# **DVM Registers**

## ACTIVE\_VECTOR\_0/1/2/3: Register indicating active DVM agents in the system

The DVM IP module is parameterized for a maximum number of agents supporting DVM in the system. This parameterization is taken care of by NocStudio. When reset is de-asserted, the DVM IP module expects that all specified DVM agents in the system are active. This is reflected in this readable, writable ACTIVE\_VECTOR register. If, due to low power mode, or other reasons, a DVM agent in the system is shut down, the DVM IP module needs to be made aware of this. To do this, the bit position of the DVM agent in this ACTIVE\_VECTOR register should be set to 0.

On seeing a 0 for an agent in the ACTIVE\_VECTOR, the DVM IP module ensures that no snoops are sent to it, and does not wait for snoop responses or completions from this agent.

The maximum number of DVM agents supported is 256. The entire vector can hence take from one to four 64-bit registers. Unused bits within the 64-bit registers are tied to 0.

Each agent, or host, in the system is assigned a corresponding bridge ID, based on the bridge it is connected to. The active vector register is based on this bridge ID.

ACTIVE\_VECTOR\_0 contains the vector for agents with bridge ID 0 to 63.

ACTIVE\_VECTOR\_1 contains the vector for agents with bridge ID 64 to 127.

ACTIVE\_VECTOR\_2 contains the vector for agents with bridge ID 128 to 191.

ACTIVE\_VECTOR\_3 contains the vector for agents with bridge ID 192 to 255.

For example, if the host at bridge ID=2 is being shut down, bit 2 of ACTIVE\_VECTOR\_0 should be set to 0. If the host at bridge ID=68 is being shut down, bit 4 of ACTIVE\_VECTOR\_1 should be set to 0.

## FAULT\_LOG\_0/1/2/3: Register logging DVM agents returning snoop responses

According to the AMBA spec on Distributed Virtual Memory transations, when an agent in the system receives a DVM transaction, it must respond in one of two ways.

* If it can perform the requested action, it must respond with CRRESP = 0b00000.
* If it is unable to perform the requested action, it must respond with CRRESP = 0b00010.

The FAULT\_LOG register logs which of the agents responded with CRRESP=0b00010. It is a sticky register, so once set, it remains set until cleared by writing 0 to that bit.

The maximum number of DVM agents supported is 256. The entire fault log can hence take from one to four 64-bit registers. Unused bits within the 64-bit registers are tied to 0.

Each agent, or host, in the system is assigned a corresponding bridge ID, based on the bridge it is connected to. The fault log register is based on this bridge ID.

FAULT\_LOG\_0 contains the fault log for agents with bridge ID 0 to 63.

FAULT\_LOG\_1 contains the fault log for agents with bridge ID 64 to 127.

FAULT\_LOG\_2 contains the fault log for agents with bridge ID 128 to 191.

FAULT\_LOG\_3 contains the fault log for agents with bridge ID 192 to 255.

For example, if the host at bridge ID=2 returns CRRESP=0b00010 for a transaction, bit 2 of FAULT\_LOG\_0 will be set to 1. If the host at bridge ID=68 returns CRRESP=0b00010 for a transaction, bit 4 of FAULT\_LOG\_1 will be set to 1.

## STS: Status register

This register indicates the “busy” or non-idle state of the internal logic within the DVM IP module. When all traffic has been serviced, and no new transactions are received, the bitwise-AND of these bits will be 0, indicating a state of quiescence.

|  |  |
| --- | --- |
| Bit position | Internal Status Location |
| 0 | Input Fifo not empty |
| 1 | DRT entry 0 not empty |
| 2 | DRT entry 1 not empty |
| 3 | DRT entry 2 not empty |
| 4 | DRT entry 3 not empty |
| 5 | DRT entry 4 not empty |
| 6 | DRT entry 5 not empty |
| 7 | DRT entry 6 not empty |
| 8 | DRT entry 7 not empty |
| 9 | Sync tracker sending snoops |
| 10 | Sync tracker waiting for completions |
| 11 | Sync tracker returning sync response to originating agent |
| 12 | Snoop arbiter not idle |

## DVM Agent Disable Register

This status register is used to indicate that a change to the active agent vector has taken effect. When an agent is removed from the active agent vector, snoops targeting that master may still be queued or inflight. While snoops are outstanding, the master must continue to operate to satisfy snoop requests.

