator revision	4	32	Revision of utility that created the table.
IO Protocol Flags	2	36	Bit 0: IO_PROTO_MON -- Set if I/O Intel RDT Monitoring capabilities are supported somewhere on the platform for I/O protocol devices. Bit 1: IO_PROTO_CTL -- Set if I/O Intel RDT Allocation capabilities are supported somewhere on the platform for I/O protocol devices. Bit 2-15 : Reserved.

Field	Byte Length	Byte Offset	Description
Cache Protocol Flags	2	38	Bit 0: IO_COH_MON -- Set if I/O Intel RDT Monitoring capabilities are supported somewhere on the platform for coherent non-IA agents. Bit 1: IO_COH_CTL -- Set if I/O Intel RDT Allocation capabilities are supported somewhere on the platform for coherent non-CPU agents. Bit 2-15 : Reserved.
Reserved	8	40	-
Resource Management Hardware Blocks[]	-	48	A list of structures. The list will contain one or more Resource Management Unit Descriptors (RMUDs). The RMUD structure is described next.

A series of high-level flags allows the basic capabilities of monitoring and control for I/O links (for example, PCIe) and coherent links (for example, CXL) to be quickly extracted. Embedded within the IRDT table is a set of one or more Resource Management Unit Descriptor Structures (RMUDs), which are typically mapped to I/O blocks and define their properties. In some instantiations, one RMUD may be defined for the system, or in a finer-grained approach, one RMUDs may be defined for each downstream link and device combination, though this is expected to be an uncommon case.

5.1.1.4.3 RMUD Table Format and Field Descriptions

The Resource Management Unit Descriptor (RMUD) structure, definition is shown in Table 5-2, and includes a number of fields including length of the RMUD instance and all embedded sub-structures (DSS and RCS entries), an integration parameter that map to the SoC properties, including the minimum and maximum RMID and CLOS tags that are available for use in monitoring and controlling devices under this RMUD. While the common case is that these parameters would match the CPU agent Intel RDT parameters, there may be certain RMUDs which support a subset of the overall RMID and CLOS space.

Table 5-2. RMUD Table Format (Variable length)

Field	Byte Length	Byte Offset	Description
Type	1	0	Type 0 = “RMUD”. Signature for the I/O Intel RDT Resource Management Unit Descriptor.
Reserved	3	1	Reserved.
Length	4	4	Total length of this RMUD and all sub-structures.
Segment	2	8	The PCI Segment containing this RMUD, and all of the devices that are within it.
Reserved	3	10	Reserved.

Field	Byte Length	Byte Offset	Description
DSS and RCS Structures []	---	13	List of devices behind this RMUD, with one DSS table instance per device. Contains a list of DSS control structures and RCS control structures, identified by their "Type" field at offset zero in the sub-structures. The DSS and RCS structures described next.

Each RMUD entry contains a number of embedded DSS and RCS structures, identified by their "Type" fields, which describe the devices and links behind a given RMUD.

5.1.1.4.4 DSS Table Format and Field Descriptions

The Device Scope Structures behind each RMUD describe the properties of a device, that is, each DSS maps 1:1 with a device behind a particular RMUD. The DSS table definition is shown in Table 5-3, including a "type" field (Type = 0 identifies a DSS), the length of the entry, device type, and an embedded channel management structure (CHMS). The CHMS defines which RCS(es) are applicable to controlling this device (DSS), and which internal I/O block Channels each of the link's virtual channels (VCs) may map to (in the case of PCIe, up to eight VCs are supported, but only the first entry is valid in the case of CXL). Valid configurations for the CHMS include one entry per RCS (link).

In the DSS Device Type field, a value of 0x02 denotes that a PCIe Sub-hierarchy is described by this DSS. Each root port described by a DSS will have type 0x02. System software may use the enumerated devices found under such a root port to comprehend share bandwidth relationships in the channels under an RMUDS.

DSS type 0x01 indicates the presence of a root complex integrated endpoint device (RCEIP), such as an accelerator. Note that a PCI sub-hierarchy may denote a root port, and for every DSS that corresponds to a root port it is expected that Device Type = 0x2.

Note that the CHMS field contains a list of CHMS structures, which may describe for instances DSS entries which are capable of sending traffic over multiple channels (which are in turn described by unique RCS entries).

Note that no discrete pluggable devices (for example, PCIe cards) are directly described by the DSS entries, rather the root ports are indicated (Device Type 0x2).

Table 5-3. DSS Table Format (Variable length)

Field	Byte Length	Byte Offset	Description
Type	2	0	0 = DSS
Length	2	2	Length of this Entry in Bytes.

Field	Byte Length	Byte Offset	Description
Device Type	1	4	The following values are defined for this field. 0x01: Root Complex Integrated Endpoint (RCEIP) Device - The device identified by the 'Path' field is a root complex integrated PCI endpoint device. 0x02: PCI Sub-hierarchy - The device identified by the 'Path' field is a PCI-PCI bridge. In this case, the specified bridge device and all its downstream devices are included in the scope. Other values for this field are reserved for future use.
Enumeration ID	2	5	If Device Type equals 1 or 2, this field lists the BDF
Reserved	1	7	Reserved
Structure: CHMS and RCS Enumeration []	---	8	Packed as byte fields. One RCS may support multiple DSSes, and one DSS may have multiple RCSs (links), so this is an array, with size derivable from the DSS Length field. Within each entry: **Byte 0:** RCS Enumeration ID controlling this link. Corresponds to the enumeration ID of the RCS structure under this DSS. **Bytes 1-8:** Represents the index into the "RCS-CFG-Table" used by the corresponding VC. Byte 1 represents the channel for VC0, Byte 2 represents the channel for VC1, and so on. In this field, bit 7 is a valid bit (entry is not valid if enable bit is cleared). Bit 6, when set, indicates that this channel is shared with another DSS. The number of valid bytes in this field is defined in the per-RCS "Channel Count" field, any unused bytes (for example, for a single-Channel CXL link) are Reserved. **Bytes 9-15:** Reserved (padding)

5.1.1.4.5 RCS Table Format and Field Descriptions

The RCS structure provides details of the type of monitoring and controls supported for a particular link interface type, such as PCIe or CXL, and an MMIO location in which a table exists that can be used to apply monitoring and control features. The MMIO location provided is absolute location in MMIO space (64 bits), rather than hosted in a particular device and defined relative to a BAR.

Table 5-4. RCS Table Format (Currently 40B)

Field	Byte Length	Byte Offset	Description
Type	2	0	RCS = 1.
Length	2	2	Length, in bytes, of the description table including the length of the associated remapping structures.
Link Interface Type	2	4	Type of link interface: 0x0 = PCIe or CXL.IO 0x1 = CXL.Cache 0x2 and above: Reserved
RCS Enumeration ID	1	6	A unique identifier for this RCS under this RMUD.
Channel Count	1	7	Number of Channels defined for this link interface (affects the interpretation of the CHMS structure within the corresponding DSS).
Flags	2	8	Bit 0: Reserved. Bit 1: RTS: RMID Tagging supported. Bit 2: CTS: CLOS Tagging Supported. Bit 3: REGW: if set, the RMID and CLOS defined in the RCS Block MMIO locations are 2B registers. If clear, they are 8B registers. Bits 4-15: Reserved.
RMID Block Offset	2	10	Byte offset from the RCS Block MMIO Location where the RMID tagging fields begin.
CLOS Block Offset	2	12	Byte offset from the RCS Block MMIO Location where the CLOS tagging fields begin.
Reserved	18	14	Reserved.
RCS Block MMIO Location	8	32	RCS Hosting I/O Block MMIO BAR Location defines an MMIO physical address.

Note that if CXL.IO and PCIe devices share the bandwidth of a certain RCS and its channels, then traffic for both protocols is carried on the same channel entries.

Note that in the enumeration the fields, the RMID offset, and CLOS offset are specified relative to the “RCS Block MMIO Location” field, meaning that the RMID and CLOS offsets may be relocatable within the MMIO space. The offset defines the block of a contiguous set of RMID or CLOS tagging fields, and the number of entries is defined by the “Channel Count” field (for example, a value of 8 channels may be common in certain PCIe tagging implementations).

5.2 Model-Specific Intel® RDT Features for CPU Agents

This section describes BIOS knobs for Model-Specific Intel RDT features for CPU agents.

5.2.1 BIOS knobs for Resource Aware MBA

See Appendix A.3 for Resource Aware MBA (MBA 4.0) feature supported product details. See Appendix B.1.1 for Resource Aware MBA (MBA4.0) feature details.

The Resource-aware MBA feature is a model-specific extension to the Third Generation of MBA (Chapter 3) which provides a set of extended capabilities to better handle heterogenous memory types on complex modern SoCs. A model-specific implementation is used as memory types may change significantly over the course of time. A more detailed description of Resource Aware MBA is provided in the next chapter.

To support Resource Aware MBA, the system BIOS shall support a legacy BW profile configuration knob with a drop-down menu of three options as with Second-Generation MBA.

- MBA BW profile
  - Linear(default)
  - Biased
  - Legacy

In addition, BIOS shall add three knobs with a drop-down menu for Resource-Aware MBA in particular. These scaling ratios enable tuning of MBA calibration values to the typical bandwidth levels available from each type of heterogeneous downstream memory type, and tuning values may be further scaled by the number of memory channels or links populated with each type of memory. An example implementation of this tuning code will be provided with the Intel Reference BIOS implementation for each applicable platform.

1. Description: "PMM BW downscaling vs the baseline Total memory BW profile. For example: picking 1/2x at results in scaling PMM BW throttling in a 2:1 ratio versus DDR throttling."
   - PMM MBA BW downscale
     - 1x (default)
     - 1/2x
     - 1/4x
     - 1/8x
2. Description: "CXL (Type3) BW downscaling vs the baseline Total memory BW profile. For example: picking 1/2x results in scaling CXL (Type3) BW throttling in a 2:1 ratio versus DDR throttling."
   - CXL (Type3) MBA BW downscale
     - 1x (default)

- 1/2x
- 1/4x
- 1/8x

3. Description: “Remote Target BW downscaling vs the baseline Total memory BW profile. For example: picking 1/2x results in scaling Remote Target BW throttling in a 2:1 ratio versus DDR throttling.”
   - Remote Target MBA (UPI) BW downscale
     - 1x (default)
     - 1/2x
     - 1/4x
     - 1/8x

6 MMIO Register Descriptions

This chapter describes the Intel RDT related MMIO registers. As mentioned in previous chapters, traditional interfaces such as MSRs are discussed in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

6.1 Non-CPU Agent Intel® RDT Register Location

The Non-CPU agent Intel RDT related register set (MMIO interfaces) must reside on at least one 4 KB-aligned memory mapped page. The exact location for the register region is implementation-dependent and is communicated to system software by BIOS through the IRDT ACPI structure (see Chapter 5). Multiple RCSes could be mapped to the same 4 KB-aligned page, or distinct pages. No other unrelated registers may be present in the pages used for non-CPU agent Intel RDT. A Virtual Machine Monitor (VMM) or operating system may use page-based access controls to ensure that only designated entities may use the non-CPU agent Intel RDT controls.

When accessing non-CPU agent Intel RDT MMIO interfaces, note that writes to reserved fields, writes to reserved offsets within the MMIO space, or writes of values greater than the supported maximum for a field will be ignored by hardware.

6.1.1 Software Access to Registers

Software interacts with the non-CPU agent Intel RDT features by reading and writing memory-mapped registers. The following requirements are defined for software access to these registers.

