2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Efficient Debug Strategies for PCIe Gen6 Verification Using Verification IP (VIP)

Satish Kumar Padhi
(Cadence Design Systems)

cadence®

accellera SYSTEMS INITIATIVE

Abstract

PCI Express Gen6 (PCIe® 6.0/6.1) introduces significant complexity with new features such as FLIT mode, 64GT/s PAM4 signaling, and enhanced protocols, making verification and debug extremely challenging. This paper presents an efficient debug methodology for complex PCIe® Gen6 designs using Cadence's PCIe Gen6 Verification IP (VIP). We focus on two key aspects: leveraging the VIP's built-in protocol checking capabilities and utilizing trace file logs to streamline debug and analysis. The proposed approach embeds Cadence VIP as a smart agent in the testbench to automatically flag specification violations across transaction, data link, and physical layers. When issues arise, a user can utilize detailed trace files generated by the VIP to quickly pinpoint protocol errors and timing mismatches. We will demonstrate how this combination of proactive protocol checks and trace-driven debugging accelerates root-cause analysis in a PCIe Gen6 environment. Our strategy enables faster identification of FLIT format errors and credit exhaustion, reducing debug time by a significant margin. The results show improved verification efficiency, thorough coverage of PCIe 6.0 features, and insights that can be applied to other complex interface protocols. Overall, the methodology benefits teams facing advanced PCIe verification by ensuring robust, compliance-checked designs with a streamlined debug cycle

Agenda

- **Introduction**
  - Brief overview of PCIe® Gen6
  - FLIT Mode in TL and DL
- **PCIe DUT Verification with Passive VIP Monitor**
  - Architecture and setup
  - Benefits of Smart Passive Monitoring Agent
- **Efficient Error Messages for Debugging**
  - Designing meaningful error messages
  - Real-world examples and impact
- **Robust Callbacks and Error Injection Approach**
  - Callback mechanisms
  - Error injection strategies for corner-case validation
- **Extensive Debug Info Support**
  - Extend of debug information
- **Smart and Efficient Debugging with Waveforms**
  - Waveform-based debugging techniques
  - Tools and tips for faster traceability
- **Enhancing Debugging by Tracking Packets**
  - Packet-level tracking methodology
  - Correlation with protocol events
- **Automated Result Debug**
  - Automation scripts and frameworks
  - Reducing manual effort in debug cycles
- **Conclusion**
  - Summary of key takeaways
- **Q&A / Discussion**
  - Open floor for questions and feedback

PCIe Gen6 Update (Basic)

- All prior major revisions of PCIe® doubled its bandwidth by doubling the frequency with which bits were sent. For PCIe 6.0, instead of trying to double the signaling rate, it doubles the amount of data sent per unit interval with **1b/1b encoding**.
- Fixed-sized **Flits** (256B) were chosen to enable **Forward Error Correction** (FEC) to be added without drastically increasing latency and complexity.
- One or more TLPs can be held within a Flit, and a single TLP may span across multiple Flits (based on max payload size and start location within the Flit)
- The **Ack/Nak protocol** and retry mechanisms are on the Flit level and not the TLP level.
- Flit Mode introduces the idea of dedicated receive buffers and **shared receive buffers**
- When performing the Tx EQ process for the 64GT/s data rate, **TS0** ordered sets are used in Phase 0, Phase 1, and in the upstream direction during Phase 2

FLIT Mode Representation

Table 4-10 Flit Layout in a x16 Link §

Description	Lane 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16 Symbol times — ONE FLIT — TLP Bytes 236 Bytes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	224	225	226	227	228	229	230	231	232	233	234	235	DLP 0	DLP 1	DLP 2	DLP 3
DLP, CRC, ECC	DLP 4	DLP 5	CRC 0	CRC 1	CRC 2	CRC 3	CRC 4	CRC 5	CRC 6	CRC 7	ECC1 [0]	ECC2 [0]	ECC0 [0]	ECC1 [1]	ECC2 [1]	ECC0 [1]

Transaction layer adds TLP (236 Bytes).

Note: When TLP is absent in 236 bytes, it contains all zeros.

