TELEDYNE LECROY
Everywhereyoulook™

Protocol Solutions Group

Summit Exerciser™

Scripting Language

Reference Manual

PCIe Protocol Analysis Version 13.20

Generated: 10/10/2025 1:23 PM

Document Disclaimer

The information contained in this document has been carefully checked and is believed to be reliable. However, no responsibility can be assumed for inaccuracies that may not have been detected.

Teledyne LeCroy reserves the right to revise the information presented in this document without notice or penalty.

Trademarks and Servicemarks

Teledyne LeCroy, CATC Trace, PETracer, PETracer Summit, PCIe Protocol Suite, PCIe Protocol Analysis, Summit T3-16, Summit T3-8, Summit T28, Summit T24, Summit Z3-16, Summit Z416, Summit Z516, Summit Z58,Summit M616, and PETracer Automation are trademarks of Teledyne LeCroy.

Microsoft and *Windows* are registered trademarks of Microsoft Inc.

All other trademarks are property of their respective companies.

Copyright

Contents

1 Introduction

This manual describes the scripting language used to create traffic generation files for the Summit M616, Summit Z516™, Summit Z58™, Summit Z3-16™ *and* Summit Z416™ Exerciser. All references to Z3-16 Exerciser also apply to the Z416 Exerciser.

In general, the scripting language supports all Summit Exercisers:

- Summit M616
- Summit Z516
- Summit Z58
- Summit Z416
- Summit Z3-16

With the following exceptions:

- RawLtssm is currently only supported for Z416 (but not for Z3 and not yet for Z5s)
- Speed is up to 8GT/s for Z3, up to 16GT/s for Z4 and up to 32.0GT/s for Z5s
- Lane Margining commands from the script are only supported for Z4 and Z5s
- All CXL scripting elements apply only to the M616 and Z516 Exercisers but not the Z58

Support Resources

As new functionalities are added, not all of them are supported by older versions of the PCIe Protocol Suite software. For newer releases of the analyzer's software, please refer to the Teledyne LeCroy web site:
teledynelecroy.com/

Syntax

Summit Exerciser™ Script files consist of the statements with the following format:

```
COMMAND = MODIFIER {
      PARAM1 = VALUE1
      ...
      PARAMn = VALUEn
}
```

See the list of all commands with all applicable modifiers on page 3. For some commands the list of the parameters is optional.

All literals are not case sensitive.

All default values are zeros unless otherwise noted.

Integer literals represent numeric values with no fractions or decimal points.

Hexadecimal, decimal, and binary notations are supported:

- Hexadecimal numbers must be preceded by 0x: 0x2A, 0x54, 0xFFFFFF01
- Decimal numbers are written as usual: 24, 1256, 2
- Binary numbers are denoted with 0b: 0b01101100, 0b01, 0b100000

It is possible to use expressions, for example, (i - 239). Or (BASE_ADDRESS + 0x1000),

where BASE_ADDRESS is defined using config=definition construct. See Repeat Command 2.9 page 124 for more examples.

Note: Round brackets should be present only when arithmetic operations are used. If a value is placed in round brackets without any arithmetic operations, it will be reset to zero.

String literals are surrounded by double quotes.

Array data types are represented by integer or string literals surrounded by “(“ and “)” characters, and separated by a comma “,”. For example, (2,23,4).

Single-line comments are supported and should be preceded by a semicolon “;”.

Multi-line comments are also supported. Multi-line comments begin with a “/*” combination, and end with the reverse “*/” combination.

2 Command List

COMMAND	MODIFIERS	Comment
Packet	TLP, DLLP, CXL_Cache, CXL_Mem, SMBus, OrderedSet, Raw, <TemplateName>	Sends a packet.
Idle	<# of ns>	Sends idle symbols (D0.0). **OBSOLETE**. Please use the Wait command below.
Link	L0, L1, L0s, Loopback, Disabled, HotReset, Recovery, Detect, PERST_Assert, PERST_Deassert, Loopback_WComplRx, ComplianceReceive, ClearLoopback, 2_5, 5_0, 8_0, x1, x2, x4, x8, x16	Sets a link condition.
Config	General, Link, FCTx, FCRx, TLP, AckNak, Transactions, Definitions, ATS, SendInterruput, NVMe, NVMeDrive, ErrorInjection, SMBus, RawLtssm, CXL_Link, CXL_ARB_MUX, CXL_VLSM, CXL_ErrorInjectionW	Configures the Summit Exerciser™
Wait	TLP, DLLP, CXL_Cache, CXL_Mem, Error, LinkCondition, Payload, User, SMBus, RawLtssmDone, Time(<# of ns>)	Waits for the condition specified. To wait (delay) for a specific time, specify number of nanoseconds. Precision is in the microseconds range.
Include	<Include file path>	Includes a Summit Exerciser script file.
Branch	TLP, DLLP, CXL_Cache, CXL_Mem, Error, Link, Payload, User	Enables/disables an interrupt for the specified condition.
Proc	Begin, End	Declares the procedure to be used in a branch statement.
Loop	Begin, End	Creates a Summit Exerciser loop.
Repeat	Begin, End	Repeats traffic some number of times.
Template	TLP, DLLP, <TemplateName>	Creates a template for a packet that can be used in the Packet command.
AddressSpace	Read, Write	Reads/Writes address space.
Structure	AHCI, NVMe, PQI_SOP	Used for Host Emulation of a storage protocol.

COMMAND	MODIFIERS	Comment
FastTransmit	Setup, Start, Pause, Continue, Stop	Defines and controls high-performance generation.
Send	MRd32, MWr32 MRd64, MWr64	Defines a sequence of packets to send once, a number of times, or repeatedly. Available only in a **Setup** command section of a **FastTransmit** block.
PCIeFlitMode	True, False	PCIeFlitMode is used to define a global setting that enables exerciser scripting for Flit mode of link operation introduced by Gen6 revision of the PCIe specification.
CXL256BFlitMode	None, CXL_3_0	CXL256BFlitMode is used to define a global setting that enables exerciser scripting for Flit mode of link operation first introduced in CXL 3.0 specification.

2.1 Packet Command

This command initiates transmission of a specified packet on the bus.

2.1.1 Packet = TLP

This command initiates transmission of TLP packet on the bus. The parameters of the **Packet = TLP** command cover all the fields in the TLP header: TLP Payload, PSN (Packet Sequence Number), ECRC, and LCRC. This command should also be used to send CXL.io packets (applies to Z516 and M616 Exercisers in CXL Mode only).

Parameter	Values	Default	Comment
PTH *only in PBR mode	Yes, No	Yes	PTH: PBR TLP Header
PTH_RSVD *only in PBR mode	0 - 63	0	Reserved field
PTH_SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
PTH_DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID
PSN *only in NFM	0 – 4095, Incr	0	When **Incr** is specified, the PSN for the current TLP is assigned as the PSN of the previously sent TLP incremented by 1. When the PSN is generated automatically (see the **AutoSeqNumber** parameter, Page 97), this parameter has no effect. **Note**: When automatic PSN is turned off PSN=Incr will work properly only for TLP transmissions with Count=1. Count instructs the exerciser to repeat the same packet and when automatic PSN generation is turned off, all repeated packets will have the same PSN.

Parameter	Values	Default	Comment
TLPType *only in NFM	MRd32 MRdLk32 MWr32 MRd64 MRdLk64 MWr64 IoRd IoWr CfgRd0 CfgWr0 CfgRd1 CfgWr1 Msg MsgD Cpl CplLk CplD CplDLk CAS32 CAS64 Swap32 Swap64 FetchAdd32 FetchAdd64 DMWr32 DMWr64	0	Sets the **Fmt** (bits 6:5 of byte 0 in the TLP header) and **Type** (bits 4:0 of byte 0 in the TLP header) fields in the TLP header. Also, this field can be specified as a direct numeric value that specifies bits 6:0 of byte 0 in the TLP header.
TLPType * only in FM	MRdLk32 IORd MRd32 CfgRd0 CfgRd1 Cpl CplLk UIOWrCpl UIORdCpl MRd64 MRdLk64 UIOMRd MsgRtoRC MsgRbyAddr MsgRbyID MsgBfromRC MsgLocal MsgGRtoRC MWr32 IOWr CfgWr0 CfgWr1 UIORdCplD CplD CplDLk FetchAdd32 Swap32		Sets Type field [7:0].

Parameter	Values	Default	Comment
	CAS32 DMWr32 MWr64 UIOMWr FetchAdd64 Swap64 CAS64 MsgDRtoRC MsgDRbyAddr MsgDRbyID MsgDBfromRC MsgDLocal MsgDGRtoRC DMWr64		
OHC *only in FM	0-31	0	The OHC[4:0] field indicates the presence of Orthogonal Header Content.
TS *only in FM	0-7	0	The TS[2:0] field indicates Trailer Size.
TC	0 – 7	0	**Traffic Class**: bits 6:4 of byte 1 in the TLP header
LN *only in NFM	0 – 1	0	**Lightweight Notification Protocol:** Enable = 1 Disable = 0
TD *only in NFM	0 – 1	0	Bit 7 of byte 2 in the TLP header: 1 indicates presence of TLP digest in the form of a single DW at the end of the TLP.
EP	0 – 1	0	Bit 6 of byte 2 in the TLP header: 1 indicates the TLP is poisoned.
Snoop	0 – 1	0	Bit 4 of byte 2 in the TLP header: 0 indicates that hardware enforced cache coherency is expected. 1 indicates that hardware enforced cache coherency is not expected.
Ordering	0 – 1	0	Bit 5 of byte 2 of TLP header: 0 indicates PCI Strongly Ordered Model. 1 indicates PCI-X Relaxed Ordering Model.
AT	Untranslated Translation_Req Translated or 0 – 3	0 (Untranslated)	Address Type: Bits 2-3 of Byte 2 in the TLP header. Note: in Device Emulation the behavior of this field and the packet in whole is regulated by the Address Translation Services implementation (refer to 2.4.11 Config = ATS). If there is a need to send the TLP with non-default value in this field unmodified please use the DisableATS parameter described in 2.4.11 set to Yes
Length	0 – 1023	1	Length of data payload in DWORDs.

Parameter	Values	Default	Comment
			If not specified, this field is 1 for all read requests and calculated according to the actual payload for write requests. For a length of 1024, set Length to 0 (because 0 means 1024).
Tag	0 – 1023 LAST_CFG_TAG LAST_IO_TAG LAST_MEM_TAG Incr5bit Incr8bit Incr10bit	0	Byte 6 of the TLP Header for Memory, IO, and Configuration TLP packets. Byte 10 for Completion TLP packets. Additional 2 bits for 10-bit tags as defined by Gen 4 specification. When Tags are generated automatically (see the TagGeneration parameter, Page 64, this parameter has no effect for Memory, IO, and Configuration TLP packets. For Manual Tag policy the script can set the Tag values as the script writer desires or use one of the Incr... values, then the Software will calculate and wrap around each next tag value as appropriate for the selected Incr... policy, 5, 8 or 10 bits. The LAST_CFG_TAG, LAST_IO_TAG, and LAST_MEM_TAG modifiers are applicable to Completion TLPs. When a LAST_CFG_TAG, LAST_IO_TAG, or LAST_MEM_TAG modifier is used for a completion TLP, the Tag field value is set to the value of the Tag in the latest received Configuration, IO, or Memory request, as seen by the Summit Exerciser.
RequesterID	(XX:XX:X) or direct value	0	Bytes 4-5 of the TLP Header for Memory, IO, and Configuration TLP packets. Bytes 8-9 for Completion TLP packets. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
ECRC	0x00000000 – 0xFFFFFFFF	Calculated automatically	When not specified, the PCIe Protocol Suite™ software automatically calculates the ECRC. (TD field has to be specified.) When ECRC is generated automatically by the Summit

Parameter	Values	Default	Comment
			Exerciser™ hardware (see the **AutoECRC** parameter, Page 58), this parameter has no effect.
LCRC	0x00000000 – 0xFFFFFFFF	Calculated automatically	When not specified, the PCIe Protocol Suite software automatically calculates the LCRC. When LCRC is generated automatically by the Summit Exerciser™ hardware (see the **AutoLCRC** parameter, Page 97), this parameter has no effect.
Payload	(XXXX,XXXX,…) Incr Random Zeros Ones		Specified as the array of DWORDs in hexadecimal format (Big Endian). The **Payload** parameter applies only to TLP packets with data. **Incr**: Specifies a payload as the sequence (0, 1, …'**Length**'). **Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of all ones. Note: When **Incr, Random, Zeros,** and **Ones** are used, the **Length** parameter must be specified before the payload. **Payload** can be specified for Memory, IO, Configuration writes, and Completion with Data TLP packets.
Field[<start>:<end>] Field[<pos>]			The arbitrary TLP Header field could be specified by using **Field** parameter. **Start, end**, and **pos** are bit positions from the beginning of TLP Header. Position 0 corresponds to the Most Significant Bit of the first byte of TLP Header. Position 95 for 3 DWORD header (and position 127 for 4 DWORD header) correspond to the Least Significant Bit of the last byte of TLP Header. Fields are limited by 32 bit values. Use **Field[<start>:<end>]** syntax to specify multi bit field. Use **Field[<pos>]** to specify single bit field. ***see Note below**
Count	1 – 65535	1	Repeats this packet the number of times specified.

Parameter	Values	Default	Comment
AutoIncrementAddress	Yes No	No	This parameter applies to Memory Write and Memory Read Request TLPs. If the parameter is set to the "Yes" modifier, and the "Count" modifier is set to a value greater than 1, the Exerciser will perform a "burst" of Memory Writes or Reads, with the Address value for each subsequent address automatically incremented according to the Length value for the write or read. The manual or automatic Tag policy is applied to the transmitted TLPs as specified.
StoreData	(FROM_MEMxx_x, offset)	N/A	This parameter is supported only for the Summit Z3-16 and Z416 Exercisers. It can be used for any Configuration or Memory Read TLP. When used it instructs the Exerciser to collect all the data returned as a response to this Read TLP and copy it to the specified Address Space location. Values have the same notation as in Field Substitution (see 2.1.1.8). NOTE: the described behavior will work properly only for Manual Tag generation policy (”Disable automatic tag generation” is chosen in the Integrity tab of the Generation Options or “TagGeneration = Manual” is set in “Config=TLP” in the script prior to using the StoreData parameter). See **Note 2** and **Note 3** at end of table.
RawTlpPrefix	0x00000000 – 0xFFFFFFFF	0	Raw value of the TLP Prefix to be sent with this TLP. Any prefix can be generated using this. Examples: RawTlpPrefix = 0x9E000000 ; Vendor End-2-End TLP Prefix RawTlpPrefix = 0x8E000000 ; Vendor Local TLP Prefix RawTlpPrefix = 0x91D00020 ; PASID prefix with one reserved bit set
NullifyTLP	Yes No	No	Instructs Z3/Z4 Exerciser to nullify the TLP. Z3/Z4 is automatically going to invert the CRC, terminate the TLP with EDB symbol, and treat the TLP as Malformed (see below).

Parameter	Values	Default	Comment
MalformedTLP	Yes No	No	Instructs Z3/Z4 Exerciser to treat this TLP as malformed (for example, when a TLP with intentionally bad CRC is generated). When such TLP is sent out, the sequence number is not incremented, credits are not consumed, and the packet is not placed on the Replay buffer.
ForceECRCwoTD		No	Instructs Z3/Z4 Exerciser to include ECRC for this TLP even when the TD bit in the TLP's header is not set. This allows to create this type of error, that will effectively show up at the receiver as a TLP with one extra Dword of payload.
ForceTDwoECRC		No	Yes No
NVMeControllerReg	Controller Register code or number	Not a NVMe controller register access	Yes No

Note 1: The bit numbering for the Field command does NOT correspond to the numbering of the Header Fields view in PCIe Protocol Suite application. In that view, bytes are numbered from left to right. Bits in each byte are numbered from right to left from 0 to 7.
For the Field parameter the leftmost bit is bit 0 and it continues to the right to the rightmost bit that is numbered 31, and then it continues to the leftmost bit of the Second DWORD that is numbered 32 and so on.
So, to modify the Fmt field of the TLP, use Field[0:2], NOT Field[5:7]. To modify the Length field use Field[22:31], etc.

Note 2: Whenever a script is relying on the Tag value in the Completions returned by the DUT in response to the Requests sent from the script, select the Manual Tag generation policy in Generation Options (by checking the "Disable Automatic Tag generation" radio button in the Integrity tab). This way, the script can place specific values in the Tag field of the Requests it generates and expect the Request with that value of the Tag to be transmitted on the link and thus that particular value of the Tag returned in the Completion(s) sent by the DUT. This includes:

- Waiting on a Completion with a certain Tag value
- Branching on a Completion with a certain Tag value
- Using StoreData parameter on a Read Request TLP, including for further Break condition for a Loop instruction

Note 3: When using StoreData by itself or in conjunction with the LoopBreak instruction, caution has to be taken in order not to use offset/location that overlaps with any other data residing in the exerciser's memory (like data created with Structure instructions for NVMe Host emulation, for example).

Note 4: When sending Deferred Memory Writes (DMWr32, DMWr64) the receiving side (DUT) should be advertising enough Non-Posted Header and Data credits for the packet to be sent by the exerciser under normal conditions.

2.1.1.1 Z58/Z516/M616 IDE Related Parameters

IDE related parameters, only supported for Summit Z58, Z516, and M616 Exercisers.

Parameter	Values	Default	Comment
IDE_Prefix	Yes/No	No	When Yes, IDE Prefix is added to the TLP.
IDE_Stream_ID	0-255	0	Stream ID value for the IDE prefix.
IDE_SubStream	Posted, NonPosted or Completions	Posted (0)	SubStream value for the IDE prefix.
IDE_PR_Sent_Counter	0-255	0	PR sent counter value for the IDE prefix.
IDE_P IDE_M IDE_K IDE_T	Set/Cleared	Cleared	Values of the corresponding bits in the IDE prefix.
UseIDEKeyForEncryption	0-0xFFFFFFFF	No Key used	References the KeyIndex in Config=IDE_Key used to define the Key. If omitted (No Key used) when generating a single packet - no Encryption would be performed, and Payload and Length fields can be set in such way that and Encryption error is introduced. Omit this for all packets that make up the aggregation, except for the first one. PCRC and MAC calculation and presence is defined by the P and M bits in the IDE prefix.
Aggregation	Start/End	Auto	Use **Start** to send the first packet of the aggregation. **End** is set for the last packet of aggregation. If the parameter is omitted, the packet processing method is determined automatically: - a single packet if there are no pending aggregations with the corresponding IDE_Stream_ID and IDE_SubStream; - otherwise, the packet is treated as part of an aggregation.
IDE_PCRC	0-0xFFFFFFFF	Auto	If specified, replaces the calculated PCRC value. Can be used for introducing PCRC errors.
IDE_HdrEncryption	None Address_17_2, Address_25_2, Address_33_2, Address_41_2	None	Selects Partial Header Encryption Mode used when encrypting the TLP. Supported since PCIe 6.0.

NOTE: To generate IDE packets with integrity errors, the following sequence of actions must be performed:

1. Create correct encrypted IDE packet with specified length and payload.
2. Save the script.
3. Replace the “Payload” field value for the created packet with the concatenation of encrypted payload, encrypted PCRC (if present) and MAC. Despite changing the actual length of the packet payload, leave “Length” field value unchanged.
4. Remove the line specifying “UseIDEKeyForEncryption” field value, so no encryption will be performed.

5. Change any bit/byte of the new “Payload” field value to introduce integrity error.

Example 1:

Read one DWORD of data from address 0x1000.
Length parameter is not specified, so the default value of 1 is used.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      PSN = 0
      TLPType = MRd32
      Address = 0x1000
}
```

Example 2:

Read 32 DWORDs of data starting from address 0x1000.
PSN would accept values 0 for first TLP and 1 for second TLP.
TC, EP, Ordering, and **Snoop** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.
ECRC is not specified, so the **ECRC** is calculated by software.

```
Packet = TLP {
      PSN = Incr
      TLPType = MRd32
      Tag = 0
      Address = 0x1000
      TD = 1
      FirstDwBe = 0xF
      Length = 16
}
Packet = TLP {
      PSN = Incr
      TLPType = MRd32
      Tag = 1
      Address = 0x1010
      TD = 1
      FirstDwBe = 0xF
      Length = 16
}
```

Example 3:

This example does not specify **PSN, Tag,** and **LCRC**. Those values are calculated automatically by the Summit Exerciser hardware (see more on **Config = TLP** command, Page 99).

```
Config = TLP {
      AutoSeqNumber = Yes
      AutoLCRC = Yes
      TagGeneration = Default
}

Packet = TLP {
      TLPType = MRd32
      Address = 0x1010
```

```
      TD = 1
      Length = 1
}
```

Example 4:

This example shows how to specify reserved fields in the TLP header using the **Field** parameter:

```
Packet=TLP {
      TLPType=CfgRd0
      Register = 0x34
      Length = 1
      FirstDwBe = 0xF
      Field[0] = 0x1
      Field[8] = 0x1
      Field[12:15] = 0xF
      Field[20:21] = 0x3
      Field[80:83] = 0xF
}
```

Example 5:

This example shows how to specify the TLP type directly. Any invalid TLP type can be generated with this method.

```
Packet = TLP {
      TLPType = 0x4F
}
```

Example 6:

Repeat this TLP packet 64 times.

```
Packet = TLP {
      TLPType = MRd32
      Address = 0x1000
      Count = 64
}
```

Example 7:

This example shows how to use NVMeControllerReg parameter.

```
Packet=TLP {
      TLPType = MWr64
      AddressHi = 0x4
      AddressLo = 0xA0000000
      NVMeControllerReg = CC_ControllerConfig ; Address will be set to
                                                00000004:A0000014
      CC_IOCQ_EntrySize = 4
      CC_IOSQ_EntrySize = 6
      CC_ArbitrMechSel = WeightedRR_UPC
```

```
      CC_Enable = 1
}

Packet=TLP {
      TLPType = MWr64
      AddressHi = 0x4
      AddressLo = 0xA0000000
      NVMeControllerReg = SQTDBL_SubQTailDrbll ; Address will be set to
                                                 00000004:A0001010
      DoorbellQID = 2
      SQTDBL_SubQTail = 1
}

Packet=TLP {
      TLPType = MWr64
      AddressHi = 0x4
      AddressLo = 0xA0000000
      NVMeControllerReg = ASQ_AdminSQBaseAddr ; Address will be set to
                                                00000004:A0000028
      ASQ_AdminSubQBaseLo = 0xAB000000
      ASQ_AdminSubQBaseHi = 0x7
}


Packet=TLP {
      TLPType = MWr64
      AddressHi = 0x4
      AddressLo = 0xA0000000
      NVMeControllerReg = CAP_CtrllrCapabilities ; Address will be set
                                                   to 00000004:A0000000
      CAP_Reserved_63_56 = 1
      CAP_MemPgSizeMax = 1
      CAP_MemPgSizeMin = 1
      CAP_Reserved_47_45 = 1
      CAP_CmdSetSprtd = 1
      CAP_Reserved_23_19 = 1
}


Packet=TLP {
      TLPType = MRd64
      AddressHi = 0x4
      AddressLo = 0xA0000000
      NVMeControllerReg = CSTS_ControllerStatus ; Address will be set
                                                  to 00000004:A000001C
}
```

Example 8: Lightweight Notification Protocol

These examples show how to generate the Lightweight Notification Protocol. LN Mem TLPs are genarted by setting LN field, LN Messages are generated by using the proper message type.

The first example is to generate a MRd32 with Lightweight Notification enabled:

```
Packet=TLP
{
      TLPType = Mrd32
      LN = 1
}
```

The second example shows how to generate a Vendor Defined Message with the Lightweight Notification Protocol:

```
Packet=TLP
{
      TLPType = MsgD
      Length = 1
      MessageCode = Vendor_Defined_Type1
      VendorID = PCI_SIG
      Subtype = LN
      Payload = Zeroes
}
```

2.1.1.2 TLPType = MRd32, MRdlk32, MWr32, CAS32, Swap32, FewtchAdd32

Parameter	Values	Default	Comment
LastDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
Address	0x00000000 – 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.

Example 1:

This example shows how to send a 32-bit Memory Write TLP.
The **Length** field is not specified, so it would be calculated by software. (**Length = 4** would be used.)
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      TLPType = MWr32
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x1000
      Payload = ( 0x2, 0x4, 0x6, 0x8 )
}
```

Example 2:

This example shows how to send a 32-bit Memory Write TLP. This command would generate a random payload of 1024 DWORDs.

```
Packet = TLP {
      TLPType = MWr32
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x1000
      Length = 0            ; 0 means 1024 DWORDs of payload
      Payload = Random
}
```

2.1.1.3 TLPType = MRd64, MRdLk64, MWr64, CAS64, Swap64, FewtchAdd64

Parameter	Values	Default	Comment
LastDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
AddressLo	0x00000000 – 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.
AddressHi	0x00000000 – 0xFFFFFFFF	0	Bytes 12-15 in the TLP header.

Example 1:

This example shows how to send a 64-bit Memory Write TLP.
Length parameter is set to 3 intentionally in order to generate a TLP with incorrect length.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      TLPType = MWr64
      LastDwBe = 0xF
      FirstDwBe = 0xF
      AddressLo = 0x1000
      AddressHi = 0x60000000
      Payload = ( 0x2, 0x4, 0x6, 0x8, 0x2, 0x4, 0x6, 0x8 )
      Length = 3
}
```

2.1.1.4 TLPType = IoRd, IoWr

Parameter	Values	Default	Comment
LastDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
Address	0x00000000 – 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.

Example 1:

Read one DWORD of data from address 0x1000 of the IO address space.
Length parameter is not specified, so the default value of 1 is used.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software

```
Packet = TLP {
      TLPType = IoRd
      Address = 0x1000
}
```

2.1.1.5 TLPType = Cfgrd0, Cfgwr0, Cfgrd1, Cfgwr1

Parameter	Values	Default	Comment
LastDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0 – 15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
DeviceID	(XX:XX:X) or direct value	0	Bytes 8-9 in the TLP header. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
Register		0	Bytes 10-11 in the TLP header.

Example 1:

This example reads the Capability Pointer from the device's configuration space (**Bus Number 0, Device Number 2, Function Number 4**).

```
Packet = TLP {
      TLPType = CfgRd0
      DeviceId = (0:2:4)
      Register = 0x34
      Length = 1
      FirstDwBe = 0x1
}
```

Example 2:

This example writes to the Command Register of the device's configuration space (**Bus Number 0, Device Number 0, Function Number 1**).

```
Packet = TLP {
      TLPType = CfgWr0
      DeviceId = 1
      Register = 0x04
      Length = 1
      FirstDwBe = 0x3
      Payload = ( 0x03000000 )
}
```

2.1.1.6 TLPType = Msg, Msgd

Parameter	Values	Default	Comment
MessageRoute	ToRootComplex ByAddress ByID FromRootComplex Local Gather	ToRootComplex	**MessageRoute** affects the **Type** field of TLP header. (Bits 2:0).
MessageCode	Assert_INTA Assert_INTB Assert_INTC Assert_INTD Deassert_INTA Deassert_INTB Deassert_INTC Deassert_INTD PM_Active_State_Nak PM_PME PME_Turn_Off PME_TO_Ack ERR_COR ERR_NONFATAL ERR_FATAL Unlock Set_Slot_Power_Limit PTM_Request PTM_Response Vendor_Defined_Type0 Vendor_Defined_Type1 Attention_Indicator_On Attention_Indicator_Blink Attention_Indicator_Off Power_Indicator_On Power_Indicator_Blink Power_Indicator_Off Attention_Button_Pressed Direct numeric values can also be used.	0	Byte 7 in the TLP Header
AddressHi	0x00000000 – 0xFFFFFFFF	0	Used only if **MessageRoute=ByAddress**
AddressLo	0x00000000 –	0	Used only if

Parameter	Values	Default	Comment
	0xFFFFFFFF		**MessageRoute=ByAddress**
DeviceID	(XX:XX:X) or direct value	0	Used only if **MessageRoute=ById**. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)

2.1.1.6.1 For PTM Response message

Parameter	Values	Default	Comment
PTM_MasterTimeHi	0x00000000 – 0xFFFFFFFF	0	Bits 63:32 of the PTM Master Time
PTM_MasterTimeLo	0x00000000 – 0xFFFFFFFF	0	Bits 31:0 of the PTM Master Time
PTM_PropagationDelay	0x00000000 – 0xFFFFFFFF	0	PTM Propagation Delay

Example 1:

This example shows how to send a **PME_Turn_Off** Power Management Message while emulating the Root Complex.

```
Packet = TLP {
      TLPType = Msg
      MessageCode = PME_Turn_Off
      MessageRoute = FromRootComplex
}
```

Example 2:

This example shows how to send a **Vendor_Defined_Type0** Vendor Defined Message to the function 1 of device 1 on bus 0.

```
Packet = TLP {
      TLPType = Msg
      MessageCode = Vendor_Defined_Type0
      MessageRoute = ByID
      DeviceID = (0:1:1)
}
```

2.1.1.6.2 For MCTP and NVMe-MI Packets Utilizing PCIe Vendor Defined Message TLPs

MCTP and NVMe-MI packets are utilizing PCIe Vendor Defined message TLPs as a transport. For these types of packets the message code should be set to Vendor_Defined_Type1 and the Vendor ID should be set to the DMTF-defined code of 0x1AB4. When this is defined for a Message TLP, the script editor is going to recognize the packet as MCTP/NVMe-MI packet and set up appropriate fields to define the whole of the packet. The fields with the appropriate options for setting the values will appear in the Parameter list on the right of the Script Editor window.

This will include MCTP Transport Header fields:

MCTP_HDR_Rsvd
MCTP_HDR_Version
MCTP_HDR_DestEPID
MCTP_HDR_SrcEPID
MCTP_HDR_SOM
MCTP_HDR_EOM
MCTP_HDR_TO
MCTP_HDR_Tag

as well as the

MCTP_MSG_Type field that can be set to MCTP_Control or NVMe_MI. Depending on this selection more fields would be available, allowing to define an MCTP Control Message, NVMe-MI Control Primitive, NVMe-MI Management Interface Command etc.

For example, for MCTP Control Message, Set Endpoint ID command:

MCTP_MSG_Type = MCTP_Control
MCTP_MSG_RqBit = Yes
MCTP_MSG_CmdCode = SET_EP_ID
MCTP_CMD_SEID_Operation = ForceEID
MCTP_CMD_SEID_EPID = 0xAB

for NVMe-MI Control Primitive:

MCTP_MSG_IC = 1
MCTP_MSG_Type = NVMe_MI
NVMeMI_MSG_ReqOResp = Request
NVMeMI_MsgType = Ctrl_Primitive
NVMeMI_CmdSlotId = CmdSlot1
NVMeMI_CP_Opcode = GetState
NVMeMI_CP_Tag = 1
NVMeMI_CPSP_CESF = Yes

for NVMe-MI Management Interface Command Configuration Set:

MCTP_MSG_IC = 1
MCTP_MSG_Type = NVMe_MI
NVMeMI_MSG_ReqOResp = Request
NVMeMI_MsgType = NVMe_MI_Cmd
NVMeMI_CmdSlotId = CmdSlot0
NVMeMI_Cmd_Opcode = Configuration_Set
NVMeMI_Cmd_ConfigId = SMBusI2C_Freq
NVMeMI_Cmd_PortId = 1

NVMeMI_Cmd_I2C_Freq = kHz_400

Note: The Length field for the MsgD TLP has to be set properly before defining the MCTP/NVMe-MI fields in the payload.

This means in general:

For MCTP Control Message Requests length should be set to 1 (for commands that don't include Request Data) and to 2 (for commands that include Request Data).

For NVMe-MI Control Primitives Length should be set to 3.

For most NVMe-MI Management Interface Commands Length should be set to 5.

For NVMe-MI Management Interface Commands that include Request Data Length should be set appropriately, and Payload can be used to define the Request Data. In this case, the first 4 DWORDs in the Payload should be set to zero as placeholders for the header fields, also the last DWORD should be set to zero as a placeholder for the Message Intergity Check CRC. **The Payload definition should be placed before any of the MCTP and NVMe-MI field definitions.** The following is the example of defining a VPD Write
Command that writes 8 bytes of VPD at offset 16:

```
Packet=TLP
      {
            TLPType=MsgD
            Length = 7          ; 4 DWORDs of the header, 2 DWORDs of Request
            Data and 1 DWORD for MIC-CRC
            RequesterId = 0x7
            Tag = 0x0
            MessageCode = Vendor_Defined_Type1
            VendorId = 0x1AB4

            Payload = (0x0 0x0 0x0 0x0 0x01020304 0x05060708 0x0); 4 DWORDs
            placeholder, 2 DWORDs of data to be written; and 1 DWORD
            placeholder for CRC

            MCTP_HDR_Rsvd = 0xA
            MCTP_HDR_Version = 1
            MCTP_HDR_DestEPID = 0xAB
            MCTP_HDR_SrcEPID = 0xCD
            MCTP_HDR_SOM = 1
            MCTP_HDR_EOM = 1
            MCTP_HDR_TO = 1
            MCTP_HDR_Tag = 5
            MCTP_MSG_IC = 1
            MCTP_MSG_Type = NVMe_MI
            NVMeMI_MSG_ReqOResp = Request
            NVMeMI_MsgType = NVMe_MI_Cmd
            NVMeMI_Cmd_Opcode = VPD_Write
            NVMeMI_Cmd_VpdRdWr_DOFST = 16
            NVMeMI_Cmd_VpdRdWr_DLEN = 8

      }
```

NOTE: The Payload definition **must** be placed before any of the MCTP and NVMe-MI field definitions, as those definitions set different values to the fields within the Payload.

SPDM message type is supported by MCTP generation mechanism.

Example:

```
Packet=TLP
{
    TLPType=MsgD
    Length = 6
    RequesterID = (1:2:3)
    MessageRoute = ByID
    DeviceId = (3:2:1)
    Tag = 0x00000030
    MessageCode = Vendor_Defined_Type1
    VendorId = 0x1AB4
    MCTP_HDR_Version = 1
    MCTP_HDR_SOM = 1
    MCTP_HDR_EOM = 1
    MCTP_HDR_TO = 1
    MCTP_MSG_Type = SPDM
    SPDMVersion = 0x12
    SPDMRequestCode = GET_CAPABILITIES
    CTExponent = 0xAA
    HANDSHAKE_IN_THE_CLEAR_CAP = 1
    DataTransferSize = 1024
    MaxSPDMmsgSize = 4096
}
```

See 2.1.6.1 for details on generation of SPDM messages.

Emulating Correctable or Uncorrectable errors from device:
To emulate device reporting of Correctable or Uncorrectable errors to the system while in Device

Emulation, the following can be done:

1. Prepare the correctable or uncorrectable error status;

2. Generate the correctable or uncorrectable error message from script.

To do 1 (assuming the Device Emulator is booted in the System Under Test):

- Read the Device Emulator Configuration Space in using Address Space Read functionality. This will open it in the Configuration Space Editor.

- Modify the values of the relevant registers: Uncorrectable/Correctable Error Status, Header Log, TLP Prefix Log to reflect the desired error type and log data for the error.

- Write the Configuration Space back.

To do 2:

Put the correctable, nonfatal or fatal error message TLP in a script:

```
Packet=TLP
{
    TLPType=Msg
    RequesterId = (1:0:0)
    MessageCode = ERR_COR
}

Packet=TLP
{
    TLPType=Msg
    RequesterId = (1:0:0)
    MessageCode = ERR_NONFATAL
}

Packet=TLP
{
    TLPType=Msg
    RequesterId = (1:0:0)
    MessageCode = ERR_FATAL
}
```

And execute this script.

2.1.1.7 TLPType = Cpl, CplLk, CplD, CplDLk

Parameter	Values	Default	Comment
CompleterId	(XX:XX:X) or direct value	0	Identifies the Completer. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
ComplStatus	SC UR CRS CA	SC	Indicates the completion status.
BCM	0 – 1	0	**Byte Count Modified**: Must not be set by PCI Express Completers and may only be set by PCI-X completers. Indicates that the Byte Count field reports the size of just the first packet instead of the entire remaining byte count.
ByteCount	0 – 4095	0	Remaining byte count for the request
LowerAddr	0 – 63	0	Lower byte address for the starting byte of the completion

Note: You can specify the automatic Tag value modifiers to make the Completion automatically respond to incoming requests. See the modifier values for the Tag parameter on page 5, in the Packets = TLP table, for reference.

Example 1:

This example shows how to send a Completion TLP. This Completion TLP returns Unsupported Request (UR) status.
Requester is Function 0 of Device 0 on Bus 0.
Completer is Function 0 of Device 1 on Bus 0.
This completes the TLP request with Tag Number 4.

