STA™
SCSI Trade Association
A SNIA Community

Reduce Your Risk of Data Loss!

Critical Testing Insights for Developers

Paul Coddington
Amphenol

Craig Foster
Teledyne LeCroy

Rick Kutcipal
Broadcom

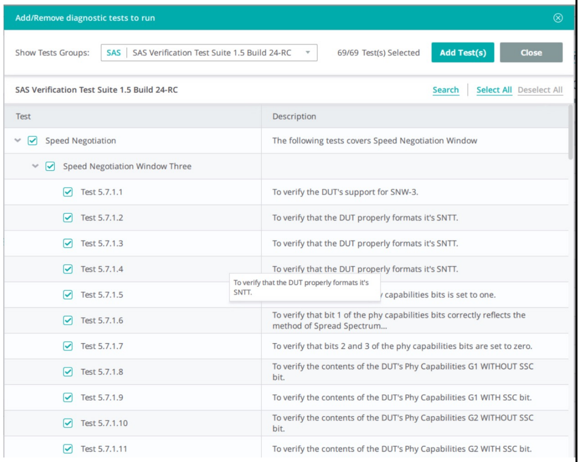
Webinar
September 25, 2025
10:00am PT / 1:00pm ET

Automation of UNH Defined Verification Tests

UNH IOL SERIAL ATTACHED SCSI (SAS) CONSORTIUM

Clause 5
SAS 2.0 Speed Negotiation Test Suite
Version 0.9

Technical Document



Add/Remove diagnostic tests to run

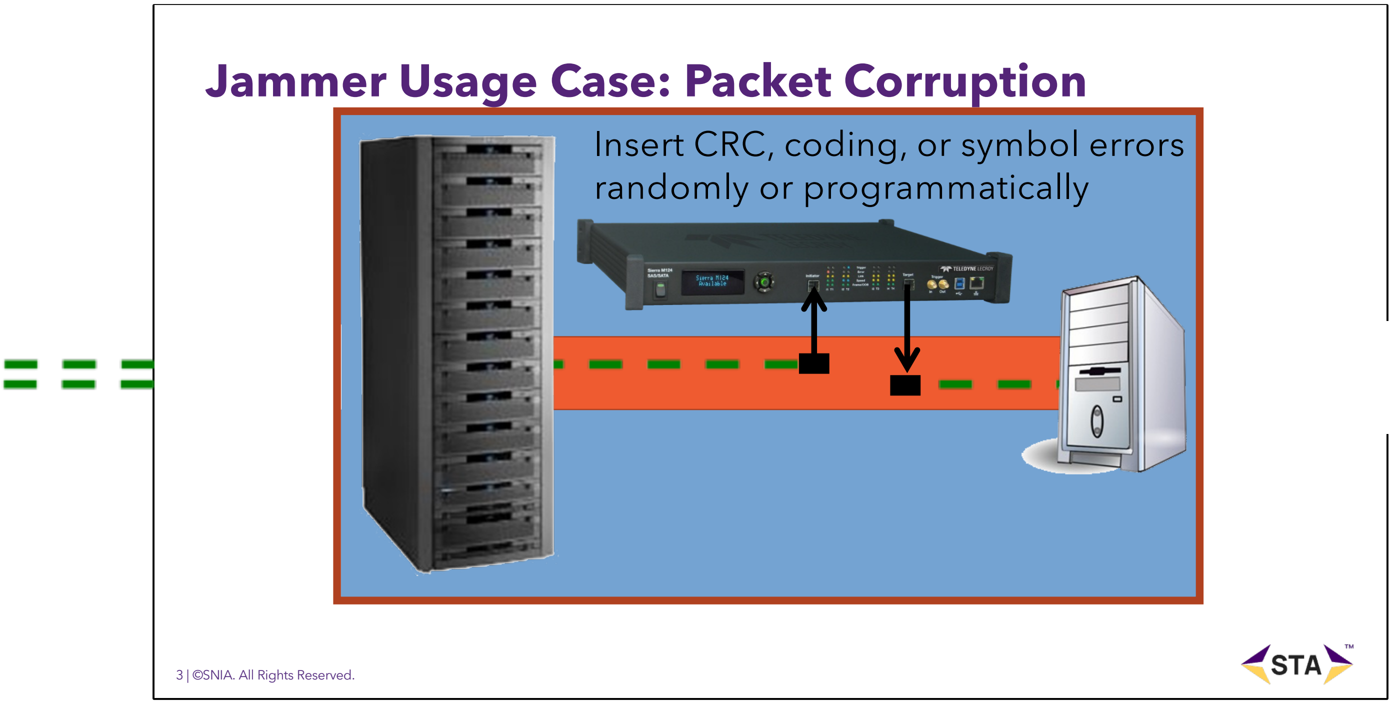
Show Tests Groups: SAS | SAS Verification Test Suite 1.5 Build 24-RC ▾   69/69 Test(s) Selected   Add Test(s)   Close

SAS Verification Test Suite 1.5 Build 24-RC   Search | Select All Deselect All

Test	Description
☑ Speed Negotiation	The following tests covers Speed Negotiation Window
☑ Speed Negotiation Window Three	
☑ Test 5.7.1.1	To verify the DUT's support for SNW-3.
☑ Test 5.7.1.2	To verify that the DUT properly formats it's SNTT.
☑ Test 5.7.1.3	To verify that the DUT properly formats it's SNTT.
☑ Test 5.7.1.4	To verify that the DUT properly formats it's SNTT.
☑ Test 5.7.1.5	y capabilities bits is set to one.
☑ Test 5.7.1.6	To verify that bit 1 of the phy capabilities bits correctly reflects the method of Spread Spectrum...
☑ Test 5.7.1.7	To verify that bits 2 and 3 of the phy capabilities bits are set to zero.
☑ Test 5.7.1.8	To verify the contents of the DUT's Phy Capabilities G1 WITHOUT SSC bit.
☑ Test 5.7.1.9	To verify the contents of the DUT's Phy Capabilities G1 WITH SSC bit.
☑ Test 5.7.1.10	To verify the contents of the DUT's Phy Capabilities G2 WITHOUT SSC bit.
☑ Test 5.7.1.11	To verify the contents of the DUT's Phy Capabilities G2 WITH SSC bit.

To verify that the DUT properly formats it's SNTT.

This video will cover how protocol analyzers with jammer and exerciser capabilities can be used to test and validate different aspects of the SAS protocol. The University of New Hampshire's interoperability lab defines tests for many different protocols and technologies. They defined some SAS related verification tests. While the tests were defined for SAS 2.0 Speeds, we have updated our exerciser tests to cover 24G as well. By automating these test, companies can quickly run verification test on their devices and identify and resolve issues early in the development cycle.

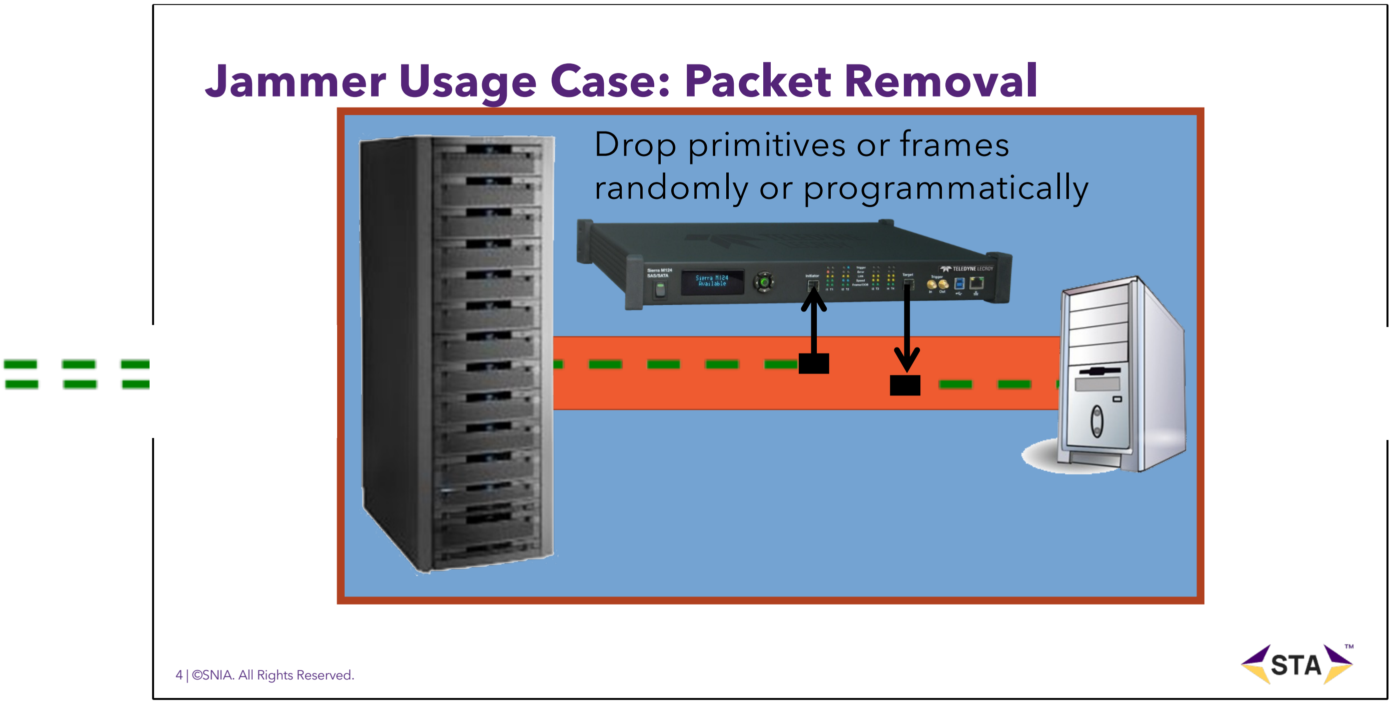


Jammer Usage Case: Packet Corruption

Insert CRC, coding, or symbol errors randomly or programmatically

3 | ©SNIA. All Rights Reserved.

Using the Jammer option on a protocol analyzer, validation teams can inject CRC, coding, or symbol errors either randomly or programmatically. This allows us to simulate real-world fault conditions and observe how devices respond to corrupted packet which is critical for validating error recovery mechanisms



Jammer Usage Case: Packet Removal

Drop primitives or frames randomly or programmatically

4 | ©SNIA. All Rights Reserved.

In addition to introducing errors, jammers can drop packets and or primitives. Here, we demonstrate how the jammer can drop primitives not just a frame. This is useful for testing timeout scenarios and retry logic in SAS devices. The analyzer / jammer can monitor both pre- and post-error traffic to analyze system behavior under stress

SAS primitives are low-level control signals used to manage link-level communication. Examples include:

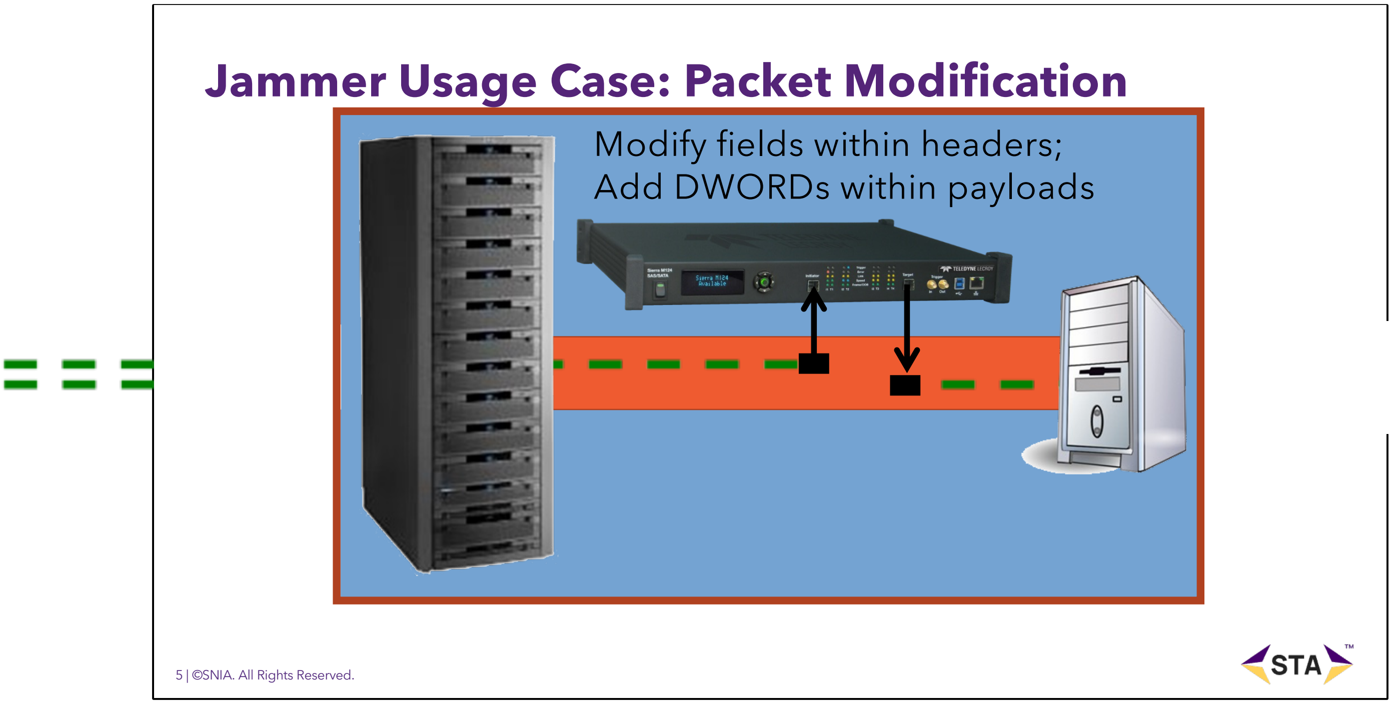
ALIGN: Used for word alignment.

IDLE: Indicates no data is being transmitted.

XRDY / RRDY: Transmit/Receive Ready.

SYNCP / SYNC: Synchronization primitives.

