# NetSpeed Pegasus Last Level Cache

## *Technical Reference Manual*

**Version: PEGASUS-18.04**

**Revision: 0.0**

# NetSpeed Pegasus Last Level Cache

## About This Document

This document describes the architecture of Pegasus Last Level Cache and NocStudio commands for Pegasus. Using Pegasus, users can configure cache hierarchy to boost system performance.

## Audience

This document is intended for users of NocStudio Orion and Gemini:

- SoC Architects
- NoC Architects
- NoC Designers

## Prerequisite

Before proceeding, you should generally understand:

- Basics of Network on Chip technology
- AMBA interconnect standards

## Related Documents

The following documents can be used as a reference to this document.

- NetSpeed NocStudio Orion User Manual
- NetSpeed NocStudio Gemini User Manual

## Customer Support

For technical support about this product, please contact support@netspeedsystems.com

For general information about NetSpeed products refer to: www.netspeedsystems.com

# Revision History

| Revision | Date | Updates |
|---|---|---|
| **0.0** | Jun 16, 2018 | Initial Release |

NETSPEED SYSTEMS

# Contents

# Figures

# 1 Introduction

Pegasus is a highly customizable and configurable last level cache that can eliminate memory bottlenecks and boost overall system performance.

Pegasus can act as a memory bandwidth multiplier. Whenever a memory read or write hits a line that's present in the cache, the access to memory can be avoided. This reduction in the accesses to memory reduces the utilized memory bandwidth, effectively increase the available memory bandwidth of the system. A cache hit rate of 50%, for instance, would allow 2X the number of memory requests by locally completing half of them and only sending the other half to memory.

Pegasus also increases system performance by reducing average latency. For every request that hits in the cache, the latency of going to memory can be eliminated and the request can be processed locally.

Each request that is completed by Pegasus also reduces dynamic power. Off-chip accesses to DRAM consume significant dynamic power. Cache hits eliminate this power consumption with a much lower power on-chip RAM access.

Pegasus allows architects significant control of their design by supporting a multitude of flexible cache hierarchies. It can be configured as a memory cache or as a coherent-only cache.

Pegasus can be configured as a coherent-only cache. Only coherent or IO coherent accesses will be sent to the cache in this configuration. This limits the benefits of the cache to these coherent accesses but can provide a lower latency and lower area solution. Non-coherent accesses go directly to memory, which can have a number of indirect benefits including support for larger than 64B requests.

If Pegasus is configured as a memory cache, all accesses to the specified address range will go to the cache and perform a cache lookup. This allows non-coherent accesses to gain the latency and bandwidth benefits of caching.

Pegasus as a coherent-only cache needs to support cache maintenance operations to push data to memory in order to facilitate communication between coherent and non-coherent devices. These are used when a memory space moves from one Shareability Domain to another. If Pegasus is configured as a memory cache, no cache maintenance is needed as requests all requests will see the same data.

The cache hierarchy connectivity can be created to have redundant paths so that the LLC can be entirely disabled and traffic can be routed directly to memory instead. This allows the entire LLC to be powered off and traffic to take a more direct route in the network.

Based on system requirements such as cache capacity and total coherent bandwidth, architects can add multiple instance of Pegasus, and customizes them before placing them in the interconnect. The benefits that Pegasus brings are:

- Lower latency by placing Pegasus where they are accessed the most.
- Reduce congestion by handling requests locally & using caches to reduce traffic to memory.
- Improve die utilization by placing Pegasus in empty die space

# 2 Functional Description

The Pegasus Last-Level Cache (LLC) module is a cache designed to supplement a coherent system. It is expected to be instantiated between the Coherency Controller and the Memory Controller.

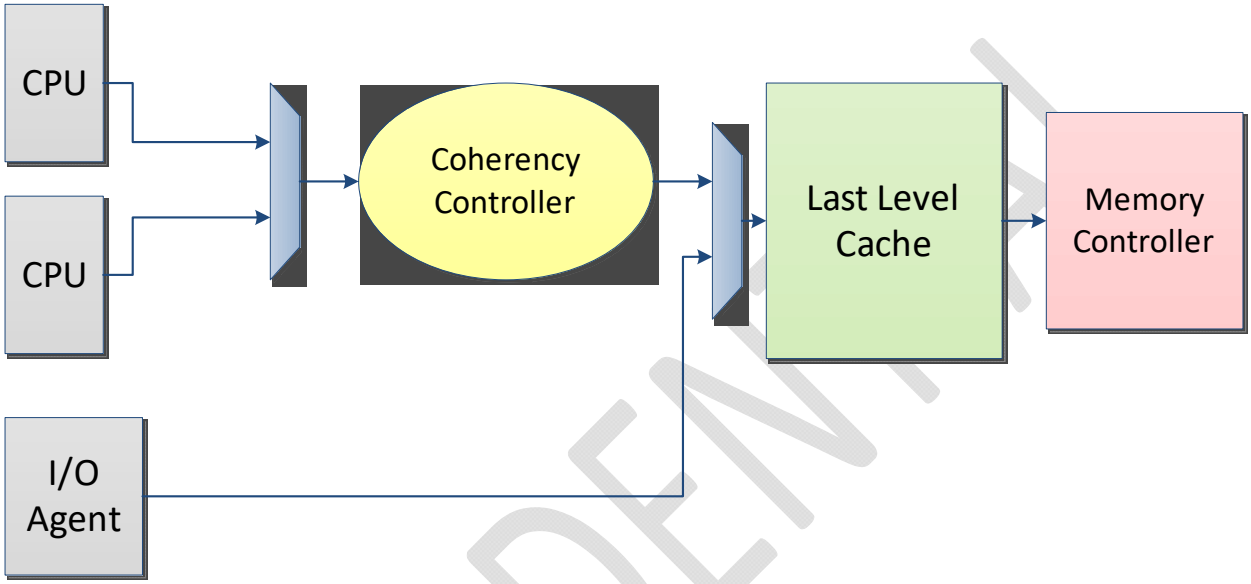


*Figure 1: Last Level Cache sits behind coherency controller*

Pegasus is a non-integrated LLC design, which means the LLC is a separate module from the coherency controller. This provides significant flexibility in how Pegasus is used in a system.

In a coherent system, this flexibility allows a different number of CCCs and LLCs in the system. It also allows flexibility in the physical location of these modules on the floorplan.

Since the LLC is not integrated with the coherency controller, it can be utilized even in non-coherent systems as a system cache. As shown below, the LLC has an ACE-lite input port and an AXI output port. In a non-coherent system, the unused ACE-lite signals can be tied off to work as an AXI port.


*Figure 2: LLC has an ACE-lite and AXI port*

The input is ACE-lite so that the cache can receive cache maintenance instructions as well as standard read and write instructions. It has an AXI output port where it is able to fetch lines from memory or evict lines to memory.

## 2.1 Relationship to Coherency

A last level cache resides between the coherency controller and the memory controller. This leaves it outside of the coherent space. The LLC contents do not need to be tracked by the directory, and the LLC does not need to retain Shared or Unique state information. The cache only needs to track Valid and Dirty state.

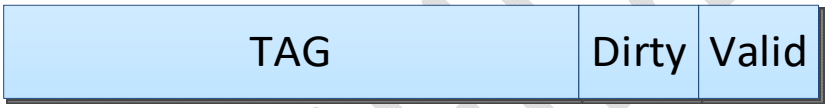


*Figure 3: State Tracking in LLC Tags*

If a coherent cache has a copy of the cache line, even in a Unique state, the LLC can also have a copy of the line. The coherency protocol will attempt to provide any requestor with the coherent version of the data, and only if it misses in the coherent system will the data in the LLC be used. This means there are no requirements on Inclusivity or Exclusivity for the LLC with respect to the coherent system.

## 2.2 Multiple LLCs

To achieve higher bandwidth, or to partition the RAMs across the chip, it is possible to have multiple, parallel last level caches.

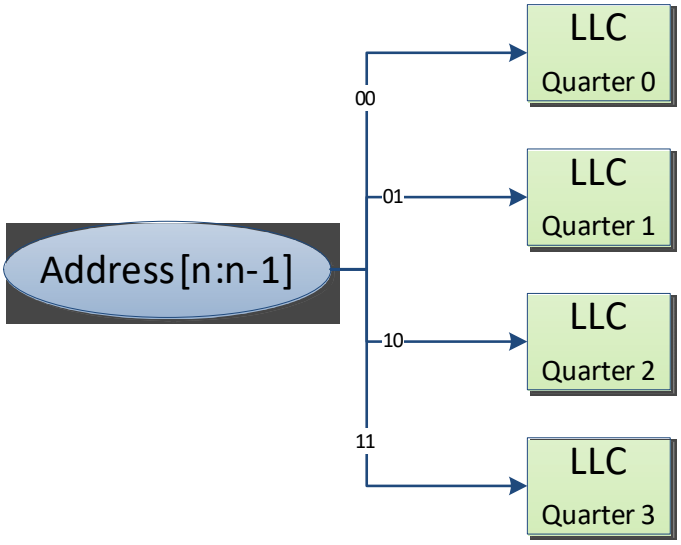


*Figure 4: Multiple Parallel LLCs*

Since the caches are not tracked by a coherency mechanism, they have to be guaranteed that they won't hold the same cache lines. This is guaranteed by splitting which addresses go to each cache. For a power-of-2 number of caches, this can be easily accomplished by using certain address bits to index which cache the request should go to. A hash function can also be used to distribute requests more evenly across the LLCs.

## 2.3 Flexible Connectivity

The LLC cache can be connected to the system in a number of ways. One method is to have dedicated LLCs for each memory controller, or to each coherency controller. Connections can then be made directly between agents.

