NETSPEED SYSTEMS

NetSpeed Pegasus Last Level Cache

Technical Reference Manual

Version: Pegasus-16.04.a

June 24, 2016

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

Contents

1 INTRODUCTION

Pegasus is a highly customizable and configurable last level cache that eliminates memory bottlenecks and boosts overall system performance.

Pegasus can lower latency between critical IP blocks by placing Pegasus LLCs in design, increase memory efficiency by using Pegasus as write-buffer and lower power by reducing expensive access to off-chip memory.

Pegasus allows architects significant control of their design by supporting a multitude of flexible cache hierarchies.

Pegasus can be configured to operate as a coherent cache to reduce latency for coherent cache accesses. By removing unnecessary lookups, the performance of the Last level caches are improved and dynamic power consumption is reduced.

Pegasus can be configured as a memory cache to give traffic with good locality bandwidth increase. Memory cache does not require the flushing of the cache and will speed up software-coherency use cases and improve memory bandwidth during transitions.

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 ARCHITECTURE

The 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.

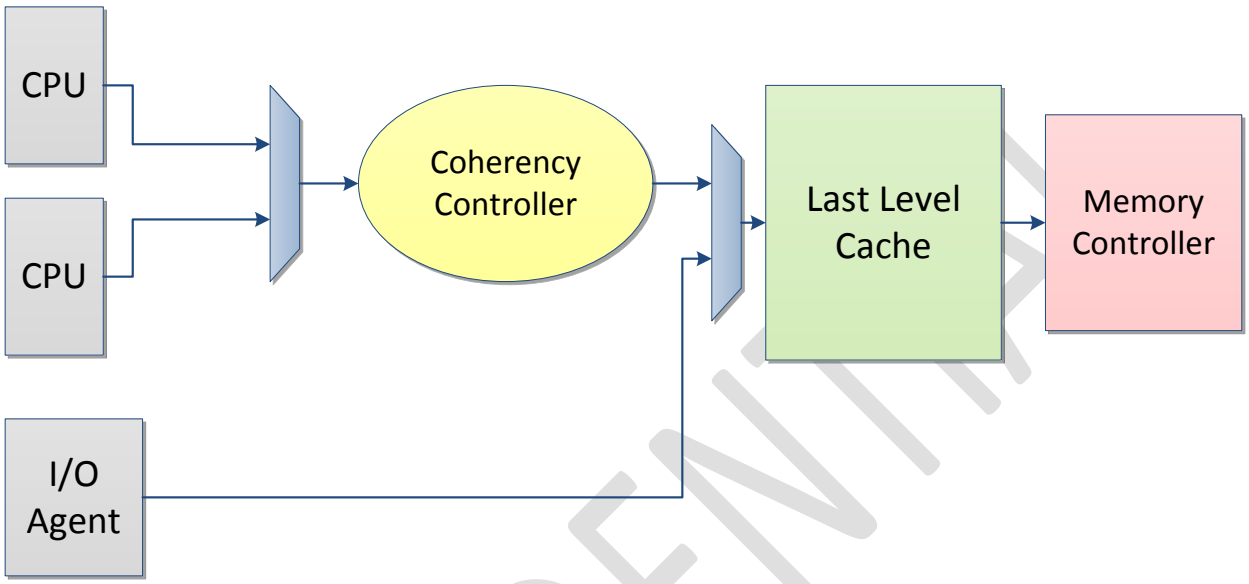

Figure 1: Last Level Cache sits behind coherency controller

This is a non-integrated LLC design, which means the LLC is a separate module from the coherency controller and access to them happens in serial. An integrated cache would allow parallel lookups of the directory and cache tags, allowing for a shorter latency. The non-integrated design is less complex and allows for independent decisions about sizing and number of instances.

Since the LLC is not integrated with the coherency controller, it can be utilized even in non-coherent systems as a generic 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.

