RISC-V Quality of Service (QoS) Extensions

version 0.1

Warning! This document contains a draft of a proposed extension for purposes of discussion contributed by Vedvyas Shanbhogue. (ved@rivosinc.com). ***It is an early draft intended to be used as a starting point for establishing a TG that will define a RISC-V QoS extension.***

Introduction

Quality of Service (QoS) is the minimal end-to-end performance that is guaranteed in advance by a service level agreement (SLA) to an application. The performance may be measured in the form of metrics such as instructions per cycle (IPC), latency of servicing work, etc.

Various factors such as the available cache capacity, memory bandwidth, interconnect bandwidth, CPU cycles, system memory, etc. affect the performance in a computing system that runs multiple applications concurrently. Further when there is arbitration required for shared resources, the prioritization of the applications requests against other competing requests may also affect the performance of the application.

When multiple applications are running concurrently on modern processors with large core counts, multiple cache hierarchies, and multiple memory controllers, the performance of an application becomes less deterministic or even non-deterministic as the performance depends on the behavior of all the other applications in the machine that contend for the shared resources leading to interference. In many deployment scenarios such as public cloud servers the application owner may not be in control of the type and placement of other applications in the platform.

System software can control some of these resources available to the application such as the number of hardware threads made available for execution, the amount of system memory allocated to the applications, the number of CPU cycles provided for execution, etc. System software needs additional tools to control interference to an application and thereby reduce the variability in performance experienced by one application due to other applications' cache capacity usage, memory bandwidth usage, and interconnect bandwidth usage through a **resource allocation extension**.

Effective use of the resource allocation extension requires hardware to provide a **resource monitoring extension** by which the resource requirements of an application needed to meet a certain performance goal can be characterized.

A typical use model involves profiling the resource usage of the application using the resource monitoring extensions and to establish resource allocations for the application using the resource allocation extensions.

Architecture overview

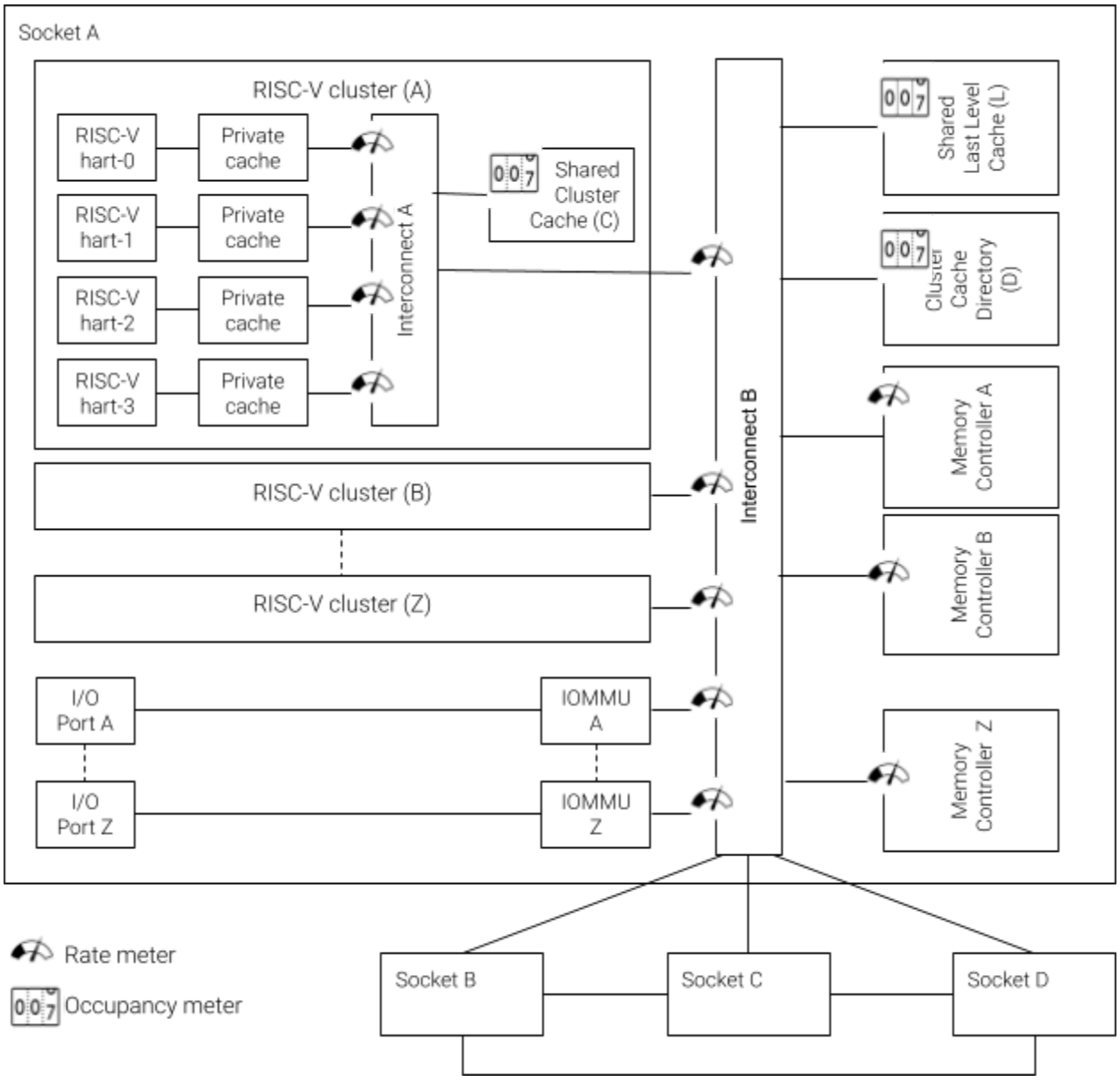


Fig 1. Example system with resource allocation and resource monitoring extension

An example system in Fig. 1 is used to illustrate the various **shared** resources that may be contended for and could lead to interference between applications. In this example system, RISC-V harts are organized as clusters connected by a **shared interconnect-A** to a **shared cluster cache C**. RISC-V clusters A through Z, and I/O ports A through Z, connect to the **shared**

interconnect B connecting a **shared last level cache L** and **shared memory controllers A through Z** in the Socket A. The shared interconnect B also connects to a **shared cache directory D** that tracks lines in upstream caches such as shared cluster cache C. Sockets A through D are connected to form the system.

With these shared resources, in order to provide deterministic performance, the system software first needs to determine the resource requirements by **monitoring** the interconnect bandwidth, memory bandwidth, and cache capacity utilized by each application. Once an application's resource requirements to achieve a performance goal are understood, the system software needs tools to **allocate** those resources to the application and limit aggressor applications from consuming excessive bandwidth and/or cache capacity..

