Gen-Z DRAM and Persistent Memory Theory of Operation

Michael Krause, Mike Witkowski
{mkrause, mike.witkowski}@hpe.com

Gen-Z is a new data access technology that can significantly enhance memory solutions built with existing or emerging memory technologies. Gen-Z abstracts the memory media, breaking the processor-memory interlock to enable a virtuous innovation cycle and to provide numerous technical and economic benefits. This paper provides a high-level theory-of-operation description of Gen-Z-attached DRAM and persistent memory (PM).

Processor-Integrated Gen-Z

Example Processor with DDR and Gen-Z Memory Attached illustrates how Gen-Z could be integrated into a Requester such as a processor without impacting the traditional memory controller. For example, a DDR memory controller would continue to independently service a portion of the processor's address space, and Gen-Z would independently service a different portion. Depending upon the design, such a processor could deliver more than double the application memory bandwidth of a processor with only DDR support. For example, a processor with 8 DDR 5 6400 channels and 64 Tx / Rx 32 GT/s lanes Gen-Z could deliver ~800 GB/s of application bandwidth (~400 GB/s DDR plus ~400 GB/s Gen-Z), and 128 Tx / Rx 112 GT/s lanes of Gen-Z could deliver 3.2 TB/s of application bandwidth (~400 GB/s DDR plus ~2.8 TB/s Gen-Z). Gen-Z can deliver significant bandwidth without increasing component package size and pin counts.

To determine which interconnect is used to access memory, a subset of the processor's address space is mapped to DDR and a subset to Gen-Z.

- When an application performs a load or store operation, the address resolves to DDR or Gen-Z.
- Once resolved, the respective protocol logic is invoked to select the egress channel / interface to the memory module that contains the targeted data.
    - Application, middleware, operating system software changes are not required as all protocol-specific logic is handled in hardware.
    - Within an enclosure, system firmware or equivalent is responsible for configuring DDR and Gen-Z memory.

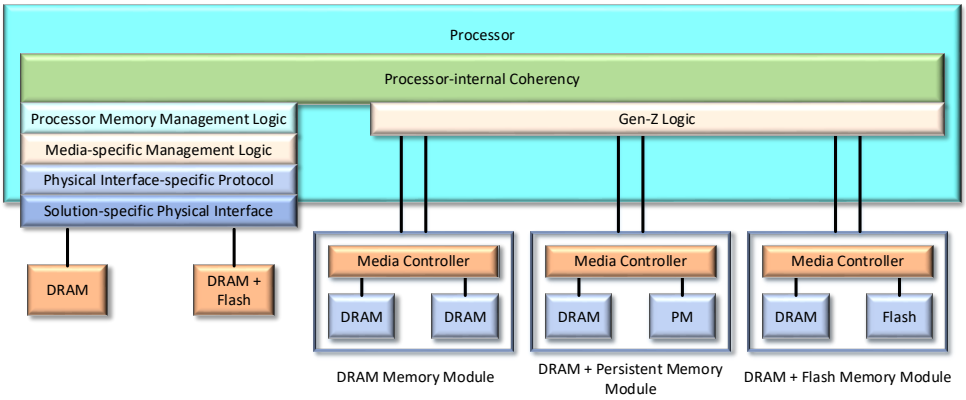

Figure 1: Example Processor with DDR and Gen-Z Memory Attached

Gen-Z OpClasses Used to Connect Memory Modules illustrates the basic topologies and the corresponding OpClasses used to generate request and response packets between a processor and a memory component.

- The P2P 64 OpClass is optimized for point-to-point topologies that contain a Requester such as a processor and one or more memory components.
- The Core 64 OpClass can be used in any topology and provides more robust capabilities than the P2P 64 OpClass.

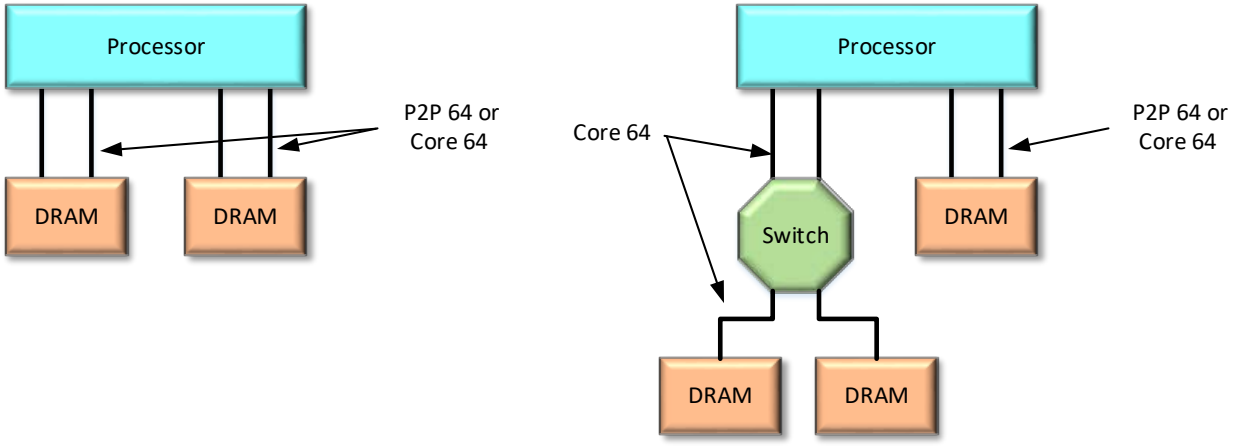

Figure 2: Gen-Z OpClasses Used to Connect Memory Modules

Example Physical Address Mapping to Processor-integrated Gen-Z Logic illustrates the functional blocks used to translate a processor physical address to a target-specific Gen-Z service.