- When updating registers through multiple accesses (whether in software or due to hardware disassembly), certain registers may have specific requirements on how the accesses should be ordered for proper behavior. These are documented as part of the respective register descriptions.
- Locked operations to non-CPU agent Intel RDT related registers are not supported. Software should not issue locked operations to non-CPU agent Intel RDT feature hardware registers.

6.1.2 Register Descriptions for Non-CPU Agents

6.1.2.1 Link Interface Type RMID/CLOS Tagging MMIO Interfaces

The IRDT ACPI structures defined in Chapter 4 define MMIO interfaces for configuring the RMID/CLOS for each link interface type, as defined in the RCS structures. An MMIO pointer defined in the RCS fields describes where the configuration interface exists for a particular link interface type. The MMIO locations are specified as absolute physical addresses.

Table 6-1 shows the MMIO field layout for RMID and CLOS tagging, and bandwidth shaping. A common format is used for all RCS types, including for instance RCS instances that support PCIe or CXL use the same field layout.

Common table format across all RCS-Enumerated MMIO.

Table 6-1. MMIO Table Format

Register Name	Mem Offset	Length (B)	Comments
IO RDT Reserved	0x0000	Variable	Reserved
IO_PQR_CLOS0	RCS :: CLOS Block Offset	RCS :: REGW	Common across all RCS types
IO_PQR_CLOS1	IO_PQR_CLOS0 + RCS :: REGW	RCS :: REGW	Per-channel
IO_PQR_CLOS2	IO_PQR_CLOS0 + RCS :: REGW*2	RCS :: REGW	Per-channel
…	Variable	Variable	-
Reserved	Variable	Variable	-
IO_PQR_RMID0	RCS :: RMID Block Offset	RCS :: REGW	Common across all RCS types
IO_PQR_RMID1	IO_PQR_RMID0 + RCS :: REGW	RCS :: REGW	Per-channel
IO_PQR_RMID2	IO_PQR_RMID0 + RCS :: REGW*2	RCS :: REGW	Per-channel
…	Variable	Variable	-
Reserved	Variable	Variable	-
IO_RDT Reserved	Variable	Variable	Remainder of the page

Note that the RCS :: REGW field indicates the register access width of the fields in Table 6-1, either 2B or 8B. Depending on the implementation, this width may be 2 bytes or 8 bytes. The width is indicated by the REGW field in the RCS Table (Section 5.1.1.4.5).

Note that the base of the RMID and CLOS fields are enumerated in the RCS structure, and the size of these fields varies with the number of supported channels. The set of configurable RMIDs and CLOSs are organized as contiguous blocks of 4B registers.

The “PQR” fields starting at the enumerated offset (RCS :: CLOS Block Offset) are defined with enumerated register field spacing of RCS :: REGW, which may require either 2B or 8B register accesses. A block of CLOS registers exists, followed by a block of RMID registers, indexed per Channel. That is, setting a value in the IO_PQR_CLOS0 field will specify the CLOS to be used for Channel[0] on this RCS.

The valid field width for RMID and CLOS is defined via CPUID leaves (see Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B for details) for shared-L3 configuration.

Higher offsets allow multiple Channels to be programmed (above Channel 0) if supported. Given that PCIe supports multiple VCs, multiple channels may be supported in the case of PCIe links, but CXL links support only two entries, one at IA_PQR_CLOS0 and one at IO_PQR_RMID0 in this table.

The RMID and CLOS fields are interpreted as numeric tags, exactly as they are in the CPU agent Intel RDT feature set, and software may assign RMID and CLOS values as needed.

Software may reconfigure RMID and CLOS field values at any point during runtime, and values may be read back at any time. As all architectural CPU agent Intel RDT infrastructure, it is dynamically reconfigurable, this enables control loops to work across the capabilities sets collaboratively and consistently.

7 Programming Guidelines

7.1 Intel® RDT Monitoring Software Flows for CPU Agents

Intel RDT Monitoring software flows for CPU agents in certain example software implementations are briefly described in this section to provide context for how an end-user could view and use the RDT features. While this chapter provides examples and recommended flows, it is in no way limiting to use models once enumeration and configuration capabilities are enabled in software, and many varied software implementations and usages of RDT beyond the listed examples have been observed.

7.1.1 Intel® RDT Monitoring Software Flows for CPU Agents

Software should first verify the existence of the RDT Monitoring feature(s) before attempting to configure it and read back monitoring data. Periodic management by software may also be required to maintain the proper RMID mapping on a logical thread when context switching or receiving an interrupt for instance (see Section 3.1.1 for details).

7.1.1.1 Step 1 – Enumeration

Before attempting to read or write MSRs associated with the Intel RDT Monitoring feature software should first execute the CPUID instruction and parse its output to ensure that Intel RDT Monitoring and any sub-features to be used (for example, CMT, MBM) are supported on the platform, otherwise General Protection (#GP(0)) faults will be generated.

As discussed in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, if CPUID feature flag CPUID.0x7(Structured Extended Feature Flags).0:EBX[12] is set to '1' then Intel RDT Monitoring is generally supported on the platform.

Once Intel RDT Monitoring support has been verified software should use CPUID.0xF(Shared Resource Monitoring Enumeration leaf).0:EDX to examine which platform resources support monitoring. After the call to CPUID, the EBX register will indicate the maximum RMID supported on the current socket (though particular resources may support fewer RMIDs and this can be enumerated on a per-resource basis as described next).

Software may use CPUID.0xF(Shared Resource Monitoring Enumeration leaf).ResID to determine the number of RMIDs supported for the specific resource in question, the event type bitmask to program into IA32_QM_EVTSEL to retrieve the data for that event in IA32_QM_CTR, and the upscaling factor as discussed in the feature-specific chapters. Software may optionally choose to

build a record of these enumeration responses for each resource to reduce overhead from repeated CPUID calls.

Given that certain processors may support multiple L2 caches, multiple-L3 caches, and a variety of logical processor types, it is recommended that software use CPUID from the perspective of each logical processor to comprehend any asymmetric resource support which may be present.

Software should parse Processor Family, Model and Stepping (FMS) to verify that a particular processor includes support for a given model-specific feature. To find out which features are supported on which specific products, refer to Appendix A.3.

7.1.1.2 Step 2 – RMID Association

After verifying that the platform supports Intel RDT Monitoring, software should associate each logical thread or VM of interest with an RMID such that resource utilization by the threads can be tracked. It is expected in general that if an OS or VMM moves an application from one core or socket to another that the RMIDs will be updated (moved along with the app or remapped onto another socket as needed) to maintain an accurate mapping between the applications of interest and the RMIDs programmed onto a logical thread.

Threads by default are initialized to RMID[0], which provides insight into memory bandwidths for the system but not necessarily cache occupancy (which would read 100% occupied in a non-idle system).

7.1.1.3 Step 3 – Event Selection Setup

After associating RMIDs with threads and updating the IA32_PQR_ASSOC register for each thread as needed while running (to account for context swaps and thread migration between cores), software may execute for an arbitrary period of time while hardware tracks occupancy before polling for the resulting occupancy.

After applications have executed for the desired time period software may program an RMID and event code into the IA32_QM_EVTSEL MSR, which will cause the corresponding data to be available in the IA32_QM_CTR MSR (discussed in the following section).

7.1.1.4 Step 4 – Data Sampling

After the IA32_QM_EVTSEL MSR has been programmed with an RMID / Event ID combination the corresponding event data can be read back from the IA32_QM_CTR MSR, which has a bit field layout as defined in Section 3.1.1. Software must check both the Error bit and the Unavailable bit to verify that the data returned is valid (along with the Overflow bit if supported) – if an error is indicated the monitoring data reported back must not be used.

As described in Section 3.1.1 the Error bit will be set if an RMID greater than the global maximum (specified in CPUID) is programmed into IA32_QM_EVTSEL, or an unknown/unsupported Event ID is programmed.

Similarly, the Unavailable bit is set when data is requested for an RMID that does not support that particular resource or does not support an RMID value that high.

An example is if occupancy monitoring of resource "A" supported four RMIDs, and resource "B" supported 2 RMIDs. If software requested the occupancy of either Resource A or B for RMIDs 0 or 1 then valid data would be reported back. If occupancy data for RMIDs 2 or 3 was requested for resource "B" however data would not be reported, and the Unavailable bit would be set.

The Overflow bit, if supported, is set when an overflow of an incrementing counter is triggered, allowing software to correct or discard errant values that may lead to erroneous bandwidth calculations.

If an error is indicated, it will be cleared automatically once valid values are programmed into IA32_QM_EVTSEL and any hardware conditions preventing accurate monitoring are resolved. The Overflow bit, if implemented, is cleared on a read of IA32_QM_CTR.

7.1.1.5 Step 5 – Sample CMT/MBM Data Collection and Analysis

Once CMT and MBM data has been collected it can be interpreted as described in the following example.

Consider the case where CMT and MBM are supported on a platform, and a large number of RMIDs are available. On this platform the user seeks to profile two threads within an application, so both threads are assigned individual RMIDs and run on separate physical cores for a period of one second, then occupancy and bandwidths are read back via the MSR interface (IA32_QM_EVTSEL and IA32_QM_CTR). In this example, the following parameters are key to interpreting the results:

- System topology – two Intel® Xeon™ CPUs with 14 cores per socket, and a 3-level cache subsystem, where the last-level cache totals 35 MB per socket.
- The last-level cache is verified using CPUID leaf 0x4 as the last level cache between the cores and memory, meaning L3 external bandwidth values can be used to measure memory bandwidth.
- As enumerated via CPUID the upscaling factor (CPUID.0xF(Shared Resource Monitoring Enumeration leaf).1:EBX) to convert counter values to final values in bytes is 0xE000 (decimal 57344).
- Since the total L3 cache size is 36700160 bytes and the upscaling factor is 57344, we know that the maximum possible CMT occupancy counter value reported by the system will be total cache size divided by the conversion factor, or 36700160/57344 = 640.
    - As the threads are profiled, we can compare the reported occupancy to the maximum occupancy counter value, giving an indication of what fraction of the total cache an application is using without needing to convert to bytes first.

Suppose that the threads are configured as follows:

- Associate thread[0] with RMID[1].
- Associate thread[1] with RMID[2].
- Leave all other threads in the system with the default RMID[0] association.

In order to profile memory bandwidth an initial sampling of the free-running MBM counters is required:

- Program IA32_QM_EVTSEL with RMID[1] for the first thread and event code 0x2 for total L3 external bandwidth, then read the corresponding data from IA32_QM_CTR (and verify that the Unavailable and Error bits in IA32_QM_CTR are not set so the data is valid).
- Program IA32_QM_EVTSEL with RMID[1] for the first thread and event code 0x3 for local L3 external bandwidth, then read the corresponding data from IA32_QM_CTR (and verify that the Unavailable and Error bits in IA32_QM_CTR are not set so the data is valid).
- Repeat these steps with RMID[2] for the second thread.

Note that we assume that RMID[1] and RMID[2] have previously been used for profiling other applications, so they may initially contain nonzero occupancy and bandwidth counter values.

Note that in this example we assume that RMID[1] and RMID[2] are set up exclusively for the use of the two threads being profiled, and that these threads are not currently scheduled, and they have no data in the L3 cache, so the bandwidth counters, even if they contain initial values, are not changing. The occupancy counters may change even if no threads are scheduled using RMID[1] and RMID[2] however if they have previously run and have data in the L3 cache as other threads on the system run and cache space is dynamically redistributed due to evictions and standard cache LRU policies.