```
Packet = TLP {
      TLPType = Cpl
      RequesterId = (0:0:0)
      CompleterId = (0:1:0)
      Tag=4
      ComplStatus = UR
}
```

Example 2:

This example shows how to send a Completion with Data TLP. This Completion TLP returns Successful Completion (SC) status.
Requester is Function 0 of Device 0 on Bus 0.
Completer is Function 0 of Device 1 on Bus 0.
This completes the TLP request with Tag Number 4.
This is the last Completion of the Split Transaction, since **ByteCount** field is equal to the number of bytes transferred and **BCM** is not set.

```
Packet = TLP {
      TLPType = CplD
      RequesterId = (0:0:0)
      CompleterId = (0:1:0)
      Tag=4
      ComplStatus = SC
      ByteCount = 32
      Payload = ( 0x00000001, 0x00000002, 0x00000003, 0x00000004,
                  0x00000005, 0x00000006, 0x00000007, 0x00000008 )
}
```

2.1.1.8 TLP Field or Payload Substitution

When the Summit Z3-16/Z416 Exerciser is Emulating a PCI Express™ Endpoint Device

When you use the Summit Z3-16/Z416 Exerciser to emulate a PCI Express Endpoint Device, you must enable automatic handling of the device Configuration space.

You can also enable automatic handling of Memory and IO spaces. If you enable automatic handling of Memory and IO spaces, and you perform BAR setup for the initial device Configuration Space image, you can enable up to three memory spaces (Mem64, Mem32 A, and Mem32 B) and up to two IO spaces (IO A and IO B). The enabled address spaces have a corresponding data image in the internal memory of the Summit Z3-16/Z416 Exerciser. Whatever the system writes into those spaces over PCI Express can be read back over PCI Express.

The Summit Exerciser script language provides extensions that allow data written into Configuration, Memory, or IO spaces to modify behavior of the running script. If the script specifies values of specific fields and/or Payload in the TLPs for transmission, you can define that the system use the value from a location in a Configuration, Memory, or IO space of the device that the Summit Z3-16 and Summit Z416 Exercisers are emulating.

Field payload substitution allows modifying the behavior of running scripts. When a script that has substitution definitions is running, a test scenario can write new values to specific locations in Configuration, Memory, or IO spaces. Subsequent TLPs transmitted by the script use the updated values of the corresponding fields and/or Payload.

A substitution definition has the following format:

```
...
FieldName = ( address_space_designation, [address_space_offset], [SwapBytes])
...
```

The **address_space_designation** parameter specifies the address space from which to take the value for the field. The defined address-space-ID keywords are:

- FROM_CFG
- FROM_MEM64
- FROM_MEM32_A
- FROM_MEM32_B
- FROM_IO_A
- FROM_IO_B

The **address_space_offset** parameter is optional. It defines the byte offset into the address space for the location from which to take the substitution value. If you omit this parameter, the offset is zero and the system takes the value from the beginning of the specified address space.

The **SwapBytes** parameter, when present, specifies that the Summit Exerciser Z3/Z4 should swap the Endian order of the bytes in a 2-byte or 4-byte field before copying it into the specified field of the TLP to transmit.

Some example field name substitution definitions are:

```
Address = ( FROM_MEM32_A, 32 )
```

The system takes the value of the Address field from the Mem32 A address space, starting with byte 32 from the beginning of the memory space image.

```
Tag = ( FROM_CFG, 256 )
```
The system takes the value of the Tag field from Configuration Space byte location 256 (which is register location 64).

```
Payload = ( FROM_MEM64 )
```
The system copies the data Payload stream for this TLP from the beginning of the Memory Space 64 image.

Scripts can define substitution only for the following TLP fields:

- Payload
- Tag
- RequesterId
- CompleterId
- Address
- AddressLo
- AddressHi
- ComplStatus

Note: The Summit Z3-16/Z416 Exerciser tracks all accesses and locations of address spaces. The substitution **address_space_offset** parameter can be the string value **LAST_WRITTEN**. When performing a substitution that uses **LAST_WRITTEN** for the **address_space_offset** parameter, the Summit Z3-16/Z416 Exerciser uses the value from the beginning of the location where the system performed the latest **Write** transaction. The **LAST_WRITTEN** option allows writing new values at random locations or at locations that the script writer cannot determine beforehand and making such values available for later field or Payload substitution.

An example Packet definition with field name substitution using the **LAST_WRITTEN** option is:

```
Packet=TLP
{
  TLPType=MemWr32
  Length = 16
  ...
  Payload= ( FROM_MEM32_A, LAST_WRITTEN )
}
```

If the script transmits this TLP packet in a loop, initially the system sends a Payload filled with zeroes (or whatever value initialized memory space 32 A). Before the system writes into a memory space, the last-written address initializes to the beginning of the memory space, so the **LAST_WRITTEN** parameter starts at 0. Now suppose that the system writes into memory space 32 A at offset 0x100. The next transmitted TLP Payload specified by the above code includes 16 dwords of the data the system wrote starting at offset 0x100 of memory space 32 A.

2.1.1.9 TLPs Orthogonal Header Content (OHC)

For PCIe 6.0 and CXL 3.0, we have OHC-A(0b1), OHC-B(0b10), OHC-C(0b100). TLPs support the next OHCs.

TLP	OHC
Cfg	OHC-A3, OHC-B, OHC-C
Mem32, Mem64	OHC-A1, OHC-B, OHC-C
Msg	OHC-A4(if routed by ID) OHC-A1(if routed by address or routed to root complex), OHC-B(if routed by address) OHC-C
IO	OHC-A2, OHC-B, OHC-C
Cpl	OHC-A5, OHC-C

OHC contains 2 field types: specific and legacy.

- A specific field is a new field(was added in 6.0).
- A legacy field is an old field(was added before 6.0).

OHC	Specific Fields	Legacy Fields
OHC-A1	NW, PMR, ER, PASID, PV (PASID_A1, PV_A1 in gen scripts)	FirstDwBe, LastDwBe
OHC-A2	None	FirstDwBe, LastDwBe
OHC-A3	DSV, DestinationSegment	FirstDwBe, LastDwBe
OHC-A4	DSV, DestinationSegment, PASID, PV (PASID_A4, PV_A4 in gen scripts)	None
OHC-A5	DSV, DestinationSegment, CompleterSegment	ComplStatus
OHC-B	FM_TLP_OHC_B_PH, HV, AMA, AV	SteeringTag
OHC-C	RequesterSegment, RSV	DE_PR_Sent_Counter, IDE_Stream_ID, IDE_SubStream, IDE_K, IDE_T

2.1.1.9.1 Error/Warning Processing

We need to be compatible with old scripts, so in the case of a legacy field we set OHC field.

```
PCIeFlitMode=True
Packet=TLP
{
      TLPType = MRd64
      SteeringTag = 1
}
```

OHC is 0b10 because of SteeringTag (OHC-B's legacy field)
For specific fields, we always set the error if OHC is incorrect or missing.

```
PCIeFlitMode=True
Packet=TLP
{
      TLPType = MRd64
      RequesterSegment = 1
```

```
}
```

1. No OHC field and a legacy field mean a warning:

```
PCIeFlitMode=True
Packet=TLP
{
    TLPType = MRd64
    SteeringTag = 1 ; OHC-B legacy field
}
```

2. Incorrect OHC field and a legacy field mean an error:

```
PCIeFlitMode=True
Packet=TLP
{
    TLPType = MRd64
    OHC = 4         ; OHC-C
    SteeringTag = 1 ; OHC-B legacy field
}
```

3. No OHC field and a specific field mean an error:

```
PCIeFlitMode=True
Packet=TLP
{
    TLPType = MRd64
    RequesterSegment = 1 ; OHC-C specific field
}
```

4. Incorrect OHC field and a specific field mean an error:

```
PCIeFlitMode=True
Packet=TLP
{
    TLPType = MRd64
    OHC = 1              ; OHC-A
    RequesterSegment = 1 ; OHC-C specific field
}
```

2.1.2 Packet = DLLP

This command initiates transmission of DLLP packets on the bus.
Parameters for the **Packet = DLLP** command cover all the fields in a DLLP.

Parameter	Values	Default	Comment
DLLPType	Ack Nak InitFC1_P InitFC1_NP InitFC1_Cpl InitFC2_P InitFC2_NP InitFC2_Cpl UpdateFC_P UpdateFC_NP UpdateFC_Cpl PM_Enter_L1 PM_Enter_L23 PM_Active_State_Request_L1 PM_Request_Ack Vendor NOP		First byte in the DLLP
CRC	0 – 65535	Automatically calculated	Bytes 4-5 in the DLLP. When not specified, it is calculated automatically.
Field[<start>:<end>] Field[<pos>]			The arbitrary DLLP field could be specified by using **Field** parameter. **Start, end**, and **pos** are bit positions from the beginning of DLLP. Position 0 corresponds to the Most Significant Bit of the first byte of DLLP. Position 31 corresponds to the Least Significant Bit of the last byte of DLLP. Use **Field[<start>:<end>]** syntax to specify multi bit field. Use **Field[<pos>]** to specify single bit field.
Count	1 – 65535	1	Repeats this packet the number of times specified.

Example 1:

This example shows how to send a **PM_Active_State_Request_L1** power management DLLP. This DLLP would be sent 132 times.
The DLLP's **CRC** is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = PM_Active_State_Request_L1
      Count = 132
}
```

Example 2:

This example shows how to send a DLLP with an incorrect CRC.

```
Packet = DLLP {
      DLLPType = PM_Enter_L1
      CRC = 0x1234
}
```

Example 3:

This example shows how to specify reserved fields in a DLLP using the **Field** parameter.

```
Packet = DLLP {
      DLLPType = Ack
      Field[8:19] = 0b101001000111
}
```

2.1.2.1 DLLPType = Ack, Nak

Parameter	Values	Default	Comment
isValidAck	Yes No		When this variable is set to Yes the exerciser will treat this as an actual ACK DLLP and will cause flow control and sequence number to increment. If this parameter is set to No, the custom Ack DLLP will not cause the flow control and sequencer numbers to advance.
AckNak_SeqNum	0 – 4095	0	Bytes 2-3 in the DLLP

Example 1:

This example acknowledges all TLP packets with a sequence number less than or equal to 120 and initiates retransmission of TLP packets with a sequence number more than 120. The DLLP's CRC is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = Ack
      AckNak_SeqNum = 120
}
```

2.1.2.2 DLLPType = InitFC1_P, InitFC1_NP, InitFC1_Cpl, InitFC2_P, InitFC2_NP, InitFC2_Cpl, UpdateFC_P, UpdateFC_NP, UpdateFC_Cpl

Parameter	Values	Default	Comment
VC_ID	0 – 7	0	Virtual Channel, bits 2:0 in the first byte of the DLLP
HdrFC	0 – 255	0	Contains the credit value for headers of the indicated type (P, NP, or Cpl)
DataFC	0 – 4095	0	Contains the credit value for payload Data of the indicated type (P, NP, or Cpl)

Example 1:

The following example initializes credits for VC 0 for posted TLP requests.
Credit value for headers is 0. Credit value for data payload is infinite.
The DLLP's **CRC** is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = InitFc1_P
      VC_ID = 0
      HdrFC = 2
      DataFC = 0

}
```

2.1.2.3 DLLPType = Vendor

Parameter	Values	Default	Comment
Data	0x000000 – 0xFFFFFF	0	Vendor specific data, bytes 1-3 in the DLLP

Example 1:

```
Packet = DLLP {
      DLLPType = Vendor
      VendorSpecific = 0x010203
}
```

2.1.2.4 Sending modified (user defined) Ack or Nak for an incoming TLP

In an earlier version of PCIe exercisers at lower PCIe speeds it was possible to program a wait for a TLP followed by a manual DLLP defined by the script in order to transmit a nontrivial Ack or Nak or corrupted DLLP as a response to the TLP.

In the Summit exercisers the manual response DLLP has to be pre-programmed in BusEngine for it to be ready to respond with it quickly when the TLP comes from the link partner.

Option 1. Available on Summit M616, Z416, Z516 and Z58 (not Z3-16).
Use Config=AckNak with the ModifyAck action. See description and Example 3 in 2.4.6 Config = AckNak.

Option 2. The DLLP can be sent as a response only while the Automatic Ack/Nak policy in the BusEngine is disabled.

With this option the scripting for sending a manual DLLP Ack/Nak scenario would look like:

1. Set AckNak = Disabled
2. Send the Request TLP
3. Wait for the Response TLP
4. Send the response Ack/Nak the way you want it to go out
5. Set AckNak = Auto

The Exerciser software when seeing Ack/Nak policy disabled would look ahead to find the DLLP to be sent in step 4. and make sure it is pre-programmed to BusEngine before the Response TLP comes. This scenario can happen for 1 TLP at a time.

Example:

```
      ;
      ; Code to train the link to L0
      ;

      Config=AckNak
      {
          AckNak = Disable
      }


/* Request a configuration read of register 0 (Device and Vendor IDs)
*/
      Packet=TLP
      {
          TLPType=CfgRd0
          Length = 1
          FirstDwBe = 0xF
      }

/* Wait for the Configuration Read Completion (with data) */
      wait=TLP
      {
           TLPType = CplD
      }
```

```
/* Acknowledge the Configuration Read Completion. Use the modified Ack
DLLP. */
      packet=DLLP
      {
          DLLPType = Ack
          AckNak_SeqNum = 0 ; Expected for the fitst TLP sent after link
          training
          Field[8:19] = 0b101010101010  ; Modify the reserved filed of
      the
          DLLP to have non-zero bits

          IsValidAck = Yes
      }

/* Switch back to the 'Automatic' ACK/NAK policy */
      Config=AckNak
      {
          AckNak = Auto
      }

      wait = 500000

      Config=AckNak
      {
          AckNak = Disable
      }

/* Request a configuration read of register 0 (Device and Vendor IDs)
*/
      Packet=TLP
      {
          TLPType=CfgRd0
          Length = 1
          FirstDwBe = 0xF
      }

/* Wait for the Configuration Read Completion (with data) */
      wait=TLP
      {
           TLPType = CplD
      }


/* Acknowledge the Configuration Read Completion. Use the modified Ack
DLLP. */
```

```
      packet=DLLP
      {
          DLLPType = Ack
          AckNak_SeqNum = 1 ; Expected for the second TLP after link
retrain
          CRC = 0xABCD       ; Set bad CRC for the Ack DLLP
      }




/* Switch to the 'Always Ack' ACK/NAK policy */
      Config=AckNak
      {
          AckNak = Auto
      }

      ; etc.
```

2.1.3 Packet = CXL_Cache

This command initiates transmission of a CXL.cache packet on the bus. All the CXL.cache/mem packets will be automatically packed into flits according to the CXL specification rules. This command is only available for the M616 and Z516 Exercisers and requires CXL Mode to be enabled in generation options. Other parameters of the **Packet = CXL_Cache** command depend on the **CXLCacheType** parameter, which defines the type of the CXL.cache packet.

Parameter	Values	Default	Comment
CXLCacheType	H2D_CacheReq H2D_CacheResp H2D_CacheData D2H_CacheReq D2H_CacheResp D2H_CacheData	The parameter is required, there is no default value for it.	Depending on the **CXLCacheType**, the other parameters vary. **H2D_*** values can be used only if “Host“ Emulation Role is chosen in Generation Options setup. **D2H_*** values can be used only if “Device“ Emulation Role is chosen in Generation Options setup. It cannot be set if packet is created from a template (as type has already been specified there).

2.1.3.1 CXLCacheType = H2D_CacheReq

Parameter	Values	Default	Comment
H2DReqOpcode	SnpData SnpInv SnpCurr	0 (reserved value)	Indicates the kind of H2D request.
Address	0 – 2^{46-1}	0	Indicates which cache line the request targets. Note that according to the CXL specification the parameter specifies the [51:6] bits of the address.
AutoIncrementAddress *Only in 68BFlitMode	Yes No	No	If **Count** parameter is more than 1, indicates if the **Address** should be updated for each subsequent request.
AddressIncrementValue *Only in 68BFlitMode	0 – 65535	1	If **Count** parameter is more than 1, specifies the value which **Address** should be updated with for each subsequent request.
UQID	0 – 4095	Automatically generated	Unique Queue ID: indicates which Host entry is the source of the request. If not specified explicitly, it will be automatically generated to be unique.
CacheID	0-15	0	Logical CacheID of the destination of the message. Value is

Parameter	Values	Default	Comment
*Only in CXL 256B Flit Mode			assigned by Switch edge ports and not observed by the device.
Count *Only in 68BFlitMode	1 – 255	1	Repeats this packet the number of times specified.
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.3.2 CXLCacheType = H2D_CacheResp

Parameter	Values	Default	Comment
H2DRespOpcode	WritePull GO GO_WritePull ExtCmp GO_WritePull_Drop Fast_GO Fast_GO_WritePull GO_ERR_WritePull	0 (reserved value)	Indicates the kind of the H2D response.
UQID	0 – 4095	0	Unique Queue ID. It is relevant for **H2DRespOpcode** is one of the following values: **WritePull**, **GO_WritePull**, **GO_WritePull_Drop**, **Fast_GO_WritePull**, **GO_ERR_WritePull**. It is stored in RspData field of the packet (see the CXL specification).
CacheState	Shared Exclusive Invalid Error Modified	0 (reserved value)	It is relevant for **H2DRespOpcode = GO**. It is stored in RspData field of the packet (“MESI information”, see the CXL specification).
RSP_PRE	MissLocal Hit MissRemote	MissLocal	Carries the performance monitoring information for requests that do not receive data.
CQID	0 – 4095	0	Command Queue ID: This is a reflection of the CQID sent with the D2H Request and indicates which device entry is the target of the response.

Parameter	Values	Default	Comment
CacheID *Only in CXL 256B Flit Mode	0 – 15	0	Logical CacheID of the destination of the message. Value is assigned by Switch edge ports and not observed by the device.
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.3.3 CXLCacheType = H2D_CacheData

Parameter	Values	Default	Comment
CQID	0 – 4095	0	Command Queue ID: This is a reflection of the CQID sent with the D2H Request and indicates which device entry is the target of the data transfer.
ChunkValid	Lower Upper	Lower	In case of a 32B transfer on CXL.cache, this indicates which 32 byte chunk of the cacheline is represented by this transfer. This field is ignored for a 64B transfer.
Poison	Yes No	No	Indicates to the device that this data is corrupted and as such should not be used.
GO_Err	Yes No	No	The GO-ERR field indicates to the agent that this data is the result of an error condition and should not be cached or provided as a response to snoops.
CacheID *Only in CXL 256B Flit Mode	0-15	0	Logical CacheID of the destination of the message. Value is assigned by Switch edge ports and not observed by the device.
DropPayload	Yes No	No	Send data header without data
PayloadSeed	0 – 255	0	Specifies the fixed value from which to generate the payload.
PayloadSize *Only in 68B Flit Mode	32byte 64byte	64byte	If **Payload** is one of **Incr**, **Random**, **Zeros**, **Ones**, **AllOnes** values, **PayloadSize** determines the amount of payload to generate. If **Payload** is specified explicitly (with **(XXXX,XXXX,...)**) the size is determined automatically.
Payload	(XXXX,XXXX,...) Incr Random	Zeros	Specified packet data as the array of DWORDs in hexadecimal format (Big Endian). **Incr**: Specifies a payload as an increasing sequence

Parameter	Values	Default	Comment
	Zeros Ones AllOnes		(0, 1, …). **Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of ones (i.e. (1, 1, …)). **AllOnes**: Specifies a payload of all ones (i.e. (0xFFFFFFFF, 0xFFFFFFFF, …).
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.3.4 CXLCacheType = D2H_CacheReq

Parameter	Values	Default	Comment
D2HReqOpcode	RdCurr RdOwn RdShared RdAny RdOwnNoData ItoMWr MemWr CLFlush CleanEvict DirtyEvict CleanEvictNoData WOWrInv WOWrInvF WrInv CacheFlushed	0 (reserved value)	Indicates the kind of D2H request.
D2HReqOpcode	* In CXL 3.0 “MemWr” renamed to “WrCur”		
CQID	0 – 4095	Automatically generated	Command Queue ID: contains the ID of the tracker entry that is associated with the request. If not specified explicitly, it will be automatically generated to be unique.
NonTemporal	0 – 1	0	For cacheable reads the NonTemporal field is used as a hint to indicate to the Host how it should be cached.
Address	0 – 2^{46-1}	0	Carries the physical address of coherent requests. Note that according to the CXL

Parameter	Values	Default	Comment
			specification the parameter specifies the [51:6] bits of the address.
CacheID *Only in CXL 256B Flit Mode	0-15	0	Logical CacheID of the destination of the message. Value is assigned by Switch edge ports and not observed by the device.
AutoIncrementAddress *Only in 68B Flit Mode	Yes No	No	If **Count** parameter is more than 1, indicates if the **Address** should be updated for each subsequent request.
AddressIncrementValue *Only in 68B Flit Mode	0 – 65535	1	If **Count** parameter is more than 1, specifies the value which **Address** should be updated with for each subsequent request.
Count *Only in 68B Flit Mode	1 – 255	1	Repeats this packet the number of times specified.
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.3.5 CXLCacheType = D2H_CacheResp

Parameter	Values	Default	Comment
D2HRespOpcode	RspSHitSE RspIHitI RspIHitSE RspVHitV RspSFwdM RspIFwdM RspVFwd	0 (reserved value)	Indicates the kind of the D2H response.
UQID	0 – 4095	0	Unique Queue ID: This is a reflection of the UQID sent with the H2D Request and indicates which Host entry is the target of the response.
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.3.6 CXLCacheType = D2H_CacheData

Parameter	Values	Default	Comment
UQID	0 – 4095	0	Unique Queue ID: this is a reflection of the UQID sent with the H2D Response and indicates which Host entry is the target of the data transfer.
ChunkValid *Only in 68B Flit Mode	Lower Upper	Lower	In case of a 32B transfer on CXL.cache, this indicates what 32 byte chunk of the cacheline is represented by this transfer. This field is ignored for a 64B transfer.
Bogus	0 – 1	0	Indicates that the data associated with this evict message was returned to a snoop after the D2H request was sent from the device but before a WritePull was received for the evict. This data is no longer the most current, so it should be dropped by the Host.
BEP *Only in CXL 256B Flit Mode	Yes No	No	Byte-Enables Present - Indication that 5 data slots are included in the message where the 5th data slot carries the 64-bit Byte Enables.
DropPayload	Yes No	No	Send data header without data
Poison	Yes No	No	Indicates that this data chunk is corrupted and should not be used by the Host.
PayloadSeed	0 – 255	0	Specifies the fixed value from which to generate the payload.
PayloadSize *Only in 68B Flit Mode	32byte 64byte	64bytes	If **Payload** is one of **Incr**, **Random**, **Zeros**, **Ones**, **AllOnes** values, **PayloadSize** determines the amount of payload to generate. If **Payload** is specified explicitly (with **(XXXX,XXXX,...)**) the size is determined automatically.
ByteEnable	(XXXX,XXXX,...)	(0xFFFFFFFF, 0xFFFFFFFF, 0, 0)	Specifies packet Byte Enable data as the array of DWORDs in hexadecimal format (Big Endian). Reserved bits in the end can also be set to create erroneous case.
Payload	(XXXX,XXXX,...) Incr Random Zeros	Zeros	Specified packet data as the array of DWORDs in hexadecimal format (Big Endian). **Incr**: Specifies a payload as an increasing sequence (0, 1, …).

Parameter	Values	Default	Comment
	Ones AllOnes		**Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of ones (i.e. (1, 1, …)). **AllOnes**: Specifies a payload of all ones (i.e. (0xFFFFFFFF, 0xFFFFFFFF, …).
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.4 Packet = CXL_Mem

This command initiates transmission of a CXL.mem packet on the bus. All the CXL.cache/mem packets will be automatically packed into flits according to the CXL specification rules. This command is only available for the M616 and Z516 Exercisers and requires CXL Mode to be enabled in generation options. Other parameters of the **Packet = CXL_Mem** command depend on the **CXLMemType** parameter, which defines the type of the CXL.mem packet.

Parameter	Values	Default	Comment
CXLMemType	M2S_MemReq M2S_MemReqWithData S2M_MemNoDataResp S2M_MemDataResp Two more values in CXL 256B Flit Mode: M2S_MemBIResponse S2M_MemBISnoop	The parameter is required, there is no default value for it.	Depending on the **CXLMemType**, the other parameters vary. **M2S_*** values can be used only if “Host“ Emulation Role is chosen in Generation Options setup. **S2M_*** values can be used only if “Device“ Emulation Role is chosen in Generation Options setup. It cannot be set if packet is created from a template (as type has already been specified there).

2.1.4.1 CXLMemType = M2S_MemReq

Parameter	Values	Default	Comment
MemReqOpcode	MemInv MemRd MemRdData MemRdFwd	MemInv	Specifies which operation needs to be performed on the data and associated information.

Parameter	Values	Default	Comment
	MemWrFwd MemInvNT More values in CXL 256B Flit Mode: • MemSpecRd • MemClnEvct More values in CXL 3.1: • MemRdTEE • MemRdDataTEE • MemSpecRdTEE • TEUpdate		
SnpType	NoOp SnpData SnpCur SnpInv	NoOp	Snoop Type: specifies which snoop type, if any, needs to be issued by the DCOH and the minimum coherency state required by the Host.
MetaField	Meta0State NoOp More values in CXL 3.1: ExtMetaState	Meta0State	Specifies which, if any, Meta Data Field needs to be updated.
MetaValue	For **MetaField = Meta0State**: Invalid Any Shared For **MetaField = ExtMetaState:** ExplicitNoOp	Invalid	Specifies the value of the field needs to be updated to.
Tag16	For **MemReqOpcode** in [**MemRdFwd**, **MemWrFwd**]: 0: 4095 (CQID) Otherwise: 0: 65535	0	This value needs to be reflected with the response from the Subordinate so the response can be routed appropriately.
AutoIncrementTag *Only in 68B Flit Mode	Yes No	Generation Options ->Integrity ->CXL.cache/mem ->Autoincrement Tag	If **Count** parameter is more than 1, indicates if the **Tag16** should be updated for each subsequent request.

Parameter	Values	Default	Comment
		By default, this option is No	
Address	0 – 2^{47-1}	0	Specifies the Host Physical Address associated with the MemOpcode. Address[5] is provisioned for future usages such as critical chunk first. Note that according to CXL specification the parameter specifies the [51:5] bits of the address.
AutoIncrementAddress *Only in 68B Flit Mode	Yes No	No	If **Count** parameter is more than 1, indicates if the **Address** should be updated for each subsequent request.
AddressIncrementValue *Only in 68B Flit Mode	0 – 65535	1	If **Count** parameter is more than 1, specifies the value which **Address** should be updated with for each subsequent request.
TC	0 – 3	0	Traffic Class: can be used by the Master to specify the Quality of Service associated with the request.
Count *Only in 68B Flit Mode	0 – 255	1	Repeats this packet the number of times specified.
LD-ID * not in PBR mode	0 - 15	0	Logical Device Identifier: This identifies a Logical Device within a Multiple-Logical Device. Not applicable in PBR mode where SPID infers this field.
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID
CKID *CXL 3.1 only in 256B Flit mode	0 – 8192	0	Context Key ID: Optional key ID that references preconfigured key material utilized for device-based data-at-rest encryption.

Parameter	Values	Default	Comment
			If the device has been configured to utilize CKID-based device encryption and locked utilizing the CXL Trusted Execution Environment (TEE) Security Protocol (TSP), then this field shall be valid for Data Read access types (**MemRd/MemRdTEE/ MemRdData*/MemSpecRd***) and treated as reserved for other messages

2.1.4.2 CXLMemType = M2S_MemBIResponse

Parameter	Values	Default	Comment
MemBIRespOpcode	BIRspI BIRspS BIRspE BIRspIBlk BIRspSBlk BIRspEBlk	BIRspI	Specifies response type
BIID	0 – 4095	0	BI-ID of the device that is the destination of the message.
BITag	0 – 4095	0	Tracking ID from the device
LowAddr	0 - 3	0	The lower 2 bits of Cacheline address (Address[7:6]). This is needed to differentiate snoop responses when a Block Snoop is sent and receives snoop response for each cacheline. For block response (opcode names *Blk), this field is Reserved.
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.4.3 CXLMemType = M2S_MemReqWithData

Parameter	Values	Default	Comment
MemReqWithDataOpcode	MemWr MemWrPtl One more value in CXL 256B Flit Mode: BIConflict More values in CXL 3.1: MemRdFill MemWrTEE MemWrPtlTEE MemRdFillTEE	0 (reserved value)	Specifies which, if any, operation needs to be performed on the data and associated information.
SnpType	NoOp SnpData SnpCur SnpInv	NoOp	Snoop Type: this specifies which snoop type, if any, needs to be issued by the DCOH and the minimum coherency state required by the Host.
MetaField	Meta0State NoOp More values in CXL 3.1: * ExtMetaState	Meta0State	Specifies which, if any, Meta Data Field needs to be updated.
MetaValue	For **MetaField = Meta0State**: Invalid Any Shared For **MetaField = ExtMetaState** ExplicitNoOp	Invalid	Specifies the value of the field needs to be updated to.
Tag16	0 – 65535	0	This value needs to be reflected with the response from the Subordinate so the response can be routed appropriately.
Address	0 – 2^{46-1}	0	Specifies the Host Physical Address associated with the MemOpcode. Note that according to CXL

Parameter	Values	Default	Comment
			specification the parameter specifies the [51:5] bits of the address.
Poison	Yes No		Indicates that the data contains an error.
TC	0 – 3		Traffic Class: can be used by the Master to specify the Quality of Service associated with the request.
BEP *Only in CXL 256B Flit Mode	Yes No	No	Byte-Enables Present - Indication that 5 data slots are included in the message where the 5th data slot carries the 64-bit Byte Enables.
DropPayload	Yes No	No	Send data header without data
PayloadSeed	0 – 255	0	Specifies the fixed value from which to generate the payload.
ByteEnable	(XXXX,XXXX,...)	(0xFFFFFFFF, 0xFFFFFFFF, 0, 0)	Specifies packet Byte Enable data as the array of DWORDs in hexadecimal format (Big Endian). Reserved bits in the end can also be set to create erroneous case.
Payload	(XXXX,XXXX,...) Incr Random Zeros Ones AllOnes	Zeros	Specified packet data as the array of DWORDs in hexadecimal format (Big Endian). **Incr**: Specifies a payload as an increasing sequence (0, 1, …). **Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of ones (i.e. (1, 1, …)). **AllOnes**: Specifies a payload of all ones (i.e. (0xFFFFFFFF, 0xFFFFFFFF, …).