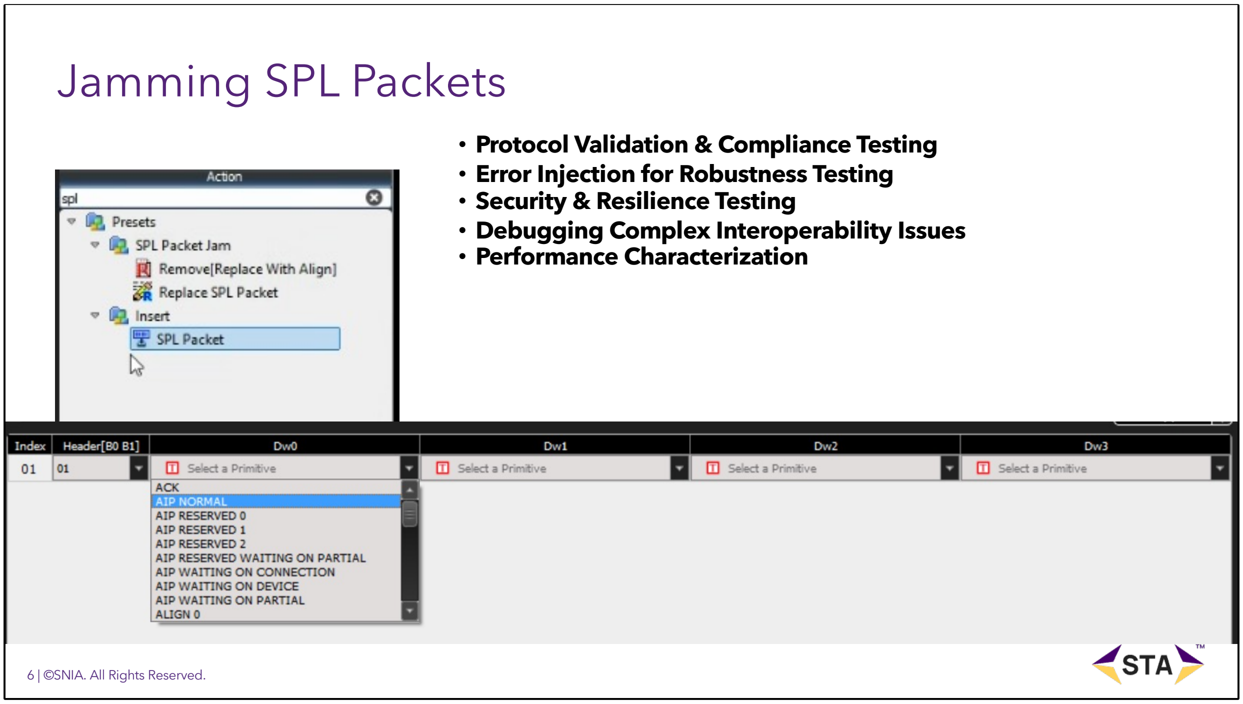
BREAK / CONTINUE: Used for link resets or flow control.



Jammer Usage Case: Packet Modification

Modify fields within headers;
Add DWORDs within payloads

Packet modification involves altering header fields or inserting DWORDs into payloads. This helps verify how devices handle unexpected or malformed data. The jammer platform recalculates CRCs and FEC encoding to maintain protocol integrity during injection



Jamming SPL Packets

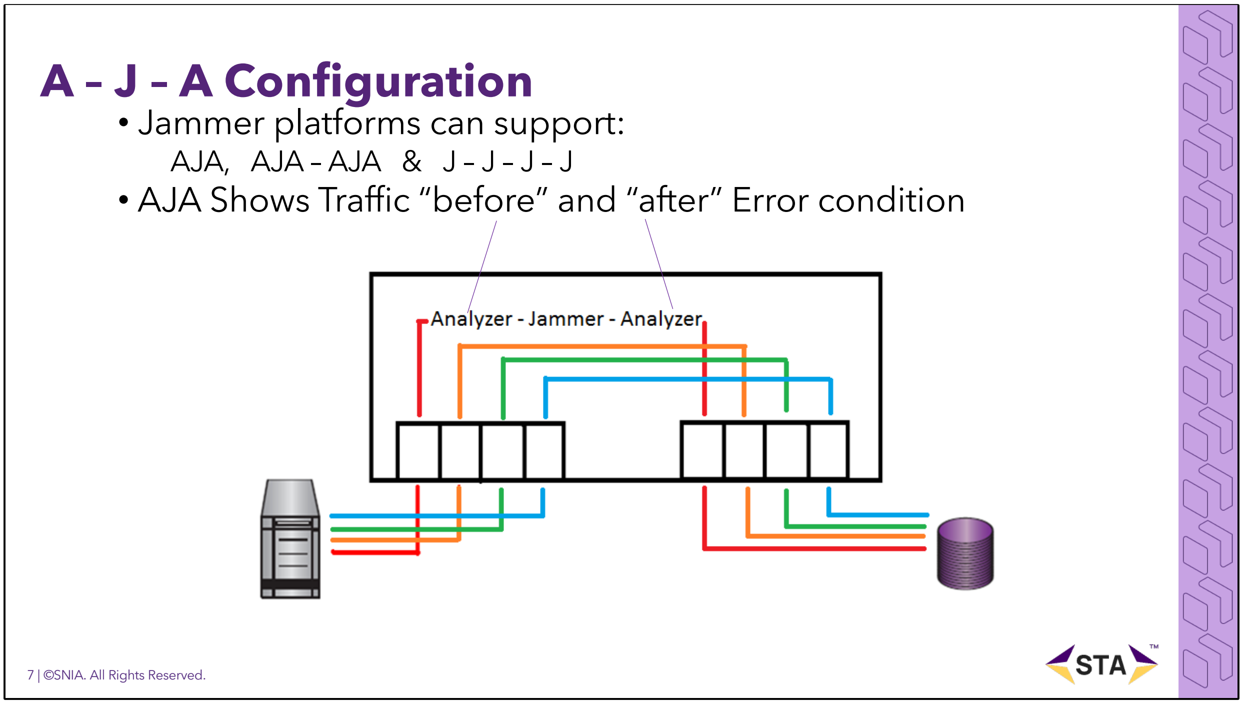
- **Protocol Validation & Compliance Testing**
- **Error Injection for Robustness Testing**
- **Security & Resilience Testing**
- **Debugging Complex Interoperability Issues**
- **Performance Characterization**

6 | ©SNIA. All Rights Reserved.

In SAS-4 **SPL packets** were introduced as part of the physical layer encoding scheme. These packets are not protocol-level constructs like SSP frames; rather, they are low-level transmission units used to carry data and control primitives across the link. SPL packets are built on **128b/150b encoding** with **forward error correction (FEC)**, and their role is similar to **Fibre Channel's 66-bit symbols—** they define how bits are grouped and transmitted, but they do not interpret or process SCSI commands. This encoding enables higher data rates and improved signal integrity, especially during link training and high-speed operation. To clarify on the higher data rates which may seem counter intuitive due to the overhead: Without robust encoding, PHYs would struggle to maintain reliable communication at 24G due to noise, jitter, and crosstalk.
Encoding makes it **feasible to operate at higher line rates**, even though it slightly reduces the usable bandwidth.

SPL packet-level control is essential for deep protocol validation. Since SPL packets are the fundamental transmission units in SAS-4, being able to remove, modify, or insert them allows engineers to simulate low-level faults, timing issues, or malformed traffic that would be difficult to generate otherwise. This is critical for testing how devices handle error correction, link recovery, and adaptive equalization.

Jammer-level control over SPL packets also enables precise manipulation during link training and speed negotiation, helping validate conformance to the SAS spec and uncover interoperability issues. It’s a powerful tool for stress testing PHY behavior, verifying FEC performance, and ensuring robust operation in high-speed SAS environments.



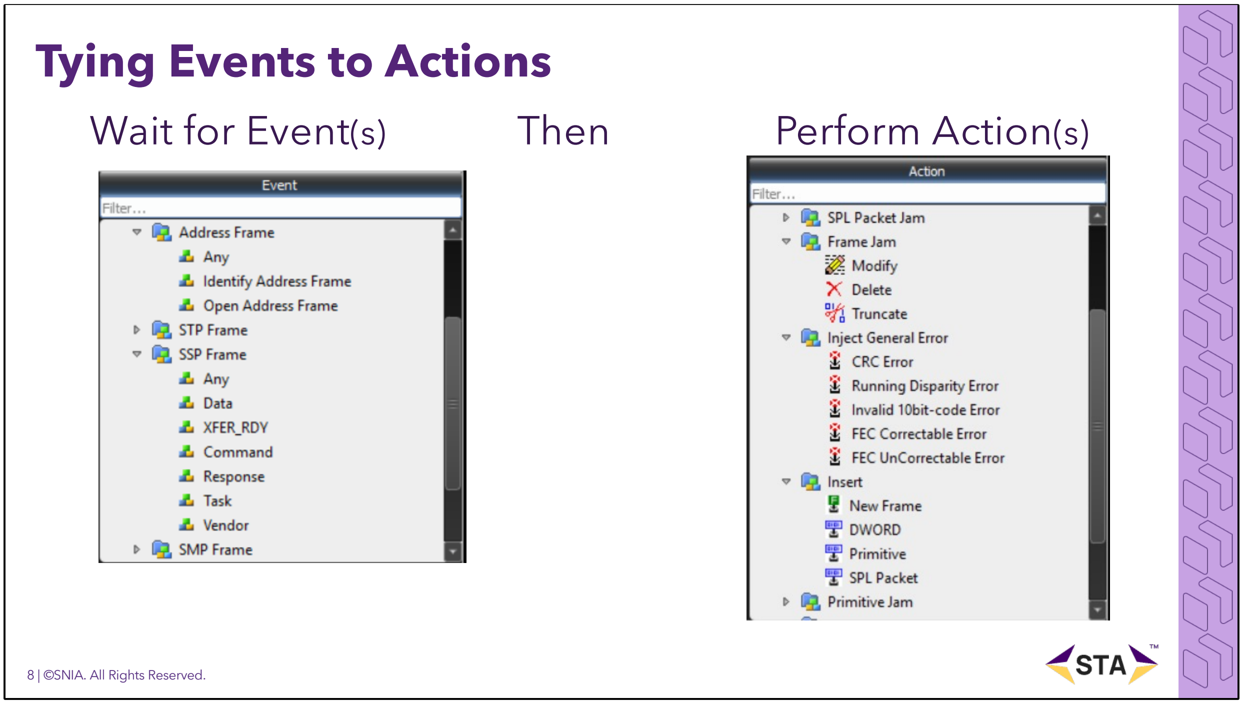
A - J - A Configuration

- Jammer platforms can support:

  AJA, AJA - AJA & J - J - J - J

- AJA Shows Traffic “before” and “after” Error condition

It is often important to see the original packet that came in as well as the modified. The A–J–A setup shows traffic before and after error injection. Depending on the speed targeted, a separate analyzer may be needed for analysis which will be combined in the GUI. This dual-channel view is essential for understanding how injected faults affect communication. The jammer supports various configurations including AJA–AJA and J–J–J–J for flexible testing



Tying Events to Actions

Wait for Event(s) Then Perform Action(s)

Event

Filter...

- Address Frame
  - Any
  - Identify Address Frame
  - Open Address Frame
- STP Frame
- SSP Frame
  - Any
  - Data
  - XFER_RDY
  - Command
  - Response
  - Task
  - Vendor
- SMP Frame

Action

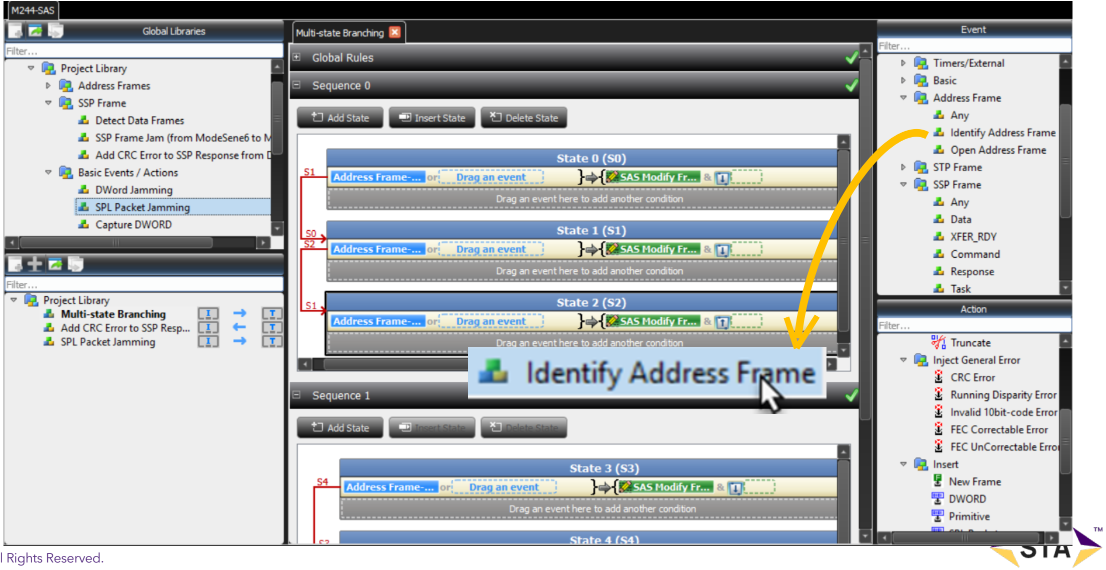
Filter...

- SPL Packet Jam
- Frame Jam
  - Modify
  - Delete
  - Truncate
- Inject General Error
  - CRC Error
  - Running Disparity Error
  - Invalid 10bit-code Error
  - FEC Correctable Error
  - FEC UnCorrectable Error
- Insert
  - New Frame
  - DWORD
  - Primitive
  - SPL Packet
- Primitive Jam

It's often important for the Jammer's actions to occur when the device is already in a specific state. By looking for specific events, the analyzer can take various actions based on the sate of the environment.

