Gen-Z Interrupts

April 2019

This presentation covers Gen-Z interrupt functionality.

Disclaimer

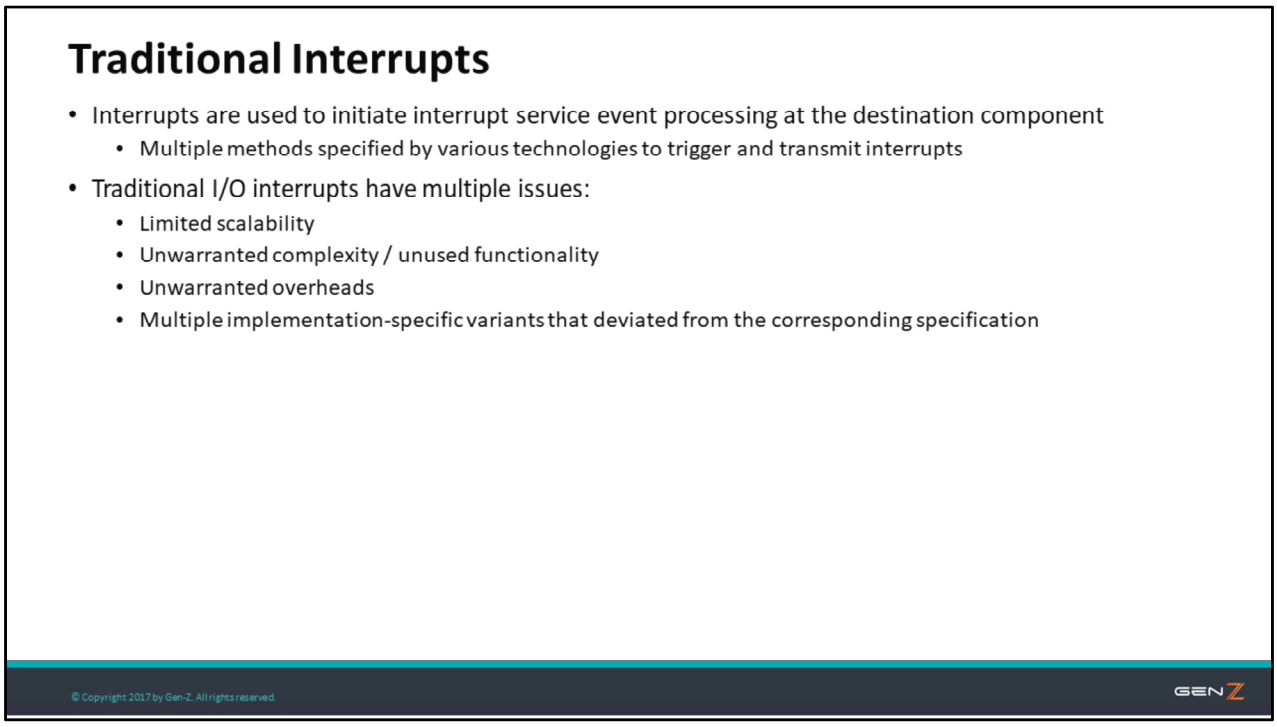
This document is provided 'as is' with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Gen-Z Consortium disclaims all liability for infringement of 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.