Parameter	Values	Default	Comment
LD-ID * not in PBR mode	0 - 15	0	Logical Device Identifier: This identifies a Logical Device within a Multiple-Logical Device. Not applicable in PBR mode where SPID infers this field.
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID
CKID *CXL 3.1 only in 256B Flit mode	0 – 8192	0	Context Key ID: Optional key ID that references preconfigured key material utilized for device-based data-at-rest encryption. If the device has been configured to utilize CKID-based device encryption and locked utilizing the CXL Trusted Execution Environment (TEE) Security Protocol (TSP), then this field shall be valid for Data Read access types (**MemRd/MemRdTEE/ MemRdData*/MemSpecRd***) and treated as reserved for other messages
TRP (Trailer Byte Present) (formerly BE) * only for CXL 3.1	Yes No	No	Trailer Present: Indicates that a trailer is included on the message This bit was formerly referred to as Byte-Enables Present (BEP), but has been redefined as part of an optional extension to support message trailers.
ExtendedMetaData *only for CXL 3.1	0xXXXXXXXX	No default value – must be specified if TRP set	Only when Opcode is MemWr/ MemWrTEE / MemWrPtl/ MemWrPtlTEE, MetaField = ExtMetaState and TRP is set

2.1.4.4 CXLMemType = S2M_MemNoDataResp

Parameter	Values	Default	Comment
MemNoDataRespOpcode	Cmp CmpS CmpE One more value in CXL 256B Flit Mode: BI_ConflictAck More values in CXL 3.1: CmpM CmpTEE	Cmp	Specifies which, if any, operation needs to be performed on the data and associated information.
MetaField	Meta0State NoOp More values in CXL 3.1: ExtMetaState	Meta0State	Meta Data Field: for devices that support memory with meta data, this is a reflection of the value sent in the associated M2S Req or M2S RwD. For devices that do not, this field is a don't care.
MetaValue	For **MetaField = Meta0State**: Invalid Any Shared For **MetaField = ExtMetaState**: ExplicitNoOp	Invalid	Meta Data Value: for M2S Req, for devices that support memory with meta data, this is the initial value of the Meta Data Field as read from memory for a M2S Req that does not return a S2M DRS. For M2S RwD and for devices that do not support memory with meta data, this field is a don't care
Tag16	0 – 65535	0	This is a reflection of the Tag field sent with the associated M2S Req or M2S RwD.
LDID *Only in 256B Flit Mode	0-15	0	Logical Device Identifier - This identifies a logical device within a multiple-logical device. Not applicable in PBR mode where DPID infers this field.
DevLoad	LightLoad OptimalLoad ModerateOverload SevereOverload	LightLoad	Device Load - Indicates device load

DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.4.5 CXLMemType = S2M_MemDataResp

Parameter	Values	Default	Comment
MemDataRespOpcode	MemData One more value in CXL 256B Flit Mode: MemData_NXM More values in CXL 3.1: MemDataTEE	MemData	Specifies which, if any, operation needs to be performed on the data and associated information.
MetaField	Meta0State NoOp More values in CXL 3.1: ExtMetaState	Meta0State	Meta Data Field: for devices that support memory with meta data, this is a reflection of the value sent in the associated M2S Req or M2S RwD. For devices that do not, this field is a don't care.
MetaValue	For **MetaField = Meta0State**: Invalid Any Shared For **MetaField = ExtMetaState** ExplicitNoOp	Invalid	Meta Data Value: for M2S Req, for devices that support memory with meta data, this is the initial value of the Meta Data Field as read from memory. For M2S RwD and for devices that do not support memory with meta data, this field is a don't care.
Tag16	0 – 65535	0	This is a reflection of the Tag field sent with the associated M2S Req or M2S RwD.
Poison	Yes No	No	Indicates that the data contains an error.
LDID *Only in 256B Flit Mode * not in PBR mode	0-15	0	Logical Device Identifier - This identifies a logical device within a multiple-logical device. Not applicable in PBR mode where DPID infers this field.
DevLoad	LightLoad OptimalLoad	LightLoad	Device Load - Indicates device load

Parameter	Values	Default	Comment
	ModerateOverload SevereOverload		
DropPayload	Yes No	No	Send data header without data
PayloadSeed	0 – 255	0	Specifies the fixed value from which to generate the payload.
PayloadSize *Only in 68B Flit Mode	32byte 64byte	64bytes	If **Payload** is one of **Incr**, **Random**, **Zeros**, **Ones**, **AllOnes** values, **PayloadSize** determines the amount of payload to generate. If **Payload** is specified explicitly (with **(XXXX,XXXX,...)**) the size is determined automatically.
Payload	(XXXX,XXXX,...) Incr Random Zeros Ones AllOnes	Zeros	Specified packet data as the array of DWORDs in hexadecimal format (Big Endian). **Incr**: Specifies a payload as an increasing sequence (0, 1, …). **Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of ones (i.e. (1, 1, …)). **AllOnes**: Specifies a payload of all ones (i.e. (0xFFFFFFFF, 0xFFFFFFFF, …).
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID
TRP (Trailer Byte Present) * only for CXL 3.1	Yes No	No	Trailer Present: Indicates that a trailer is included on the message This bit was formerly referred to as Byte-Enables Present (BEP)
ExtendedMetaData *only for CXL 3.1	0xXXXXXXXX	No default value – must be specified if TRP set	Only for MemData/MemDataTEE opcodes, when TRP is set and MetaField = ExtMetaState.

2.1.4.6 **CXLMemType = S2M_MemBISnoop**

Parameter	Values	Default	Comment
MemBISnoopOpcode	BISnpCur BISnpData BISnpInv BISnpCurBlk BISnpDataBlk BISnpInvBlk	BISnpCur	Specifies snoop type
BIID	0 – 4095	0	BI-ID of the device that is the destination of the message. Not applicable in PBR messages where SPID infers this field.
BITag	0 – 4095	0	Tracking ID from the device
Address	$0 – 2^{46 - 1}$	0	Host Physical Address. For *Blk opcodes, the lower 2 bits (Address[7:6]) are encoded as defined in Table 3-41 of the CXL 3.0 specification. Used for all other opcodes that represent the standard definition of Host Physical Address.
BlockEnable	LowerIsValid UpperIsValid FullIsValid	0 - Reserved	The lower 2 bits (Address[7:6]), used with *Blk opcodes
SPID *only in PBR mode	0 - 4095	0	SPID: Source PBR ID
DPID *only in PBR mode	0 - 4095	0	DPID: Destination PBR ID

2.1.5 Packet = SMBus

This command initiates transmission of a packet on the SMBus lines. The parameters of the Packet = SMBus command cover the fields in the SMBus part of the packet header and all the Host side MCTP and NVMe-MI packet fields.

The Packet = SMBus command is only available on SMBus-capable Summit Z3/Z4 Exercisers and Host Platforms.

The fields in the SMBus part of the packet header are:

Parameter	Values	Default	Comment
DestSlaveAddr	0 – 0xFF	0	SMBus Destination Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to zero for SMBus write transaction as the MCTP binding spec mandates.
CommandCode	0 – 0xFF MCTP NVMeMiBasicManagement PrepareToARP ResetDeviceGeneral ResetDeviceDirected AssignAddress NotifyARP GetUDIDGeneral GetUDIDDirected	0	SMBus command code. The MCTP constant sets it to 0x0F for MCTP over SMBus. Optional NVM Express Basic Management Command The other constants correspond to SMBus ARP commands.
ByteCntSMBus	0 – 255	0	Used only for **CommandCode = MCTP** and **CommandCode = AssignAddress** Should be set to the exact length corresponding to the desired packet.
SrcSlaveAddr	0 – 0xFF	0	SMBus Source Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to 1 as the MCTP binding spec mandates.

2.1.5.1 CommandCode = MCTP

The fields in the SMBus part of the packet header are:

Parameter	Values	Default	Comment
DestSlaveAddr	0 – 0xFF	0	SMBus Destination Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to zero for SMBus write transaction as the MCTP binding spec mandates.
SrcSlaveAddr	0 – 0xFF	0	SMBus Source Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to 1 as the MCTP binding spec mandates.

The rest of the fields are MCTP and NMVe-MI fields that are the same as in inband VDM transport binding (see 2.1.1.4.2).

This will include MCTP Transport Header fields:

MCTP_HDR_Rsvd
MCTP_HDR_Version
MCTP_HDR_DestEPID
MCTP_HDR_SrcEPID
MCTP_HDR_SOM
MCTP_HDR_EOM
MCTP_HDR_TO
MCTP_HDR_Tag

as well as the

MCTP_MSG_Type field that can be set to MCTP_Control or NVMe_MI. Depending on this selection more fields would be available, allowing to define an MCTP Control Message, NVMe-MI Control Primitive, NVMe-MI Management Interface Command etc.

For example, for MCTP Control Message, Set Endpoint ID command:

MCTP_MSG_Type = MCTP_Control
MCTP_MSG_RqBit = Yes
MCTP_MSG_CmdCode = SET_EP_ID
MCTP_CMD_SEID_Operation = ForceEID
MCTP_CMD_SEID_EPID = 0xAB

for NVMe-MI Control Primitive:

MCTP_MSG_IC = 1
MCTP_MSG_Type = NVMe_MI
NVMeMI_MSG_ReqOResp = Request
NVMeMI_MsgType = Ctrl_Primitive
NVMeMI_CmdSlotId = CmdSlot1
NVMeMI_CP_Opcode = GetState
NVMeMI_CP_Tag = 1
NVMeMI_CPSP_CESF = Yes

for NVMe-MI Management Interface Command Configuration Set:

MCTP_MSG_IC = 1
MCTP_MSG_Type = NVMe_MI
NVMeMI_MSG_ReqOResp = Request
NVMeMI_MsgType = NVMe_MI_Cmd
NVMeMI_CmdSlotId = CmdSlot0
NVMeMI_Cmd_Opcode = Configuration_Set
NVMeMI_Cmd_ConfigId = SMBusI2C_Freq
NVMeMI_Cmd_PortId = 1
NVMeMI_Cmd_I2C_Freq = kHz_400

Note: The ByteCntSMBus field is set to the count of bytes that follow the Byte Count field up to, but not including, the PEC byte.

This means in general:

For MCTP Control Message Requests Byte Count should be set to 10 (for commands that don't include request data).

And to 11 (for commands that include Request Data).

For NVMe-MI Management Interface Commands, Byte count should be set to 25.

For NVMe-MI Management Interface Commands that include Request Data Length should be set appropriately, and Payload can be used to define the Request Data. In this case, the first 4 DWORDs in the Payload should be set to zero as placeholders for the header fields, also the last DWORD should be set to zero as a placeholder for the Message Intergity Check CRC. **The Payload definition should be placed before any of the MCTP and NVMe-MI field definitions.** The following is the example of defining a VPD Write
Command that writes 8 bytes of VPD at offset 16:

```
Packet=SMBus
      {
            DestSlaveAddr = DEST_SLAVE_ADDR
            CommandCode=MCTP
            ByteCntSMBus = 25
            SrcSlaveAddr = SRC_SLAVE_ADDR

            Payload = (0x0 0x0 0x0 0x0 0x01020304 0x05060708 0x0); 4 DWORDs
            placeholder, 2 DWORDs of data to be written; and 1 DWORD
            placeholder
            for CRC

            MCTP_HDR_Rsvd = 0xA
            MCTP_HDR_Version = 1
            MCTP_HDR_DestEPID = 0xAB
            MCTP_HDR_SrcEPID = 0xCD
            MCTP_HDR_SOM = 1
            MCTP_HDR_EOM = 1
            MCTP_HDR_TO = 1
            MCTP_HDR_Tag = 5
            MCTP_MSG_IC = 1
            MCTP_MSG_Type = NVMe_MI
            NVMeMI_MSG_ReqOResp = Request
            NVMeMI_MsgType = NVMe_MI_Cmd
```

```
            NVMeMI_Cmd_Opcode = VPD_Write
            NVMeMI_Cmd_VpdRdWr_DOFST = 16
            NVMeMI_Cmd_VpdRdWr_DLEN = 8
      }
```

NOTE: The Payload definition **must** be placed before any of the MCTP and NVMe-MI field definitions, as those definitions set different values to the fields within the Payload.

SPDM message type is supported by MCTP generation mechanism.

Example:

```
Packet=TLP
{
    TLPType=MsgD
    Length = 6
    RequesterID = (1:2:3)
    MessageRoute = ByID
    DeviceId = (3:2:1)
    Tag = 0x00000030
    MessageCode = Vendor_Defined_Type1
    VendorId = 0x1AB4
    MCTP_HDR_Version = 1
    MCTP_HDR_SOM = 1
    MCTP_HDR_EOM = 1
    MCTP_HDR_TO = 1
    MCTP_MSG_Type = SPDM
    SPDMVersion = 0x12
    SPDMRequestCode = GET_CAPABILITIES
    CTExponent = 0xAA
    HANDSHAKE_IN_THE_CLEAR_CAP = 1
    DataTransferSize = 1024
    MaxSPDMmsgSize = 4096
}
```

See 2.1.6.1 2.1.6.1for details on generation of SPDM messages.

2.1.5.2 CommandCode = PrepareToARP, ResetDeviceGeneral

Parameter	Values	Default	Comment
PEC	0 – 0xFF	Automatically calculated value	Packet Error Code

2.1.5.3 CommandCode = ResetDeviceDirected

Parameter	Values	Default	Comment
TargetedSlaveAddr	0x10 – 0x7F	0x2E	7-bit Targeted Slave Address
PEC	0 – 0xFF	Automatically calculated value	Packet Error Code

2.1.5.4 CommandCode = AssignAddress

Parameter	Values	Default	Comment
UDID_DeviceCapabilities	0 – 0xFF	0	Device Capabilities field from UDID data
UDID_AddrType	FixedAddr DynamicAndPersistent DynamicAndVolatile Random	FixedAddr	Device Capabilities' Address type field
UDID_PECSupported	yes no	no	Device Capabilities' PEC Supported field
UDID_VerRev	0 – 0xFF	0	Version/Revision field from UDID data
UDID_UDIDVersion	0 – 0x7 Ver1 (001b)	0	Version/Revision's UDID Version field
UDID_SiliconRevID	0 – 0x7	0	Version/Revision's Silicon Revision ID field
UDID_VendorID	0 – 0xFFFF	0	Vendor ID field from UDID data
UDID_DeviceID	0 – 0xFFFF	0	Device ID field from UDID data
UDID_Interface	0 – 0xFFFF	0	Interface field from UDID data
UDID_ZONE	yes no	no	Interface's ZONE field
UDID_IPMI	yes no	no	Interface's IPMI field
UDID_ASF	yes no	no	Interface's ASF field
UDID_OEM	yes no	no	Interface's OEM field
UDID_SMBusVersion	0 – 0xF Ver1_0 (0000b) Ver1_1 (0001b) Ver2_0 (0100b) Ver3_0 (0101b)	Ver1_0	SMBus Version field from UDID data
UDID_SubsystemVendorID	0 – 0xFFFF	0	Subsystem Vendor ID field from UDID data
UDID_SubsystemDeviceID	0 – 0xFFFF	0	Subsystem Device ID field from UDID data
UDID_VendorSpecificID	0 – 0xFFFFFFFF	0	Vendor Specific ID field from UDID data
AssignedAddr	0 – 0x7F	0	Assigned Address field
PEC	0 – 0xFF	Automatically calculated value	Packet Error Code

Example:

```
Packet=SMBus
{
    CommandCode = AssignAddress
    UDID_AddrType = FixedAddr
    UDID_PECSupported = yes
    UDID_VerRev = 0x1
```

```
    UDID_UDIDVersion = Ver1
    UDID_SiliconRevID = 0x2
    UDID_VendorID = 0x54
    UDID_DeviceID = 0x55
    UDID_ZONE = yes
    UDID_IPMI = yes
    UDID_ASF = no
    UDID_OEM = yes
    UDID_SMBusVersion = Ver3_0
    AssignedAddr = 0x12
}
```

2.1.5.5 CommandCode = NotifyARP

Parameter	Values	Default	Comment
DataByteLow	0 – 0xFF	0	Data Byte Low field from Notify ARP Master command
DataByteHigh	0 – 0xFF	0	Data Byte High field from Notify ARP Master command

2.1.5.6 CommandCode=GetUDIDDirected

Parameter	Values	Default	Comment
TargetSlaveAddr	0x10 – 0x7F	0	7-bit Targeted Slave Address

Example:

```
Packet=SMBus
{
    CommandCode=GetUDIDGeneral ; Send Get UDID General ARP command
}

Packet=SMBus
{
    CommandCode=GetUDIDDirected ; Send Get UDID Directed ARP command to
the address below
    TargetedSlaveAddr = 0x53
}
```

2.1.5.7 CommandCode = NVMeMiBasicManagement

Executes the optional NVM Express Basic Management Command to address 0xD4 defined by the specification.

Parameter	Values	Default	Comment
BMCmdCode_Offset	Status (0) StaticData (8) VendorSpecific (32) ResetArbitrationBit (0xFF) RWORS_ReadWORptStartOffset0 (special read w/o a repeated start) or any numeric value of the offset, 0-0xFF	0 (beginnig of the Subsystem Management Data Structure)	Defines the command or starting offset into the Subsystem Management Data Structure. The predefained values also set the appropriate lenght to read, which can be overrid with the next parameter. RWORS_ReadWORptStartOffset0 executes special read command without a repeated start from offset zero.
BMCLengthToRead	Numeric, length to read	0, or as appropriate for the predefined command value	Can be used to modify default amount to read, or set the length to read when using non-predefined offset

Examples:

```
Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = Status ; Will read 8 bytes starting at offset 0
}

Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = StaticData  ; Will read 24 bytes starting at
offset 8
}

Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = VendorSpecific  ; Will read 32 bytes starting at
offset 32
}

Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = ResetArbitrationBit ; Will send Reset Arbitration
Bit command writing to offset 0xFF, no read portion
}
```

```
Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = RWORS_ReadWORptStartOffset0 ; Will executes
special read command without a repeated start from offset zero, read
number of bytes specified below
    BMCLengthToRead = 8
}

Packet=SMBus
{
    CommandCode=NVMeMiBasicManagement
    BMCmdCode_Offset = 2 ; Will start reading at offset 2 in the Status
area and continu into the Information area
    BMCLengthToRead = 10
}
```

2.1.6 Packet = DOECommand

This command initiates transmission of the DOE Command in the Host emulation mode using DOE Mailbox registers of the DUT. It is only available on Summit Z58, Z516, and M616 in Host Emulation mode. Z5 Exerciser Firmware will execute the sequence of the Configuration Writes and Reads required to transmit the Request Object and receive the Response Object.

Parameter	Values	Default	Comment
DOECapabilityAddress	0 – 0xFFF	0	Register offset to the start of the DUT's DOE Extended Capability structure.
UseDeviceId	(x:x:x) or value	(0:0:0)	Device ID to use for Configuration transaction during the command execution.
InterruptAddressHigh	0 – 0xFFFFFFFF	0	High portion of the address for MSI or MSI-X interrupt. If set to 0, 32-bit access is assumed.
InterruptAddressLow	0 – 0xFFFFFFFF	0	Low portion of the address for MSI or MSI-X interrupt, or 32-bit address for the interrupt. If both High and Low addresses are zero, the exerciser won't be expecting the interrupt and would poll for the Data Object Ready bit set instead.
DOEProtocol	DOE_Discovery, CMA_SPDM, CMA_Secured_SPDM or data object type	0	For DOE_Discovery, CMA_SPDM, CMA_Secured_SPDM sets both Data Object Type and Vendor ID in the DOE Data Object Header of the Request Object. If a numeric value is used, it is assigned to Data Object Type, and Vendor ID should be set separately.
DOEVendorID	0 – 0xFFFF	0	Can be used to specify the Vendor ID for the DOE Request Object when Data Object Type is set as a numeric value.
RequestObject	Array of Data Object DWORDS starting with Data Object DW 2	N/A	Allows to specify all the DWORDs of the Request Data Object (ones following the 2 DW header which has values set by the other parameters).
ResponseObjectSize	Auto or size	Auto	Size of the Response Object to receive. If Auto, the exerciser monitors the Data Object Ready bit to determine when the Object ends.
Discovery_Index	0 – 255	0	When the protocol is DOE Discovery allows to set the Index value in the Request Object.

Setting DOEProtocol to CMA_SPDM allows the user to generate SPDM request messages over DOE using SPDM-specific fields.

2.1.6.1 Generation of SPDM messages over Various Transport Potocols

The interface provides two ways to set values for SPDM message fields: using a generic layout and using a code-specific layout.

The generic layout contains fields for specifying the SPDM header: version, code, param1, param2, and for specifying the payload as a DWORD array for PCIe transports and a byte array for SMBus transport.

The list of available code-specific fields is generated after processing the request code value. For each SPDM message, the code must be set once before using the code-specific layout.
SPDM code-specific fields take one of such types of values:
- non-negative integer (for fixed-size fields up to 32 bits). Example: ENCRYPT_CAP = 1;
- array of bytes (for fixed-size fields over 32 bits and for variable-size fields). Example: VendPayload = (1 0x02);
- array of template names (for field structures and arrays of structures). Example: ReqAlgStruct = (Supported_DHE Supported_AEAD). See 2.10.1 for details.

The resulting message is displayed in Packet View after the script file is saved. Correct display of message layouts with fields of variable size, as well as with the conditions for the presence of fields, is provided if the user first defines the corresponding values in Config=SPDM.

The table below describes the common SPDM fields.

Parameter	Values	Default	Comment
SPDMVersion	0-0xFF	0	Version value
SPDMRequestCode	0-0xFF	0	RequestResponseCode value
SPDMParam1	0-0xFF	0	Param1 value
SPDMParam2	0-0xFF	0	Param2 value
SPDMPayload	DWORD/byte array	N/A	Specifies SPDM payload

Other fields are code-specific. SPDM fields with a fixed offset and size are zero by default.

Note: The set of SPDM cells created at the stage of processing the text of the generation script may differ from the one obtained as a result of decoding of the recorded trace due to differences between the options specified in Config=SPDM and those selected by the responder.

Example:

```
Packet=DOECommand
{
    DOECapabilityAddress=0x180
    UseDeviceId = (1:0:0)
    InterruptAddressLow = MSI_MESSAGE_ADDRESS
    DOEProtocol = CMA_SPDM
    ResponseObjectSize = 5

    SPDMVersion = 0x10
    SPDMRequestCode = GET_VERSION
}
Packet=DOECommand
{
    DOECapabilityAddress=0x180
```

```
    UseDeviceId = (1:0:0)
    InterruptAddressLow = MSI_MESSAGE_ADDRESS
    DOEProtocol = CMA_SPDM
    ResponseObjectSize = 7
    SPDMVersion = 0x12
    SPDMRequestCode = GET_CAPABILITIES
    CERT_CAP = 1
    CHAL_CAP = 0
    KEY_EX_CAP = 1
    DataTransferSize = 1024
    MaxSPDMmsgSize = 4096
}

Packet=DOECommand
{
    DOECapabilityAddress=0x180
    UseDeviceId = (1:0:0)
    InterruptAddressLow = MSI_MESSAGE_ADDRESS
    DOEProtocol = CMA_SPDM
    ResponseObjectSize = 5
    SPDMVersion = 0x12
    SPDMRequestCode = VENDOR_DEFINED_REQUEST
    StandardID = 0x02
    VendorIDLength = 0x02
    SPDMVendorID = (0x56 0x78)
    VendPayloadLength = 4
    VendPayload = (0x10 0x20 0x30 0x40)
```

2.1.6.2 Generation of Secured SPDM Messages

Secured SPDM messages are generated over the same transport protocols used by SPDM. The Secured SPDM fields become available after setting the DOEProtocol/MCTP_MSG_Type to Secured_SPDM.

The table below describes Secured SPDM fields.

Parameter	Values	Default	Comment
SecuredSPDM_KeyIndex	0-0xFFFFFFFF	N/A	References the SPDM_KeyIndex in Config=SPDM_Key used to define the Key. See 2.4.31.
SecuredSPDM_SessionId	0-0xFFFFFFFF	0	Bytes 0-3 in a secure message
SecuredSPDM_SequenceNumber	0-0xFFFF	Auto	Bytes 4-5 in a secure message
SecuredSPDM_Length	0-0xFFFF	Auto	Bytes 6-7 in a secure message
SecuredSPDM_AppDataLength	0-0xFFFF	Auto	Bytes 8-9 in a secure message
SecuredSPDM_RandomData	Byte array	N/A	Optional random data

Application data is a nested SPDM message.

NOTE: Before generating messages, you must specify the AEAD_SEL, RSP_MAC_CAP and RSP_ENCRYPT_CAP in Config=SPDM. See 2.4.30 for details on SPDM configuration.
Example:

```
Packet=DOECommand
{
          DOECapabilityAddress=0x180
          UseDeviceId = (1:0:0)
          InterruptAddressLow = 1
          DOEProtocol = CMA_Secured_SPDM
          ResponseObjectSize = 5

          SecuredSPDM_KeyIndex = 1
          SecuredSPDM_SessionId = 54321
          SPDMVersion = 0x12
          SPDMRequestCode = VENDOR_DEFINED_REQUEST
          StandardID = DMTF
          VendPayloadLength = 8
          VendPayload = (0 1 2 3 4 5 6 7)
      }
```

NOTE: Find sample scripts utilizing SPDM and Secured SPDM generation mechanisms in:
`...\Users\Public\Documents\LeCroy\PCIe Protocol Suite\Sample Files\Z5TrainerScripts\SPDM`

2.1.7 Packet = OrderedSet

This command initiates transmission of ordered set(s) on the bus. It can only be placed within the Raw Ltssm block and is available for M616, Z416, Z58 and Z516 Exercisers.

Parameter	Values	Default	Comment
SetType	TS1 TS2 FTS Pattern Idle EIEOS Skip		
RawData@<start>			Inserts raw data symbols at **<start>** byte position from the beginning of the ordered set. See **Packet = Raw** (see Packet = Raw page 75), description for possible raw data formats.
Count	1 – 65535	1	Repeats this packet the number of times specified.

Note: This command is supported by the PE Exerciser ML and EML Exercisers.

Example:

The following example sends 255 Fast Training Sequences.

```
Packet = OrderedSet {
      SetType = FTS
      Count = 255
}
```

See 2.16.3 RawLtssm Examples for more detailed examples.

2.1.7.1 SetType = TS1, TS2

Parameter	Values	Default	Comment
LinkNumber	0 – 255, PAD	PAD	Link Number within component
LaneNumber	0 – 31, PAD	PAD	Lane Number within Port
N_FTS	0 – 255	0	The number of fast training ordered sets required by the Receiver to obtain reliable bit and symbol lock.
TrainingControl	(X,X,X,X)	(0,0,0,0)	Training control bits. The order of the bits is as follows: **(HotReset, DisableLink, Loopback, DisableScrambling)**
Data Rate	Number	0	Data Rate field value. For example, value of 0x9E sets all the seed bits (2.5G, 5.0G, 8.0G and 16G) as well as the Speed Change bit.
Identifier	(X,X,X…)	D10.2 for TS1 and D5.2 for TS2 (Gen1/2)	Use the same format as in **Packet = Raw** (see Packet = Raw page 75), with exception of 10-bit codes.

Specific to Gen3, 4 and 5 TS1s/TS2s are the following equalization parameters:

Parameter	Values	Default	Comment
EqControl	Number, ID	ID	EqControl equalization parameter. Value of ID means that the parameter is not used and normal TS1/TS2 Identifier value is used
PreCursor	Number, ID	ID	PreCursor equalization parameter. Value of ID means that the parameter is not used and normal TS1/TS2 Identifier value is used
Cursor	Number, ID	ID	Cursor equalization parameter. Value of ID means that the parameter is not used and normal TS1/TS2 Identifier value is used
PostCursor	Number, ID	ID	PostCursor equalization parameter. Value of ID means that the parameter is not used and normal TS1/TS2 Identifier value is used
ManualParity	Yes/No	No	Only applicable to TS1 Ordered Set for speeds 8GT/s and higher. If set to No, the Parity bit will be calculated automatically. If set to Yes, allows to use the upper bit of the PostCursor parameter to create TS1 parity error.

In x2, x4, x8 or x16 configurations, the keys listed above apply to all lanes.

When you want to specify parameters for a particular lane, use the following format:

```
<key>@<lane_number> = <value>
```

Example 1:

The following example sends a TS1 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
TrainingControl bits are zeroes for all lanes.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
     LinkSpeed = 2_5
      N_FTS = 255
}
```

Example 2:

The following example sends a TS1 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber is 0 for all lanes.
LaneNumber are 3, 2, 1, 0 for lanes 0, 1, 2, 3, and PADs for all other lanes.
TrainingControl bits are zeroes for all lanes.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
     LinkSpeed = 2_5
      LinkNumber = 0
      LaneNumber@0 = 3
      LaneNumber@1 = 2
      LaneNumber@2 = 1
      LaneNumber@3 = 0
      N_FTS = 255
}
```

Example 3:

The following example sends a TS2 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
TrainingControl's **Disable Scrambling** bit is asserted on all lanes. All other **TrainingControl** bits are de-asserted.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
     LinkSpeed = 2_5
      N_FTS = 255
      TrainingControl = (0,0,0,1)
}
```

Example 4:

The following example sends a TS2 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
All **TrainingControl** bits are de-asserted.
Identifier symbols are (D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.1, D5.2) for lane 2.
Identifier symbols are (D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2) for all other lanes.
This would send a corrupted TS2 ordered set, since the **Identifier** is incorrect for lane 2.

```
Packet = OrderedSet {
    SetType = TS2
    LinkSpeed = 5_0
    N_FTS = 255
    Identifier@2 = (D5.2,D5.2,D5.2,D5.2,D5.2,D5.2,D5.2,D5.2,D5.1, D5.2)
}
```

2.1.7.2 SetType = Skip

Parameter	Values	Default	Comment
SkipCount	0 – 5	3	Number of SKIP symbols to send after COMMA

Example 1:

This example sends a Skip ordered set. Comma followed by 3 SKIP symbols would be sent on each lane.

```
Packet = OrderedSet {
      SetType = Skip
}
```

Example 2:

This example sends a Skip ordered set. Comma followed by 2 SKIP symbols would be sent on each lane.

```
Packet = OrderedSet {
      SetType = Skip
      SkipCount = 2
}
```

2.1.8 Packet = Raw

This command initiates transmission of raw data on the bus.

Parameter	Values	Default	Comment
RawData	(X,X,X…)		Specifies the array of bytes or 10-bit symbols to send.
Count	1 – 65535	1	Repeats packet specified number of times.

Note: This command is supported by Z416 only and can only be placed within RawLtssm block.

The elements of data can be specified in the following formats:

1) Symbols:

```
Packet = Raw
{
      RawData = ( K28.5, D21.5, K28.5, D10.2 )
}
```

2) Bytes in hexadecimal format with preceding K/D modifier:

```
Packet = Raw
{
      RawData = ( KBC, DB5, KBC, D4A )
}
```

2.1.9 Packet = <TemplateName>

This command initiates transmission of the packet specified by the **Template** command (see page 184). User can override packet fields according to the template.

Example 1:

This sequence issues three 32-bit Memory read requests. The address field of TLP header would accept the values 0, 64, and 128. Every other field in the TLP header would accept the value from the packet template.

```
Template = TLP {
      Name = "TestPacket"
      Type = MRd32
      RequesterID = (1:0:0)
      Length = 64
      Address = 0
}


Packet = "TestPacket"
{
}

Packet = "TestPacket"
{
      Address = 64
}

Packet = "TestPacket"
{
      Address = 128
}
```

2.1.10 Packet = CXL_LLCTRL

This command initiates transmission of LLCTRL flit on the bus. Fields that you can't fill manually are automatically filled by the bus engine. The bus engine takes other settings and current traffic into account.

The Packet = CXL_LLCTRL command is ignored when LLCTRLUserControl is off. This mode can be turned on in Generation Options by checking checkbox "CXL Allow User Controlled LLCTRL" on Link tab or using Config=CXL_Link command in a script.

NOTE: InterconnectVersion and LLR Wrap fields of LLCTRL Init.Param flit may be set using Config = CXL_Link command or in Generation Options.

Parameter	Values	Default	Comment
LLCTRLType	InitParam, LLCRDAck, RetryAck, RetryReq	0	Sets the CTL_FMT/LLCTRL (bits 5:7 of byte 0 / bits 0:3 of byte 4) and SubType (bits 4:7 of byte 4) fields.

Example 1:

Send one LLCTRL Init.Param flit.

```
Config=CXL_Link
{
    LLCTRLUserControl = On
}

Packet=CXL_LLCTRL
{
    LlctrlType=InitParam
}

Config=CXL_Link
{
    LLCTRLUserControl = Off
}
```

Example 2:

Send one LLCTRL Retry.Ack flit.

```
Config=CXL_Link
{
    LLCTRLUserControl = On
}

Packet=CXL_LLCTRL
{
      LlctrlType =RetryAck
}

Config=CXL_Link
{
    LLCTRLUserControl = Off
}
```

Example 3:

Send one LLCTRL Retry.Req flit.

```
Config=CXL_Link
{
    LLCTRLUserControl = On
}

Packet=CXL_LLCTRL
{
      LlctrlType =RetryReq
}
Config=CXL_Link
{
    LLCTRLUserControl = Off
}
```

2.1.10.1 LLCTRLType = LLCRDAck

When LLCTRLType = LLCRDAck, additional parameters are available, as shown in the table.

Parameter	Values	Default	Comment
LLCRDFullAck	0 – 255	0	Full_Ack = {Acknowledge[7:4], Ak, Acknowledge[2:0]}, where the Ak bit is from the flit header.
LLCRDCacheReqCredits	0 – 255	0	Defines the number* of cache request credits to be returned. See NOTE below.
LLCRDCacheRspCredits	0 – 255	0	Defines the number* of cache response credits to be returned. See **NOTE** below.
LLCRDCacheDataCredits	0 – 255	0	Defines the number* of cache data credits to be returned. See **NOTE** below
LLCRDMemReqRspCredits	0 – 255	0	Defines the number* of mem request credits to be returned. See **NOTE** below
LLCRDMemDataCredits	0 – 255	0	Defines the number* of mem data credits to be returned. See **NOTE** below

NOTE: To match the CXL specification requirement that the credit return value must be a power of 2, credit return values may be split between several LLCRD.Ack flits.

Example:

Send LLCTRL LLCRD.Ack flit with Full_Ack and credits return values. **Note** that credit return values are not powers of two. This means that this credit return values will be split between several LLCTRL LLCRD.Ack flits.