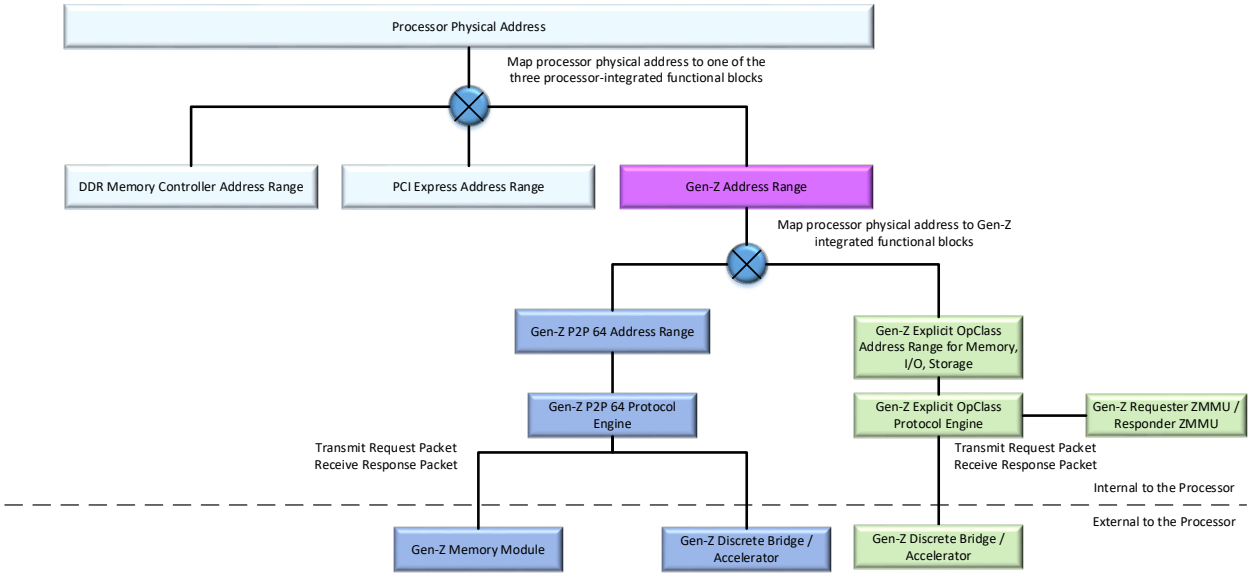

Figure 3: Example Physical Address Mapping to Processor-integrated Gen-Z Logic

The steps to translate a processor physical address are extremely light weight, and provide optimal latency and bandwidth. These steps are as follows:

1. The processor contains multiple integrated functional blocks located through different address ranges: DDR memory controller, PCI Express, and Gen-Z. Upon receipt of a processor request,

e.g., a load or store request, the logic maps the associated processor physical address to one of the functional blocks.

2. Within the Gen-Z functional block, there multiple address ranges:
    a. P2P 64 address ranges that correspond to memory, I/O, accelerator, storage, Smart I/O, etc. components that are directly attached to the processor and that use the P2P 64 OpClass to communicate.
    b. Explicit OpClass address ranges that correspond to memory, I/O, accelerator, storage, Smart I/O, etc. components that are fabric-attached (point-to-point or switch-based topologies) to the processor and that use explicit OpClasses, e.g., Core 64, to communicate.
3. If a processor request corresponds to a P2P 64 address range, then the following steps are taken:
    a. The physical address is mapped to one of the provisioned Requester P2P structures that corresponds to a Responder (memory, I/O, accelerator) that has been directly mapped by software (e.g., system firmware) to a processor physical address range.
    b. An egress interface is selected (components can support multiple interfaces to improve aggregate performance and resiliency), and the request is passed to the Gen-Z P2P 64 protocol engine, and a Gen-Z request packet is generated and transmitted.
4. If the processor request corresponds to the explicit OpClass address range, then the following steps are taken:
    a. The physical address acts as an input into the provisioned Requester ZMMU to identify a Requester PTE (Page Table Entry). Each Requester PTE is preconfigured by software with all of the information necessary to target a Responder and to translate the processor physical address to a Responder address and identifiers.
    b. Using information derived from the Requester PTE, the request is passed to the Gen-Z explicit OpClass protocol engine, an egress interface is selected (components can support multiple interfaces and multiple paths to improve aggregate performance, adaptively route around congestion and failures, and to improve resiliency), and the request packet is generated and transmitted.

Memory Media Abstraction

Example Media Controllers Supporting Different Media Types and Mix illustrates a variety of Gen-Z memory modules that support any type and mix of media types including DRAM, PM, and Flash. To avoid media-specific processor support, the media is abstracted through the Component Media Structure, which describes the memory component's attributes and capabilities, and controls its operation.

