

Gen-Z Buffer Operations

April 2019

This presentation covers Gen-Z Buffer operations. Buffer operations are used to move large quantities of data between components.

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.

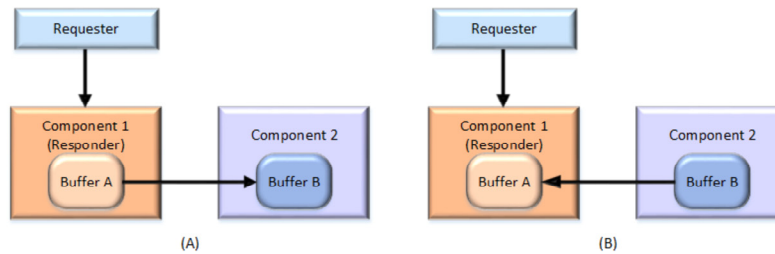
Gen-Z is a trademark or registered trademark of the Gen-Z Consortium.

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

All material is subject to change at any time at the discretion of the Gen-Z Consortium

<http://genzconsortium.org/>

Buffer Operations



- Buffer operations enable large data movement between two buffers—Buffer A and Buffer B
 - Maximum data movement size is 4 GiB
 - Buffers A and B can target any media type
 - Buffers A and B can be storage buffers, messaging buffers, etc.
- Data movement is done as a put, i.e., write, from A to B or get, i.e., read, from B to A
 - Conceptually, a Buffer Put is a much more robust / capable RDMA Write with fewer constraints
 - Conceptually, a Buffer Get is a much more robust / capable RDMA Read with fewer constraints

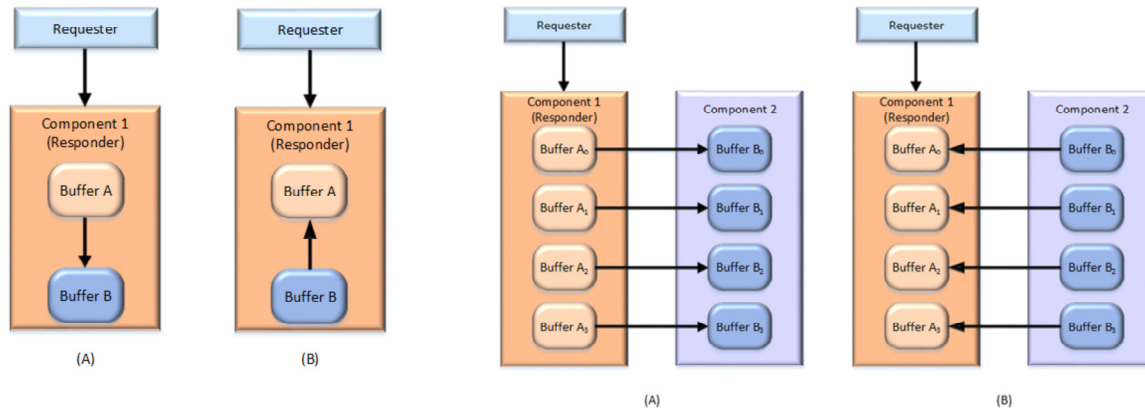
© Copyright 2016 by Gen-Z. All rights reserved.

GEN Z

RDMA Read and RDMA Write operations such as specified by multiple industry bodies are less flexible and capable, e.g.,

- RDMA Read and RDMA Write operations operate across a single connection between two interfaces. In contrast, Gen-Z Buffer operations use datagrams to exchange data enabling data movement across all component interfaces and all paths.
- RDMA Read and RDMA Write operations are between two components. In contrast, Gen-Z Buffer operations can be within a single component, between two components, or used to support third-party data movement.
- Unlike Gen-Z Buffer operations, RDMA Read and RDMA Write operations do not support optional in-line data integrity and cryptographically-secured hash to protect data and detect errors or malicious tampering.
- Unlike Gen-Z Buffer operations, RDMA Read and RDMA Write operations operate only between pre-allocated and mapped buffers. In contrast, Gen-Z Buffer operations support dynamic buffer allocation methods to simplify buffer management and improve performance.
- Etc.

Buffer Operations (continued)



© Copyright 2016 by Gen-Z. All rights reserved.