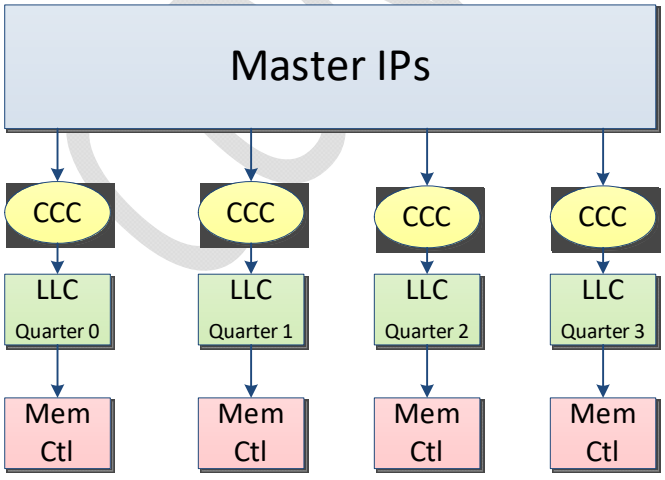


*Figure 5: Dedicated caches connectivity*

In the example above, each of the components in a column would be responsible for the same portion of the address space. An alternative approach is to allow LLCs to be split using some other addressing mechanism, so there isn't a 1:1 association.

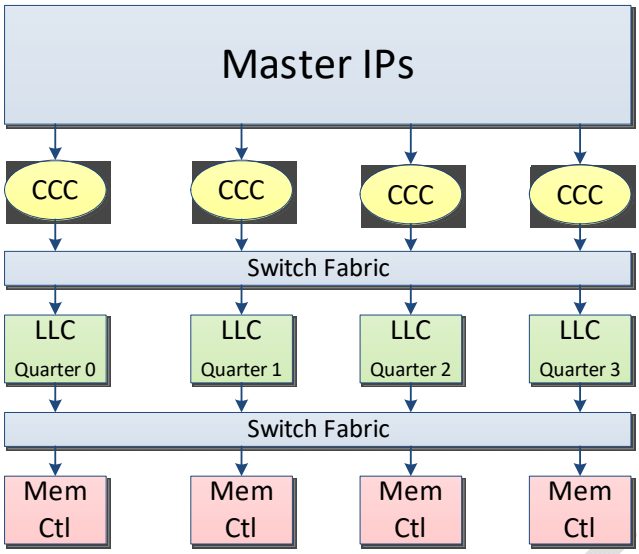


*Figure 6: Flexible Connectivity*

## 2.4 LOGICALLY PARTITIONED RAM ARRAYS

A major issue with building Last-Level Cache IP is that RAM designs are technology specific and cannot be synthesized with the rest of the logic. The RAM arrays can be built in a number of ways with different latencies, frequencies, and bandwidths. They can also be logically banked for increased bandwidth. Pegasus is built to have enough flexibility to use different RAM designs.

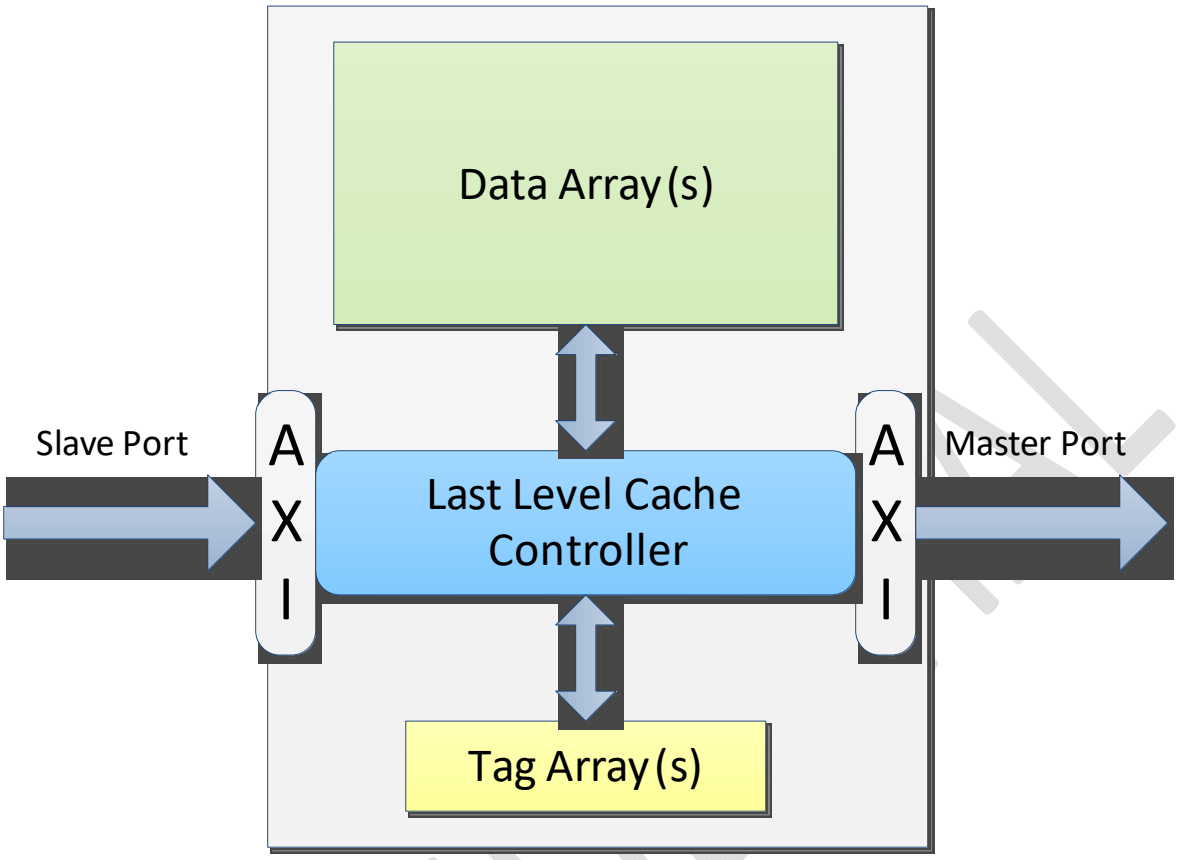


*Figure 7: LLC has Controller, Tag Arrays and Data Arrays. Controller has separate interface to access the arrays.*

The diagram above shows the LLC in more detail. The cache controller is separate from the tag and data arrays. There is a simple interface between the controller and each of the array blocks. The controller is the primary IP that is provided to customers. The Arrays are either built with compilers or are custom designed by the customer. The controller must be flexible in how it accesses the arrays, but this loose coupling allows the arrays to be built independently.

## 2.5 Single-ported RAMs

Since a last-level cache is big, it needs to be built with dense arrays. The standard RAM design for big arrays is a 6-T cell with a single port that allows either a read or a write. The cache controller is built assuming only single-ported RAMs are utilized.

## 2.6 Flexible Timing of Arrays

Pegasus is configurable to handle the various timing requirements of the RAMs. For each RAM array, the controller must know the latency of the access to the RAMs, as well as the bandwidth. This will affect the rate at which requests are sent to the RAM, as well as the expected delay until the RAM response returns.

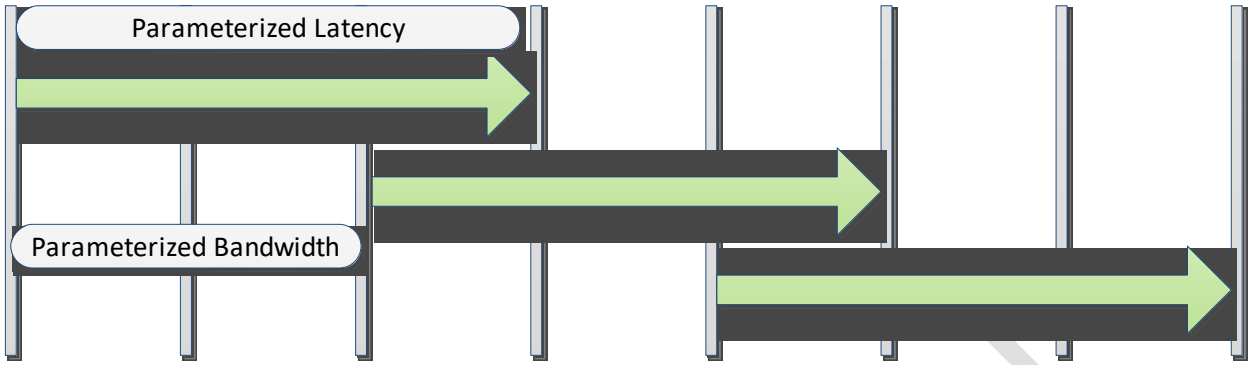


*Figure 8: RAM access show flexible latency and repeat rate*

The cache control logic provides a flexible specification of the RAMs for latency and bandwidth. This allows the cache to adjust to the specific RAM implementation.

Read and Write latencies and bandwidth are assumed to be identical, which is typically the case for single-ported RAM arrays.

## 2.7 Banking of the RAMs

Since the data arrays may not be fully pipelined, higher throughput can be achieved through a banked design. Parallel requests can be made to the different banks, allowing more requests per cycle. Pegasus supports the use of multiple data banks.

