White Paper

Triggering a PCIe Analyzer from your SBExpress System

SANBlaze Built-In Analyzer Trigger Mechanism
Makes Finding a Needle in a Haystack a Snap

Author:

Vince Asbridge
President & Director of Software Engineering, SANBlaze

Contents

Table of Figures

Version Information

V0.1 – Initial review copy
V0.2 – SE, SH, DM, comments
V0.3 – PB, formatting

Introduction

SANBlaze SBExpress and the Certified by SANBlaze test methodology have greatly simplified qualification of PCIe NVMe devices by providing a simple means of creating complex test suites, validating specification compliance, data integrity, power and reset testing and MI compliance.

With the SANBlaze built-in transactional tracing, low level error counters including Link Training and Status State Machine (LTSSM), the ability to subject devices to variations in voltage and complex reset, link training and data integrity testing and log their behavior, one may ask, "why do I need a PCIe analyzer at all?".

The bottom line is that when the unexpected occurs on the PCIe bus, such as a transaction to a device that never completes, there is no better tool than a PCIe analyzer between the CPU Root Complex and the end point device to determine root cause of the issue.

A Needle in a Haystack

With up to 8Gbytes per second[1] of data flying back and forth between the PCIe root complex and your Device Under Test, finding the source of your SBExpress error and coordinating it with the data in your analyzer becomes like finding a Transaction Layer Packet (TLP) the size of a needle in the proverbial haystack of data in your analyzer.

To further compound the difficulty of coordinating the error flagged by your SBExpress system with the data now in your analyzer, the data you're looking for may in fact no longer be in your analyzer. Take, for example an analyzer with a reasonably large buffer storage capacity of 64GBytes of data, and a typical PCIe sequential read rate of 4GB/Second, your analyzer can capture up to 16 seconds of data until the buffer begins to wrap and the haystack and needle have been loaded onto a flatbed and are half way to Paris by the time you realize they're missing.

You Need a Trigger

What you need is a trigger! A way to stop your analyzer around the time the error occurs so the error itself and some events leading up to the error including the cause of the error, and some events after the error which show how your device handled the error.

But What to Trigger On

You call the smartest engineer you can find from your firmware team and describe the problem to her. "An error occurred while my SANBlaze system was testing our latest firmware, and the SANBlaze error log shows an error. Can you please set up a trigger on my analyzer to find an event that occurs once in a 12 hour test? The trigger needs to occur coordinated with the unknown event, and needs to occur within 16 seconds of the event".

When she finishes laughing, your engineer tells you what you're asking is next to impossible, you're looking for a needle in a haystack.

You Need a Giant Magnet

To stay with the proverbial Needle in a Haystack analogy, what you need is a giant magnet, which will pull the tiny needle from the massive data haystack so you can have a closer look at it. You need the SANBlaze SBExpress Trigger.

Figure 1: SBExpress Trigger, Your Giant Magnet

Introducing the SANBlaze SBExpress Trigger

The SANBlaze team feels your pain, and in fact we have the same issue in our labs and at our customers. A problem is reported, the customer uploads the system log to the JIRA system. We look through the logs and conclude the device misbehaved on the PCIe bus, please send us a trace.

The question of what to trigger on so that the trace is coordinated with the SANBlaze system is a difficult one. Until you see a trace around the error, you won't know how to set up your analyzer to capture it, but you need to capture a trace to see what you need to trigger on. A classic Catch22 situation.

Enter the SANBlaze SBExpress Trigger. We have built a trigger mechanism into our V8.2 software so that we will trigger your analyzer at the exact point we determine the failure occurred, so now when we say "send us a trace", you can simply upload your analyzer trace around the SANBlaze Trigger.

The remainder of the paper will walk through how to set up the SANBlaze Trigger on your SBExpress system and your analyzer.

A Word about PCIe Gen4 Analyzers

There are a number of high quality, state of the art, tools for the capture and analysis of PCIe Gen4 (and now Gen5) traces. We have chosen to use the Kodiak Gen4 PCIe Analyzer from SerialTek to demonstrate the SBExpress trigger, but the configuration should be similar on any Gen4 PCIe analyzer.

The examples below will translate directly to your PCIe Gen4 or Gen5 analyzer and will behave in a similar manner. Contact your analyzer vendor or SANBlaze for help with configuring the SANBlaze Trigger.

When Bad Things Happen

When the SBExpress system determines something unexpected has happened while testing your NVMe Device, a special pattern will be generated and sent over PCIe. The correctly configured analyzer will detect the special SANBlaze Trigger and capture the trace for further analysis.