GENZ

Buffer operations can be used to move data within a component. For example, if a component contains two addressable resources (primary and secondary media), then a buffer operation can be used to put or get data from one media to the other. Further, if the Responder contains an accelerator (e.g., an encryption / compression / graph / KVS / etc. engine), then as the data is moved between the two buffers, the Responder can perform additional accelerator-specific operations in parallel (these accelerations are transparent to the application).

A vector buffer operations enables up to 4 independently-addressed buffers to be moved between components, or within a single component.

Buffer Operation Observations (1)

- Buffer operations can be supported by any component type—SoC, memory, storage, FPGA, GPU, etc.
- Buffer operations can be used between Gen-Z components on any topology and at nearly any scale
 - Gen-Z's inherent scalability eliminates the need for intermediate interconnects / gateways
 - For example, Gen-Z scales to 1000s of storage modules; eliminates the need, cost and complexity of NVMe over Fabrics
- Buffer operations can be used to move data:
 - Between a SoC / Accelerator and a memory or storage module
 - Between two memory modules or two storage modules
 - Between a memory module and a storage module (tiered memory / storage solutions)
 - Between a SoC / Accelerator and a storage module
 - Between two SoCs / Accelerators (two-party or three-party data movement)
- Buffer operations simplify data movement
 - No work queues, completion queues, etc. to manage—single buffer request / response packet exchange
 - Enables a Requester to off-load data movement to a Buffer Responder
 - For example, a SoC could ask a storage component to push or pull data to / from the SoC's memory or to other modules eliminating any SoC performance impact commonly experienced using alternative interconnects

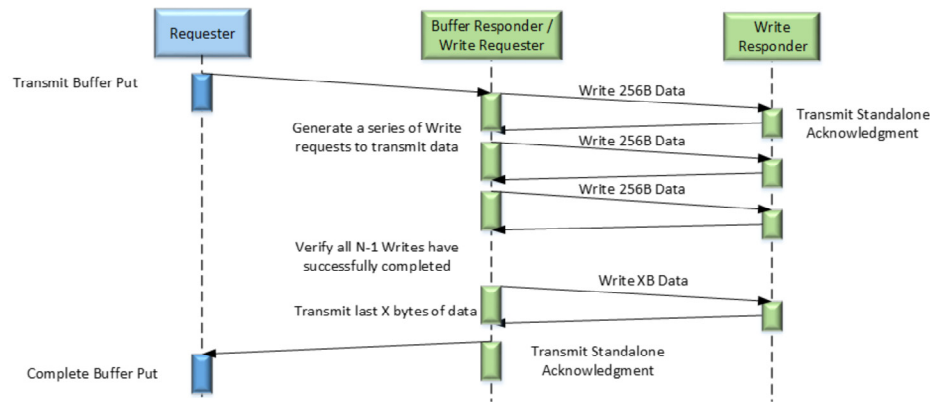
Buffer operations provide numerous application functional and performance benefits. Buffer operations are more than just simple put and get data movement mechanisms. Buffer operations can be used to augment or enhance solution stacks while minimizing data movement, or can be combined with acceleration technology to provide value-add processing transparent to applications and middleware.

Buffer Operation Observations (2)

- Buffer operations can improve performance in multiple ways
 - No work queues / completion queues / etc. to manage—just a simple Buffer request and response
 - Three-party data movement eliminates need for all data to flow through a SoC / Accelerator
 - Data movement offload minimizes application software involvement
 - Signaled buffer operations eliminate multi-thread / multi-node coordination overhead and complexity
 - Dynamic buffer operations offload buffer allocation / release to Responder
 - Data Integrity generation / validation offload reduces buffer protection and validation costs
 - Cryptographically-secured Hash generation / validation offload reduces very strong buffer protection and validation costs
- Buffer operations can work with any memory media type—DRAM, PM, Flash, HDD, etc.
 - Buffer operations are non-deterministic, so intrinsically tolerate variable media latencies
- Underneath, buffer operations are executed using standard Gen-Z Reads and Writes
 - Only Buffer Requester and Buffer Responder need to comprehend buffer operations
 - Buffer B node only needs to comprehend Gen-Z Reads and Writes

Buffer operations provide numerous application functional and performance benefits. Buffer operations are more than just simple put and get data movement mechanisms. Buffer operations can be used to augment or enhance solution stacks while minimizing data movement, or can be combined with acceleration technology to provide value-add processing transparent to applications and middleware.