```
Config=CXL_Link
{
    LLCTRLUserControl = On
}

Packet=CXL_LLCTRL
{
    LlctrlType=LLCRDAck
    LLCRDFullAck = 45
    LLCRDCacheReqCredits = 10
    LLCRDCacheRspCredits = 10
    LLCRDCacheDataCredits = 10
    LLCRDMemReqRspCredits = 10
    LLCRDMemDataCredits = 10
}
```

```
Config=CXL_Link
    LLCTRLUserControl = Off
}
```

2.1.10.2 **LLCTRLType = RetryReq**

When LLCTRLType = RetryReq, additional parameters are available, as shown in the table.

Parameter	Values	Default	Comment
RetryEseq	0 – 255	0	Requester's Retry Sequence Number (Eseq)
RetryNumRetry	0 – 31	0	NUM_RETRY
RetryNumPhyReinit	0 – 31	0	NUM_PHY_REINIT

Example:

```
Config=CXL_Link
{
    LLCTRLUserControl = On
}

Packet=CXL_LLCTRL
{
    LlctrlType=RetryReq
    RetryEseq = 1
    RetryNumRetry = 2
    RetryNumPhyReinit = 3
}
Config=CXL_Link
    LLCTRLUserControl = Off
}
```

2.2 Idle Command

Note: OBSOLETE.

Please use the Wait command below.

2.3 Link Command

Most of these commands are controls to the Link Training and Status State Machine (also known as the LTSSM). These commands are issued to the LTSSM to steer it to a particular state. This is not a means to force the Link state to a particular value. For instance, if the Script contains the **Link = L0** command, it is a request to bring the link to the L0 state. The LTSSM is responsible for managing all of the link training and all of the intermediate link states to accomplish this.

Note: about the relationship of Link=L0 and Link=*speed* (2_5, 5_0, 8_0, 16_0, 32_0) commands.
A speed switch command (Link=16_0 etc.) makes sense to be executed from a state where the link to the DUT is already in L0 state at certain (other) speed. When the command completes, if successful, the link is expected to be in L0 state at the target speed. So, in general, Link=*speed* command shouldn't be followed by Link=L0 command.

2.3.1 Link = L0

Transitions the link into the L0 state.

2.3.2 Link = L0s

Transitions the link into the L0s (low power) state. Applies only in L0 state.

2.3.3 Link = L1

Transitions the link into the L1 (low power) state. Applies only in L0 state.
The transition to L1 state will happen under the protocol selected (ASPM or PCIPM) in the generation options file.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.4 Link = L23

Z3/Z4 initiates protocol to enter L2 (or L3 if Vaux is not present on the bus) low power state.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.5 Link = Loopback

Z3/Z4 Exerciser will initiate entry to Loopback state.

Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.6 Link = Disabled

Tells the LTSSM to move into the Disabled State. To get to this state, the LTSSM must either be in the Configuration State or the Recovery State. If the link is currently in the Detect state, and the **Link=Disabled** command is issued, it goes to Configuration first and then goes directly to Disabled. Once in the Disabled state, the LTSSM sends 16 TS1's with the Disable Link modifier bit set, followed by an electrical Idle ordered set, followed by electrical idle. To exit the Disabled state, simply set **Link=Detect** or **Link=L0**.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.7 Link = HotReset

Tells the LTSSM to move into the HotReset State. To get to this state, the LTSSM must first be in the Recovery state. Once in the HotReset State, the LTSSM sends TS1 ordered sets with the HotReset modifier bit set. The LTSSM then goes to the Detect state automatically after 2 ms.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.8 Link = Recovery

Transitions the link into the Recovery state. Applies only in L0, L0S, or L1 States.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.9 Link = Detect

Tells the Summit Exerciser™ to immediately bring the Link down. In this state, the LTSSM drives all of the PCI Express lanes to electrical idle. Before the lanes go to electrical idle, a single electrical idle ordered set is transmitted. Applies while in any state.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.10 Link = PERST_Assert

In host mode, the Z3/Z4/Z5 Exerciser will drive a low in the bus for PERST#.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.11 Link = PERST_Deassert

In host mode, the Z3/Z4/Z5 Exerciser will release PERST# and the bus will drive PERST# high.
Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.12 Link = PERST

In host mode, the Z3/Z4/Z5 Exerciser will drive a low in the bus for PERST#, then wait for the time specified by the Duration parameter (with a high timing precision) and then release PERST# and the bus will drive PERST# high.
Note: Command supported by Z3/Z4/Z5 Exercisers. Use when high precision timing of the PERST assertion is needed with respect to assert / wait for time / deassert that can be as much as 10% off.

Example:

```
Link=PERST
        {
          Duration = 1000000 ; in nanoseconds
        }
```

2.3.13 Link = Loopback_WComplRx

Z3/Z4/Z5 Exerciser will initiate entry to Loopback state, in addition asserting the Compliance Receive bit.

Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.14 Link = ComplianceReceive

Z3/Z4/Z5 Exerciser will enter Detect state and will assert the Compliance Receive bit during the following training.

Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.15 Link = ClearLoopback

Z3/Z4/Z5 Exerciser will exit from the Loopback state if it was previously initiated.

Note: Command supported by Z3/Z4/Z5 Exercisers.

2.3.16 Link = 2_5

Initiates speed switch to 2.5 GT/s data rate. The system attempts to switch if the link is in the L0 state at 5.0 GT/s data rate.

2.3.17 Link = 5_0

Initiates speed switch to 5.0 GT/s data rate. The system attempts to switch if the link is in the L0 state at 2.5 GT/s data rate, the Summit Z3-16/Z416/Z58/Z516/M616 Exerciser and the DUT advertise the 5.0 GT/s data rate.

2.3.18 Link = 8_0

Initiates speed switch to 8.0 GT/s data rate. The system attempts to switch if the link is in the L0 state at 2.5 GT/s or 5.0 GT/s data rate, and the Summit Z3-16/Z416/Z58/Z516/M616 Exerciser and the DUT advertise the 8.0 GT/s data rate. 8.0GT/s data rate is available only on the Summit Z3-16/Z416/Z58/Z516/M616 Exercisers.
Note: This command applies to the Summit Z3-16/Z416/Z58/Z516/M616 Exercisers.

2.3.19 Link = 16_0

Initiates speed switch to 16.0 GT/s data rate. The system attempts to switch if the link is in the L0 state at 2.5 GT/s or 5.0 GT/s or 8.0 GT/s data rate, and the Summit Z416/Z58/Z516/M616 Exerciser and the DUT advertise the 16.0 GT/s data rate. 16.0GT/s data rate is available on the Summit Z416, Z58 and Z516 Exercisers.

Note: This command only applies to the Summit Z416, Z58, Z516 and M616 Exercisers.

2.3.20 Link = 32_0

Initiates speed switch to 32.0 GT/s data rate. The system attempts to switch if the link is in the L0 state at 2.5 GT/s or 5.0 GT/s or 8.0 GT/s or 16.0 GT/s data rate. The Summit Z58/Z516/M616 Exerciser, the DUT advertise the 32.0 GT/s data rate. 32.0GT/s data rate is available on the Summit Z58, Z516, and M616 Exercisers.

Note: This command only applies to the Summit Z58, Z516, and M616 Exercisers.

2.3.21 Link = x1, x2, x4, x8, x16

Transitions the width of the link to the one specified by the command. Applies only in L0 state.

Note: This command applies to the Summit Z3-16/Z416/Z58/Z516/M616 Exerciser.

Example:

To execute LTSSM Arc Tests, a script is available at:

```
      C:\Users\Public\Documents\LeCroy\PCIe Protocol
Suite\Automation\python\ArcTestExample.py
```

.

2.3.22 Link = Power_ON, Link = Power_OFF

In host mode, allows to operate the DUT Power on the Host Platform, switching it On or Off.

Important: The command only takes effect when the DUT Power switch on the Host Platform is in the OFF position.

Note: This command is currently available on the Summit Z3-16, Summit Z58, Summit Z516 and Summit M616.
This functionality is also available on the Summit Z416 Exerciser when used with the PCIe-Gen4 Test Platform (Rev C and above).

2.3.23 Link = RedoEQ

Directs the link to redo equalization at the data rate indicated by the speed parameter. As a host, uses the Initiate parameter to decide whether to direct the link to recovery EQ and perform equalization (Initiate = yes), or when request equalization before eventually redoing equalization (when Initiate = no). As a device, the exerciser will always go to recovery and then request equalization. This command will perform any necessary speed changes prior to requesting or performing equalization at the desired data rate.

When AtTargetSpeed=Yes, equalization is performed at the speed indicated by Speed = 8GTs/16GTs/32GTs without first going to Gen1 or Gen2.

Example:

```
Link = RedoEQ

{

Initiate = Yes/No

Speed = 8GTs/16GTs/32GTs

AtTargetSpeed = Yes/No

}
```

2.3.24 Link= L0pRequest

This Command is available only if **PCIeFlitMode=True**. See 2.17 PCIeFlitMode Command.

Parameter	Values	Default	Exerciser Default	Comment
L0pRequestPriority	0-1	Unchanged	0	L0p Priority bit value
L0pRequestLinkWidth	x1, x2, x4, x8, x16	Unchanged	x16	L0p link width

Example:

```
Link=L0pRequest
{
    L0pRequestPriority = 1
    L0pRequestLinkWidth=x4
}
```

2.4 Config Command

This command configures the Summit Exerciser™.

2.4.1 Config = General

This command should precede any statement in a Summit Exerciser script file. There should be only one **Config = General** command in a Summit Exerciser script file. All **Config = General** commands from included files (see page 187) are ignored.

Parameter	Values	Default	Comment
AutoDetect	Yes, No	No	Automatically detect link parameters.
LinkWidth	1,4,8,16	4	Ignored if AutoDetect is set.
DirectionRx	Upstream, Downstream	U	
DisableScrambleTx	Yes, No	No	Ignored if AutoDetect is set.
DisableDescrambleRx	Yes, No	No	Ignored if AutoDetect is set.
ReverseLanes	Yes, No	No	Reverses Lanes (Tx/Rx)
FollowLaneReversal	Yes, No	No	Follows lane reversal initiated by the DUT

Parameter	Values	Default	Comment
InvertPolarityTx	(X,X,X,X,…)		The array of 1/0 elements. The size of the array should match the link width.
InvertPolarityRx	(X,X,X,X,…)		The array of 1/0 elements. The size of the array should match the link width. Ignored if AutoDetect is set.
BaseSpec10	Yes, No	No	
SkewTx	(X,X,X,X,…)		The array of integer elements. The size of the array should match the link width. Measured in symbols, valid values are from 0 to 15.
TrainerReset	Yes, No	No	When set, resets Summit Exerciser before script execution.
UseExtRefClock	Yes	Yes, No	No: Summit Z3-16/Z416/Z58/Z516/M616 uses an on board clock (no SSC). Yes: Summit Z3-16/Z416/Z58/Z516/M616 uses the reference clock from the bus.
PHY Parameters for Z3-16 (see Note)			
EmphasisTx	Preset Value	Preset_7	Select the Preset value, which sets the Preshoot and the De-emphasis values for the transmitter.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AdvertisedTx	Preset Value	Preset_7	Select the Advertised value, which sets the Preshoot and De-emphasis broadcasted values from the transmitter to the receiver as PCIe data traffic.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
DCGainRx	0_dB, 3_dB, 6_dB, 9_dB, 12_dB	0_dB	Receiver DC Gain in dB.
CTLEGainRx	0_dB, 1_dB, 2_dB,	15_dB	Receiver Continuous Time Linear Equalizer (CTLE) value in dB.

Parameter	Values	Default	Comment
	3_dB, 4_dB, 5_dB 6_dB, 7_dB, 8_dB, 9_dB, 10_dB, 11_dB 12_dB, 13_dB, 14_dB, 15_dB,		
AdvertisedRx	Preset Value	P0	Select the Advertised value, which sets the Preshoot and De-emphasis broadcasted values from the receiver to the transmitter.
	P0		-6 dB
	P1		-7 dB
	P2		-8 dB
	P3		-9 dB
	P4		-10 dB
	P5		-11 dB
	P6		-12 dB
DeEmphasis_Gen2	DeEmphasis Value on all lanes	DeEmphasis_1	Sets the DeEmphasis value on all lanes for Gen2 traffic
	DeEmphasis_0		-6.0 dB
	DeEmphasis_1		-3.5 dB
PHY Parameters for Z416			
AppliedTx8G	Preset Value	Preset_7	Select the Preset value for 8GT/s, which sets the Preshoot and the De-emphasis values for the transmitter
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AdvertisedTx8G	Preset Value	Preset_7	Select the Advertised value for 8GT/s, which sets the Preshoot and De-emphasis broadcasted values

Parameter	Values	Default	Comment
			from the transmitter to the receiver as PCIe data traffic.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AppliedTx16G	Preset Value	Preset_7	Select the Preset value for 16GT/s, which sets the Preshoot and the De-emphasis values for the transmitter.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AdvertisedTx16G	Preset Value	Preset_7	Select the Advertised value for 16GT/s, which sets the Preshoot and De-emphasis broadcasted values from the transmitter to the receiver as PCIe data traffic.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
Additional PHY Parameters for Z58/Z516/M616			
AppliedTx32G	Preset Value	Preset_5	Select the Preset value for 32GT/s, which sets the Preshoot and the De-emphasis values for the transmitter.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB

Parameter	Values	Default	Comment
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AdvertisedTx32G	Preset Value	Preset_5	Select the Advertised value for 32GT/s, which sets the Preshoot and De-emphasis broadcasted values from the transmitter to the receiver as PCIe data traffic.
	Preset_0		Preshoot = 0 dB, De-emphasis = -6 dB
	Preset_1		Preshoot = 0 dB, De-emphasis = -3.5 dB
	Preset_2		Preshoot = 0 dB, De-emphasis = -4.5 dB
	Preset_3		Preshoot = 0 dB, De-emphasis = -2.5 dB
	Preset_4		Preshoot = 0 dB, De-emphasis = 0 dB
	Preset_5		Preshoot = 2 dB, De-emphasis = 0 dB
	Preset_6		Preshoot = 2.5 dB, De-emphasis = 0dB
	Preset_7		Preshoot = 3.5 dB, De-emphasis = -6 dB
	Preset_8		Preshoot = 3.5 dB, De-emphasis = -3 dB
	Preset_9		Preshoot = 3.5 dB, De-emphasis = 0 dB
	Preset_10		Preshoot = 0 dB, De-emphasis = Maximum
AppliedTx64G	Preset Value	Preset_5	Select the Preset value for 64GT/s, which sets the Preshoot and the De-emphasis values for the transmitter.
	Preset_0		Preshoot = 0 dB Preshoot2, 0 dB Preshoot1, De-emphasis = 0 dB
	Preset_1		Preshoot = 0 dB Preshoot 2, 1.6 Preshoot1, De-emphasis = 0 dB
	Preset_2		Preshoot = 0 dB Preshoot 2, 3.5 Preshoot1, De-emphasis = 0 dB
	Preset_3		Preshoot = 0 dB Preshoot 2, 0 Preshoot1, De-emphasis = -1.6 dB
	Preset_4		Preshoot = 0 dB Preshoot 2, 0 Preshoot1, De-emphasis = -3.5 dB
	Preset_5		Preshoot = -1.3 dB Preshoot 2, 4.7 Preshoot1, De-emphasis = 0 dB
	Preset_6		Preshoot = -1.6 dB Preshoot 2, 3.5 Preshoot1, De-emphasis = -3.5 dB
	Preset_7		Preshoot = -2.9 dB Preshoot 2, 4.7 Preshoot1, De-emphasis = 0 dB
	Preset_8		Preshoot = -3.5 dB Preshoot 2, 6.0 Preshoot1, De-emphasis = 0 dB
	Preset_9		Preshoot = -4.4 dB Preshoot 2, 6.9 Preshoot1, De-emphasis = -1.6 dB
	Preset_10		Preshoot = 0 dB Preshoot 2, 0 Preshoot1, De-emphasis = Maximum
AdvertisedTx64G	Preset Value	Preset_5	Select the Advertised value for 64GT/s, which sets the Preshoot and De-emphasis broadcasted values from the transmitter to the receiver as PCIe data traffic.
	Preset_0		Preshoot = 0 dB Preshoot2, 0 dB Preshoot1,

Parameter	Values	Default	Comment
			De-emphasis = 0 dB
	Preset_1		Preshoot = 0 dB Preshoot2, 1.6 dB Preshoot1, De-emphasis = 0 dB
	Preset_2		Preshoot = 0 dB Preshoot2, 3.5 dB Preshoot1, De-emphasis = 0 dB
	Preset_3		Preshoot = 0 dB Preshoot2, 0 dB Preshoot1, De-emphasis = -1.6 dB
	Preset_4		Preshoot = 0 dB Preshoot2, 0 dB Preshoot1, De-emphasis = -3.5 dB
	Preset_5		Preshoot = -1.3 dB Preshoot2, 4.7 dB Preshoot1, De-emphasis = 0 dB
	Preset_6		Preshoot = -1.6 dB Preshoot2, 3.5 dB Preshoot1, De-emphasis = -3.5 dB
	Preset_7		Preshoot = -2.9 dB Preshoot2, 4.7 dB Preshoot1, De-emphasis = 0 dB
	Preset_8		Preshoot = -3.5 dB Preshoot2, 0 dB Preshoot1, De-emphasis = 6.0 dB
	Preset_9		Preshoot = -4.4 dB Preshoot2, 6.9 dB Preshoot1, De-emphasis = -1.6 dB
	Preset_10		Preshoot = 0 dB Preshoot2, 0 dB Preshoot1, De-emphasis = Maximum

Note: For a more detailed description of PHY Parameters (Emphasis Tx, Advertised Tx, DCGain Rx, CTLEGain Rx and Advertised Rx) consult the PCI Express Base Specification, Rev 3.0.

Example 1:

The following example configures Summit Exerciser to generate traffic on an x4 link (**LinkWidth = 4**) as a host emulator (**DirectionRx = Upstream**) and invert polarity on the first two lanes on incoming traffic (**InvertPolarityRx = (1,1,0,0)**).

The Summit Exerciser is reset before script execution (**TrainerReset = Yes**).

All options that are not specified (**DisableScrambleTx**, **DisableDescrambleRx**, **ReverseLanes**, **InvertPolarityTx**, **BaseSpec10**, **SkewTx**, and **UseExtRefClock**) are taken from the Generation Options dialog.

```
Config = General
{
      LinkWidth = 4
            DirectionRx = Upstream
      InvertPolarityRx = (1,1,0,0)
      TrainerReset = Yes
}
```

Example 2:

The following example configures Summit Exerciser to generate traffic on an x8 link (**LinkWidth = 8**) as a device emulator (**DirectionRx = Downstream**).

Outgoing lanes are reversed (**ReverseLanes = Yes**).
Polarity on the last four outgoing lanes on outgoing traffic is inverted:

(**InvertPolarityTx = (0,0,0,0,1,1,1,1)**).
Lanes 0 and 4 have a skew value of 1 symbol time.
Summit Exerciser is reset before script execution (**TrainerReset = Yes**).

```
Config = General
{
      LinkWidth = 8
      DirectionRx = Downstream
      SkewTx = (1,0,0,0,1,0,0,0)
      InvertPolarityTx = ( 0,0,0,0,1,1,1,1 )
      ReverseLanes = Yes
      TrainerReset = Yes
}
```

2.4.1.1 Equalization Settings

The Equalization Settings can be accessed from PCIe Protocol Analysis GUI through
for Z3-16: Generation Options-> Phy Parameters Tab-> EQSettings,
for Z416: Generation Options-> Phy Parameters Tab-> 8.0 GT/s Tab and 16.0 GT/s Tab,
for M616/Z516/Z58: Generation Options-> Phy Parameters Tab-> 8.0 GT/s Tab, 16.0 GT/s Tab and 32.0 GT/s Tab.
Any parameter that is set from Phy Parameters can be overridden by a generation script.
In order to override the EQ Settings from the GUI, you can create a Generation Script containing Config=General and save it.

Parameters that can be overridden include:

Z3-16:

EmphasisTx,
AdvertisedTx,
DCGainRx,
CTLEGainRx,
AdvertisedRx,
DeEmphasis_Gen2

Z416:

AppliedTx8G,
AdvertisedTx8G,
AppliedTx16G,
AdvertisedTx16G

M616/Z516/Z58:

AppliedTx8G,
AdvertisedTx8G,
AppliedTx16G,
AdvertisedTx16G
AppliedTx32G,
AdvertisedTx32G
AppliedTx64G,
AdvertisedTx64G

The Generation Script takes priority over the parameters in the GUI. For Parameters that are not specified in the Generation script, their values are obtained from the GenOptions GUI.

Example 1:

```
Config=General
{
      EmphasisTx = Preset_3 /* Sets Emphasis Tx to Preset 3 for all the
      lanes.*/
      AdvertisedTx = Preset_3 /* Sets Advertised Tx to Preset 3 for all the
                                 lanes.*/
      DCGainRx = 3_dB /* Sets DCGainRx to 3db for all the lanes.*/
      CTLEGainRx = 3_dB /* Sets CTLEGainRx to 3db for all the lanes.*/
      AdvertisedRx = P3 /* Sets AdvertisedRx to P3 for all the lanes.*/
      DeEmphasis_Gen2 = DeEmphasis_0 /* Sets De_emphasis to de-emphasis 0 for
      all
                                        the lanes.*/
}
```

The Equalization parameter can be set for selected lanes by using the following syntax "ParameterName""@""LaneNumber"

Example 2:

```
Config=General
{
      EmphasisTx@2 = Preset_3 EmphasisTx@3 = Preset_3
           /* EmphasisTx is set to Preset 3 on Lanes 2 and 3,
             All other lanes derive their values from the Generation Options
GUI */
}
```

2.4.2 Config = Link

Parameter	Values	Default	Comment
FTSCount	0 – 255	255	Number of FTS ordered sets required (as sent in TS)
ExtendedSynch	Yes, No	Yes	When set, forces the transmission of 4096 FTS ordered sets.
SkipTimer	On, Off	On	Used to turn On or Off automatic SKP generation.
SkipTimeG12Symbols	Count in symbols, 1180:1538	1360	Defines the Skip Timer for 2.5 and 5.0 GT/s, expressed in symbols
SkipTimeG34Blocks	Count in blocks, 1 – 511	370	Defines the Skip Timer for 8.0 and 16.0 GT/s, expressed in blocks
SkipTimeG6Blocks	Count in blocks, 1 – 1023	740	Defines the Skip Timer for 8.0 and 16.0 GT/s, expressed in blocks
DropLanes	XXX…X	0x0000	Hexadecimal, decimal or binary value. The value is converted to a binary that matches the link width. Bit zero as lane 0 to bit 15 as lane 15, 0 for normal lane operation, 1 to drop this lane.
PhyProgrammableEqPhase23Timer	0 – 16777212	2000000	Time in ns to spend in recovery equalization phase 2 as a device, or recovery equalization phase 3 as a host. This only applies to Z4, Z58, Z516 and M616 exercisers.

Example:

This example configures the number of Fast Training Sequences to send when transitioning from L0s state (see Page 81).

This command also configures the periodic timer for SKIP Ordered Sets in 2.5 and 5.0 GT/s - sent every 1239 symbols. This command also configures the periodic timer for SKIP Ordered Sets in 8.0 and 16.0 GT/s - sent every 373 blocks.

```
Config = Link
       {
         SkipTimeG12Symbols = 1239
         SkipTimeG34Blocks  = 373
```

```
  FTSCount = 255
}
```

Examples: Drop Lanes

```
; Hex
Config=Link
{
      DropLanes = 0x100  ; Drop lane 8, this should cause the link
to retrain to x4
}

; Binary
Config=Link
{
      DropLanes = 0b1000 ; Drop lane 3, this should cause the link
to retrain to x2
}

; Decimal
Config=Link
{
      DropLanes = 2  ; Drop lane 1, this should cause the link to
retrain to x1
}
```

2.4.3 Config = FCTx

This command allows you to specify the policy for TLP transmission for received Flow Control DLLP packets.

Parameter	Values	Default	Comment
CareForFC	Yes, No	Yes	When not set, the TLP packets are sent without regard for how many credits are available.

Example:

In this example, Flow Control checking is turned off for outgoing TLP packets.
The TLP packets that are declared after this **Config = FCTx** command are sent without checking for available FC credits.

```
Config = FCTx {
CareForFC = No
}

      Packet = TLP {
TLPType = CfgRd0
Length = 1
Register = 0
Count = 10000
 }
```

2.4.4 Config = FCRx

This command configures automatic **UpdateFC** DLLP generation.

Parameter	Values	Default	Comment
Timer	In ns (rounded to nearest 8), Off	4200	Periodic timer that controls the sending of **UpdateFC** DLLP packets
PH	0 – 255	1	Posted Request Headers
NPH	0 – 255	1	Non-Posted Request Headers
CplH	0 – 255	1	Completion Headers
PD	0 – 4095	1024	Posted Request Data Payload
NPD	0 – 4095	1	Non-Posted Request Data Payload
CplD	0 – 4095	1024	Completion Data Payload
StallFC	(PH, PD, NPH, NPD, CPLH, CPLD)	0	Instructs Z3/Z4 Exerciser to stall updates of Flow control credits of specified type until cleared. Any combination of comma- or space-separated six credit types can be specified. Set as 0 for normal operation.
EnableDLFeatureExchange	Yes, No	No	Enables/Disables Data Link Feature Exchange handshake on exerciser's side.

The following examples outline several ways of using the StallFC parameter.

Example 1:

In this example, the timer for sending Update FC DLLP packets is specified.
Also, the initial number of FC credits for headers to advertise is specified.
The default value is used for data credits.

```
Config = FCRx {
Timer = 4000    ; Send UpdateFC DLLP packets every 4000 ns
PH = 1          ; 1 credit for Posted Request Headers
NPH = 2         ; 2 credits for Non-Posted Request Headers
CplH = 0        ; Infinite number of credits for Completion Headers
}
```

Example 2:

"Eat up" 2 Posted header credits

```
Config=FCRx
{
    StallFC = ( PH )
}

wait=TLP
```

```
{
    TLPType = MWr32
    Count = 2
}

Config=FCRx
{
    StallFC = 0 ; This will remove the stalling for Flow Control

}
```

Example 3:

"Eat up" 5 Posted or Non-posted header credits

```
Config=FCRx
      {
          StallFC = ( PH NPH )
      }

      wait=MultiTLP
      {
          MultiTLPTypes = ( MWr32, MRd32 )
          Count = 5
      }

Config=FCRx
      {
          StallFC = 0 ; This will remove the stalling for Flow Control

      }
```

Example 4:

"Eat up" some amount of Posted or Non-posted and then just posted header credits

```
Config=FCRx
      {
          StallFC = ( PH NPH )
      }

      wait=10000

Config=FCRx
      {
          StallFC = ( PH )
      }

      wait=10000

Config=FCRx
      {
          StallFC = 0 ; This will remove the stalling for Flow Control

      }
```

2.4.5 Config = FCRx

This command configures automatic UpdateFC DLLP generation for Flit and CXL 3.0 modes.

Parameter	Values	Default	Comment
NumberVCs	1, 2	1	Allows to specifify the number of VCs available for Mapping_TC_VC0
Mapping_TC_VC0	(X,X,X,X,...)	(1, 0, 0, 0, 0, 0, 0, 0)	The array of 1/0 elements. The size of the array is 8 values that corresponds to TC0-TC7.
Timer	In ns (rounded to nearest 8), Off	8400	Periodic timer that controls the sending of UpdateFC DLLP packets
EnableDLFeatureExchange	Yes, No	No	Enables/Disables Data Link Feature Exchange handshake on exerciser's side.
MergeCredits	Yes, No	No	
Dedicated_VC0_PH	0-127	1	Dedicated Posted Request Headers
Dedicated_VC0_NPH	0-127	1	Dedicated Non-Posted Request Headers
Dedicated_VC0_CplH	0-127	1	Dedicated Completion Headers
Dedicated_VC0_PD	0-2047	256	Dedicated Posted Request Data Payload
Shared_VC0_NPD	0-2047	1	Shared Non-Posted Request Data Payload
Shared_VC0_CplD	0-2047	256	Shared Completion Data Payload
Shared_ScPH	1, 4, 16	1	Shared Data Scale factor
Shared_ScNPH	1, 4, 16	1	Shared Data Scale factor
Shared_ScCplH	1, 4, 16	1	Shared Data Scale factor
Shared_ScPD	1, 4, 16	1	Shared Data Scale factor
Shared_ScNPD	1, 4, 16	1	Shared Data Scale factor
Shared_ScCplD	1, 4, 16	1	Shared Data Scale factor
StallFC	(Dedicated_VC0_PH, Dedicated_VC0_PD, Dedicated_VC0_NPH, Dedicated_VC0_NPD, Dedicated_VC0_CPLH, Dedicated_VC0_CPLD)	0	Instructs M616 Exerciser to stall updates of Flow control credits of specified type until cleared. Any combination of comma- or space-separated six credit types can be specified. Set as 0 for normal operation.

Example:

```
PCIeFlitMode=True
Config = FCRxGen6
{
Timer = 8600 ; Send UpdateFC DLLP packets every 8600 ns
Dedicated_VC0_PH = 1 ; 1 credit for Dedicated Posted Request Headers
Dedicated_VC0_NPH = 2 ; 2 credits for Dedicated Non-Posted Request Headers
Dedicated_VC0_CplH = 0 ; Infinite number of credits for Dedicated Completion
Headers
Shared_VC0_PH = 3 ; 3 credit for Shared Posted Request Headers
Shared_VC0_NPH = 120 ; 120 credits for Shared Non-Posted Request Headers
Shared_VC0_CplH = 50 ; 50 credits for Shared Completion Headers
}
```

2.4.6 Config = TLP

This command facilitates data integrity control.

Parameter	Values	Default	Comment
AutoSeqNumber	Yes, No	Yes	If set to 0, overrides automatic generation of the TLP sequence number and uses the user-defined value as set in the **Packet = TLP** command (see Page 4).
AutoLCRC	Yes, No	Yes	If set to 0, overrides automatic generation of the TLP LCRC and uses the user-defined value as set in the **Packet = TLP** command (see Page 4).
AutoECRC	Yes, No	Yes	If set to 0, overrides automatic generation of the TLP ECRC by logic and uses the user-defined or software calculated value as set in the Packet = TLP command (see Page 3)
ReplayTimer	In ns (rounded to nearest 8), Off	4200	Timeout in the TLP transmitter path that counts time since the latest **Ack** or **Nak** DLLP was received. If set, automatically retransmits TLP packets that were **Nak'**ed or on replay timer expiration.
AutoRetrain	Yes, No	Yes	If set, enables automatic retraining of the link in case the number of retransmitted TLPs is 4. Applicable only when the **ReplayTimer** is not turned off.
TagGeneration	Manual, Auto5Bit, Auto8Bit, Auto10Bit	Manual	Tag generation policy for posted TLP packets: Manual: Tags are taken from the script. Auto5Bit: Automatically generate Tag using lower 5-bits of Tag field. Zero out higher 3 bits. Auto8Bit: Automatically generate Tag using 8-bits of Tag field. Auto10Bit: Automatically generate Tag using 10-bit Tag. (Available on Z416 only, Z3 will default to 8-bit). .

Example 1:

This example shows how to turn off automatic PSN and LCRC generation for outgoing TLP packets. The **ReplayTimer**, **AutoRetrain**, and **TagGeneration** parameters are omitted so the default values are used.

```
Config = TLP {
      AutoSeqNumber = No
```

```
        AutoLCRC = No
    }
```

Example 2:

; By default automatic ECRC calculation is on. Whenever TD is set, software would calculate ECRC for a packet ; ; in the script as a placeholder, and then logic will recalculate correct ECRC when transmitting the packet.

```
Packet=TLP
{
    TLPType=MRd32
    TD = 1 ; need to set to enable ECRC generation
    Length = 2
    LastDwBe = 15
    FirstDwBe = 15
}

Config=TLP
{
    AutoECRC = No; Turn off automatic ECRC calculation by logic
}

Packet=TLP
{
    TLPType=MRd32
    TD = 1
    Length = 2
    LastDwBe = 15
    FirstDwBe = 15
    ECRC = 0xAB001122 ; specify whong ECRC
}

Config=TLP
{
    AutoECRC = Yes ; Turn automatic ECRC calculation by logic back on
}

Packet=TLP
{
    TLPType=MRd32
    TD = 1 ; packet will be transmitted with correct ECRC again
    Length = 2
    LastDwBe = 15
    FirstDwBe = 15
}
```

2.4.7 Config = AckNak

Parameter	Values	Default	Comment
AckNak	Auto, Ack, Nak, Disable	Auto	**Auto**: Automatic Ack/Nak **Ack**: Always Ack **Nak**: Always Nak

Parameter	Values	Default	Comment
	NakSeveral TimeOutSeveral NakSeqNumber TimeOutSeqNumber ModifyAck		**Disable**: Disable automatic Ack/Nak DLLP generation. The Summit Z3-16/Z416/Z516/Z58/M616 Link layer can execute an action a number of times specified by the **NakSeveral, TimeOutSeveral, NakSeqNumber**, and **TimeOutSeqNumber** settings. After executing the action the specified number of times, the Link layer goes back to automatic Ack/Nak DLLP generation. **NakSeveral**: Sends Nak for a number of incoming TLPs defined by the **ActionCount** parameter. **TimeOutSeveral**: Does not send Ack or Nak for a number of incoming TLPs defined by the **ActionCount** parameter. **NakSeqNumber**: Sends Nak for the next incoming TLP with a sequence number defined by the **SeqNumberForAction** parameter. The number of times to send the Nak is defined by the **ActionCount** parameter. **TimeOutSeqNumber**: Does not send Ack or Nak for the next incoming TLP with a sequence number defined by the **SeqNumberForAction** parameter. The number of times not to send is defined by the **ActionCount** parameter. Also, for Summit Z416/Z516/Z58/M616, **ModifyAck**: provides an additional way to set up sending a modified Acknowledge DLLP in response to next incoming TLP(s). Has several additional parameters (below) aside of the **ActionCount** parameter. Allows to modify the ack without having to anticipate a specific PSN in the incoming TLP.
ActionCount	Number of times	0	Specifies the number of times to perform the action for the **NakSeveral, TimeOutSeveral, NakSeqNumber**, and **TimeOutSeqNumber** settings.
SeqNumberForAction	Sequence number	0	Specifies the sequence number for the **NakSeqNumber** and **TimeOutSeqNumber** settings.
AckMask	Mask Value for Ack	0	32- bit mask value for Ack modification
AckSubstValue	Value for Ack	0	32- bit value for Ack modification to be applied with the mask
ModifiedValidAck	Yes/No	No	To indicate if the modified Ack is going to serve as a valid Acknowledge for the incoming TLP
DoBadCRC16	Yes/No	No	To instruct the exerciser to invert the DLLPs CRC in addition to the applied mask/value modification

Example 1:
This example shows how to configure the Summit Exerciser so that it **Naks** each incoming TLP packet.

```
Config = AckNak {
      AckNak = Nak
}
```

This example shows how to configure the Summit Exerciser to NAK only one incoming TLP packet:

```
Config=AckNak
{
      AckNak = NakSeveral
      ActionCount = 1
}
```

Also note that encountering a Config=AckNak statement in the script overrides the setting for the ACK/NAK policy in
the Generation Options. An empty Config=AckNak statement

```
Config=AckNak
{

}
```

is the same as an Automatic Ack/Nak policy.

```
Config=AckNak
{
      AckNak = Auto
}
```

Note: Using Config = AckNak across multiple exerciser scripts.

Users should be aware of the fact that in case Config = AckNak setting is changed in a certain script that is executed separately, and then another script is executed to create the stimulus for TLPs to come in from the DUT, the AckNak setting from the first script will no longer be active, as it is getting overwritten with the generation options when the second script is getting executed.

Example 2:

```
Config=AckNak
{
     AckNak=NakSeqNumber
     Actioncount = 1
     SeqNumberForAction=5
}
```

; Don't run the above code as a separate script, otherwise behavior will not be as intended!
; Run above and below as a single script.

```
repeat = Begin
{
      Count = 10
      Counter = i
```

```
}

packet = TLP
{
      PSN = i
      TLPType = CfgRd0
      Tag = ( i & 0xFF )
      FirstDwBe = 0xF
      Register = ( (4*i) & 0xFFF )
      Length = 1
}

wait = TLP
{
      TLPType = CplD
}

repeat = End
```

Example 3:

This example shows how to configure the exerciser using the ModifyAck action.