This RISC-V QoS extension defines a mechanism to allocate following resources to applications:

1. Cache capacity
2. Interconnect bandwidth
3. Memory bandwidth

This RISC-V QoS extension defines a mechanism to report usage of following resources:

1. Cache capacity
2. Interconnect bandwidth
3. Memory bandwidth

The QoS extension may be expanded in the future to include additional resource types that may be shared such as MMU TLBs, retirement bandwidth, power, etc.

Resource Control ID (RCID) and Monitoring Counter ID (MCID)

Monitoring or allocation of resources requires a way to identify the originator of the request. Traditionally, as the request proceeds downstream through the network of resources, there is no way to associate the request with a specific application or group of applications. In some usages, in addition to providing differentiated service among applications, the ability to differentiate between resource usage for code execution and for data accesses of the same application may be required.

The QoS extension provides a mechanism by which an application can be associated with a **resource control ID** (RCID) and a **monitoring counter ID** (MCID) that accompany each request made by the application and each request made by a device on behalf of the application.

To provide differentiated services to applications, the QoS extension defines a mechanism to configure resource usage limits and optionally a priority for an RCID in the components that hold these shared resources.

To monitor the resource utilization by an application the QoS extension defines a mechanism to configure counters identified by the MCID to count events in the components that hold these shared resources.

An application is associated with one RCID and one MCID that accompany its requests for data accesses and one RCID and one MCID that accompany its requests for code accesses. Data accesses include requests generated by load and store instructions as well as the implicit loads and stores to the first-stage and second-stage page tables. Where differentiated QoS for code vs. data is not required, the code and data RCID and MCID may be programmed to be the same.

A group of applications may be associated with the same RCID and one or more of these applications may be associated with a unique MCID for code and/or data. This allows measuring the resource consumption of a subset of applications that share a RCID to determine if the resource partitioning is optimal and to make adjustments as needed.

RCID and MCID have a global scope across all caches, interconnect, and memory controllers that a request may access. The RCID and MCID are defined to be up to 16-bits wide.

> *It is recommended that an implementation supports the same number of RCIDs and MCIDs in all components that support the QoS extension.*

Associating RCID and MCID with requests

RISC-V hart initiated requests

The RCID and MCID are associated with an application by programming them into a per-privilege-mode RISC-V hart CSR. The CSR may be programmed with a RCID and MCID for code requests that may be different from the RCID and MCID to be used for data requests. All data requests at a privilege-mode from the hart carry the data-access RCID and MCID held in the corresponding CSR. All code requests originating from the hart at a privilege mode carry the code-access RCID and MCID held in the corresponding CSR.

Implementations that do not support differentiating code-access vs. data-accesses may hardwire the code-access RCID and MCID fields in the CSR to 0. In such implementations the data-access RCID and MCID is associated with both code as well as data requests.

A single RCID and MCID, selected based on the request being a code-access or a data-access, is carried with the request downstream through interconnects, caches, memory controllers, etc.

Machine-mode QoS CSR

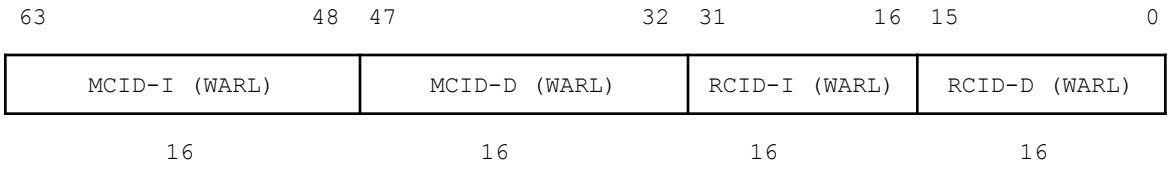


Fig 2. `mqoscfg` CSR for RV32 and RV64

The mqoscfg CSR configures code-access and data-access RCID and MCID for M-mode execution. In RV32 systems the mqoscfg CSR accesses the lower 32-bits of the configuration and a mqoscfgh CSR accesses the upper 32 bits of the configuration.

Supervisor-mode QoS CSR

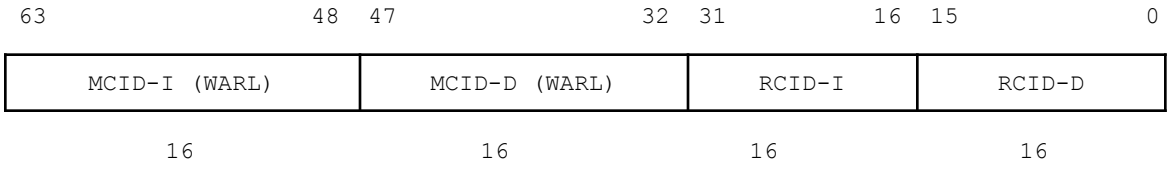


Fig 3. sqoscfg CSR for RV32 and RV64

The sqoscfg CSR configures RCID and MCID for HS/S-mode execution. In RV32 systems the sqoscfg CSR accesses the lower 32-bits of the configuration and a sqoscfgh CSR accesses the upper 32 bits of the configuration.

Virtual-Supervisor-mode QoS CSR

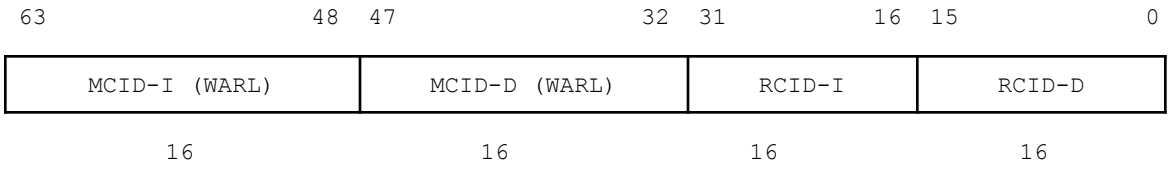


Fig 4. vsqoscfg CSR for RV32 and RV64

The vsqoscfg CSR configures RCID and MCID for VS-mode execution. In RV32 systems the vsqoscfg CSR accesses the lower 32-bits of the configuration and a vsqoscfgh CSR accesses the upper 32 bits of the configuration. When V=1, the vsqoscfg and vsqoscfgh substitute for the usual sqoscfg and sqoscfgh respectively, so instructions that normally read or modify vsqoscfg and vsqoscfgh actually access sqoscfg and sqoscfgh respectively.

Device initiated requests

Devices that are attached to an interconnect may be configured with an RCID and MCID for data access requests from the device if the device implementation supports such capability. Where the device does not natively tag its request with an RCID and an MCID, the platform implementation may provide a shim at the device interface that may be configured with the RCID and MCID. If the system supports an IOMMU, the IOMMU may be configured with the RCID and MCID to associate with inbound requests from the device connected to the IOMMU.