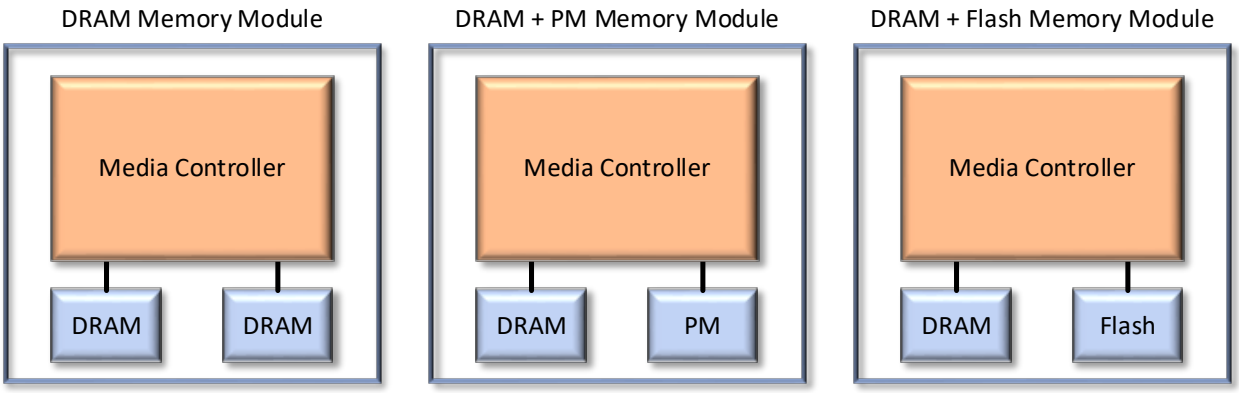

Figure 4: Example Media Controllers Supporting Different Media Types and Mix

Eliminating Stranded Memory

Memory resources become stranded, i.e., inaccessible, when a hardware failure prevents memory module access. For example, in a DDR-based system if the processor / memory controller or the DDR channel fails, then the channel-attached memory is inaccessible. To ensure application data resiliency, software-based data replication techniques can be periodically applied to create data snapshots or, for select applications, transaction intent logs. Though software-based data replication techniques have been successfully deployed for a number of years now, these techniques increase solution cost (e.g., requires higher performance processors, NICs, switches, etc. to handle the additional overheads required to replicate data or intent logs) and negatively impact application performance (applications need to wait until replication / intent log updates have been successfully completed and such replication requires additional software and I/O operations which can cause non-deterministic application performance). As a result, these techniques are best-suited for solutions composed of memory modules that contain 10s of GiB of volatile media.

In contrast, processors and memory modules that support Gen-Z can eliminate stranded memory resources, and therefore support any mix media types and capacity without requiring software-based data replication techniques and their associated cost, do not incur any performance loss, and are inherently simpler and resilient. *Example Topologies to Prevent Stranded Resources* illustrates three example Gen-Z-based topologies to prevent stranded memory resources.

- Topology 1 illustrates a memory module that is attached to two processors. If the memory module supports a single media controller (recommended), then all memory can be accessed by either processor in the event of failure. During normal operation the memory can be partitioned with one partition assigned to a single processor. If the memory needs to be shared, then the processors can use Gen-Z Atomic operations, software mutex, or processor-specific techniques to control shared memory access.
- Topology 2 illustrates a memory module that is attached to one processor and to a switch. Similar to the topology 1, the switch enables one or more local or remote Requesters to access the memory module.
- Topology 3 illustrates a switch-based topology with full redundancy as well as shared access by multiple processors. This topology can be used to implement fully-composable infrastructures / modular rack-scale solutions.

