# 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.

## 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.

## 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.

## 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

## 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.

## 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.

## 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.

## 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.

## 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.

## 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

## 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.

## 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.

## 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.

## 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.

## 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.

### 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.

### 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

### 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.

## 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.

## 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.

## 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.

## 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.

## 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