QoS enforcement by Caches

Cache Capacity Allocation

Caches that support the QoS extension allow the cache capacity to be allocated to applications and provide mechanisms to monitor the cache usage by the applications.

The granularity of allocation is $1/MaxCacheBlocks$ where `MaxCacheBlocks` is a property of the cache controller. A cache that supports this extension, defines the number of blocks supported by the cache. A cache block mask may then be configured in the cache controller, for each supported RCID, where each bit of the mask corresponds to a cache block.

All cache lookups scan the entire cache to determine if the requested line is present. If the requested cache line is not found then a cache line may be allocated from the set of cache blocks selected by the RCID. If allocating a line requires an eviction of a previously allocated cache line then the eviction candidate is obtained from the set of cache blocks selected by the RCID.

Reclaiming and reusing RCIDs

When a RCID is re-used for a new application, it is possible that the cache may already have lines allocated and tagged by that RCID. Thus temporary cache occupancy tracking may not accurately reflect the application's use of the cache. The tracking should shortly correct itself as the application brings its working set of memory into the caches. Software may optionally flush the cache of the cache lines allocated by the previous application prior to reusing the RCID. The cache controller may provide a mechanism to flush all lines tagged by a specified RCID to assist software with this operation.

Similarly if a memory range is unmapped from one application and mapped to another application, the cache may already hold cache lines from that memory range. Software may optionally flush the cache lines in this memory range prior to reallocating the memory range to a new application if this is not desirable.

Cache Usage Monitoring

The cache controller implements a monitoring counter per RCID and the counter can be programmed with a monitoring event ID that selects an event to count for requests with matching RCID. One such event ID would be to count the number of cache lines allocated and resident in the ache by requests with the matching RCID.

Some events counted by the cache controller may not be precise but are expected to be statistically accurate over a reasonable monitoring period.

When a monitoring counter is enabled, the count held in the counters may not be accurate till an implementation-defined number of requests have been observed by the cache controller. The controller provides a validity indiation to indicate when the count is valid.

QoS enforcement by Interconnects and memory controllers

Bandwidth Allocation

The interconnect and memory controller capacity i.e. bandwidth allocation enables restricting the bandwidth consumed by an application to a programmed limit.

The bandwidth allocation is represented as a ratio of the maximum available bandwidth.

The granularity of allocation is $1/MaxBWBlocks$ where `MaxBWBlocks` is a power of 10 with the smallest value of 100 (e.g., 100, 1000, or 10000). The `MaxBWBlocks` is a property of the interconnect or memory bandwidth controller.

Allocating bandwidth to an RCID involves configuring:

- A guaranteed bandwidth - `Gbw`
- A maximum bandwidth - `Mbw`
- Priority - `Mprio` - high, medium, or low

The `Gbw` is the minimum bandwidth in units of bandwidth blocks that is reserved for the RCID and must be at least one. The sum of `Gbw` across all RCID must not exceed `MaxGBWBlocks`. The `MaxGBWBlocks` is a property of the interconnect or memory controller. In some implementations it may be the same as `MaxBWBlocks`. Other implementations may limit to a fraction (e.g. 90%) of `MaxBWBlocks`.

The `Mbw` is the maximum bandwidth in units of bandwidth blocks that the RCID may consume. If `Mbw` is <= `Gbw` then `Mbw` does not constrain the bandwidth usage. If `Mbw` is > `Gbw` the bandwidth beyond `Gbw` is not guaranteed and actual bandwidth available may depend on the priority - `Mprio` - of the RCID that contend for the non-guaranteed bandwidth.

To enforce these limits, the controller needs to meter the bandwidth. The bandwidth metering involves counting bytes transferred (`B`), in both directions, over a time interval (`T`) to determine the bandwidth B / T.

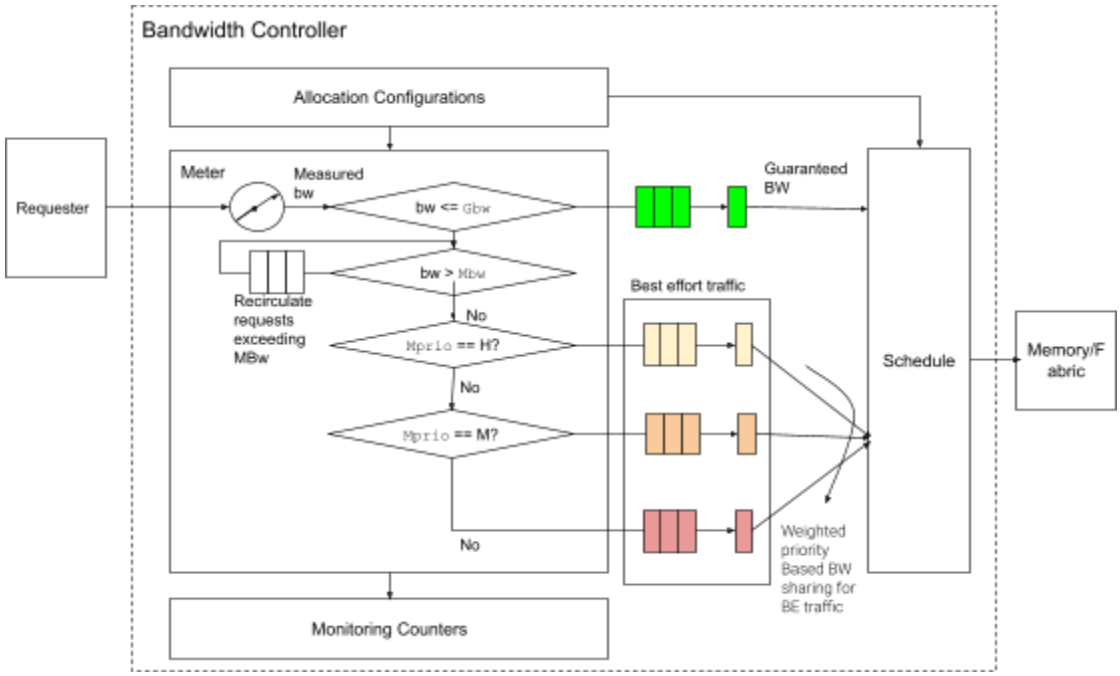
The physical manifestations of such meters are outside the scope of this specification.

> *Implementation may use discrete time intervals to count bytes such that no history is preserved from one time interval to the next. In such implementations, the counter B is reset at the start of each time interval.*

Other implementations may use a sliding time interval where in the start of the time interval advances at an uniform rate. In such a sliding time interval scheme, the counter B increments on each request and decreases by the number of bytes of older requests that are no longer in the time interval. Such a scheme may require carrying a history of requests received in any interval T.

