# NocStudio Commands and Properties for LLC

CFG’s NocStudio allows the instantiation of last level caches and the configuration of several of its properties.

## Adding a Last Level Cache

CFG’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*.

## 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.

## 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 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.

## 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.

## 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.