Note that if the threads in RMID[1] and RMID[2] are running while we measure initial counter values then skew may appear in the counter values, proportional to the time delay between reading each of the event codes (which should be minimized) and the bandwidths consumed by the application (which may vary significantly based on application behavior).

Now that initial MBM counter values have been established, the program can be left to run for a period of time, in this case one second. The Intel RDT Monitoring data can then be read back as follows:

- Program IA32_QM_EVTSEL with RMID[1] for the first thread and event code 0x1 for L3 cache occupancy, then read the corresponding data from IA32_QM_CTR (and verify that the Unavailable and Error bits in IA32_QM_CTR are not set so the data is valid).
- Program IA32_QM_EVTSEL with RMID[1] and the event code for total L3 external bandwidth (0x2), read the data from IA32_QM_CTR and again verify that the “U” and “E” bits are not set.
- Similarly read back local L3 external bandwidth using the event code 0x3 and verify that the data is valid.

- Repeat the previous three steps with RMID[1] to read back the Intel RDT monitoring metrics for the second thread.

Example data read back after profiling for one second is shown in the following table.

Table 7-1. Example CMT and MBM Counter Values

	Thread 0		Thread 1	
Event Type	First Sample	Second Sample	First Sample	Second Sample
L3 Cache Occupancy	N/A	0x25	N/A	0x180
Total L3 External Bandwidth	0x00FE985E	0x00FEBC14	0x00002541	0x0000D9F7
Local L3 External Bandwidth	0x0A8C9512	0x0A8CB5ED	0x00000314	0x0000AC5D

Note that in the previous sample data the counter values are shown as 32-bit values, implying that the upper fields in the counter MSR were either zeroes or not changing and can be disregarded – this may not always be the case however when bandwidths are high, or in the case of future counters which may increment quickly.

In the example, the final cache occupancy for the threads can be calculated as follows:

- Thread[0]: CounterValue * UpscalingFactor = 37*57344 = 2121728 bytes (roughly 2.02 MB).
- Thread[1]: CounterValue * UpscalingFactor = 22020096 bytes = 21 MB.

Thus, based on the CMT profiling of the two example threads, we see that Thread[0] consumes around 2MB of cache space, and Thread[1] consumes around 21MB, over 10x more, which indicates that it likely has a larger data working set or it may be partly streaming through memory. Software should also consider memory bandwidth readings to determine whether Thread[1] is simply cache-friendly or whether it is a streaming application.

Total memory bandwidth values for the two threads can be determined as follows:

- Thread[0]: (Second counter reading – First counter reading)*UpscalingFactor = (0x00FEBC14-0x00FE985E)*57344 = 9142*57344 = 524238848 bytes/second, or around 500 MB/s since we sampled for one second.
- Thread[1]: (Second counter reading – First counter reading)*UpscalingFactor = (0x0000D9F7-0x00002541)*57344 = 46262*57344 = 2652848128 bytes/second, or around 2.5 GB/s.

Local memory bandwidth values for the two threads can be determined as follows:

- Thread[0]: (Second counter reading – First counter reading)*UpscalingFactor = (0x0A8CB5ED-0x0A8C9512)*57344 = 8411*57344 = 482320384 bytes/second, or around 460 MB/s.

- Thread[1]: (Second counter reading – First counter reading)*UpscalingFactor = (0x0000AC5D-0x00000314)*57344 = 43337*57344 = 2485116928 bytes/second, or around 2.3 GB/s.

Based on the prior calculations we observe that Thread[0] has low memory bandwidth demands at roughly 500 MB/s, and Thread[1] uses more bandwidth at 2.5 GB/s, but not enough to classify it as a streaming thread. With its 21 MB cache occupancy and moderate memory bandwidth, Thread[1] is best classified as a cache-friendly thread, though observing its behavior over a longer period of time and sampling other system metrics to better understand its time-variant behavior and compute requirements is recommended if detailed profiling is the goal.

Note that in this example most of the bandwidth demands of the threads are satisfied by the memory controller on the local CPU, meaning bandwidth associated with the QPI link and other sources is low, implying that the NUMA-aware OS properly located the memory allocation for the threads on the same socket as the running threads.

This may not always be the case however, and if a bandwidth imbalance is detected then we may choose to either move the compute threads to the other CPU (closer to the data in memory) or move the data in memory to another address range within the scope of the local CPU memory controller for better performance.

7.1.2 Native OS Environments

In a non-virtualized environment, the RMIDs can be associated with applications or application threads. The OS may even choose to associate different parts of a single application to be associated with different RMIDs if needed. But a typical usage would save and restore the RMIDs along with the context information during the context switch.

For multi-threaded applications, multiple threads can share the same RMID. The implications stated earlier also apply to multi-threaded applications with the following additional considerations for shared code/data. For example, if app0 was multi-threaded (for example, two threads per application), then we can get occupancy information for each thread of application. The only additional implication here is that the occupancy of the threads that share data will be associated to the thread that filled the shared data. Heuristics that minimize contention in the shared cache for single threaded workloads to optimize total system throughput and to provide QoS will also be effective for the multi-threaded workloads.

7.1.3 Virtualization Scenarios

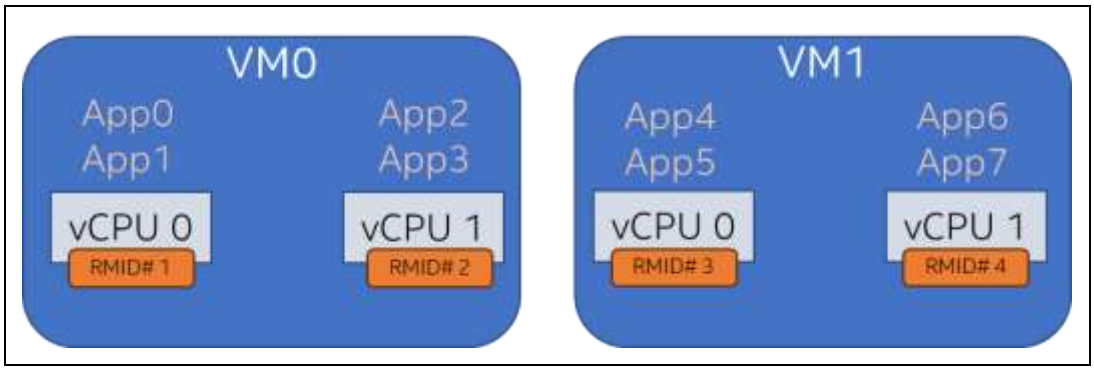
In case of virtualization, RMIDs can be allocated in different ways. The VMM can choose to allocate the RMIDs to different VMs or vCPUs. The current planned implementations do not support reporting individual occupancies of applications running within a VM unless the VMM and guest OS are both

enabled to support Intel RDT. The RMID assignment at VM and VCPU level are described next.

RMIDs assigned to Virtual Machines (VMs): In this usage case RMIDs are assigned to VMs instead of applications and the occupancy reported is on a per VM basis. Multiple applications running within a VM will have a consolidated occupancy which will be reported by the RMID. Profiling of workloads and heuristics that optimize for overall system throughput and for providing QoS based on SLAs would be based in the granularity of VMs. Hence to provide QoS, HP applications can be mapped to a VM with a high priority so that scheduling decisions to minimize contention will treat all applications running in the HP VM as high priority. The heuristics that work on occupancy monitoring based on contention in the shared cache will still be effective but will work in the granularity of VMs. When scheduling VMs, the VMM can use the occupancy monitoring information available for the VMs from the RMIDs. There are no other additional implications for VMs.

RMIDs assigned to vCPUs within VMs: In this usage case scenario, RMIDs are assigned to vCPUs within a VM. Since there maybe multiple applications within a VM running on the vCPUs, the occupancy reported by the RMID for a vCPU will represent the consolidated occupancy of the applications running on that vCPU. As an example, if there are two VMs with 2 vCPUs each and there are four applications in each VM as shown in Figure 7-1.

Figure 7-1. RMIDs Assigned to vCPUs



The occupancy reported by the RMID assigned to vCPU0 will represent the consolidated occupancy of App0 and App1. Similarly, only the consolidated occupancy of App2 and App3 is what will be reported and so on. Hence optimizations for system throughput, QoS and application profiling would have to be at the granularity of vCPUs. The OS running within a VM will have its own scheduling policy that would determine how applications are scheduled to the vCPUs.

When applications migrate within a VM from one vCPU to another, the consolidated occupancy reported will also be affected as it would depend on the nature of the applications scheduled to a vCPU. Hence any policy or heuristic that is implemented should be in the granularity of VCPU profiling. The recommended approach is to profile the workload at a vCPU level and then design heuristics based on vCPU profiles to optimize for throughput and provide QoS.

7.2 Intel® RDT Allocation Software Flows for CPU Agents

RDT Allocation software flows for CPU agents are briefly described in this section to provide context for how and end-user may view the feature.

7.2.1 Intel® RDT Software Allocation Flows for CPU Agents

7.2.1.1 Step 1 – Enumeration

Before attempting to read or write MSRs associated with the Intel RDT Allocation feature software should first poll CPUID to ensure that Intel RDT Allocation and any sub-features to be used (for example, L3 CAT, L2 CAT, MBA) are supported on the platform, otherwise General Protection (#GP(0)) faults will be generated. As discussed in Section 3.2, if CPUID feature flag CPUID.0x7(Structured Extended Feature Flags).0:EBX[15] is set to '1' then Intel RDT Allocation is generally supported on the platform.

Once Intel RDT Allocation support has been verified software should poll and examine CPUID.0x10.0:EBX to examine which platform resources support allocation. After the call to CPUID, the EBX register will indicate the supported Intel RDT Allocation features on the current socket.

Software may use CPUID.0x10.ResID to determine the number of CLOS supported for the specific resource in question, the max length of the CAT bitmask, the max MBA delay value, and so on, and presence of sub-features like CDP on top of CAT for a given level of the cache. Software may optionally choose to build a record of these enumeration responses for each resource to reduce overhead from repeated CPUID calls.

Software should parse Processor Family, Model and Stepping (FMS) to verify that a particular processor includes support for a given model-specific feature. To find out which features are supported on which specific products, refer to Appendix A.3.

Note that it is important that software enumerate the Intel RDT Monitoring capabilities of the platform in the order specified in Section 3.1.1.

7.2.1.2 Step 2 – Optionally Enable CDP

If software wants to use CDP, enable it via the IA32_PQOS_CFG MSR.

7.2.1.3 Step 3 – Mask and Bandwidth Control Setup

After determining the presence of hardware Intel RDT Allocation support software should configure the CAT masks and MBA delay values if supported to provide capacity allocation/bandwidth hints to the hardware via the IA32_ResourceType_QOS_MASK_n MSRs and IA32_L2_QOS_Ext_BW_Thrtl_n

MSRs, depending on the usage model specified in Section 3.1.1 and the number of CLOS available (enumerated in feature-specific ResID sub-leaves).

It is considered good practice to first verify that IA32_L3_QOS_MASK_0 contains all "1" to the length of the bitmask (such that CLOS0 can access the entire cache) and that all threads are in CLOS0 before making changes to the masks (which may otherwise result in rapidly changing cache available to applications, which may lead to performance variation, though no functional errors are possible). Also verify that no bandwidth enforcement is configured in the IA32_L2_QOS_Ext_BW_Thrtl_n MSRs. It is also considered best practice to set up CLOS[0] as the highest priority CLOS with a large fraction of the cache, CLOS1 as the next highest, and so on.