If there is contention for bandwidth then requests from RCID that have not consumed their `Gbw` have priority irrespective of the `Mprio` configured for the RCID. Requesters that have consumed their `Gbw` contend with other requesters for the best effort available bandwidth till they have consumed `Mbw`. The contention for the non-guaranteed bandwidth is resolved using `Mprio`. The proportion of excess bandwidth that may be allocated to each `Mprio` class is configurable in the form of a configurable weight associated with each priority level.

An example of the bandwidth metering and enforcement concept is illustrated where the requests are classified into 4 groups based on bandwidth usage by an RCID. Requests from RCID that have not consumed their guaranteed bandwidth are services at priority. Requests from RCID that have consumed their guaranteed bandwidth are classified into three categories based on the priority associated with the RCID and the non-guaranteed bandwidth is made available to the RCIDs based on the weights configured for their priorities.



Bandwidth Monitoring

The bandwidth controllers implement a monitoring counter for each MCID. The bandwidth monitoring counter reports the bytes that go past the monitoring point in the bandwidth controller.

The bandwidth controller provides a mechanism to obtain a snapshot of the counter value and a timestamp at which the snapshot was taken. The timestamp shall be based on a timer that increments at the same rate as the clock used to provide timestamp on reading `time` CSR.

By computing the difference between the byte counter values from two snapshots separated in time and by computing the difference between the timestamp of the two snapshots the bandwidth consumed by the MCID in that interval can be determined.

Each counter can be programmed with a monitoring event ID such as “local read bandwidth”, “local write bandwidth”, “local read and write bandwidth”, “remote read bandwidth”, “remote write bandwidth”, “remote read and write bandwidth”, “total read bandwidth”, “total write bandwidth”, or “total read and write bandwidth” to select the event to count.

When the event ID selects read bandwidth, the counter increments by the number of bytes transferred in response to a read request. When the event ID selects write bandwidth, the counter increments by the number of bytes transferred by a write request.

The distinction of local vs. remote exists for non-uniform memory architectures where local bandwidth is the bandwidth consumed by the MCID when it accesses resources in its NUMA domain and remote bandwidth is bandwidth consumed accessing resources outside NUMA domain. The distinction of local vs. remote may not exist in some bandwidth controllers and such controllers may only support monitoring of total read and/or write bandwidth.

QoS Configuration Interface

Cache controllers

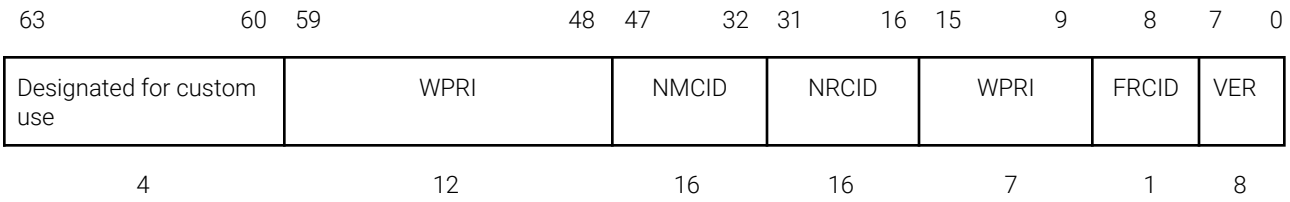
The cache controllers provide a memory-mapped programming interface. The memory-mapped registers of each cache controller are located within a naturally aligned 4-KiB region (a page) of physical address space. The table 1 lists the memory-mapped registers.

Offset	Name	Description
0	`cc_capabilities`	Cache controller capabilities
8	`cc_info`	Cache controller information
16	`cc_hart_mask`	Used to determine place in topology.

24	cc_hart_pattern	Used to determine place in topology.
32	cc_mon_csr	Cache capacity monitoring CSR
40	cc_mon_counter_value	Cache monitoring counter snapshot
48	cc_alloc_csr	Cache capacity allocation CSR
56	cc_cache_block_mask	Configure cache block mask for the RCID

Table 1. Memory-mapped cache-controller registers

Cache Controller capabilities (cc_capabilities)



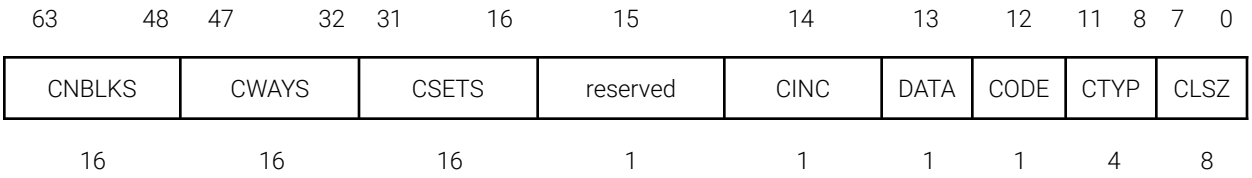
The `cc_capabilities` register is a read-only register that holds the cache controller capabilities. The `VER` field holds the version of the specification implemented by the cache controller. The low nibble is used to hold the minor version of the specification and the upper nibble is used to hold the major version of the specification. For example, an implementation that supports version 1.0 of the specification reports 0x10.

If `FRCID` is 1, the cache controller supports an operation to flush lines allocated by an RCID.

The `NRCID` field holds the number of RCID implemented by the cache controller and the `NMCID` field holds the number of monitoring counters that are implemented by the cache controller.

Cache Controller Information (cc_info)

The `info` is a read-only register that holds details of the capacity and structure of the cache.



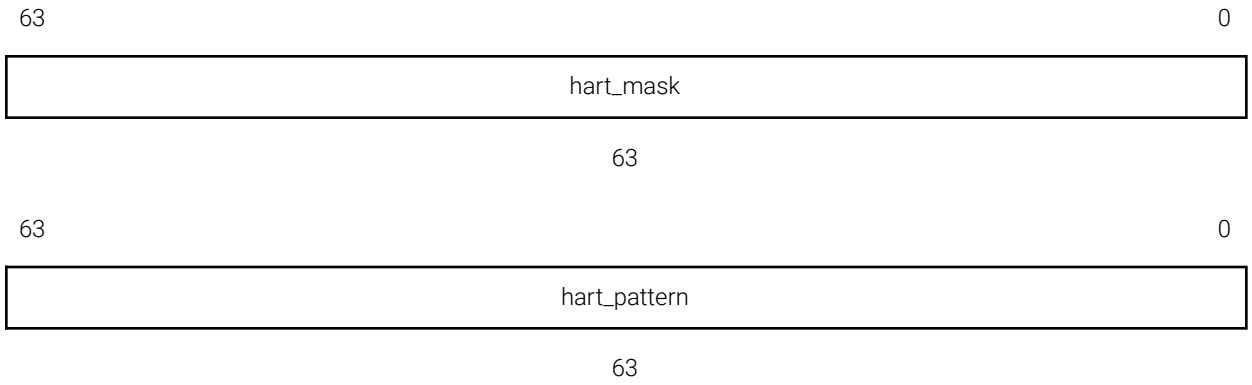
The cache controller has a line size of 2^{CLSZ} bytes. The `CTYP` field holds the organization of the cache and its encodings are listed in table 1.