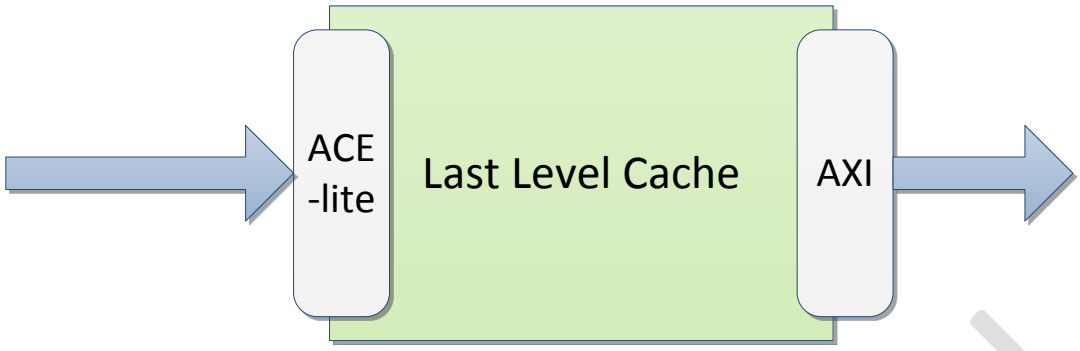

Figure 2: LLC has an ACE-lite and AXI port

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.

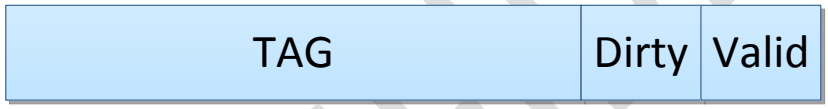

Figure 3: State Tracking in LLC Tags

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.

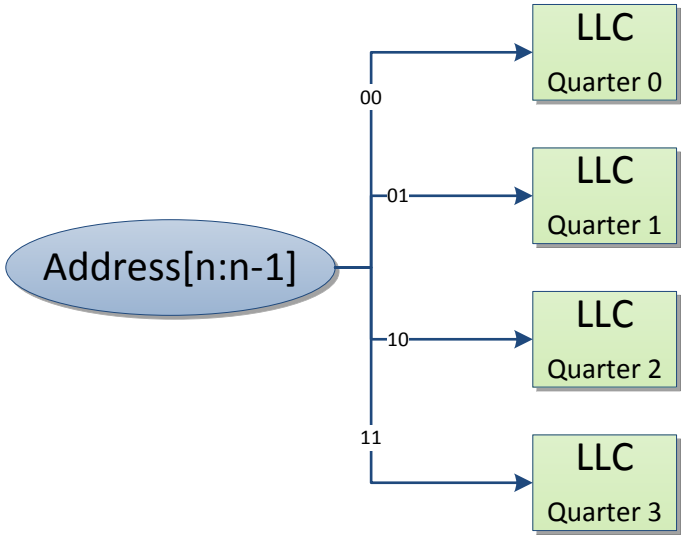

Figure 4: Multiple Parallel LLCs

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.

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.

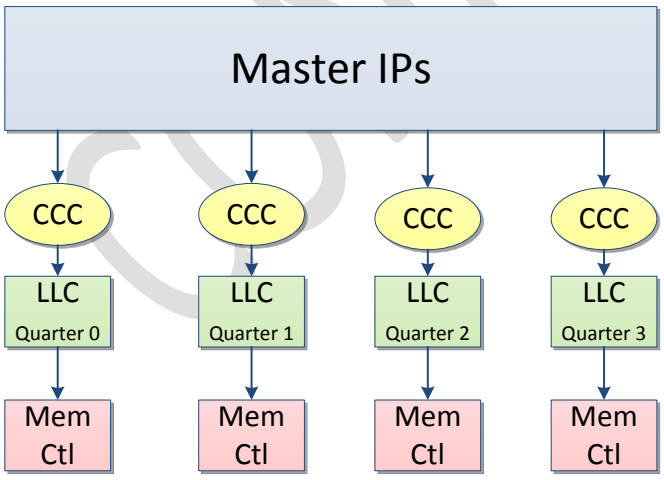

Figure 5: Dedicated caches connectivity

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.