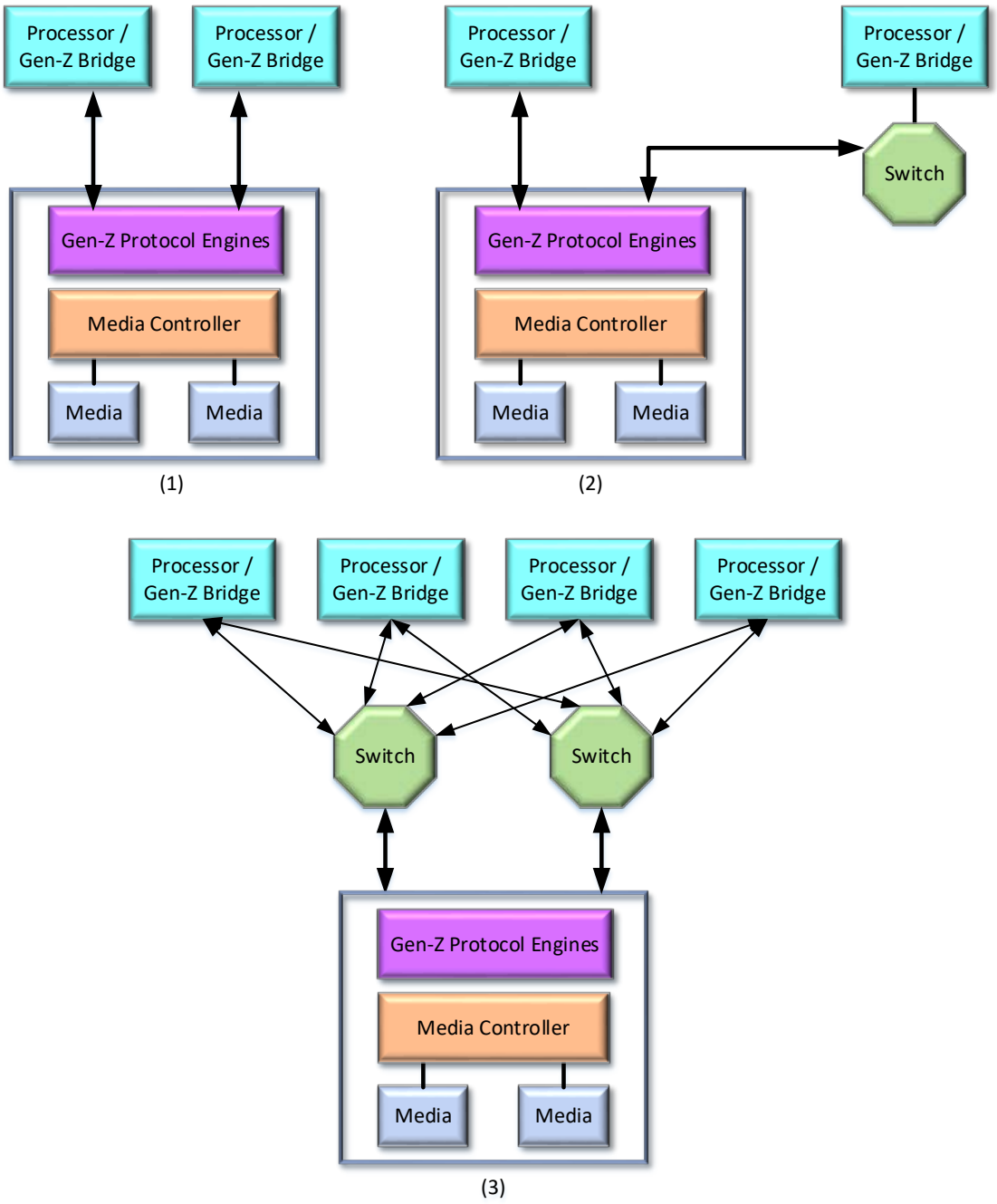

Figure 5: Example Gen-Z Topologies to Prevent Stranded Resources

Gen-Z Memory Component Fundamentals

Example Memory Component Functional Blocks illustrates the basic functional blocks within a memory component.

- A memory component supports one or more interfaces / links to provide connectivity. Depending upon the supported OpClasses, these links can connect directly to a Requester (processor, GPU / GPGPU, FPGA, DSP, etc.) or to a switch.
- Logically, each link is connected to a Gen-Z protocol engine within the memory component's media controller logic that is used to receive request packets and transmit response packets. Each protocol engine can support multiple links.
- The media controller logic is used to execute a request packet and to generate a response packet. If the media controller supports a Responder ZMMU (Gen-Z Memory Management Unit), it transparently accesses the ZMMU logic / tables to validate request packet access, provide hardware-enforced isolation, decrypt (if applicable) the request packet, locate responder-specific resources, etc. required to execute the request packet.
- Once the request packet is validated, the media controller invokes media-specific logic to access the attached media devices. For example, if the component supports DDR DRAM media devices, then unbeknownst to the memory controller, the media controller performs an Activate to pull the row / page that contains the addressed data into a media-controller-specific resource that it subsequently accesses to read or write the data.
    - Media controllers use the packet's Address field and optional Responder ZMMU (Responder ZMMU is not used with P2P 64) to identify the target data within the component's Data Space.
    - The *Gen-Z Core Specification* describes a set of optimizations and techniques that improve media controller performance (bandwidth and latency) significantly without requiring any changes to the underlying media devices. Developers are strongly encouraged to incorporate these performance optimizations into their designs.
- Once request packet execution is completed, the media controller generates a response packet if needed. Depending upon the number of supported interfaces, VCs (Virtual Channels), etc. the media controller accesses Gen-Z control structures to determine the correct egress interface, VC, etc. to use to generate, encrypt (if applicable), and transmit the response packet.

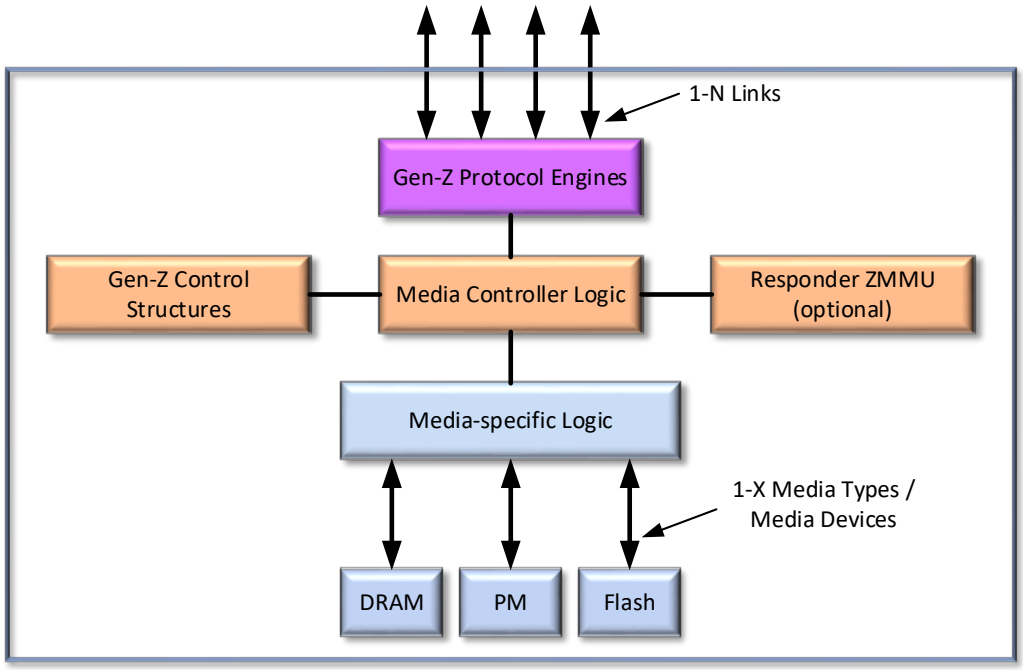

Figure 6: Example Memory Component Functional Blocks

Memory Access

Example Gen-Z Write and Read Operations illustrates the logical steps taken to translate a processor store into a Gen- Write request packet and a processor load into a Gen-Z Read request packet.