CTYP	Description
0	Set-associative cache

1	Direct-mapped cache
2	Fully-associative cache
3	Set-associative directory
4	Direct-mapped directory
5	Fully-associative directory
6 - 12	reserved
13-15	Designated for custom use

The `CINC` field indicates if the cache is inclusive of all upstream caches. If `CODE` is 1, then code accesses may create entries in the cache and if `DATA` is 1 then data accesses may create entries in the cache. The `CSETS` and `CWAYS` field holds the number of sets and ways respectively in the cache if the cache is a set-associative cache The `CNBLKS` field holds the total number of allocatable cache blocks in the cache.

HART mask and pattern (cc_hart_mask and cc_hart_pattern)



The `cc_hart_mask` and `cc_hart_pattern` are used to determine the topological placement of the cache controller in the system in relation to the RISC-V harts. A cache controller caches data from a RISC-V hart if the unique identifier ("hart ID"), `hart_id`, satisfies the relationship `(hart_id & ~hart_mask) == hart_pattern`.

A system may not have a RISC-V hart for all hart IDs satisfying this relationship.

Applying this test on each RISC-V hart that is present in the system allows software to determine if a cache controller may cache data from a hart.

Cache Usage monitoring control and status register (cc_mon_csr)

The `cc_mon_csr` is used to control monitoring of cache usage by a RCID. An implementation that does not support cache usage monitoring is allowed to hardwire this register to 0.

63 56	55 41	40	39 32	31 24	23 8	7 4	3 0
Designated for custom use	WPRI	busy (RO)	status (RO)	event_id (WARL)	MCID (WARL)	WPRI	operation (WARL)
8	15	1	8	8	16	4	4

The `operation` field instructs the cache controller to perform an operation listed in Table 2. When the `cc_mon_csr` is written, the cache controller may need to perform several actions that may not complete synchronously with the write. A write to the `cc_mon_csr` sets the `busy` bit to 1 indicating the controller is performing the requested operation. When the `busy` bit reads 0 the operation is complete. Behavior of additional writes to the `cc_mon_csr` when `busy` is 1 is implementation defined. Some implementations may ignore the second write and others may perform the operation determined by the second write. Software must verify that `busy` is 0 before writing to the `cc_mon_csr`.

Value	Name	Description
0	Reserved	
1	`CONFIG_EVENT`	Configure the counter selected by MCID to count the event selected by `event_id`. The encodings of `event_id` are listed in Table 3.
2	`READ_COUNTER`	Snapshot the value of the counter selected by RCID and MCID into `cc_mon_counter_value` register.
3	`RESET_COUNTER`	Snapshot the value of the counter selected by RCID and MCID into `cc_mon_counter_value` register and reset the counter value to 0.
4 - 13	-	Reserved for standard use
14 - 15	--	Designated for custom use

Table 2. `cc_mon_csr` operation field encodings

The `event_id` holds the identifier of the event to count in the counter selected by MCID. The encodings of `event_id` are listed in Table 1.

Event ID	Name	Description
0	None	Counter does not count and retains its value.
1	Cache Capacity Usage	Counter is incremented by 1 when a request with a matching MCID allocates a new entry in the cache. Counter is decremented by 1 a when a line allocated by a matching MCID is evicted from the cache.
2	Cache	Counter is incremented by 1 when a request with a matching MCID

	Evictions	causes an eviction from the cache.
3	Cache Allocation	Counter is incremented by 1 when a request with a matching MCID causes an allocation in the cache.
4-31	-	Reserved for standard use
31-63	-	Designated for custom use
64-191	-	Reserved for standard use
192-255	-	Designated for custom use

Table 3. `cc_mon_csr` event_id field encodings

When the `event_id` for a counter is selected as 0, the counter retains its value. When `event_id` is programmed with a non-zero and legal value, the counter resets to 0 and starts counting matching events for requests with matching RCID and MCID.

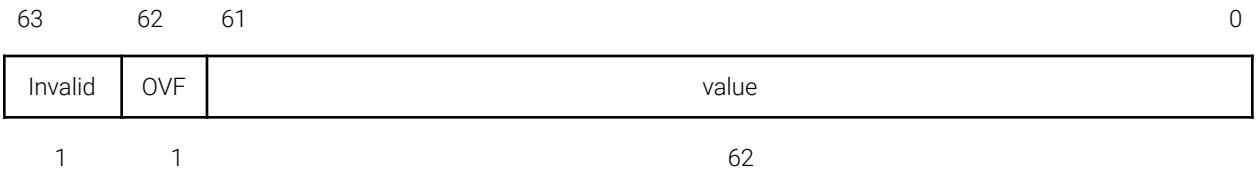
When the `busy` bit is 0, the `status` field holds the result of the last operation and its encodings are as listed in Table 4.

Status	Description
0	Reserved
1	Operation was successfully completed.
2	Invalidation operation requested
3	Operation requested for invalid RCID
4	Operation requested for invalid event_id
5-31	Reserved for standard use
32-63	Designated for custom use
64-191	Reserved for standard use
192-255	Designated for custom use

Table 4. `cc_mon_csr` status field encodings

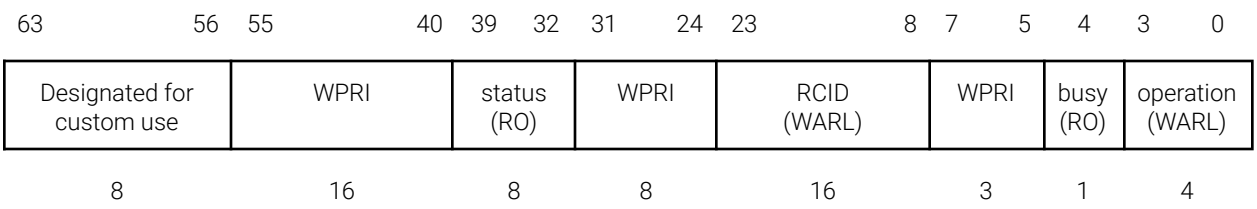
Cache usage monitoring counter value (cc_mon_counter_value)

The `cc_mon_counter_value` is a read-only register that holds a snapshot of the counter requested by `READ_COUNTER` or `RESET_COUNTER` operation. The register is 64-bit wide on RV32 and RV64 systems. The counter is valid if the `invalid` field holds 0. If the `OVF` bit is 1 then an overflow has occurred..