In FM, the basic unit of transfer is no longer packets, but is a fixed-sized (256B) Flit.

Data Link Layer adds DLP (6 Bytes).

Physical Layer adds CRC (8 bytes) and ECC (6 bytes)

Non-Flit Mode vs Flit Mode as per DL

Non-Flit Mode

- Data Link Layer adds CRC (4 bytes) and LCRC (2 bytes)
- Ack/Nak DLLPs are transmitted by DL as per Ack/Nak Frequency
- Sequence number size is 12 bytes and is used to track TLPs
- Framing symbols are present in front of each TLP and DLLP

Flit Mode

- No LCRC
- CRC generation/checking has moved to the physical layer with much stronger protection and an 8-byte size. CRC covers TLP and DLP bytes ECRC is still a part of each TLP
- DLLPs can be transmitted as part of Flit only
- Ack/Nak DLLP encodings are not supported in FM. Rather, the information is accommodated in 6 DLP bytes present in Flit Mode. The replay command (as part of DLP bytes) holds the Ack/Nak information, and the sequence number is processed accordingly. The concept of Ack/Nak Protocol is the same
- Apart from the Standard Nak, **Selective Nak** is new and requesting a replay of Single Flit
- Sequence number size is 10 bytes and is used to track Payload Flits only (not tracking TLPs)
- Traditional Flow credits are termed Dedicated Credits
- New Flow credits are introduced as **Shared Credits**
- No framing symbols

Shows NOP Flit shows Zero means No TLP is present, Replay command as ACK with ACK's Sequence number and NEW DLLP Optimized_Update_FC

Snippet from VIP Trace Debug Print:

```
0 H Debug: DL[port_0] : TX DLLP Optimized_Update_FC (vc:0, sharedNPH:72, sharedPH:12, sharedPD:112)
0 H Debug: [port_0] PL_TX_end_packet PlFlit(Nop) Seq={usage : 0, prior : 0, cmd : Ack, num : 42} DllpPayload={Optimized_Update_FC vc:0
sharedNPH:72 sharedPH:12 sharedPD:112 bytes:0x0480c070} (256 bytes) (CRC=0x4234664e38781770 ECC_0=0xf1ac ECC_1=0x6ab1 ECC_2=0xe8fa ) {00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...}
```

Trace File Debug on FLIT Structure

FLIT type

Sequence Number Info

2A H Debug: [port_0] PL_TX_end_packet PlFlit(**Payload**) Seq={usage : 1, prior : 0, cmd : Ack, num : 6} DllpPayload={0x31000000} (256 bytes) (CRC=0xcc6d0f448b96c476 ECC_0=0x710c ECC_1=0xb8ac ECC_2=0x8e25) {60 00 00 01 00 00 00 00 00 00 10 00 00 00 00 00 5A 5A 5A 5A 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 44 06 31 00 00 00 76 C4 96
8B 44 0F 6D CC AC 25 0C B8 8E 71}