Use Case-Driven Protocol Jamming with State Control

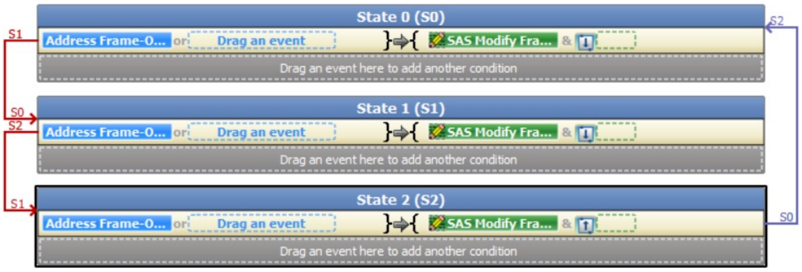
Drag Events/Actions to Sequencer to create test scenarios



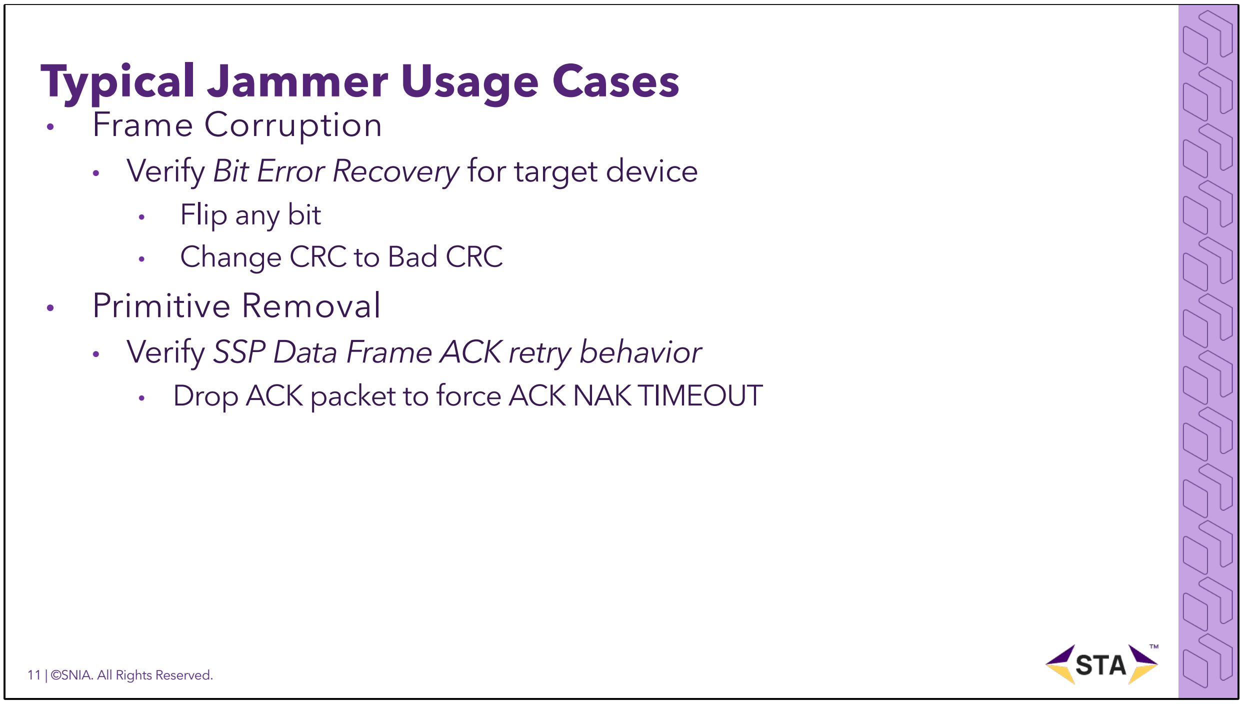
In addition to looking for a single event, a state machine can be created to identify complex states of the traffic and trigger or jam only when certain conditions are met.

Multiple Sequential States with Branching

- Quickly and Easily Create sophisticated test scenarios



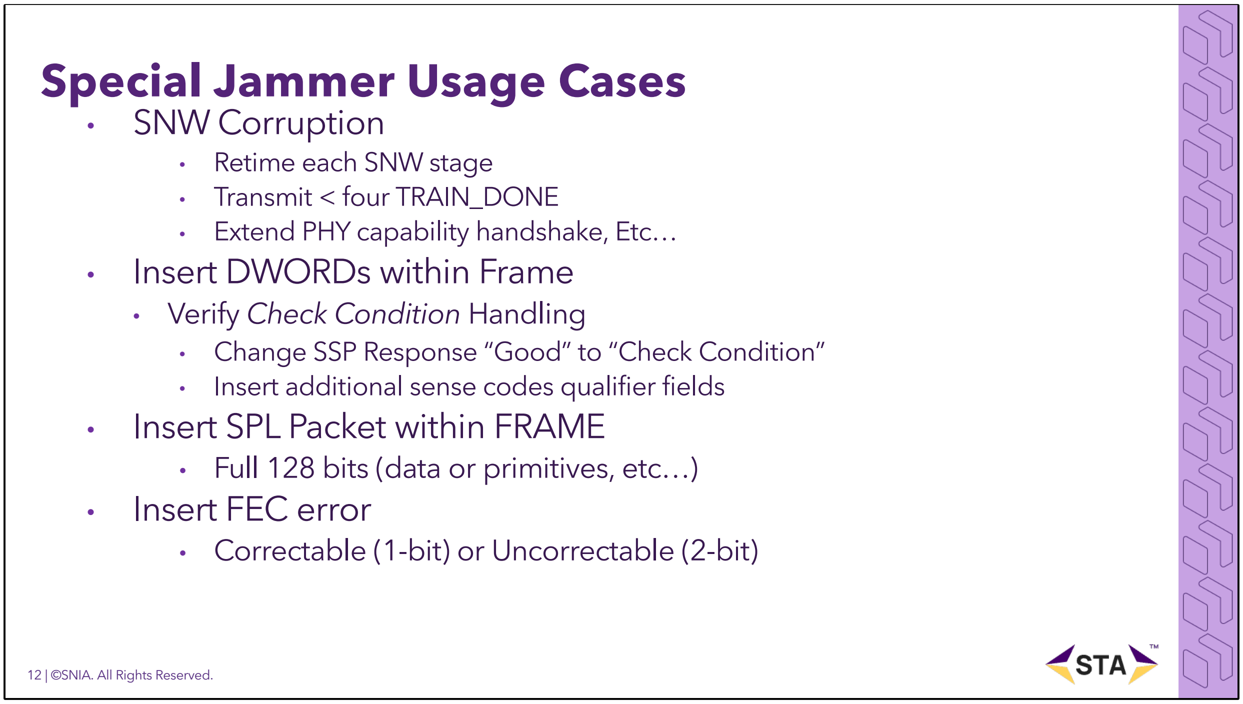
Advanced test scenarios can include multiple states with branching logic. This enables simulation of intricate protocol behaviors and fault conditions, making it easier to validate device robustness



Typical Jammer Usage Cases

- Frame Corruption
  - Verify *Bit Error Recovery* for target device
    - Flip any bit
    - Change CRC to Bad CRC
- Primitive Removal
  - Verify *SSP Data Frame ACK retry behavior*
    - Drop ACK packet to force ACK NAK TIMEOUT

Common use cases include frame corruption, bit flipping, CRC errors, and primitive removal. These help verify error recovery features like ACK retry behavior and timeout handling in SAS devices



Special Jammer Usage Cases

- SNW Corruption
    - Retime each SNW stage
    - Transmit < four TRAIN_DONE
    - Extend PHY capability handshake, Etc…
- Insert DWORDs within Frame
    - Verify *Check Condition* Handling
        - Change SSP Response "Good" to "Check Condition"
        - Insert additional sense codes qualifier fields
- Insert SPL Packet within FRAME
    - Full 128 bits (data or primitives, etc…)
- Insert FEC error
    - Correctable (1-bit) or Uncorrectable (2-bit)

Advanced scenarios include speed negotiation window corruption, PHY handshake extension, and FEC error injection. These tests push devices to their limits and ensure compliance with SAS 4.0 specifications

Frame Modification Example: Check Condition

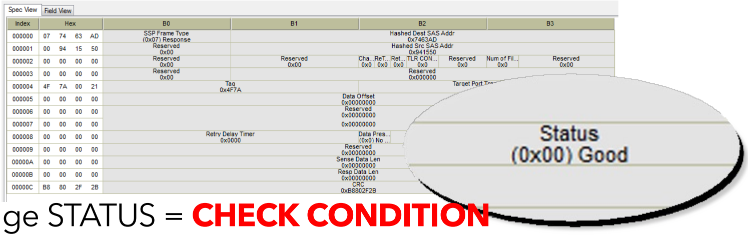
- Verify *Check Condition* Handling
  - Change SSP Response "Good" to SSP Response "CHECK CONDITION", Append additional sense codes qualifier fields: ABORT TASK with ASC/Q of "*Overlapped Command*"

Sierra M244 SAS/SATA
TELEDYNE LECROY
Trigger
Error
Link
Speed
Frame/OOB
I1 T1
I2 T2
Initiator
Target
I3 T3
I4 T4
TELEDYNE LECROY
Trigger
In
Out
STA

In SCSI-based transports, it's important to verify how initiators respond to error conditions. One common method is to replace a normal SSP response—such as "GOOD"—with a "CHECK CONDITION" status. This allows us to simulate fault scenarios and observe how the initiator reacts. The goal here is not to test specific sense codes yet, but to confirm that the system correctly transitions into error-handling mode when a check condition is received. This sets the stage for more advanced fault injection techniques.

Frame Modification Example: Check Condition

Wait for Response with STATUS = **GOOD**



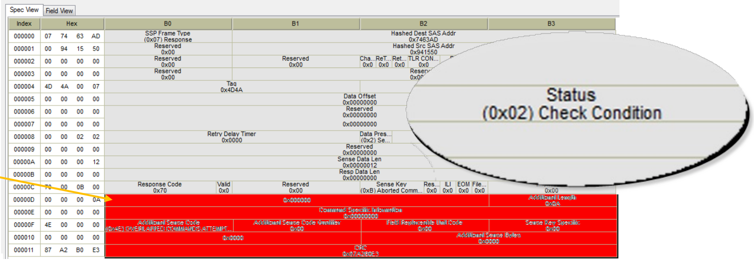
Spec View | Field View

Index	Hex	B0	B1	B2	B3
000000	07 74 63 AD	SSP Frame Type (0x07) Response	Hashed Dest SAS Addr 0x7463AD		
000001	00 94 15 50	Reserved 0x00	Hashed Src SAS Addr 0x941550		
000002	00 00 00 00	Reserved 0x00	Reserved 0x00	Cha... 0x0 ReT... 0x0 Ret... 0x0 TLR CON... 0x0 Reserved 0x0	Num of Fil... 0x0 Reserved 0x00
000003	00 00 00 00	Reserved 0x00	Reserved 0x000000		
000004	4F 7A 00 21	Tag 0x4F7A		Target Port Tr...	
000005	00 00 00 00	Data Offset 0x00000000			
000006	00 00 00 00	Reserved 0x00000000			
000007	00 00 00 00	0x00000000			
000008	00 00 00 00	Retry Delay Timer 0x0000		Data Pres... (0x0) No ...	
000009	00 00 00 00	Reserved 0x00000000			
00000A	00 00 00 00	Sense Data Len 0x00000000			
00000B	00 00 00 00	Resp Data Len 0x00000000			
00000C	B8 80 2F 2B	CRC 0xB8802F2B			

Status
(0x00) Good

Change STATUS = **CHECK CONDITION**

Insert Additional Sense Codes = **INVALID COMMAND**



Spec View | Field View

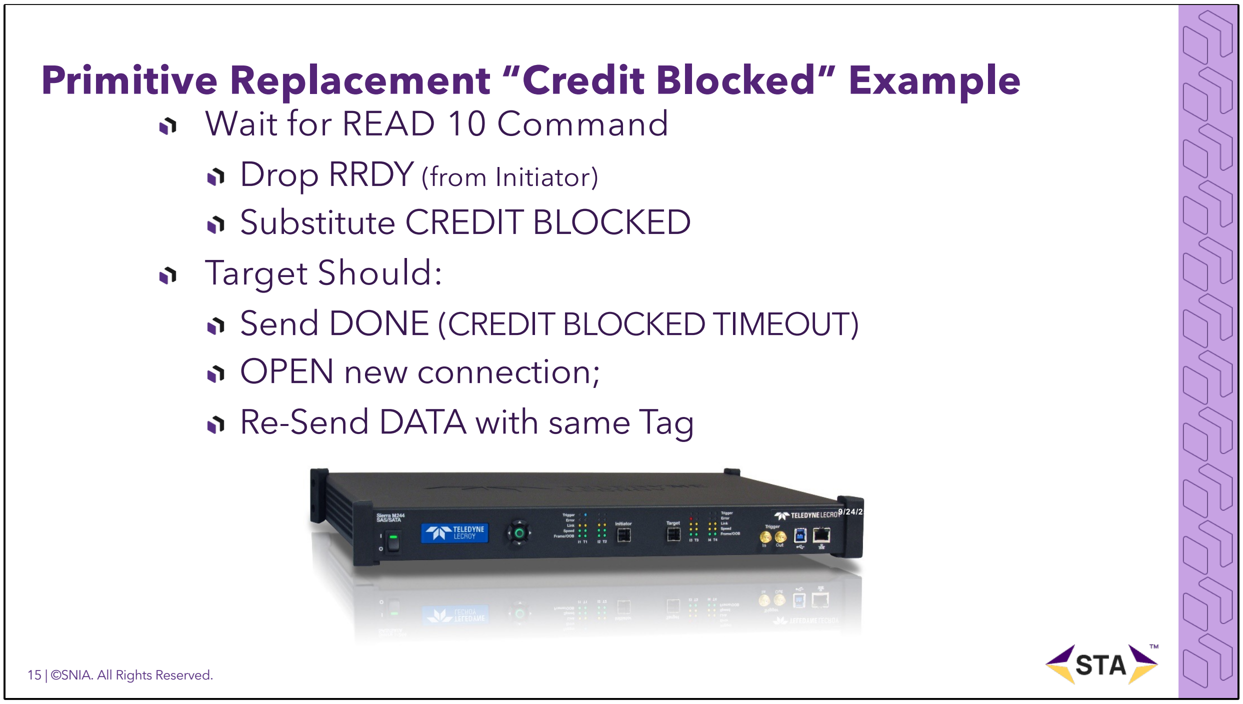
Index	Hex	B0	B1	B2	B3
000000	07 74 63 AD	SSP Frame Type (0x07) Response	Hashed Dest SAS Addr 0x7463AD		
000001	00 94 15 50	Reserved 0x00	Hashed Src SAS Addr 0x941550		
000002	00 00 00 00	Reserved 0x00	Reserved 0x00	Cha... 0x0 ReT... 0x0 Ret... 0x0 TLR CON... 0x0	
000003	00 00 00 00	Reserved 0x00		Reserv 0x00	
000004	4D 4A 00 07	Tag 0x4D4A			
000005	00 00 00 00	Data Offset 0x00000000			
000006	00 00 00 00	Reserved 0x00000000			
000007	00 00 00 00	0x00000000			
000008	00 00 02 02	Retry Delay Timer 0x0000		Data Pres... (0x2) Se...	
000009	00 00 00 00	Reserved 0x00000000			
00000A	00 00 00 12	Sense Data Len 0x00000012			
00000B	00 00 00 00	Resp Data Len 0x00000000			
00000C	70 00 0B 00	Response Code 0x70 Valid 0x0	Reserved 0x00	Sense Key (0xB) Aborted Comm... Res... 0x0 ILI 0x0 EOM 0x0 File... 0x0	0x00
00000D	00 00 00 0A	0x000000			Additional Length 0x0A
00000E	00 00 00 00	Command Specific Information 0x00000000			
00000F	4E 00 00 00	Additional Sense Code (0x4E) OVERLAPPED COMMANDS ATTEMPT...	Additional Sense Code Qualifier 0x00	Field Replaceable Unit Code 0x00	Sense Key Specific 0x00
000010	00 00 00 00	0x0000		Additional Sense Bytes 0x0000	
000011	87 A2 B0 E3	CRC 0x87A2B0E3			

Status
(0x02) Check Condition

Additional Sense Codes

14 | ©SNIA. All Rights Reserved.