```
; Set up to send Ack with non-zero reserved bits
Config=AckNak
{
      AckNak = ModifyAck
      ActionCount = 1
      AckMask = 0x00F0FFFF ; Mask in DLLP type and Reserved bits
      AckSubstValue = 0x00A0AA00 ; 0 is Ack type, set Reserved to 0xAAA
      ModifiedValidAck = Yes ; This is a valid Ack with proper PSN
}
; Send the Configuration read, the modified Ack will be sent as response
; to the incoming Completion
Packet=TLP
{
    TLPType=CfgRd0
    Length = 1
    FirstDwBe = 15
}
Wait=TLP
{
    TLPType=CplD
}
;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Set up to send Ack with bad CRC16
Config=AckNak
{
      AckNak = ModifyAck
      ActionCount = 1
      AckMask = 0x000000FF ; Mask in DLLP type
      AckSubstValue = 0x00000000 ; 0 is Ack type
      DoBadCRC16 = Yes ; instruct to invert CRC
}
; Set up to send DLLP with undefined encoding instead of Ack
Config=AckNak
{
      AckNak = ModifyAck
```

```
            ActionCount = 1
            AckMask = 0x000000FF ; Mask in DLLP type
            AckSubstValue = 0x00000011 ; Bad DLLP type encoding
      }
```

2.4.8 Config = Transactions

This command determines the behavior of Summit Exerciser as it responds to Memory, Configuration, and IO TLP requests. So that it properly responds to Memory and IO TLP requests, Configuration Address Space must be defined (see Page 190)

Parameter	Values	Default	Comment
AutoCfgCompletion	Yes, No	No	If set, automatically handles Configuration Read and Write TLP transactions. For a Configuration Read transaction, Completion TLP contains the data read from the internal Configuration Space according to specified register address. For a Configuration Write transaction, internal Configuration Space is updated at the address with the data taken from Configuration Write TLP, and a Configuration Write Completion is returned.
AutoMemIoCompletion	Yes, No	No	If set, automatically handles Memory and IO Read/Write TLP transactions. For Memory and IO Read transactions, a Completion TLP contains the data read from the internal Memory/IO Address Space according to specified address. For Memory and IO Write transactions, internal Memory/IO Address Space is updated at the address with the data taken from the TLP. (For Summit Exerciser Z3/Z4 also applicable to Host Memory Regions)
EnableUR	Yes, No	No	If set, enables Unsupported Request (UR) status for Memory/IO completions. **AutoMemIoCompletion** must be set to enable UR completions. (For Summit Exerciser Z3/Z4 also applicable to Host Memory Regions)
EnableCA	Yes, No	No	If set, enables Completer Abort (CA) status for Memory/IO completions. **AutoMemIoCompletion** must be set to enable CA completions. (For Summit Exerciser Z3/Z4 also applicable to Host Memory Regions)
Poisoned	Yes, No	No	If set, all Memory/IO completions have the **Poisoned** bit set. (For Summit Exerciser Z3/Z4 also applicable to Host Memory Regions)
FastMemCompleter	On, Off	Off	If On, enables the high performance Fast Memory Completer.
GenerateECRCsFastMC	Yes, No	No	Generate ECRC for each completion generated by fast memory completer.

Example:
This example enables automatic completion for Configuration TLP requests.
To automatically complete Configuration TLP requests, the Configuration Space must be configured first (see Page 192).

```
Config = Transactions {
      AutoCfgCompletion = Yes
      ; Automatically complete Configuration TLP requests.
}
```

Note: After this command, automatic completion for Memory and I/O TLP requests are turned off, since the default value (**No**) is used for the **AutoMemIoCompletion** parameter.

2.4.9 Config = Definitions

Parameter	Values	Default	Comment
Any literal	Any integer, string, payload array, or predefined value		The defined values can be used anywhere in the script as a parameter value. Only payload field supports arrays.

Example 1:

```
Config = Definitions {
      my_register     = 0x24
      my_tlptype      = CfgWr0
      my_payload      = ( 0x12345678 0xAABBCCDD 0x01020304 )
      my_wait_message = "my wait"
}
Packet = TLP {
      PSN      = Incr
      TlpType  = my_tlptype
      Register = my_register
      Payload  = my_payload
}
Config = Definitions {
      my_register = 0x20
      my_tlptype  = CfgWr1
}
Packet = TLP {
      PSN      = Incr
      TlpType  = my_tlptype
      Register = my_register
      Payload  = my_payload
}
wait = my_wait_message
```

Example 2:

This example shows how to use definitions in the expressions (see Page 181) and how to redefine the values.

```
Config = Definitions {
      READ_START = 0x10
}
```

```
; Repeat 10 times.
Repeat = Begin {
      Count=10
      Counter = i
}
; Send TLP using repeat counter (i) and
; READ_START to specify the address.
Packet = TLP {
      TLPType = CfgRd0
      Register = ( READ_START + ( 4 << i ) )
}
Repeat=End
; Redefine READ_START, now READ_START is 0x40.
Config = Definitions
{
      READ_START = ( READ_START + 0x30 )
}
; Send TLP using READ_START to specify the address.
Packet = TLP {
      TLPType = CfgRd0
      Register = READ_START
}
```

2.4.10 Config = SendInterrupt

This command is only valid for Device Emulation and allows initiating MSI or MSI-X interrupt to the system. It is only supported on Summit Exerciser Z3/Z4. In order to be able to use this feature, Z3/Z4 has to be booted in the system with initial configuration space image loaded that contains MSI and/or MSI-X capability structures. After the system has enabled MSI or MSI-X, this command can be used to send the interrupt for the specified vector. The Z3/Z4 will figure out by itself which interrupt scheme is enabled and generate the appropriate Memory Write TLP to the address containing data that was configured by the system for this interrupt vector.

Parameter	Values	Default	Comment
Vector	Zero to Maximum allocated vector	0	If neither MSI nor MSI-X is enabled or vector doesn't exist, Z3/Z4 will do nothing upon this command

Example:

```
Config=SendInterrupt
{
    Vector = 1  ; change to the desired vector number
}
```

2.4.11 Config = ATS

Summit Exerciser Z3/Z4 supports Address Translation Services for Device Emulation. Exerciser Z3/Z4 would keep the Address Translation Cache table in its memory and populate it when a Translation Request (Memory Read with AT set to Translation_Req) is sent from the script and responded by the Host system.

Z3/Z4 Exerciser would use the Address Translation Cache to convert the untranslated address to translated whenever a memory Read or Write is sent with Translated AT attribute. This will include suspending memory transactions to protected regions or to regions for which address translation is not present. This suspension can be override by using the IgnorePermissions parameter below. Z3/Z4 would also process Invalidation Request messages and reply with the proper Invalidate Completion.

The following parameters are available to configure ATS-related behavior.

Parameter	Values	Default	Comment
ClearATC	Yes/No	No	When Yes, clears all current entries in the Address Translation Cache
IgnorePermissions	Yes/No	No	When Yes, send Translated memory transactions with no regard to permissions or translation present
InvalidateCplDelay	time (15 ms units)	0	Time to delay the transmission of Invalidate Completion upon processing Invalidation Request. (expressed in 15 millisecond units)
DisableATS	Yes/No	No	When Yes, disables automated ATS functionality, so unmodified TLP packets with any AT field can be sent from the script.

2.4.12 Config = NVMe

This command is only valid for Device Emulation. It starts and configures NVMe Drive (Controller) emulation. It is only supported on Summit Exerciser Z3/Z4.

In order to use this feature, the Z3/Z4 has to be configured for Device Emulation, and inserted in a PCI Express slot on the system under test while it is powered down. The Z3/Z4 has to have the initial configuration space image loaded that contains:

- MSI and/or MSI-X capability structures;
- BAR 0 that is set up as defined by the NVMe specification;
- Class Code register as defined by the NVMe specification for NVMe controller.

Also, the binary Identify data for the Controller and Namespace(s) has to be written to a certain memory location on the Z3/Z4 (which is referenced by this instruction).

After this is done, the script containing this instruction and then the Connect script can be executed and system under test booted up. After an NVMe driver is loaded for the Drive Emulation device, it will come up in the system as an unformatted drive. It can be initialized and formatted, after which it acts as a normal disk drive. Drive capacity up to 320 Megabytes is supported.

Parameter	Values	Default	Comment
Enable	Yes/No	No	Turns on or off automatic handling of NVMe drive emulation
IdentifyDataLocation	(FROM_MEMxx_x, offset)	N/A	Same notation as in Field Substitution (see 3.1.7)

Example:

```
Config=NVMe
{
    Enable = Yes
    IdentifyDataLocation = ( FROM_MEM32_B, 0 )
}
```

The SampleFiles\Z3/Z416/Z5TrainerScripts\NVMe_DriveEmulation folder of the PCIe Protocol Suite software contains files needed to set up NVMe drive emulation. The file "nvme_drive_config_space.dat" can be loaded as the initial Configuration Space image. Also, the file "nvme_identify_data.dat" included in this sample folder can be written to the Mem32B location, offset zero. The sample file start_nvme_drive.peg contains the instruction described here.

After the Z3/Z4/Z5 configured for NVMe Drive Emilation has been booted in the PCI Express system, if the OS is Windows7, the Windows7 64-bit driver can be loaded from the ZxTrainerScripts\NVMe_DriveEmulation\Drivers\Windows7 folder.
For Windows 8.0. 8.1, 10 and 11 use the native drivers provided by the OS.

For NVMe drive emulation in Linux environments refer to the corresponding folder under ZxTrainerScripts\NVMe_DriveEmulation\Drivers\Linux.

Note on specific handling of certain Admin and NVM commands for Drive Emulation.

The Security Receive and Get Log Page Admin and the Reservation Report NVM commands are supposed to return some very specific data as mandated by the NVMe specification. In order to facilitate most flexible emulation of these commands, a mechanism is added for the user to completely specify the data to be returned.
This is achieved by writing the data to specific offsets in the Z3/Z4's **Mem64** Address Space. Whatever data is written to these locations, it would be returned upon subsequent commands, should they be issued by the NVMe Host.

The offsets for the locations for the Z3-16 are:

Reservation Report: 0x1F850000
Security Receive: 0x1F810000
Get Log Page: 0x1F900000 + 0x4000 * LogPageId

For example:
- Error Information: 0x1F904000
- SMART/Health : 0x1F908000
- FW Slot Info : 0x1F90C000

The offsets for the locations for the Z416 are:

Reservation Report: 0x1E050000

Security Receive: 0x1E010000
Get Log Page: 0x1E100000 + 0x4000 * LogPageId

For example:
- Error Information: 0x1E104000
- SMART/Health : 0x1E108000
- FW Slot Info : 0x1E10C000

etc.

The specified locations can be written from the PCIe Protocol Suite application using the Write Address Space dialog, or from the script using the AddressSpace = Write command (*refer to 14.2 AddressSpace = Write*).

Note on Interrupt Coalescing:
Starting with PCIe Protocol Suite version 7.34 NVMe Drive Emulation supports Interrupt Coalescing. It will process Set Feature commands for Interrupt Coalescing (Feature Identifier 08h) and Interrupt Vector Configuration (Feature Identifier 09h), and use the supplied Aggregation Time, Aggregation Threshold and Coalescing Disable settings when generating interrupts upon IO completions.

2.4.13 Config = NVMeDriveErrorInjection

This command is only valid for Device Emulation. It configures error injection for NVMe Drive (Controller) emulation. It is only supported on Summit Exerciser Z3/Z4.

In order to use this feature, the Z3/Z4 has to be configured for NVMe Drive Emulation, as described in the Config = NVMe section, and brought up as a drive in the System Under Test.
The command adds a certain type of Error Injection to the Drive Emulation. The error injection is going to be applied on the next matching NVMe command(s) the Drive Emulator receives from the system.

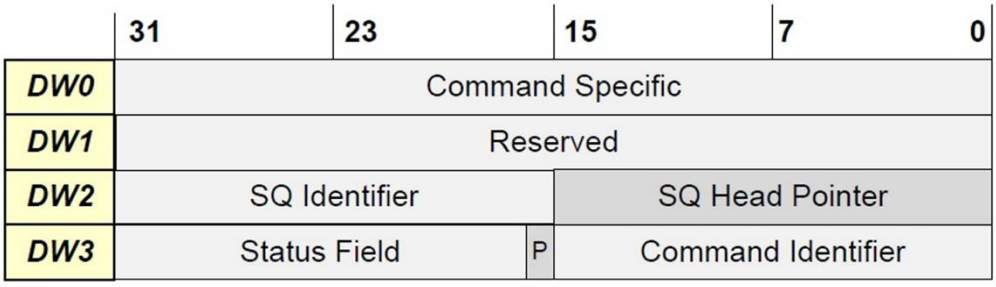
Parameter	Values	Default	Comment
ErrorInjectionType	NVME_DRV_ERR_REGISTER NVME_DRV_ERR_COMPLETION NVME_DRV_ERR_COMMAND	N/A	Specifies the type of NVMe drive error injection: - Error injection for Controller Registers - Error injection for NVMe command completions - Error injection for NVMe commands
Action	CPL_ACTION_DROP_CPL CPL_ACTION_DROP_INT CPL_ACTION_DROP_CPL_INT CPL_ACTION_CORRUPT_DATA CPL_ACTION_CORRUPT_RESERVED CPL_ACTION_CORRUPT_STATUS CPL_ACTION_CORRUPT_FIELD	N/A	Applies to command completions: - Don't update the completion queue with the completion, but send interrupt - Update the completion queue with the completion, but don't send interrupt - Don't update the completion queue with the completion, and don't send interrupt - Corrupt the completion data by setting all the bytes to 0xFF - Introduce non-zero data in the reserved fields of the completion - Substitute the status (sf) field with data supplied in the RegisterData parameter. Can be use to insert any type of status in the completion

Parameter	Values	Default	Comment
			- Corrupt the completion data by setting the specified field value to the provided value instead of the correct value. If this action is selected, the FieldBitOffset and FieldBitSize parameters will be available to specify location and size of the field. The value to substitute should be specified using the RegisterData parameter. - Also used for the CPL_ACTION_CORRUPT_FIELD error injection to specify the value to be returned in the corrupted field.
FieldBitOffset	Number (0 – 128)	0	Used for CPL_ACTION_CORRUPT_FIELD action. Specifies the starting bit for the field to corrupt (see note below).
FieldBitSize	Number (0 – 32)	0	Used for CPL_ACTION_CORRUPT_FIELD action. Specifies the bit size for the field to corrupt (see note below).
	CMD_ACTION_DROP_CMD	N/A	Applies to NVMe commands: - Don't read and execute the command upon the doorbell write
PersistsThroughControllerReset	Yes/No	No	If set to "No", a controller reset to the NVMe Drive emulator from the host clears this error injection. If "Yes", error injection stays active
ErrorCount	Number	1	Number of times to assert this error injection
ControllerReg	Address	0	Used for NVME_DRV_ERR_REGISTER error injection only.

Parameter	Values	Default	Comment
			Specifies the address of the register to apply the injection to
QueueID	Number	0	Used for NVME_DRV_ERR_COMPLETION and NVME_DRV_ERR_COMMAND error injections. Specifies the number of admin (0) or IO queue to apply this error injection on
RegisterData	Number or Timeout	0	Used for NVME_DRV_ERR_REGISTER error injection. Specifies the value to return upon a read from the register instead of the real value. A special value of 0xFFFFFFFF specifies Timeout error injection – a completion with data will not be returned upon a read from the register. Also used for the CPL_ACTION_CORRUPT_STATUS error injection to specify the value to be returned in the command status field

Note: For CPL_ACTION_CORRUPT_FIELD action the FieldBitOffset parameter specifies bit offset according
to the layout of the Completion structure from the NVM Express Specification Revision 1.2, Dated: 3 November 2014. See Figure 25 from that specification below.

Figure 25: Completion Queue Entry Layout – Admin and NVM Command Set



For example, in order to corrupt the SQ Identifier field, use:

```
Config=NVMeDriveErrorInjection
{
    ErrorInjectionType = NVME_DRV_ERR_COMPLETION
```

```
    RegisterData = 0xABCD
    QueueID = 0
    Action = CPL_ACTION_CORRUPT_FIELD
    ErrorCount = 1
    FieldBitOffset = 80  ; beginning of the SQ Identifier field in DW2 32 + 32
+ 16
    FieldBitSize = 16
}
```

2.4.14 Config = ErrorInjection

This command allows the user to injection specific types of errors into data traffic. It is supported on Summit Z416, Z58, Z516 and M616 exercisers. Not supported on Summit Z3-16.

Parameter	Values	Default	Comment
Gen12DisparityRate	1 – 65,535	1	For Gen1 and Gen2 means a disparity error is injected every Nth symbol
Gen12DisparityLane	0x0000 – 0xFFFF (Mask value in hex for 16 bit register)	None	Defines the lane or lanes on which the disparity error is injected. For example if you want errors injected on lane 0 use 0x0001. If you want errors injected on lanes 0, 2, 4, 6, 8, 10,12 and 14 use 0x5555. If you want errors on all lanes enter 0xFFFF.
EnableDisparityErrorInjection	Yes No	No	Yes = Error Injection is enabled. No = Error Injection is disabled.
Gen12SymbolRate	1 – 65,535	1	For Gen1 and Gen2 means a symbol error is injected every Nth symbol
Gen12SymbolLane	0x0000 – 0xFFFF (Mask value in hex for 16 bit register)	None	Defines the lane or lanes on which the symbol error is injected. For example if you want errors injected on lane 0 use 0x0001. If you want errors injected on lanes 0, 2, 4, 6, 8, 10,12 and 14 use 0x5555. If you want errors on all lanes enter 0xFFFF.
EnableSymbolErrorInjection	Yes No	No	Yes = Error Injection is enabled. No = Error Injection is disabled.
Gen34SyncBitRate	1 – 65535	1	For Gen3 or Gen4 means a sync bit error is injected every Nth block
Gen34SyncBitLane	0x0000 – 0xFFFF (Mask value in hex for 16 bit register)	None	Defines the lane or lanes on which the sync bit error is injected. For example if you want errors injected on lane 0 use 0x0001. If you want errors injected on lanes 0, 2, 4, 6, 8, 10,12 and 14 use 0x5555. If you want errors on all lanes enter 0xFFFF.
EnableSyncBitErrorInjection	Yes No	No	Yes = Error Injection is enabled. No = Error Injection is disabled.

Notes: For Gen1 and Gen2 the Error Rate means an error will be injected every Nth Symbol.
For Gen3 and Gen4 the Error Rate means an error will be injected every Nth Block.

The values in GUI become the default values and values written from script overwrite them.

For example, to injection errors into the data traffic generated by the Z416:

```
Config=ErrorInjection
{
    Gen12DisparityRate = 2
    Gen12DisparityLane = 1
    EnableDisparityErrorInjection = Yes
    Gen12SymbolRate = 5
    Gen12SymbolLane = 1
    EnableSymbolErrorInjection = Yes
    Gen34SyncBitRate = 2
    Gen34SyncBitLane = 1
    EnableSyncBitErrorInjection = No
}
```

This example will inject a Disparity error every 2nd symbol on Lane 0, a Symbol error every 5th symbol on Lane 0 but no SyncBit errors will be injected.

2.4.15 Config = SMBus

This command allows the specification of attributes specific to exerciser's SMBus operation. It is only available on SMBus-capable Summit Z3/Z4 Exercisers and Host Platforms.

Note: In addition to the above the command effectively asserts the SMBus reset by making sure the SMBus clock is pulled low by the host for more than 35 milliseconds

Parameter	Values	Default	Comment
I2CSlaveAddress	0 – 0xFF	0	The I2C Slave address for the Exerciser to receive response packets. This should match the SrcSlaveAddr for the packets that are transmitted by the script, otherwise the response packets won't be acknowledged by the Exerciser.
I2CSpeed	100KHz 400KHz 1MHz	100KHz	Defines I2C frequency for Exerciser's SMBus transmission.
DUTI2CSpeed	100KHz 400KHz 1MHz	100KHz	Defines I2C frequency for the DUT card's SMBus transmission (which can be set separately by an NVMe-MI command). Used to configure Exerciser's I2C receiver.

Example:

```
Config=SMBus
      {
          I2CSlaveAddress = 0x14
          I2CSpeed = 100KHz
          DUTI2CSpeed = 400KHz
      }
```

2.4.16 Config = RawLtssm

This command is used to reconfigure the Raw LTSSM link parameters. It can only be placed within RawLtssm block. The parameters are the same as for the RawLtssm=Setup command. Supported on the Z416, Z58, Z516 and M616 Exercisers.

See 2.16 Raw Ltssm Commands for more details.

Parameter	Values	Default	Comment
LtssmSpeed	2_5, 5_0, 8_0, 16_0 and 32_0	2_5	Specifies one of five PCIe speeds of operation (32 GT/s only available for M616, Z58 and Z516)
LtssmLinkWidth	x1, x2, x4, x8 or x16	x1	Specifies starting link width
LtssmSkipType	None, Gen12, Gen34, Gen34wEDS, Gen4wCtrl, Gen4wCtrlEDS, Gen5wCtrl, Gen5wCtrlEDS	None	Specifies automatic Skip insertion policy. Can be one of: No Skips inserted, Gen1/2 style Skips inserted, Gen3/4 Skips or Gen3/4 Skips followed by EDS, Gen4 and Gen5 alternating Standard and Control Skips, without or with EDS are inserted. Skips are inserted where appropriate in between the programmed Ordered Sets / patterns based on the Skip Frequency.
LtssmSkipFreq	Number. Frequency expressed in DWORDs	0	Specifies the base frequency of the automatic Skip generation expressed in number of DWORDs of traffic in between.

Refer to examples: See 2.16.3 RawLtssm Examples.

2.4.17 Config = HostMemoryPartitions

This command is used to add, modify or remove a memory partition to a Host Memory Region. There are three Host Memory Regions (one 64 bit and two 32 bit) that can be enabled for the case of Host Emulation in the Transactions tab of the Generation Options dialog. There, each one can be assigned a fixed Base Physical Address on PCIe. Which means that whatever structures and data are placed in these regions for the purpose of Host Emulation are going to be close to each other in terms of their physical addresses.

The Host Memory Partitions allow to define sub-regions within the three main Host Memory Regions that are assigned different Base Physical Addresses. This allows for Host Emulation with structures and data that are residing anywhere within the 64-bit memory space.

The command is available on Summit Z3/Z416.

Parameter	Values	Default	Comment
Location	Mem64 Mem32A Mem32B	Mem64	Specifies Host or Device memory region in which the Partition is to be located (see 2.12 AddressSpace command for more information).
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies **Offset** in bytes from the beginning of memory region specified in the **Location** parameter. This will determine the starting address for the Partition.
Size	Any number from 0. The combination of **Offset** and **Size** parameters is limited by the maximum allowed address for the region.	0	Specifies the size in bytes for the Partition, starting with the **Offse**t. If the command is executed with an **Offset** for a partition that already exists and the **Size** of zero, that Partition will be deleted.
AddressLo	0x00000000 – 0xFFFFFFFF	0	The lower 32 bits of the physical address to be assigned for the Partition, in case of the Mem64 **Location**.
AddressHi	0x00000000 – 0xFFFFFFFF	0	The upper 32 bits of the physical address to be assigned for the Partition, in case of the Mem64 **Location**.
Address	0x00000000 – 0xFFFFFFFF	0	The physical address to be assigned for the Partition, in case of the Mem32A or B **Location**.

Examples:

The following example can be also found at the end of the readme.txt located in the NVMe Host Emulation sample Exerciser script folder.

Note on creating Queues/structures with very different base physical adresses.

The Config=HostMemoryPartitions construct can be used to set up the Host Emulation in a such way that different queues are placed at physical address ranges that are quite disparate.
For example, if there is a need to have the Admin Submission Queue at the same Base Address - 0x00000004 : 0x2FAA8000,

but have the Admin Completion Queue to reside very far from this address, the following code can be added to nvme_config.peg or nvme_init.peg:

```
Config=HostMemoryPartitions
{
    Location = Mem64
    Offset = 0x2000   ;  this is the offset for the Admin Completion Queue,
                         see     above
    Size = 0x800      ;  assuming the queue of 128 entries
    AddressLo = 0xAB000000
    AddressHi = 0x1C002D00 ; very far from the 0x00000004 : 0x2FAA8000, base
                             address for the Admin Submission Queue
}
```

only the code for creation of Admin Completion Queue will need to change.

(in nvme_init.peg lines 74-86 the Payload for the low address should be changed from 00A0AA2F to 000000AB, and the Payload for the high address - from 04000000 to 002D001C)

Similarly, if the address for IO 1 Submission Queue has to be reassigned, the following code should be added

```
Config=HostMemoryPartitions
{
    Location = Mem64
    Offset = 0x10000   ;  this is the offset for IO 1 Submission Queue, see
above
    Size = 0x10000   ;
    AddressLo = 0x78340000
    AddressHi = 0x8E112235 ; very far from all other queue addresses
}
```

only the code for creation of IO 1 Submission Queue will need to change.

(in nvme_admin_cmd_create_sub_q.peg, lines 116-117 need to be changed from
PRP1_Low = 0x2FAB8000
PRP1_High = 0x4
to
PRP1_Low = 0x78340000
PRP1_High = 0x8E112235)

However, all the code that puts Command structures in the queue will stay the same, as the structures are referenced by the Offset only.

2.4.18 Config = MemRegionErrorInjection

This command is used to associate certain types of error injection with regions (address ranges) in Host or Device memory, so that errors are injected only when those regions are accessed. For example, it can be used for NVMe Host emulation to associate the error with a read of particular command, PRP/SGL list or data location that the Controller performs from the Host memory. The error can be set up much before the command is executed and will be applied to the Completion TLPs that Exercizer returns upon such read.

The command is available on Summit Z3/Z416.

Parameter	Values	Default	Comment
Location	Mem64 Mem32A Mem32B IOA IOB	Mem64	Specifies Host or Device memory region in which the range with Error Injection to be located (see 2.12 AddressSpace command for more information). IOA and IOB are only applicable to Device Emulation.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the Location parameter	0	Specifies **Offset** in bytes from the beginning of memory region specified in the **Location** parameter. This will determine the starting address for the range with Error Injection.
Size	Any number from 0. The combination of Offset and Size parameters is limited by the maximum allowed address for the region.	0	Specifies the size in bytes for the range with Error Injection, starting with the **Offse**t.
MemErrorType	None . PoisonedTLP ErrorLCRC	None	Specifies the type of Error Injection. **PoisonedTLP** type instructs to set the Poisoned (EP) bit in the applicable Completion TLPs returned for the range. **ErrorLCRC** type instructs to provide incorrect LCRC in the applicable Completion TLPs returned for the range. **None** type can be used to clear previously set up Error Injection for the range (especially if the infnite **ErrorCount** was specified for the range).
ErrorCount	Number (0 means infinite)	1	Specifies the number of times to apply the error injection. Each Completion TLP consumes one count (so if multiple completions are returned upon a Read, each one counts). After an error injection is applied for the specified of times it is cleared. A count

Parameter	Values	Default	Comment
			of **0**, though can be used for infinite number of error injections. In this case another error injection with the same **Location** and **Offset** and the error type of **None** can be used to clear such injection.

Examples:

```
Config=MemRegionErrorInjection
      {
         Location = Mem32A
        Offset = 0x100
      Size = 512
       MemErrorType = ErrorLCRC   ; Set up bad LCRC error injection for 5
times at
                                    Offset 0x100 in Mem32A
        ErrorCount = 5
      }

Config=MemRegionErrorInjection
      {
        Location = Mem64
      Offset = 0x2000
      Size = 512
       MemErrorType = PoisonedTLP ; Set up Poisoned TLP error injection at
Offset
                                      0x2000 in Mem64
       ErrorCount =  0            ; Infinite
      }

Config=MemRegionErrorInjection
      {
       Location = Mem64
       Offset = 0x2000
      Size = 512
      MemErrorType = None        ; Clear the error injection at Offset 0x2000
in
                                   Mem64
       ErrorCount =  1            ; Doesn't matter
      }
```

2.4.19 Config = LaneMargining

This command produces Lane Margining transaction. It could be transmitted either while emulating Host (to initiate Lane Margining commands) or Device (to emulate Lane Margining responses).

The command is available on Summit Z416, Z58, Z516 and M616.

Parameter	Values	Default	Comment
LM_Ctrl_UsageModel	0, 1	0	Usage Model
LM_Ctrl_Command	No_Command, Access_Retimer_register, Report_Margin_Control_Capabilities, Report_M_NumVoltageSteps, Report_M_NumTimingSteps, Report_M_MaxTimingOffset, Report_M_MaxVoltageOffset, Report_M_SamplingRateVoltage, Report_M_SamplingRateTiming, Report_M_SampleCount, Report_M_MaxLanes, Report_Reserved, Set_Error_Count_Limit, Go_to_Normal_Settings, Clear_Error_Log, Step_Margin_timing_to_offset, Step_Margin_voltage_to_offset, Vendor_Defined	No_Command	Lane Margining Command
LM_Ctrl_MarginType	000b, 001b, 010b, 011b, 100b, 101b, 110b, 111b	111b	Margin Type.
LM_Ctrl_ReceiverNumber	Broadcast, RxA, RxB, RxC, RxD, RxE, RxF, Reserved	Broadcast	Receiver Number
LM_Ctrl_MarginPayload	0x00 .. 0xFF	0x9C	Margin Payload.

Note This command is not supported for Z3-16 Exerciser.
LM_Ctrl_Command suppresses LM_Ctrl_MarginType, LM_Ctrl_ReceiverNumber and LM_Ctrl_MarginPayload values if there were listed before. So it is better to use LM_Ctrl_Command first.
You can use arrays for all fields except LinkWidth and SkipCount. Like this you can use individual values for each lane. If you use one value, it will be used for all lanes.

Example:

```
Config=LaneMargining
{
    LM_Ctrl_UsageModel     = 0
    LM_Ctrl_Command        = ( No_Command, Access_Retimer_register,
                               Report_Margin_Control_Capabilities,
                                Report_M_NumVoltageSteps )
    LM_Ctrl_MarginPayload  = ( 0x9C, , , 4 )
}
```

In this example Usage Model is 0 for all lanes. Command and Margin Payload are individual for first 4 lanes. Margin Payload is specified for lane 1 and lane 4, lanes 2, 3 and for 5, 6, ... are filled by default value.

Note LM_Ctrl_UsageModel, LM_Ctrl_Command, LM_Ctrl_MarginType, LM_Ctrl_ReceiverNumber and LM_Ctrl_MarginPayload fields can be used to emulate Lane Margining Control Registers write. LM_Ctrl_Command suppresses LM_Ctrl_MarginType, LM_Ctrl_ReceiverNumber and LM_Ctrl_MarginPayload values if there were listed before. So it is better to use LM_Ctrl_Command first.

Example:

```
Packet=TLP
{
    TLPType=CfgWr0
    PSN = 1025
    Register = 0x108
    LM_Ctrl_MarginType = 001b
    LM_Ctrl_ReceiverNumber = RxA
    LM_Ctrl_MarginPayload = 0x89
}
```

2.4.20 Config = LinkEqualization

This command is used to send either TestPreset values or TestCoefficient values from the exerciser's transmitter requests to the peer device.

If both TestPreset values and TestCoefficient values are defined the TestPreset values will be used.

Parameter	Values	Default	Comment
TestCoefficients8GTS	(PreCursor, Cursor, PostCursor)	0, 0, 0	PreCursor values: 0 - 63 Cursor values: 0 – 63 PostCursor values: 0 - 63
TestPreset8GTS	Preset	0	Preset: 0 - 15
TestCoefficients16GTS	(PreCursor, Cursor, PostCursor)	0, 0, 0	PreCursor values: 0 - 63 Cursor values: 0 – 63 PostCursor values: 0 - 63
TestPreset16GTS	Preset	0	Preset: 0 - 15
TestCoefficients32GTS	(PreCursor, Cursor, PostCursor)	0, 0, 0	PreCursor values: 0 - 63 Cursor values: 0 – 63 PostCursor values: 0 – 63 (only applicable to Z58, Z516, and M616)
TestPreset32GTS	Preset	0	Preset: 0 – 15 (only applicable to Z58 Z516 and M616)
TestCoefficients64GTS	(PreCursor, Cursor, PostCursor)	0, 0, 0	PreCursor values: 0 - 63 Cursor values: 0 – 63 PostCursor values: 0 – 63 (only applicable to M616)
TestPresets64GTS	Preset	0	Preset: 0 – 15 (only applicable to M616)

If exerciser is EndPoint (EP), in phase 2, exerciser will request the Downstream Preset (DSP) to set its transmitter coefficients/presets as specified above.

If exerciser is RootComplex (RC), in phase 3, exerciser will request Upstream Preset (USP) to set its transmitter coefficients/presets as specified above.

Example1:

```
Config=LinkEqualization
{
      TestCoefficients8GTS = (12, 15, 3)
      TestPreset8GTS = 3
      TestCoefficients16GTS = (10, 33, 63)
      TestPreset16GTS = 15
    TestCoefficients32GTS = (10, 12, 35)
    TestPreset32GTS = 2
}
```

Example2:

```
PCIeFlitMode=True
Config=LinkEqualization
{
    TestCoefficients8GTS = (5, 12, 4)
    TestPreset16GTS = 6
    TestCoefficients32GTS = (19, 5, 43)
    TestPreset64GTS = 12
}
```

2.4.21 Config = Low Power

Active State Power Management (ASPM) is a power management protocol used to manage PCI Express-based (PCIe) serial link devices as links become less active over time.
While ASPM brings a reduction in power consumption, it can also result in increased latency as the serial bus needs to be 'woken up' from low-power mode, possibly reconfigured and the host-to-device link re-established. This is known as ASPM exit latency and takes up valuable time.
You can utilize the L1 state to reduce power consumption. The traditional L1 state allows the reference clock to be disabled on entry to L1. L1 sub-states add two “pseudo sub-states,” called L1.1 and L1.2, to the LTSSM, which can be used to turn off additional analog circuits in the PHY. L1.1 allows the common-mode voltage to be maintained, while L1.2 allows all high-speed circuits to be turned off.

Note: Due to long exit latency from low power sub states some loss of traffic has been observed when analyzer latency exceeds DUT latency using the common reference clock. Teledyne recommends the use of the internal reference clock setting for low power sub-states testing.

Parameter	Values	Default	Comment
ProgTimerState	None, TxL0s, L1		The programmble LTSSM timer allows the Z4 to automatically exit a chosen low power state after a programmable time, and then return to L0. None - timer is disabled. This is the default and the normal use of the low power states. TxL0s - the programmable timer will be active while in TxL0s.Idle. When the timer expires, the link will automatically be directed to L0, and then return to TxL0s L1 - The programmable Timer will be active in L1.Idle, L1.1.Idle or L1.2.Idle. When the timer expires, the link will be directed out of L1 and back to L0, and will return to L1 when requested by the DS device. Timer value: this will be the time in ns that the Z4 LTSSM will spend in the chosen state before exiting that state. The time spent will be +- 8 ns from the chosen value.
ProgTimer	0 – 524285		Time the link will remain in that state until it automatically exists that state.
Modes	ASPM, PCIPM		ASPM (Active State Power Management) PCIPM (PCIe Power Management) In this mode a PM Ack is always sent in response to a PM Nak.
L1SubStates	L1_1, L1_2, Disable		Enable L1-1: The link common mode voltages are maintained. You can use a bidirectional open-drain clock request (CLKREQ#) signal for entry to and exit

Parameter	Values	Default	Comment
			from this state. Upstream and Downstream Port are not required to be enabled to detect electrical idle (EI) Enable L1-2: The link common mode voltages are not required to be maintained. You can use a bidirectional open-drain clock request (CLKREQ#) signal for entry to and exit from this state. Upstream and Downstream Port are not required to be enabled to detect electrical idle (EI)
SendPMAcksInRespToL1	Yes, No		Send PM ACK in response to every L1 ASPM request.
PMAckNakCount	0 – 255	0	This allows you to vary the number of PM Acks/Naks received before either a PM Ack is sent to the host or prior to transitioning to the L1 state.
TimePowerOn	0 -	10000	Port T POWER ON Value This parameter sets the time (in usec) that this Port requires the port on the opposite side of the link to wait in L1.2 EXIT after sampling CLKREQ# is asserted before actively driving the interface.
TCommonMode	0 -	10000	Common Mode Restore Time This parameter sets the value of TCOMMONMODE (in usec), which must be used by the Downstream Port for timing the re-establishment of common mode.

Example:

```
Config=LowPower
{
    ProgTimer = 777
    ProgTimerState = L1
    Mode = ASPM
    L1SubStates = L1_1
    SendPMAcksInRespToL1 = Yes
    PMAckNackCount = 444
    TimePowerOn = 456
    TimeCommonMode = 123
}
```

2.4.22 Config = StoreMessageData

This command allows to set up storing of the payload for an incoming Message with Data TLP at a location in an Address Sapce or a Host Memory Region.

Parameter	Values	Default	Comment
DoStore	Yes/No	No	Turns Message payload storage on or off
StoreAt	(FROM_MEMxx_x, offset)	N/A	See comment below

Comment: Specifies the location in the Address Space (Host Memory Region) where the data should be stored.
Values have the same notation as in Field Substitution (see 2.1.1.7).

Example:

```
Loop=Begin
{
    Count=100
}

; Set up storing of the payload for the incoming MsgD
Config=StoreMessageData
{
    DoStore = Yes
    StoreAt = (FROM_MEM64, 16) ; Mem 64 Host memory Region should be enabled
in the generation options
}

; Send a request message
Packet=TLP
{
    TLPType=MsgD
    Length = 1
    MessageCode = Vendor_Defined_Type0
    VendorId = PCI_SIG
    Payload = ( 0x01000000 )
}

; Wait for the response message
Wait=TLP
{
    TLPType=MsgD
}
```

; Use the stored data to break out of the loop only when the least significant bit in the first DWORD of the
; response message payload was set to zero

```
Loop=Break
{
    Location=Mem64
    Offset = 16 ;
    FieldSize = Dword
    Endian = Little
    FieldAction = MaskAND
    Mask = 0x1
    Result = 0 ;
}

Loop=End
```

2.4.23 Config = TriggerOut

This command can be used to send a trigger out signal from the Summit Z416 Analyzer/Exerciser to the Summit T416/T48's Trigger In SMA connector and cause the analyzer to start collecting data traffic. In order to use the Trigger Out function, you would need to connect the Summit Z416 Sync Port with a sync cable and connect it to the Trigger In port of a T48/T416.

Note: The trigger latency for the Z4 Trigger Output is 240ns +/-10ns measured from the time the triggering pattern happens on the bus until the trigger output shows up on the Summit Z416 Trigger Out connector.

Parameter	Values	Default	Comment
Trigger	Yes No	No	If set to Yes, will generate a signal from the Z416, which is typically sent to the Summit T416/T48 to start data collection.

Example:

```
Config=triggerOut
{
      Trigger = Yes
}
```

2.4.24 Config = LaneTerminations

This command can be used to apply/remove lane terminations on the exerciser's PCIe lanes in order to simulate surprise removal and/or hot plug of the device. It is available for Summit Z416, Z58, Z516 and M616 exercisers.

Parameter	Values	Default	Comment
LaneBitmap	0 – 0xFFFF	0xFFFF	Provides a bitmap for up to 16 PCIe lanes where 1 means termination is ON for this lane and 0 means termination is OFF

Examples:

```
Config=LaneTerminations
{
      LaneBitmap = 0 ; all lanes OFF, surprise removal
}

Wait=1000000000 ; wait for 1 second

Config=LaneTerminations
{
      LaneBitmap = 0xF ; hot plug, lower 4 lanes ON, x4
}
…
Config=LaneTerminations
{
```

```
      LaneBitmap = 0b10101010101010101010101010101010 ; each even lane
OFF, each odd lane ON
}
```

2.4.25 Config = CXL_Link

This command configures the CXL link parameters. Some of them are advertised during the CXL link negotiation, thus the command should be used before the link is initialized.
This command is only available for Z516 and M616 Exercisers and requires CXL Mode to be enabled in generation options.

Parameter	Values	Default	Comment
InterconnectVersion	1_0	1_0	CXL Protocol Interconnect Version advertised during the negotiation. NOTE: CXL specification 1.1 doesn't define the 1.1 value.
CacheMemSupport	CacheOnly MemOnly CacheMem	CacheMem	Advertises support for CXL.cache/CXL.mem or both.
MultiDataHeaderSupport	Yes No	Yes	Advertises Multi Data Header Support.
AckForceThreshold	0 – 255	16	This specifies how many Flit Acks the Link Layer should accumulate before injecting a LLCRD.
AckFlushRetimer	0 – 1023	512	This specifies how many link layer clock cycles the entity should wait in case of idle, before flushing accumulated Acks using a LLCRD.
LLRWrapValue	0 – 255	255	Value after which LLR sequence counter should wrap to zero.
MaxNumRetry	0 – 31	10	A threshold for the NUM_RETRY counter, after which the local retry state machine transitions to the RETRY_PHY_REINIT.
MaxNumPhyReinit	0 – 31	10	A threshold for the NUM_PHY_REINIT counter, after which, instead of transitioning from RETRY_LLRREQ to RETRY_PHY_REINIT, the

Parameter	Values	Default	Comment
			LRSM will transition to RETRY_ABORT.
TimeoutThreshold	0 – 16383	1024	A threshold for TIMEOUT counter, after which if Retry.Ack sequence was not received, then the RETRY.Req request is sent again to retry the same flit.
TxEnableFC	Yes No	Yes	If flow control mechanism is disabled, exerciser will send the transactions even if credits are not available from the remote side.
RxEnableFC	Yes No	No	Use flow control mechanism to indicate to the remote side that the exerciser is not able to take any more request/response/data.
CacheReqCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.cache request credits advertised during CXL link up.
CacheRspCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.cache response credits advertised during CXL link up.
CacheDataCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.cache data credits advertised during CXL link up.
MemReqCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.mem request credits advertised during CXL link up. Note, that it is used only for S2M packets and has no sense for M2S, as there are no requests back.
MemRspCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.mem response credits advertised during CXL link up. Note, that it is used only for M2S packets and has no sense for S2M, as there are no responses back.

Parameter	Values	Default	Comment
MemDataCredits	0, 1, 2, 4, 8, 16, 32, 64, 128, 256	256	The number of CXL.mem data credits advertised during CXL link up.
LLCTRLUserControl	On Off	Default value is set in generation options. It is turned off by default. However, when RetryReqUserTestPreference is set explicitly, LLCTRLUserControl is automatically set to **On**.	Disables automatic LLCTRL flit generation. User gets full control over LLCTRL traffic.
RetryReqUserTestPreference	Auto Manual	Auto	Defines whether ESeq, NUM_RETRY and NUM_PHY_REINIT values are set automatically by device, or manually in the script. **NOTES:** When set explicitly in the script, LLCTRLUserControl=On automatically. This configuration can't be changed within a single script. It should be set once before sending any LLCTRL packets.

2.4.26 Config = CXL_ARB_MUX

This command configures the CXL ARB/MUX parameters.
This command is only available for Z516 and M616 Exercisers and requires CXL Mode to be enabled in generation options.

Parameter	Values	Default	Comment
CXLArbitration	Normal 1CacheMem_7IO 2CacheMem_6IO 3CacheMem_5IO 4CacheMem_4IO 5CacheMem_3IO 6CacheMem_2IO 7CacheMem_1IO	Normal	Specifies arbitration scheme between CXL.io and CXL.cache/mem packets. **Normal** uses Round Robin Arbitration Scheme. **{X}CacheMem_{Y}IO** specifies out of 8 clock cycles, CXL.cache/mem flits will be given priority for only X clock cycles and CXL.io flits will be sent for the remaining Y clock cycles.

2.4.27 Config = CXL_VLSM

This command gives control over the virtual link state machines (vLSMs). It allows to change their states, to send ARB/MUX Link Management request or response packets (ALMP), to corrupt ALMP creating erroneous conditions.
This command is only available for Z516 and M616 Exercisers and requires CXL Mode to be enabled in generation options.

Parameter	Values	Default	Comment
VLSMTarget	IO CacheMem	IO	Specifies the target VLSM.
VLSMAction	StateChange ForceStateChange ForceALMPRequest ForceALMPStatus PreventRetrainEntering RestoreRetrainEntering	StateChange	Specifies the necessary action on VLSM. **StateChage** is a graceful VLSM state change to the VLSM state specified in **VLSMState**. It should be used during normal operation. **ForceStateChange** is a forceful state change to the VLSM state specified in **VLSMState**. It is expected to be used for creating erroneous conditions. **ForceALMPRequest** is a force emission of the ALMP Request. It is expected to be used for creating erroneous conditions. **ForceALMPStatus** is a force emission of the ALMP Status. It is expected to be used for creating erroneous conditions. **PreventRetrainEntering** prevents selected VLSM from entering Retrain state. **RestoreRetrainEntering** restores normal Retrain entering settings on selected VLSM.
VLSMState	Reset Active DeepestAllowablePmState Idle_L1_1 Idle_L1_2 Idle_L1_3 Idle_L1_4 L2 LinkReset LinkError Retrain Disable	Reset	Target VLSM state for a state change or the VLSM state to put into the ALMP. Note, that not all the states are allowed for all the actions. See the CXL specification for details or follow the script processing errors.
CorruptALMP	Yes No	No	Specifies if all the ALMPs should be corrupted from now on. It is expected

Parameter	Values	Default	Comment
			to be used for creating erroneous conditions.
ALMPErrorType	1BitError 2BitError	1BitError	Specifies type of injected ALMP error. Should be used with CorruptALMP field set to “Yes”.

Examples:

```
; Send ALMP Request without errors
Config=CXL_VLSM
{
VLSMTarget = CacheMem
VLSMAction = ForceALMPStatus  ; Without forcing there will be no ALMP
                              ; in the trace
}

; Inject 1 bit error into the next ALMP
Config=CXL_VLSM
{
VLSMTarget = CacheMem
VLSMAction = ForceALMPStatus  ; Without forcing there will be no ALMP
                              ; in the trace
CorruptALMP = Yes
ALMPErrorType = 1BitError
}

; Inject 2 bit error into the next ALMP
Config=CXL_VLSM
{
VLSMTarget = CacheMem
VLSMAction = ForceALMPStatus  ; Without forcing there will be no ALMP
                              ; in the trace
CorruptALMP = Yes
ALMPErrorType = 2BitError
```

2.4.28 Config = CXL_Slot_Mappings

This command sets hints for the device to which slot each packet should be mapped. Slot mappings settings can be changed dynamically in the script if needed. It is only available for the M616 Exerciser and requires CXL Mode to be enabled in generation options and CXL256BFlitMode enabled in the script as well. Other parameters of the Config = CXL_Slot_Mappings command depend on the SlotMappingType parameter, which defines the type of the mapping to be set.

Parameter	Values	Default	Comment
SlotMappingType	CacheH2D CacheD2H MemM2S MemS2M	The parameter is required, there is no default value for it.	Depending on the SlotMappingType, the other parameters vary.

2.4.28.1 SlotMappingType = CacheH2D

Parameter	Values	Default	Comment
H2DReqSlotFormat	FirstH0 FirstG0 FirstHS0	FirstG0	Specifies slot format and position inside the slot where H2D Requests should be mapped.
H2DReqSlotNumber	0-14	0	Specifies slot number where H2D Requests should be mapped
H2DRspSlotFormat	FirstH1 SecondH1 FirstHS1 SecondHS1 G0 FirstG1 SecondG1 ThirdG1	FirstG1	Specifies slot format and position inside the slot where H2D Responses should be mapped.
H2DRspSlotNumber	0-14	0	Specifies slot number where H2D Responses should be mapped
H2DDataHeaderSlotFormat	FirstH12 SecondH12 ThirdH12 FirstHS12 SecondHS12 ThirdHS12	FirstH12	Specifies slot format and position inside the slot where H2D Data Headers should be mapped.

Parameter	Values	Default	Comment
	FirstG12 SecondG12 ThirdG12 FourthG12		
H2DDataHeaderSlotNumber	0-14	0	Specifies slot number where H2D Data Headers should be mapped

2.4.28.2 SlotMappingType = CacheD2H

Parameter	Values	Default	Comment
D2HReqSlotFormat	FirstH2 FirstHS2 FirstG2	FirstG2	Specifies slot format and position inside the slot where D2H Requests should be mapped.
D2HReqSlotNumber	0-14	0	Specifies slot number where D2H Requests should be mapped
D2HRspSlotFormat	H2 FirstH3 SecondH3 ThirdH3 FourthH3 FirstHS3 SecondHS3 ThirdHS3 FirstG2 SecondG2 FirstG3 SecondG3 ThirdG3 FourthG3	FirstG3	Specifies slot format and position inside the slot where D2H Responses should be mapped.
D2HRspSlotNumber	0-14	0	Specifies slot number where D2H Responses should be mapped
D2HDataHeaderSlotFormat	FirstH13 SecondH13 ThirdH13 FourthH13 FirstHS13 SecondHS13	FirstH13	Specifies slot format and position inside the slot where D2H Data Headers should be mapped.

Parameter	Values	Default	Comment
	ThirdHS13 FirstG13 SecondG13 ThirdG13 FourthG13		
D2HDataHeaderSlotNumber	0-14	0	Specifies slot number where D2H Data Headers should be mapped

2.4.28.3 SlotMappingType = MemM2S

Parameter	Values	Default	Comment
M2SReqSlotFormat	FirstH4 FirstHS4 FirstG4		Specifies slot format and position inside the slot where M2S Requests should be mapped.
M2SReqSlotNumber	0-14	0	Specifies slot number where M2S Requests should be mapped
M2SRwDSlotFormat	FirstH14 FirstHS14 FirstG14	FirstG3	Specifies slot format and position inside the slot where M2S Request With Data should be mapped.
M2SRwDSlotNumber	0-14	0	Specifies slot number where M2S Request With Data should be mapped
M2SBIRspSlotFormat	FirstH5 SecondH5 FirstHS5 SecondHS5 FirstG5 SecondG5 ThirdG5	FirstH13	Specifies slot format and position inside the slot where M2S BI Responses should be mapped.
M2SBIRspSlotNumber	0-14	0	Specifies slot number where M2S BI Responses should be mapped

2.4.28.4 SlotMappingType = MemS2M

Parameter	Values	Default	Comment
S2MNDRSlotFormat	FirstH7 SecondH7 FirstHS7 SecondHS7 G6 FirstG7 SecondG7 ThirdG7	FirstG7	Specifies slot format and position inside the slot where S2M No Data Responses should be mapped.
S2MNDRSlotNumber	0-14	0	Specifies slot number where S2M No Data Responses should be mapped
S2MDRSSlotFormat	FirstH15 SecondH15 FirstHS15 SecondHS15 FirstG15 SecondG15 ThirdG15	FirstH15	Specifies slot format and position inside the slot where S2M Data Responses should be mapped.
S2MDRSSlotNumber	0-14	0	Specifies slot number where S2M Data Responses should be mapped
S2MBISnpSlotFormat	FirstH6 FirstHS6 FirstG6	FirstG6	Specifies slot format and position inside the slot where S2M BI Snoops should be mapped.
S2MBISnpSlotNumber	0-14	0	Specifies slot number where S2M BI Snoops should be mapped

Examples:

```
CXL256BFlitMode=CXL_3_0

Config=CXL_Slot_Mapping
{
    SlotMappingType = CacheH2D
    H2DReqSlotFormat = FirstG0
    H2DReqSlotNumber = 14
    H2DRspSlotFormat = FirstH1
    H2DRspSlotNumber = 0
    H2DDataHeaderSlotFormat = FourthG12
    H2DDataHeaderSlotNumber = 1
}

Config=CXL_Slot_Mapping
```

```
{
    SlotMappingType = CacheD2H
    D2HReqSlotFormat = FirstH2
    D2HReqSlotNumber = 0
    D2HRspSlotFormat = ThirdHS3
    D2HRspSlotNumber = 8
    D2HDataHeaderSlotFormat = FourthG13
    D2HDataHeaderSlotNumber = 1
}

Config=CXL_Slot_Mapping
{
    SlotMappingType = MemM2S
    M2SReqSlotFormat = FirstH4
    M2SReqSlotNumber = 0
    M2SRwDSlotFormat = FirstH14
    M2SRwDSlotNumber = 0
    M2SBIRspSlotFormat = FirstH5
    M2SBIRspSlotNumber = 0
}

Config=CXL_Slot_Mapping
{
    SlotMappingType = MemS2M
    S2MNDRSlotFormat = FirstH7
    S2MNDRSlotNumber = 0
    S2MDRSSlotFormat = SecondH15
    S2MDRSSlotNumber = 0
    S2MBISnpSlotFormat = FirstH6
    S2MBISnpSlotNumber = 0
}
```

2.4.29 Config = CXL_ErrorInjection

This command controls different kinds of error injection into the traffic to check error conditions. By default, no errors are injected. Error injection may be enabled or disabled at any time.
NOTE: This command is only available for Z516 and M616 Exercisers and requires CXL Mode to be enabled in generation options.

Parameter	Values	Default	Comment
InjectFlitCRCErrorType	ToAllFlits ToProtocolFlits ToControlFlits	No	Injects a CXL Flit CRC error. Specifies the type of an error injected into traffic. Use with **InjectFlitCRCErrorCount**.
InjectFlitCRCErrorCount	0 – 0xFFFFFFFF	0	Allows to inject CXL Flit CRC error. Specifies the

Parameter	Values	Default	Comment
			number of errors to inject into traffic. Use with **InjectFlitCRCErrorType**. NOTE: If no number is selected, no CRC errors get injected into the traffic.
InjectProtocolIDError	No Correctable Uncorrectable Unexpected Uncorrectable _Mismatch	No	Injects Protocol ID error into the flit that is following this instruction in the traffic.
SendViral	Yes No	No	Enables or disables sending Viral information to the remote side. Yes = device responds with MetaField = No-op. Confirm in the trace that the device is going into a viral state.
SetRSVDBitsTo_1	Yes No	No	Yes = sets the reserved bit to 1
InjectFlitHeaderErrorCount	0 – 65535	0	If it is necessary to inject any type of errors in a flit header, the quantity of errors in this cell must be indicated. Example: 5 errors are selected, after generation is done there will be 5 flits with the selected error
ReqCreditErrorValue	0 – 15	0	NOTE: These errors will be injected only if
DataCreditErrorValue			

Parameter	Values	Default	Comment
RespCreditErrorValue			**InjectFlitHeaderErrorCount** has a value exceeding 0. Example: If **InjectFlitHeaderErrorCount** = 1, only one flit will have the injected values"2" in the cell of RequestCreditErrorValue, DataCreditErrorValue, ResponseCreditDataValue.
Slot0ErrorValue	0 – 7		Allows to set specific value of Slot0, Slot1, Slot2, or Slot3 Format Type field in the flit header (for protocol flits). NOTE: Option available with **InjectFlitHeaderErrorCount** with a value higher than 0.
Slot1ErrorValue			
Slot2ErrorValue			
Slot3ErrorValue			
ToggleType	Yes No`	No	Allows to toggle (change to opposite) the flit header. NOTE: Active with **InjectFlitHeaderErrorCount** with a value higher than 0.
ToggleAck	Yes No	No	Allows to toggle (change to opposite) the specific field (Ack, BE, or SZ) of the flit header. NOTE: Active with **InjectFlitHeaderErrorCount** with a value higher than 0.
ToggleBE			
ToggleSZ			
ToggleRsvdBit1	Yes No	No	Allows to toggle (change to opposite) the Reserved Bit (1, 17, 18 or 19) of the flit header. To inject errors in any of the bits: 1,17,18,19 reserved bits, select **Yes**.
ToggleRsvdBit17			
ToggleRsvdBit18			

Parameter	Values	Default	Comment
ToggleRsvdBit19			NOTE: Active with **InjectFlitHeaderErrorCount** with a value higher than 0.
EnableNullFlitError	Yes No	No	Injects CXL Null Flit errors to all subsequent Null Flits until it is turned off. Use with NullFlitErrorPayload.
NullFlitErrorPayload	0x0 – 0xFFFFFFFF	0x00000001	First 32 bits of Null Flits' payload

2.4.30 Config = IDE_Key

This command allows to specify key and IV used for IDE encryption.

NOTE: Only available on Summit Z58, Z516, and M616 in Host Emulation mode.

Parameter	Values	Default	Comment
IDEKey	Array of 32 bytes specifying the Key	N/A	Hexadecimal values should use 0xXX format.
IDE_IV	Array of 12 bytes specifying the initial IV	N/A	Hexadecimal values should use 0xXX format.
KeyIndex	0 – 0xFFFFFFFF	0	Unique value to identify this Key/IV. Should be used in IDE TLPs to reference the key to be used for Encryption.

2.4.31 Config = LinkGen5

This command defines which physical layer capabilities will be supported in Device mode or which will be used in Host mode for 32 GTS.

Parameter	Values	Default	Comment
TSMod12Supported	Yes, No	No	When Yes, Modified TS1 TS2 Ordered Sets are supported.
TSModUsageMode0	Yes, No	No	When Yes, Modified TS Usage Mode 0 (PCIe) supported.
TSModUsageMode2	Yes, No	No	When Yes, Modified TS Usage Mode 2 (Alternate Protocol Negotiation) supported.
FullEqRequired	Yes, No	Yes	When Yes, Full Equalization required supported.
EqBypass	Yes, No	No	When Yes, Equalization bypass to highest rate supported.

Parameter	Values	Default	Comment
NoEqNeeded	Yes, No	No	When Yes, No Equalization Needed supported.
TxPrecodeRequest	0 – 1	1	This is the value the that Z4 requests the DUT to use in their Tx and will be transmitted in TS.
RxPrecodeSettings	FollowOurPrecodeRequest, PrecodingOn, PrecodingOff	FollowOurPrecodeRequest	FollowOurPrecodeRequest = the Rx Precode will match the Transmitter Precode Request. PrecodingOn = Turn on precode on Rx. PrecodingOff = Turn off precode on Rx.
TxPrecodeSettings	FollowDUTPrecodeRequest, PrecodingOn, PrecodingOff	FollowDUTPrecodeRequest	FollowDUTPrecodeRequest = will use the TxPrecodeRequest value sent by the DUT in the TSes.. PrecodingOn = Turn on precode on Tx. PrecodingOff = Turn off precode on Tx.
AlternateProtocolVendorID	0 – 0xFFFF	0x1E98	Specifies Alternate Protocol/Vendor ID.

NOTES:

- As host, choose one Modified TS Usage Mode – this will be the one that is advertised. As device can choose multiple Modified TS Usage Modes and will use this to decide how to respond to negotiation requests.
- Modified TS Usage Modes are only applicable when TSMod12Supported is set to Yes.
- As host, choose one Enhanced Link Behavior Control supported option (among FullEqRequired, EqBypass, NoEqNeeded).

Example:

```
Config=LinkGen5
{
    TSMod12Supported = Yes
    TSModUsageMode0 = Yes
    TSModUsageMode2 = No
    FullEqRequired = No
    EqBypass = Yes
```

```
NoEqNeeded = No
TxPrecodeRequest = 0
RxPrecodeSettings = PrecodingOn
TxPrecodeSettings = PrecodingOn
```

2.4.32 Config = SPDM

Config=SPDM is used for setting the selected algorithms, supported capabilities and other parameters that are used by SPDM processing mechanism. See 2.1.6.1 for details on generation of SPDM messages. Default values for all parameters are not set (equivalent to no responses).

The user is expected to set the SPDM configuration once at the start of the file.

Parameter	Values	Default	Comment
MEAS_CAP	NotSupportMeasResp, SupportMeasResp, SupportMeasRespWithSig	N/A	MEASUREMENTS response capabilities of the Responder.
KEY_EXCHANGE_PARAM1	NoMeasSummaryHashReq, TCB, AllMeas	N/A	Type of measurement summary hash requested in KEY_EXCHANGE request.
RSP_HANDSHAKE_IN_THE_CLEAR_CAP	set, cleared	N/A	The HANDSHAKE_IN_THE_CLEAR_CAP value of the responder.
REQ_HANDSHAKE_IN_THE_CLEAR_CAP	set, cleared	N/A	The HANDSHAKE_IN_THE_CLEAR_CAP value of the requestor.
RSP_MAC_CAP	set, cleared	N/A	If set, both devices support message authentication in a secure session.
RSP_ENCRYPT_CAP	set, cleared	N/A	If set, both devices support message encryption in a secure session.
OPAQUE_DATA_FMT	VendorDefined, General	N/A	Opaque data format selected by the responder.
MEAS_HASH_ALG	RAW_BIT_STREAM, TPM_ALG_SHA_256, TPM_ALG_SHA_384, TPM_ALG_SHA_512, TPM_ALG_SHA3_256, TPM_ALG_SHA3_384, TPM_ALG_SHA3_512, TPM_ALG_SM3_256	N/A	SPDM-enumerated hashing algorithm used for measurements selected by the responder.
BASE_ASYM_SEL	TPM_ALG_RSASSA_2048, TPM_ALG_RSAPSS_2048, TPM_ALG_RSASSA_3072, TPM_ALG_RSAPSS_3072, TPM_ALG_ECDSA_ECC_NIST_P256, TPM_ALG_RSASSA_4096, TPM_ALG_RSAPSS_4096, TPM_ALG_ECDSA_ECC_NIST_P384, TPM_ALG_ECDSA_ECC_NIST_P521, TPM_ALG_SM2_ECC_SM2_P256, EDDSA_ED25519, EDDSA_ED448	N/A	SPDM-enumerated asymmetric key signature algorithm selected by the responder.

Parameter	Values	Default	Comment
BASE_HASH_SEL	TPM_ALG_SHA_256, TPM_ALG_SHA_384, TPM_ALG_SHA_512, TPM_ALG_SHA3_256, TPM_ALG_SHA3_384, TPM_ALG_SHA3_512, TPM_ALG_SM3_256	N/A	SPDM-enumerated cryptographic hashing algorithm selected by the responder.
DHE_SEL	FFDHE2048, FFDHE3072, FFDHE4096, SECP256R1, SECP384R1, SECP521R1, SM2_P256	N/A	Responder-selected, SPDM-enumerated DHE group.
AEAD_SEL	AES_128_GCM, AES_256_GCM	N/A	Selected AEAD algorithm for secured SPDM processing.

Example:

```
Config=SPDM
{
    OPAQUE_DATA_FMT = VendorDefined
    BASE_ASYM_SEL = EDDSA_ED25519
    BASE_HASH_SEL = TPM_ALG_SHA_256
    DHE_SEL = SECP256R1
}
```

2.4.33 Config=SPDM_Key

This command allows to specify key and IV used for SPDM encryption.

Parameter	Values	Default	Comment
SPDM_Key	Array of bytes	N/A	Hexadecimal values should use 0xXX format. The size of the array must be equal to the size of the encryption key of the selected AEAD algorithm.
SPDM_IV	Array of bytes	N/A	Hexadecimal values should use 0xXX format. The size of the array must be equal to the size of the IV of the selected AEAD algorithm.
SPDM_KeyIndex	0-0xFFFFFFFF	N/A	Unique value to identify this Key/IV. Should be used in secure SPDM messages to reference the key to be used for encryption.

2.4.34 Config=MCTP

This config helps to specify MTU Size for MCTP.

Parameter	Values	Default	Comment
MTU_Size	Any Number	64	Defines Maximum MCTP Transmission Unit Size. MCTP Structure uses this size.

Example:

The next code defines the global MTU Size as 128 bytes.

```
Config=MCTP

{

      MTU_Size = 128

}
```

We may now use MTU_Size as a value.

The next code shows how to set MTU Size.

```
Structure=MCTP

{

    MCTPStructType = SMBus

    ; SMBUS specific info
    ; MCTP specific info
    MCTP_MSG_Type = NVMe_MI

    NVMeMI_MSG_ReqOResp = Request

    NVMeMI_MsgType = NVMe_MI_Cmd

    NVMeMI_CmdSlotId = CmdSlot0

    NVMeMI_Cmd_Opcode = Configuration_Set

    NVMeMI_Cmd_ConfigId = MCTP_TUSize

    NVMeMI_Cmd_PortId = 1

    NVMeMI_Cmd_UnitSize = MTU_Size
```

2.4.35 Config=L0p

Config=L0p is supported in FM and allows specifying L0p capabilities. See 2.17 PCIeFlitMode Command.

Parameter	Values	Default	Exerciser Default	Comment
L0pSupported	0-1	Unchanged	0	Set to 1 to enable L0p functionality
L0pRequestCount	1-3	Unchanged	1	Number of consecutive L0p requests
L0pLowPriorityRequestHandling	Ack/Nak/Ignore	Unchanged	Ack	Response to low priority requests
L0pHighPriorityRequestHandling	Ack/Nak/Ignore	Unchanged	Ack	Response to high priority requests

Example:

```
Config=L0p
{
    L0pSupported = 1
    L0pRequestCount = 1
    L0pLowPriorityRequestHandling = Nak
    L0pHighPriorityRequestHandling = Ignore
}
```

2.4.36 Config=LinkGen6

This command defines which physical layer capabilities will be supported in Device mode or which will be used in Host mode for 64 GTS.

Parameter	Values	Default	Comment
xPrecodeRequest64GTS	0 – 1	1	This is the value the that M616 requests the DUT to use in their Tx and will be transmitted in TS.
RxPrecodeSettings64GTS	FollowOurPrecodeRequest, PrecodingOn, PrecodingOff	FollowOur PrecodeRequest	FollowOurPrecodeRequest = the Rx Precode will match the Transmitter Precode Request. PrecodingOn = Turn on precode on Rx. PrecodingOff = Turn off precode on Rx
TxPrecodeSettings64GTS	FollowDUTPrecodeRequest, PrecodingOn, PrecodingOff	FollowDUTPrecodeRequest	FollowDUTPrecodeRequest = will use the TxPrecodeRequest value sent by the DUT in the TSes.. PrecodingOn = Turn on precode on Tx. PrecodingOff = Turn off precode on Tx.

Example:

```
PCIeFlitMode=True
Config=LinkGen6
```

```
{
TxPrecodeRequest64GTS = 0
RxPrecodeSettings64GTS = PrecodingOn
TxPrecodeSettings64GTS = PrecodingOn
}
```

2.4.37 Config=CXL_CM_IDE

This command allows to set IDE encryption parameters and Start/Stop IDE encryption of CXL 2.0 Cache Mem packets with set parameters.
NOTE: Only available on Summit M616 in CXL 2.0 IDE Mode.

Parameter	Values	Default	Comment
CM_IDEControl	Start Stop	N/A	Switches link to IDE mode (starts encryption) or to Standard mode (stops encryption).
ReplyToIDEStart	Yes No	N/A	If set to Yes sends automatic reply when IDE Start is received from remote side. Ignores CM_IDEControl value if set. *Device only
CM_IDEMode	Containment Skid	N/A	*Must be set and can only be set when CM_IDEControl = Start or ReplyToIDEStart = Yes
PCRCDisabled	Yes No	N/A	If Yes PCRC generation is disabled, and MAC calculation does not include PCRC. *Must be set and can only be set when CM_IDEControl = Start or ReplyToIDEStart = Yes
CM_IDEKey	Array of 32 bytes specifying the Key	N/A	Hexadecimal values should use 0xXX format. *Must be set and can only be set when CM_IDEControl = Start or ReplyToIDEStart = Yes
CM_IDE_IV	Array of 12 bytes specifying the initial IV	N/A	Hexadecimal values should use 0xXX format. *Must be set and can only be set when CM_IDEControl = Start or ReplyToIDEStart = Yes
TMACMode	Yes No	N/A	Enables TMAC mode.
FlitsBeforeTMACCount	0 - 255	N/A	In TX: Count of protocol flits needs to be sent before the insertion of TMAC. In RX: Count of protocol flits needs to be received before the arrival of TMAC.
KeyRefreshTimeControl	0 – 0xFFFFFFFF	N/A	Minimum number of flits that the transmitter needs to block transmission of protocol flits after IDE.Start has been sent.
TruncationTransmitDelayControl	0 - 255	N/A	Configuration parameter to account for the potential discarding of any precomputed values by the receiver. This parameter feeds into the computation of the minimum number of IDE.Idle flits that the Transmitter

Parameter	Values	Default	Comment
			needs to send after sending a truncated MAC flit.
KeyRefreshTimeCapability	0 – 0xFFFFFFFF	N/A	Number of IDE.Idle flits the receiver needs before it is ready to receive protocol flits after IDE.Start is received when operating in 68B Flit mode.
TruncationTransmitDelayCapability	0 - 255	N/A	Number of IDE.Idle flits the receiver needs before it is ready to receive protocol flits after a Truncated MAC is received when operating in 68B Flit mode.

Example:

```
Config=CXL_CM_IDE
      {
      KeyRefreshTimeControl = 10
      TruncationTransmitDelayControl = 10
      KeyRefreshTimeCapability = 10
      TruncationTransmitDelayCapability = 10

      FlitsBeforeTMACCount = 1

      CM_IDE_IV = (80 00 00 00 00 00 00 00 00 00 00 01)
      CM_IDEKey = ( 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11
0x11 0x11 0x11 0x11 0x11
                    0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11
0x11 0x11 0x11 0x11 0x11)

      TMACMode = Yes
      CM_IDEMode = Containment
      PCRCDisabled = No

      CM_IDEControl = Start
      }

      Loop=Begin
      {
      Count=5
      }

      Packet=CXL_Cache
      {
      CXLCacheType=H2D_CacheReq
      H2DReqOpcode = SnpData
      }

      Loop=End

      Config=CXL_CM_IDE
      {
      CM_IDEControl = Stop
      }
```

2.5 Wait Command

This command yields script execution until condition specified is true or timeout expires.

Parameter	Values	Default	Comment
Timeout		0	Timeout in nanoseconds. **0** means infinite timeout.
Display	Any string literal		Message displayed during the waiting in status bar
Count	1 – 65535	1	Repeats wait specified number of times.

NOTE: Setup of wait condition for Summit Z3/Z4 Exerciser during script execution may take tens of microseconds. The Summit Z3/Z4 Exerciser will miss events that are coming during wait condition setup.

2.5.1 Wait = TLP

This command waits for a TLP that matches the defined condition. Only TLP Header fields can be specified. All parameters from **Packet = TLP** command (see Page 4) are valid, except **PSN**, **ECRC**, **LCRC** and **Payload** parameters.

TLP Header fields can be masked using the following format:

`0x0XAXX` For hexadecimal values
`0b0001XX` For binary values

Example:

This command waits infinitely for a Configuration Write request to registers from 0x1000 to 0x1FFF.