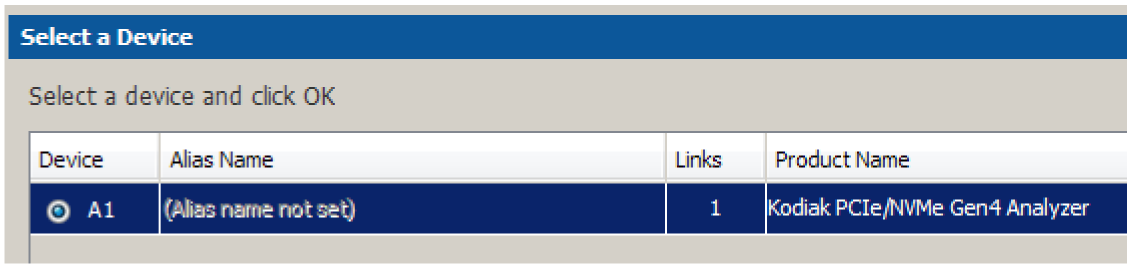
In addition to a fixed trigger pattern, the SANBlaze Trigger will include 16 bits of diagnostic information which the SANBlaze support team can use to hone directly into the place in our code which generated the trigger, greatly simplifying your efforts in capturing the trace and our efforts in determining why we detected an error.

Setting Up the Analyzer to Trigger

In order to use the SANBlaze Trigger, you need to teach your analyzer what to look for. This is done by configuring an analyzer trigger. When the analyzer sees the trigger, it will stop tracing, and the trace will be perfectly coordinated with the error detected by the SANBlaze SBExpress system.

Start by Connecting to your Analyzer

First connect to the analyzer. Based on your specific analyzer, this procedure will vary.


Select a Device

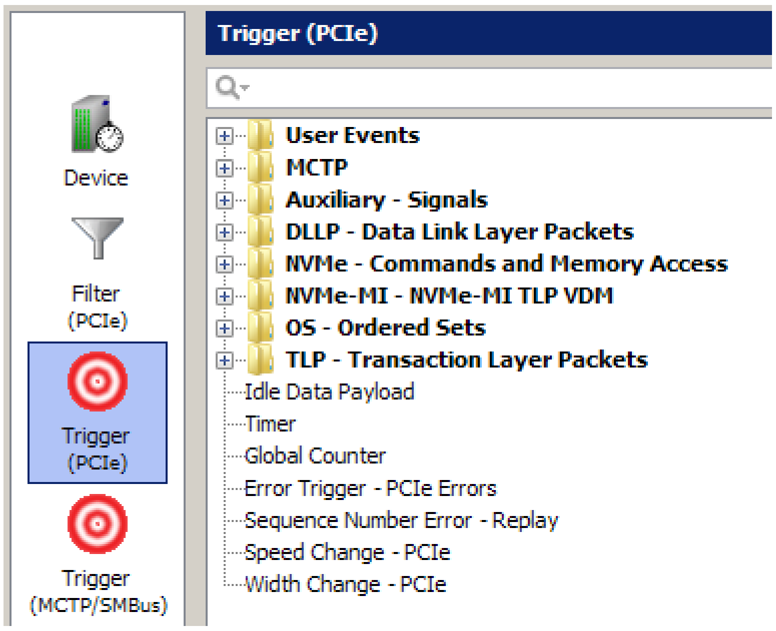
Select a device and click OK

Device	Alias Name	Links	Product Name
A1	(Alias name not set)	1	Kodiak PCIe/NVMe Gen4 Analyzer

Figure 2: Connect to your Analyzer

Select the Trigger Page

Find the page on your analyzer which controls triggers. If your analyzer supports multiple physical buses, you need to select the PCIe trigger page as shown in the figure below.



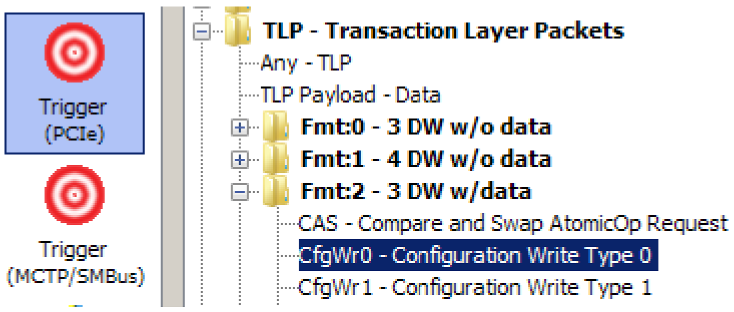
Trigger (PCIe)

- User Events
- MCTP
- Auxiliary - Signals
- DLLP - Data Link Layer Packets
- NVMe - Commands and Memory Access
- NVMe-MI - NVMe-MI TLP VDM
- OS - Ordered Sets
- TLP - Transaction Layer Packets
- Idle Data Payload
- Timer
- Global Counter
- Error Trigger - PCIe Errors
- Sequence Number Error - Replay
- Speed Change - PCIe
- Width Change - PCIe

Figure 3: Select the PCIe Trigger Page

Add a Trigger on a Configuration Write Type

You will add a trigger on a PCIe CfgWr0 - Configuration Write Type 0 (or Configuration Write Type 1 if Configuration Write Type 0 doesn't work). See below.