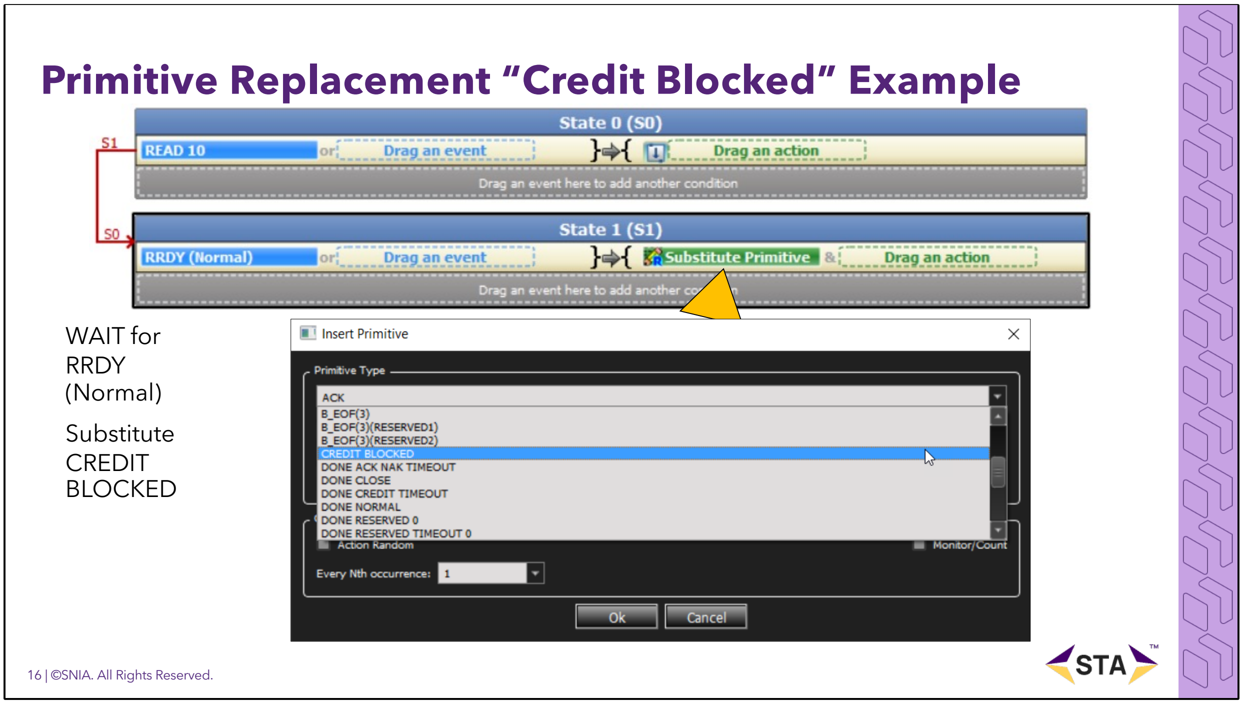
Building on that concept, this example demonstrates how a **jammer** can be used to intercept and modify a valid SSP response, transforming it into a **CHECK CONDITION** with a specific error type—such as an **invalid command**—and a custom **Additional Sense Code (ASC)** and **Qualifier (ASCQ)**. For instance, we can simulate an "ABORT TASK" scenario with an ASC/Q of "Overlapped Command." This kind of targeted fault injection is essential for validating how initiators interpret and respond to detailed sense data. It helps ensure that error recovery logic, logging mechanisms, and diagnostic tools behave correctly under edge-case conditions. This technique is especially valuable in conformance testing and debugging complex SAS environments.



Primitive Replacement "Credit Blocked" Example

- Wait for READ 10 Command
  - Drop RRDY (from Initiator)
  - Substitute CREDIT BLOCKED
- Target Should:
  - Send DONE (CREDIT BLOCKED TIMEOUT)
  - OPEN new connection;
  - Re-Send DATA with same Tag

In many protocols, flow control is key to performance and understanding and preventing performance issue. Often jammers are used to exhaust credit and observe both performance impact as well robustness and performance of error handling with compromised flow control.



Primitive Replacement “Credit Blocked” Example

Primitive Type: CREDIT BLOCKED

Every Nth occurrence: 1

WAIT for RRDY (Normal)

Substitute CREDIT BLOCKED

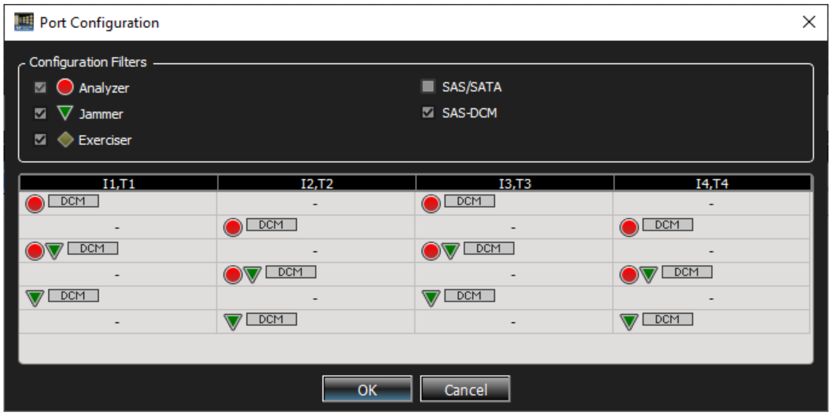
16 | ©SNIA. All Rights Reserved.

STA™

By substituting CREDIT BLOCKED primitives, we can test how targets respond to flow control issues. This includes verifying timeout behavior and reconnection logic

Special “DCM” Jammer & Analysis

DCM Jammer (Error Injection)



Configuration Filters

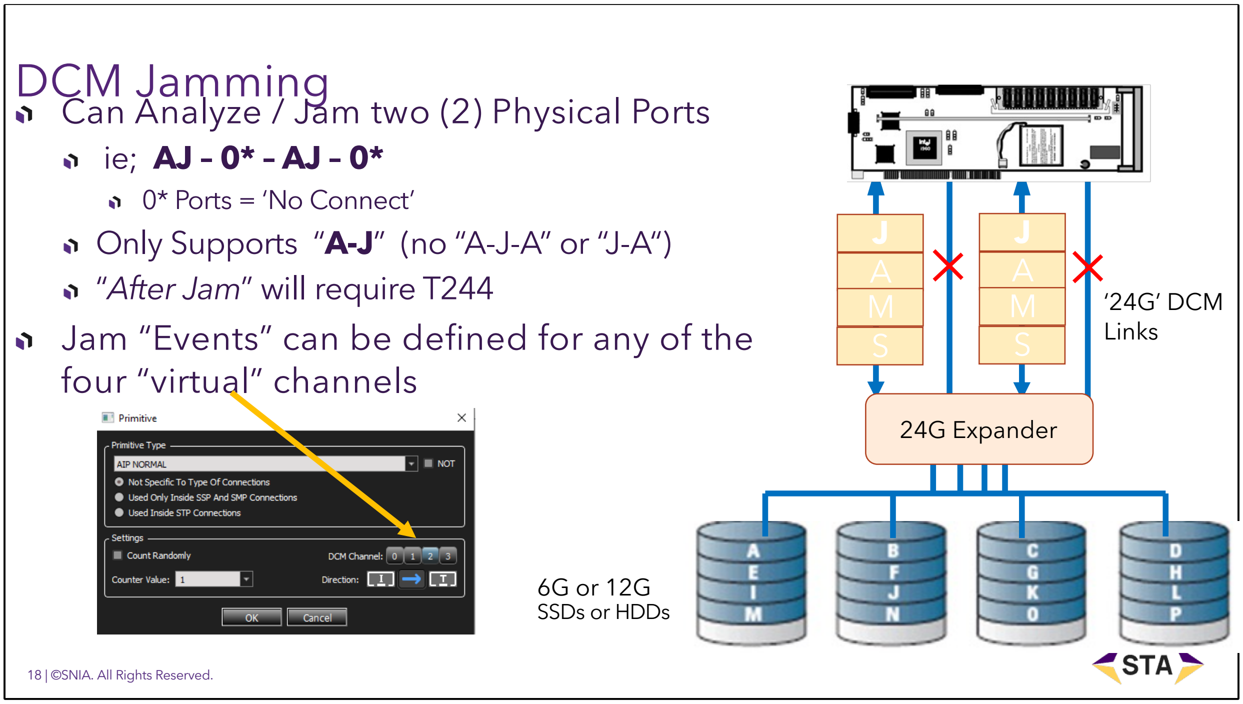
- ☑ ● Analyzer
- ☑ ▼ Jammer
- ☑ ◆ Exerciser
- □ SAS/SATA
- ☑ SAS-DCM

I1,T1	I2,T2	I3,T3	I4,T4
● DCM	-	● DCM	-
-	● DCM	-	● DCM
● ▼ DCM	-	● ▼ DCM	-
-	● ▼ DCM	-	● ▼ DCM
▼ DCM	-	▼ DCM	-
-	▼ DCM	-	▼ DCM

Dynamic Channel Multiplexing (DCM) is a link aggregation feature introduced in 24G SAS to improve bandwidth utilization and reduce latency, particularly between initiators and expanders or between expanders themselves. It enables up to four virtual channels over a single 24G physical link, allowing traffic equivalent to 4×6G or 2×12G streams to be carried simultaneously.

DCM works by tagging each SPL packet with a channel identifier, allowing the PHY to dynamically route traffic across multiple logical paths. This breaks the traditional one-to-one initiator-target model and supports cut-through routing, reducing blocking and improving overall efficiency.

DCM is negotiated automatically during link setup and is not used between initiators and targets. Instead, it enhances routing and aggregation in the SAS fabric without requiring changes to existing SAS or SATA devices. By reducing cabling complexity and improving throughput, DCM helps extend the utility of legacy devices while eliminating fairness issues like drive starvation.



DCM Jamming

- Can Analyze / Jam two (2) Physical Ports
  - ie; **AJ - 0* - AJ - 0***
    - 0* Ports = 'No Connect'
  - Only Supports "**A-J**" (no "A-J-A" or "J-A")
  - "*After Jam*" will require T244
- Jam "Events" can be defined for any of the four "virtual" channels

Primitive ×

Primitive Type

AIP NORMAL ▾ □ NOT

● Not Specific To Type Of Connections

○ Used Only Inside SSP And SMP Connections

○ Used Inside STP Connections

Settings

□ Count Randomly

DCM Channel: 0 1 [✓ 2] 3

Counter Value: 1 ▾

Direction: [I] → [I]

OK Cancel

'24G' DCM Links

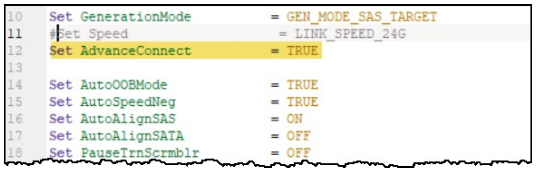
24G Expander

6G or 12G
SSDs or HDDs

By jamming different virtual lanes, we can simulate noisy or hostile environments. Again, the goal is to test different aspects for error handling and link recovery. SAS uses **multiple virtual channels** (especially in wide-port configurations). Jamming one or more channels can help assess **crosstalk** and **signal isolation** between them. In addition, SAS uses different fairness algorithms that can be tested with this methodology. It is also useful when stress testing the environment for performance.

AdvanceConnect Automatically Performs 24G Link-Up

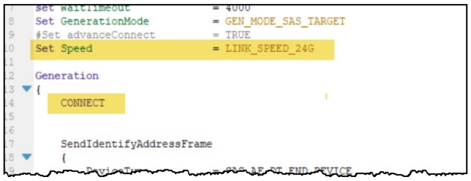
- When
  - *AdvanceConnect = True:*
    - Automatically Performs:



```
Set GenerationMode            = GEN_MODE_SAS_TARGET
#Set Speed                     = LINK_SPEED_24G
Set AdvanceConnect            = TRUE

Set AutoOOBMode               = TRUE
Set AutoSpeedNeg              = TRUE
Set AutoAlignSAS              = ON
Set AutoAlignSATA             = OFF
Set PauseTrnScrmblr           = OFF
```

    - Links at highest supported rate
- Else
  - *Set Speed = LINK_SPEED_24G*
  - Combined with CONNECT
    - Automatically Performs:



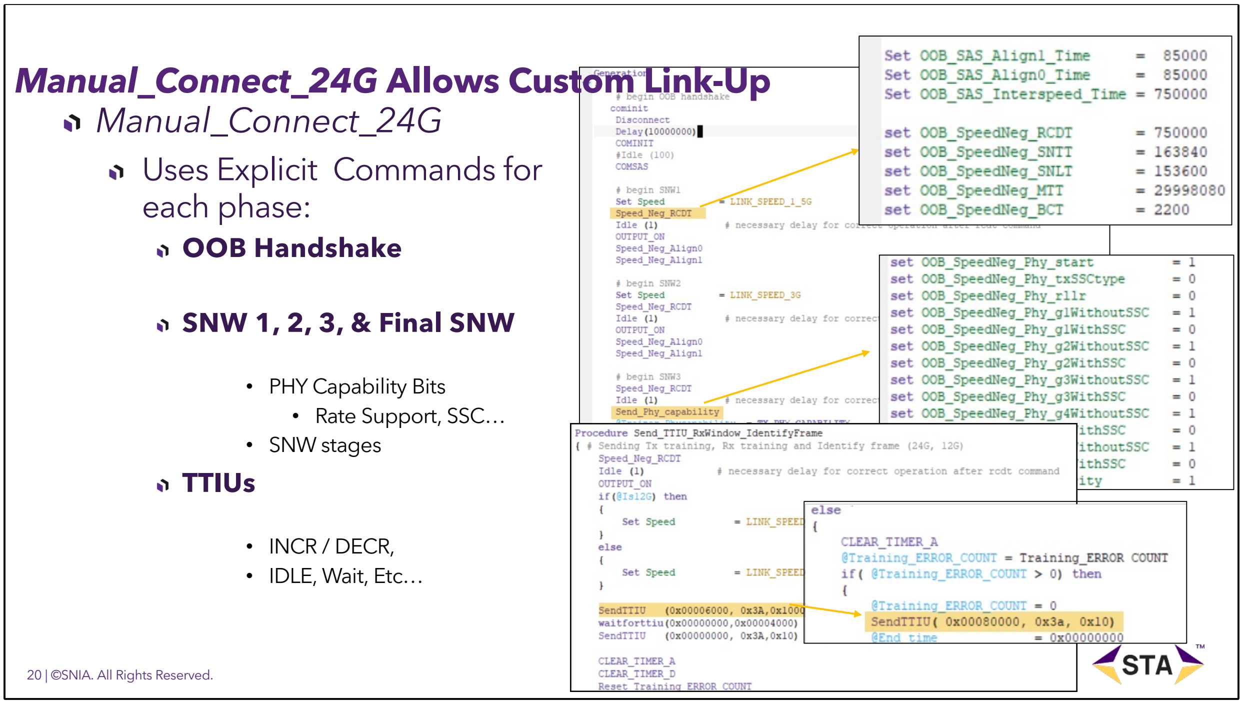
```
Set WaitTimeout               = 4000
Set GenerationMode            = GEN_MODE_SAS_TARGET
#Set advanceConnect            = TRUE
Set Speed                     = LINK_SPEED_24G

Generation
{
    CONNECT


    SendIdentifyAddressFrame
    {
        DeviceType            = SAS_AF_DT_END_DEVICE
```

    - Forces link at 24G rate

For more granular control scripts can be created to simulate a variety of device related implementations. In this case we are looking specifically at the capabilities related to link bring up. Advance Connect automates 24G link-up sequences including Speed Negotiation Windows stages and training. There is also a manual connect option which allows customization of each phase, offering granular control over link negotiation and PHY capabilities. This enables users to test their equipment as if they were connecting to a device that may not be available to the validation team.