Cache Controller allocation control and status register (cc_alloc_csr)

The `cc_alloc_csr` is used to control allocation of cache capacity to a RCID.



The `operation` field instructs the cache controller to perform an operation listed in Table 5. When the `cc_alloc_csr` is written, the cache controller may need to perform several actions that may not complete synchronously with the write. A write to the `cc_alloc_csr` sets the `busy` bit indicating the controller is performing the requested operation. When the `busy` bit reads 0 the operation is complete. When the `busy` bit is 1, behavior of additional writes to the `cc_alloc_csr` is implementation defined. Some implementations may ignore the second write and others may perform the operation determined by the second write. Software must verify that the `busy` bit is 0 before writing to the `cc_alloc_csr`.

Value	Name	Description
0	Reserved	
1	`CONFIG_LIMIT`	The `CONFIG_LIMIT` operation is used to establish a cache usage limit for the RCID using values held in the `cc_block_mask` register.
2	`FLUSH_RCID`	The `FLUSH_RCID` operation requests the cache controller to scan the cache entries and evict lines allocated by the specified RCID. This operation may have long latency to complete. New requests to the cache controller with the RCID being flushed are allowed. The cache controller is allowed to flush cache lines that were allocated after the flush operation was initiated.
3 - 13	-	Reserved for standard use
14 - 15	--	Designated for custom use

Table 5. `cc_alloc_csr` operation field encodings

When the busy bit is 0, the status field holds the result of the last operation and its encodings are as listed in Table 6.

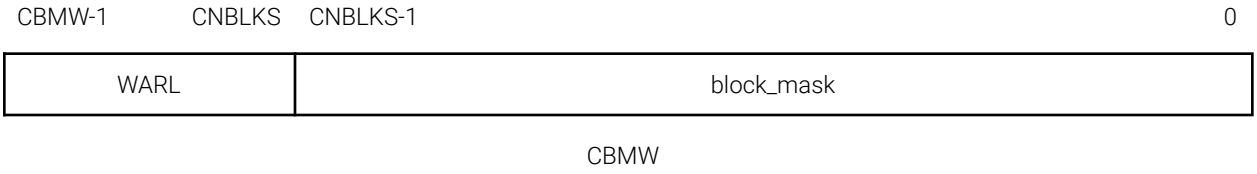
Status	Description
0	
1	Operation was successfully completed.
2	Invalid operation requested
3	Operation requested for invalid RCID
4-31	Reserved for standard use
32-63	Designated for custom use
64-191	Reserved for standard use
192-255	Designated for custom use

Table 6. cc_alloc_csr status field encodings

Cache block mask (cc_block_mask)

The cc_block_mask is a read/write register. When CONFIG_LIMIT operation is requested in cc_alloc_csr, this register holds the cache block mask for the RCID. The register has CNBLKS (see cc_capabilities) read/write bits each corresponding to one allocatable cache block in the cache. A request may allocate into a cache block if the cache block mask programmed for RCID in the request has the corresponding bit set to 1 in the cache block mask.

The width of this register is variable but always a multiple of 32 bits. The width in bits is determined as CBMW = (CNBLKS + 31)/32.



Behavior of writes to the cc_block_mask register when busy bit 1 in cc_alloc_csr is implementation defined.

To allocate cache capacity to an application identified by an RCID, software writes the cc_block_mask with the desired settings and then writes the cc_alloc_csr to request a CONFIG_LIMIT operation.

Bandwidth controllers

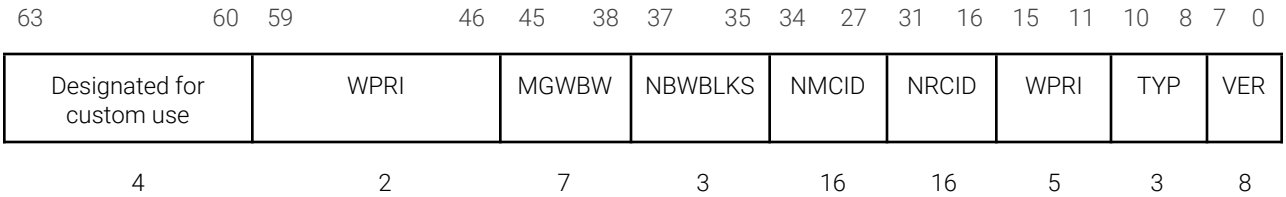
The memory or interconnect controllers that support the QoS extension have a bandwidth controller to limit the bandwidth usage by an RCID and to monitor the bandwidth usage by an RCID.

Each bandwidth controller provides a memory-mapped programming interface. The memory-mapped registers of each cache controller are located within a naturally aligned 4-KiB region (a page) of physical address space. The table 7 lists the memory-mapped registers.

Offset	Name	Description
0	bc_capabilities	Cache controller capabilities
8	bc_info	Cache controller information
16	bc_hart_mask	Used to determine place in topology.
24	bc_hart_pattern	Used to determine place in topology.
32	bc_mon_csr	Bandwidth monitoring CSR
40	bc_mon_counter_value	Snapshot of a bandwidth monitoring counter
48	bc_mon_counter_timestamp	Timestamp of the counter snapshot
56	bc_alloc_csr	Bandwidth allocation CSR
64	bc_bw_alloc	Bandwidth allocation parameters
72	bc_mprio_weight	Weights for supported priorities

Table 7. Memory-mapped bandwidth-controller registers

Bandwidth controller capabilities (bc_capabilities)



The `bc_capabilities` register is a read-only register that holds the bandwidth controller capabilities. The `VER` field holds the version of the specification implemented by the controller. The low nibble is used to hold the minor version of the specification and the upper nibble is used to hold the major version of the specification. For example, an implementation that supports version 1.0 of the specification reports 0x10.

The TYP field holds the type of bandwidth controller and its encodings are as listed in Table 8.

TYP	Description
0	Reserved
1	Interconnect bandwidth controller
2	Memory bandwidth controller
3-12	Reserved for standard use
13-15	Designated for custom use

Table 8. TYP field encodings

The `NRCID` field holds the number of RCID implemented by the bandwidth controller. The `NMCID` field holds the number of monitoring counters per RCID implemented by the bandwidth controller.

The `NBWBLKS` field holds the total exponent of 10 used to represent `MaxBWBlocks`. The smallest legal value for `NBWBLKS` is 2.