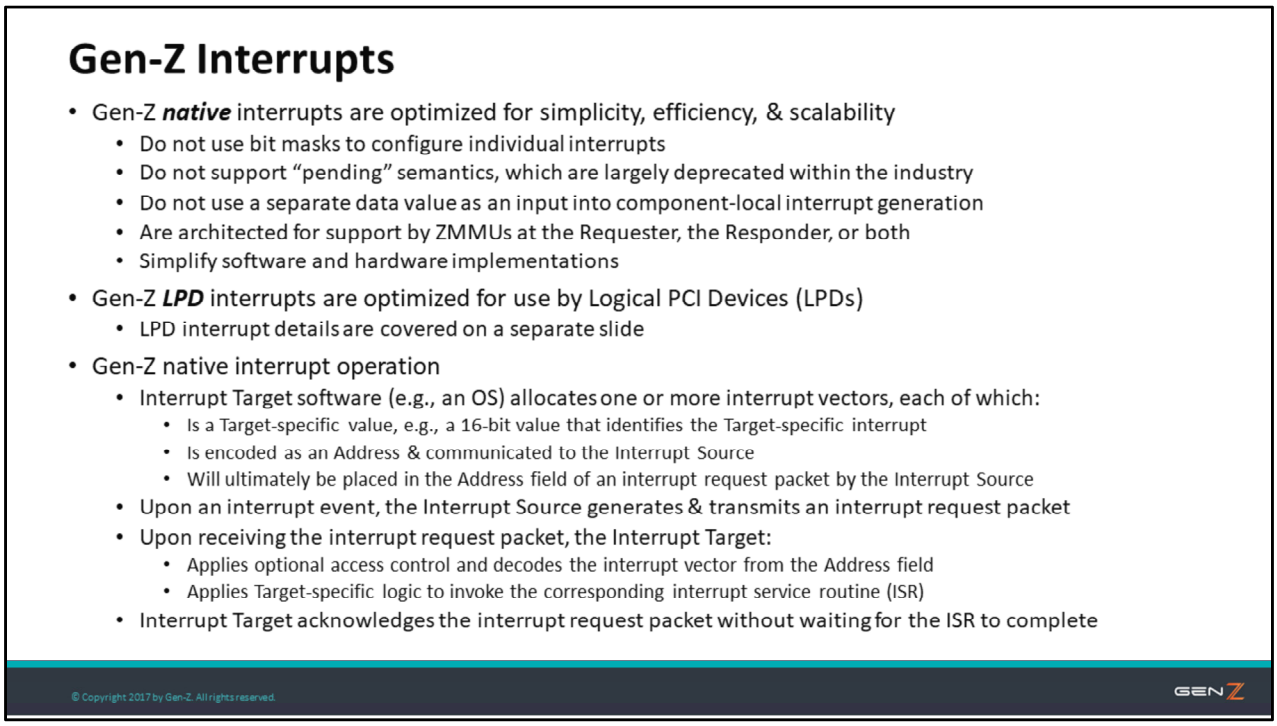
All material is subject to change at any time at the discretion of the Gen-Z Consortium

http://genzconsortium.org/



Interrupts are used to inform an application or OS of a specific event. For example, a NIC receives a packet that needs to be serviced by the OS network stack. Interrupts have evolved over multiple decades, and due to a variety of reasons, there are a number of issues:

- Some technologies support a limited number of interrupts which requires applications, device drivers, etc. to share interrupts (increased complexity and overheads)
- Due to the way traditional interrupts have evolved, legacy functionality and legacy software have created unwarranted complexity and a hodgepodge of functionality is no longer needed.
- Interrupt processing is complex and difficult to control. In some cases, interrupt storms occur which can make a system unresponsive or cause significant application jitter.
- Due to interrupt specification complexity and functionality with questionable value, vendors do not consistently implement interrupts, and in some cases, components are not even compliant with industry specifications. This leads to interoperability problems, and in turn, to more complex OS / middleware software that tries to mitigate or hide implementation differences from applications.