Manual_Connect_24G Allows Custom Link-Up

- *Manual_Connect_24G*
  - Uses Explicit Commands for each phase:
    - **OOB Handshake**
    - **SNW 1, 2, 3, & Final SNW**
      - PHY Capability Bits
        - Rate Support, SSC…
      - SNW stages
    - **TTIUs**
      - INCR / DECR,
      - IDLE, Wait, Etc…

```
Generation
   # begin OOB handshake
  cominit
   Disconnect
   Delay(10000000)
   COMINIT
   #Idle (100)
   COMSAS

   # begin SNW1
   Set Speed            = LINK_SPEED_1_5G
   Speed_Neg_RCDT
   Idle (1)              # necessary delay for correct operation after rcdt command
   OUTPUT_ON
   Speed_Neg_Align0
   Speed_Neg_Align1

   # begin SNW2
   Set Speed            = LINK_SPEED_3G
   Speed_Neg_RCDT
   Idle (1)              # necessary delay for correct
   OUTPUT_ON
   Speed_Neg_Align0
   Speed_Neg_Align1

   # begin SNW3
   Speed_Neg_RCDT
   Idle (1)              # necessary delay for correct
   Send_Phy_capability
```

```
Set OOB_SAS_Align1_Time      =  85000
Set OOB_SAS_Align0_Time      =  85000
Set OOB_SAS_Interspeed_Time = 750000

set OOB_SpeedNeg_RCDT        = 750000
set OOB_SpeedNeg_SNTT        = 163840
set OOB_SpeedNeg_SNLT        = 153600
set OOB_SpeedNeg_MTT         = 29998080
set OOB_SpeedNeg_BCT         = 2200
```

```
set OOB_SpeedNeg_Phy_start          = 1
set OOB_SpeedNeg_Phy_txSSCtype      = 0
set OOB_SpeedNeg_Phy_r1lr           = 0
set OOB_SpeedNeg_Phy_g1WithoutSSC   = 1
set OOB_SpeedNeg_Phy_g1WithSSC      = 0
set OOB_SpeedNeg_Phy_g2WithoutSSC   = 1
set OOB_SpeedNeg_Phy_g2WithSSC      = 0
set OOB_SpeedNeg_Phy_g3WithoutSSC   = 1
set OOB_SpeedNeg_Phy_g3WithSSC      = 0
set OOB_SpeedNeg_Phy_g4WithoutSSC   = 1
                           ithSSC   = 0
                        ithoutSSC   = 1
                           ithSSC   = 0
                              ity   = 1
```

```
Procedure Send_TTIU_RxWindow_IdentifyFrame
{ # Sending Tx training, Rx training and Identify frame (24G, 12G)
    Speed_Neg_RCDT
    Idle (1)            # necessary delay for correct operation after rcdt command
    OUTPUT_ON
    if(@Is12G) then
    {
        Set Speed           = LINK_SPEED
    }
    else
    {
        Set Speed           = LINK_SPEED
    }

    SendTTIU   (0x00006000, 0x3A,0x1000
    waitforttiu(0x00000000,0x00004000)
    SendTTIU   (0x00000000, 0x3A,0x10)

    CLEAR_TIMER_A
    CLEAR_TIMER_D
    Reset_Training_ERROR_COUNT
```

```
else
{
    CLEAR_TIMER_A
    @Training_ERROR_COUNT = Training_ERROR COUNT
    if( @Training_ERROR_COUNT > 0) then
    {
        @Training_ERROR_COUNT = 0
        SendTTIU( 0x00080000, 0x3a, 0x10)
        @End_time              = 0x00000000
```

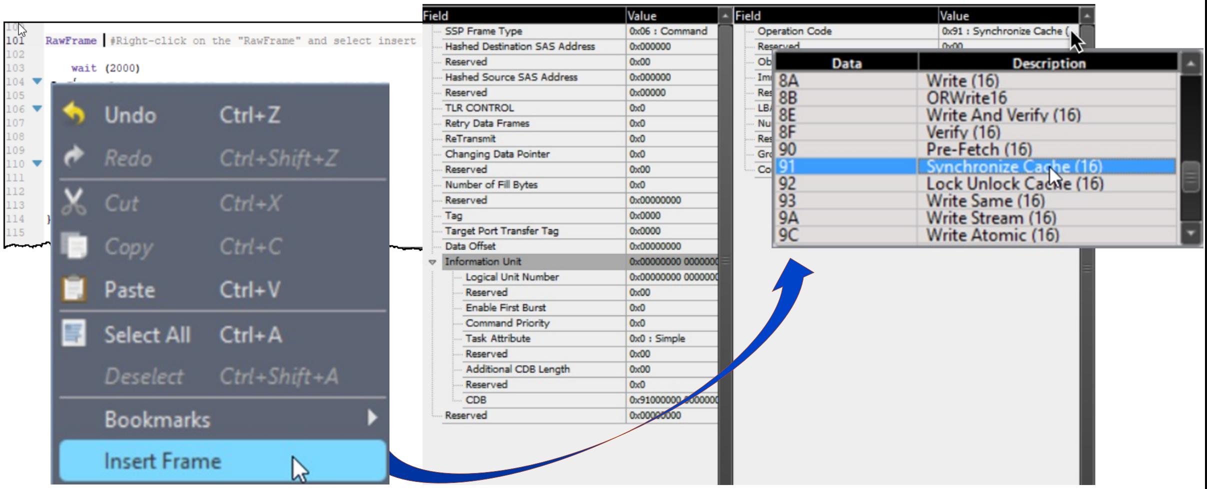
20 | ©SNIA. All Rights Reserved.

The SAS protocol begins with an **Out-of-Band (OOB) handshake**, a low-speed signaling process used to establish a physical connection between two PHYs before any high-speed data is exchanged. This sequence includes key primitives like **COMINIT**, sent by the initiator to signal link initialization, and **COMWAKE**, sent by the target to confirm readiness. The **COMSAS** primitive identifies the device as SAS-capable (as opposed to SATA), while **COMRESET** can be used to clear errors or reinitialize the link. These primitives are essential for ensuring reliable link detection, device compatibility, and proper transition into the next phase of link setup.

Following OOB, the link enters the **Speed Negotiation Window**, where both devices determine the highest mutually supported data rate. During this phase, **Transmitter Training Initialization Units (TTIUs)** are exchanged to optimize signal quality and equalization settings. Having an exerciser with **complete control over OOB handshaking, SNW parameters, and TTIU frames** is critical for testing and validation. It allows engineers to simulate edge cases, inject faults, and verify how devices respond to non-standard or stress conditions. This level of control is especially important for **conformance testing**, which ensures that devices behave according to the SAS specification as defined by the **T10 Technical Committee** under **INCITS**. While T10 defines the standards, it does not enforce compliance through certification programs—so vendors rely on conformance testing to validate interoperability and adherence to the spec.

Use “RawFrame” to Access Fully Decoded Field View

- Type “RawFrame” > Right-click> to access menu of Decoded SSP fields:



```
RawFrame |#Right-click on the "RawFrame" and select insert

    wait (2000)
```

Field	Value
SSP Frame Type	0x06 : Command
Hashed Destination SAS Address	0x000000
Reserved	0x00
Hashed Source SAS Address	0x000000
Reserved	0x00000
TLR CONTROL	0x0
Retry Data Frames	0x0
ReTransmit	0x0
Changing Data Pointer	0x0
Reserved	0x00
Number of Fill Bytes	0x0
Reserved	0x00000000
Tag	0x0000
Target Port Transfer Tag	0x0000
Data Offset	0x00000000
Information Unit	0x00000000 0000000
Logical Unit Number	0x00000000 0000000
Reserved	0x00
Enable First Burst	0x0
Command Priority	0x0
Task Attribute	0x0 : Simple
Reserved	0x00
Additional CDB Length	0x00
Reserved	0x0
CDB	0x91000000 0000000
Reserved	0x00000000

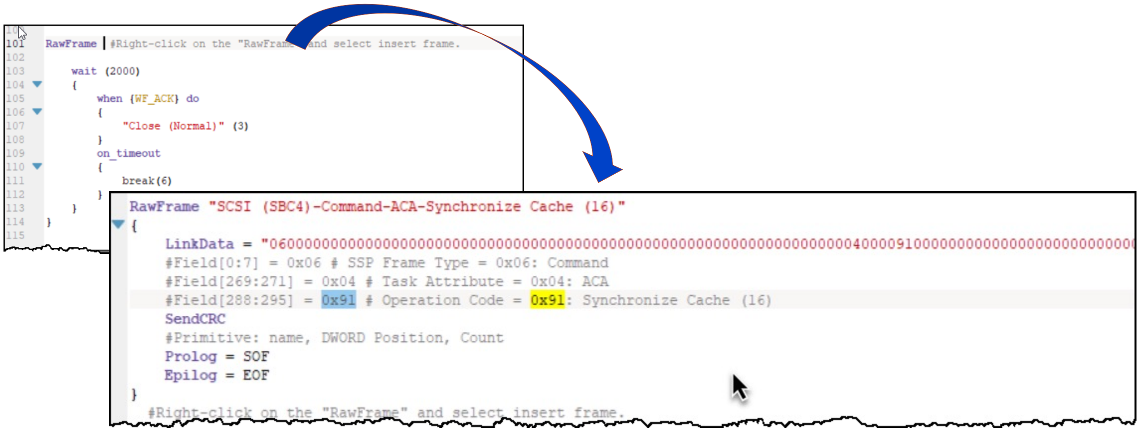
Field	Value
Operation Code	0x91 : Synchronize Cache (
Reserved	0x00
Ob	
Im	
Res	
LBA	
Nu	
Res	
Gro	
Co	

Data	Description
8A	Write (16)
8B	ORWrite16
8E	Write And Verify (16)
8F	Verify (16)
90	Pre-Fetch (16)
91	Synchronize Cache (16)
92	Lock Unlock Cache (16)
93	Write Same (16)
9A	Write Stream (16)
9C	Write Atomic (16)

RawFrame provides a fully decoded view of serial SCSI protocol fields. Users can right-click to access detailed protocol information, making it easier to debug and validate frame-level behavior.

Use “RawFrame” to Access Fully Decoded Field View

- Type “RawFrame” > Right-click> to access menu of Decoded SSP fields:



```
RawFrame |#Right-click on the "RawFrame" and select insert frame.

    wait (2000)
    {
        when {WF_ACK} do
        {
            "Close (Normal)" (3)
        }
        on_timeout
        {
            break(6)
        }
    }
}
```

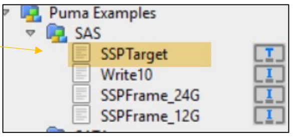
```
RawFrame "SCSI (SBC4)-Command-ACA-Synchronize Cache (16)"
{
    LinkData = "0600000000000000000000000000000000000000000000000000000000000000000400009100000000000000000000000000000
    #Field[0:7] = 0x06 # SSP Frame Type = 0x06: Command
    #Field[269:271] = 0x04 # Task Attribute = 0x04: ACA
    #Field[288:295] = 0x91 # Operation Code = 0x91: Synchronize Cache (16)
    SendCRC
    #Primitive: name, DWORD Position, Count
    Prolog = SOF
    Epilog = EOF
}
  #Right-click on the "RawFrame" and select insert frame.
```

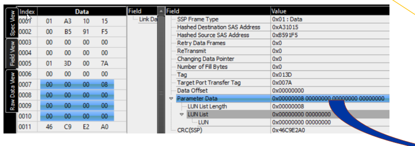
Adds the full frame “LinkData” including comments

In this example, we’re demonstrating how the combination of an analyzer and its integrated exerciser capabilities can be leveraged for **capture and replay** testing. By replaying specific link conditions—either exactly as captured or with targeted modifications—engineers can simulate a wide range of traffic scenarios. This enables **stress testing** of devices under controlled conditions, helping to uncover edge-case behaviors, validate error recovery mechanisms, and assess protocol robustness. The ability to precisely control replayed sequences, including OOB handshakes, SNW parameters, and TTIU frames, is essential for thorough **conformance testing** and debugging interoperability across different SAS implementations.