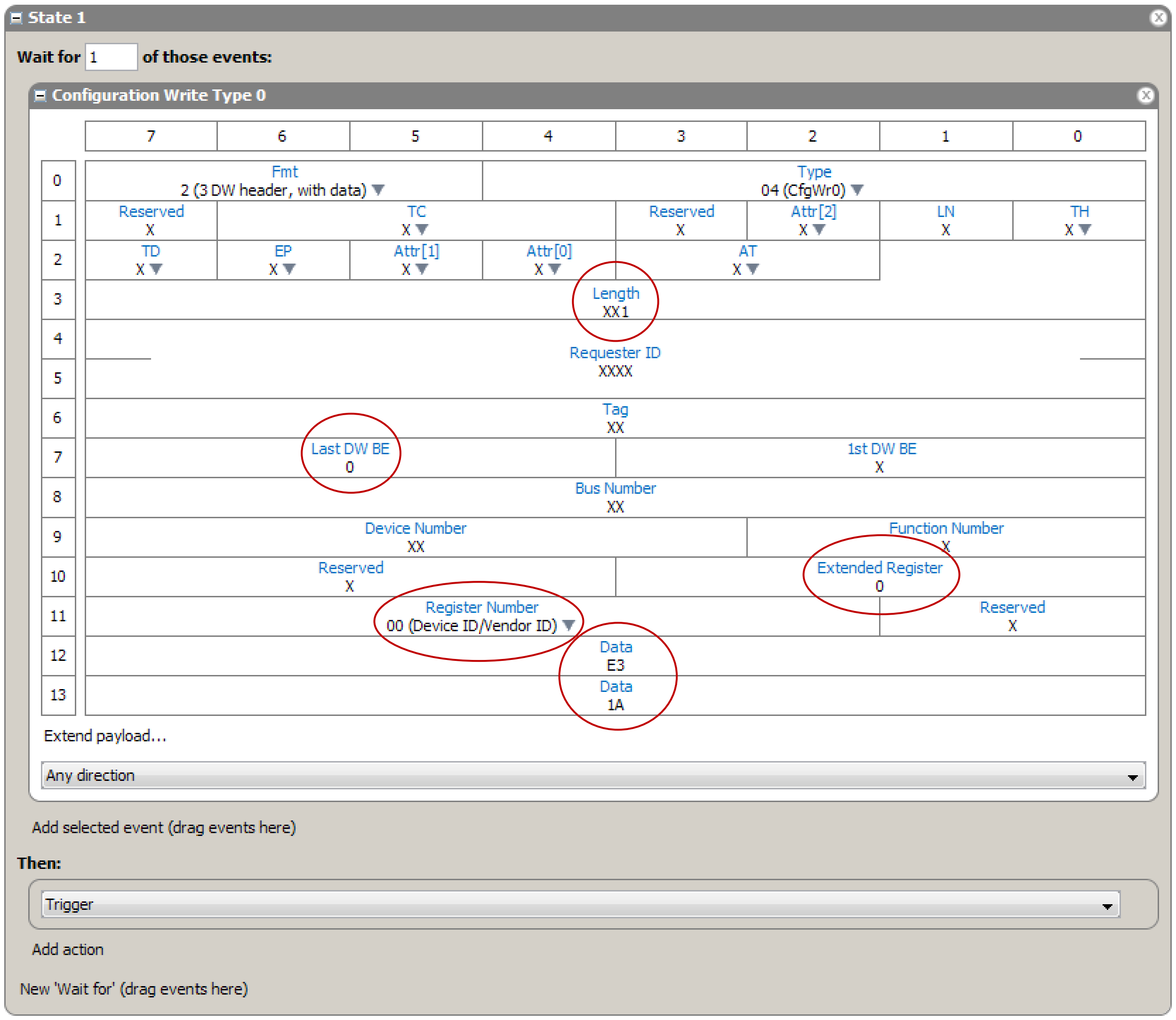


- TLP - Transaction Layer Packets
  - Any - TLP
  - TLP Payload - Data
  - Fmt:0 - 3 DW w/o data
  - Fmt:1 - 4 DW w/o data
  - Fmt:2 - 3 DW w/data
    - CAS - Compare and Swap AtomicOp Request
    - CfgWr0 - Configuration Write Type 0
    - CfgWr1 - Configuration Write Type 1

Figure 4: Add a Trigger on a Configuration Write Type

And Look for the SANBlaze Trigger

Set up your analyzer to look for a Configuration Write and make exactly these changes to the data.



State 1

Wait for 1 of those events:

Configuration Write Type 0

	7	6	5	4	3	2	1	0
0	Fmt 2 (3 DW header, with data)				Type 04 (CfgWr0)			
1	Reserved X	TC X			Reserved X	Attr[2] X	LN X	TH X
2	TD X	EP X	Attr[1] X	Attr[0] X	AT X			
3	Length XX1							
4	Requester ID XXXX							
5								
6	Tag XX							
7	Last DW BE 0				1st DW BE X			
8	Bus Number XX							
9	Device Number XX					Function Number X		
10	Reserved X				Extended Register 0			
11	Register Number 00 (Device ID/Vendor ID)						Reserved X	
12	Data E3							
13	Data 1A							

Extend payload...