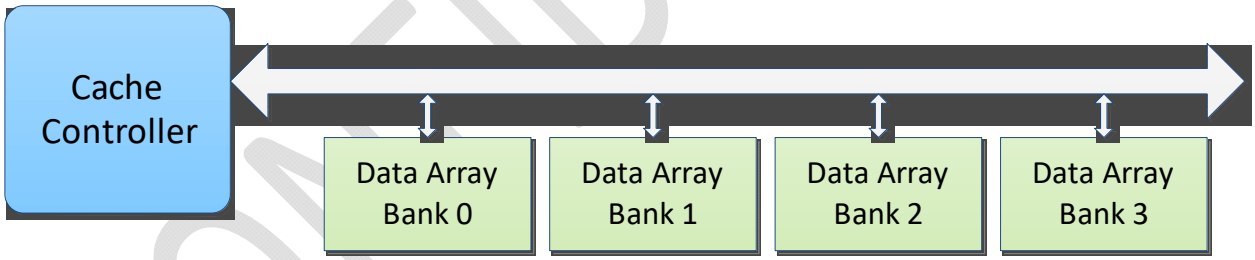


*Figure 9: Cache with banked data arrays*

Normally data array banks can be split by address or by cache ways. Pegasus supports splitting by associative ways. This allows the cache to easily support power-gating sections of the data array. Ways can be disabled for allocation, flushed, and power gated. Re-enabling the bank is easy and does not affect the enabled arrays.

## 2.8 Flexible Capacity and Associativity

The LLC is designed to be configured for different sizes of caches. Both the number of sets and the number of ways is configurable. This allows control over the size and associativity of the cache.

Like most caches, the number of sets in the cache is required to be a power of 2 in size. This allows straightforward indexing and avoids uneven pressure across sets.

The number of ways has more flexibility and can be used to create non-power-of-2 sized caches.

## 2.9 Way Groups and Data Banking: Relationship

The LLC is built with an implied relationship between the number of associative ways, and the number of data banks.

Ways are always instantiated in groups of 4. So, the associativity of the cache must be a multiple of 4. This group of 4 is called a Way Group. For each Way Group, there are two data array banks. One if for the first half of the cache line. The other is for the second half of the cache line. As more associativity is added, so are more data banks.

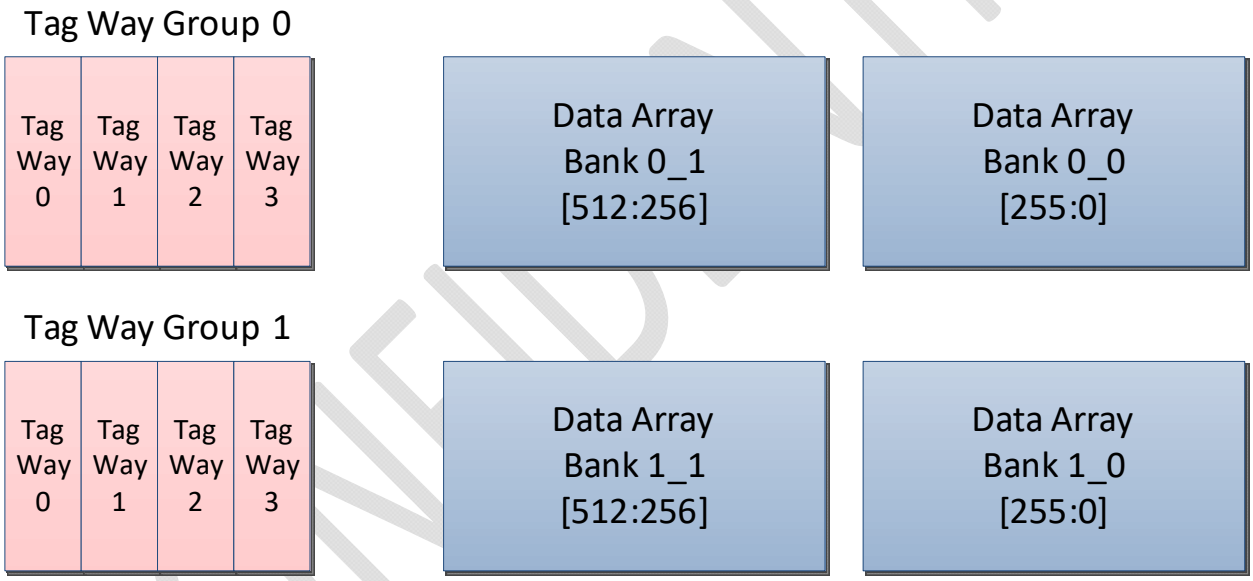


*Figure 10: Ways and Banking are related*

## 2.10 Scratchpad RAM Mode

Since Pegasus contains a significant amount of on-chip RAM, it is often useful for a system to utilize this RAM directly instead of as a cache. This is often called scratchpad mode. Part or all of a cache can be modified to act as a backing store for a range of addresses. The addresses will always hit in the RAM and will never miss or evict the address.

The LLC can allow portions of its cache to be utilized as a scratchpad RAM. The amount that is converted to scratchpad is flexible.

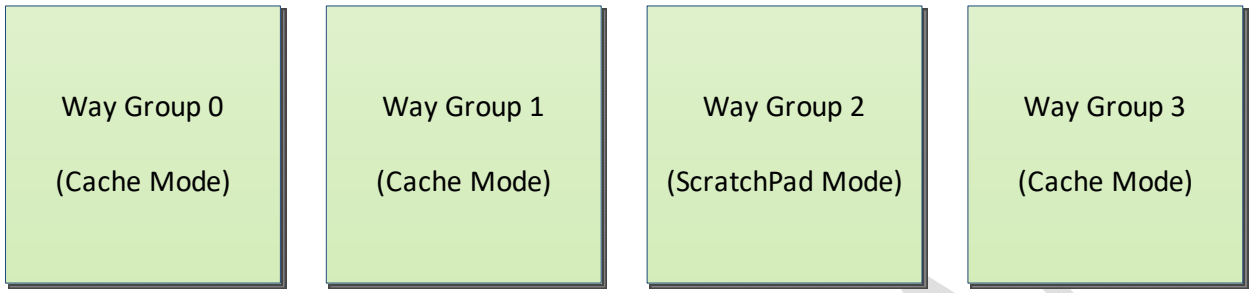


*Figure 11: Portions of cache can be modified to act as a scratchpad RAM*

To transition a portion of the cache to scratchpad mode, the associated ways must be disabled and the lines must be flushed from the cache. Once the designated ways are removed, the way can be programmed to act as scratchpad. During this mode, the tag array for these ways will be unused.

The scratchpad mode requires an address space. This range is programmable.

The scratchpad space can be protected with Trust-Zone security. If the space is indicated as secure, only secure accesses will be able to read or write the data. Non-secure accesses will receive a decode error.

## 2.11 Replacement Policy

Each Way Group tracks 4 associative ways. It also keeps a 3-bit tree-LRU indicator to determine which lines are more recently used and which are less. If a Way Group is selected for replacement, the tree-LRU will choose the least recently used line.

Across Way Groups, there is no information that tracks relative age or use. Among the possible Way Groups, a global round-robin mechanism is used to select ways for replacement. This amounts to a random policy within the same set.

If there are empty ways to choose from, the algorithm will ignore the normal replacement method and pick the first available empty position.

## 2.12 Partial Reads and Writes

Like all caches, the LLC is primarily used for cache line accesses. However, it is possible for a partial cache line access to occur.

For read requests, the behavior is simple. It will either retrieve data from the cache or will go to memory. The partial read will never allocate into the cache.

For partial writes, the write will either merge with data already in the cache (utilizing a read-modify-write sequence), or it will send the write directly to memory. The LLC will not fetch a line from memory to merge with the write and allocate in the cache. If it isn't already in the cache, the LLC will send the partial write to memory and let it be handled there.

In scratchpad mode, the partial write must merge with the existing cache line.

## 2.13 TRUST-ZONE BIT

The LLC is aware of trust-zone support. It will use the AxPROT[1] bit, which is utilized for Trust-Zone, as an additional address bit stored in the TAG. This prevents accesses that are non-secure from seeing secure data, or the reverse. The same address, with different AxPROT[1] bits, can exist as two separate entries in the cache because the cache treats them as different addresses.

## 2.14 ALLOCATION CONTROL

LLC supports way allocation controls. LLC Allocation Class can be set to one of 8 allocation classes. Each allocation class has a set of allocation control capabilities which are programmable with defaults configurable in NocStudio. This includes selection of which ways in the cache a request can allocate into, as well as allocation rules. All agents in an allocation class share these properties.

### 2.14.1 Way Allocation Controls

Choosing which lines to replace on a cache line allocation can have significant performance implications. The first consideration for replacement is determining which cache ways are available for allocation. Some ways may be disabled, and an agent may be limited to a subset of the remaining ways when choosing a position for allocation.

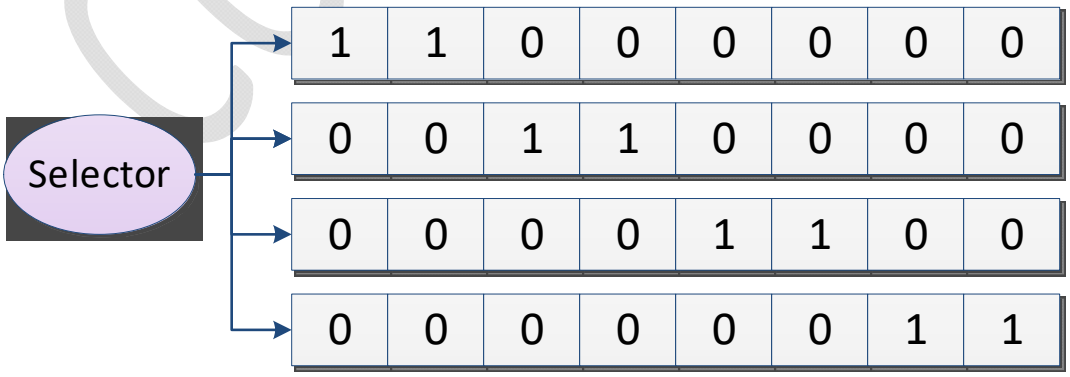


*Figure 12: Programmable allocation vectors*

Pegasus supports 8 allocation classes per system. Each allocation class has a programmable register that indicates which ways the allocation class is allowed to allocate into. This allows several agents to have different allocation vectors, which enables software to statically allocate cache entries to specific agents. These vectors can overlap as well, allowing some agents to share cache ways.