TLP

DLLP

CRC

ECC

DL FLIT and PL FLIT Debug Representation

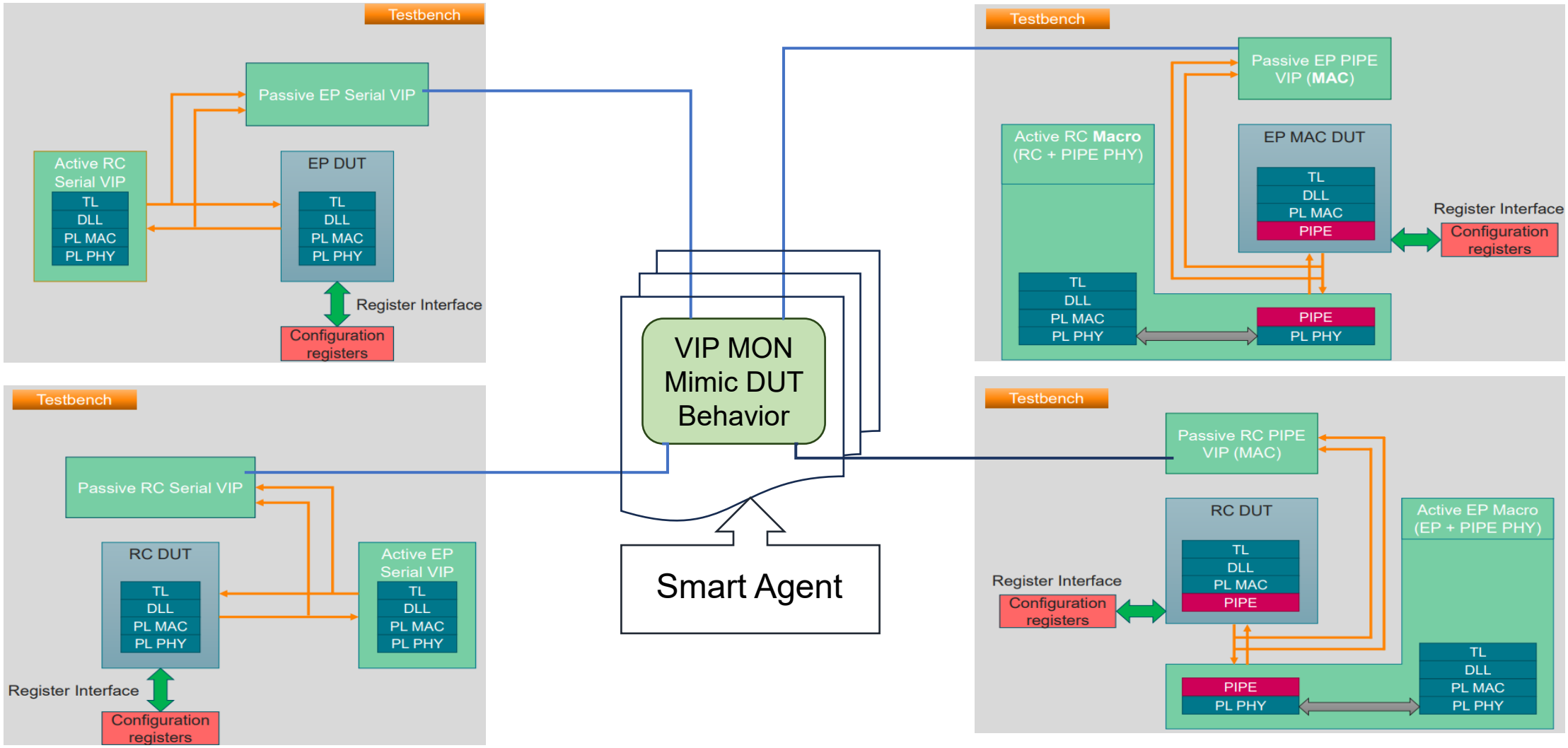
2A H Debug: [port_0] DL_FLIT_TX_queue_enter **DlFlit**{Payload[242]=60 00 00 01 00 00 00 00 00 00 10 00 00 00
00 00 5A 5A 5A 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 **44 06 31 00 00 00**}

P H Debug: [port_0] PL_TX_end_packet **PlFlit**(**Nop**) Seq={usage : 0, prior : 0, cmd : Ack, num : 27}
DllpPayload={0x31000000} (256 bytes) (CRC=0xaf1062222490796b ECC_0=0xef63 ECC_1=0xc30f ECC_2=0x5b9e)
{00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 1B 31 00 00 00 6B 79 90 24 22 62 10 AF 0F 9E 63 C3 5B EF}

IDLE FLIT (Logical Idle)

```
4 H Debug: [port_0] PL_TX_end_packet PlFlit(LogicalIdle) Seq={usage : 0, prior : 0, cmd : Explicit, num : 0}
DllpPayload={0x00000000} (256 bytes) (CRC=0x0000000000000000 ECC_0=0x0000 ECC_1=0x0000
ECC_2=0x0000 ) {00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00}
```