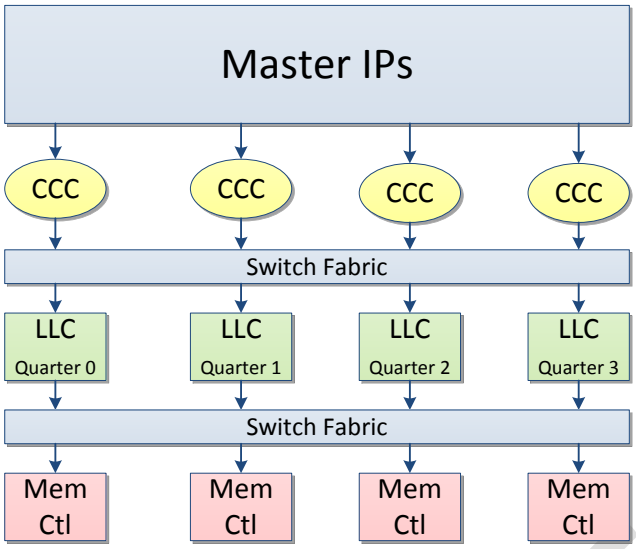

Figure 6: Flexible Connectivity

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. Since IP must be general purpose and usable across different fabrication processes, the LLC must be built to have enough flexibility to use different RAM designs.

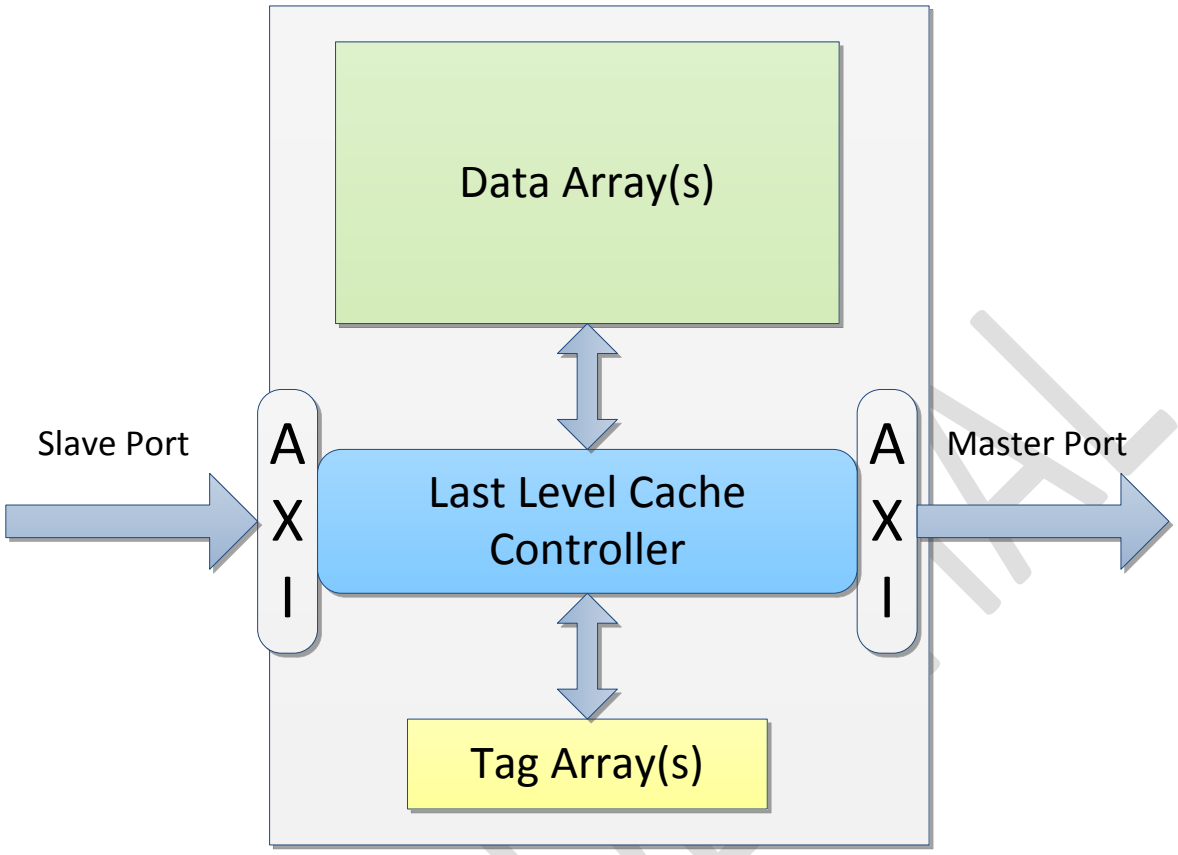

Figure 7: LLC has Controller, Tag Arrays and Data Arrays. Controller has separate interface to access the arrays.