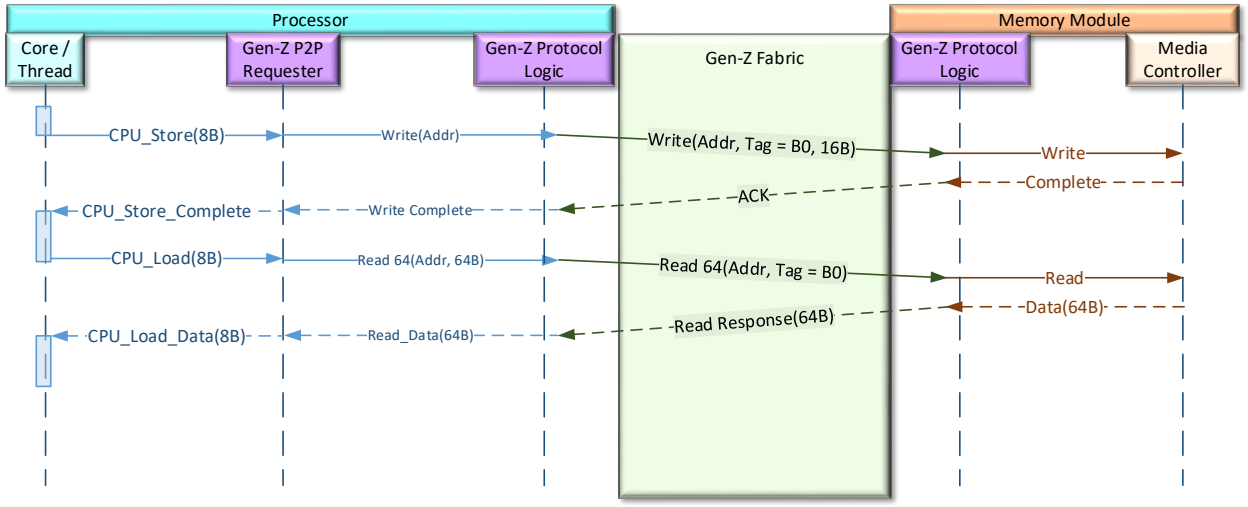

Figure 7: Example Gen-Z Write and Read Operations

- The processor core executes a Store (8B) instruction that is routed to a Gen-Z Requester.
  - The Gen-Z Requester receives the Store (8B) request and forwards the request to the Gen-Z protocol logic.
    - The Gen-Z Requester accesses one or more Gen-Z mapping logic to identify the Responder and to determine the egress interface.
    - The Gen-Z protocol logic transmits a Write request packet.
  - The media controller within the memory module receives and validates the Write request packet. If successfully validated, then it executes the packet using media-specific methods to update the underlying media.
    - If a persistent write, then the media controller generates an acknowledgment packet once request packet execution has reached a processing state that the media update is guaranteed to succeed.
    - If a non-persistent write, then the media controller does not transmit an acknowledgment packet unless an error is detected. Non-persistent writes can be used to access persistent media that is configured to behave as though it were volatile or behaves as persistent but requires the application to periodically issue a Gen-Z Persistent Flush operation to ensure persistency.
    - In a high-performance processor design, once the processor core passes the Store request to the Gen-Z Requester, the processor immediately completes the request without waiting to ascertain if the request was successfully executed (this is not illustrated).
- The processor core executes a Load (8B) instruction that is routed to a Gen-Z Requester.
  - The Gen-Z Requester receives the Load (8B) request and forwards the request to the Gen-Z protocol logic.
    - The Gen-Z Requester logic maps the address to determine the egress interface.

        - The Gen-Z protocol engine logic transmits a Read request packet.
    - The media controller within the memory module receives and validates the Read request packet. If successfully validated, then it executes the packet using media-specific methods to read the underlying media.
        - The media controller transmits a Read Response packet to return the targeted data.
        - The processor's Gen-Z logic receives and validates the response packet, and returns the requested 8B to the processor core.

Memory Module Sharing

A memory module can be shared by multiple Requesters, e.g., multiple servers, servers and GPGPU / FPGA / DSP / ASIC accelerators, multiple I/O or SSD components, etc. Gen-Z supports multiple types of memory module sharing:

- Serial sharing is where only a single Requester has access to the module at a time. Serial sharing is used to logically migrate memory resources (volatile or PM) from a Requester or a platform to another. Hardware-enforced isolation is performed using Gen-Z's component-level access control services.
- Partitioned memory sharing is where multiple Requesters can simultaneously access a module, but only one Requester has access to a given memory range / partition at a time. Partitioned memory sharing is used in composable infrastructures that need to provide dynamic incremental memory expansion to meet workload-specific needs, i.e., burst capacity. For example, a wide SFF-TA-1008 module can support up to 256 GiB DDR 5 DRAM or 4 TiB PM of addressable memory. If using 8 GiB partitions (Gen-Z enables any number and size of partitions to be supported), a DRAM module could support up to 32 memory partitions and a PM module could support up to 512 memory partitions. One or more memory partitions can be dynamically assigned to individual Requesters to "right size" resources to workloads.
- Shared memory is where multiple Requesters can simultaneously read and write the same memory resource as though the Requester applications were running on the same hardware. Multiple operating systems currently support shared memory and ensure consistency, i.e., only one application or thread can modify a memory location, through the use of atomics. These atomics can be mapped to Gen-Z atomic operations to provide seamless fine-grain access control.

If a memory module supports only the P2P 64OpClass, then sharing is limited to direct-attached Requesters (requires memory modules to support 2 or more interfaces), and sharing is controlled through software partitioning and / or use of software or hardware-initiated Gen-Z atomic operations.

If a memory module supports explicit OpClasses, then the associated media controller needs to implement a Responder ZMMU to provide hardware-enforced isolation. Responder ZMMU designs can vary, e.g., *Example Responder ZMMU Implemented as a Page Grid* illustrates a very light-weight, low-latency Responder ZMMU.

- A request packet's address is transformed using an implementation-specific method to identify a Page Grid.
- Each Page Grid is a configurable structure that points to a range of Responder PTEs (Page Table Entry) that in turn point to the configured pages.
- The number of Page Grids, PTEs, pages, etc. are implementation specific. For example:
    - At a minimum, the *Gen-Z Core Specification* requires that ZMMUs support 4 specified page sizes that are commonly supported by host processors. To meet the minimum requirements, a Responder ZMMU would need to support at least 4 Page Grids. Since

many processors support more than these 4 page sizes, a Responder ZMMU could support additional Page Grids, one per supported page size.