To disable ways for allocation, the corresponding bit for each of the allocation vectors can be set to zero. As new lines are brought into the cache, they will only allocate if an available way is active. If there are no available ways to allocate into, the line will pass through the cache without allocating.

Disabling allocation into a way for an agent or even all agents does not prevent the way from being looked up and compared during a cache access. Even if an agent can't allocate in a way, the line it is trying to access could be present in that way. Disabling allocation does not disable the way.

Disabling allocation in a way for all agents can be used as the first step for flushing cache lines from that way. Once new entries can't allocate, a flush engine can walk through the entries in that way and evict them until the entire way is empty.

### 2.14.2 Dynamic control of allocation behavior

When a miss occurs, whether to allocate into the cache is primarily controlled by the two system-wide 8-bit vectors:

- llc_class_read_allocate_use_arcache
- llc_class_write_allocate_use_awcache

These vectors indicate for each class (bit-position) whether dynamic-allocation is allowed (value=1) or disabled (value=0). If dynamic-allocation is allowed, then the value of the AxCACHE bit is used for the final determination of whether to allocate.

The AxCACHE bits are supplied by the master agent on the AXI/ACE interface, but may be overridden by the master bridge through use of the following parameters:

- for writes: axi4_ovrd_awcache_enb and axi4_ovrd_awcache_val
- for reads: axi4_ovrd_arcache_enb and axi4_ovrd_arcache_val

### 2.14.3 Static control of allocation behavior

If dynamic control of allocation is disabled for a class (relevant bit of the parameter is zero) then two vectors determine the allocation policy statically for reads and writes:

- llc_class_read_allocate
- llc_class_write_allocate

Each bit indicates whether allocation is enabled or disabled for the corresponding class.

## 2.15 ECC Support

Both tag and data arrays can be protected with ECC. These are independent configuration options, so a cache can be built with just ECC on DATA, or just ECC on TAG, or both, or neither. Additionally, each LLC in the system can be independently configured, so some caches can have ECC while others may choose not to.

## 2.16 Configurable Index and Tag Bits

Pegasus has configurability on how many bits of the address are index bits and how many are tag bits. It also has flexibility in which bits of the address are used for the index and which are used for the tag.

If multiple LLCs are utilized, certain bits of the address will likely be used to select which of the LLCs an address goes to. The effect is that for a particular LLC, those selected address bits will always be a constant. That means they should not be utilized as part of the index, since it would make a sizable portion of the cache inaccessible.

These bits can also be removed from the Tags, since the value is constant. This optimization can have a big impact on tag size. However, this optimization may be undesirable if the address slicing is reprogrammed at any time. For instance, if 3 out of 4 LLCs are powered off, and all addresses are sent to the last LLC, it will need to have sufficient tag space to track every line. This area/power vs. flexibility tradeoff is up to the customer.

In addition to slice bits not being stored, it is common for upper address bits to be constant as well. A memory space may only take a fraction of the total address space.

## 2.17 Control Sequences

The LLC has built-in state machines that allows automated methods of flushing or invalidating the caches.

**Invalidate Tags:** This sequence will invalidate all tags within a programmed vector of ways. This can be used at reset or power up when the contents of the RAM are unknown.

**Invalidate data:** This sequence can zero out the contents of the data array. This is useful when switching to scratchpad mode, as a way of initializing the data. More importantly, it is a way of invalidating any secure data that was stored either before the scratchpad mode was entered, or

after secure scratchpad was exited. Since scratchpad allows a direct access of the RAM contents, remnants of secure data must be invalidated before direct access is enabled.

**Cache way flush engine**: This engine will run through the cache flushing and invalidating all cache lines in the programmed ways. The lines are invalidated in case a write to an already flushed line occurs, hitting in the way. While the allocation for those ways is disabled, access to them is still allowed.

## 2.18 Cache Maintenance Instructions

Since the LLC has an ACE-lite input, it can receive cache maintenance instructions. This include CleanInvalid, CleanShared, and MakeInvalid. These instructions may flush or invalidate particular cache lines.

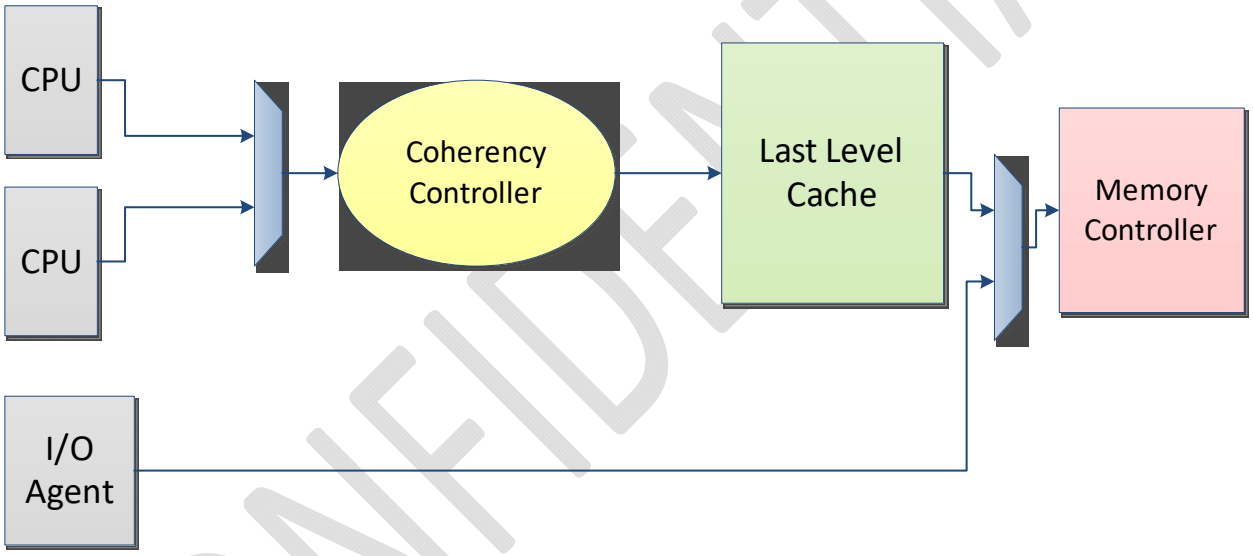


*Figure 13: Non-coherent DMA bypasses LLC*

This is required in a system where non-coherent traffic bypasses the LLC and directly accesses memory, as shown above. These instructions are used to push lines to memory so that prior coherent stores are visible to non-coherent reads, and new coherent reads will see the results of non-coherent writes.

The Cache Maintenance operations are not an effective way of flushing or invalidating the cache, and not intended to be used for that purpose. They are used for transitioning between Shareability domains and their different use models.

## 2.19 AXI EXCLUSIVE FUNCTIONALITY

Pegasus can be configured to provide AXI Exclusive functionality (see AXI/ACE specification section A7) when it is configured as a memory cache. Exclusive sequences require a monitor per agent to track whether a line has been modified between an Exclusive read and the Exclusive write. Pegasus can be configured with a variable number of Exclusive monitors.

A single AXI/ACE port can have multiple logical agents, each capable of performing Exclusive operations. In order to avoid thrashing between these requests, Pegasus supports a separate monitor for each logical agent behind the port. For agents that never perform Exclusive requests, Pegasus will remove the corresponding monitor hardware.

The Exclusive monitors track requests on a 64B granularity. If requests are made for smaller than 64B, an Exclusive update to part of the cache line can invalidate the monitors for the other sub-blocks.

Pegasus does not support Exclusive requests greater than 64B in size. If larger granularity is needed, an address range can be created that goes straight to memory instead of to the LLC

# 3 NocStudio Commands and Properties for LLC

NetSpeed's NocStudio allows the instantiation of last level caches and the configuration of several of its properties.

## 3.1 Adding a Last Level Cache

NetSpeed's Pegasus or Last Level Cache (LLC) can be added using the **add_llc** command.

```
add_llc <name> [color <value>] [pos <pos>] [size <size_x> <size_y>] <bridge
        <slave name> llcs [pos] bridge <master name> llcm [pos]>
```

The last level cache name, color, position, size, bridge names and bridge positions can all be specified similar to the **add_host** command. As noted above, the last level cache has 2 bridges: 1 master of type *llcm* and 1 slave of type *llcs*.

## 3.2 LLC with 2 Slave Ports

The LLC can be configured to have a second slave port. This feature allows a more latency optimized solution.

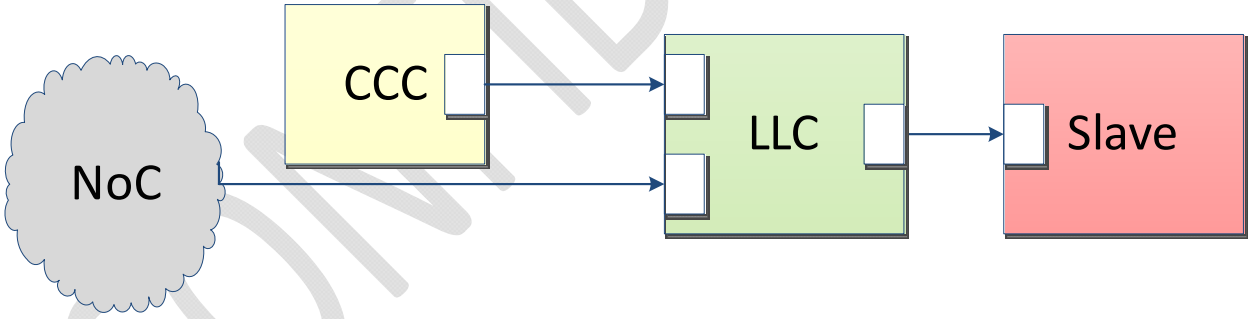


*Figure 14: Use of second slave port in LLC*

The second slave port is used when there are multiple masters talking to the LLC. This can happen in a variety of cases, including the one shown above where CCC talks to LLC, but so do other master. When LLC is set up as a memory cache, this will happen.