7.2.1.4 Step 4 – CLOS Association

After the CAT/CDP per-CLOS mask MSRs are set up to known values, whether overlapped, shared or a combination depending on application needs and goals, and after MBA delay values are set up, each of the threads should be associated into a desired Class of Service via the IA32_PQR_ASSOC MSR. This MSR may be read or written at any time.

As part of some implementations an OS may choose to set up masks then change the IA32_PQR_ASSOC MSR on context switches (to associate a portion of the cache with an application or thread for instance).

7.3 Intel® RDT Software Flows for Non-CPU Agents

This section describes software architecture considerations for Intel RDT features for non-CPU agents, recommended usage flows and related considerations. This builds upon the architectural concepts and software usage examples discussed in Chapter 4.

Software seeking to use RDT for non-CPU agents has a number of tasks to comprehend:

- Enumeration of the capabilities of Intel RDT for CPU agents (through CPUID) and Intel RDT for non-CPU agents (through CPUID and ACPI).
- Reservation of (or comprehension of the sharing implications of using) RMIDs and CLOS from the pools available at each resource level and subject to the RMID and CLOS management best practices on a particular processor.
- Pre-configuration of any resource limits to be used for modulating device activity, such as a cache mask for a CLOS intended to be used with a device.
- Configuration of each device's tagging properties through the MMIO interface described by the ACPI structures, such as associating a device with a particular RMID, CLOS and bandwidth limit, as applicable.

- Enabling the Intel RDT features for non-CPU agents through the enable MSR infrastructure -- the IA32_L3_IO_QoS_CFG MSR is shown in Figure 4-2, at MSR address 0xC83.
- Periodically adjusting resource limits subject to software policies and any control loops which may be present.
- Comprehending the implications of Sub-NUMA clustering (SNC) if present and enabled.

A Intel® RDT Feature Details

A.1 Intel® RDT Feature Evolution

This section describes various generations of product and Intel RDT feature intercepts. Intel RDT provides a number of monitoring and control capabilities for shared resources in multiprocessor systems. This section covers updates to the feature that are available in current and future Intel processors, starting with brief descriptions followed by tables with details.

1. **Intel® RDT on the 3rd Gen Intel® Xeon® Scalable Processor Family.**
   The 3rd Gen Intel® Xeon® Scalable Processor Family, based on Ice Lake server microarchitecture, adds the following Intel RDT enhancements:
   - 32-bit MBM counters (versus 24-bit in prior generations), and new CPUID enumeration capabilities for counter width.
   - Second generation Memory Bandwidth Allocation (MBA): Introduces an advanced hardware feedback controller that operates at microsecond timescales, and software-selectable min/max throttling value resolution capabilities. Baseline descriptions of the MBA "throttling values" applied to the threads running on a core are described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.
     Second generation MBA capabilities also add a work-conserving feature in which applications that frequently access the L3 cache may be throttled by a lesser amount until they exceed the user-specified memory bandwidth usage threshold, enhancing system throughput and efficiency, in addition to adding more precise calibration and controls. Certain BIOS implementations may further aid flexibility by providing selectable calibration profiles for various usages.
   - 15 MBA / L3 CAT CLOS: Improved feature consistency and interface flexibility. The previous generation of processors supported 16 L3 CAT Class of Service tags (CLOS), but only 8 MBA CLOS. The changes in enumerated CLOS counts per-feature are enumerated in the processor as before, via CPUID.
2. **Intel® RDT on Intel Atom® Processors, Including the P5000 Series.**
   Intel Atom® processors, such as the P5000 series, based on Tremont microarchitecture add the following Intel RDT enhancements:
   - L2 CAT/CDP: L2 CAT/CDP and L3 CAT/CDP may be enabled simultaneously on supported processors. As these are existing features defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, no new software enabling should be required.
   - Supported processors match the capabilities of the 3rd Gen Intel Xeon Scalable Processor Family based on Ice Lake Server microarchitecture, including traditional Intel RDT uncore features: L3 CAT/CDP, CMT, MBM, and second-generation MBA. As these features are architectural, no new software enabling is required. Related enhancements in Intel Xeon processors also carry forward to

supported Intel Atom processors, with consistent software enabling. These features include 32-bit MBM counters, second generation MBA, and 15 MBA/L3 CAT CLOS.

3. **Intel® RDT in processors based on the 4th Gen Intel® Xeon® Scalable Processor Family.**
   Processors based on 4th Gen Intel® Xeon® Scalable Processor Family add the following Intel RDT enhancements:
   - STLB QoS: Model-specific capability to manage the second-level translation lookaside buffer structure within the core (STLB) in a manner quite similar to CAT (CLOS-based, with capacity masks). This may enable software that is sensitive to TLB performance to achieve better determinism. This is a model-specific feature due to the microarchitectural nature of the STLB structure. The code regions of interest should be manually accessed.

4. **Intel® RDT in Processors Based on 5th Gen Intel® Xeon® Processors.**
   Processors based on 5th Gen Intel® Xeon® Processors add the following Intel RDT enhancements:
   - L2 CAT and CDP: Includes control over the L2 cache and the ability to partition the L2 cache into separate code and data virtual caches. No new software enabling is required; this is the same architectural feature described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

5. **Future Intel® RDT.**
   Future processors add the following Intel RDT enhancements:
   - Third generation Memory Bandwidth Allocation (MBA): new per-logical-processor capability for bandwidth control (rather than the more coarse-grained core-level throttling value resolution in prior generations). This capability enables more precise bandwidth shaping and noisy neighbor control. Some portions of the control infrastructure now operate at core frequencies for controls that are responsive at the nanosecond level.
   - Intel® RDT features support for non-CPU agents, enabling advanced monitoring and control capabilities for PCIe and CXL devices, as well as integrated processor accelerators.

A.2 Intel® RDT Architectural Features and Supported Products

	Intel RDT Feature Category	Shared Resource	Agent	Intel RDT Sub-Feature	Intel RDT Scope	Supported Products
Monitoring	Cache Monitoring Technology (CMT)	L3	CPU	L3 CMT for CPU agents	Per-thread RMID-based	Intel® Xeon® E5/E7 v3,v4, Intel® Xeon® D, Intel® Xeon® Scalable Processor, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable processor, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
		L3	I/O	L3 CMT for non-CPU agents	Per-agent RMID-based	Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
	Memory Bandwidth Monitoring (MBM)	-	CPU	MBM Local for CPU agents	Per-thread RMID-based	Intel® Xeon® E5/E7 v4, Intel® Xeon® D, Intel® Xeon® Scalable Processor, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable processor, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
			CPU	MBM Total for CPU agents	Per-thread RMID-based	Intel® Xeon® E5/E7 v4, Intel® Xeon® D, Intel® Xeon® Scalable Processor, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable processor, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series (Selected Processors), Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
			I/O	MBM Local for non-CPU agents	Per-agent RMID-based	Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
			I/O	MBM Total for non-CPU agents	Per-agent RMID-based	Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)

	Intel RDT Feature Category	Shared Resource	Agent	Intel RDT Sub-Feature	Intel RDT Scope	Supported Products
Allocation	Cache Allocation Technology (CAT)	L2	CPU	L2 CAT for CPU agents	Per-thread CLOS-based	Atom Server C3000, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® Scalable processor (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
				L2 CDP for CPU agents	Per-thread CLOS-based	5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® Scalable processor(codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
		L3	CPU	L3 CAT for CPU agents	Per-thread CLOS-based	Intel Atom® X Series (Selected Processors), Intel® Xeon® E5/E7 v3 (Selected Processors), Intel® Xeon® E5/E7 v4 , Intel® Xeon® D, Intel® Xeon® Scalable, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable processor, 5th Gen Intel® Xeon® Scalable processor, Intel® Xeon® W, Intel Atom® Processor P5000 Series, Intel® Xeon® Scalable processor(codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
				L3 CDP for CPU agents	Per-thread CLOS-based	Intel® Xeon® E5/E7 v4, Intel® Xeon® Scalable Processor, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® Scalable processor(codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
			I/O	L3 CAT for non-CPU agents	Per-agent CLOS-based	Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
	Memory Bandwidth Allocation (MBA)	-	CPU	MBA for CPU agents (Second Generation MBA)	Per-thread CLOS-based	Intel® Xeon® Scalable Processor, 2nd Gen Intel® Xeon® Scalable Processor, 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable processor, 5th Gen Intel® Xeon® Scalable processor, Intel Atom® Processor P5000 Series, Intel® Xeon® Scalable processor(codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
			CPU	MBA for CPU agents (Third Generation MBA)	Per-thread CLOS-based	Intel® Xeon® processors (codename Granite Rapids), Intel® Xeon® processors (codename Sierra Forest)
	Cache Bandwidth Allocation (CBA)	-	CPU	CBA for CPU agents	Per-Logical Processor based	Future Intel® Processors

A.3 Intel® RDT Model-Specific Features and Supported Products

Intel RDT Feature Category	Supported Products
Resource Aware MBA (MBA4.0)	• Intel® Xeon® processors (codename Granite Rapids). • Intel® Xeon® processors (codename Sierra Forest).
Intel® RDT and Sub-NUMA Clustering (SNC) Compatibility	• 3rd Gen Intel® Xeon® processors. • 4th Gen Intel® Xeon® processors. • 5th Gen Intel® Xeon® processors.
STLB QoS	4th Gen Intel® Xeon® processors. The following product generations on SKUs with Intel® Time Coordinated Computing (Intel® TCC) support: • 11th Gen Intel® Core™ Processors (UP3-Series). • Intel® Xeon® W Processors (TGL-H). • 12th Gen Intel® Core™ Processors (S-Series). • 13th Gen Intel® Core™ Processors (P-Series). • 13 Gen Intel® Core™ Processors (S-Series). • Intel Atom® x7000E Series Processors.

A.4 Feature Mapping: CPU Agents, Non-CPU Agents in Different L3 Configurations

Configuration	Intel RDT Feature	CPU Agents Intel RDT Scope	Non-CPU Agents Intel RDT Scope	Comments
Shared-L3	Cache Monitoring Technology (CMT)	Per-thread RMID-based	Per-agent RMID-based	Unified per-RMID counters across CPU Agents and non-CPU Agents.
Shared-L3	Memory Bandwidth Monitoring (MBM)	Per-thread RMID-based	Per-agent RMID-based	Unified per-RMID counters across CPU Agents and non-CPU Agents.
Shared-L3	Cache Allocation Technology (CAT)	Per-thread CLOS-based	Per-agent CLOS-based	Unified per-CLOS controls across CPU Agents and non-CPU Agents.
Shared-L3	Code and Data Prioritization (CDP)	Per-thread CLOS-based	N/A	CDP is not supported for non-CPU Agents.
Shared-L3	Memory Bandwidth Allocation (MBA)	Per-thread MBA throttling (MBA3.0 and higher) or Per-core MBA throttling (MBA1.0-2.0)	N/A	MBA is not supported for non-CPU Agents.

A.5 Architectural MSRs used with Intel® RDT Features

The following architectural Model-Specific Registers are used with Intel® RDT features.