Buffer Put Translated to a Series of Write Requests



© Copyright 2016 by Gen-Z. All rights reserved.

GEN Z

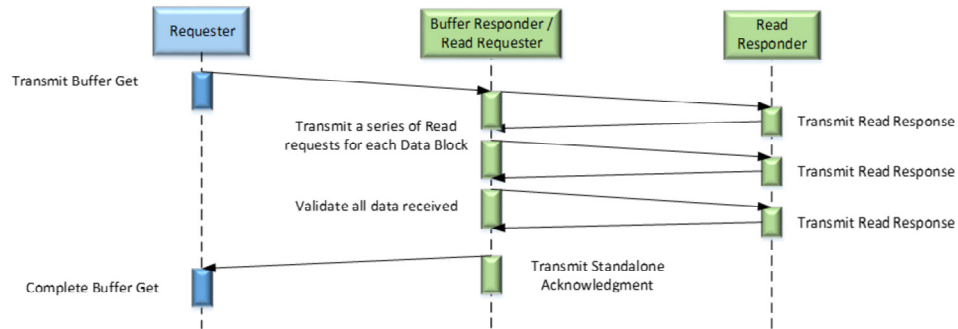
A Buffer Put operation is translated by the Responder (component A) into a series of Write requests to component B. Only the Requester and Component A need to support buffer operations; component B only needs to support normal read and write operations.

This example illustrates how buffer operations can eliminate unnecessary data movement. For example, in some file or storage solutions, the data needs to flow through the host processor and memory (this is often accomplished by having component A perform a series of DMA Writes data to host memory, signal component B when the DMA Writes are completed, and then component B performs a series of DMA Reads to pull the data down.

Though not shown in this example, the Requester could issue a Buffer Put request to instruct component A to write a buffer to the Requester's memory, i.e., the Requester contains buffer B. This provides multiple advantages:

- Eliminates the need to provision and program work queues as well as the associated control plane protocol overheads
- Eliminates per Requester static resource provisioning—a Responder supports N outstanding buffer operations usable by multiple Requesters based on workload needs
- Enables all data movement to be offloaded to the Responder. Significant software stack reductions are possible (perhaps a single processor instruction to perform a put or get operation).

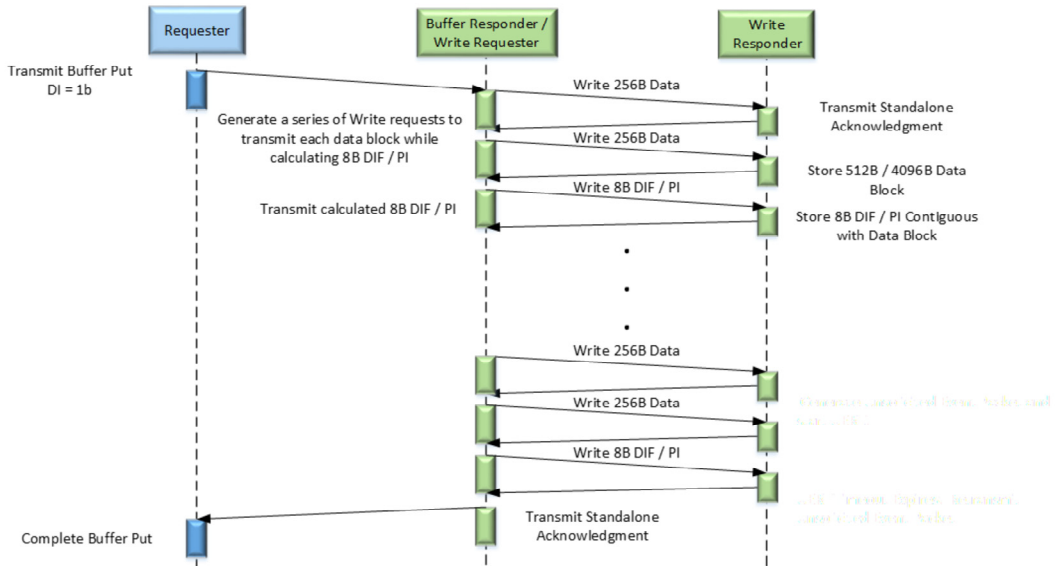
Buffer Get Translated to a Series of Read Requests