This register indicates that all snoops to the disabled agents have completed and no new snoops will be issued to those agents. The status indicates whether any snoops outstanding at the time of the last modification of the DVM active vector register(s) have all completed or not. A value of 1 indicates that the snoops have all completed and it is safe to power off or otherwise disable the masters that were removed from the active vector. A value of 0 indicates that snoops are still outstanding and the master must continue to operate and accept snoops.

# **CCC Registers**

## ccc\_spec\_fetch 0-7

This register controls the speculative fetch behavior of the Cache Coherency Controller. Speculative fetch means that the CCC will issue reads to memory or next level of cache before the directory lookup is performed or the snoops are sent. This allows a lower latency for these reads in the case of a directory miss. If snoops need to be sent for this request, data may be retrieved from a caching agent. This means the speculative fetch may be increasing memory bandwidth in case where it wouldn’t have needed to send a read.

This set of registers specifies a bit vector where each bit corresponds to a master agent on the NoC. When requests from this agent arrive at the CCC, a comparison with the bit vector will determine whether speculative fetch is currently enabled for that agent.

The bit vector is sized based on the number of masters in the system. The bit position of each agent is determined by the agent’s bridge ID. Unused bits are not included in the register and cannot be written.

A value of 1 means speculative fetch is enabled for that agent. A value of 0 means speculative fetch is disabled for that agent. The initial values are determined by the bridge property cc\_axi4m\_speculative\_fetch.

## ccc\_active\_vector\_0

This register specifies which of the ACE masters are currently active and capable of receiving snoop requests. This register contains a bit vector where each bit corresponds to an ACE master bridge. The Bridge ID determines which bit corresponds to each bridge.

This vector can be used to disable snoop-only ACE agents, or to power off an ACE agent that may still have stale directory content. Note that ACE CPUs will often leave stale directory content.

A value of 1 indicates that the ACE agent is enabled and capable of accepting snoops. A value of 0 indicates that the ACE agent is disabled and snoops should not be sent. The default for this register is all ACE agents are active.

## ccc\_ecc\_disable

This control register can be used to disable ECC correction and detection, if the IP is configured to have ECC. If no ECC is present, the control register doesn’t do anything. All other bits are unused.

A value of 1 disables ECC. A value of 0 indicates ECC is enabled when available. Default value is enabled.

## ccc\_hash\_bypass

This control register allows the directory to avoid index hashing. The directory is designed with two different indexing mechanisms. For way 0, the index bits specified in the configuration are used directly from the address. For way 1, additional address bits XORed with the index bits. This hashing is used to reduce the probability of conflicts within the directory.

The hash bypass register will skip the hashing that is performed for way 1. This can allow a more straightforward debug. It can also be used with the indirect access control registers to enable a more intuitive access of the RAM array.

A change of this bypass will change the lookup mechanism of the RAM. Since it changes the location where addresses can reside, this bypass should only be used when the directory is invalid or during debug operations. If changed during normal operation, coherency will likely be violated.

A value of 0 indicates hashing should be used, which is also the default. A value of 1 indicates hashing should be ignored during.

## ccc\_agent\_disable\_status

This status register is used to indicate that a change to the active agent vector has taken effect. When an agent is removed from the active agent vector, snoops targeting that master may still be queued or inflight. While snoops are outstanding, the master must continue to operate to satisfy snoop requests.

This register indicates that all snoops to the disabled agents have completed and no new snoops will be issued to those agents. The status indicates whether any snoops outstanding at the time of the last modification of the ccc\_active\_vector register(s) have all completed or not. A value of 1 indicates that the snoops have all completed and it is safe to power off or otherwise disable the masters that were removed from the active vector. A value of 0 indicates that snoops are still outstanding and the master must continue to operate and accept snoops.

## ccc\_llc\_control

When the CCC is connected to a Last Level Cache, it has some unique programmable features that allows the two to interact more closely. This register allows programmable control of these features.