By using the second port, the CCC can talk directly to the LLC, allowing it to skip the bridges and routers if the ports are co-located in the same grid position in NocStudio. This can provide a significant performance improvement by reducing the latency of coherent requests.

The third port can be added by adding another bridge to the LLC with protocol *llcs2*. When this bridge is added, the LLC will have a new host property called *llc_second_slave_port_connect*. This property needs to be set to the port that is directly connecting to the additional slave port. In the example above, it is the CCC master port that would be specified.

Traffic to the LLC is always specified through an LLC group or RAM group. Neither of these specifies a slave bridge, and NocStudio recognizes that all traffic applies to both ports.

The second port has its own properties for read and write max outstanding.

## 3.3 CONFIGURABLE PROPERTIES OF THE LLC

Several properties specific to the LLC can be configured through properties available in NocStudio.

- **hysteresis_count:** This property specifies the number of clock cycles before LLC goes into low power mode when coarse clock gating is enabled.
- **llc_cache_associativity:** This property specifies the number of associative ways in the cache. This value must be a multiple of 4. The default value is 8.
- **llc_cache_capacity:** This property specifies the total capacity of this cache, specified in MB. The maximum allowed cache size is 1024 MB.
- **llc_class*_alloc_waygroups:** This property is used to set up allocation vectors for each of the LLC Allocation classes. The property is a bit field specifying which waygroups that class is allowed to allocate into.
- **llc_class_read_allocate:** This property indicates whether an LLC allocation class defaults to allocate on reads. This property is an 8-bit field that specifies the default for each allocate class. If that class sets the llc_class_read_allocate_use_arcache property, this property is unused.
- **llc_class_read_allocate_use_arcache:** This property indicates whether an LLC allocate class should use the ARCACHE bits to determine if a read should allocate. The property is an 8-bit field allowing each LLC Allocation Class to control whether it uses the ARCACHE bits. If not, the llc_class_read_allocate field will determine if allocation happens on reads.
- **llc_class_write_allocate:** This property indicates whether an LLC allocation class defaults to allocate on writes. This property is an 8-bit field that specifies the default for each allocate class. If that class sets the llc_class_write_allocate_use_awcache property, this property is unused.
- **llc_class_write_allocate_use_awcache:** This property indicates whether an LLC allocate class should use the AWCACHE bits to determine if a write should allocate. The property is an 8-bit field allowing each LLC Allocation Class to control whether it uses

the AWCACHE bits. If not, the llc_class_write_allocate field will determine if allocation happens on writes.

- **llc_data_ecc_enable:** If enabled, the data RAM stores ECC bits along with the data, enabling error correction and detection. By default, this property is disabled.
- **llc_data_ram_bandwidth_delay:** This value specifies the bandwidth of the data RAM. The format specifies the bandwidth as one access (read or write) every N cycles, where N is the value specified. The default value is 2 cycles.
- **llc_data_ram_latency:** This value specifies the latency of the data array for this cache. A one-cycle RAM lookup would have the value 1. The default value is 2 cycles.
- **llc_index_bits:** This bit vector specifies which of the address bits is used for indexing into the cache.
- **llc_master_port_read_max_outstanding:** This property specifies the number of outstanding read requests the LLC master port can support from all sources. The default value is 16.
- **llc_master_port_write_max_outstanding:** This property specifies the number of outstanding write requests the LLC master port can support from all sources. The default value is 16.
- **llc_max_address_size:** This value specifies the maximum number of address bits that need to be tracked by the tag. If the cacheable address space is smaller than the system address, the tag can be reduced in size by not storing those bits.
- **llc_slave_port_read_max_outstanding:** This property specifies the number of outstanding read requests the LLC slave port can support from all sources. The default value is 16.
- **llc_slave_port_write_max_outstanding:** This property specifies the number of outstanding write requests the LLC slave port can support from all sources. The default value is 16.
- **llc_tag_bits:** This bit vector specifies which of the address bits is stored in the cache tag and used for lookup comparison.
- **llc_tag_ecc_enable:** If enabled, the tag RAM stores ECC bits along with the tag value, enabling error correction and detection. By default, this property is disabled.
- **llc_tag_ram_latency:** This value specifies the latency of the tag array for this cache. A one-cycle RAM lookup would have the value 1.
- **llc_waygroup_cache_mode_enable:** This vector property specifies for each LLC waygroup whether cache mode is enabled by default. If neither cache mode or RAM mode are enabled for a waygroup, it will start of disabled and will be inaccessible until programmed to be enabled in one of those modes. A waygroup cannot be enabled for cache mode and RAM mode.

- **llc_waygroup_ram_mode_enable:** This vector property specifies which LLC waygroups work as RAM. Address range has to be specified to LLC bridge before specifying this property. Cache mode and RAM mode cannot be enabled for the same waygroup on reset.
- **llc_waygroup_ram_mode_secure:** This vector property specifies which LLC waygroups work as secure RAM. Address range has to be specified to LLC bridge before specifying this property.

When a second slave port is added, the following properties are available:

- **llc_slave_port2_read_max_outstanding:** This property specifies the number of outstanding read requests the second LLC slave port can support from all sources. The default value is 16.
- **llc_slave_port2_write_max_outstanding:** This property specifies the number of outstanding write requests the second LLC slave port can support from all sources. The default value is 16.
- **llc_second_slave_port_connect:** This property specifies the master port that connects to the LLC second slave port.

There is an LLC related property in master bridges which talk to LLC.

- **llc_allocation_class:** This property specifies the allocation class for the bridge which LLC uses to determine which way groups to use for the allocation.

## 3.4 GROUPING LLC'S

For systems with larger bandwidth requirements, it may be necessary to utilize multiple LLCs to increase bandwidth. One common method for supporting this is to take an address range and slice it into equal parts, with each LLC responsible for one of the parts. If requests are well distributed to the different slices, bandwidth will increase proportional to the number of components. Having 4 instances of a cache can get 4x the bandwidth of a single instance.

The slicing function uses specified address bits to assign responsibility to each component. Slicing can be done using a power-of-2 number of slices. This allows a simple decode of address bits.

To support slicing of address space, NocStudio allows for a group of LLCs to be declared so that they function together to split responsibility of an address space. This group can then be declared in an *add_range* command to place the elements of the group into the correct hierarchy.

An LLC group can be specified using the **add_llc_group** command.

```
add_llc_group –name <llc group name> -hash_fns <[hash name1]
        [hash_name0]..> | -slice_bits <slice bits> [-memory_cache_enable]
        [-llc_disableable] –members <[llc_host1]…>
```

Each LLC group is assigned a unique name. The slicing function uses specified address bits to assign responsibility to each component. Slicing can be done using a power-of-2 number of slices. This allows a simple decode of address bits. The number of LLCs must be a power of 2. This ensures that address slicing can be done with a direct decode of the address bits chosen for slicing. N components requires log2(N) address bits. If an incorrect number of address bits or components are specified, the command will be rejected.

If the memory_cache_enable option is specified, all the LLC's in this group function as a memory cache. A memory cache is a cache that is accessible by all traffic, including coherent and non-coherent. If the LLC is not marked as a memory cache, it is not accessible by non-coherent traffic.

The llc_disableable option implies that the LLC's in this group can be completely powered off and disabled if required. If this option is specified, any traffic flow that has a path to the LLC will also have a redundant path to the slave.

An LLC group can also be added to the address range of the slave. Specifying an LLC group with an address range means that requests targeting the specified address range can access the LLC's that are part of the specified group to read or write data. This can reduce latency. An LLC group can be added to both coherent (with CCC group) and non-coherent (without CCC group) address ranges.

## 3.5 CONFIGURING PEGASUS AS A SCRATCHPAD RAM

Part or all of the LLC can be configured as Scratchpad RAM. To configure this in NocStudio, the user must add the LLC or LLCs into a RAM group using the *add_ram_group* command in NocStudio. When multiple LLCs are added to a RAM group, it indicates that a single address range will be divided across the specified LLCs. The range can be divided using slice bits or a hash function.

Once the RAM group is created, an address range must be assigned to the RAM group. Only a single range can be added for a RAM group, and the range must a power-of-2 size that is minimally inclusive of the combined LLC capacities. So, if two LLCs have 1.5MB of cache each, totally 3MB, the Scratchpad RAM address range must be 4MB in size. The range can be marked as coherent by adding a CCC group in the *add_range* command.

If a hash function is used, the hash function is not allowed to be programmable. Unlike a cache, the RAM acts as a backing store, and needs to drop one of the specified hash function bits in order to compress the address space.

Once the range is created, traffic can be specified to the RAM group through the *add_traffic_b* command.

Once these steps are finished, the range and traffic to the Scratchpad RAM is configured. But the defaults configuration of the LLC still needs to be specified. For each waygroup of each cache, the user can specify whether the waygroup should reset to cache mode, or to RAM mode, or neither. For RAM mode, the host property *llc_waygroup_ram_mode_enable* specifies which waygroup is enabled as RAM. The user can also specify if that waygroup allows secure or non-secure accesses with the property *llc_waygroup_ram_mode_secure*.