If the components support LDM Reads, then translated to a series of LDM Reads and LDM Read Responses

A Buffer Get operation is conceptually the same as a Buffer Put operation, except it uses Read or Large Read operations instead of Write operations. Buffer Get operations can improve solution efficiency by enabling data to be pulled at the rate that component A can consume the data.. This can reduce fabric congestion as well as improve component A's responsiveness since it controls when the data is read.

Buffer Put with In-Line Data Integrity Generation

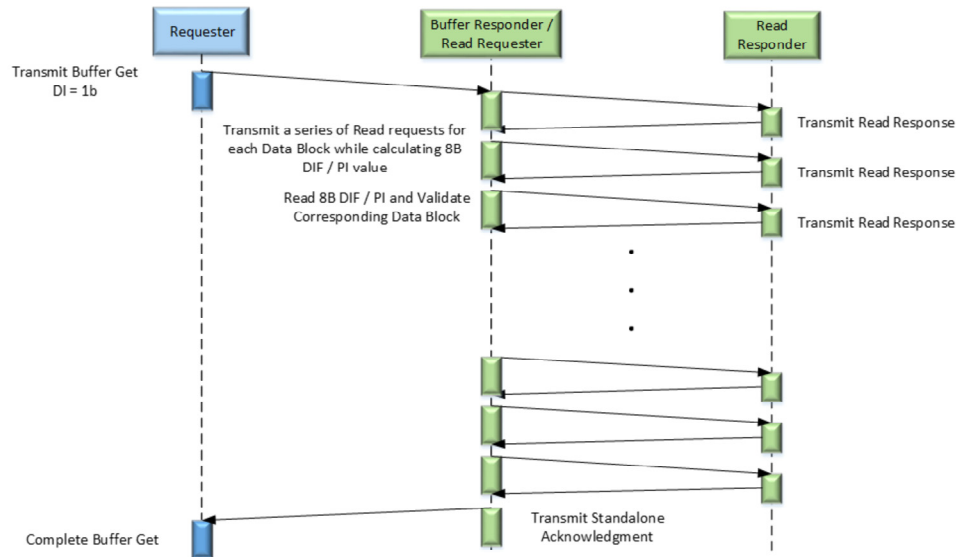


© Copyright 2016 by Gen-Z. All rights reserved.

GENZ

T10 DIF / PI values can be dynamically calculated and inserted into the buffer as it is transferred to ensure end-to-end data integrity and improve solution performance.

Buffer Get with In-line Data Integrity Validation

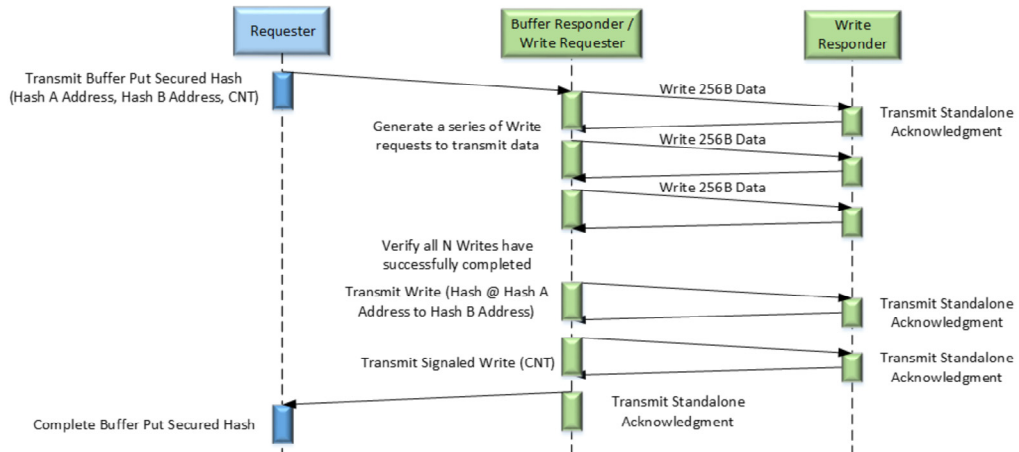


© Copyright 2016 by Gen-Z. All rights reserved.