- The number of Responder PTEs associated with a Page Grid can be fixed or configurable (strongly recommended). If configurable, then each Page Grid points to a sub-range of Responder PTEs within a shared table of Responder PTEs.

- A PTE may support additional services, e.g., page-level encryption and data authentication. See the *Gen-Z Core Specification* for details.

A Responder ZMMU enables multiple Requesters to simultaneously share a memory module. Hardware-enforced isolation is accomplished using Gen-Z Region Keys (R-Keys) to control access at the page level and at the component level through access permissions configured through the Component PA (Peer Attribute) Structure. Pages can be exclusively "owned" by a single Requester or shared by multiple Requesters.

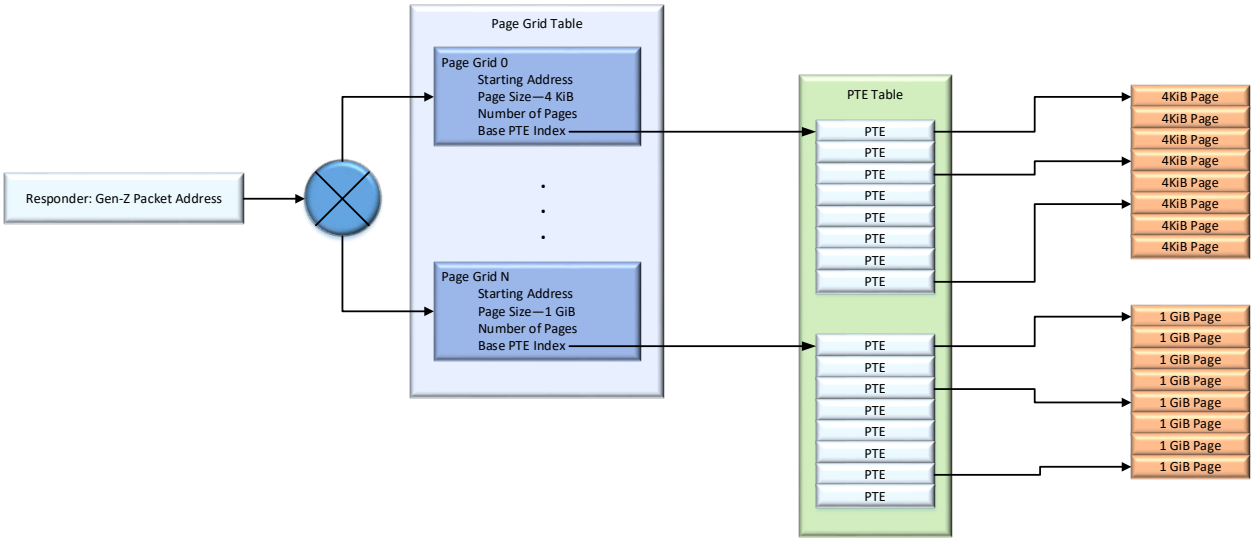

Figure 8: Example Responder ZMMU Implemented as a Page Grid

Security

In addition to multiple types of hardware-enforced isolation, Gen-Z supports or will support the following security capabilities:

- Strong packet authentication and anti-replay attack protection against malicious packet injection. Gen-Z supports multiple cryptographically-secured hash functions as well as multiple anti-replay tag types to provide maximum flexibility and meet global regulatory needs.
- Strong component authentication based on new DMTF (www.dmtf.org) security data objects to enable components, modules, enclosures, and racks to be dynamically authenticated at any time from manufacturing to deployment to decommissioning. See *Gen-Z Component Authentication—Foundation for a Secured Infrastructure* for a high-level overview of the threats and how these new security data objects are used.
- In P2P 64 and P2P Vendor-defined solutions, Gen-Z enables request and response packets to be encrypted using one key for primary media data, one key for secondary media data, and one key for configuration and control.

- In explicit OpClass-based solution, Gen-Z enables request and response packet encryption at the page level. Implementations can support 1000s of security keys and certificates.

Nonce Operation Overview

To prevent a rogue component from being inserted during C-DLP (deep low-power state), each component can be configured with a nonce, i.e., a random 64-bit unsigned integer, and each component interface can be configured with the directly-attached peer component's nonce. Whenever nonce validation is enabled or a link transitions to an operational state, each connected interface performs a Nonce Notification link-level exchange using Link CTL packets (point-to-point link-local packets) to obtain the peer interface's nonce. If the received nonce is not equal to the configured Interface Peer Nonce value, then the interface takes the configured error handling (Interface AE) actions, e.g., triggering interface containment to prevent further packet exchange. *Example Nonce Configuration* illustrates the in-band management configuration and nonce validation.