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

The controller must be 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.

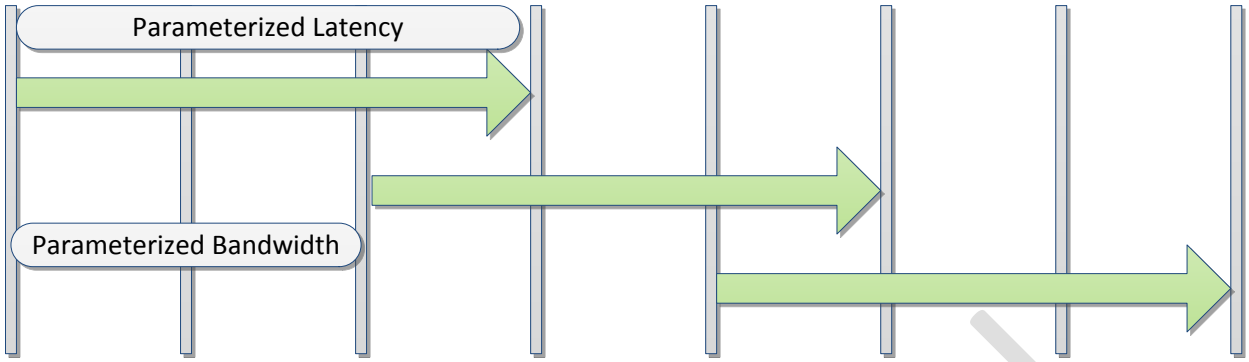

Figure 8: RAM access show flexible latency and repeat rate

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. For the controller to take advantage of the banking, it must recognize which portions of the data array are assign to each bank, and track whether a bank is busy on any particular cycle.

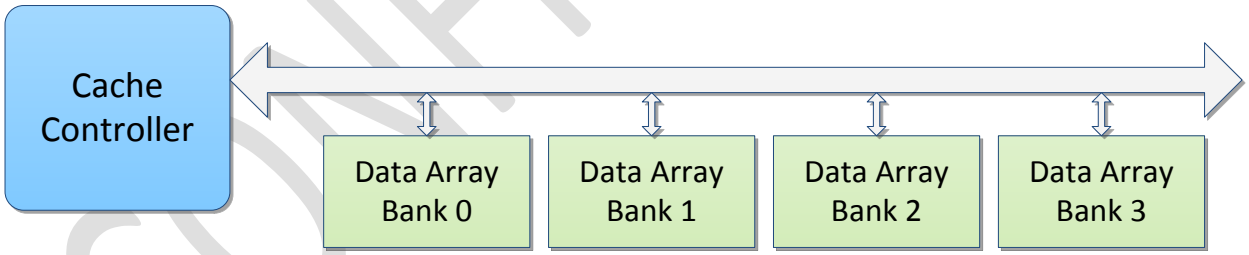

Figure 9: Cache with banked data arrays

Normally data array banks can be split by address or by cache ways. Splitting by ways is convenient because it allows a relatively easy method of power-gating sections of the data array. Ways can be easily disabled, flushed, and power gated. Re-enabling the bank is easy and does not affect the enabled arrays. This design will only support banking by ways.

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.