Any direction

Add selected event (drag events here)

Then:

Trigger

Add action

New 'Wait for' (drag events here)

Figure 5: Modify Exactly These Fields

Modify the data in the following fields, leave everything else unchanged.

- Transfer Length = 1
- Last DW BE = 0
- Extended Register = 0
- Register Number = 00 (Device ID/Vendor ID)
- Extend Payload 2 bytes (4 if you intend to look for a specific Type/Subtype)
- Data Byte12 = E3
- Data Byte13 = 1A

And set the Action to Trigger

Save Your SANBlaze Trigger

After setting up the SANBlaze Trigger, access the "Save Trigger" function and save your trigger.

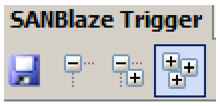


Figure 6: Save Your SANBlaze Trigger

Just so we remember what we did, rename your trigger to "SANBlaze Trigger" as shown above. This isn't strictly required, but it's good practice so that when you get a trigger, we'll all know why.

Testing your SANBlaze Trigger

At this point, we've set our analyzer to look for the specific trigger that SANBlaze will send on an error. This is our proverbial magnet. Now let's lower it over our haystack and see if we can find that needle.

From a Chrome browser, access the SANBlaze Web Based GUI and select the SBExpress Manager page from the left hand menu. From the GUI determine the Controller number of the device where your analyzer is connected.

You can pick up the IP address of the SANBlaze system by looking at the front panel.

From the browser interface the default login is:

```
Username: system
Password: SANBlaze
```

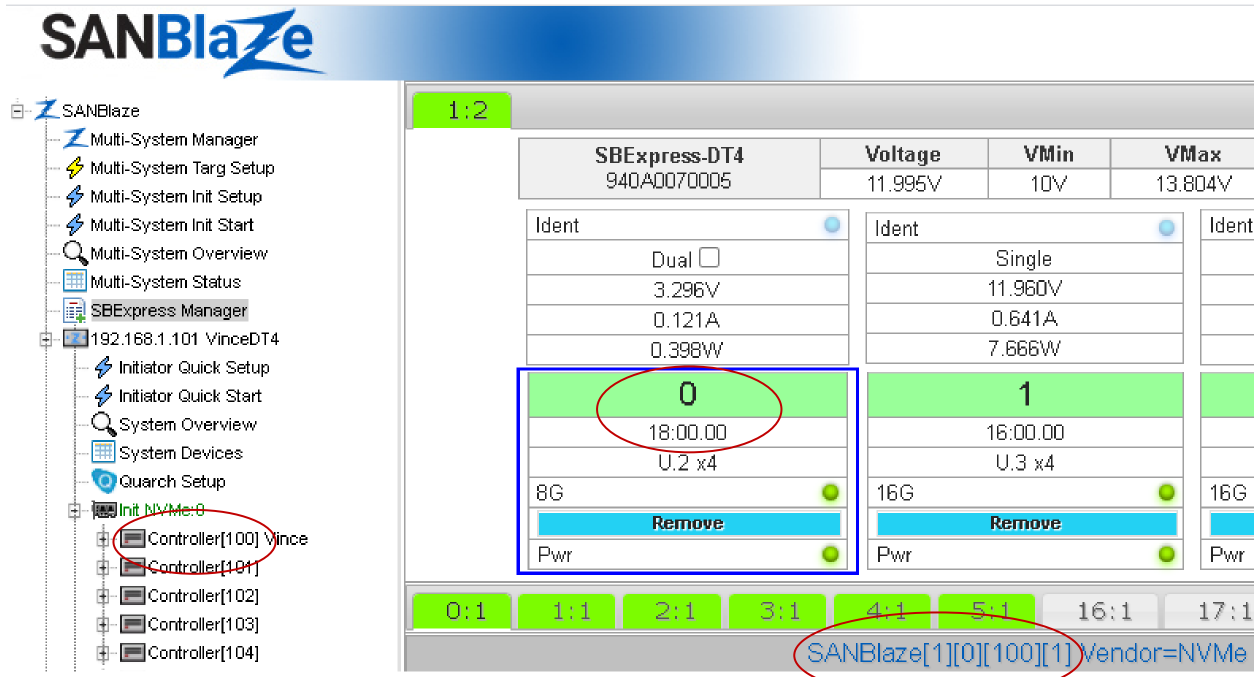


Figure 7: Use your SBExpress Page to Determine Controller Number

Note from the illustration above that slot zero is at PCIe ID 18:00.00 and at Controller number 100. We can use either of these numbers in our syntax to test our trigger.

To test the trigger, start an SSH session to your SANBlaze.

```
ssh 192.168.1.101 -l vlun
vlun@192.168.1.101's password: SANBlaze
```

Note, your system administrator may have changed the password. Once logged in, you need to grant yourself root privileges using the su command:

```
su
password: SANBlaze
```

Issue the command:

```
sbecho trigger=100 > /proc/vlun/nvme
```

Where 100 is the controller number from the figure above, or alternatively by the PCI ID of the device:

```
sbecho trigger=0000:18:00.0 > /proc/vlun/nvme
```

Your analyzer will trigger, verifying the correct configuration. If your analyzer does not trigger, check your settings, your controller number and try again.