The mapping of address to waygroup is needed for software to know which addresses are available. For the range specified, each LLC will look at address bits up to the size of the address range. So, a 4MB ram group would have 22 address bits, or [21:0]. The index bits used by the scratchpad are the lower order bits, while way bits are the top bits. The way bits are broken into {waygroup number, way}. So, a 4MB cache with 8 waygroups would use bits 21:19 as waygroup, 18:17 as the way bits within the waygroup, and 16:6 as the index bits. 8 waygroups is 32 ways, so each way would have 128KB of storage.

# 4 Programmers Model

## 4.1 Transitions for Way Group State

Each waygroup can exist in one of three states. It can be enabled for Cache, enabled for RAM, or disabled. When disabled, it may be powered off. This section will describe the transitions between these states.

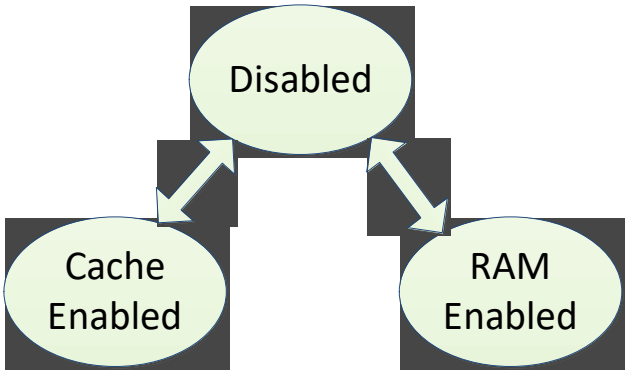


*Figure 15: Each WayGroup supports 3 operation states*

As seen in this figure, while 3 state exists, the transitions should always go through the Disabled state. Directly changing Cache Enabled to RAM enabled or vice versa is not supported.

### 4.1.1 Cache Mode to Disabled Mode Transition

To transition a waygroup or multiple waygroups from cache mode to disabled mode, the first step is to disabled allocation for those waygroups. This will prevent any new lines from being added to the waygroups. This can be achieved by writing to the Global Allocation Way Group Enable register. By marking the appropriate waygroups as zero, no new lines will allocate into that cache.

After allocation is disabled, the lines present in that waygroup must be flushed and invalidate. The flush will write back any dirty data to memory, while the invalidation will prevent any new requests to an address from writing into the cache in that location. The flush engine is controlled by the LLC Way Flush register. The waygroups that you want to flush can be written to the wayflush register, and it will trigger the flush engine. Once the flush process has completed, the way flush control register will reset to zero. Since the flush sequence can take a significant amount of time, the status can be periodically polled by reading the register to determine if any of the bits are still set. If all are zeroed out, the flush engine has completed.

Once the flush engine has completed, those ways will be guaranteed to be empty and no new lines will be added. During this time, requests will still perform cache lookups for these waygroups, but at this point, they will always miss.

At this point, the Cache Way Enable register can be programmed to indicate that these ways are no longer enabled for Caching. Once they are disabled, new cache accesses will stop reading the cache tags and will ignore the contents of those pins. At this point, the corresponding tag arrays and data arrays can be powered off.

### 4.1.2 RAM Mode to Disabled Mode

To transition a waygroup from RAM mode to the Disabled mode, the hardware transition is straightforward. A write to the RAM Way Enable control register can mark the waygroup as disabled.

Software may need more elaborate control sequences. When the waygroup transitions to disabled, the prior content of the RAM is lost. If that data needs to be preserved, a copy sequence must be performed by software to move the data from the RAM to a location in memory. Before this sequence happens, software should coordinate traffic so that new requests to the RAM region are not expected.

### 4.1.3 Disabled Mode to Cache Mode

To transition from disabled mode to cache mode, the LLC must go through a sequence where the tag RAMs are invalidated, where they are enabled for use, and then finally they are enabled for allocation.

The first part of this sequence must ensure that the corresponding tag arrays are invalidated. If the ways were previously invalidated and the state was maintained (there was no power sequences, for instance), no further action is required. If the tags are in an unknown state, or have just powered on, an invalidation sequence must occur. The LLC Tag Invalidation Engine control register can be used to initiate an invalidation of the tags for the waygroups in question. The registers can be polled for status to determine when it has completed.

Once invalidated, the cache ways can be enabled for cache mode accesses by writing to the Cache Way Enable register. Once set, those waygroups will be looked up by any cache access. However, the entries will be invalidated until the allocation enables are set up. The allocation controls can be set before the Cache Way Enable register is set because allocation is limited to waygroups that are enabled.

#### 4.1.4 Disabled Mode to RAM Mode

To transition a disabled waygroup to RAM mode requires three steps. First, the data contents must be overridden. Second, security permissions must be determined for the RAM waygroups, and finally, the waygroups should be set to RAM mode.

The data invalidation must happen in order to ensure that any data that was previously stored into the data array (either cache data or RAM data) is completely invalidated. Without this step, it is possible for the RAM mode to have visibility of unrelated data, including potentially secure data. Invalidation is needed to ensure security is preserved.

Data invalidation happens by writing to the Data Invalidation Engine Control register, indicating which waygroups should be invalidated. This register can also be polled for status.

Once invalidation has completed, but before the RAM mode enable occurs, the security permission must be decided. This can be controlled by writing to the Scratchpad Ram Way Group Security register. This register has one bit per way group, with a value of 1 requiring accesses to be secure, and a value of 0 requiring non-secure.

Finally, the RAM mode can be enabled by setting the Scratchpad RAM Way Group Enable register.

## 4.2 Register-based Access of RAMs

Both the Data RAMs and Tag RAMs are accessible through register-based accesses. This allows a backdoor method of reading or writing the arrays, which can be useful for debug, DFT follow up, or even for controlling stimulus in a test.

The register-based accesses use two sets of registers. It uses the indirect content registers, and the indirect trigger register. The content registers hold data to be written to the RAMs, or data read from the RAMs. The indirect trigger actually performs the access and indicates RAM address (index and way).

There are 4 kinds of accesses applicable to each of the LLC RAMs. The operations are read, write raw, write+ecc, and read-modify-write.

The read request is triggered by a write to the indirect trigger indicating it should perform a read. It will read the specified RAM location and copy the contents of that RAM into the indirect content registers. The completion of the trigger access occurs only after the RAM access has occurred, so there is no need to poll for completion. Once the trigger access is done, software can load the content registers to view the data in the RAM.

The write request is initiated by a write to the indirect trigger, and it copies the values in the indirect content registers into the RAM. There are two variants of the write request. The first will write the entire array, including ECC, based on the indirect content register. The second variant will write the non-ECC data with the indirect content register but will perform an ECC generation to determine the checkbits to set for the ECC portion of the RAM.

The read-modify-write sequence performs an atomic access to the RAM where it reads the content, XORs the content with the indirect content registers, and writes it back. This allows an atomic method of flipping a specified number of bits. This is useful for forcing the generation of single or double bit errors. The read-modify-write can be triggered during normal operation to introduce single or double bit errors. The operation is atomic, so it will modify the existing content of the RAM without any chance of that entry being modified in the middle of the sequence.

## 4.3 LLC Allocation Controls

There are several programmable registers that can change the allocation properties of the LLC. These registers correspond to the following NocStudio LLC properties:

- llc_class*_alloc_waygroups
- llc_class_read_allocate
- llc_class_read_allocate_use_arcache
- llc_class_write_allocate
- llc_class_write_allocate_use_awcache

The NocStudio properties set the initial values of these registers, but the registers can be reprogrammed at any time to modify the allocation behavior of the system. These registers control which ways an LLC Allocation Class is allowed to allocate into, and whether they should allocate on reads or writes.

These allocation controls are modified by which ways are enabled for cache mode. If a way is disabled for cache mode, programming it to be allocated into will have no effect. This means these properties do not need to be modified as waygroups are enabled or disabled for cache use.

## 4.4 LLC Host Registers

### 4.4.1 LLC_ALLOC_ARCACHE_EN

This register holds one bit for each of the LLC Allocation Classes. If the bit is marked as one, this indicates that read allocation for that LLC Allocation Class is controlled by the ARCACHE bits.

If marked as zero, it means the allocation will be controlled by the llc_alloc_rd_en register. The default of this register is configured within NocStudio.

Attribute: RW

Bit field description:

ARCACHE_Allocation_Enable[7:0] - Read allocation for that LLC Allocation Class is controlled by the ARCACHE bits/llc_alloc_rd_en register.

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | ARCACHE_Allocation_Enable | | | | | | | |

*Table 1 LLC_ALLOC_ARCACHE_EN register.*

### 4.4.2 LLC_ALLOC_AWCACHE_EN

This register holds one bit for each of the LLC Allocation Classes. If the bit is marked as one, this indicates that write allocation for that LLC Allocation Class is controlled by the AWCACHE bits. If marked as zero, it means the allocation will be controlled by the llc_alloc_wr_en register. The default of this register is configured within NocStudio.

Attribute: RW

Bit field description:

AWCACHE_Allocation_Enable[7:0] - Write allocation for that LLC Allocation Class is controlled by the AWCACHE bits/llc_alloc_wr_en register.

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | AWCACHE_Allocation_Enable | | | | | | | |

*Table 2 LLC_ALLOC_AWCACHE_EN register.*

### 4.4.3 LLC_ALLOC_RD_EN

This register holds one bit for each of the LLC Allocation Classes. The use of this register is controlled by the llc_alloc_arcache_en register, which indicates whether ARCACHE bits should be used for allocation, or whether this register should decide on allocation. If this register is used

for an LLC Allocation Class, a value of one will indicate that reads should allocate into the LLC. A value of zero indicates that reads should not allocate.