Overview: Creating a Target Emulation script

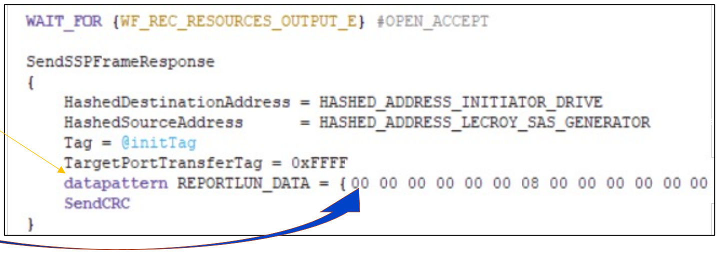
- Capture a Trace between real SAS 12G Initiator/Target
- Start with sample script "SSPTarget";
  Modify sample using responses from actual trace:
  - Change the "SASAddress" & "Hashed-Address" fields
  - Insert "WAIT_FOR" *events (Open_Accept, Xfer_RDY, etc..)*
  - Create Data Response frames by pasting *data-pattern* payload from the actual trace packet:





Index	Data			
0001	01	A3	10	15
0002	00	B5	91	F5
0003	00	00	00	00
0004	00	00	00	00
0005	01	3D	00	7A
0006	00	00	00	00
0007	00	00	00	08
0008	00	00	00	00
0009	00	00	00	00
0010	00	00	00	00
0011	46	C9	E2	A0

Field	Value
SSP Frame Type	0x01 : Data
Hashed Destination SAS Address	0xA31015
Hashed Source SAS Address	0xB591F5
Retry Data Frames	0x0
ReTransmit	0x0
Changing Data Pointer	0x0
Number of Fill Bytes	0x0
Tag	0x013D
Target Port Transfer Tag	0x007A
Data Offset	0x00000000
Parameter Data	0x00000008 00000000 00000000 00000000
LUN List Length	0x00000008
LUN List	0x00000000 00000000
LUN	0x00000000 00000000
CRC(SSP)	0x46C9E2A0

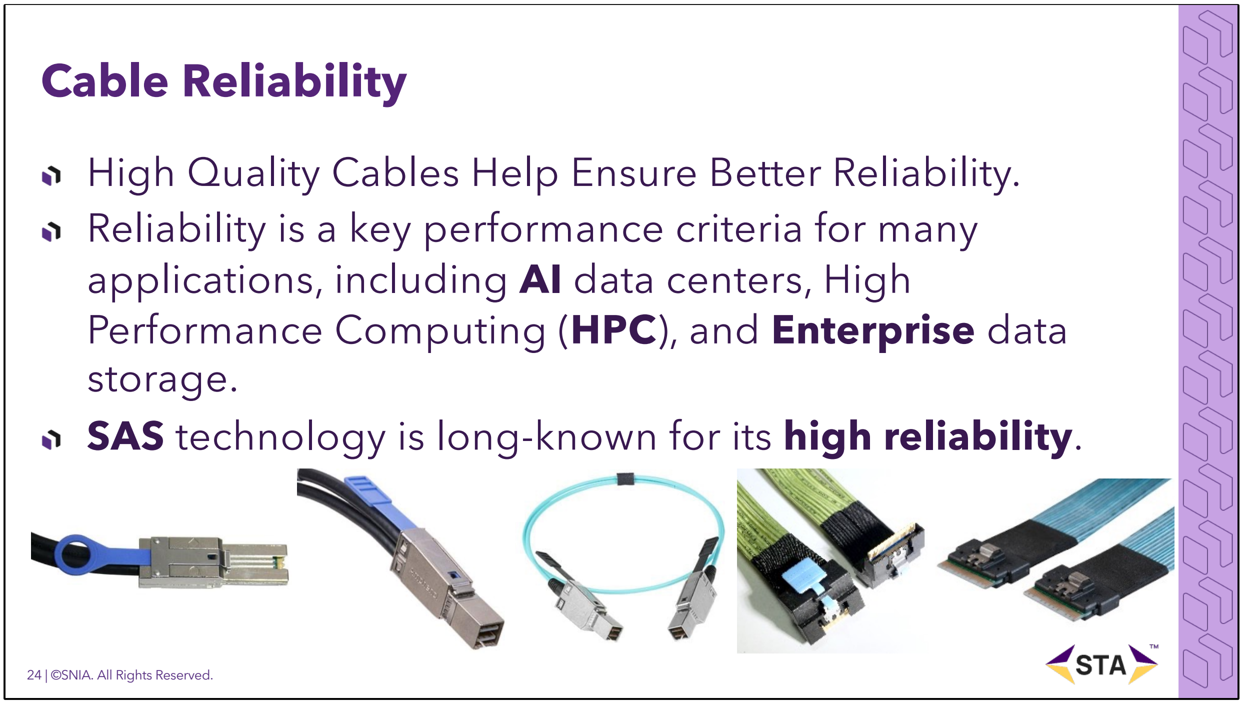


```
WAIT_FOR {WF_REC_RESOURCES_OUTPUT_E} #OPEN_ACCEPT

SendSSPFrameResponse
{
    HashedDestinationAddress = HASHED_ADDRESS_INITIATOR_DRIVE
    HashedSourceAddress      = HASHED_ADDRESS_LECROY_SAS_GENERATOR
    Tag = @initTag
    TargetPortTransferTag = 0xFFFF
    datapattern REPORTLUN_DATA = { 00 00 00 00 00 00 08 00 00 00 00 00 00
    SendCRC
}
```

Target emulation scripts can be built from real traces. By modifying sample scripts and inserting actual data patterns, users can simulate realistic device responses for thorough testing.

I hope this video showed how protocol jamming, exercising and analysis can uncover hidden issues in SAS devices—whether it's corrupting, dropping, or modifying packets, or simulating complex link-up scenarios. With automation, state control, and real trace emulation, you can test smarter, faster, and with more confidence. These tools help you catch problems early, validate recovery behavior, and ensure your devices and environment are ready for real-world deployment.

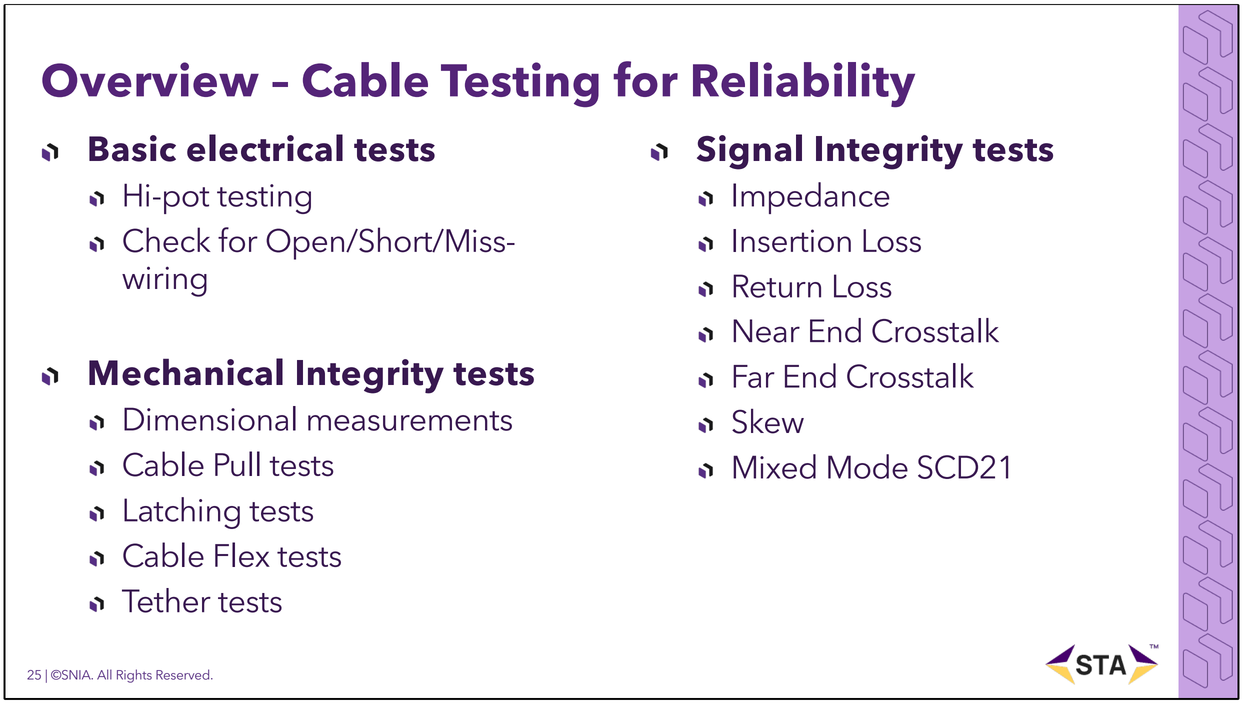


Cable Reliability

- High Quality Cables Help Ensure Better Reliability.
- Reliability is a key performance criteria for many applications, including **AI** data centers, High Performance Computing (**HPC**), and **Enterprise** data storage.
- **SAS** technology is long-known for its **high reliability**.

Hello and thank you for the introduction. My name is Paul Coddington and today I will be talking about cable reliability.

So, reliability is a key performance criteria for many applications, including AI, high performance computing, and enterprise data storage.
SAS technology, including SAS internal cables and SAS external cables, have been long known for their high reliability.



Overview – Cable Testing for Reliability

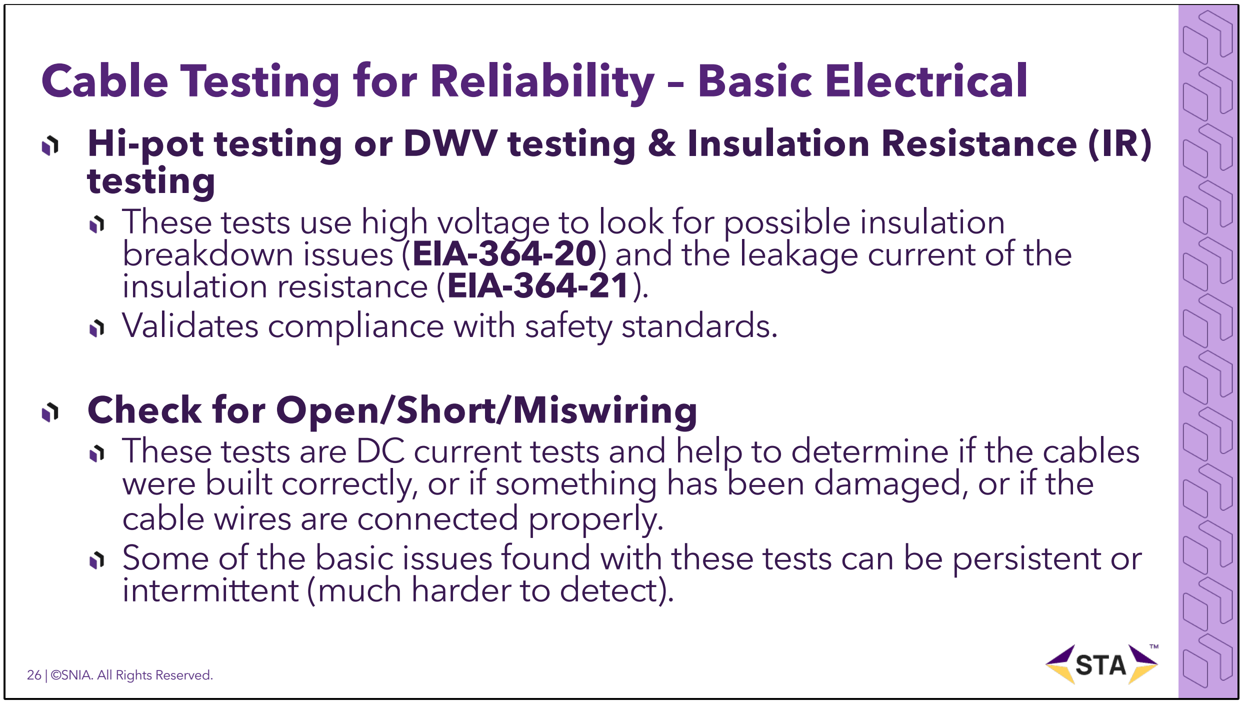
- **Basic electrical tests**
  - Hi-pot testing
  - Check for Open/Short/Miss-wiring
- **Mechanical Integrity tests**
  - Dimensional measurements
  - Cable Pull tests
  - Latching tests
  - Cable Flex tests
  - Tether tests
- **Signal Integrity tests**
  - Impedance
  - Insertion Loss
  - Return Loss
  - Near End Crosstalk
  - Far End Crosstalk
  - Skew
  - Mixed Mode SCD21

So, we're going to talk about reliability testing.

We'll start off with **basic electrical tests**. We'll go through a couple of those.

Then, we’ll go through a few **mechanical integrity tests**.

Finally, we will go through several **signal integrity tests**. All of these will help with making sure that you have reliable cables.



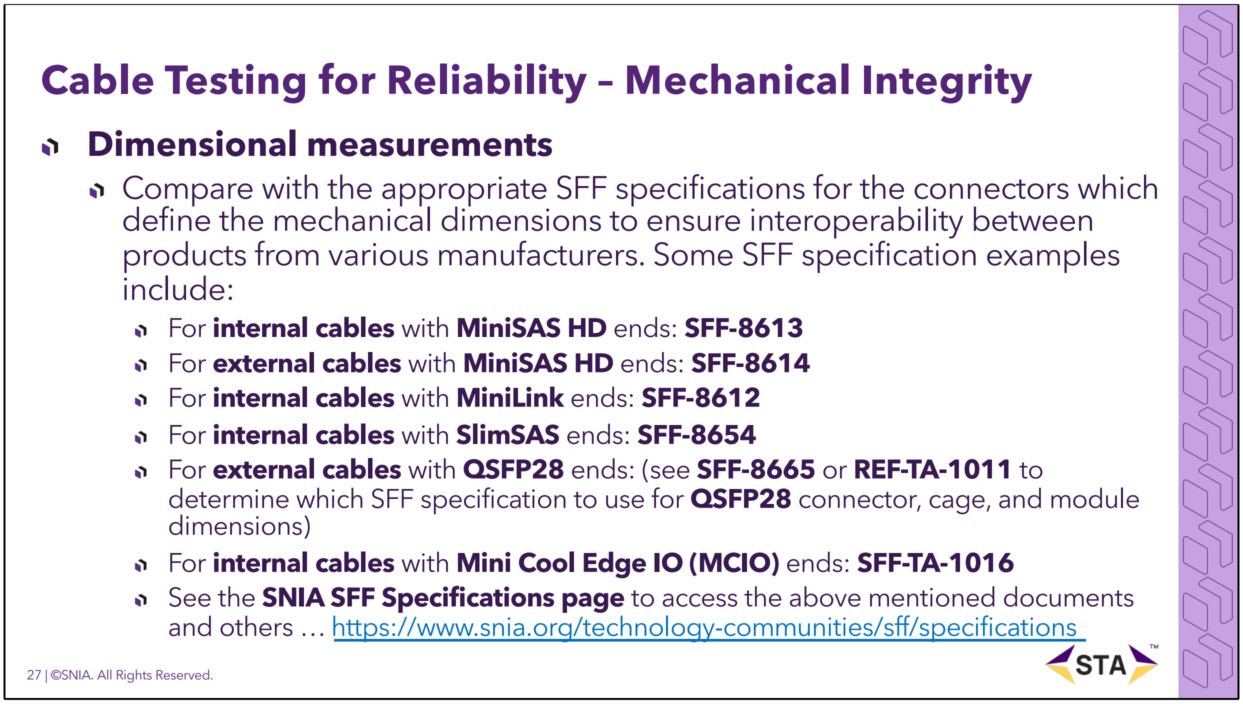
Cable Testing for Reliability – Basic Electrical

- **Hi-pot testing or DWV testing & Insulation Resistance (IR) testing**
  - These tests use high voltage to look for possible insulation breakdown issues (**EIA-364-20**) and the leakage current of the insulation resistance (**EIA-364-21**).
  - Validates compliance with safety standards.
- **Check for Open/Short/Miswiring**
  - These tests are DC current tests and help to determine if the cables were built correctly, or if something has been damaged, or if the cable wires are connected properly.
  - Some of the basic issues found with these tests can be persistent or intermittent (much harder to detect).