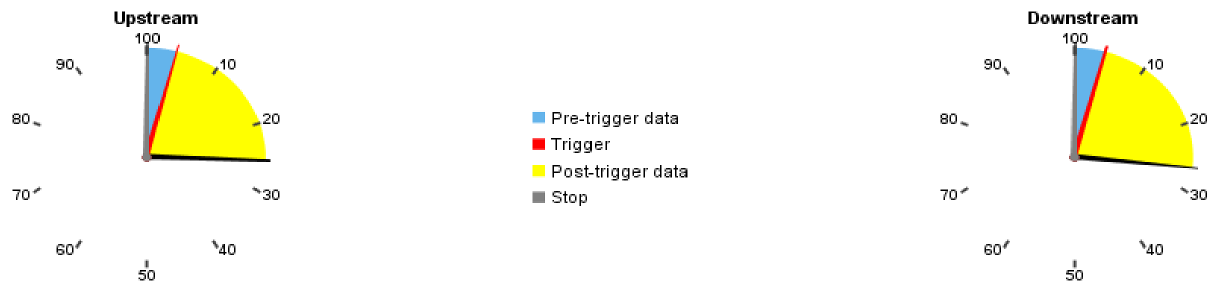


Figure 8: Analyzer Triggered Successfully

Your analyzer will continue to gather "Post-Trigger" data until the specified buffer is full.

Ready to Throw the Needle into the Haystack

At this point we're ready to do some real testing. You can clear the test trace and re-start tracing on your analyzer.

Build Your Data Haystack

We'll use the NameSpace Write test to generate our haystack data as follows:

```
sbecho Write_haystack,1,1,0,0,0,0,100,0,0,0,1,1,0,1:1,1:1,0,-0, >
/iport0/target100lun1
```

Note: I'm using a custom data pattern as found in "Appendix A: Data Patterns", but any traffic is fine. You can use SBExpress Manager to run your tests. The trigger will be generated automatically if a PCIe error occurs, or you can force a stop by echoing the trigger.

Once you issue the command above, or start IOs from the GUI, you will see the analyzer collecting data.

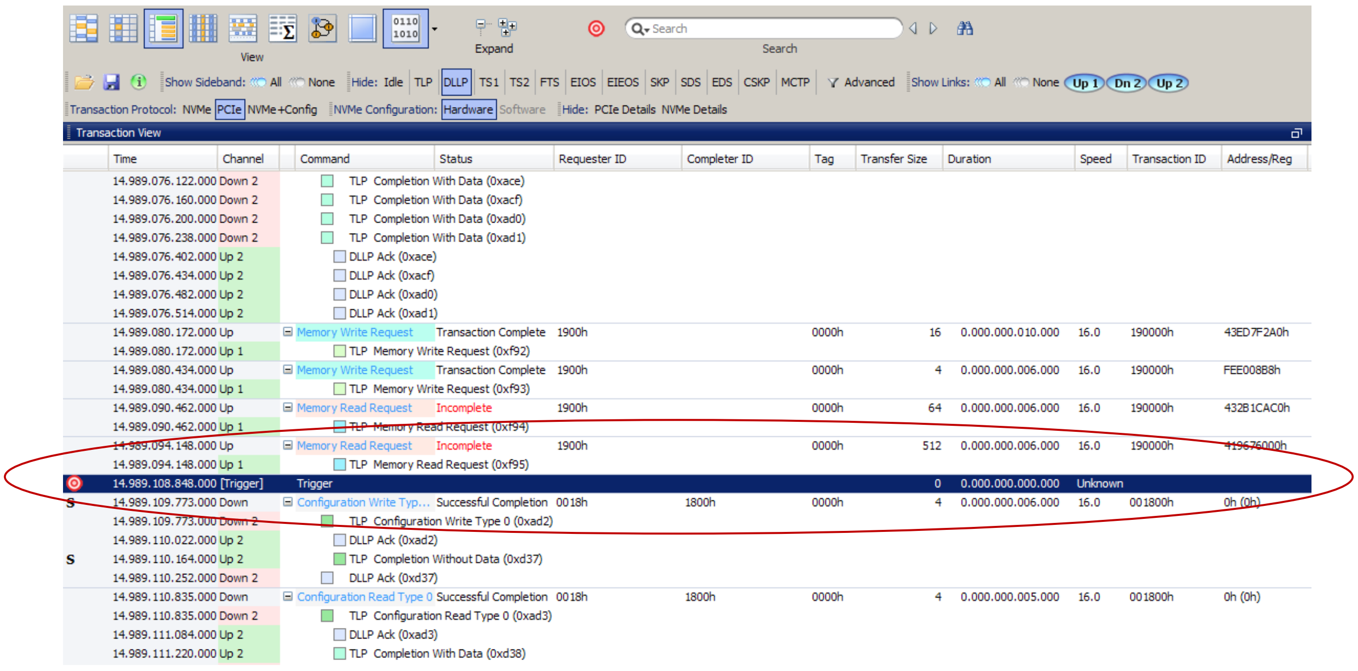
Now Toss in Your Needle and Trigger

With the analyzer collecting data, toss in the needle and trigger using this command:

> **sbecho** Write_needle,1,1,0,0,0,0,102,0,0,0,1,1,0,1:1,1:1,0,-0, > /iport0/target100lun1; **sbecho** trigger=100 > /proc/vlun/nvme

The command above started writing our "needle" pattern (see Appendix A), and then immediately wrote our trigger to the analyzer.