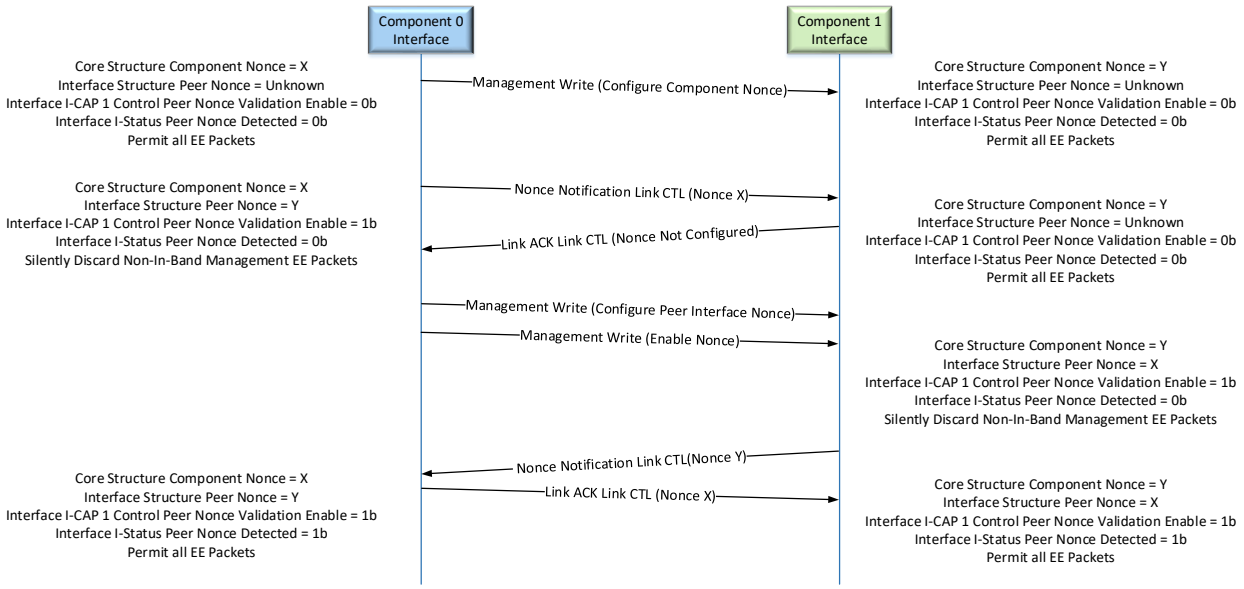

Figure 9: Example Nonce Configuration

Software Model

Conceptual Gen-Z Software Model illustrates how various software modules access Gen-Z bridge logic. Though not shown, some persistent memory software stacks also use file system and / or a middleware module to support persistent object identifiers, invoke persistent flush semantics, etc.

- Process / Threads
  - Uses load-store semantics to access DRAM / PM.
  - Does not require any application modifications unless the memory module surfaces advanced capabilities, e.g., data-centric acceleration.
- Operating System
  - Uses load-store semantics to access DRAM / PM.
  - Does not require any operating system modifications to support the basic memory operations.
  - Operating systems executing within a virtual machine and containers do not require any modifications to support basic memory operations.

- System firmware—generic name for software that provides the following services to higher-level software:
  - Abstracts Gen-Z hardware implementation details from the operating system.
  - Provides low-level Gen-Z configuration, event handling, and monitoring services.
  - Maps the applicable memory component Data Space into the Gen-Z Bridge's Requester ZMMU.
  - Presents Gen-Z memory resources to the operating system using industry standard interfaces such as ACPI. From the operating system's perspective, Gen-Z memory is indistinguishable from DDR memory.
  - System firmware uses load-store semantics to access each component's Control Space using in-band protocol packets or through an out-of-band interconnect, e.g., I^3C.

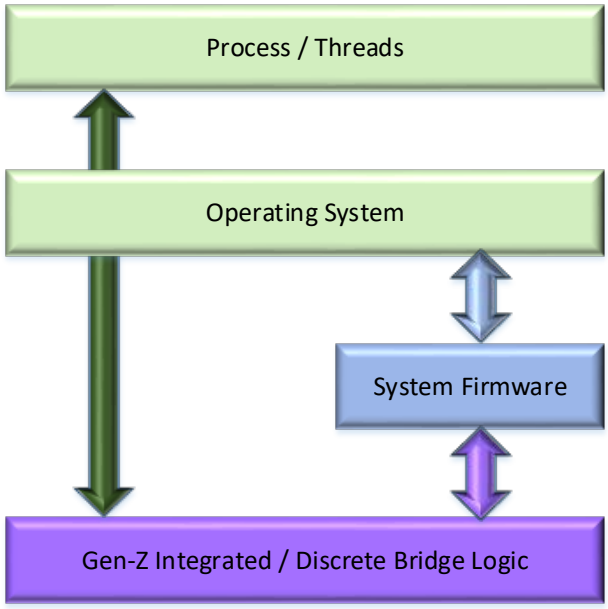

Figure 10: Conceptual Gen-Z Software Model

Example Processor Initialization Sequence illustrates how system firmware discovers and enumerates a Gen-Z topology during power-on initialization.