PCIE DUT Verification with VIP MON



Efficient Error Messages for Debugging

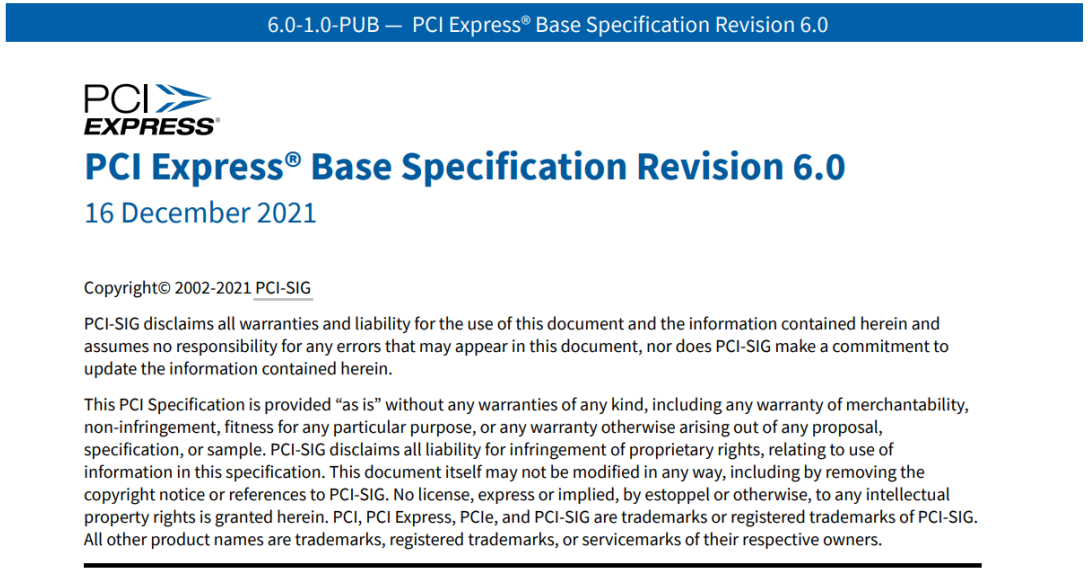
To enable users to identify the root cause of errors unambiguously, PCIe® GEN6 VIP error messages should include the following structured information:

- **Error Context – What Happened?**
  Clearly describe the event or condition that triggered the error
- **Root Cause – Why Did It Happen?**
  Provide a concise explanation of the underlying reason for the error
- **Debug Guidance – How to Proceed?**
  Suggest actionable steps or a recommended approach to debug and resolve the issue
- **Specification Reference**
  Include direct references to the **PCIe® Base Specification**, along with the **version details**, to support a deeper investigation

Continued…

- **Relevant Data Checks**
  Highlight key data points to verify, such as:
  - Configuration settings
  - Register values
  - Pin states
- **Direction Details**
  Specify the data flow direction involved in the error:
  Tx (Transmit) or **Rx (Receive)**
- **Protocol Layer Information**
  Indicate the relevant **protocol layer**, **port**, and **lane** involved, as observed inside the checker
- **Debug Hint**
  Provide a helpful hint or insight that can accelerate the debugging process

PCIe Base Spec Reference on Error:


6.0-1.0-PUB — PCI Express® Base Specification Revision 6.0

PCI EXPRESS

PCI Express® Base Specification Revision 6.0

16 December 2021

- 357464433825 fs –
C H Detected[cfg_0_0] [] PL_LTSSM_STATE [PCISIG]. [port_0] LTSSM state L0 => Recovery.RcvrLock