The analyzer will trigger, you may need to manually stop the post trigger data collection if it is still in progress, and you will find our trigger here:

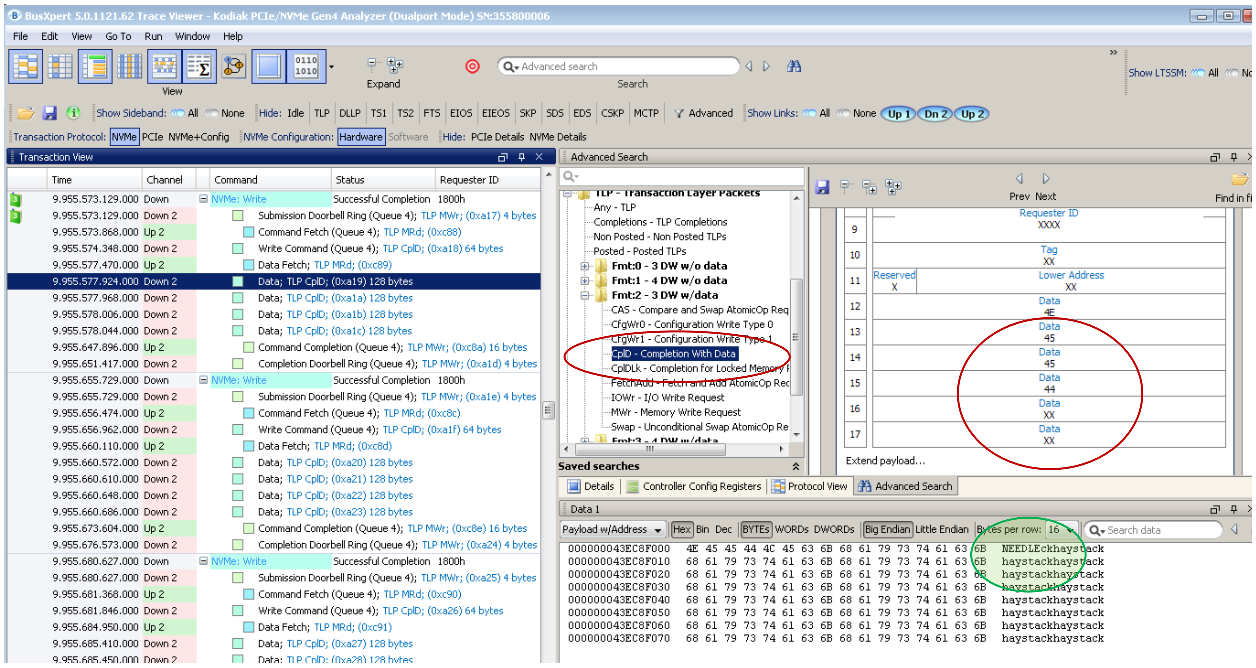

View
Expand
Search
Show Sideband: All None
Hide: Idle TLP DLLP TS1 TS2 FTS EIOS EIEOS SKP SDS EDS CSKP MCTP
Advanced
Show Links: All None Up 1 Dn 2 Up 2

Transaction Protocol: NVMe [PCIe] NVMe+Config | NVMe Configuration: [Hardware] Software | Hide: PCIe Details NVMe Details