The CM bit is the enable for the Cache Maintenance Operation propagation. A Cache Maintenance Operation flushes or invalidates a cache line from the coherency domain. This can allow interaction with non-coherent devices that access the slave directly. Since a Last Level Cache can hold additional data, it may be necessary to flush or invalidate lines from the LLC to memory. When this register bit is set, the CCC will propagate any Cache Maintenance Operations to the LLC, where the cache can take the specified action. When this bit is set to zero, the Cache Maintenance Operations will not be propagated.

The WE bit is the enable for Write Evict propagation. In ACE protocol rev E, the WriteEvict command was created in order to write clean data to a downstream cache, such as the Last Level Cache. This allows the cache to only allocate a line when it is dropped by one of the ACE masters, allowing a better utilization of RAM storage. When the WE bit is set to 1, the WriteEvict will be propagated from the CCC to the LLC. If set to 0, the WriteEvict will drop the data and only update the CCC directory to indicate that the master has given up its copy of the line.

## ccc\_qos\_overrides

This control register allows the QoS field of requests sent from the CCC to be overridden. By default, read requests sent from the CCC use the same QoS value as the read request sent to the CCC that triggered the read. For writes, the QoS value is dependent on the cause of the writes. If the write on the master port is triggered by a WriteUnique, WriteBack, WriteClean, or WriteEvict, the QoS field is determined by the originating write. Writes are also generated by the CCC when a snoop response with dirty data is returned for a read request. The default value of the QoS is the QoS value of the request that triggered the snoop. And finally, if the directory needs an eviction due to storage conflict, the directory eviction needs a QoS value.

The control register controls each of these four QoS values. For AR requests, Write requests, and Snoop responses, there is an override field and an enable to determine whether to use the default value or the override value. For Directory Evict, the DirEvt QoS value in this register always determine the QoS value of the generated write request, with a default value of 0.

This QoS override register is unrelated to the master port bridge QoS override, which if enabled can override any of these values.

## ccc\_directory\_inv

This is a control register that can be used to invalidate the entire directory. Setting this register will kick off a hardware engine that will block normal coherent traffic and invalidate all entries of the directory.

The register also acts as a status register. When the control bit is set, it triggers the hardware state machine. The value of that register will stay high until the state machine completes. At that point, it will automatically transition to 0. Since coherent traffic will be blocked until the invalidation sequence has completed, it is not always necessary to check the status of this register.

Writing a value of 1 will trigger the invalidation engine, and it will transition to 0 when completed. All other bits are unused, and the default value is 0.

## ccc\_indirect\_access\_trig

This registers is the indirect access trigger. Indirect access is a mechanism that allows register-based access to the directory RAM. 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.

1. Read Raw data. When triggered, a read to the directory RAM will be performed and the resulting data, without ECC correction, will be copied into the content register.
2. Write Raw Data. When triggered, the content register values will be written into the directory RAM. This will include the ECC bits if present.
3. 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 a directory entry to be written with correct ECC value without needing to calculate it first.
4. Read-Modify-Write. This command will perform a specific kind of read-modify-write operation on a directory entry. It will read the content of the directory, XOR that content with the indirect content register, and write the combined value into the same directory 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 is 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 whether RAM 0 or 1 should be accessed. The RAM index indicates the entry to access within the RAM. The RAM index is a pure index, which will avoid any hashing function for WAY 1.

## ccc\_indirect\_ram\_cont\_0-7

This is the indirect access RAM content register. Its use is 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.

## ccc\_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 directory miss for a Shared line request. The events that can be counted are all related to directory 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 ccc\_event\_counter\_value 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.

## ccc\_event\_counter\_value

This register holds the current event count. As selected events occur, the count will increase. When this register rolls over, it can generate an interrupt if the interrupt mask is set correctly.

The register can be read or written through register access. By writing the register, a counter can be clear. It can also be set to a value to force an overflow earlier, to create an interrupt when desired.

## ccc\_crt\_status

This register tracks the current state of the Coherency Request Trackers. These are the state-machines that track coherent request from the time they are started to the time they are completed. This can be used to determine if there are requests that have made it to the CCC but are not done being processed.

A bit value of 1 indicates a request is still in flight. There are 32 CRTs in a CCC. The reset value for this register is 0, and that is the value under idle conditions where no request are outstanding.