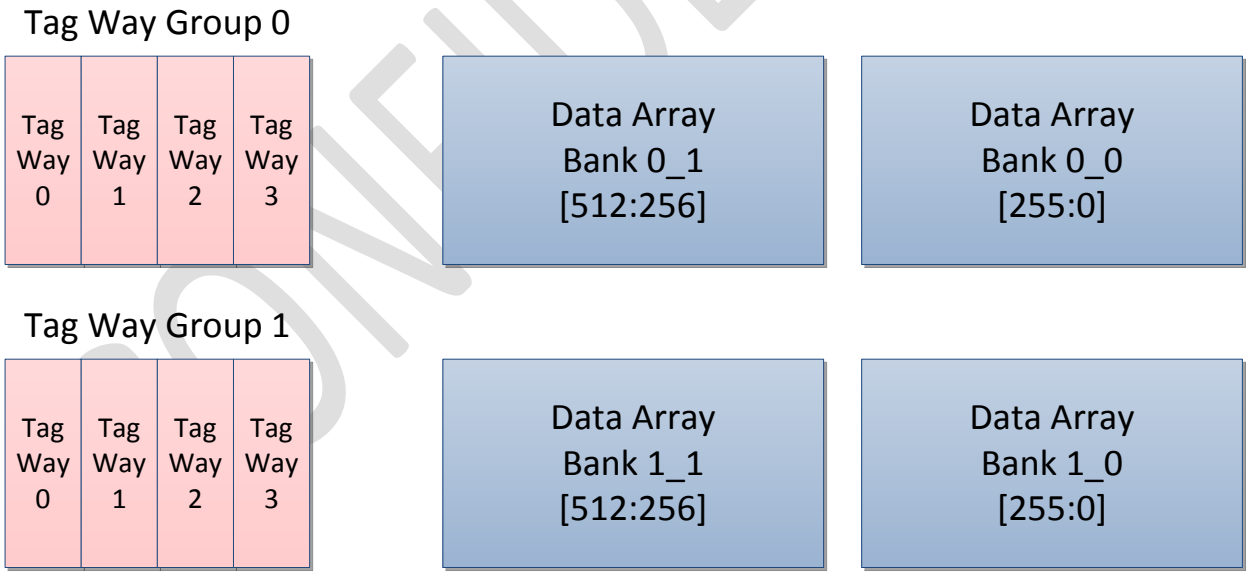

Figure 10: Ways and Banking are related

One of the reasons for this association is power reduction. If a portion of the cache wants to be turned off, it can be reduced by turning off Way Groups. Since the Tags and Data Banks are associated, turning off a Way Group also allows powering off a Data Bank, with no loss of functionality.

2.10 SCRATCHPAD MODE

Since the L3 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.

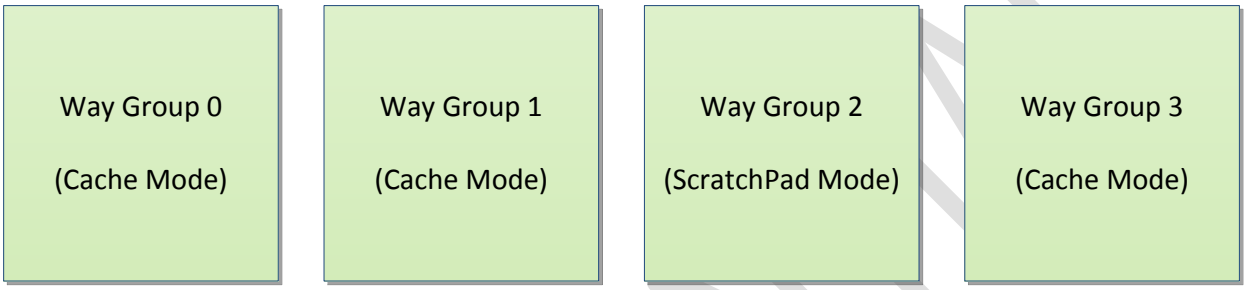

Figure 11: Portions of cache can be modified to act as a scratchpad RAM

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 Allocation and AxCACHE bits

The LLC uses the AxCACHE bits [3:2] to indicate whether a line should be allocated in the cache. If the line is already present in the cache, these bits are ignored and data is read from the cache.

2.13 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, which is why read-modify-write operation is required. This operation is utilized in normal cache operation as well.

2.14 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.

2.15 Agent Allocation Vectors

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.