So, we'll start off with the **basic electrical tests**.

The Hi-pot or DWV testing and the IR testing are very common tests for validating compliance with safety standards.

In addition to those, you can check for opens, short circuits, and miswiring. **Open/Short/Miswiring tests** are simple DC current tests that can determine if a cable was built properly, the cable wires were connected in the right places, and checks for certain manufacturing defects or possible damage to the cables. I will point out that **Intermittent problems** can be much, much harder to find because they might only present themselves in certain situations, such as when the cable is pulled, moved around, or the connector is wiggled in the port it is plugged into. Those types of intermittent problems can be very difficult to find.



Cable Testing for Reliability – Mechanical Integrity

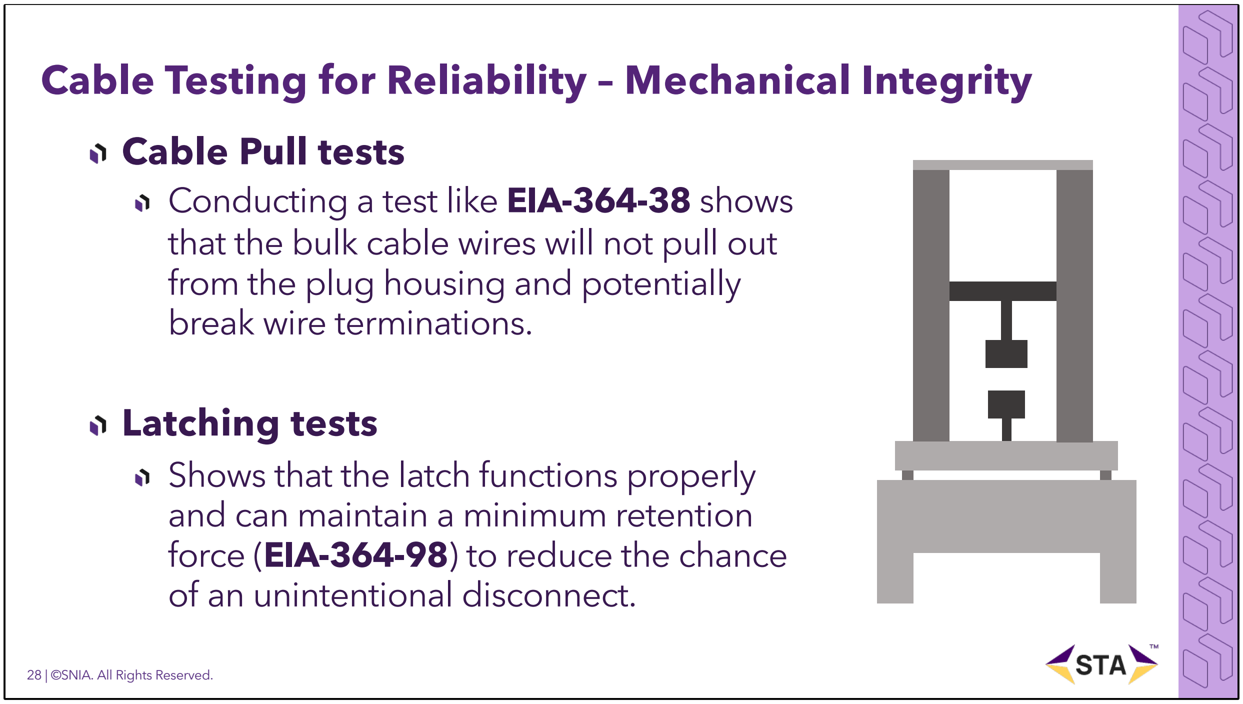
- **Dimensional measurements**
  - Compare with the appropriate SFF specifications for the connectors which define the mechanical dimensions to ensure interoperability between products from various manufacturers. Some SFF specification examples include:
    - For **internal cables** with **MiniSAS HD** ends: **SFF-8613**
    - For **external cables** with **MiniSAS HD** ends: **SFF-8614**
    - For **internal cables** with **MiniLink** ends: **SFF-8612**
    - For **internal cables** with **SlimSAS** ends: **SFF-8654**
    - For **external cables** with **QSFP28** ends: (see **SFF-8665** or **REF-TA-1011** to determine which SFF specification to use for **QSFP28** connector, cage, and module dimensions)
    - For **internal cables** with **Mini Cool Edge IO (MCIO)** ends: **SFF-TA-1016**
    - See the **SNIA SFF Specifications page** to access the above mentioned documents and others … https://www.snia.org/technology-communities/sff/specifications

Now, we will move on to some **mechanical integrity tests**.

First, you'll want to check with the **dimensional measurements**. This involves verifying dimensional measurements in comparison to the documented specifications that define the connectors that make up your cables.

Some **example SFF specifications** are listed here for various common types of **SAS internal** or **SAS external** cables.

If you need to locate one of these SFF specifications listed here or any other SFF specifications, you can find them on the **SNIA SFF Specifications page**. The link is shown here at the bottom of this slide.

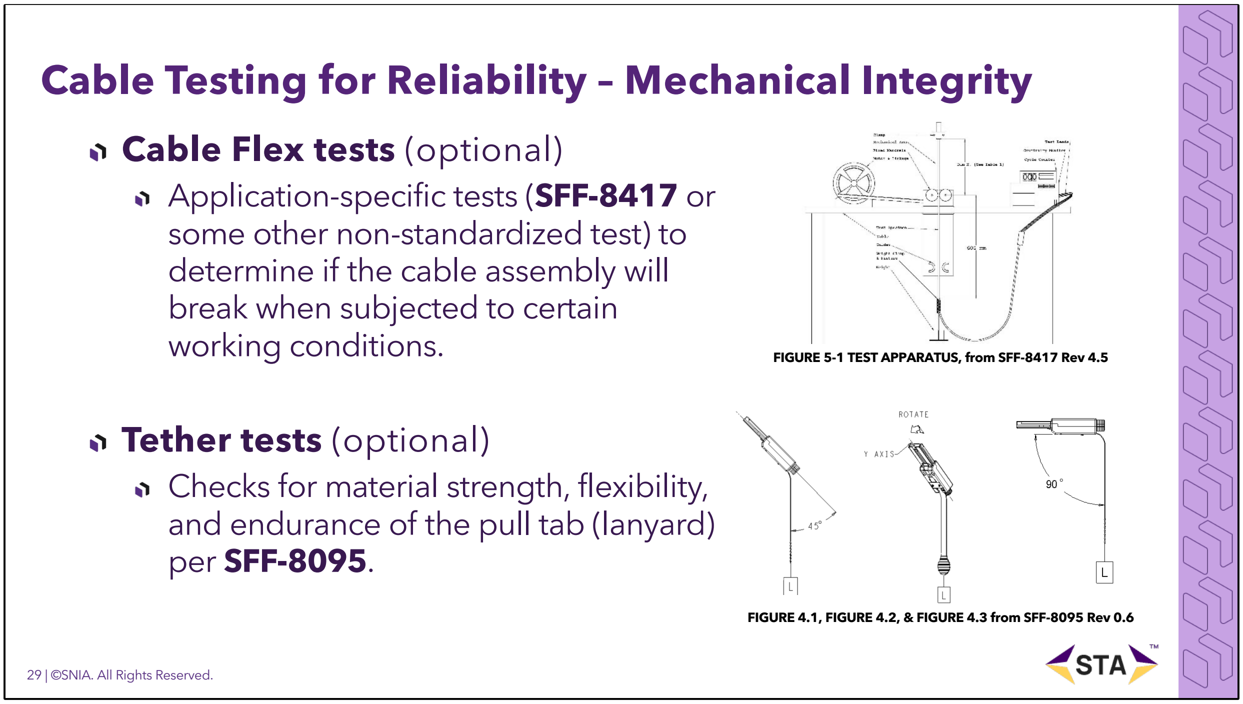


Cable Testing for Reliability – Mechanical Integrity

- **Cable Pull tests**
  - Conducting a test like **EIA-364-38** shows that the bulk cable wires will not pull out from the plug housing and potentially break wire terminations.
- **Latching tests**
  - Shows that the latch functions properly and can maintain a minimum retention force (**EIA-364-98**) to reduce the chance of an unintentional disconnect.

Continuing with the **Mechanical Integrity tests**, we'll talk about **Cable Pull tests**. These tests will check to make sure that the bulk cable wires will not pull out of the cable plug housings, causing breakage of wire terminations or the important connections of the cable shielding.

You can also do **Latching tests**. this will ensure that the latches are functioning properly and that they can maintain the required minimum retention forces. This will help to ensure that your cables don't unintentionally or accidentally get unplugged when the bulk wire of the cable happens to get pulled or some sort of force gets applied to the cable or connector by some means. Basically, the latching makes sure that your stays connected and maintains a good connection.



Cable Testing for Reliability – Mechanical Integrity

- **Cable Flex tests** (optional)
  - Application-specific tests (**SFF-8417** or some other non-standardized test) to determine if the cable assembly will break when subjected to certain working conditions.

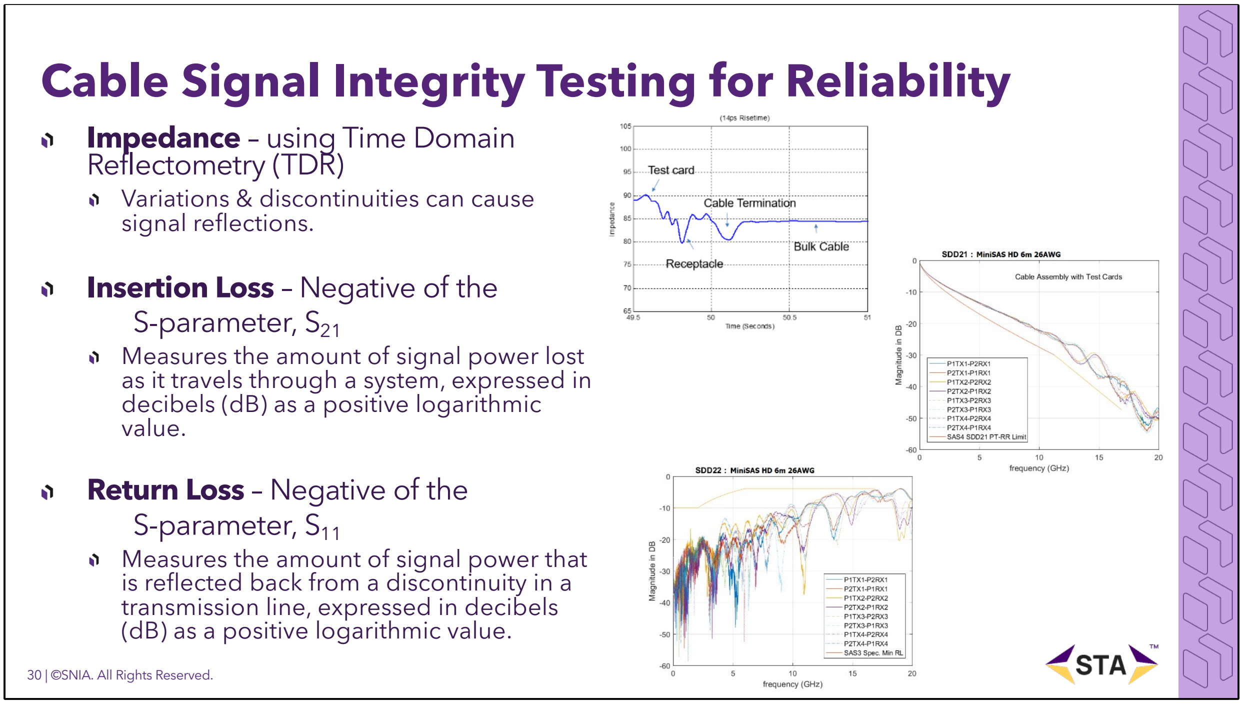
FIGURE 5-1 TEST APPARATUS, from SFF-8417 Rev 4.5

- **Tether tests** (optional)
  - Checks for material strength, flexibility, and endurance of the pull tab (lanyard) per **SFF-8095**.

FIGURE 4.1, FIGURE 4.2, & FIGURE 4.3 from SFF-8095 Rev 0.6

Moving on with some further **mechanical integrity tests**, you have these a variety of **cable flex tests** that can be done. These tests, if required, are usually **application-specific** and may be **customer-specific** too. A lot of times, the Flex tests are **optional** and are not needed for a cable that is intended to be plugged in once and hardly ever moved again, which can happen in a lot of instances in data centers. The SFF-8417 Cable Flex test shown in the figure to the right is sometimes referred to the **tic-toc test** due the pendulum-like back & forth motion that it does … much like a metronome.

In addition to the flex tests, you may want to consider doing some **Tether tests**. Tether tests can check the toughness of the pull tab or lanyard which may be a part of the cable plug. Plastic pull tab or lanyard parts can fail if the material is too brittle due to some poorly controlled molding process during manufacturing. Premature aging of some plastic materials can also sometimes result if the plastic compound pellets were not properly dried prior to the molding process, especially with certain plastic materials, such as nylon, that tend to absorb moisture from the ambient air around them over time.