MSR Name	Comments
IA32_PQR_ASSOC	Set the RMID and CLOS pair.
IA32_QM_EVTSEL	Set event codes and RMID to be monitored.
IA32_QM_CTR	Reports monitoring telemetry data.
IA32_L3_MASK_n	Bitmask to assign L3 cache ways for each CLOS. "n" registers, one register per CLOS.
IA32_L2_QoS_Ext_BW_Thrtl_n	Set valid throttling levels. "n" registers, one register per CLOS
IA32_L2_QOS_MASK_n	Bitmask to assign L2 cache ways for each CLOS. "n" registers, one register per CLOS.
IA32_L3_IO_QOS_CFG	Set to enable Allocation and Monitoring for non-CPU Agents
IA32_QoS_Core_BW_Thrtl_n	Set valid throttling levels, one byte per CLOS. "n = 0 to (((CLOS_MAX+1)/8) -1)" registers

A.6 Model-Specific Registers for Intel® RDT Model Specific Features

The following notable non-architectural Model-Specific Registers are used with Intel® RDT features and will be expanded over time. Others are discussed in preceding model-specific chapters.

MSR Name	Comments
MBA_CFG	Set the RMID and CLOS pair.
RMID_SNC_CONFIG	Clear to enable RMID Sharing Mode.
STLB_QOS_INFO	Discover STLB QOS parameters
STLB_QOS_MASK_N	STLB QOS Capacity Bitmasks
STLB_FILL_TRANSLATION	Fill a logical address into the STLB
PQR_ASSOC	Resource Association Register
L3_QOS_MASK_N	L3 Class of Service Mask

B Model-Specific Intel® RDT Features

B.1 Model-Specific Intel® RDT Features for CPU Agents

This section gives an overview of non-architectural features that are implemented on specific products. To find out which features are supported on which specific products, refer to Appendix A.3.

In certain cases, model-specific features may be implemented rather than architectural features in cases where the cache or memory hierarchies are rapidly evolving, or in cases where usages are specialized and require intricate software enabling and tuning, or in cases where a subset of special-purpose processors are enabled with certain features within a broader product line.

Support for a certain model-specific feature in a particular product generation does not imply that future products will support the same model-specific feature; furthermore, this does not guarantee software forward-compatibility. Software should use Processor Family, Model and Stepping (FMS) to verify that a particular processor includes support for a given model-specific feature.

B.1.1 Resource Aware MBA

Resource Aware MBA (MBA 4.0) for CPU-agent was formerly known as Fourth Generation MBA (MBA 4.0) which supports over Third Generation MBA capabilities as Bandwidth management support is implemented to support up to three different resources – DDR Memory, CXL links, and UPI Links on a pre thread basis. Third generation MBA capabilities (see Section 3.2.3.3) are the default mode of operation, with Resource Aware MBA being opt-in. See Appendix A.3 for Resource Aware MBA feature intercept details.

B.1.1.1 Overview

Resource Aware MBA allows per-thread tracking and control of Bandwidth to different resources – that is, enabling bandwidth control per-thread and per-resource simultaneously. As in the third generation of MBA, each resource and thread are managed by a hardware controller which modulates the bandwidth of each thread targeting a particular downstream resource around a bandwidth target set by Intel RDT software interfaces.

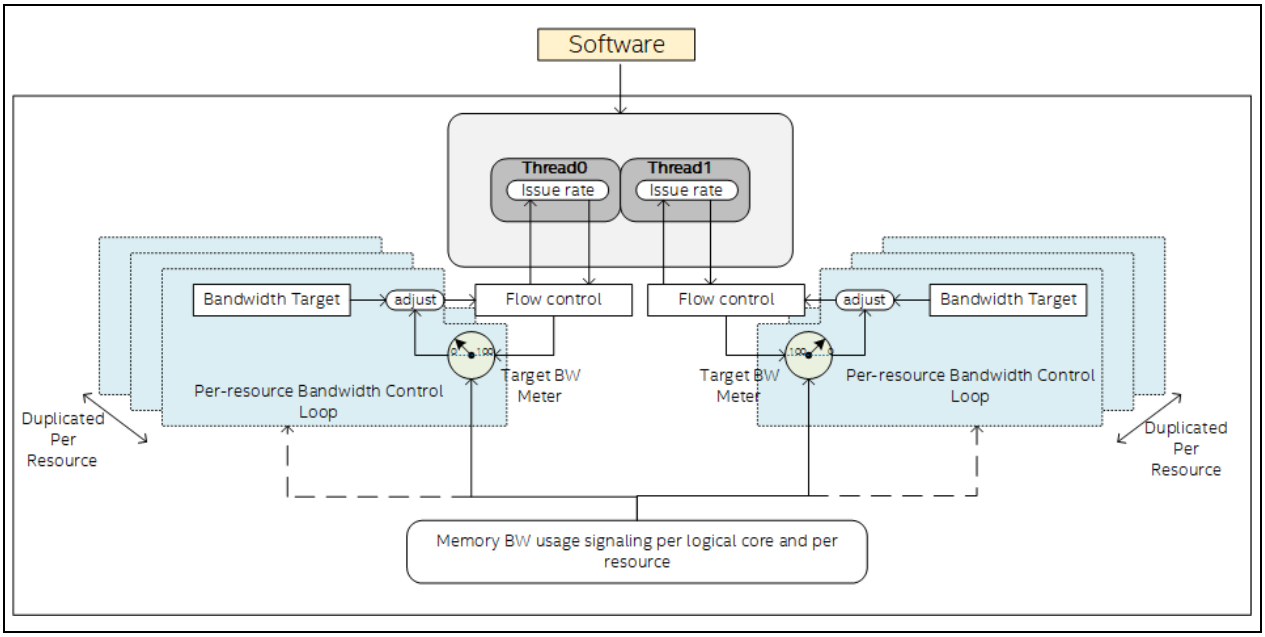
The resource types that are managed are:

1. DDR – All traffic towards DDR Memory regardless of location of location (local, remote or CXL).
2. CXL – All traffic towards CXL resources such as CXL.mem pools including remote.

3. UPI - All traffic that utilizes the Intel® Ultra Path Interconnect (Intel® UPI) link(s) for cross socket data transfer regardless of target on the remote socket.

The high-level implementation of Resource Aware MBA is shown in Figure B-1.

Figure B-1. High-Level Overview of the Resource Aware MBA (MBA 4.0)



B.1.1.2 Enable MSR

Resource Aware MBA (MBA 4.0) is opt-in feature. Before configuring MBA throttling values per-thread and per-resource, the feature should be enabled (through a configuration MSR). The MBA_CFG MSR is used to enable the Resource Aware MBA feature for CPU agents.

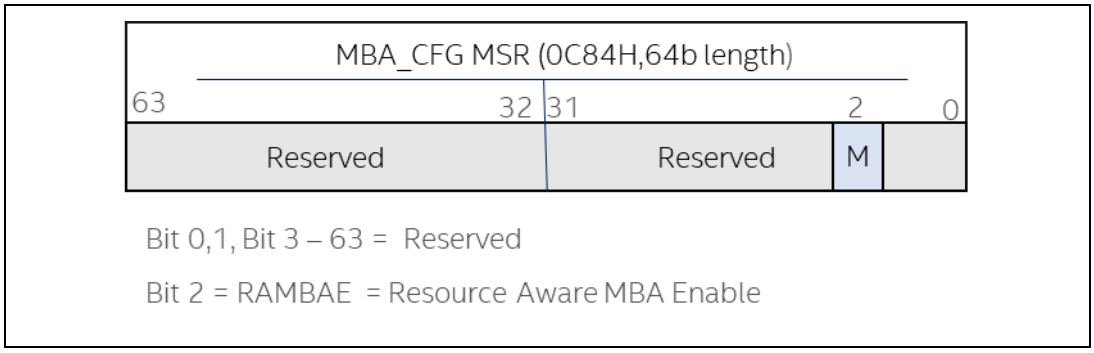
One bit is defined in this MBA_CFG MSR, bit[2], which when set enables the Resource Aware MBA feature and switches between third-generation MBA and Resource Aware MBA modes.

The default value is 0x0 (Resource Aware MBA is disabled by default), and all bits not defined are reserved. Any writes to reserved bits will generate a General Protection Fault (#GP(0)).

This MSR is scoped at the die level and is cleared on system reset. It is expected that software will configure this MSR consistently across all L3 caches that may be present in the SoC.

The definition of the MBA_CFG MSR is shown in Figure B-2, and its MSR address is 0xC84.

Figure B-2. The MBA_CFG MSR for Enabling Resource Aware MBA Feature



Reference BIOS implementations supporting Resource Aware MBA will extend the legacy bandwidth profile knobs from Second Generation MBA with a drop-down menu of three options (see Section 5.2 for details)

B.1.2 Intel® RDT and Sub-NUMA Clustering Compatibility

The following sub-sections describe Intel RDT and Sub-NUMA Clustering (SNC) compatibility enabling components. Utilizing SNC and RDT simultaneously may provide resource contention isolation benefits but requires incremental software enabling with the introduction of SNC.

B.1.2.1 Introduction

Following sub-sections describe Intel RDT monitoring features behavior in the presence of either multiple NUMA domains per socket, other product implementations in which multiple NUMA domains may appear per processor, due to either logical or physical resource partitioning. This section references Intel RDT features such as MBA, MBM, CMT and CAT for CPU agents and non-CPU agents described in Chapter 3 and Chapter 4 respectively.

The Sub-NUMA Clustering (SNC) feature creates localization domains within a processor by mapping addresses from a local memory controller to a subset of the L3 slices that are at a reduced distance to nearby memory controller(s), reducing latency, and increasing equivalent traffic isolation across memory channels controllers.

MBA usage is not affected in presence SNC; bandwidth targets apply globally across all SNC domains. L3 CAT and Monitoring features (L3 CMT and MBM) usage is affected in the presence of SNC. Following sections provide details. See Appendix A.3 for Intel RDT and Sub-Numa Clustering (SNC) Compatibility feature supported product details (for example, products where the features are simultaneously supported).

B.1.2.2 SNC Enabled and L3 Cache Allocation Technology

L3 Cache Allocation Technology (L3 CAT) allows an Operating System (OS), Hypervisor / Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of L3 cache capacity of the Resource Allocation Domain (RAD) into which an application can fill.

In the presence of SNC, cache capacity bitmasks are still die-scoped and apply across multiple-L3 domains. Each bit in the cache capacity bitmask manages all clusters and dictates the portion of each SNC cluster available for a given Resource Management Domain. For example, each bit in cache capacity bitmask represents half as much L3 cache capacity at each cluster when SNC2 is enabled, or one-quarter as much L3 cache capacity at each cluster when SNC4 is enabled and so on. Note that total L3 cache capacity does not change.

Software may choose to apply consistent policies across SNC domains utilizing this property, such as CLOS[0] having full access to the cache across any SNC domain in which it may run, but CLOS[1] having access to only half of the cache, implying that it contains a set of lower-priority threads.

B.1.2.3 SNC Enabled and RMID Distribution Modes

There are two modes available to control Resource Monitoring ID (RMID) distribution when SNC is enabled: Default mode and RMID Sharing.

Software should consider and select the mode in which RMIDs are distributed or shared across the SoC and SNC domains depending on its usage needs.