```
Wait = TLP {
      TLPType = CfgWr
      Register = "0x1XXX"
      Timeout = 0
}
```

Note 1: When using wait commands in the Z3/Z4 script keep in mind that it takes time for the command to be configured in the Bus Engine. Thus when running a script that starts with a wait on a packet command, the DUT should not issue the packet within at least 100 microseconds. However, when the DUT issues its TLP as a response to the TLP in the script, the Z3/Z4 takes care of properly setting up the wait condition prior to issuing it's TLP, so the wait would always work properly for a construct like this:

```
packet=TLP
{
 ...
}

wait=TLP   ; send as the response to the TLP from the script
{
 ...
}
```

Note 2: Whenever a script is relying on the Tag value in the Completions returned by the DUT in response to the Requests sent from the script, the Manual Tag generation policy should be selected in the Generation Options (by checking the "Disable Automatic Tag generation" radio button in the Integrity tab). This way the script can place specific values in the Tag field of the Requests it generates and expect the Request with that value of the Tag to be transmitted on the link and thus that particular value of the Tag returned in the Completion(s) sent by the DUT. This includes:

- Waiting on a Completion with a certain Tag value;
- Branching on a Completion with a certain Tag value;
- Using StoreData parameter on a Read Request TLP, including for further Break condition for a Loop instruction.

2.5.2 Wait = MultiTLP

This command waits for a TLP which type matches one of the specified types. Only a set of the TLP types can be specified.

Parameter	Values	Default	Comment
MultiTLPTypes	(T1, T2…)	No TLP Types	The value is a list of comma separated TLP Type strings (same strings as defined for TLPType parameter of the Packet=TLP command)

Example:

```
Packet=TLP
{
    TLPType=CfgRd0   ; Send a read of a register in the Configuration space
    ...
}
```

; The DUT might return a CplD with configuration data or a Cpl with UR status for an unimplemented portion
; of the Configuration Space

```
Wait=MultiTLP
{
    MultiTLPTypes=( Cpl, CplD )
}
```

2.5.3 Wait = DLLP

This command waits for a DLLP that matches the defined condition. All parameters from **Packet = DLLP** command (see Page 30) are valid, except the **CRC** field.

DLLP fields can be masked using the following format:

`0x0XAXX`	For hexadecimal values
`0b0001XX`	For binary values

Example 1:

This command waits for Ack DLLP.
The execution continues when Ack DLLP is received or after the 256 ns timeout expires.

```
Wait = DLLP {
      DLLPType = Ack
      Timeout = 256
}
```

Example 2:

This command waits for a Vendor DLLP with the Least Significant Bit of the vendor specific data set.
The execution continues when such DLLP is received or after the 256 ns timeout expires.

```
Wait = DLLP {
      DLLPType = Vendor
      VendorSpecific = "0bXXXXXXXXXXXXXXXXXXXXXXXX1"
      Timeout = 256
}
```

2.5.4 Wait = CXL_Cache, Wait = CXL_Mem

These commands wait for the CXL.cache/mem packet that matches the defined condition. All the parameters from **Packet = CXL_Cache** / **Packet = CXL_Mem** commands are valid, except **AutoIncrementAddress**, **AddressIncrementValue**, **AutoIncrementTag**.
This command is only available for Z516 and M616 Exerciser and requires CXL Mode to be enabled in generation options.

2.5.5 Wait = LinkCondition

Parameter	Values	Default	Comment
Conditions	SKIP TS1 TS2 FTS DLLP TLP COMMA LinkAlive CXLLinkLevelInitDone CXLFirstCRCCleanFlit CXLRetryReq CXLCRCErrorDetected CXLLlcrd CXLInitParam CXLRetryAck		The list of conditions for which to wait. **Note:** Condition LinkAlive waits for bus to be at L0 state.

Example 1:

This command waits for the COMMA symbol in incoming traffic.
The script execution continues when the COMMA symbol is received or after the 1024 ns timeout expires.

```
Wait = LinkCondition {
      Conditions = ( COMMA )
      Timeout = 1024
}
```

Example 2:

This command waits for a Training Sequence Ordered Set (TS1 or TS2) in incoming traffic with the **HotReset** bit asserted in the **TrainingControl** bits.
The script execution also continues after the 1024 ns timeout expires.

```
Wait = LinkCondition {
      Conditions = ( TS1, TS2 )
      Timeout = 1024
}
```

2.5.6 Wait = Payload

This command waits for TLP payload match.

Parameter	Values	Default	Comment
Payload			Mask and Match up to four DWORDs of TLP payload from the beginning of payload.
Payload@<offset>			Mask and Match up to four DWORDs of TLP payload starting from **<offset>** offset from the beginning of payload.

Up to four DWORDs of the payload can be specified.

Payload DWORDs are listed in brackets and create Mask/Match definitions that are applied against incoming TLPs payload to determine the match. The don't care parts of the DWORDs are defined by using 'X' in hexadecimal or binary representation, for example 0x1X34X678 or 0b1X1X0X1X1X1X1X1X1X1X1X0X1X1X1X1X.

Example 1:
This command waits for a TLP with first data payload 0x12345678.

Script execution continues when a TLP with the specified payload is received or after the 100 microseconds timeout expires.

```
Wait = Payload {
Payload  = ( 0x12345678 )
Timeout = 100000
}
```

Example 2:

This command waits for a TLP with a data payload that matches the following criteria:
1) The 1st DWORD's upper-most word must have **0xABCD**.
2) The 4th DWORD's lowest word must have **0x1234**.
3) The 2nd DWORD is insignificant.
4) The 3rd DWORD must have the most significant bit set to 1 and the lease significant bit set to 0.

Script execution continues when a TLP with specified payload is received or after the 1 millisecond timeout expires.

```
Wait = Payload {
Payload = ( 0xABCDXXXX, 0xXXXXXXXX, 0b1XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX0,
0xXXXX1234 )
Timeout = 1000000
}
```

Example 3:

This command waits for a TLP with a data payload that matches the following criteria:
1) The 3rd DWORD's upper-most word must have **0xABCD**.
2) The 9th DWORD's lowest word must have **0x1234**.
3) The 10th DWORD's upper-most byte must have 0x56.
Script execution continues when a TLP with specified payload is received or after the 5 milliseconds timeout expires.

```
Wait = Payload {
Payload@2 = ( 0xABCDXXXX )
Payload@8 = ( 0xXXXX1234, 0x56XXXXXX )
Timeout = 5000000
}
```

2.5.7 Wait = User

This command waits for user input. The script execution would continue when user resumes the script from PCIe Protocol Suite™ software UI.

Example:

This example pauses the script execution and displays the message to the user.

```
Wait = User {
      Display = "Now you can continue"
}
```

2.5.8 Wait = FastTransmitIdle

This command synchronizes execution of different **FastTransmit** blocks and synchronizes high-performance and regular generation.

During high-performance generation, the **Wait=FastTransmitIdle** command waits for the current **FastTransmit** block to finish execution. If the system is not executing a **FastTransmit** block when the script calls this command, the **Wait** condition is immediately satisfied.

Note: Wait = argument value should not exceed 32 bits so the max could be 0xFFFFFFFF(hex) or 4294967295 in decimal.

2.5.9 Wait = SMBus

This command waits for an incoming SMBus packet that matches the defined condition. It would be mostly used for waiting for Response packets after transmitting the Request packet.

The command is only available on SMBus-capable Summit Z3/Z4 Exercisers and Host Platforms.

The fields for this Wait command represent the fields in the SMBus portion of the incoming packet header:

Parameter	Values	Default	Comment
DestSlaveAddr	0 – 0xFF	0	SMBus Destination Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to zero for SMBus write transaction as the MCTP binding spec mandates.
CommandCode	0 – 0xFF, MCTP	0	SMBus command code. The MCTP constant sets it to 0x0F for MCTP over SMBus and should be used in most cases.
ByteCntSMBus	0 – 255	0	Should be set to the exact length corresponding to the desired packet.
SrcSlaveAddr	0 – 0xFF	0	SMBus Source Slave Address. Note this represents the whole 8-bit address, bit 0 of which will be set to 1 as the MCTP binding spec mandates.

Example:

```
Config=SMBus
      {
          I2CSlaveAddress = 0x14 ; Set the source slave address to accept the
                                   response packet I2CSpeed = 100KHz
      }
Packet=SMBus
      {
          DestSlaveAddr = 0x3A  ; Send the request packet to SMBus DUT address
0x3A
          CommandCode=MCTP
          ByteCntSMBus = 10
          SrcSlaveAddr = 0x14

          MCTP_HDR_Version = 1
          MCTP_HDR_DestEPID = 0
          MCTP_HDR_SrcEPID = 0x10
          MCTP_HDR_SOM = 1
          MCTP_HDR_EOM = 1
          MCTP_HDR_TO = 1
          MCTP_MSG_Type = MCTP_Control
          MCTP_MSG_RqBit = Yes

          MCTP_MSG_CmdCode = SET_EP_ID
          MCTP_CMD_SEID_Operation = SetEID
          MCTP_CMD_SEID_EPID = 0xAB
```

```
      }


Wait=SMBus
      {
          CommandCode=MCTP

          SrcSlaveAddr = 0x3A ; Wait for the response packet from the DUT
      slave
                                address with a timeout of 100 milliseconds
          Timeout = 100000000
      }
```

2.5.10 Wait = RawLtssmDone

This command waits for the finish of execution of previously started Raw LTSSM block. Should be used after RawLtssm = Start if there is a need to synchronize with following traffic or another Raw LTSSM block.

See 2.16 Raw Ltssm Commands for more details.

See 2.16.3 RawLtssm Examples for examples of usage.

2.5.11 Additional "Wait" Modifiers

1. **Wait = <number>**

   Unconditionally yields script execution for the specified number of nanoseconds.

   Example:

   ```
   Wait = 500
   ```

2. **Wait = <Text>**

   Equivalent to

   ```
   Wait = User {
         Display = <Text>
   }
   ```

   Example:

   ```
   Wait = "Press the button to continue script execution"
   ```

   A count parameter can be applied to this command, which causes it to wait for that number clicks on the user input button.

2.6 Branch Command

This command enables/disables the branch procedure for the condition specified.

2.6.1 Branch = <condition>

This command enables the branch procedure for the condition specified. The conditions are the same as in the **Wait** command (see Section 2.5,

Parameter	Values	Default	Comment
MEAS_CAP	NotSupportMeasResp, SupportMeasResp, SupportMeasRespWithSig	N/A	MEASUREMENTS response capabilities of the Responder.
KEY_EXCHANGE_PARAM1	NoMeasSummaryHashReq, TCB, AllMeas	N/A	Type of measurement summary hash requested in KEY_EXCHANGE request.
RSP_HANDSHAKE_IN_THE_CLEAR_CAP	set, cleared	N/A	The HANDSHAKE_IN_THE_CLEAR_CAP value of the responder.
REQ_HANDSHAKE_IN_THE_CLEAR_CAP	set, cleared	N/A	The HANDSHAKE_IN_THE_CLEAR_CAP value of the requestor.
RSP_MAC_CAP	set, cleared	N/A	If set, both devices support message authentication in a secure session.
RSP_ENCRYPT_CAP	set, cleared	N/A	If set, both devices support message encryption in a secure session.
OPAQUE_DATA_FMT	VendorDefined, General	N/A	Opaque data format selected by the responder.
MEAS_HASH_ALG	RAW_BIT_STREAM, TPM_ALG_SHA_256, TPM_ALG_SHA_384, TPM_ALG_SHA_512, TPM_ALG_SHA3_256, TPM_ALG_SHA3_384, TPM_ALG_SHA3_512, TPM_ALG_SM3_256	N/A	SPDM-enumerated hashing algorithm used for measurements selected by the responder.
BASE_ASYM_SEL	TPM_ALG_RSASSA_2048, TPM_ALG_RSAPSS_2048, TPM_ALG_RSASSA_3072, TPM_ALG_RSAPSS_3072, TPM_ALG_ECDSA_ECC_NIST_P256, TPM_ALG_RSASSA_4096, TPM_ALG_RSAPSS_4096, TPM_ALG_ECDSA_ECC_NIST_P384, TPM_ALG_ECDSA_ECC_NIST_P521, TPM_ALG_SM2_ECC_SM2_P256,	N/A	SPDM-enumerated asymmetric key signature algorithm selected by the responder.

Parameter	Values	Default	Comment
	EDDSA_ED25519, EDDSA_ED448		
BASE_HASH_SEL	TPM_ALG_SHA_256, TPM_ALG_SHA_384, TPM_ALG_SHA_512, TPM_ALG_SHA3_256, TPM_ALG_SHA3_384, TPM_ALG_SHA3_512, TPM_ALG_SM3_256	N/A	SPDM-enumerated cryptographic hashing algorithm selected by the responder.
DHE_SEL	FFDHE2048, FFDHE3072, FFDHE4096, SECP256R1, SECP384R1, SECP521R1, SM2_P256	N/A	Responder-selected, SPDM-enumerated DHE group.

), except **User**. The parameter list is the same as the **Wait** command, except for the **Timeout**, **Display**, and **Count** parameters.

The following lists the additional parameters for the **Branch = <Condition>** command:

Parameter	Values	Default	Comment
BranchName	Any string literal		Name of the branch. Must be specified if this branch is to be disabled later.
ProcName	Any string literal		Name of the procedure to execute when branch conditions are met.
RunConcurrently	Yes No	No	If the value is Yes, the Exerciser firmware runs the branch procedure concurrently with the main script procedure and other concurrent branches, as a separate task within the Exerciser firmware.

ProcName parameter is mandatory.

BranchName parameter could be omitted if you do not plan to disable the branch later in the script.

The procedure that handles the branch condition must be defined before the **Branch = <Condition>** command (see Page 35).

Example:

```
...
Proc = Begin {
      ProcName = "Procedure1"
}
...

Proc = End

; The following statement specifies that if Delimiter, Disparity
; or Symbol error occurs, then the code declared in "Procedure1"
; should be executed.

Branch = Error {
      BranchName = "SomeErrorBranch"
      ProcName = "Procedure1"
      Errors = (Delimiter, Disparity, Symbol)
}
...

; Disable the branch "SomeErrorBranch" that is specified above.

Branch = Disable {
      BranchName = "SomeErrorBranch"
}
...
```

2.6.2 Branch = Disable

This command disables the branch procedure that was previously enabled.

Parameter	Values	Default	Comment
BranchName	Any string literal		Name of the branch

Branch with the name specified in **BranchName** parameter must be defined.

2.7 Proc Command

This command declares the procedure to be executed for the **Branch** command. Procedure declaration must precede its usage in the **Branch** statement.

2.7.1 Proc = Begin

This command declares the start point of the procedure.

Parameter	Values	Default	Comment
ProcName	Any string literal		Name of the procedure

2.7.2 Proc = End

This command declares the end point of the procedure.

2.8 Loop Command

This command causes the Summit Exerciser™ BusEngine™ to re-execute a block of commands a predefined number of times.

Note: Loops require up to 1 us to branch to the beginning of the loop. During this time, script execution is paused. Internally generated packets, such as SKIP ordered sets, Ack/Nak DLLP packets, and flow control updates, still occur as programmed.

Loops can be nested up to 8 deep.

2.8.1 Loop = Begin

This command marks the beginning of the loop.

Parameter	Values	Default	Comment
Count	0 – 65535 Infinite		Specifies how many times to repeat the loop. Setting Count to 0 causes an infinite loop.

2.8.2 Loop = End

This command marks the end of the loop.

Example:

```
Loop = Begin { count = 10 }
      Packet = TLP { TLPType = CfgRd0  Length = 1  Register = 0 }
Loop = End
```

2.8.3 Loop = Break

This command allows breaking the Loop execution when contents of a memory location in Device Memory Space or Host Memory Region match the specified value.
This command is only available for Summit Exerciser Z3/Z4.

Parameter	Values	Default	Comment
Location	Cfg Mem64 Mem32A Mem32B IOA IOB		Specifies the memory region where the data should be monitored from. For the case of Device emulation the memory region is mapped to the address space. For the case of Host emulation only **Mem64**, **Mem32A**, **Mem32B** are applicable and should correspond to a Host Memory Region enabled in the Transactions tab of the Generation Options.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies offset in bytes from the beginning of memory region specified in the **Location** parameter.
FieldSize	Byte Word Dword	0 (one of the values has to be specified)	Specifies the size of the variable to be monitored at the address specified in the **Offset** parameter.
Endian	Little Big	Little	Specifies the byte ordering for the variable for the case of **Word** or **Dword** FieldSize.
FieldAction	CompareValue MaskAND MaskOR	CompareValue	Specifies the match criteria for the variable at the specified memory location to be used for decision to break the loop. **CompareValue** – compares the value of the variable with the **Result** and breaks the loop if it matches; **MaskAND** – performs bitwise AND operation with value of the variable and the value of the **Mask**, and breaks the loop if the produced value matches the **Result**; **MaskOR** – performs bitwise OR operation with value of the variable and the value of the **Mask**, and breaks the loop if the produced value matches the **Result**.

Parameter	Values	Default	Comment
Mask	Any number	0	Specifies the value of the **Mask** to be used for **MaskAND** or **MaskOR** action.
Result	Any number	0	The resulting value to be used in comparison specified by **FieldAction**.

This instruction allows using Loops in Host emulation in order to monitor the content of DUT's Configuration or Memory registers and continue execution only when conditions are meant (certain bits are set or cleared etc.). This could be done along with the StoreData parameter in the TLP transmission.
For the case of Device emulation it can be used the same way as above to monitor for changes in Host memory.
For both Host and Device emulation it can be used to monitor changes in Z3/Z4's own memory space or Host Memory Region which are modified by the incoming Memory Writes.

Note 1: When using StoreData by itself or in conjunction with the LoopBreak instruction, caution has to be taken in order not to use offset/location that overlaps with any other data residing in the exerciser's memory (like data created with Structure instructions for NVMe Host emulation, for example).

Note 2: Whenever a script is relying on the Tag value in the Completions returned by the DUT in response to the Requests sent from the script, the Manual Tag generation policy should be selected in the Generation Options (by checking the "Disable Automatic Tag generation" radio button in the Integrity tab). This way the script can place specific values in the Tag field of the Requests it generates and expect the Request with that value of the Tag to be transmitted on the link and thus that particular value of the Tag returned in the Completion(s) sent by the DUT. This includes:

- Waiting on a Completion with a certain Tag value;
- Branching on a Completion with a certain Tag value;
- Using StoreData parameter on a Read Request TLP, including for further Break condition for a Loop instruction.

Example 1:

The following example is taken from the nvme_init.peg NVMe Host Emulation sample script

```
; Poll for the Ready status
      Loop=Begin
            {
                Count=50000
            }

            packet="Temp_OneDwordRead"
            {
                Address = ( CONTROLLER_REGISTERS_BASE + 0x1C ) ; Controller
            Status Register

                StoreData = ( FROM_MEM64, 0 ) ; the read data to be stored at
            offset 0 in Mem64 HostMemRegion
            }

      ; using the Break instruction to break the loop when Ready bit is set
      Loop=Break
```

```
        {
      Location=Mem64  ; the variable to monitor is at offset 0 in Mem64 –
      where StoreData puts the value
          Offset = 0
          FieldSize = Dword  ; compare the Dword
          Endian = Little
          FieldAction = MaskOR ; Mask the bit 0, which is the Ready bit
          Mask = 0x1

      Result = 0x1  ; the condition will be met when the ready bit is set
      }

Loop=End
```

2.9 Repeat Command

This command causes one or more commands to be repeated. This is not implemented as a branch instruction in the BusEngine™, but is a replication of commands during script compilation in the software.

This allows back-to-back execution of these commands with as little as 0 symbol times of IDLE traffic between them.

This command increases the size of the script object that is downloaded to the Summit Exerciser™ and increases download time accordingly.

2.9.1 Repeat = Begin

This command marks the beginning of the code being repeated.

Parameter	Values	Default	Comment
Count	1 – 65535		Values of Infinite and 0 are not supported
Counter			See the description below (2.9.2)
Variable 1			See the description below (2.9.3)
Variable 2			See the description below (2.9.3)

2.9.2 Counter Parameter

Any string literal can be used for the **Counter** parameter.

The value of the **Counter** parameter can be used within the **Repeat** statement (i.e., between **Repeat=Begin** and **Repeat=End**) in arithmetic expressions for any parameter, except the parameters that require the array data type (such as Payload for TLP packet).

The value of the **Counter** parameter changes from 0 to the value of the **Count** parameter minus one.

Arithmetic expressions must be included in round brackets (parentheses).

The operators are: +, -, *, /, <<, >>, &, |, ~.

Example 1:

Within this repeat, **ppp** can be used in arithmetic expressions for any packet field. The value of **ppp** changes from 0 to 3 in the example.
The **Tag** parameter accepts the values **0x10, 0x11, 0x12**, and **0x13**.
The **AddressHi** parameter accepts the values **0x00400000, 0x00400001, 0x00400001**, and **0x00400002.**

```
Repeat = Begin { Count = 4 Counter = ppp }

Packet = TLP {
            TLPType = MRd64
            Tag = ( ppp + 0x10 )
            AddressHi = ( 0x400000 + 4 / ( 5 - ppp ) )
}

Repeat = End
```

Example 2:

The following example shows the usage of the counters in nested repeats.
The counter **qqq** is used for the outer repeat.
The counter **www** is used for the inner repeat.
Packet = TLP in the inner repeat uses both counters to construct the **AddressHi** parameter.

```
      Repeat = Begin { Count = 3 Counter = qqq }

      Packet = DLLP {
            DLLPType = Ack
            AckNak_SeqNum = ( qqq + 1 )
      }

      Packet = DLLP {
            DLLPType = Ack
            AckNak_SeqNum = ( 0xf & ~qqq )
      }

            Repeat = Begin { Count=4 Counter = www }

            Packet = TLP {
                  TLPType = MRd64
                  AddressHi = ( 0x400000 + www * 4 + qqq ) )
            }

            Repeat = End

      Repeat = End
```

Example 3:

The following example shows the usage of the counter in Payload. When arithmetic expressions with counter are used inside the Payload field, square brackets [] are used to separate each DWORD of the payload.

```
Config=Definitions
{
    MY_CONSTANT = 10
}
repeat = Begin  { Count = 10    Counter = i }
; one TLP with variable fields is sent
packet = TLP
{
 TLPType = MWr32

 LastDwBe = 0xF
 FirstDwBe = 0xF
 Address = ( i << 24  )
 Length = 2
 Payload = (  [ ( ( ( i + MY_CONSTANT ) * 2 ) << 16 ) | ( i*2 ) ]   [ 0xAB <<
i ]   )  ; Two DWORDs of the Payload are defined using arithmetic expressions
with counter and constant

 Tag = ( i & 0xFF )
}
```

```
repeat = End
```

This command marks the end of the code being repeated.

Example:

```
Repeat = Begin { count = 10 }
      Packet = TLP { TLPType = CfgRd0  length = 1 register = 0 }
Repeat = End
```

2.9.3 Variable parameters

Variable parameters are available for the Z416 and Z3-16 Exercisers. Up to two variables can be defined at each level of the Repeat statement. The definition format is as follows:

Variable1 = (VariableName, VariableLocation)

VariableName san be any string literal up to 10 characters long.
VariableLocation defines the location of the variable within an Address Space or a Host Memory Region on the Exerciser.
The format of the definition is the same as for Field Substitution (see 2.1.1.8 TLP Field or Payload Substitution).

The purpose of the variables is to use a value stored in a certain memory location of the Address Space or a Host Memory Region, and perform a set of arithmetic operations with this value to arrive at the resulting value to be assigned to a certain field in the TLP before this TLP is transmitted from the script. This is while preserving the same script syntax as with the Counter parameter (see above) and along with (mixed) with the whatever Counter parameters are defined.
The difference is that the values of the Counter parameters are pre-calculated by the software when the script is interpreted, and the expressions are resolved to numeric values. And for the Variable parameters the final values of the expressions are calculated on the Exerciser when the script is being executed.

Variables can be used for both Device and Host Emulation. However, in case of Device Emulation the script usually works in accordance with some Host software that calculate desired values and place them at locations in Device Emulation address spaces, so that simple Field Substitution (see 2.1.1.8 TLP Field or Payload Substitution) would suffice.

In case of Host Emulation a value can be placed in the memory location of the Host Memory Region by three means:

- Using the StoreData parameter for the Memory or Configuration Read TLP (see 2.1.1 Packet = TLP);
- By the DUT Endpoint directly writing the location of the Host Memory Region over PCIe;
- By using the AddressSpace = Write command (see 2.12.2 AddressSpace = Write) or manual writing from PCIe Protocol Suite software.

Important: in order for the Variables to work, the corressponding Address Space or Host Memory Region have to be enabled in the Transaction tab of the
Exerciser's Generation Options.

Note: Variables cannot be used for the fields within Payload.

Example:

Writing the MSI-X table on the DUT Endpoint based on the Table Offset read from the DUT's Configuration Space.

```
Config=Definitions
{
    ASSIGNED_BAR_ADDRESS = 0xAB000000
}

Packet = TLP
{
     TLPType = CfgRd0
     Tag = 1
     FirstDwBe = 0xf
```

```
     Register = 0x98          ; address of the register in the MSI-X
                                capability structure of the
                              ; DUT containing the Table Offset/ Table BIR
                                values

     Length = 1

     StoreData = (FROM_MEM32_A, 0x100); Place the data read at offset
                                        0x100 in the Mem32A Host Memory
Region
                                      ; (Has to be enabled in the Generation
Options!)
}

wait=10000

Repeat=Begin
{
    Count=16
    Counter = i
    Variable1 = ( myVar1, (FROM_MEM32_A, 0x100) )   ; Points to the location
where
                                                      Table Offset/ Table BIR
value
                                                      was stored
}

Packet=TLP
{
    TLPType=MWr32
    Length = 4
    LastDwBe = 0xF
    FirstDwBe = 0xF
    Address = ( ASSIGNED_BAR_ADDRESS + ( myVar1&0xFFFFFFF8) + (i*16) ) ; Masks
out the BIR portion, adds the resulting Table Offset value
                                                                       ; to
the previously assigned BAR address, and repeats this 16 times
                                                                       ; for
each vector location in the MSI-X table

    Payload = ( [ 0xFE + (0x100*i) ] 0 0xA0400000 0 )
}

Repeat=End
```

2.10 Template Command

This command creates a template for a packet that can be used in the **Packet** command. The fields specified in the **Template** command may be overridden in the **Packet** command.

It is obligatory to specify the **CXLCacheType** parameter for **template = CXL_Cache** and **CXLMemType** parameter for **template = CXL_Mem**. Moreover, after packet type is specified, it cannot be changed in the corresponding packet instruction, that refers to this template.

Example 1:

The following example issues three Memory Read requests.

```
Template = TLP {
      Name = "TestPacket"
      Type = MRd32
      TC = 0
      Tag = 0
      RequesterID = (1:0:0)
      Length = 64
      Address = 0
}

Packet = "TestPacket" {
}

Packet = "TestPacket" {
      Address = 64
}

Packet = "TestPacket" {
      Address = 128
}
```

Example 2:

The following example shows nested templates (i.e. when one template is based on another template).

```
; First define the template "SomeTlp3" for TLP packet.

Template = TLP {
      Name = "SomeTlp3"
      TLPType = MRd32
      RequesterID = (0:1:2)
      Length = 0x40
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x10000
}

; The template "SomeTlp4" is based on the template "SomeTlp3"
; with Address overridden.

Template = "SomeTlp3" {
      Name = "SomeTlp4"
      Address = 0x10040
}

; This TLP packet has Address parameter equal to 0x10000.

Packet = "SomeTlp3" {
      Length = 0x80
}

; This TLP packet has Address parameter equal to 0x10040.

Packet = "SomeTlp4" {
      Length = 0x80
}
```

2.10.1 Template = SPDM

Template=SPDM is not a regular template. It is used to create a named block of data that can later be used to set the values of the SPDM payload fields. This is done by including the template name in an array when defining the SPDM payload structure data (see 2.1.6.1 for details on generation of SPDM messages).

Parameter	Values	Default	Comment
Name	Quoted string	N/A	Specifies the data block name that is used to identify it. Must be unique.
SPDMTemplateType	AlgStruct, ExtAlgStruct, OpaqueData, OpaqueElement,	N/A	
SPDMTemplateData	Array of bytes	N/A	Implements a generic way to specify a payload without specifying a template type.

The other fields are type-specific.

Example:

```
Template=SPDM
{
    Name="OpaqueElement1"
    SPDMTemplateType = OpaqueElement
    StandardID = 0x02
    VendorIDLength = 2
    SPDMVendorID = 0x0123
    OpaqueElementDataLen = 2
    OpaqueElementData = (0xFE 0xDC)
}
Template=SPDM
{
    Name="OpaqueData1"
    SPDMTemplateType = OpaqueData
    TotalElements = 1
    OpaqueList = (OpaqueElement1)
}
```

2.11 Include Command

This command includes the Summit Exerciser™ script file inline. All commands in the included file are executed, with the exception of the **Config = General** command.

The format of this command is following:

```
Include = <file_path>
```

where **file_path** is a path to the file to be included. If **file_path** is not a fully qualified path, then the relative path to the current script file would be used.

Example 1:

In this example, all commands from the **included1.peg** file would be executed first, then all commands from the **included2.peg** file would be executed, and then the 32-bit Memory Read TLP would be sent.

```
Include = "included1.peg"     ; All packets from included1.peg file
                              ; would be inserted here.
Include = "included2.peg"     ; All packets from included2.peg file
                              ; would be inserted here.

Packet = TLP           ; Sending 32-bit Memory Read TLP request
{
      TLPType = MRd32  ;  Memory Read request (32 bit)
      TC = 0x7         ; Traffic class is 7.
      TD = 0x1         ; TLP digest is present.
      EP = 0x0         ; TLP is not poisoned.
      Address = 0x1000 ; Reading from address 1000h of memory space
      Length = 0x40    ; Reading 40h DWORDs
}
```

Example 2:

The first command of this example includes all commands from the file **c:/Testing/included1.peg**.
If we assume that the current script is located in the folder **c:/Testing/TLP**, then the second command of this example includes all commands from the file **c:/Testing/TLP/included2.peg**.
If we assume that the current script is located in the folder **c:/Testing/TLP**, then the third command of this example includes all commands from the file **c:/Testing/included3.peg**.

```
Include = "c:/Testing/included1.peg"; All packets from included1.peg
                                    ; file would be inserted here.
Include = "included2.peg"           ; All packets from included2.peg
                                    ; file would be inserted here.
Include = "../included3.peg"        ; All packets from included3.peg
                                    ; file would be inserted here.
```

2.12 AddressSpace Command

This command reads/writes the Summit Exerciser™ memory region.

Summit Exerciser maps Memory and IO address spaces to its internal memory region according to Base Address Registers (BAR) specified in the Configuration Address Space.

Summit Exerciser uses its memory regions when processing Memory, IO, and Configuration TLP requests.

Summit Exerciser maps Configuration address space to its internal memory region (**Cfg**).

Summit Exerciser supports one 64-bit Memory region, two 32-bit Memory regions, and two IO Memory regions.

Maximum address space sizes supported by Summit Exerciser are as follows:

Address Space	Size
Configuration	4 KB
32-bit memory	128 MB
64-bit memory	512 MB
IO	256 MB

Mapping of BARs to Summit Exerciser memory regions:

Memory Region	BAR
Mem64	First BAR that defines 64-bit Memory Address Space
Mem32A	First BAR that defines 32-bit Memory Address Space
Mem32B	Second BAR that defines 32-bit Memory Address Space
IOA	First BAR that defines IO Address Space
IOB	Second BAR that defines IO Address Space

In order to properly respond to Memory and IO TLP requests, the Configuration space must be written to the Summit Exerciser first.

2.12.1 AddressSpace = Read

This command reads specified memory region from Summit Exerciser and stores it in specified file.

Parameter	Values	Default	Comment
Location	Cfg Mem64 Mem32A Mem32B IOA IOB		Specifies the memory region from which to read. The memory region is mapped to the address space according to the rules described above.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies offset in bytes from the beginning of memory region specified in the **Location** parameter.
Size	Any number from 0. The combination of **Offset** and **Size** parameters is limited by the maximum allowed address. (The maximum allowed address is determined by memory region specified in the **Location** parameter.)	Maximum allowed size for memory region specified in the **Location** parameter	Specifies number of bytes to read starting from the address specified in the **Offset** parameter.
SaveTo	Any file path		File path to store the memory read.

Example 1:

This command reads the whole Mem32A memory region and stores it in the **c:/mem.bin** file.
The offset is 0. The read size is 128MB.

```
AddressSpace = Read {
      Location = Mem32A
      SaveTo = "c:/mem.bin"
}
```

Example 2:

This command reads 16 bytes from address 0x1000 of **Mem64** memory region and stores it in the **c:/mem.bin** file.