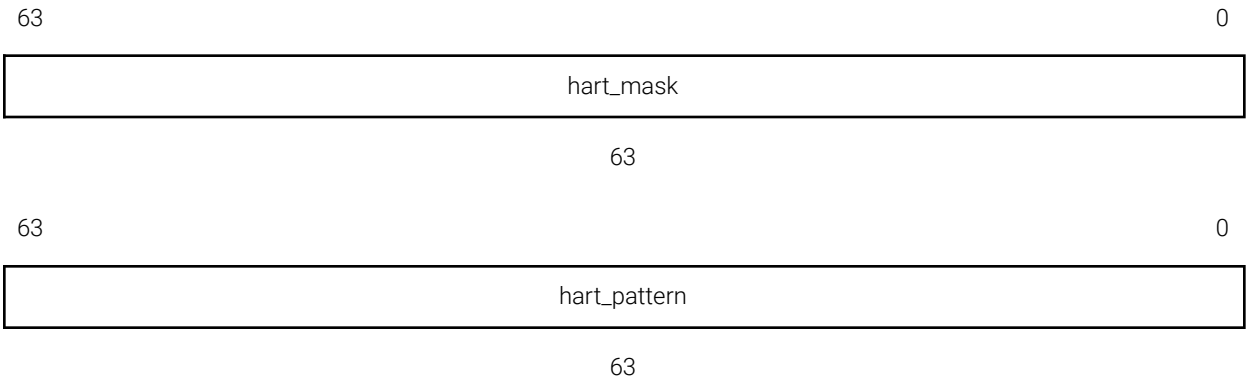
$$MaxBWBlocks = 10^{NBWBLKS}$$

Bandwidth allocations may be made in multiples of $1/MaxBWBlocks$.

Bandwidth controllers may limit the maximum available bandwidth that may be reserved as guaranteed bandwidth i.e. `MaxGBWBlocks` to a fraction of `MaxBWBlocks`. The `MGWBW` holds a number between 1 and 100 to represent this fraction.

$$MaxGBWBlocks = Round((MGWBW * MaxBWBlock)/100)$$

HART mask and pattern (bc_hart_mask and bc_hart_pattern)



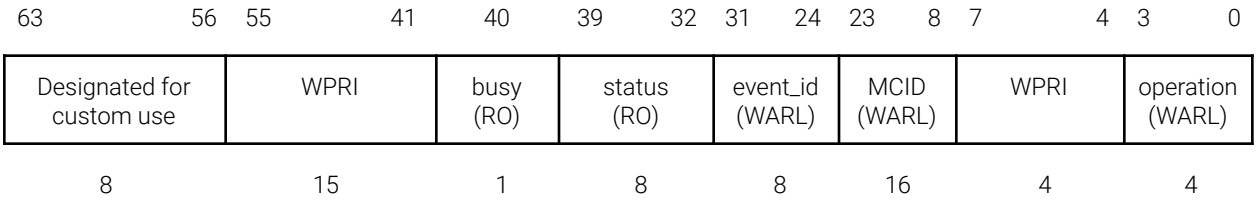
The `bc_hart_mask` and `bc_hart_pattern` are used to determine the topological placement of the bandwidth controller in the system in relation to the RISC-V harts. A controller receives

requests from a RISC-V hart if the unique identifier (“hart ID”), `hart_id`, satisfies the relationship `(hart_id & ~bc_hart_mask) == bc_hart_pattern`.

> *A system may not have a RISC-V hart for all hart IDs satisfying this relationship. Applying this test on each RISC-V hart that is present in the system allows software to determine if a bandwidth controller may receive requests from a hart.*

Bandwidth Usage monitoring control and status register (bc_mon_csr)

The `bc_mon_csr` is used to control monitoring of bandwidth usage by a RCID. Implementations that do not support bandwidth monitoring can hardwire this register to 0.



The `operation` field instructs the cache controller to perform an operation listed in Table 9. When the `bc_mon_csr` is written, the bandwidth controller may need to perform several actions that may not complete synchronously with the write. A write to `operation` field of the `bc_mon_csr` sets the `busy` bit to 1 indicating the controller is performing the requested operation. When the `busy` bit reads 0 the operation is complete. Behavior of additional writes to the `bc_mon_csr` when `busy` is 1 is implementation defined. Some implementations may ignore the second write and others may perform the operation determined by the second write. Software must verify that `busy` is 0 before writing to the `bc_mon_csr.` In a RV32 system, the write to the high 32-bits of the register should be done first and the write to the low 32-bits of the register done last.

operation	Name	Description
0	Reserved	
1	`CONFIG_EVENT`	Configure the counter selected by MCID to count the event selected by `event_id`. The encodings of `event_id` are listed in Table 11.
3	`READ_COUNTER`	Snapshot the value of the counter associated with the RCID and MCID into the `bc_mon_counter_value` register.
3	`RESET_COUNTER`	Snapshot the value of the counter associated with the RCID and MCID into the `bc_mon_counter_value` register and reset the counter to 0.
4 - 13	-	Reserved for standard use

14 - 15	--	Designated for custom use

Table 9. `cc_mon_csr` operation field encodings

The `event_id` holds the identifier of the event to count in the counter selected by RCID and MCID. The encodings of `event_id` are listed in Table 10.

Event ID	Name	Description
0	None	The counter retains its value and does not count.
1	Total Read and Write bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request and the number bytes of data written by a write request as they go past the monitor.
2	Total Read bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request as they go past the monitor.
3	Total Write bandwidth	The bandwidth monitor counters the number of bytes of data written by a write request as they go past the monitor
4	Local Read and Write bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request and the number bytes of data written by a write request as they go past the monitor to local memory.
5	Local Read bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request as they go past the monitor to local memory
6	Local Write bandwidth	The bandwidth monitor counters the number of bytes of data written by a write request as they go past the monitor to local memory
7	Remote Read and Write bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request and the number bytes of data written by a write request as they go past the monitor to remote memory.
8	Remote Read bandwidth	The bandwidth monitor counters the number of bytes transferred in response to a read request as they go past the monitor to remote memory
9	Remote Write bandwidth	The bandwidth monitor counters the number of bytes of data written by a write request as they go past the monitor to remote memory
10-127	-	Reserved for standard use
128-255	-	Designated for custom use

Table 10. `cc_mon_csr` event_id field encodings

When the `event_id` for a counter is selected as 0, the counter retains its value. When `event_id` is programmed with a non-zero and legal value, the counter resets to 0 and starts counting events that match `event_id` for requests with matching `RCID` and `MCID`.