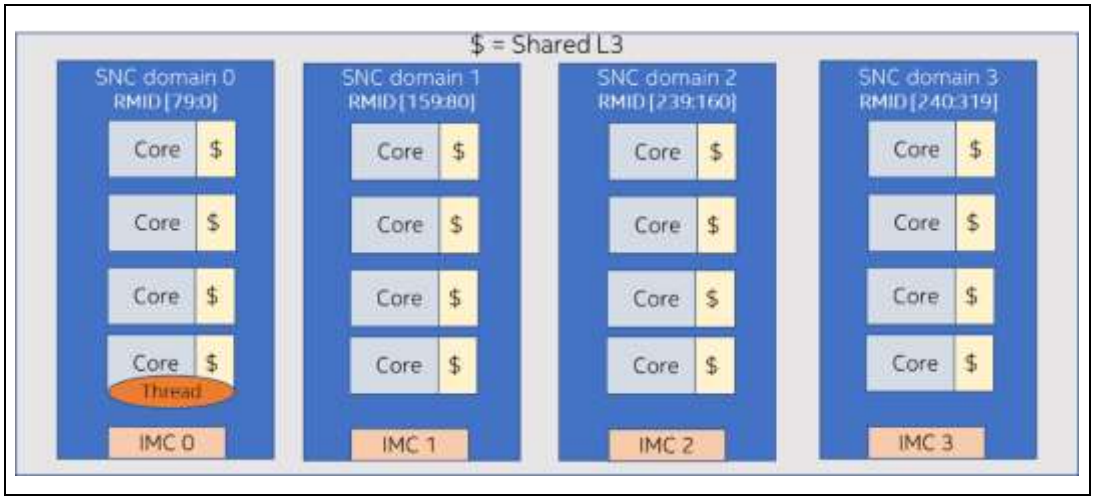
B.1.2.3.1 Default Mode

When SNC is enabled the available pool of RMIDS are distributed across all the L3 slices. RMIDs are distributed across the cores in the same fashion as done when SNC is not enabled, see Figure B-3.

This distribution scheme allows the RMIDS enumerated by CPUID to be directly used. Software should be aware of the distribution of RMIDs between the SNC domains. For instance, if there are 320 RMIDs available (enumerated via CPUID.(EAX=0FH(Shared Resource Monitoring Enumeration leaf), ECX=0H)) and an SNC-4 configuration is selected, four localization domains exist within a processor.

These 320 RMIDs can be dived into four groups of 80 RMIDS with first 80 allocated to SNC domain 0, the next 80 to SNC domain 1 and so forth. Due to this distribution policy, RMIDs may be visualized as localized to SNC domains, and there maybe cases where bandwidth is not counted. Consider for instance the case where thread with RMID 0 accesses will generate counts only for traffic in SNC domain 0. Any traffic from this thread that accesses other SNC domains will not increment any of the other counters. In other words, each SNC domain will get an equal number of distinct RMIDS from the global pool of RMIDS that are not shared.

Figure B-3. Default Mode Demonstrating SNC-4 and RMID Distribution

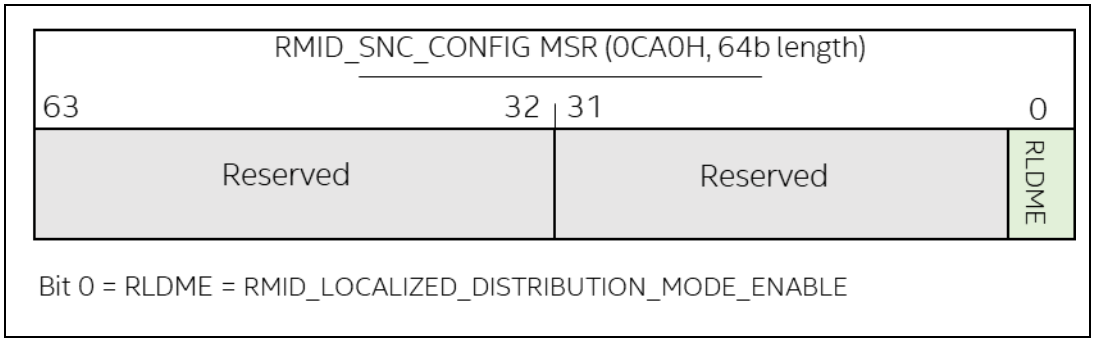


B.1.2.3.2 RMID Sharing Mode

RMID sharing mode allows the same RMID to be distributed with traffic accessing any and all SNC domains, but at the cost of a reduced number of SoC-level RMIDs available. This model-specific mode aims to mitigate the disadvantage of the Default mode where software should be aware of the RMID distribution per SNC domain (and NUMA-aware) and where traffic tagged with an RMID in one domain will not be counted if it accesses resources in another SNC domain. RMID sharing mode allows same RMID to sample across SNC domains, thus ensuring a complete count.

- This is an opt-in mode and requires that the software clears an enable bit defined in the following MSR 0XCA0, bit[0], see Figure B-4. Note that as a model-specific capability, this mode is not guaranteed to be supported on all processors (see Appendix A.3 for support details).

Figure B-4. The RMID_SNC_CONFIG MSR for Enabling RMID Sharing Mode



In this mode the number of RMIDs are distributed across all the L3 slices effectively reducing the number of RMIDs by the number of SNC domains. In the case of four SNC domains, the number of RMIDs are divided by four.

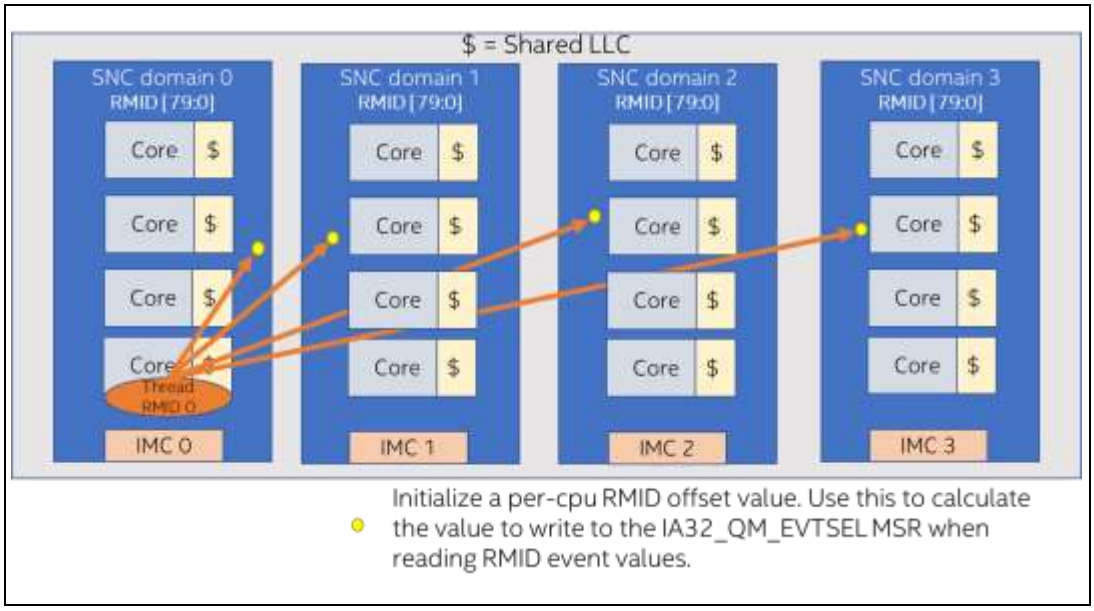
Number of valid RMIDs = (Highest RMID value/#SNC_clusters)

Using the previous example of 320 RMIDs, in this mode with SNC-2 enabled there would be (320/2), that is, 160 RMIDs, with SNC-4 enabled there would be (320/4), that is, 80 RMIDs.

Note: In SNC4 mode, to determine the count for RMID0, the count for RMID0, RMID80, RMID160, and RMID240 should be read and added to provide the total count for RMID0.

Note: It is the responsibility of software to read the values from each of the counters and calculate and interpret the sum using the output of the IA32_QM_CTR MSR. This is illustrated in Figure B-5.

Figure B-5. RMID Sharing Mode Demonstrating SNC-4 and RMID Distribution



B.1.2.4 Intel® RDT Software Considerations

Depending on its preferred use model and whether this model-specific capability is supported on a particular processor, software may select either the mode in which RMIDs are distributed or shared across the SoC and SNC domains. The default mode where each SNC cluster has a defined group of RMIDs or the opt-in mode which shares the same RMID across the SNC domains.

- Without SNC mode enabled the Remote Memory Bandwidth can be calculated by:
  - Remote Memory BW = (Total Memory BW – Local Memory BW) * Scaling Factor.
- With SNC Mode enabled software should scale the measured BW depending on the SNC_RMID Mode.
- CMT is similarly affected.

Table B-1. SNC Enabled and RMID Distribution Mode Summary

	Default Mode	Opt-In : RMID Sharing Mode
Key highlights	• RMID_SNC_CONFIG MSR is Set. • Each SNC domain has its own group of RMIDs.	• RMID_SNC_CONFIG MSR is Clear. • Number of RMIDs divided by the number of SNC Domains. • Opt-In mode is enabled by software setting the MSR 0xCA0[0] = 0.
Example: **RMID Distribution per SNC Example for each Mode: SNC-4 config and Max 320 RMIDs**	1. For each SNC domain, the software should select an RMID from the range mentioned next to program IA32_PQR_ASSOC MSR. This range will be dependent on NUMA cluster you choose: • SNC_Domain_0 : RMID[79:0] • SNC_Domain_1 : RMID[159: 80] • SNC_Domain_2 : RMID[239:160] • SNC_Domain_3 : RMID[319:240] 2. To obtain monitoring data read via IA32_QM_EVTSEL, MSR uses only the RMID value to read counter value.	1. Number of Valid RMIDs = (#RMIDS/#SNC_Domains). Choose d in {0...79} in this example. **This range is used to program RMID field in the IA32_PQR_ASSOC MSR so that the appropriate hardware counters within the SNC domain are updated. 2. To obtain monitoring data via IA32_QM_EVTSEL MSR read 4 counter value from using the next formula: MAX_VALID_RMID = #RMIDS/#SNC_DOMAINS SNC_DOMAIN_0: RMID[0+d] SNC_DOMAIN_1: RMID[MAX_VALID_RMID*1 + d] SNC_DOMAIN_2: RMID[MAX_VALID_RMID*2 + d] SNC_DOMAIN_3: RMID[MAX_VALID_RMID*3 + d] For this example: SNC_DOMAIN_0: RMID[0+d] SNC_DOMAIN_1: RMID[80+d] SNC_DOMAIN_2: RMID[160+d] SNC_DOMAIN_3: RMID[240+d]
Differences	• Same number of RMIDS across SoC.	• RMIDS divided down by the number of SNC Domains and hence reduced number of RMIDS available for use.
Differences	• Miss traffic count due to software that traverses SNC domains. This can lead to inaccurate counts for CMT/MBM.	• Counts traffic that traverses SNC domains.
Differences	• Software needs to know the distribution of RMIDS to SNC domains.	• Software required to read all the RMID counters in the SNC domains and add up the individual count to get the final count.

Note: Only the monitoring features of Intel RDT are affected by the SNC feature. The allocation features, that is, CAT and MBA are not affected. Bit masks and BW targets apply globally across all domains. See Table B-1 for SNC enabled and RMID distribution summary.

B.1.2.5 Scaling Factor Adjustment

CPUID-provided scaling factor (CPUID(0xF(Shared Resource Monitoring Enumeration leaf).0x1).EBX[31:0]), which software will use to convert MBM counts into bandwidth figures, needs adjustment in software when the system is configured in SNC mode. Moreover, calculating different types of bandwidths, such as local, total, or remote, also needs special considerations. This section describes how software needs to handle these special cases.