GENZ

T10 DIF / PI values can be dynamically calculated and validated as the buffer is transferred to ensure end-to-end data integrity and improve solution performance.

Buffer Put Secured Hash

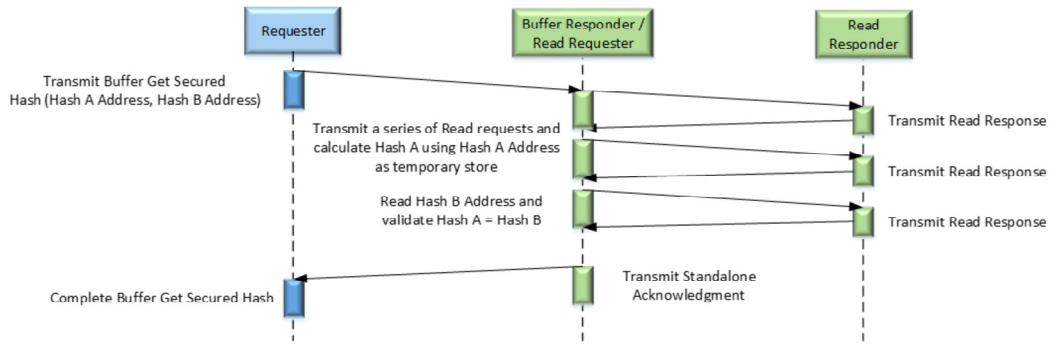


© Copyrights 2016 by Gen-Z. All rights reserved.

GENZ

Cryptographically-secured Hash values can be dynamically calculated and inserted into the buffer as it is transferred to ensure end-to-end data integrity and improve solution performance.

Buffer Get Secured Hash

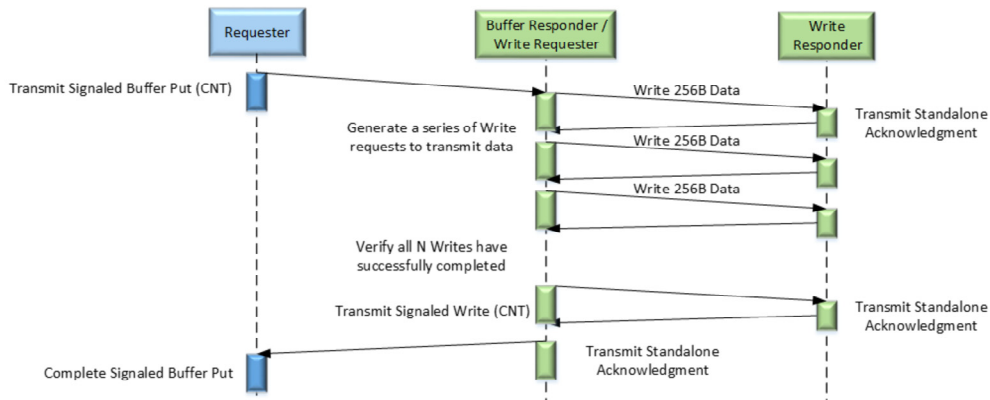


© Copyright 2016 by Gen-Z. All rights reserved.

GENZ

Cryptographically-secured Hash values can be dynamically calculated and validated as the buffer as is transferred to ensure end-to-end data integrity and improve solution performance.