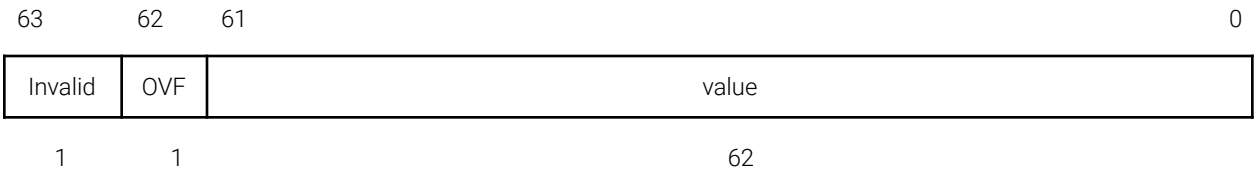
When the `busy` bit is 0, the `status` field holds the result of the last operation and its encodings are as listed in Table 11.

Status	Description
0	Reserved
1	Operation was successfully completed.
2	Invalidation operation requested
3	Operation requested for invalid MCID
4	Operation requested for invalid event_id
5-31	Reserved for standard use
32-63	Designated for custom use
64-191	Reserved for standard use
192-255	Designated for custom use

Table 11. `cc_mon_csr` status field encodings

Bandwidth monitoring counter value (bc_mon_counter_value)

The `cc_mon_counter_value` is a read-only register that holds a snapshot of the counter value requested using the `READ_COUNTER` operation. The register is 64-bit wide on RV32 and RV64 systems. The `bc_mon_counter_value` is valid if `invalid` field is 0.If the `OVF` bit is 1 then an overflow has occurred..



Bandwidth monitoring snapshot timestamp (bc_mon_timestamp)

The `cc_mon_timestamp` is a read-only register that holds the snapshot of a timer in the bandwidth controller when the counter value snapshot was recorded in bc_mon_counter_value. The register is 64-bit wide on RV32 and RV64 systems.

63	0
timestamp	
64	

Software can determine the bandwidth in bytes/second by requesting two snapshots of the bandwidth monitoring counter - B2 and B1 - and the timestamp at which the two snapshots were recorded - T2 and T1 - respectively (where T2 > T1).

The bandwidth consumed in this monitoring interval is then

$$((B2 - B1) / (T2 - T1)) * T_{freq}$$

T_{freq} is the frequency in Hz of the timer providing the timestamps and is the same as the frequency of the real-time clock used by all harts to count wall-clock real time in `time` CSR.

Bandwidth allocation control and status register (bc_alloc_csr)

The `bc_alloc_csr` is used to control allocation of bandwidth y to a RCID.

63 56	55 40	39 32	31 24	23 8	7 5	4	3 0
Designated for custom use	WPRI	status (RO)	WPRI	RCID (WARL)	WPRI	busy (RO)	operation (WARL)
8	16	8	8	16	3	1	4

The `operation` field instructs the bandwidth controller to perform an operation listed in Table 12. When the `bc_alloc_csr` is written, the bandwidth controller may need to perform several actions that may not complete synchronously with the write. A write to the `bc_alloc_csr` sets the `busy` bit indicating the controller is performing the requested operation. When the `busy` bit reads 0 the operation is complete. When the `busy` bit is 1, behavior of additional writes to the `bc_alloc_csr` is implementation defined. Some implementations may ignore the second write and others may perform the operation determined by the second write. Software must verify that the `busy` bit is 0 before writing to the `bc_alloc_csr`.

Value	Name	Description
0	Reserved	
1	`CONFIG_LIMIT`	Program the RCID bandwidth allocation with parameters held in the `bc_bw_alloc` register.
2 - 13	-	Reserved for standard use

14 - 15	--	Designated for custom use

Table 12. `bc_alloc_csr` operation field encodings

When the `busy` bit is 0, the `status` field holds the result of the last operation and its encodings are as listed in Table 13.

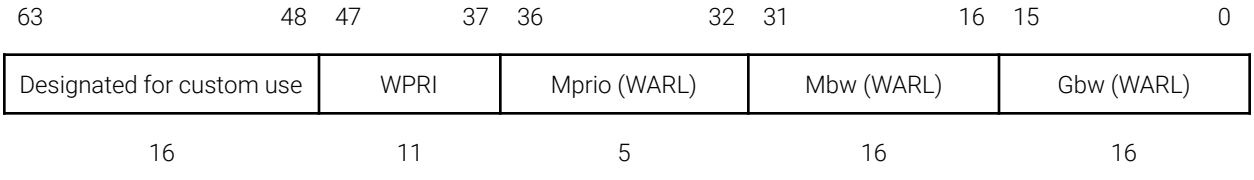
Status	Description
0	Reserved
1	Operation was successfully completed.
2	Invalid operation requested
3	Operation requested for invalid RCID
4	Guaranteed BW configuration exceeds capacity.
5	Invalid guaranteed BW requested
6	Invalid maximum BW requested
7	Invalid priority requested
8-31	Reserved for standard use
32-63	Designated for custom use
64-191	Reserved for standard use
192-255	Designated for custom use

Table 13. `bc_alloc_csr` status field encodings

Bandwidth allocation and priority (bc_bw_alloc)

The `bc_bw_alloc` is a read/write register that is used to program a guaranteed bandwidth (`Gbw`), maximum bandwidth (`Mbw`), and priority `Mprio` for a RCID.

The bandwidth is allocated in multiples of bandwidth blocks and the value in `Gbw` or `Mbw` must be at least 1 and must not exceed `MaxBWBlocks`. The sum of `Gbw` allocated across all RCID must not exceed `MaxGBWBlocks`. When multiple priority levels are supported, the Mprio must be a number between 0 and (`NPRIO` - 1) (see `bc_capabilities`). `Mprio` value of 0 implies highest priority and a value of (`NPRIO` - 1) implies the lowest priority and is used to arbitrate the non-guaranteed bandwidth among the RCIDs based on a weight configured per priority.



Weighted priorities (mprio_weight)

The `mprio_weight` is a read/write WARL register used to configure a weight as a value between 1 and 255 for the priorities supported by the bandwidth controller. Implementation that does not support priorities or weighted priorities can hardwire this register to 0.

The register holds `NPRIO` (see `bc_capabilities`) read/write bytes each corresponding to a supported priority level in the controller. The width of this register is variable but always a multiple of 32 bits. The width in bits is determined as `MPWW` = ((`NPRIO * 8`) + 31)/32.

The non-guaranteed BW is proportionately shared among the `Mprio` based on the configured weight. A larger weight implies a greater fraction of the non-guaranteed bandwidth is allowed for that priority level. The weight for priority `N` is held at the byte offset N in `weight_array`. The default value of the weights in `weight_array` following a reset is 255.