This is a read-only register.

## ccc\_ecc\_info

This register monitors ECC errors and saves information for potential debug support. The register holds two kinds of information. It keeps a count of all ECC errors that have been detected in a 16 bit ECC counter.

It also tracks at least one RAM location where an ECC event has occurred. The first ECC even to occur will update the SB or DB bits and the index. The SB stands for single-bit, and connotes a correctable error. DB is double-bit, and applies to all uncorrectable error cases, which include double-bit corruption or more. The register also tracks which of the two RAMs the error was detected in.

While the first error of either kind will be added to the register, and the first uncorrectable error will overwrite any status that was caused by an earlier single-bit event. This is because the uncorrectable error is typically fatal and the status is more important.

This register is readable and writeable, with default value of 0.

## ccc\_interrupt\_mask

This register is used for determining what kind of events can trigger an interrupt from the CCC. The four events it can control are:

1. Multi-bit ECC error
2. Single-bit ECC Error
3. Event Counter Overflow
4. Snoop Response Error

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 value is 4’b0110, which means only the Multi-bit error case and snoop response error case will send an interrupt, as they are expected to be fatal errors.

## ccc\_interrupt\_err

This interrupt is a status register that tracks the interrupt generating events. This includes multi-bit ECC error, single-bit ECC error, event counter overflow, and snoop response error. 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 value of 0 when it wants to clear.

# **LLC Registers**

## llc\_class\_allocate 0-7

The class allocation control registers are used to specify which way groups (a group of 4 associative ways) can be written to. Each master in the system belongs to an LLC class, and each class allocation control register indicates which way groups 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 group, none of the agents can allocate into those ways regardless of what the llc\_class\_allocate registers indicate.

## llc\_global\_alloc

This register controls whether lines can be allocated into a way group by any agent. If a way group 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 group is set to 1. To disable allocation, the bits should be set to 0.

## llc\_cache\_way\_enable

This register indicates whether a way group 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 group, allowing each way group to be individually enabled or disabled. A value of 1 indicates that the way group is enabled, while a value of 0 indicates it is disabled. All way groups are enabled by default. Before disabling a cache way group, the way group must be disabled in the llc\_global\_alloc register, and the contents should be flushed.

## 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.

## 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.

## 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.

## 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.

## 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 way groups 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 way groups 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.

## llc\_ram\_way\_secure

This register allows the Scratchpad RAM to have trust-zone security checking. Each way group 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 no security check is needed (secure or non-secure accesses are enabled). By default, this register is 0, so no security checking occurs.

## 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.

## llc\_tag\_inv\_ctl

The Tag Invalidation Control register triggers a state machine that will invalidate the contents of one or more way groups of the cache. The register has one bit per way group, and the bit vector written into this register will invalidate the corresponding ways. 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 control allows portions of the cache to be powered down and restarted later, with the ability to reset just the ways that were powered down and powered back on.

A write to the register will kick off the invalidation engine, invalidating the specified ways. 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.

## llc\_data\_inv\_ctl

This register controls a state machine that can invalidate the contents of data array banks. Each way group 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 way groups. Writing a value of 1 to a bit indicates that the corresponding way group should be invalidated. Writing a value of 0 to a bit indicates that the content of that way group 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 group, 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.

## llc\_way\_flush

This register controls a state machine that flushes specified way groups 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 invalidate 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 way groups 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.

## 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.

## 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.

## 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.

## llc\_indirect\_ram\_cont 0-4

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.

## 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.

1. 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.
2. Write Raw Data. When triggered, the content register values will be written into the RAM. This will include the ECC bits if present.
3. 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.
4. 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 is 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.

## 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.

## 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.

## llc\_interrupt\_mask

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

1. Tag ECC Single-Bit Error (Default Disabled)
2. Tag ECC Double-Bit Error (Default Enabled)
3. Data ECC Single-Bit Error (Default Disabled)
4. Tag ECC Double-Bit Error (Default Enabled)
5. Scratchpad RAM access while Disabled (Default Enabled)
6. Scratchpad Security Check Failure (Default Enabled)
7. Event Counter Overflow (Default Disabled)

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 above.

## 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.