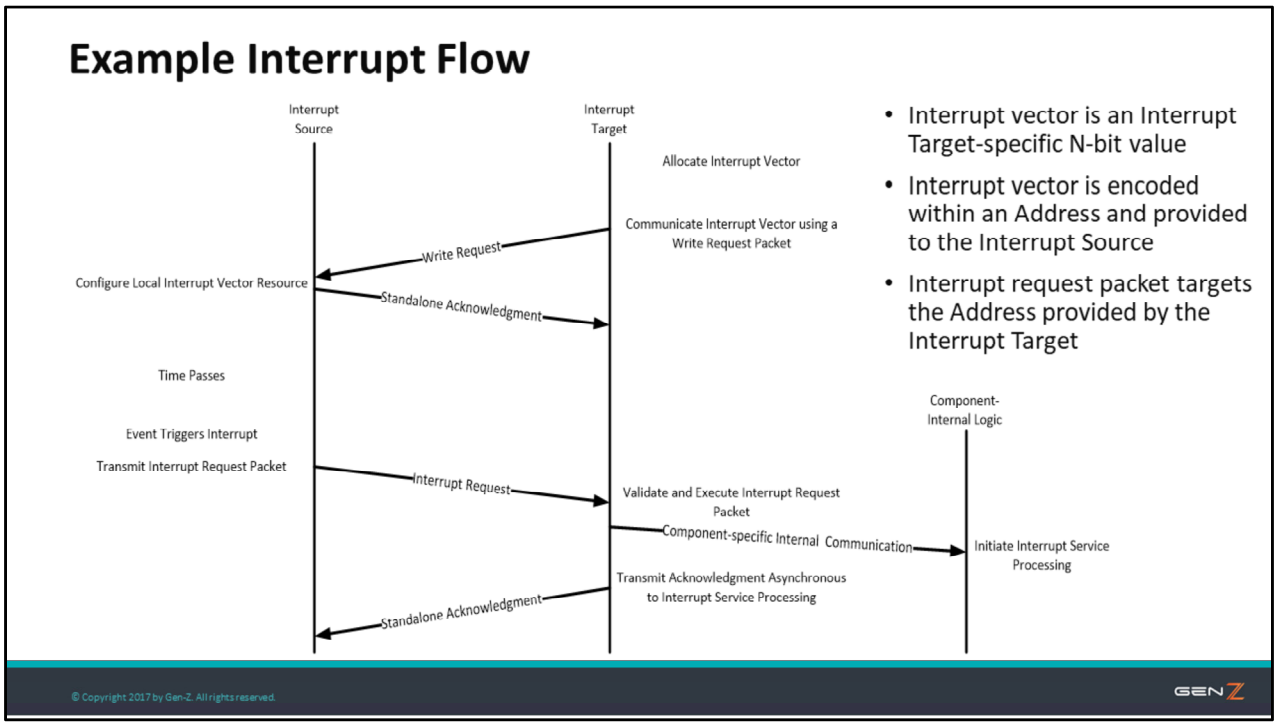


Gen-Z supports two types of interrupts: native and LPD (Logical PCI Device) interrupts.

Gen-Z native interrupts are optimized for simplicity, efficiency, and scalability. Legacy functionality and complexity have been stripped away to leave a lean protocol that can be easily and consistently implemented by hardware.

- By not supporting bit masks, interrupts can be scaled and do not need to be shared.
- By not supporting pending (aka polling) semantics, hardware and software complexity is eliminated
- By not supporting data values, hardware and software complexity is eliminated and scalability is improved
- Gen-Z native interrupts can be tied into ZMMU (Gen-Z memory management unit) at the Requester, the Responder, or both. This simplifies interrupt management, reduces software complexity, and improves performance.
- Gen-Z native interrupts have been reduced to the bare minimum in order to simplify software and hardware, and enable consistent, interoperable implementations.

Gen-Z LPD interrupts use essentially the same packet protocol with some additional fields required to enable a Gen-Z I/O component to interrupt with an existing host's PCI-based infrastructure.

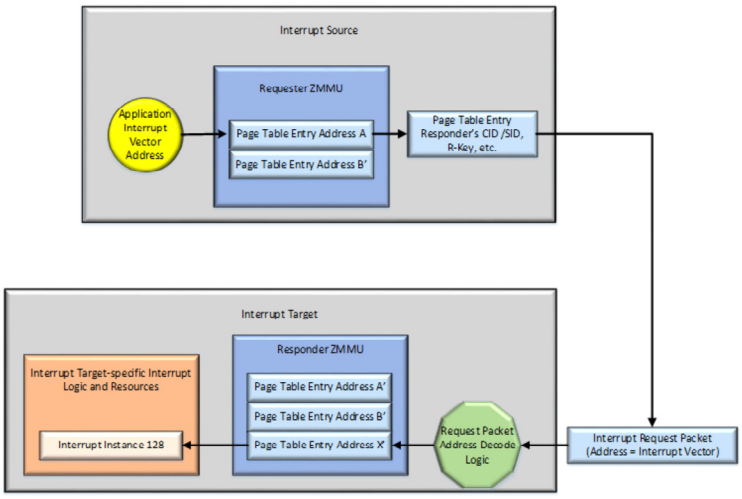


This slide illustrates the sequence of protocol packets used to configure an interrupt and to transmit an interrupt request packet.