Transaction View

	Time	Channel	Command	Status	Requester ID	Completer ID	Tag	Transfer Size	Duration	Speed	Transaction ID	Address/Reg
	14.989.076.122.000	Down 2	TLP Completion With Data (0xace)									
	14.989.076.160.000	Down 2	TLP Completion With Data (0xacf)									
	14.989.076.200.000	Down 2	TLP Completion With Data (0xad0)									
	14.989.076.238.000	Down 2	TLP Completion With Data (0xad1)									
	14.989.076.402.000	Up 2	DLLP Ack (0xace)									
	14.989.076.434.000	Up 2	DLLP Ack (0xacf)									
	14.989.076.482.000	Up 2	DLLP Ack (0xad0)									
	14.989.076.514.000	Up 2	DLLP Ack (0xad1)									
	14.989.080.172.000	Up	Memory Write Request	Transaction Complete	1900h		0000h	16	0.000.000.010.000	16.0	190000h	43ED7F2A0h
	14.989.080.172.000	Up 1	TLP Memory Write Request (0xf92)									
	14.989.080.434.000	Up	Memory Write Request	Transaction Complete	1900h		0000h	4	0.000.000.006.000	16.0	190000h	FEE008B8h
	14.989.080.434.000	Up 1	TLP Memory Write Request (0xf93)									
	14.989.090.462.000	Up	Memory Read Request	Incomplete	1900h		0000h	64	0.000.000.006.000	16.0	190000h	432B1CAC0h
	14.989.090.462.000	Up 1	TLP Memory Read Request (0xf94)									
	14.989.094.148.000	Up	Memory Read Request	Incomplete	1900h		0000h	512	0.000.000.006.000	16.0	190000h	419676000h
	14.989.094.148.000	Up 1	TLP Memory Read Request (0xf95)									
	14.989.108.848.000	[Trigger]	Trigger					0	0.000.000.000.000	Unknown		
S	14.989.109.773.000	Down	Configuration Write Typ...	Successful Completion	0018h	1800h	0000h	4	0.000.000.006.000	16.0	001800h	0h (0h)
	14.989.109.773.000	Down 2	TLP Configuration Write Type 0 (0xad2)									
	14.989.110.022.000	Up 2	DLLP Ack (0xad2)									
S	14.989.110.164.000	Up 2	TLP Completion Without Data (0xd37)									
	14.989.110.252.000	Down 2	DLLP Ack (0xd37)									
	14.989.110.835.000	Down	Configuration Read Type 0	Successful Completion	0018h	1800h	0000h	4	0.000.000.005.000	16.0	001800h	0h (0h)
	14.989.110.835.000	Down 2	TLP Configuration Read Type 0 (0xad3)									
	14.989.111.084.000	Up 2	DLLP Ack (0xad3)									
	14.989.111.220.000	Up 2	TLP Completion With Data (0xd38)									

And searching from our trigger point, we find our missing needle.



Time	Channel	Command	Status	Requester ID
9.955.573.129.000	Down	NVMe: Write	Successful Completion	1800h
9.955.573.129.000	Down 2	Submission Doorbell Ring (Queue 4); TLP MWr; (0xa17) 4 bytes		
9.955.573.868.000	Up 2	Command Fetch (Queue 4); TLP MRd; (0xc88)		
9.955.574.348.000	Down 2	Write Command (Queue 4); TLP CplD; (0xa18) 64 bytes		
9.955.577.470.000	Up 2	Data Fetch; TLP MRd; (0xc89)		
9.955.577.924.000	Down 2	Data; TLP CplD; (0xa19) 128 bytes		
9.955.577.968.000	Down 2	Data; TLP CplD; (0xa1a) 128 bytes		
9.955.578.006.000	Down 2	Data; TLP CplD; (0xa1b) 128 bytes		
9.955.578.044.000	Down 2	Data; TLP CplD; (0xa1c) 128 bytes		
9.955.647.896.000	Up 2	Command Completion (Queue 4); TLP MWr; (0xc8a) 16 bytes		
9.955.651.417.000	Down 2	Completion Doorbell Ring (Queue 4); TLP MWr; (0xa1d) 4 bytes		
9.955.655.729.000	Down	NVMe: Write	Successful Completion	1800h
9.955.655.729.000	Down 2	Submission Doorbell Ring (Queue 4); TLP MWr; (0xa1e) 4 bytes		
9.955.656.474.000	Up 2	Command Fetch (Queue 4); TLP MRd; (0xc8c)		
9.955.656.962.000	Down 2	Write Command (Queue 4); TLP CplD; (0xa1f) 64 bytes		
9.955.660.110.000	Up 2	Data Fetch; TLP MRd; (0xc8d)		
9.955.660.572.000	Down 2	Data; TLP CplD; (0xa20) 128 bytes		
9.955.660.610.000	Down 2	Data; TLP CplD; (0xa21) 128 bytes		
9.955.660.648.000	Down 2	Data; TLP CplD; (0xa22) 128 bytes		
9.955.660.686.000	Down 2	Data; TLP CplD; (0xa23) 128 bytes		
9.955.673.604.000	Up 2	Command Completion (Queue 4); TLP MWr; (0xc8e) 16 bytes		
9.955.676.573.000	Down 2	Completion Doorbell Ring (Queue 4); TLP MWr; (0xa24) 4 bytes		
9.955.680.627.000	Down	NVMe: Write	Successful Completion	1800h
9.955.680.627.000	Down 2	Submission Doorbell Ring (Queue 4); TLP MWr; (0xa25) 4 bytes		
9.955.681.368.000	Up 2	Command Fetch (Queue 4); TLP MRd; (0xc90)		
9.955.681.846.000	Down 2	Write Command (Queue 4); TLP CplD; (0xa26) 64 bytes		
9.955.684.950.000	Up 2	Data Fetch; TLP MRd; (0xc91)		
9.955.685.410.000	Down 2	Data; TLP CplD; (0xa27) 128 bytes		
9.955.685.450.000	Down 2	Data; TLP CplD; (0xa28) 128 bytes		

```
000000043EC8F000   4E 45 45 44 4C 45 63 6B 68 61 79 73 74 61 63 6B   NEEDLEckhaystack
000000043EC8F010   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F020   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F030   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F040   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F050   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F060   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
000000043EC8F070   68 61 79 73 74 61 63 6B 68 61 79 73 74 61 63 6B   haystackhaystack
```

Starting at the SANBlaze Trigger, using "Advanced Search" search for a "Completion with Data" containing 0x4e, 0x45, 0x45, 0x44, and there it is!