When using scaling factor under SNC mode, the scaling factor provided by CPUID will not account for the reduced number of L3 slices that will be handling local traffic. The scaling factor value will remain the same as any other clustering mode. software will then need to adjust the scaling factor. For this purpose, we define:

AdjustedScalingFactor = ScalingFactor / SNCClusterCount

B.1.2.6 SNC and Intel® RDT for Non-CPU Agent Implications

Intel RDT for non-CPU agents is affected similarly to traditional Intel RDT features in the presence of SNC. To obtain a correct CMT or MBM data sampling software should either localize I/O device memory allocations to a given cluster or sum RMID counts periodically, depending on the RMID localization mode selected.

In cases where multiple contexts are present on a device (SR-IOV, SIOV, with attached VMs that may span multiple SNC domains for their execution or multiple devices are behind an IOSF channel, if memory accesses are distributed across SNC clusters, then monitoring accuracy decreases considerably, and the risk of missing cache occupancy or memory bandwidth increases considerably.

SNC also affects I/O traffic. Software seeking to monitor I/O capacity or overflow BW to memory (I/O equivalent of CMT or MBM), should determine which SNC cluster a given address falls into using NUMA-aware supporting constructs (for example, ACPI HMAT, SLIT tables [4]) and pick a corresponding RMID for that cluster. As an example, if a device DMA write assigned to an RMID which does not land in the same SNC cluster as the address and its memory controller will not be tracked.

B.1.2.7 Calculating Local MBM Bandwidth per Cluster

When MSR 0xCA0 is set to 1 (Default Mode) software will be able to monitor local BW only from one SNC cluster. If MSR 0xCA0 is set to 0 (RMID Sharing Mode) then software will be able to monitor Local BW from all SNC clusters. Independent of the value in MSR 0xCA0, Local MBM Counts from a given SNC cluster can be converted to BW figures using the adjusted scaling factor following the same mechanism used under non-SNC modes:

*LocalMbmBwClusterN = (LocalMbmCountDeltaClusterN * AdjustedScalingFactor) / SampleTime*

Where:

- ‘LocalMbmCountDeltaClusterN” = (Second Sample of LocalMbmCounter value (ClusterN) – First sample of LocalMbmCounter value (ClusterN).
- ‘AdjustedScalingFactor’ = ScalingFactor / SNCClusterCount.

B.1.2.8 Calculating Local MBM Bandwidth for Entire Socket

While operating under any non-SNC mode Local MBM BW will correspond to all the total traffic within the full socket. To obtain the same metric under SNC

mode software may add up the Local BW from each cluster. This can be achieved only when MSR 0xCA0 is set to 0. Otherwise, software will only be able to capture the local BW from a single cluster.

*LocalMbmBwSocket = ((LocalMbmCountDeltaCluster0 + ... LocalMbmCountDeltaClusterN) * AdjustedScalingFactor) / SampleTime*

Where:

- 'LocalMbmCountDeltaCluster0" = (Second Sample of LocalMbmCounter value (Cluster0) – First Sample of LocalMbmCounter value (Cluster0)… Similarly, delta for for each 1,2,…N.
- 'AdjustedScalingFactor' = ScalingFactor / SNCClusterCount.

B.1.2.9 Calculating Total MBM Bandwidth for the Socket

Calculating the Total MBM BW for the full socket, including the traffic from all clusters, will require that MSR 0xCA0 is set to 0.

*TotalMbmBwSocket = ((TotalMbmCountDeltaCluster0 + ... TotalMbmCountDeltaClusterN) * AdjustedScalingFactor) / SampleTime*

Where:

- 'TotalMbmCountDeltaCluster0" = (Second Sample of TotalMbmCounter value (Cluster0) – First Sample of TotalMbmCounter value (Cluster0)… Similarly, delta for each 1,2,…N.
- 'AdjustedScalingFactor' = ScalingFactor / SNCClusterCount.

B.1.2.10 Estimating Remote Traffic

As with Non-SNC modes, remote traffic can be estimated out of the socket's Total MBM BW and Local MBM BW with this simple relation:

RemoteMbmBwSocket = TotalMbmBwSocket – LocalMbmBwSocket

Calculating both TotalMbmBwSocket and LocalMbmBwSocket will require MSR 0xCA0 to be set to 0. However, if software decides to keep MSR 0xCA0 set to "1", its default value, an alternative mechanism exists to calculate the socket's MBM Remove BW as described in the following section.

B.1.2.11 Estimating Remote Bandwidth with MSR 0xCA0 set to 1

If software decides not to switch MSR 0xCA0 to value 0 (for example, out of Default mode) the mechanism described earlier to calculate the socket remote traffic will not work. However, it is still possible to estimate the remote traffic of the entire socket by using MBM counts from a single cluster.

*RemoteMbmBwSocket = (TotalMbmBwClusterN - LocalMbmBwClusterN) * SNCClusterCount*

B.1.2.12 Example for Local and Total MBM Bandwidth

In this example, software runs on a system configured in SNC-4 mode where CPUID(0xF(Shared Resource Monitoring Enumeration leaf).0x1).EBX[31:0]

reads 0x1E000 (ScalingFactor). AdjustedScalingFactor is then calculated to be 0x7800. If the system is configured with MSR 0xCA0=0 (RMID Distribution Mode) then software will have the ability to sample BW across all four clusters in this example. After sampling MBM counts with a delay of one second the following MBM Count increments are observed:

Table B-2. Local and Total Count Increment

Cluster	Local MBM Count Increment	Total MBM Count Increment
0	174762	192238
1	43690	61166
2	0	17476
3	0	17476

Software can then calculate Local Bandwidth (BW), Total Bandwidth(BW) and Remote Bandwidth(BW) following these steps.

1. Calculate Local BW for Cluster 0 using the formula for LocalMbmBw described earlier:
   LocalMbmBwCluster0 = (LocalMbmCountDeltaCluster0 * AdjustedScalingFactor) / SampleTime
   LocalMbmBwCluster0 = (174762 * 0x7800) / 1
   LocalMbmBwCluster0 = 5368688640 B/s ~= 5GB/s
2. Calculate Total BW for Cluster 0 using the formula for TotalMbmBw described earlier:
   TotalMbmBwCluster0 = (TotalMbmCountDeltaCluster0 * AdjustedScalingFactor) / SampleTime
   TotalMbmBwCluster0 = (192238* 0x7800) / 1
   TotalMbmBwCluster0 = 5905551360B/s ~= 5.5GB/s
3. Following the same procedure Local and Total BWs for the different clusters may be calculated as shown in Table B-3.

Table B-3. Local and Total Bandwidth Example

Cluster	LocalMbmBwClusterN	TotalMbmBwClusterN
0	5 GB/s	5.5 GB/s
1	1.25 GB/s	1.75 GB/s
2	0	0.5 GB/s
	0	0.5 GB/s

4. We can also calculate the socket Local and Total BWs:
   LocalMbmBwSocket =((LocalMbmCountDeltaCluster0 + ... LocalMbmCountDeltaClusterN) * AdjustedScalingFactor) / SampleTime
   LocalMbmBwSocket = ((174762 + 43690 + 0 + 0) * 0x7800) / 1
   LocalMbmBwSocket = 6710845440B/s ~= 6.25GB/s
   TotalMbmBwSocket =((TotalMbmCountDeltaCluster0 + ... TotalMbmCountDeltaClusterN) * AdjustedScalingFactor) / SampleTime

TotalMbmBwSocket = ((192238 + 61166 + 17476 + 17476) * 0x7800) / 1
TotalMbmBwSocket = 8858296320B/s ~= 8.25GB/s

5. Finally, the remote BW for the socket can be estimated:
   RemoteMbmBwSocket = TotalMbmBwSocket – LocalMbmBwSocket
   RemoteMbmBwSocket = 8.25GB/s - 6.25GB/s ~= 2GB/s

We can also use this example to show to estimate the socket's remote BW if MSR 0xCA0 is set to 1 (Default mode). Under such conditions only MBM counts from a single cluster can be obtained. Assuming that the software has picked and RMID from cluster 0, we can use the values calculated earlier for LocalMbmBwCluster0 and TotalMbmBwCluster0. Then:

RemoteMbmBwSocket = (TotalMbmBwCluster0-LocalMbmBwCluster0) * SNCClusterCount
RemoteMbmBwSocket = (5.5GB/s – 5.0GB/s) * 4 ~= 2GB/s

Note that the value for RemoteMbmBwSocket obtained through this mechanism matches that obtained by using the MBM counts from all clusters.

By analyzing the results from this example, we can conclude, from the thread or threads assigned to the selected RMID that:

- Thread(s) are generating 5 GB/s of traffic towards cluster 0.
- Thread(s) are generating 1.25 GB/s of traffic towards cluster 1.
- Thread(s) are not generating local traffic towards clusters 2 or 3.
- Thread(s) are generating 2 GB/s of traffic towards a remote socket.
- Each SNC cluster is handling 0.5 GB/s of that remote traffic.

B.1.3 STLB QoS

Translation Lookaside Buffer (TLB) misses can cause costly execution delays due to page walks. Considered from a capacity management perspective, STLB QoS behaves in a similar manner as Cache Allocation Technology (CAT) does on the data caches, by giving software the ability to provide hints to hardware that guide the placement of translations in the STLB. This control can provide fair sharing or improved isolation of TLB resources between applications organized by Classes of Service.

Note: This model-specific feature is intended for use primarily with specialized real-time operating systems that provide extensions to bound the number of tasks running on a core and thus sharing a TLB. Depending on the software environment, additional runtime restrictions and software optimizations may be needed to observe the potential performance benefits of STLB QoS. Contact your Intel representative for additional details.

Refer to Appendix A.3 for supported product details, which vary across generation and processor type.

B.1.3.1 Enumerating Support for STLB QoS

STLB QoS is model specific and support for it is enumerated through the IA32_CORE_CAPABILITIES MSR. To determine if the processor supports the IA32_CORE_CAPABILITIES MSR, software can check whether the CPUID Extended Feature flag at CPUID.0x7(Structured Extended Feature Flags).0:EDX[30] is set to '1'.

If CPUID.0x7(Structured Extended Feature Flags).0:EDX[30] is '1', then support for STLB QoS can be confirmed via the IA32_CORE_CAPABILITIES MSR as defined next.

Table B-4. STLB QoS Enumeration in IA32_CORE_CAPABILITIES MSR

Name	Address	Scope	Bit	RW	Bit Name	Description
IA32_CORE_CAPABILITIES	CFh	Core	0	RO	STLB_QOS	When set to 1, processor supports STLB QoS

B.1.3.2 STLB QoS Register Interfaces

This section contains the register interfaces for configuring STLB QoS. Software should first read the STLB_QOS_INFO to determine the maximum number of classes of service and capacity bitmask length and may then proceed to partitioning the STLB using the STLB_QOS_MASK_n registers.

B.1.3.2.1 STLB_QOS_INFO

Software may discover the necessary information for configuring STLB QoS via the STLB_QOS_INFO MSR as defined next.

Table B-5. STLB_QOS_INFO MSR Definition

Name	Address	Scope	Bit	RW	Bit Name	Description
STLB_QOS_INFO	1A8Fh	Core	5:0	RO	NCLOS	Number of CLOS supported for STLB resource using minus-1 notation.
			19:16	RO	4K_2M_CBM	Length of capacity bitmask for 4K and 2M pages using minus-1 notation.
			29:29	RO	STLB_FILL_ TRANSLATION _MSR_SUPPORTED	MSR interface to fill STLB translations supported.
			30:30	RO	4K_2M_ALIAS	Indicates that 4K/2M pages alias into the same structure.