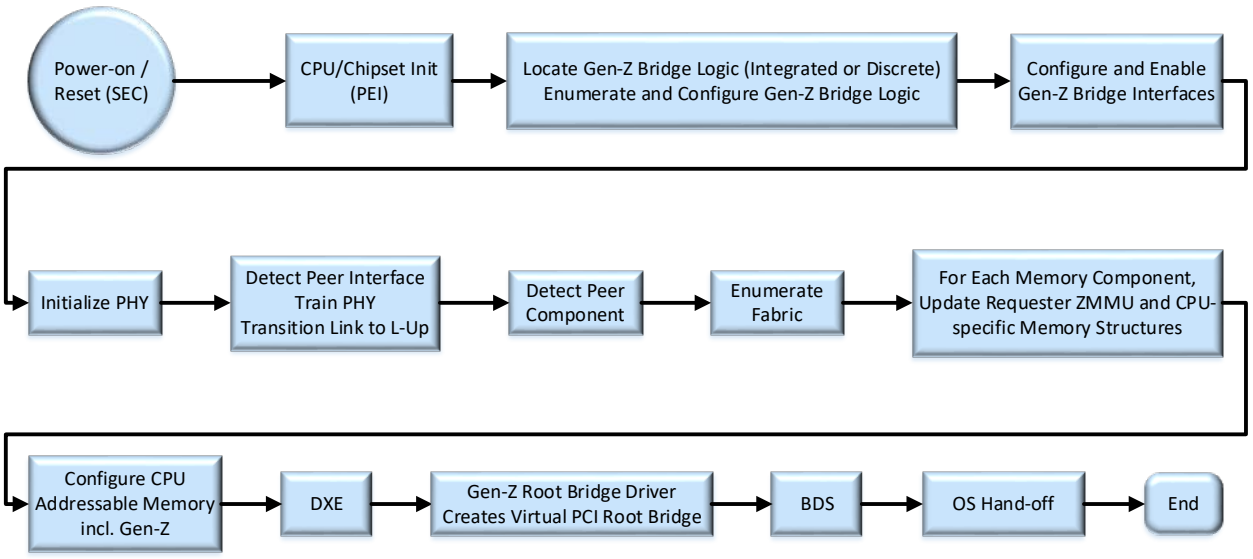

Figure 11: Example Processor Initialization Sequence

Exception Handling

The following high-level descriptions highlight the primary exceptions and how they are handled. In the following, a Requester is any component that transmits request packets (e.g., a processor, GPU / GPGPU, FPGA, DSP, ASIC, etc.), and a Responder is a memory component that executes received request packets and transmits response packets.

- Request packet validation and execution errors
  - When such failures occurs, a Responder generates a response packet which includes the highest-precedence root cause error.
- Media failure—this includes media device failure, excessive uncorrectable media errors, etc.
  - When such failures occur, the media controller should generate a primary or secondary media component log entry (see *Component Media Structure*). If Component Media structure's Management Event Notification is configured to inform management, then do so as configured in the *Component Error and Signal Event Structure*.
  - Media device failures can be handled by provisioning one or more spare media devices that can be transparently substituted for the failed device. Transparent substitution requires the media controller to support data resiliency services either at the memory module level (e.g., flash-based data checkpoints, internal RAID / erasure code, etc.) or through use of a memory resiliency service that operates across multiple memory modules (e.g., a Transparent Router (TR) with multi-module RAID / erasure code).
  - Uncorrectable media errors can be handled by provisioning additional rows that can be substituted for the impacted rows. Transparent substitution requires the media controller to support data resiliency at the memory module level (e.g., robust ECC) or through use of a memory resiliency services that operates across multiple memory modules. If this is not possible, then the media controller needs to return an uncorrectable error detected indicator in response packets when the row is accessed. Operating system / resiliency software handling could be required to rebuild the lost data.
- Media controller failure—this includes power loss, ASIC failure, complete connectivity loss, etc.

- When such failures occur, each Requester such as a processor should inform management as configured in the *Component Error and Signal Event Structure,* and initiate Requester-specific error handling and recovery.
- Unless the memory component participates in a memory resiliency service that operates across multiple memory modules, the corresponding data is no longer available. Failure can be detected by request packets exceeding the maximum retransmission permitted, link loss detection, intermediate component detection and notification using Unsolicited Event Packets, etc.

- Memory controller failure—this includes power loss, ASIC failure, complete connectivity loss, etc. If using P2P 64 / directly attached, then a Responder detects memory controller failure as complete connectivity loss. If not directly attached, then a Responder is not able to detect memory controller failure; it can detect only the lack of new request packets to process.
  - If using P2P 64 / directly attached, then the responder should take the actions configured in the *Component Error and Signal Event Structure,* and initiate Responder-specific error handling and recovery. Responder-specific error handling can include initiating component containment.
- Path failure—this includes partial connectivity loss, e.g., link failure (Requester or Responder), switch failure (if applicable), etc.
  - When such failures occur, the component should generate a component log entry (see *Component Error and Signal Event Structure*).
  - To avoid a single point of failure and stranded resources, Requesters and Responders are strongly encouraged to support multiple component interfaces.
  - When the retransmission timer associated with a request packet expires, the Requester should retransmit the request packet using an alternative path (e.g., egress interface).
  - If a Responder link fails, then the Responder should inform management as configured in the *Component Error and Signal Event Structure,* and silently discard any pending response packets that cannot be transmitted using an alternative interface. If the Responder is communicating using the P2P 64 OpClass, then it should initiate component containment.
- Asynchronous insertion—this includes memory module insertion and dynamic surfacing of memory resources to a Requester.
  - Upon detecting the insertion of a memory module (e.g., due to link transition to L-Up and receipt of Link RFC packets), the detecting component (Requester or switch) should inform management as configured in the *Component Error and Signal Event Structure.*
  - If management surfaces memory resources to a Requester, then system firmware can map these resources into a Requester ZMMU, and invoke ACPI to make these visible to the operating system. Alternatively, if a memory broker service is available and the operating system has been updated to work with a memory broker, then these resources can be mapped into a Requester ZMMU through the Gen-Z bridge driver software, and then the memory broker could inform the operating system.
- Asynchronous removal—this includes memory module removal and dynamic removal of memory resources from a Requester.
  - If the memory resiliency services are unavailable, then prior to performing planned memory module / resource removal from a Requester, management should inform the

operating system / component to enable data to be migrated / copied to alternative memory resources.

- Upon detecting unplanned memory module / resource removal, the detecting component (Requester or switch) should inform management as configured in the *Component Error and Signal Event Structure.* If the Requester is unable to recover from the corresponding data loss, then it should initiate component containment. If the Requester can recover, then all Requester ZMMU PTEs corresponding to the lost resources should be released, and all control structure fields used to access the lost memory component / resources should be reset to the uninitialized state to prevent future access.

DISCLAIMER

This document is provided 'as is' with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Gen-Z Consortium disclaims all liability for infringement of proprietary rights, relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Gen-Z is a trademark or registered trademark of the Gen-Z Consortium.

All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.