Built in Automatic SANBlaze Triggers

The SANBlaze Trigger can be sent manually as in the examples above, but will also be sent automatically if any of the defined error conditions are detected by the driver. You can modify your analyzer trigger by setting your trigger to ignore or detect the optional and built in trigger codes as described below.

Sending the Trigger Manually and User Defined Error Codes

In the example above, we introduced our manual trigger command, which can be sent by the SANBlaze system at any time, from the CLI or from a script.

The basic form of the command in this example is:

`sbecho trigger=100 > /proc/vlun/nvme`
or
`sbecho trigger=100,2211 > /proc/vlun/nvme`

sbecho trigger=X[,Y] > /proc/vlun/nvme

X identifies a controller, either by target number or PCI name. In our example, 100 for Target100.

Y is an optional 16-bit "error code". Error codes are divided into two 8-bit halves, an error type (TT) and an error subtype (SS).

The trigger works by writing the value 0xTTSS1ae3 to the controller's vendor/device registers in PCI config space (these are read-only, so the write will be ignored). This write can be used to trigger an analyzer, where TT and/or SS can be ignored or specified, depending on whether the desire is to trigger on any error or a specific error.

Automatic Triggers Sent by the SANBlaze NVMe System

The trigger is also sent automatically by the driver when certain errors are detected.

Automatic Triggers:

- 0100 -- error requiring Controller Reset
- 0200 -- command timeout
- 0300 -- Controller Fatal Status detected
- 0400 -- error during Controller Reset or Initialization
- 0500 -- error during NVM Subsystem Reset
- 0600 -- error during PCI Reset
- 0700 -- error waiting for Controller Ready (being enabled)
- 0800 -- error waiting for Controller Shutdown (being disabled)
- 0900 -- device is "Gone"

Conclusion

SANBlaze SBExpress system now contains the ability to trigger your PCIe analyzer at the instant we determine an unexpected event has occurred. This SANBlaze Trigger will coordinate your PCIe trace with our log files so that your engineers and ours can cooperatively determine the cause of the error.

The SANBlaze Trigger is the giant electromagnet hanging over your PCIe haystack making it trivial to find the specific needle which caused the failure.

Contact SANBlaze sales for more information at sales@sanblaze.com.

Appendix A - Patterns Used in this Paper

Two patterns were used for the demonstration of the SANBlaze Analyzer Trigger in this paper. They are executed with the following CLI commands.

Start a write test with a given pattern from this directory on the SANBlaze system:

```
pwd
/virtualun/lundata/initpatternfiles
ls
pattern_100 (haystack)
pattern_102 (needle)
```

Start Writing the Haystack Pattern

This command will begin a write test to the NVMe device at Target 100 with Namespace 1.

```
sbecho Write_haystack,1,1,0,0,0,0,100,0,0,0,1,1,0,1:1,1:1,0,-0, >
/iport0/target100lun1
```

Send the Needle Pattern and the SANBlaze Trigger

This command will begin a write test with the needle pattern to the NVMe device at Target 100 with Namespace 1, and immediately send a trigger.

```
sbecho Write_needle,1,1,0,0,0,0,102,0,0,0,1,1,0,1:1,1:1,0,-0, > /iport0/target100lun1;
sbecho trigger=100 > /proc/vlun/nvme
```

pattern_100 (haystack)

```
hexdump -Cv pattern_100
00000000  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000010  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000020  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000030  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000040  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000050  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000060  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000070  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000080  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000090  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000a0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000b0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000c0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000d0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000e0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000f0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000100  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000110  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000120  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000130  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000140  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000150  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000160  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000170  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000180  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000190  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001a0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001b0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001c0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001d0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001e0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001f0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
```

pattern_102 (needle)

```
hexdump -Cv pattern_102
00000000  4e 45 45 44 4c 45 63 6b  68 61 79 73 74 61 63 6b  |NEEDLEckhaystack|
00000010  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000020  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000030  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000040  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000050  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000060  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000070  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000080  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000090  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000a0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000b0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000c0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000d0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000e0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000000f0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000100  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000110  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000120  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000130  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000140  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000150  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000160  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000170  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000180  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
00000190  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001a0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001b0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001c0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001d0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001e0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
000001f0  68 61 79 73 74 61 63 6b  68 61 79 73 74 61 63 6b  |haystackhaystack|
```

These specific example data patterns were created to illustrate the SANBlaze Trigger. If you wish to reproduce them, the following commands will create the pattern files:

```
cd /virtualun/lundata/initpatternfiles
for (( i=0; i<64; i++ ));do echo -n haystack >> pattern_100;done
echo -n NEEDLEck > pattern_102
for (( i=0; i<63; i++ ));do echo -n haystack >> pattern_102;done
```