B.1.3.2.2 STLB_QOS_MASK_N

STLB_QOS_MASK_n registers define the capacity bitmask to be applied when filling new translations into the STLB. The mask used will depend on the core's current Class of Service at the time of TLB miss, as configured via the IA32_PQR_ASSOC MSR (covered in Chapter 3.2 Intel RDT Allocation Common Framework). The STLB_QOS_MASK_n registers are dynamic and may be changed at runtime.

Software should limit the number of mask registers used to the number of supported STLB QoS CLOS. For example, if STLB_QOS_INFO[NCLOS] returns 0x7, then a total of eight classes of service are supported and valid STLB_QOS_MASK_n registers would be 1A90h – 1A97h as defined in Table B-6. Attempts to use unsupported STLB QoS mask registers will generate #GP(0).

Table B-6. STLB_QOS_MASK_N MSR Definition

Name	Address	Scope	Bit	RW	Bit Name	Description
STLB_QOS_MASK_n	1A90h - 1A9Fh	Core	7:0	RW	WAY_MASK	STLB QoS mask for CLOS n. The number of mask bits is enumerated in MSR STLB_QOS_INFO. '1 in bit indicates allocation to the way is allowed. '0 indicates allocation to the way i' not allowed.[1,2]

NOTES: 1. Mask values must be contiguous 1s.
2. Way mask only applies to 4K/2M STLB.

B.1.3.2.3 STLB_FILL_TRANSLATION

As a further specialized extension to STLB QoS, certain processors support a mechanism to manually populate entries in the STLB, rather than requiring that pages of interest be accessed by software as part of a TLB fill flow to populate the entries.

If STLB_QOS_INFO[STLB_FILL_TRANSLATION_MSR_SUPPORTED] is '1', software may populate entries in the STLB directly by writing the logical address (LA) and Class of Service to use for the fill to STLB_FILL_TRANSLATION as defined next.

Table B-7. STLB_FILL_TRANSLATION MSR Definition

Name	Address	Scope	Bit	RW	Bit Name	Description
STLB_FILL_ TRANSLATION	1A8Eh	Core	3:0	WO	CLOS	Class of service to use for the fill.
			10:10	WO	X	Set to 1 when LA is to an executable page.
			11:11	WO	RW	Set to 1 when LA is to a writeable page.
			63:12	WO	LA	Logical address to use for fill.

Note: The STLB_FILL_TRANSLATION MSR should not be used in the VMX load list as a #GP(0) will occur.

B.1.4 L3 Cache Allocation Technology

Certain Intel® Core™ and Intel Atom® processors with support for Intel® Time Coordinated Computing (Intel® TCC), and certain communications related Intel® Xeon® processors implement a model specific, non-architectural version

of L3 Cache Allocation Technology (L3 CAT). In model-specific implementations, parameters such as CBM bitmask length and number of supported CLOS are specified on a per-processor basis rather than in CPUID (see the following section).

The non-architectural implementations of L3 CAT behave similarly to the architectural implementation, however under certain circumstances the performance characteristics may vary. Intel recommends evaluating overall system performance with model-specific non-architectural L3 CAT to verify performance targets are met.

B.1.4.1 Processor Support List

The following table can be used to identify which processors support the model specific non-architectural implementation of L3 CAT. Registers for programming the capacity bitmask for a given CLOS follow the same location and definition of the IA32_L3_MASK_n MSR's as defined in the Intel® Software Developer's Manual.

Table B-8. Processor support list

Processor	Brand String	# L3 Classes of Service (CLOS)	Capacity Bitmask Length (CBM)
Intel Atom® Processors	Intel Atom® x6427FE Processor	4	16
	Intel Atom® x6425RE Processor		16
	Intel Atom® x6414RE Processor		16
	Intel Atom® x6212RE Processor		16
	Intel Atom® x6200FE Processor		8
	Intel Atom® X6416RE Processor		16
	Intel Atom® X6214RE Processor		16
	Intel Atom® x7211E Processor	16	12
	Intel Atom® x7425E Processor		12
	Intel Atom® x7213E Processor		12
11 Gen Intel® Core™ Processors (UP3-Series)	Intel® Core™ i7-1185GRE Processor	4	12
	Intel® Core™ i5-1145GRE Processor		8
	Intel® Core™ i3-1115GRE Processor		12
Intel® Xeon® W Processors (TGL-H)	Intel® Xeon® W-11865MRE Processor	4	12
	Intel® Xeon® W-11865MLE Processor		12
	Intel® Xeon® W-11555MRE Processor		8
	Intel® Xeon® W-11555MLE Processor		8
	Intel® Xeon® W-11155MRE Processor		8
	Intel® Xeon® W-11155MLE Processor		8
12 Gen Intel® Core™ Processors (S-Series)	Intel® Core™ i9-12900E Processor	16	12
	Intel® Core™ i7-12700E Processor		10
	Intel® Core™ i5-12500E Processor		12

Processor	Brand String	# L3 Classes of Service (CLOS)	Capacity Bitmask Length (CBM)
	Intel® Core™ i3-12100E Processor		12
13 Gen Intel® Core™ Processors (P-Series)	Intel® Core™ i7-1365UE Processor	16	12
	Intel® Core™ i7-1365URE Processor		12
	Intel® Core™ i5-1345UE Processor		12
	Intel® Core™ i5-1345URE Processor		12
	Intel® Core™ i3-1335UE Processor		12
	Intel® Core™ i3-1315UE Processor		10
	Intel® Core™ i3-1315URE Processor		10
	Intel® Core™ i7-1370PE Processor		12
	Intel® Core™ i7-1370PRE Processor		12
	Intel® Core™ i5-1350PE Processor		8
	Intel® Core™ i5-1350PRE Processor		8
	Intel® Core™ i3-1340PE Processor		8
	Intel® Core™ i3-1320PE Processor		8
	Intel® Core™ i3-1320PRE Processor		8
	Intel® Core™ i7-13800HE Processor		12
	Intel® Core™ i7-13800HRE Processor		12
	Intel® Core™ i5-13600HE Processor		12
	Intel® Core™ i5-13600HRE Processor		12
	Intel® Core™ i3-13300HE Processor		8
	Intel® Core™ i3-13300HRE Processor		8
13 Gen Intel® Core™ Processors (S-Series)	Intel® Core™ i9-13900E Processor	16	12
	Intel® Core™ i9-13900TE Processor		12
	Intel® Core™ i7-13700E Processor		12
	Intel® Core™ i7-13700TE Processor		12
	Intel® Core™ i5-13500E Processor		12
	Intel® Core™ i5-13500TE Processor		12
	Intel® Core™ i5-13400E Processor		10
	Intel® Core™ i3-13100E Processor		12
	Intel® Core™ i3-13100TE Processor		12

NOTES: 1. L3 CDP is not supported on any Intel® Core™ or Intel® Atom™ processors that implement model specific L3 CAT.
2. Communications-oriented processors from the Intel® Xeon® E5 v3 Family also support a form of model-specific L3 CAT.

B.1.4.2 Register Definitions

This section identifies deltas in the register definitions for programming model specific L3 CAT. The deltas are derived against the architectural equivalent

register as documented in the Intel® 64 Architecture Software Developer's Manual (SDM), Volume 4: Chapter Title: MSRS IN THE 6TH GENERATION, 7TH GENERATION, 8TH GENERATION, 9TH GENERATION, 10TH GENERATION, 11TH GENERATION, 12TH GENERATION, AND 13TH GENERATION INTEL® CORE™ PROCESSORS, INTEL® XEON® SCALABLE PROCESSOR FAMILY, 2ND, 3RD, AND 4TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILY, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, AND INTEL® XEON® E PROCESSORS.

The naming convention for model specific L3 CAT registers mirrors the architectural L3 CAT registers without the "IA32_" prefix, for example, PQR_ASSOC (model specific) versus IA32_PQR_ASSOC (architectural).

The following deltas are consistent across all platforms that support model specific L3 CAT:

- Resource Monitoring ID's (RMIDs) are not guaranteed to be supported unless indicated by CPUID.
- L3 CDP is not supported.

B.1.4.2.1 PQR_ASSOC

The PQR_ASSOC MSR closely follows the IA32_PQR_ASSOC definition with exception of RMID. Platforms that support model specific L3 CAT typically do not support RDT Monitoring, with the exception of the Intel® Xeon® E5 v3 Family, and software should carefully consult CPUID before assuming support for any RDT Monitoring features.

B.1.4.2.2 L3_QOS_MASK_n

The L3_QOS_MASK_N MSRs are identical in definition to the IA32_L3_QOS_MASK_N for architectural L3 CAT. For the number of mask registers supported and acceptable CBM bit vector lengths, refer to Table B-8 for the processor support list.

B.1.4.3 Shareable Bit Mask

Processors with an integrated GPU may be configured, by default, to allow the GPU full access to the L3 cache in certain performance modes. This behavior remains consistent independent of the values written to the L3_QOS_MASK_n registers, as these mask registers do not affect the cache policy for transactions initiated from the GPU. Software should consider all L3 cache ways as shared with the GPU.

For processors that support Intel® Time Coordinated Computing (Intel® TCC), optimizations are available for those that require improved isolation in the L3 cache. Contact your Intel representative for additional details.

B.1.4.4 Software considerations

Software that discovers enumerated support for architectural L3 CAT using CPUID.(EAX=07H(Structured Extended Feature Flags), ECX=0) will not

automatically work with the non-architectural implementation. This section will cover known nuances and recommendations for working with the model specific non-architectural L3 CAT.

Note: Processors that support both L2 CAT and L3 CAT may have a delta in the number of CLOS supported between the L2 and L3. Intel recommends limiting software to use no more classes of service than the lesser of the two values.

B.1.4.4.1 Linux* Resource Control Groups (/sys/fs/resctrl)

Intel enables support for Intel RDT features in the Linux* kernel via Resource Control (CONFIG_X86_CPU_RESCTRL). Resource control provides an OS interface for configuring and using Cache Allocation Technology (CAT), Cache Monitoring Technology (CMT), Memory Bandwidth Monitoring (MBM), and Memory Bandwidth Allocation (MBA).

Resource Control leverages CPUID to detect hardware support for the various Intel RDT sub-features. On processors that support model specific non-architectural L3 CAT, CPUID.(EAX=07H(Structured Extended Feature Flags), ECX=0) will not enumerate support and therefore Resource Control will not support L3 CAT. Configuring of the L3_MASK_n registers will not be possible through the *resctrl* interface and must be completed through direct MSR access.

One feature of Resource Control is being able to associate a Class of Service with a Process Identifier (PID), and having the kernel automatically update the CLOS on context switch. If using a CPU that supports model specific non-architectural L3 CAT and updating the class of service on context switch is desired, it is possible to achieve this if the platform also supports L2 CAT. Resource Control would be utilized to configure L2 CAT and create the appropriate PID to CLOS mapping, while the L3 masks would need to be configured out-of-band (for example, direct MSR programming).

B.1.4.4.2 Intel-cmt-cat Tool

The Intel RDT software package intel-cmt-cat is a software library that supports the Allocation and Monitoring features of Intel® RDT. It can work with or without kernel support of Intel RDT, which makes intel-cmt-cat a useful tool when working with model specific non-architectural L3 CAT.

Intel-cmt-cat provides a pqos utility which access to the Intel RDT features through a command line interface. pqos can be used to program the L3_MASK_n registers on platforms that support non-architecture L3 CAT. Use the '--iface=msr' parameter to force enumeration and programming to be completed through MSR interfaces and not the OS interfaces.