1. A device driver or OS on the interrupt target allocates an interrupt vector. This is a target-specific value. For example, a processor uses an implementation-specific mechanism to support interrupts. An OS invokes processor-specific software to create an interrupt vector that is easily supported by the processor-specific interrupt logic. Gen-Z native interrupts encode the interrupt vector within an Address that is subsequently transmitted in an interrupt request packet.
2. The interrupt target communicates the interrupt vector to the interrupt source, e.g., software on the target writes the interrupt vector to an interrupt source-specific location using a Gen-Z write request packet.
3. When an interrupt event is triggered at the interrupt source, the component generates an interrupt request packet using the encoded interrupt vector within the Address field.
4. Upon receipt of the interrupt request packet, the interrupt target validates and executes the packet, and transmits an acknowledgement. Asynchronous to the acknowledgment, the interrupt target initiates the logic to invoke the corresponding interrupt service routine.

Native Interrupt Generation & Reception Using ZMMUs

- Native Gen-Z interrupts may be handled by a ZMMU at the Interrupt Source, the Interrupt Target, or both
- Interrupt generation by Requester ZMMU is focused more on messaging solutions
  - I/O components can use component-specific means to generate interrupt request packets
- Interrupt Target allocates Responder PTE entries based on Address with encoded interrupt vector
- Interrupt Source causes an interrupt request packet to be generated through a local CPU store operation
  - This enables interrupts to be generated easily & securely from application space
  - This enables interrupts to scale out



Interrupts are not requested only by I/O components. Interrupts play an integral role in messaging applications. Since Gen-Z does not require a NIC to exchange messages, applications can use Gen-Z native interrupts to signal a destination component. With a focus on simplicity, Gen-Z uses the Requester ZMMU to trigger an interrupt request packet. The Requester PTE entry contains the Responder's Address, which is encoded with the Interrupt Target's interrupt vector. To transmit an interrupt request, the application simply issues a store (no need to trap to the OS kernel) to the corresponding Responder page, and the Requester generates an interrupt request packet with the encoded Address field. Using this approach, the interrupt can be easily created with minimal latency and overhead (i.e., delivers higher performance). Further, the interrupt request packet can contain an R-Key to provide an additional level of access control to prevent rogue interrupt requests from being executed.

LPD Interrupts for Logical PCI Devices

- LPD interrupts are configured by the host OS/driver using an LPD function's MSI or MSI-X Capability structure
  - The host OS need not be Gen-Z aware
- LPD interrupts carry more info than native Gen-Z interrupts
  - Like native Gen-Z interrupts, carry a 64-bit Address field
  - In addition, carry 32-bit Data & 16-bit LPD BDF fields
- The 64-bit Address & 32-bit Data fields specify the interrupt vector
- The 64-bit Address, 32-bit Data, & 16-bit LPD BDF fields may be used for access control by the host platform, using platform-specific mechanisms
- LPD interrupts are intended to be generated only by LPDs
  - Only architected packet format is Core 64 OpClass
  - There's no architected PTE format for Requester ZMMUs to use for LPD interrupt generation

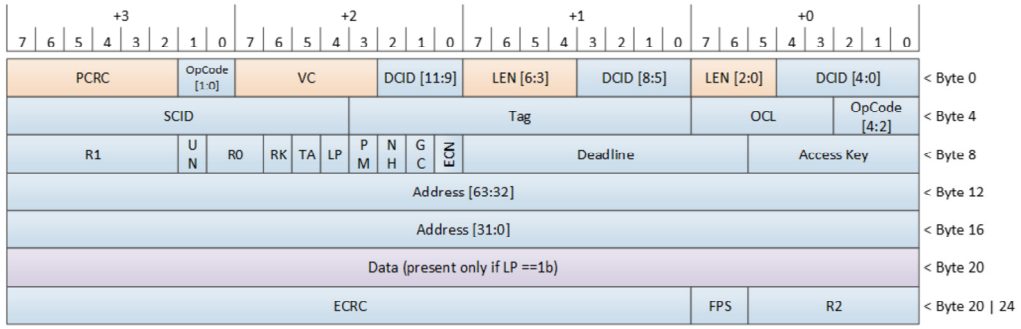
GEN Z

LPD interrupt requests use the same interrupt request packet as a native Gen-Z interrupt, but include the LPD field and the requisite data field required by PCI / PCIe.