Attribute: RW

Bit field description:

Read_Allocation_Enable[7:0] - Reads should/should not allocate into the LLC

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | Read_Allocation_Enable | | | | | | | |

*Table 3LLC_ALLOC_RD_EN register.*

### 4.4.4 LLC_ALLOC_WR_EN

This register holds one bit for each of the LLC Allocation Classes. The use of this register is controlled by the llc_alloc_awcache_en register, which indicates whether AWCACHE bits should be used for allocation, or whether this register should decide on allocation. If this register is used for an LLC Allocation Class, a value of one will indicate that writes should allocate into the LLC. A value of zero indicates that writes should not allocate.

Attribute: RW

Bit field description:

Write_Allocation_Enable[7:0] - Writes should/should not allocate into the LLC.

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | Write_Allocation_Enable | | | | | | | |

*Table 4 LLC_ALLOC_WR_EN register.*

### 4.4.5 LLC_CACHE_WAY_ENABLE

This register indicates whether a way is enabled for cache access. If enabled, a cache lookup will read the associated Tag values and perform an address comparison. If disabled, the Tags won't be accessed and the contents of the Tags won't be compared.

This allows the Tags to be powered down or the lines to be used for RAM access. The register has one bit per way, allowing each way to be individually enabled or disabled. A value of 1 indicates that the way is enabled, while a value of 0 indicates it is disabled. All ways are enabled by default. Before disabling a cache way, the way must be disabled in the llc_global_alloc register, and the contents should be flushed.

Attribute: RW

Bit field description:

Cache_Way_Enable[31:0] - Cache Way Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Cache_Way_Enable | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 5 LLC_CACHE_WAY_ENABLE register.*

### 4.4.6 LLC_CLASS_ALLOC

The class allocation control registers are used to specify which associative ways can be written to. Each master in the system belongs to an LLC class, and each class allocation control register indicates which ways that class of agents can allocate into.

These registers can be used to provide dedicated associativity for different agents or groups of agents. The default value of these registers indicates that all ways are accessible by all agents, with a value of one indicating allocation is allowed. Setting the value to zero will disable allocation for an agent.

It is permissible to turn off allocation for all ways, which will prevent any accesses from that class from allocating into the cache.

Note that the llc_global_alloc register can override these values. If global allocation is disabled for a way, none of the agents can allocate into those ways regardless of what the llc_class_allocate registers indicate.

Attribute: RW

Bit field description:

WGE_0[7:0] - Class 0 Allocation Way Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | WGE_0 | | | | | | | |

*Table 6 LLC_CLASS_ALLOC register.*

### 4.4.7 LLC_DATA_INV_CTL

This register controls a state machine that can invalidate the contents of data array banks. Each way has a corresponding bit in this register. When the register is written, the invalidation engine will kick off and invalidate the data for each of the specified ways. Writing a value of 1 to a bit indicates that the corresponding way should be invalidated. Writing a value of 0 to a bit indicates that the content of that way shouldn't be invalidated.

The register can be read to determine the current status of the data bank invalidation sequence. When hardware has completed the invalidation sequence for a way, it will change the value of that register bit from 1 to 0. If the entire register has a value of 0, then the invalidation engine has completed. The reset value of this register is zero.

Invalidation of the data array is needed when switching between cache mode and scratchpad RAM mode, since the RAM mode allows direct access to the data. Any secure data that was stored in the cache may be visible to RAM mode accesses unless it is invalidated first. Similarly, if security permissions are removed for the Scratchpad RAM, the prior contents should be invalidated before removing the security check.

An invalidation sequence should be completed before a second sequence is requested.

Attribute: RW

Bit field description:

Data_Bank_Invalidation_Enable[31:0] - Data Bank Invalidation Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Data_Bank_Invalidation_Enable | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 7 LLC_DATA_INV_CTL register.*

### 4.4.8 LLC_ECC_DATA_INFO

This is a status register that tracks ECC errors that occur in the Data array. The register will track the number of ECC errors, as well as whether single-bit or double-bit errors have been detected. If the SB bit is set, at least one single bit error has been detected. If the DB bit is set, at least one double-bit error has been detected.

Additionally, the register tracks information about the first error detected. It stores the index of the tag array that had the error, as well as the way group. It also tracks which half of the cache line failed, which is needed to identify the sub-bank that failed. If a double-bit error occurs after a single-bit error has already been recorded, the double-bit error will overwrite the content of the register. This is because double-bit errors are fatal, and the information about how a fatal error is more important that the information about a non-fatal error.

The register can be read for status but can also be written. If the SB and DB bit are written with zeros, the sampling of the first detected error will happen as described above.

Attribute: RW

Bit field description:

Index[45:32] - index of first detected error

ECC_Count[31:16] - number of ECC errors found

hlf[9] - Which half of cache line reported error

way[8:2] - Way group of first detected error

db[1] - Detected double or multi bit error

sb[0] - Detected single bit error

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ECC_Count | | | | | | | | | | | | | | | | u | | hlf | way | | | | | | | | | | db | sb |

*Table 8 LLC_ECC_DATA_INFO register.*

### 4.4.9 LLC_ECC_DISABLE

This register allows ECC to be disabled for either the Data arrays or the Tag arrays. These are independently controlled. A bit value of 1 indicates that ECC is disabled. A bit value of 0 indicates ECC is enabled, if present. The register value resets to value 0, meaning ECC is enabled.

Attribute: RW

Bit field description:

D[1] - Disable Data ECC Check/Correct

T[0] - Disable Tag ECC Check/Correct

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | D | T |

*Table 9 LLC_ECC_DISABLE register.*

### 4.4.10 LLC_ECC_TAG_INFO

This is a status register that tracks ECC errors that occur in the Tag array. The register will track the number of ECC errors, as well as whether single-bit or double-bit errors have been detected. If the SB bit is set, at least one single bit error has been detected. If the DB bit is set, at least one double-bit error has been detected.

Additionally, the register tracks information about the first error detected. It stores the index of the tag array that had the error, as well as the way group. If a double-bit error occurs after a single-bit error has already been recorded, the double-bit error will overwrite the content of the register. This is because double-bit errors are fatal, and the information about how a fatal error is more important that the information about a non-fatal error.

The register can be read for status but can also be written. If the SB and DB bit are written with zeros, the sampling of the first detected error will happen as described above.

Attribute: RW

Bit field description:

Index[45:32] - Index of first detected error

ECC_Count[31:16] - Number of ECC errors found

way[6:2] - Way group of first detected error

db[1] - Detected double or multi bit error

sb[0] - Detected single bit error

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ECC_Count | | | | | | | | | | | | | | | | u | | | | | | | | way | | | | | db | sb | |

*Table 10 LLC_ECC_TAG_INFO register.*

### 4.4.11 LLC_EVENT_COUNTER

This register is an event counter. When the event counter mask control enables certain events to be counted, they will increment this counter. When the counter overflows, in can produce an interrupt if the interrupt mask is enabled. This can be used to trap to software after a number of specified events has occurred.

The counter can be read or written. Writing the value can initialize the counter to a larger value which can speed up the point at which counter will overflow and the interrupt will be triggered.

Attribute: RW

Bit field description:

Event_Counter_Value[31:0] -

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Event_Counter_Value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 11 LLC_EVENT_COUNTER register.*

### 4.4.12 LLC_EVENT_COUNTER_MASK

This register is used to program the event counter. Each bit of this register enables the performance counter to increment if the event occurs. A value of 1 for a bit indicates that this event should be counted.

If multiple bits are set to 1, the logic will only count if all of the corresponding events occur on the same cycle. This allows for combinations of events, such as a cache miss that causes an eviction. The events that can be counted are all related to cache accesses and occur in the same cycle of the pipeline, so they can be combined easily.

When an event satisfies all of the requirements, the llc_event_counter register will be updated. A value of 1 indicates that the event is selected for counting. A value of 0 the event is not selected. By default, this register will be set to 0 for all bits, indicating no event counting should occur.

Attribute: RW

Bit field description:

e9[9] - Retry access

e8[8] - Retry needed

e7[7] - Eviction

e6[6] - Cache maint op

e5[5] - Partial write

e4[4] - Cache miss

e3[3] - Cache hit

e2[2] - Scratchpad access

e1[1] - Cache write

e0[0] - Cache read

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | e9 | e8 | e7 | e6 | e5 | e4 | e3 | e2 | e1 | e0 |

*Table 12 LLC_EVENT_COUNTER_MASK register.*

### 4.4.13 LLC_GLOBAL_ALLOC

This register controls whether lines can be allocated into a way by any agent.

If a way is disabled from allocation in this register, no agents can allocate even if the llc_class_allocate registers are set. This register is used as part of a sequence to remove ways from use by the cache for either Scratchpad RAM usage, or for power gating. By removing allocation ability, a flush engine can remove the existing contents of the line without fear that new entries will be added during or after the flush.

The default value of this register enables allocation for all ways of the cache, and so each bit corresponding to a way is set to 1. To disable allocation, the bits should be set to 0.

Attribute: RW

Bit field description:

Global_Allocation_Way_Enable[31:0] - Global Allocation Way Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Global_Allocation_Way_Enable | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 13 LLC_GLOBAL_ALLOC register.*

### 4.4.14 LLC_INDIRECT_RAM_CONT

This is the indirect access RAM content register. It is used in conjunction with the indirect access trigger register. On an indirect read, data is written to this register. On an indirect write, content from this register is written into the RAM. On a read-modify-write, content from this register is used for the XOR function.

Since the RAM data width may be larger than 64 bits, multiple registers are used to hold the data. Any bits beyond the data width are unused.