```
AddressSpace = Read {
      Location = Mem64
      Offset = 0x1000
      Size = 0x10
      SaveTo = "c:/mem.bin"
}
```

2.12.2 AddressSpace = Write

This command writes specified memory region into Summit Exerciser from specified data source.

Parameter	Values	Default	Comment
Location	Cfg Mem64 Mem32A Mem32B IOA IOB CtrlrMemBuf		Specifies the memory region into which to write. The memory region is mapped to the address space according to the rules described above. See the note below about the CtrlrMemBuf Location usage.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies offset in bytes from the beginning of the memory region specified in the **Location** parameter.
Size	Any number from 0. The combination of **Offset** and **Size** parameters is limited by the maximum allowed address. (The maximum allowed address is determined by the memory region specified in the **Location** parameter.)	If **Zeros**, **Ones**, **Random**, or **Incr** is specified for the **LoadFrom** parameter, then the default value is the maximum allowed size for the memory region specified in the **Location** parameter. Otherwise, the default size is the size of data specified in the **LoadFrom** parameter.	Specifies number of bytes to write starting from the address specified in the **Offset** parameter.
LoadFrom	Any file path Any array of bytes Zeros Ones Random Incr	Zeros	

Note: CtrlrMemBuf Location can be used within AddressSpace=Write command to write data from a file to NVMe optional Controller Memory Buffer. The binary file can be prepared to allow writing of series of NVMe commands (up to 64 at once, or up to 4096 bytes size).

The command would be converted to a series of Memory Write packets, sequentially writing the file data into locations in the Controller Memory Buffer, given the starting Offset and the base address specified by the CMB_BaseAddrHi and CMB_BaseAddrLo additional parameters.

Please see 2.13.1 in the Structure Command section for additional description of Controller Memory Buffer and the CMB_BaseAddrHi, CMB_BaseAddrLo parameters.

Example 1:

This command clears the whole **Mem32A** memory region.

```
AddressSpace = Write {
      Location = Mem32B
      LoadFrom = Zeros
}
```

Example 2:

This command writes 16 bytes, starting from address **0x1000**, into the **Mem64** memory region from file **c:/mem.bin**.

```
AddressSpace = Write {
      Location = Mem64
      Offset = 0x1000
      Size = 0x10
      LoadFrom = "c:/mem.bin"
}
```

Example 3:

This command writes 7 bytes, starting from address **0x1000**, into the **Mem64** memory region from data specified.

```
AddressSpace = Write {
      Location = Mem64
      Offset = 0x1000
      LoadFrom = ( 0x02, 0x08, 0x01, 0x03, 0x06, 0x07, 0x07 )
}
```

Example 4:

This command writes 48 bytes of random data, starting from address **0x10**, into the **IOA** memory region.

```
AddressSpace = Write {
      Location = IOA
      Offset = 0x10
      Size = 0x30
      LoadFrom = Random
}
```

Example 5:

This command writes 64 NVMe commands to NVMe Controller Memory Buffer (will be converted to series of Memory Write TLPs).

```
AddressSpace = Write {
      Location=CtrlrMemBuf
      Offset = 0x100
      Size = 4096
      CMB_BaseAddrHi = 2
      CMB_BaseAddrLo = 0xB0000000
      LoadFrom = "C:\NVMe\cmb_commands.bin"
}
```

2.13 Structure Command

This command writes the Summit Exerciser™ memory region, allowing you to specify various structures to be placed in the previously defined Host Memory Region. It is used for Host Emulation of a storage protocol using the Z3/Z4/Z5 Exercisers. The structure command is equivalent to the AddressSpace=Write command, but it allows specifying the data to be written field by field and conforms to a specific PCIe-based storage specification.

Currently, three types of structures are supported: AHCI, NVMe and PQI/SOP.

2.13.1 Command Parameters

The Location and Offset parameters are the same as for AddressSpace = Write command, with one exception. For the Location parameter, in addition to the defined applicable locations (Mem64, Mem32A, Mem32B) there is one location added - CtrlrMemBuf.

This location parameter can be used to simulate writing to the optional NVMe Controller Memory Buffer, and is only applicable to NVMe structures. When this value is used for the Location parameter, two more parameters will appear to be filled. These parameters are used to define the base memory address for the Controller Memory Buffer to be used in conjunction with the Offset parameter.

Parameter	Values	Default	Comment
CMB_BaseAddrHi	0x00000000 – 0xFFFFFFFF	0	High 32 bits of the Controller Memory Buffer base address.
CMB_BaseAddrLo	0x00000000 – 0xFFFFFFFF	0	Low 32 bits of the Controller Memory Buffer base address

When CtrlrMemBuf value is used for the Location parameter, all the NVMe structure fields are applicable and define the binary representation of the structure. But instead of being placed at the specified Offset in the specified PEExerciser Z3/Z4 Host Memory Region, ready for the NVMe Controller to read (as it is with all the "Mem" locations), this type of structure will be converted to a MemWr32 (if CMB_BaseAddrHi iz zero) or MemWr64 TLP to the memory address that is equal to (CMB_BaseAddr + Offset), and carrying the payload which is the binary representation of the structure.

For the sample script using Controller Memory Buffer location see nvme_cmd_controller_memory_buffer.peg in the NVMe Host Emulation samples.

Only the structure type parameters are discussed here. There are many different fields that will appear in the list on the right as the structure type and other parameters are specified.

Additional parameters for NVMe structures:

Parameter	Values	Default	Comment
NVMeStructType	AdminCommand NVMCommand PRP SGL_Descriptor		Choosing one of the structure types determines the total size of the structure. Additional parameters will display in the list on the right allowing you to further specify the opcodes and fields of the corresponding structure.

Additional parameters for AHCI structures:

Parameter	Values	Default	Comment
AhciStructType	Command FIS PRDT		Choosing one of the structure types determines the total size of the structure. Additional parameters will display in the list on the right allowing you to further specify the opcodes and fields of the corresponding structure.

Additional parameters for PQI/SOP structures:

Parameter	Values	Default	Comment
PqiSopStructType	AdminReqIU ReportGenIU ReportEvtCfgIU SetEvtCfgIU LimitedCmdIU CommandIU ExtendedCmdIU TaskMgmtIU		Choosing one of the structure types determines the total size of the structure. Additional parameters will display in the list on the right allowing you to further specify the opcodes and fields of the corresponding structure.

In case of SOP Command IUs additional parameters will appear to specify SCSI command set and the fields in SCSI CDB for selected command. Currently SPC-4 and SBC-3 command sets are supported.

Examples of NVMe structures:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 0

    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_IDENTIFY
    PRP1_Low = 0x3F25B190
    PRP1_High = 0x8
    PRP2_Low = 0x3F25C000
    PRP2_High = 0x8
    CNS = Controller
}

Structure=NVMe
{
    Location=Mem64
    Offset = 0x1000

    NVMeStructType = NVMCommand
    OpcodeNvm = Write
    CID = 5
    NamespaceId = 1
    PRP1_Low = 0xA0000000
```

```
    PRP1_High = 0x4
    StartLBALow = 12
    NumLBlocks = 0
}

Structure=NVMe
{
    Location=Mem64
    Offset = 0x1040

    NVMeStructType = PRP
    PageBaseAddrLow = 0xCD000000
    PageBaseAddrHigh = 0xA0B
}
```

Note: the PRP and SGL structures are only related to PRPs, PRP Lists and SGLs as they are used as part of the NVMe commands. The data that is being transferred during the data phase of the Write command (and any other command that inwolves data transfer from the Host memory to the controller can be loaded using the AddressSpace=Write instruction (see 2.12.2). Please refer to readme.txt mentioned below for suggested location and offset for this data.

//Inserting Number of Queues

```
Structure=NVMe
{
    Location = Mem64
    Offset = 22E0620C0
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_SET_FEATURES // or ADMIN_GET_FEATURES
    FUSE = Normal
    CID = 3
    NamespaceId = 0
    MetadataPtrLow = 0
    MetadataPtrHigh = 0
    PRP1_Low = 0
    PRP1_High = 0
    PRP2_Low = 0
    PRP2_High = 0
    FeatureID = NumQueues // One of many options for Feature ID & subsequent
values
    Save = No
    NCplQRqstd = 5
    NSubmQRqstd = 5
    HighPriWeight = 1
    MedPriWeight = 2
    LowPriWeight = 3
    ArbBurst = 4
}
```

You can find sample scripts utilizing NVMe structures in ...\Users\Public\Documents\LeCroy\PCIe Protocol Suite\Sample Files\Z3-16TrainerScripts\NVMe_HostEmulation

In this folder you can also find readme.txt file, describing the sample files and theory of operation.

Examples of AHCI structures:

```
Structure=AHCI
{
    Location=Mem64
    Offset = 0

    AhciStructType=Command
    PRDTL = 1
    FISLength = 5
    CTBA = 0xA0001000
    CTBAU = 0x4
}

Structure=AHCI
{
    Location=Mem64
    Offset = 0x1000

    AhciStructType=FIS
    FISType = RegisterH2D
    ATACommand = IDENTIFY_DEVICE
    C = Yes
    Device = 0xA0
}

Structure=AHCI
{
    Location=Mem64
    Offset = 0x1080

    AhciStructType=PRDT
    DBA = 0x3F3B2048
    DBAU = 0x2
    DBC = 511
}
```

You can find sample scripts utilizing AHCI structures in ...\Users\Public\Documents\LeCroy\PCIe Protocol Suite\Sample Files\Z3-16TrainerScripts\SATA_AHCI_HostEmulation

Examples of PQI/SOP structures:

```
; REPORT GENERAL IU
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x10000

    PqiSopStructType=ReportGenIU
    RequestID = 1
    ResponseQ_ID = 1
    AllocLength = 76
    SGLDescType = DataBlock
    SGLAddressHi = 0x4
```

```
    SGLAddressLo = 0xA0100000
    SGLLength = 76
}

; REPORT EVENT CONFIGURATION IU
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x10040

    PqiSopStructType = ReportEvtCfgIU
    RequestID = 2
    ResponseQ_ID = 1
    AllocLength = 12
    SGLDescType = DataBlock
    SGLAddressHi = 0x4
    SGLAddressLo = 0xA0100000
    SGLLength = 12
}
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x10080 ; Third element of the Operational IQ1

    PqiSopStructType = SetEvtCfgIU
    RequestID = 3
    ResponseQ_ID = 1
    DataBufSize = 8
    GQISEOQ = 1
    SGLDescType = DataBlock
    SGLAddressHi = 0x4
    SGLAddressLo = 0xA0100000
    SGLLength = 8
}

; TASK MANAGEMENT IU
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x100C0 ; Fourth element of the Operational IQ1

    PqiSopStructType=TaskMgmtIU
    RequestID = 4
    ResponseQ_ID = 1
    TaskMgmtFn = ABORT_TASK
    OQ_ToManage = 1
    RID_ToManage = 10
}

; READ10 SCSI command in Limited Command SOP IU
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x10040

    PqiSopStructType = LimitedCmdIU
```

```
    RequestID = 4
    ResponseQ_ID = 1
    DataBufSize = 512
    DataDirection = ToDataIn

    CommandSet = SBC_3
    SBC_Command = READ_10
    TRANSFER_LENGTH = 1

    SGLDescType = DataBlock
    SGLAddressHi = 0x4
    SGLAddressLo = 0xA0101000
    SGLLength = 512
}
```

You can find sample scripts utilizing AHCI structures in ...\Users\Public\Documents\LeCroy\PCIe Protocol Suite\Sample Files\Z3-16TrainerScripts\PQI_SOP_HostEmulation

2.13.2 Creating Errors in Structures Section

2.13.2.1 General

There are several ways of inserting various errors in the NVMe, AHCI/ATA and PQI/SOP structures.

- Using “Field” construct to set non-zero values to reserved fields.

The “Field” directive (described in 3.1) can be used to set non zero values in reserved fields within first 8 DWORDs of any structure:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 0

    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_IDENTIFY
    PRP1_Low = 0x3F25B190
    PRP1_High = 0x8
    PRP2_Low = 0x3F25C000
    PRP2_High = 0x8
    CNS = Controller

    Field[63:71] = 0xA5 ; setting non-zero value of a reserved byte in Command
DW 2
}
```

- Assigning undefined values to structure fields.

A numeric value can be assigned to any of the defined structure fields that is undefined or invalid:

```
Structure=AHCI
{
    Location=Mem64
    Offset = 0x1000
    AhciStructType=FIS
    FISType = 0x15   ;  undefined FIS type instead of RegisterH2D
    Features = 0x02
    ATACommand = SET_FEATURES
    C = Yes
    Device = 0xA0
}
```

- Using AddressSpace=Write to supply raw byte image of the structure with any contents desired.

As before Structure command was implemented, AddressSpace=Write can be used to place the byte stream of the desired structure in the Host Memory Region. This way any byte image of a structure can be programmed, creating any type of error desired.

- Specific for SOP: using CDB = (…) to create errors in SCSI CDBs.

For the case of SOP Command IUs a way to specify the raw byte stream for SCSI CDB is added. It can coexist with setting of the specific fields in that CDB and be used to corrupt other fields:

```
; MODE_SENSE_10 command
Structure=PQI_SOP
{
    Location=Mem64
    Offset = 0x10040 ; Offset of the Operational IQ1, second element, element
size 64 bytes

    PqiSopStructType = LimitedCmdIU
    RequestID = 4
    ResponseQ_ID = 1
    DataBufSize = 512
    DataDirection = ToDataIn

    CommandSet = SPC_4
    CDB = ( 0x0 0x0 0x0 0x0 0xAA 0xBB 0xCC ) ; Puts non-zero values 0xAA 0xBB
0xCC into reserved bytes 4-6
    SPC_Command = MODE_SENSE_10
    MODE_PAGE_CODE = 1
    MODE_SUBPAGE_CODE = 1
    SCSI_AllocLength = 64
    SGLDescType = DataBlock
    SGLAddressHi = 0x1
    SGLAddressLo = 0xB0000000
    SGLLength = 64
}
```

2.13.2.2 NVME command errors

The following are examples of inserting errors in NVMe commands that result in command specific error defined by the NVMe specification. Usually, the value of a certain field in the command that makes it invalid depends on previous setup and/or DUT capabilities.

- Inserting Completion Queue Invalid error:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 64
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_CREATE_SQ
    CID = 1
    Queue_ID = 1
    QueueSize = 0x3FF
    PhysContig = Yes
    CQID = 44          ; Assuming a completion queue with this ID hasn't been
created yet or would never be
    QPriority = Urgent
    PRP1_Low = 0x2FAB8000
    PRP1_High = 0x4
}
```

- Inserting Invalid Queue Identifier error:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 0
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_CREATE_CQ
    CID = 0
    PRP1_Low = 0x2FAC8000
    PRP1_High = 0x4
    Queue_ID = 115        ; Any value over the supported number of completion
queues would work
    QueueSize = 0x3FF
    PhysContig = Yes
    IntEnable = Yes
    IntVector = 1
}
```

- Inserting Invalid Queue Size error:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 0
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_CREATE_CQ
    CID = 0
    PRP1_Low = 0x2FAC8000
    PRP1_High = 0x4
    Queue_ID = 1
    QueueSize = 0x7FF ; will be invalid if controller can only support up to
0x400 entries
    PhysContig = Yes
    IntEnable = Yes
    IntVector = 1
}
```

- Inserting Invalid Interrupt Vector error:

```
Structure=NVMe
{
    Location=Mem64
    Offset = 0
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_CREATE_CQ
    CID = 0
    PRP1_Low = 0x2FAC8000
    PRP1_High = 0x4
    Queue_ID = 1
    QueueSize = 0x3FF
    PhysContig = Yes
    IntEnable = Yes
    IntVector = 0x801 ; 2049 - guaranteed invalid vector number for MSI-X or
MSI
}
```

- Inserting Invalid Log Page error:

```
Structure=NVMe
```

```
{
    Location=Mem64
    Offset = 0
    NVMeStructType=AdminCommand
    OpcodeAdmin = ADMIN_GET_LOG_PAGE
    CID = 0
    PRP1_Low = 0xD0000000
    PRP1_High = 0x1
    LogPageID = 7    ;  Any value 06h – 7Fh is reserved per NVMe spec
    NumLogDWs = 36
}
```

- Inserting LBA Out of Range error:

```
Structure=NVMe
{
    Location = Mem64
    Offset = 0x10000
    NVMeStructType=NVMCommand
    OpcodeNvm = Write
    CID = 0
    NamespaceId = 1
    PRP1_Low = 0x2FB08000
    PRP1_High = 0x4
    NumLBlocks = 1

    StartLBALow  = 0xA0000000
    StartLBAHigh = 0xFABCDE00 ; - in most cases this will exceed drive's
capacity
}
```

- Inserting Invalid Namespace ID error:

```
Structure=NVMe
{
    Location = Mem64
    Offset = 0x10000
    NVMeStructType=NVMCommand
    OpcodeNvm = Write
    CID = 0
    NamespaceId = 3 ; assuming the controller has 2 namespaces, any higher
value works too
    PRP1_Low = 0x2FB08000
    PRP1_High = 0x4
    NumLBlocks = 1
}
```

2.13.3 Structure=MCTP

This structure defines a MCTP message via SMBus or VDM.

Parameter	Values	Default	Comment
MCTPStructType	SMBus or VDM	None	

MCTP Structure depends on a selected struct type. It reuses most of the available keys of the base level(see Packet=SMBus or Packet=TLP).

Structure doesn't have the next SMBus specific fields:

Parameter	Default	Comment
CommandCode	MCTP	
ByteCntSMBus	Unknown	It depends on the MTU Size
Payload	None	Shouldn't be used

Structure doesn't have the next VDM specific fields:

Parameter	Default	Comment
TLPType	MsgD	
MessageCode	Vendor_Defined_Type1	
VendorId	0x1AB4	
MessageRoute	ByID	
Length	Unknown	It depends on the MTU Size
Payload	None	Shouldn't be used
PSN	None	

Structure doesn't have the next MCTP specific fields:

Parameter	Default	Comment
MCTP_HDR_SOM	Unknown	It depends on the MTU Size
MCTP_HDR_EOM	Unknown	It depends on the MTU Size
MCTP_HDR_Psn	Unknown	It depends on the MTU Size

MCTP Structure splits a mctp message in the needed number of packets according to MTU_Size which is defined in MCTP Config.

If MTU_Size is not defined, we use the default MTU size value which is 64 bytes.

Example:

The next code defines Set Features(NVMe Admin Command) with FeatureID = 0x7E and Data_Length = 128 bytes. It uses Request_Data field to set first 16 dword(64 bytes) of the requested data.

```
Structure=MCTP

{

    MCTPStructType = VDM

    MCTP_HDR_Version = 1

    MCTP_HDR_DestEPID = 0x10

    MCTP_HDR_SrcEPID = 0xAB

    MCTP_HDR_TO = 1

    MCTP_MSG_RqBit = Yes

    MCTP_MSG_IC = 1

    MCTP_MSG_Type = NVMe_MI

    NVMeMI_MsgType = NVMe_Admin_Cmd

    OpcodeAdmin = ADMIN_SET_FEATURES

    Controller_ID = FIRST_CTRL_ID

    NamespaceId = DUT_ACTIVE_NAMESPACE

    FeatureID = 0x7E

    Element_Action = AddEntryMult

    Command_Flags = 0x1 ; Enable DLEN

    Data_Length = 128

    Request_Data = ( 0x1 0x02003412 0x0 0x0

                     0x0 0x0 0x0 0x0

                     0x0 0x0 0x0 0x0

                     0x0 0x0 0x0 0x0 )}
```

The next code defines VPD Read(NVMe-MI command) with DOFST = 5 and DLEN = 300.

```
Structure=MCTP

{

    MCTPStructType = SMBus

    DestSlaveAddr = 0x10

    SrcSlaveAddr = 0xAB

    MCTP_HDR_Version = 1

    MCTP_HDR_DestEPID = 5

    MCTP_HDR_SrcEPID = 10

    MCTP_HDR_TO = 1

    MCTP_MSG_Type = MCTP_Control

    MCTP_MSG_RqBit = Yes

    MCTP_MSG_IC = 1

    MCTP_MSG_Type = NVMe_MI

    NVMeMI_MSG_ReqOResp = Request

    NVMeMI_MsgType = NVMe_MI_Cmd

    NVMeMI_CmdSlotId = CmdSlot0

    NVMeMI_Cmd_Opcode = VPD_Write

    NVMeMI_Cmd_VpdRdWr_DOFST = 5

    NVMeMI_Cmd_VpdRdWr_DLEN = 300

}
```

The possible results of this script.

MTU_Size	Result
64	5 packets with 64 bytes of mctp payload
128	2 packets with 128 bytes of mctp payload and 1 packet with 64 bytes of mctp payload
200	1 packet with 200 bytes of mctp payload and 1 packet with 120 bytes of mctp payload

2.14 FastTransmit Engine Commands

2.14.1 FastTransmit Command

This command defines and controls high-performance generation, which executes using **FastTransmit** blocks. A **FastTransmit** block has **Setup** commands followed by the **Start** command:

```
FastTransmit = Setup
     <Send commands>
FastTransmit = Start
```

FastTransmit block script commands can also **Pause**, **Continue**, and/or **Stop** execution of high-performance generation.

FastTransmitIdle between FastTransmit Blocks

Defining a **FastTransmit** block initializes the High-Performance Transmitter. Therefore, high-performance generation requires that each **FastTransmit** block finish before executing the next **FastTransmit** block. To ensure that the previously started high-performance generation **FastTransmit** block has finished execution, use the **Wait=FastTransmitIdle** command before starting a new **FastTransmit** block. Inserting this **Wait** command between **FastTransmit** blocks guarantees that the previous block completes and the system goes to the **FastTransmitIdle** state before starting the next block. There is no other way to guarantee previous-block execution completion.

Fast Transmit and regular TLP Generation Operate in Parallel

Fast Transmit and regular TLP transmit generation occur in parallel. Regular-performance commands have priority. Even if a **FastTransmit** block contains infinite loops or commands, regular TLP transmit code still runs in parallel.

Fast Transmit Mode and High Performance Completer Are Independent

The High-Performance Completer is independent of the Fast Transmit Mode. **FastTransmit** blocks initialize the Fast Transmit Mode but do not control the High-Performance Completer. Scripts can use the **Config** command to enable or disable **Fast Completion** anywhere in the script, except in a **FastTransmit** block. (See Section 4.3.1 High Performance Mode of *the Summit Z416 PCI Express 4.0 Protocol Exerciser User Manual* to enable high performance modes*).*

Send Command

A **Send** command defines a sequence of high-performance packets to send once, a number of times, or repeatedly. **Send** commands are in the **Setup** command section of a **FastTransmit** block. A **FastTransmit** block can have up to 32 **Send** commands. (For more information about the **Send** command, see the next section of this manual.)

Note: Currently, **Send** commands are the only commands allowed in the **Setup** section of **FastTransmit** blocks.

2.14.2 FastTransmit = Setup

Begins the definition section of a **FastTransmit** block.

2.14.3 FastTransmit = Start

Ends the definition section of a **FastTransmit** block and starts execution of the high-performance generation defined by the block.

2.14.4 FastTransmit = Pause

Pauses high-performance generation of the currently running **FastTransmit** block.

If a **FastTransmit** block is not currently defined or running, this command does nothing.

2.14.5 FastTransmit = Continue

Continues high-performance generation of the paused **FastTransmit** block.

If a **FastTransmit** block is not currently defined or paused, this command does nothing.

2.14.6 FastTransmit = Stop

Stops high-performance generation of the currently running or paused **FastTransmit** block and goes to the **FastTransmitIdle** state, which has no defined or running **FastTransmit** block.

If a **FastTransmit** block is not currently running or paused, this command does nothing.

2.15 Send Command

A **Send** command defines a sequence of high-performance packets to send once, a number of times, or repeatedly. **Send** commands are allowed only in the **Setup** command section of a **FastTransmit** block. A **FastTransmit** block can have up to 32 **Send** commands.

Note: Currently, **Send** commands are the only commands allowed in the **Setup** section of **FastTransmit** blocks.

The **Send** command is similar to the **Packet=TLP** command. The difference is that the **Send** command specifically sends memory transactions, which send MRd32/64 or MWr32/64 TLPs at the highest possible rate. Also, the **Send** command omits definition of some low-level TLP fields, because they are set automatically during transmission.

The **Send** command contains parameters to specify creation of a TLP set or sets:

Parameter	Values	Default	Comment
Snoop	0 – 1 (Default: NoSnoop)	0	Bit 4 of byte 2 in the TLP header: 0 = hardware enforced cache coherency is expected. 1 = hardware enforced cache coherency is not expected.
Ordering	0 – 1 (Default: Relaxed)	0	Bit 5 of byte 2 of TLP header: 0 = PCI Strongly Ordered Model. 1 = PCI-X Relaxed Ordering Model.
Length	0 – 1023	1	Length of data payload in DWORDs. If not specified, this field is 1 for all read/write requests. For a length of 1024, set Length to 0 (because 0 means 1024).
RequesterID	(XX:XX:X) or direct value	0	Bytes 4-5 of the TLP Header for Memory TLP packets. This parameter can be set in the following format: (**BusNumber:DeviceNumber:Function Number**)
Count	0 – 255	1	Repeats the memory transaction the number of times specified in a row within the currently executing loop of the **FastTransmit** block.

LoopCount[1]	0 – 255	1	Repeats this command the number of times specified, within the entire **FastTransmit** block, which loops through commands.
			LoopCount indicates up to which loop increment this command repeats. For example:
			`for(i = 0; i < 255; i++)`
			`{`
			`  if(i < this LoopCount)`
			`  repeat this Send command`

Parameter	Values	Default	Comment

			`if(i < another LoopCount)`
			`repeat another Send command`
			`etc.`
			`}`

PayloadGrowth	FixedByte FixedDWord IncrementByte IncrementDWord	FixedByte	Specifies how to generate the payload of size **Length**: **FixedByte**: Specifies the payload as a repeated pattern of byte **PayloadSeed**. **FixedDWord**: Specifies the payload as a repeated pattern of dword **PayloadSeed.** **IncrementByte**: Specifies the payload as an incrementing pattern of bytes starting from Payload Seed Value. **Note**: If Payload Seed Value is not specified, the pattern will start from value 0. **IncrementDWord**: Specifies the payload as an incrementing pattern of dwords starting from Payload Seed Value. **Note**: If Payload Seed Value is not specified, the pattern will start from value 0. **Note**: The **PayloadGrowth** parameter applies to all **Send** commands.
PayloadSeed	**PayloadGrowth =** FixedByte 0x00 – 0xFF FixedDWord 0x000 – 0x3FF	0	Specifies the fixed value from which to generate the payload: **PayloadGrowth = FixedByte**: This parameter is an 8-bit field. **PayloadGrowth = FixedDWord:** This parameter is a 10-bit field. **Note**: The **PayloadSeed** parameter applies only when **PayloadGrowth** applies and is set to a growth type of FixedByte or FixedDWord.
IncrementAddress	No OnCount	No	Specifies if the **Address** value for each subsequent address of a repeated read or write TLP packet (**Count** > 1) is incremented. The Address value is incremented to point to the next memory location, according to the Length set for the memory transaction. **Note**: This parameter only increments the original **Address** value for repeating (**Count** > 1), not for looping (**LoopCount** > 1).

Note 1: When LoopCount is set to “0” the command is repeated indefinitely.

2.15.1 Send = MRd32/MWr32

Parameter	Values	Default	Comment
Address	0x00000000 – 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.

2.15.2 Send = MRd64/MWr64

Parameter	Values	Default	Comment
AddressLo	0x00000000 – 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.
AddressHi	0x00000000 – 0xFFFFFFFF	0	Bytes 12-15 in the TLP header.

Note: The **Send** command supports field substitution (see section 2.1.1.7). The **Send** command supports substitution for the following fields:

- RequesterId
- Address
- AddressLo
- AddressHi
- PayloadSeed

2.16 Raw Ltssm Commands

Raw Ltssm is a feature available in Z416, Z58 Z516 and M616 Exercisers. It allows to set up transmission of sequences of Ordered Sets and other patterns allowing for low-level emulation of LTSSM behavior.

This command defines and controls Raw LTSSM generation, which executes using RawLtssm blocks. A RawLtssm block consists of the Setup command followed by the Start command:
RawLtssm = Setup
<OrderedSet / Raw /RawLtssm Config commands>
RawLtssm = Start

Defining a RawLtssm block initializes the Raw LTSSM. Therefore, Raw LTSSM generation requires that each RawLtssm block finish before executing the next RawLtssm block. To ensure that the previously started RawLtssm block has finished execution, use the Wait=RawLtssmDone command before starting a new RawLtssm block.

2.16.1 RawLtssm = Setup

Begins the definition section of a RawLtssm block and sets up initial link parameters.

Parameter	Values	Default	Comment
LtssmSpeed	2_5, 5_0, 8_0, 16_0 or 32_0	2_5	Specifies one of five PCIe speeds of operation (32 GT/s only available for Z58, Z516 and M616)
LtssmLinkWidth	x1, x2, x4, x8 or x16	x1	Specifies starting link width
LtssmSkipType	None, Gen12, Gen34, Gen34wEDS, Gen4wCtrl, Gen4wCtrlEDS, Gen5wCtrl, Gen5wCtrlEDS	None	Specifies automatic Skip insertion policy. Can be one of: No Skips inserted, Gen1/2 style Skips inserted, Gen3/4 Skips or Gen3/4 Skips followed by EDS, Gen4 and Gen5 alternating Standard and Control Skips, without or with EDS are inserted. Skips are inserted where appropriate in between the programmed Ordered Sets / patterns based on the Skip Frequency.
LtssmSkipFreq	Number. Frequency expressed in DWORDs	0	Specifies the base frequency of the automatic Skip generation expressed in number of DWORDs of traffic in between.

2.16.2 RawLtssm = Start

Ends the definition section of a RawLtssm block and starts execution of the Raw LTSSM generation defined by the block.

2.16.3 RawLtssm Examples

```
; Setup initial Raw LTSSM link properties
RawLTSSM=Setup
{
    LtssmSpeed = 8_0
    LtssmLinkWidth = x4
    LtssmSkipType = None
}

; Add ordered sets to be sent

Packet=OrderedSet
{
    SetType = EIEOS
    LinkSpeed = 8_0
    Count = 2
}

; Send TS1 Ordered Set
packet = OrderedSet
{

    SetType = TS1
    LinkSpeed = 8_0
    LinkNumber = 0
    LaneNumber = 0
    N_FTS = 10
    TrainingControl = 0 ;x1
    EqControl = 0
    PreCursor = 50
    Cursor = 40
    PostCursor = 25
    Count = 70
    TrainingControl@2 = 0x5
    N_FTS@2 = 100             ; set number of FTS for lane 2, all other lanes
has the default value of FTS (10)
    LaneNumber@1 = PAD

    LinkNumber@3 = 3    ; set link number for lane 3, all other lane has the
default value for link number (0)
}
```

```
; Change configuration
Config=RawLtssm
{
    LtssmSpeed = 8_0
    LtssmLinkWidth = x4
    LtssmSkipType = Gen34 ; Change the automatic Skip settings and start
sending Gen3 Skips
    LtssmSkipFreq = 500
}



; Send more ordered sets
packet = OrderedSet
{
    SetType = TS2
    LinkSpeed = 8_0
    LinkNumber = 2
    LaneNumber = 3
    N_FTS = 30
    EqControl = 0

    Count = 70
}


; Start Raw LTSSM
RawLTSSM=Start

; Wait for Raw LTSSM to finish
Wait=RawLtssmDone
{
}
```

2.17 PCIeFlitMode Command

PCIeFlitMode is used to define a global setting that enables exerciser scripting for Flit mode of link operation introduced by Gen6 revision of the PCIe specification.

NOTE: PCIeFlitMode is available only for the Z6 FM (Flit Mode) Exerciser configuration available on the M616 platform.

Parameter	Values	Default	Comment
PCIeFlitMode	Yes, No	No	When set to No, then the exerciser scripting is functioning as described in other parts of this manual, relevant to Gen5 and Gen6 non-flit mode operation.

The **PCIeFlitMode** command should be the first line of the main script. If the script has included files, **PCIeFlitMode** must be the same in all files**.**

When **PCIeFlitMode=Yes**, this changes the behavior of the script making it suitable to be executed when Summit M616 platform is in the Flit Mode Exerciser configuration.

Without **PCIeFlitMode=Yes** a script would not be allowed to run on the Z6 Flit Mode Exerciser configuration, and with it will not be allowed to run on any other exerciser configuration.

PCIeFlitMode=Yes changes how the TLPs are built - now they conform to new Gen6 Flit mode format. Also, other settings and features become modified or not available (as they are not relevant to the Flit mode operation).

Some new script features specific to the Flit mode operation appear.

Most of the Link commands are applicable to the Flit mode and have the same meaning.

For **Packet=TLP**, many of the packet fields itself have the same name and meaning. Important header fields **OHC** (Orthogonal Header Content) and **TS** (Trailer Size) are added. Currently, they only accept numerical values.

Based on the **OHC** and **TS** values, defined other relevant **TLP** fields appear. Additional Flit mode specific TLP parameters are **PoisonedTLPMarke** (Yes/No) and **NullifiedTLPMarker** (Yes/No) to inject flit level errors in Payload flit carrying the TLP.

NOTE: More Flit mode information coming in future releases.

2.18 CXL256BFlitMode Command

CXL256BFlitMode is used to define a global setting that enables exerciser scripting for Flit mode of link operation first introduced in CXL 3.0 specification.

NOTE: CXL256BFlitMode is available only for the Z6 FM (Flit Mode) and CXL 3.0 Exerciser configurations available on the M616 platform.

Parameter	Values	Default	Comment
CXL256BFlitMode	None CXL_3_0 CXL_3_0_PBR CXL_3_0_L0pt CXL_3_1 CXL_3_1_PBR CXL_3_1_L0pt	None	Changes configuration to CXL 3.0/3.1.

The CXL256BFlitMode command should be the first line of the main script. If the script has included files, CXL256BFlitMode must be the same in all files.

When CXL256BFlitMode=CXL_3_0, this changes the behavior of the script making it suitable to be executed when Summit M616 platform is in the CXL 3.0 Mode Exerciser configuration.

When CXL256BFlitMode=None a script would not be allowed to run on the Z6 CXL 3.0 Exerciser configuration, and with it will not be allowed to run on any other exerciser configuration.
For CXL.io packets the script behavior is similar to PCIeFlitMode.

Some new script features specific to the PCIe Flit and CXL 256B Flit modes operation appear.

Example:

```
CXL256BFlitMode = CXL_3_0
```

3 Appendix A: How to Contact Teledyne LeCroy

Send e-mail…	psgsupport@teledynelecroy.com
Contact support…	teledynelecroy.com/support/contact
Visit Teledyne LeCroy's web site…	teledynelecroy.com
Tell Teledyne LeCroy…	Report a problem to Teledyne LeCroy Support via e-mail by selecting **Help > Tell Teledyne LeCroy** from the application toolbar. This requires that an e-mail client be installed and configured on the host machine.