Buffer Signaled Put



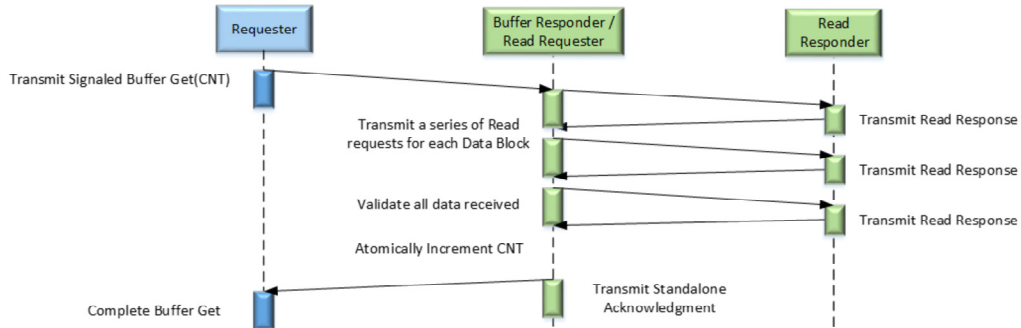
- Buffer Signaled Put performs a Put operation and then performs a Signaled Write
- Signaled Write enables multiple threads / components to periodically poll for buffer completion
 - Signaling eliminates the need to tightly coordinate multiple threads / nodes which can improve performance

© Copyrights 2016 by Gen-Z. All rights reserved.

GENZ

There are multiple types of Buffer operations. This slide illustrates a Buffer Signaled Put operation. A Buffer Signaled Put operation uses a Multi-Op Signaled Write to move the last bit of data. Upon receipt of a Signaled Write, the Responder atomically updates the signaled address. Components that periodically poll the signaled address will detect the data modification, and take action. Using a Signaled Write eliminates the need to explicitly coordinate the put operation completion (the Responder does even need to know which or how many other components or process threads are polling the signaled address). Eliminating coordination cost / complexity can yield significant performance gains and improve solution quality and robustness.

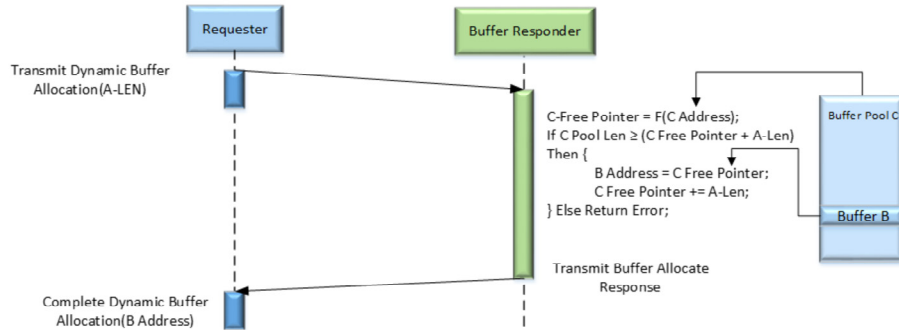
Buffer Signaled Get



- Buffer Signaled Get performs a Get operation and then performs a local atomic increment at the target address
 - Signaling eliminates the need to tightly coordinate multiple threads / nodes which can improve performance

A Buffer Signaled Get is the read analog of a Buffer Signaled Write. Instead of a Signaled Write as the last operation in the put, Component A performs a local atomic increment to signal the other components or process threads.

Dynamic Buffer Allocation / Release



- A Responder can provide dynamic buffer allocation and release services on behalf of Requesters
 - Actual buffer allocation is implementation-specific
 - Dynamic Buffer Allocation returns the address of the allocated buffer
 - Dynamic Buffer Release provides the address for the Responder to recover the buffer
 - Architecture supports combinations of Dynamic Allocation with Put, Get, and Signaled variants

© Copyright 2016 by Gen-Z. All rights reserved.

GEN Z

Gen-Z supports Dynamic Buffer Allocation and Release operations. Allocation can be executed as a standalone operation or in combination with a Put, Get, or Signaled variants. Dynamic Buffer Allocation eliminates the need for a Requester to explicitly manage buffer allocation. For example, consider an application that continually receives new data objects such as pictures or videos. Instead of having to identify a storage or memory resource capable of storing each object, the Requester can simply issue a Dynamic Buffer Put to a Responder which automatically allocates the required buffer and receives the subsequent put data. If the Responder is a Transparent Router (TR), then it could transparently represent any number of memory and storage components, thus supporting very large addressable resources, variety of data preservation technologies (e.g., RAID), aggregate performance (e.g., memory interleave), memory and storage tiers, etc. Further, since many service providers analyze objects to identify faces, places, etc., signaled put and get operations can be used to automatically inform any number of accelerators to automatically analyze the object and generate additional meta data or to improve data analytics.

Thank you

This concludes this presentation. Thank you.