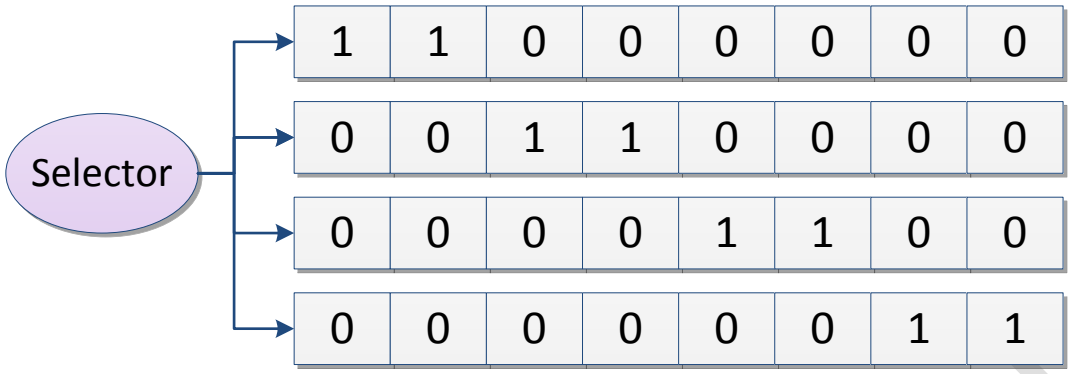

Figure 12: Programmable allocation vectors

A handful of programmable allocate vector registers can be used to decide which ways are available to an agent. Request can be sent with selector bits to decide which vector to allocate with. 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 work through the entries in that way and evict them until the entire way is empty.

2.16 ECC SUPPORT

Both tag and data arrays can be protected with ECC. This is a configurable options.

2.17 CONFIGURABLE INDEX AND TAG BITS

A configurable cache always has to have configurability on how many bits of the address are index bits and how many are tag bits.

On top of this flexibility, the LLC needs to have even more control. 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.

2.18 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.

2.19 Cache Maintenance Instructions

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

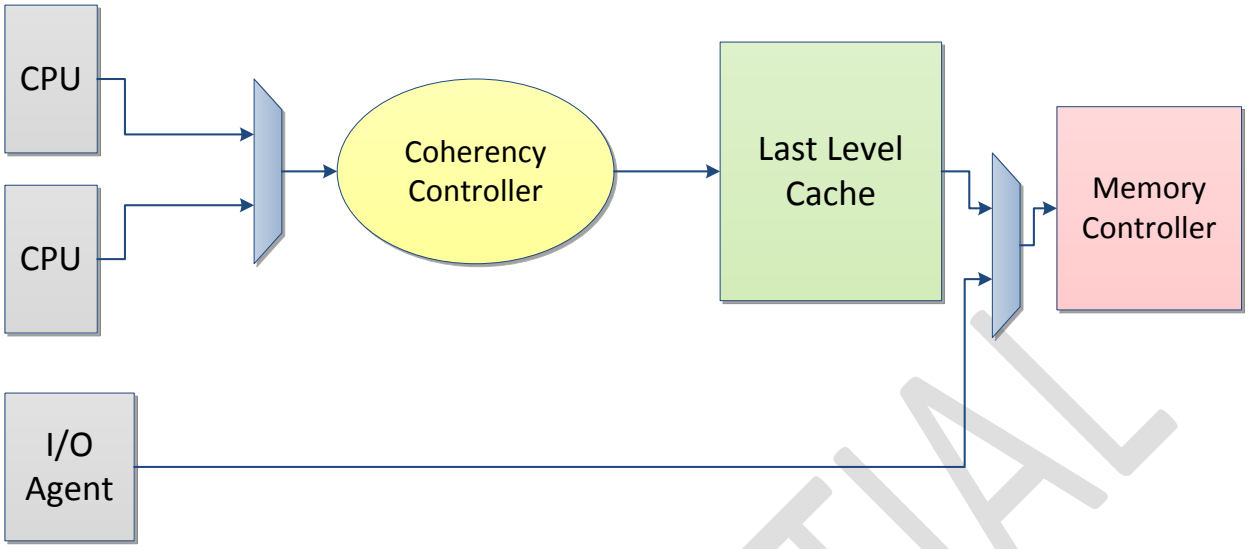

Figure 13: Non-coherent DMA bypasses LLC

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.

2.20 LLC Way Allocation Controls

LLC supports way allocation controls. A new property called LLC Allocation Class is added to each master bridge that can talk to the LLC. This can be set to one of 8 allocation classes. The LLC has a control register for each allocation class to determine which of the waygroups a line can be allocated into by that agent. This can allow soft-partitioning of associativity in the LLC. The LLC Allocation Class is controlled by a bridge property.

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 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_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_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.
- **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.

There is a 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.

3.3 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.

A 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.

2670 Seely Ave
Building 11
San Jose, CA 95134
(408) 914-6962
http://www.netspeedsystems.com