Cable Signal Integrity Testing for Reliability

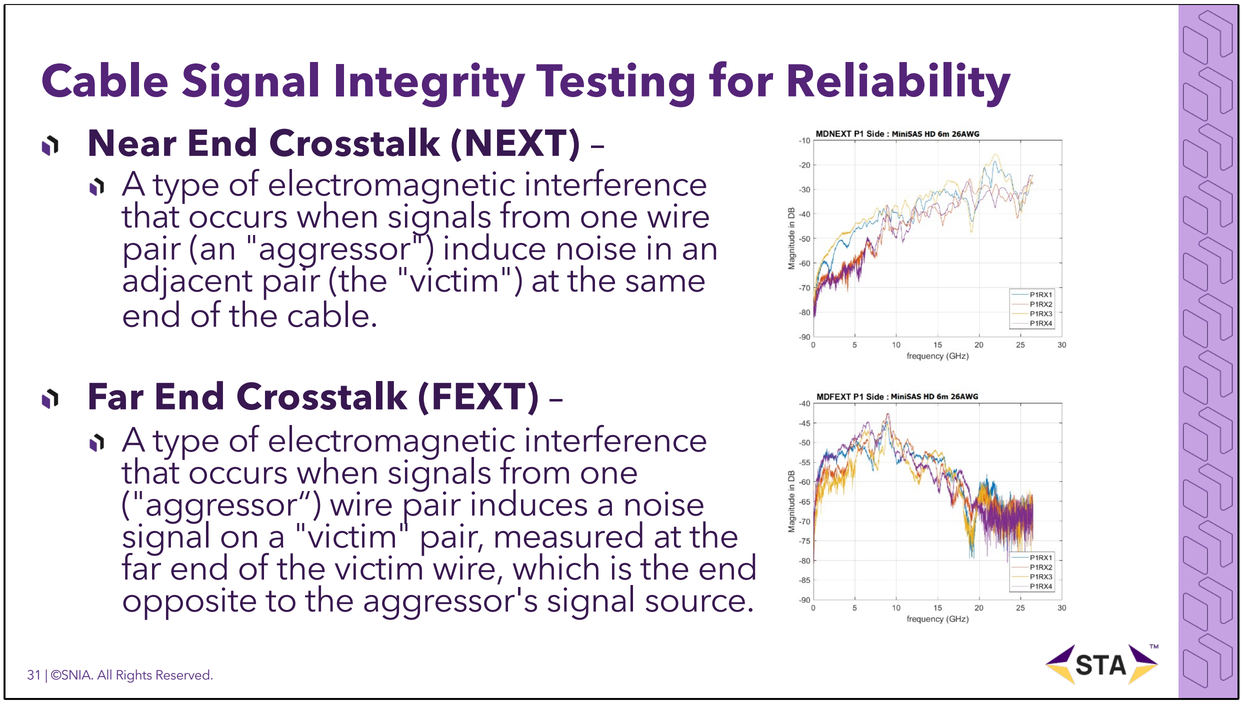
- **Impedance** – using Time Domain Reflectometry (TDR)
  - Variations & discontinuities can cause signal reflections.
- **Insertion Loss** – Negative of the S-parameter, S_{21}
  - Measures the amount of signal power lost as it travels through a system, expressed in decibels (dB) as a positive logarithmic value.
- **Return Loss** – Negative of the S-parameter, S_{11}
  - Measures the amount of signal power that is reflected back from a discontinuity in a transmission line, expressed in decibels (dB) as a positive logarithmic value.

30 | ©SNIA. All Rights Reserved.

Next, I will talk about **signal integrity testing** for reliability. There's actually a whole series of SI tests for cables. I'll start off with the **Impedance test**, a Time Domain measurement. The Impedance test measures variations of the impedance along the signal path within a cable. A TDR is what's used to make the measurements. There is a sample TDR plot shown to the right, with the various components of the cable assembly indicated on it shown by the effects they have caused.

Many of the **SI tests** can be done all at once and saved in **S-parameter files**, generated using a Vector Network Analyzer (VNA), and show how a cable's characteristics change with frequency. The standard format for these files is a Touchstone .sNp, file, where N is the number of ports, such as .s2p or .s4p, and so on. One of these S parameter measurements is **Insertion Loss**, which measures the amount of signal power lost as it travels through a system expressed in decibels. There's a sample graph to the right that that shows what an insertion loss plot might look like. Insertion Loss is important because it basically limits the effective reach or length of a cable. The longer the cable, the more insertion loss there is. This is why longer cables tend to use larger wire gauge sizes. Larger wire gauges have less insertion loss, allowing the cables to work at longer distances.

Then, you have **Return Loss**. Return loss measures the amount of signal that is reflected back from any discontinuity in the transmission line. It's also expressed in decibels. I provided a sample graph to the right that shows what a return loss plot might look like. Discontinuities and impedance mismatches can cause some harmful reflections, especially at higher data rates.

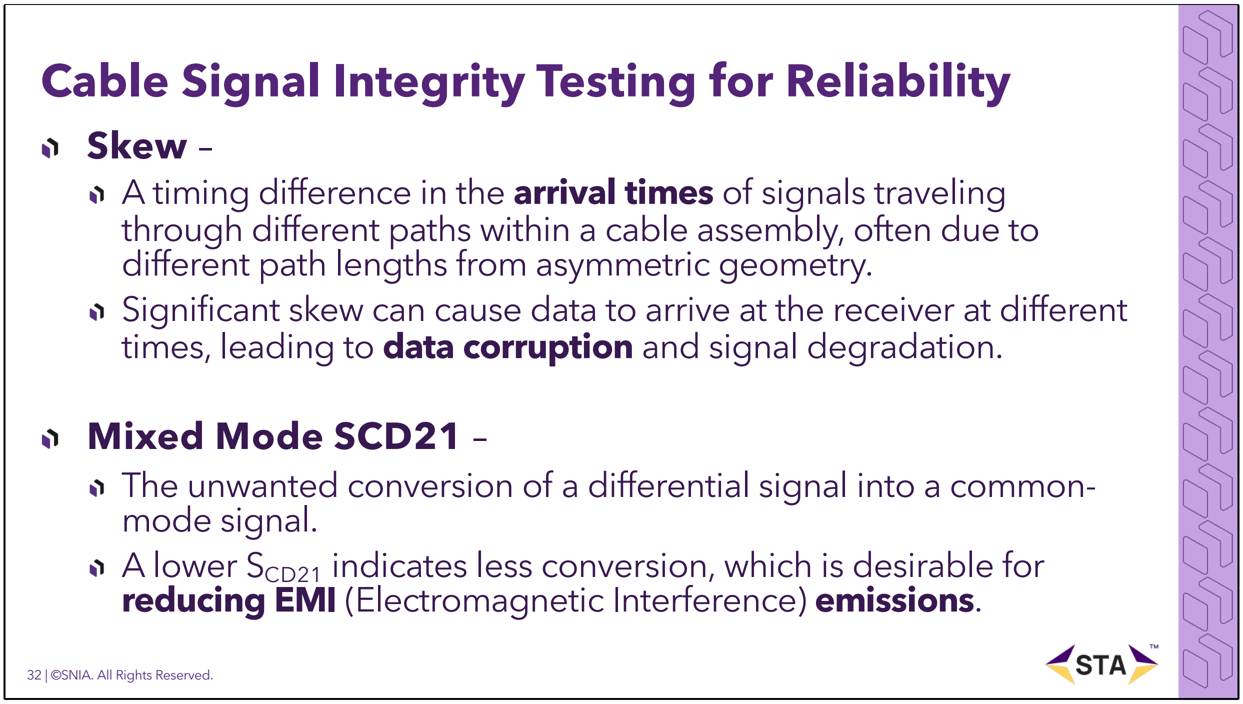


Cable Signal Integrity Testing for Reliability

- **Near End Crosstalk (NEXT)** –
  - A type of electromagnetic interference that occurs when signals from one wire pair (an "aggressor") induce noise in an adjacent pair (the "victim") at the same end of the cable.
- **Far End Crosstalk (FEXT)** –
  - A type of electromagnetic interference that occurs when signals from one ("aggressor“) wire pair induces a noise signal on a "victim" pair, measured at the far end of the victim wire, which is the end opposite to the aggressor's signal source.

Alright, next I'm going to talk about some additional **S-parameter measurements**. These S-parameters measure the amount of noise introduced into a signal by the various types of crosstalk. The first one we're going to look at is **near end crosstalk, or NEXT**, which is a type of electromagnetic interference that occurs when signal from one wire pair induces noise onto an adjacent pair at the same end of the cable. You can see a sample of what a near end crosstalk plot might look like to the right.

In addition, you can also look at **far end crosstalk, or FEXT**, which is similar, but in this case the electromagnetic interference occurs when the signals from one wire pair induces a noise on a on a victim pair measured at the far end of the victim wire, which is the end opposite of the aggressor signal source. The graph to the right shows what a far end crosstalk plot might look like. In cases where there is a lot of crosstalk, the desired signals can get degraded and reduces the signal to noise ratio. It becomes increasingly difficult for the receiver to distinguish the signal from the noise, and this can cause data errors, increasing the bit error rate or bit error ratio to a point beyond what is correctable, leading to data loss.



Cable Signal Integrity Testing for Reliability

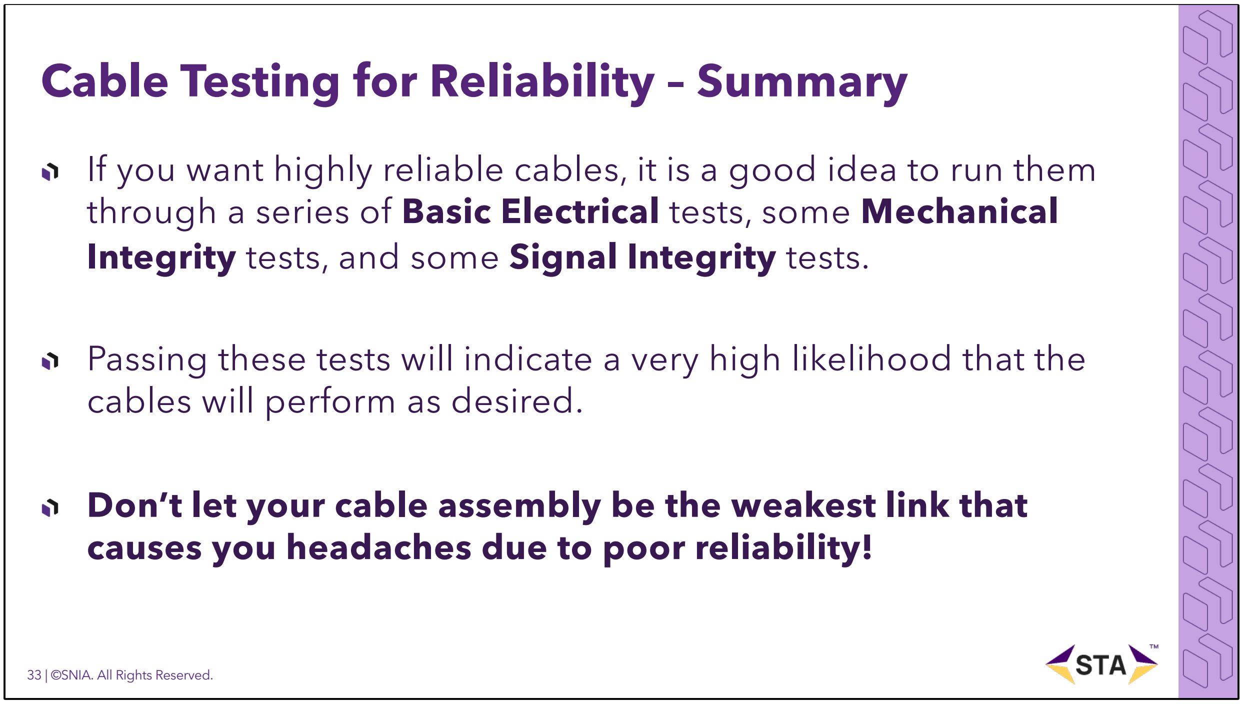
- **Skew** –
  - A timing difference in the **arrival times** of signals traveling through different paths within a cable assembly, often due to different path lengths from asymmetric geometry.
  - Significant skew can cause data to arrive at the receiver at different times, leading to **data corruption** and signal degradation.

- **Mixed Mode SCD21** –
  - The unwanted conversion of a differential signal into a common-mode signal.
  - A lower S_{CD21} indicates less conversion, which is desirable for **reducing EMI** (Electromagnetic Interference) **emissions**.

32 | ©SNIA. All Rights Reserved.

Another signal integrity test that can be done is **skew** measurement. This is another **Time-domain test** that involves very precise measurements. Skew is a timing difference that in the arrival times of signals traveling through different paths within a cable assembly, often due to different path lengths caused by asymmetric geometry. As data rates increase, so does the importance of limiting skew. Too much skew can cause data to arrive at the receiver at different times, leading to data corruption and signal degradation.

Finally, we have **mixed mode SCD21** which is another one of the **S-parameter measurements**. SCD21 is the unwanted conversion of a differential signal into a common-mode signal. It indicates mode conversion which can lead to signal distortions. Basically, a lower SCD21 indicates less mode conversion which is desirable for reducing EMI emissions and noise which degrades the desired signal. There are also other mixed-mode S-parameters, like **SDC21** which measures the common-mode input to differential output. It's kind of like the reverse, right? (which indicates a level of susceptibility). A high SDC21 indicates a significant conversion of common-mode noise to the desired differential signal. A low SDC21 signifies very good isolation. This wraps up a number of the common SI tests that are typically done on cables to ensure high reliability for the user in their application.



Cable Testing for Reliability – Summary

- If you want highly reliable cables, it is a good idea to run them through a series of **Basic Electrical** tests, some **Mechanical Integrity** tests, and some **Signal Integrity** tests.
- Passing these tests will indicate a very high likelihood that the cables will perform as desired.
- **Don’t let your cable assembly be the weakest link that causes you headaches due to poor reliability!**

So, in conclusion, if you want highly reliable cables, which we all do, right? It's a very good idea to run them through a series of **basic electrical tests**, some **mechanical integrity tests,** and some **signal integrity tests**.

There is a high likelihood that the cables will perform as desired if they pass these series of tests.

Avoid letting your cables be the weakest link that causes you headaches due to poor reliability. And with that, I am finished with my presentation and thank you for your time.

Follow STA

https://x.com/SNIA

YouTube

Serial Attached SCSI Playlist on SNIAVideo
https://www.youtube.com/@SNIAVideo

LinkedIn

https://www.linkedin.com/company/snia/