LPD interrupts are managed through the existing PCI / PCIe MSI or MSI-X capabilities. The LPD logic translates the MSI / MSI-X parameters into a Gen-Z interrupt with the LPD and data fields present. This enables software compatibility. It also enables existing hardware, e.g., an IOMMU, that uses a combination of the Address, Data, and PCI / PCIe BDF (bus number, device number, function number) to provide additional access control or to translate the address to a component-local value.

As with PCI / PCIe, LPD interrupts are generated by the I/O device. Hence, there is no architected mechanism to trigger interrupt generation; implementations use the existing mechanisms used by PCI / PCIe device drivers and hardware.

Core 64 Interrupt Packet Format

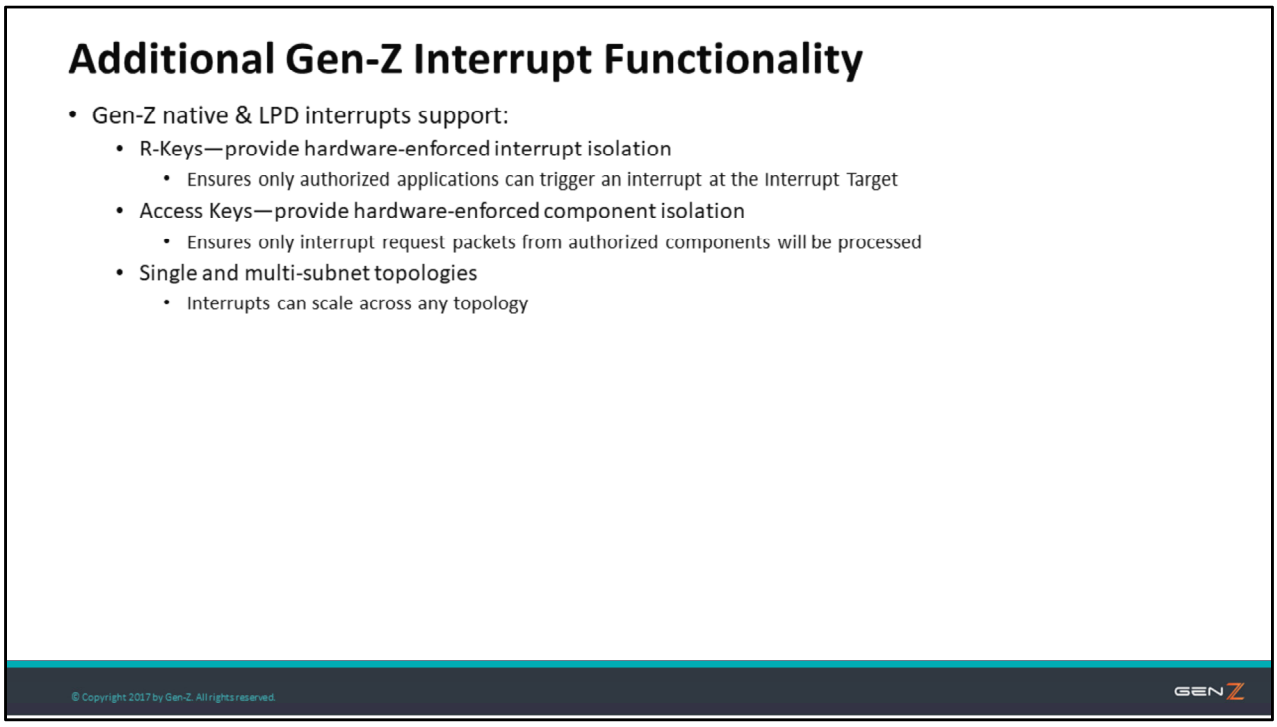


- TA indicates if the address has been translated or not
- U indicates if an acknowledgment is required
- LP indicates if the LPD field is present (not shown) and the associated data field is present

GEN Z

The Core 64 Interrupt request packet format is also used for the Control OpClass interrupts. Specific fields of interest:

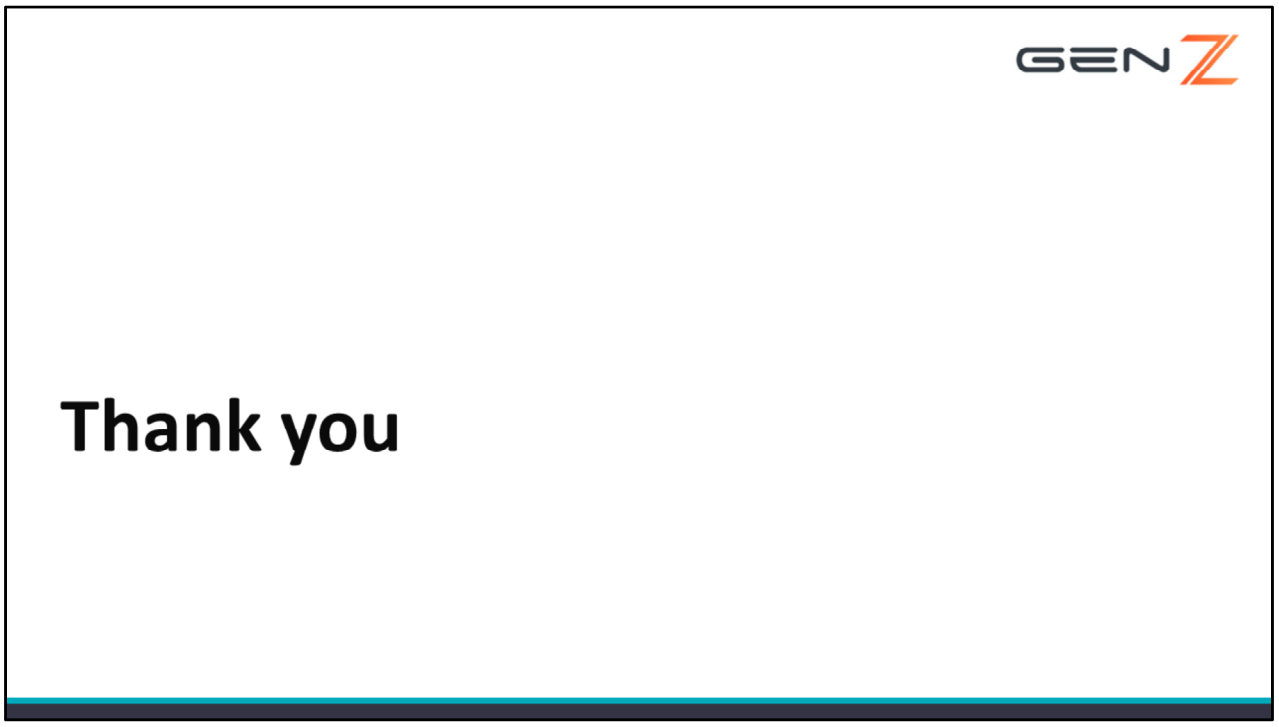
- LP indicates if the LPD field is present
- TA indicates if the address has been translated using Address Translation Services
- RK indicates if the R-Key field is present (used to provide additional access control and isolation)
- U indicates if the interrupt is to be acknowledged or not. For example, if the component is configured to support PCIe Compatible Ordering (PCO), then an acknowledgment is unnecessary since PCO requires the fabric to provide reliable delivery. As a reminder, though PCO provides software compatibility, it prevents I/O components from taking full advantage of the advanced capabilities that Gen-Z provides, e.g., multipath, resiliency, etc.



Gen-Z interrupts provide additional hardware-enforced isolation not found in some alternatives.

- R-Keys ensure only authorized applications can trigger an interrupt at the Interrupt Target. This prevents a rogue or erroneous component from triggering interrupts when it should not.
- Access Keys ensure only authorized components can communicate with the Interrupt Target. This provides a first-line of defense against rogue or erroneous components.

Gen-Z interrupts can scale to any topology. Further, because Gen-Z interrupts are not limited by the resource constraints like some alternatives, Gen-Z interrupts provide greater scaling and simpler configuration and management.



This concludes this presentation. Thank you.