Attribute: RW

Bit field description:

RAM_content[63:0] -

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RAM_content | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAM_content | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 14 LLC_INDIRECT_RAM_CONT register.*

### 4.4.15 LLC_INDIRECT_TRIGGER

This register is the indirect access trigger. Indirect access is a mechanism that allows register-based access to the RAM arrays. This can be used for testing RAM bits or reading content on an error condition.

The indirect access is based on a content+trigger mechanism. For writes, the content register is written first to accumulate the data that should be written. Once the content is ready, the trigger

register is used to kick off the hardware write mechanism. For reads, the trigger register kicks off a read, and provides data by placing the result into the content registers where it can be accessed.

The indirect access supports 4 sub-commands.

Read Raw data. When triggered, a read to the RAM array will be performed and the resulting data, without ECC correction, will be copied into the content register.

Write Raw Data. When triggered, the content register values will be written into the RAM. This will include the ECC bits if present.

Write Data with Generated ECC. When triggered, this will write to the RAM entry. The content register will be used to specify the data to be written. However, if ECC hardware is present, the ECC bits will be generated based on the data instead of coming from the content register. This allows the RAM entry to be written with correct ECC value without needing to calculate it first.

Read-Modify-Write. This command will perform a specific kind of read-modify-write operation on a RAM entry. It will read the content of the RAM, XOR that content with the indirect content register, and write the combined value into the same RAM entry. This can be used to introduce single or double bit errors into the directory to test error detection and handling. The content register will not be modified during this operation, so it can be used to introduce errors into multiple lines.

Each of the indirect access commands can be issued during normal operation, but the Write commands can have side-effects that break coherency functionality. The Read Raw is not disruptive, and the Read-Modify-Write can be performed atomically so single-bit errors can be introduced while maintaining functionality. The indirect access trigger registers are readable and writeable. To trigger the RAM access, this register must be written. Reads will not have side-effects and will only return the current value of the trigger register.

The trigger register has a number of fields that must be set correctly. The CMD field indicates which kind of indirect access to perform. The WAY field indicates which way group to access. The TYP field indicates whether the Data array or Tag array should be accessed. If the Data array is accessed, the hlf bit indicates which sub-bank is accessed. The RAM index indicates the entry to access within the RAM.

Attribute: RW

Bit field description:

Index[31:11] - Index of RAM to access

way[10:9] - Way position in the waygroup

hlf[8] - Which half of cache line reported error

waygroup[7:3] - Way group of first detected error

typ[2] - 0: Tag array access
- 1: Data array access

cmd[1:0]

- 00: Read raw array content including any ECC bits and copy to RAM Content register
- 01: Write RAM content register directly into array
- 10: Write RAM content minus ECC bits to array, use ECC generation logic to set ECC bits in array

11: Read-modify-write. Read array content, XOR with RAM content register, and write modified data into array

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Index | | | | | | | | | | | | | | | | | | | | | way | hlf | waygroup | | | | | | typ | cmd | |

*Table 15 LLC_INDIRECT_TRIGGER register.*

### 4.4.16 LLC_INTERRUPT_ERR

This is a status register that tracks the interrupt generating events. This includes multi-bit ECC error, single-bit ECC error, RAM mode disallowed accesses, and event counter overflow. When these events occur, this register is updated and will hold the bit value until cleared. It can be cleared by writing to the register. To allow per-bit clearing control, the write value should use a value of 1 when it doesn't want to make a change, or a bit value of 0 when it wants to clear.

Attribute: WZC

Bit field description:

e6[6] - Event Counter Overflow

e5[5] - Scratchpad Security Failure status

e4[4] - Scratchpad RAM Disabled status

e3[3] - Data ECC Double Bit Error status

e2[2] - Data ECC Single Bit Error status

e1[1] - Tag ECC Double Bit Error status

e0[0] - Tag ECC Single Bit Error status

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | | e6 | e5 | e4 | e3 | e2 | e1 | e0 |

*Table 16 LLC_INTERRUPT_ERR register.*

### 4.4.17 LLC_RAM_ADDRESS_BASE

This register indicates the system address offset of the RAM mode. The address range should always be allocated as the full size of the LLC capacity rounded up to a power of 2, and the address offset must be programmed to a naturally aligned address for that size. The default value of this register is the address range base specified during NoC construction.

Attribute: RW

Bit field description:

- **Address**[63:0] - Address

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 17 LLC_RAM_ADDRESS_BASE register.*

### 4.4.18 LLC_INTERRUPT_MASK

This register is used for determining what kind of events can trigger an interrupt from the LLC.

A bit value of 1 indicates that the event will not send an interrupt. A bit value of 0 means the event will cause an interrupt. The default values are listed in the bit field description.

Attribute: RW

Bit field description:

m6[6] - 1'b1: Event Counter Overflow interrupt disabled (default).
- 1'b0: Event Counter Overflow interrupt enabled.

m5[5] - 1'b1: Scratchpad Security Check Failure interrupt disabled.
- 1'b0: Scratchpad Security Check Failure interrupt enabled (default).

m4[4] - 1'b1: Scratchpad RAM Access while Disabled interrupt disabled.
- 1'b0: Scratchpad RAM Access while Disabled interrupt enabled (default).

m3[3] - 1'b1: Data ECC Double Bit Error interrupt disabled.
- 1'b0: Data ECC Double Bit Error interrupt enabled (default).

m2[2] - 1'b1: Data ECC Single Bit Error interrupt disabled (default).
- 1'b0: Data ECC Single Bit Error interrupt enabled.

m1[1] - 1'b1: Tag ECC Double Bit Error interrupt disabled.
- 1'b0: Tag ECC Double Bit Error interrupt enabled (default).

m0[0] - 1'b1: Tag ECC Single Bit Error interrupt disabled (default).
- 1'b0: Tag ECC Single Bit Error interrupt enabled.

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | | | | | | | | | | | | | | | | | | | | | | | | | m6 | m5 | m4 | m3 | m2 | m1 | m0 |

*Table 18 LLC_INTERRUPT_MASK register*

#### 4.4.19 LLC_RAM_WAY_ENABLE

This register is used to enable cache ways to be used as a Scratchpad RAM instead of a cache. The register indicates which of the ways should be used as a RAM instead of a cache. It is possible to use some of the LLC as a cache, and some as a RAM, by selecting which ways are used by each. By default, this register is set to 0 so that all ways are used as cache. To set this register, the lines must be removed from cache usage by the llc_cache_way_enable register. Any cache contents should be flushed before enabling the RAM mode.

Attribute: RW

Bit field description:

- **Scratchpad_Ram_Way_Group_Enable**[31:0] - Scratchpad Ram Way Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Scratchpad_Ram_Way_Group_Enable | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 19 LLC_RAM_WAY_ENABLE register.*

### 4.4.20 LLC_RAM_WAY_SECURE

This register allows the Scratchpad RAM to have trust-zone security checking. Each way can be individually controlled. If the security bit is set, only secure accesses (those with AxPROT[1] set to secure) can access that address. Non-secure accesses will be responded to with an error, and an interrupt will be triggered if the interrupt is enabled.

A value of 1 indicates secure accesses are required, while a value of 0 indicates non-secure accesses are required. By default, this register is 0, so non-secure accesses are allowed.

Attribute: RW

Bit field description:

- **Scratchpad_Ram_Way_Group_Security**[31:0] - Scratchpad Ram Way Security

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Scratchpad_Ram_Way_Group_Security | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 20 LLC_RAM_WAY_SECURE register.*

### 4.4.21 LLC_TAG_INV_CTL

The Tag Invalidation Control register triggers a state machine that will invalidate the contents of one or more waygroups of the cache. The register has one bit per way group, and the bit vector written into this register will invalidate the corresponding way groups. A value of 1 will indicate that the corresponding way group should be invalidated. A value of 0 will indicate that the way group should not be invalidated. This per way group control allows portions of the cache to be powered down and restarted later, with the ability to reset just the way groups that were powered down and powered back on.

A write to the register will kick off the invalidation engine, invalidating the specified way groups. A read of the register will indicate whether the invalidation is in progress. When the invalidation engine has completed, the bit vector will transition to a value of zero. So, a read value of zero will indicate that the state machine has completed. The reset value of this register is zero. An invalidation sequence should be completed before a second sequence is requested.

Attribute: RW

Bit field description:

Tag_Way_Invalidation_Enable[31:0] - Tag Way Invalidation Enable

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Tag_Way_Invalidation_Enable | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 21 LLC_TAG_INV_CTL register.*

### 4.4.22 LLC_WAY_FLUSH

This register controls a state machine that flushes specified ways of the cache. The intent of this engine is to remove all content from the specified ways, pushing any dirty data that may exist to memory. It also invalidates clean lines. The Way Flush engine should be run while the llc_cache_way_enable is still on for those ways so the contents are still accessible, but the llc_global_alloc register should have disabled the way for allocation. This ensure that as lines are removed from the cache, they won't unintentionally get added again. Clean lines are invalidated to ensure that dirty line writes do not write into the ways being flushed.

Writing the register will kick off the flush engine. If the write value specifies a bit value of 1, then that way group will be flushed. If the bit value written is zero, that way will not be flushed. When the sequence is completed, hardware will transition the bit values to zero. A register value of 0 indicates the state machine has complete. The default value for this register is zero.

A flush sequence should be completed before a second sequence is requested.

Attribute: RW

Bit field description:

Way_Flush_Control[31:0] - Way Flush Control

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Way_Flush_Control | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 22 LLC_WAY_FLUSH register.*

2870 Zanker Road,

Suite 210,

San Jose, CA 95134

(408) 617-5209

http://www.netspeedsystems.com