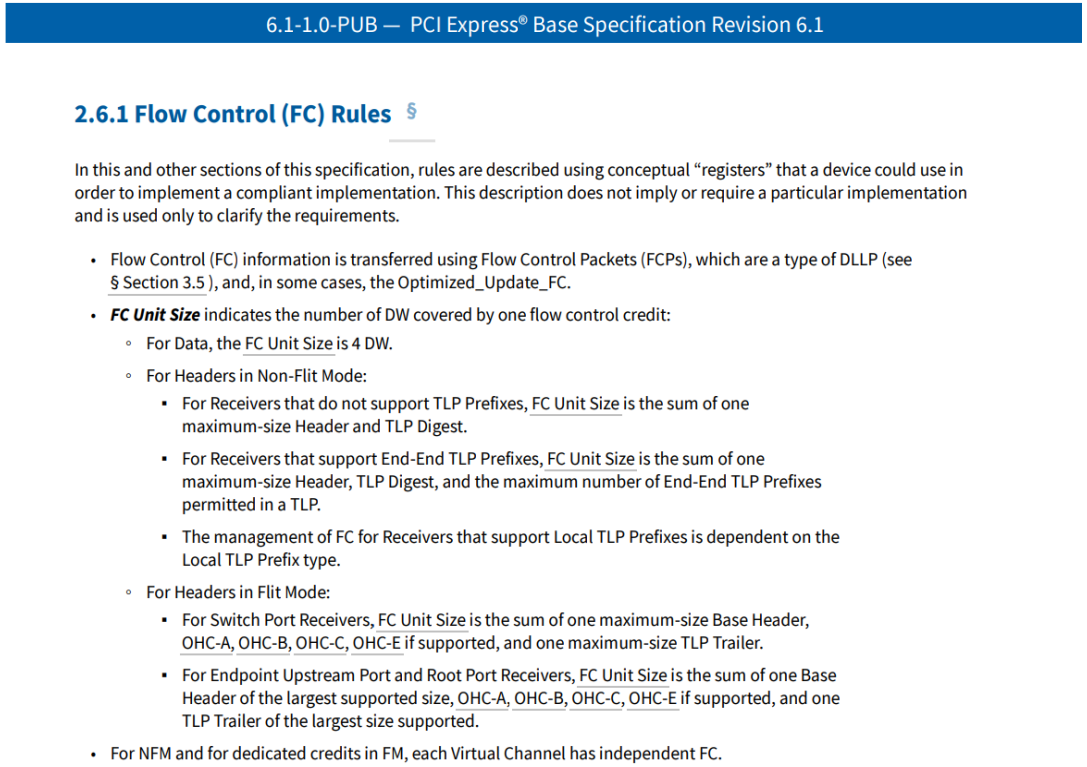
- 357470575654 fs –
C H Error: Detected[cfg_0_0] (TX) [] **PL_FLIT_FEC_ECC_SINGLE_BYTE_ERROR** [PCISIG-PCIE 6.0 Rev 0.9]. [Port_0] Detected Tx single byte error on group ECC_0. Byte position 51, error value 0, corrected value 225.

M *Denali* Error: <top.pipe_mon.phy_mon>@357470575654 fs :: Detected[cfg_0_0] (TX) [] PL_FLIT_FEC_ECC_SINGLE_BYTE_ERROR [**PCISIG-PCIE 6.0 Rev 0.9**]. [Port_0] Detected Tx single byte error on group ECC_0. Byte position 51, error value 0, corrected value 225.

Spec Reference Continued..

- 360647230375 fs –
6 H Error: Detected[cfg_0_0] (TX)
[] **TL_FC_SHARED_CREDIT_OVERFLOW** [PCI SIG-PCIE 6.1 Section 2.6.1]. Port_0: When advertising 128 allowed PH shared credits for VC 0, the sum of the shared credits across all VCs (128) should not be greater than what the current scale factor (0) can represent (127). Limiting allowed credits for the current VC to 124

Denali Error:
<uvm_test_top.m_env.rcep[0].pcie_agent.m_vip_agent>@15197.921 ns :: Detected[cfg_0_0] (TX)
[] TL_FC_SHARED_CREDIT_INIT_CPLD
[PCISIG-PCIE 6.1 Section 2.6.1]. The sum of the shared InitFC value across all VCs (2080) is less than the minimum shared CPLD FC credit requirement (UNLIMITED). The number of enabled VCs: 1



6.1-1.0-PUB — PCI Express® Base Specification Revision 6.1

2.6.1 Flow Control (FC) Rules §

In this and other sections of this specification, rules are described using conceptual "registers" that a device could use in order to implement a compliant implementation. This description does not imply or require a particular implementation and is used only to clarify the requirements.

- Flow Control (FC) information is transferred using Flow Control Packets (FCPs), which are a type of DLLP (see § Section 3.5), and, in some cases, the Optimized_Update_FC.
- ***FC Unit Size*** indicates the number of DW covered by one flow control credit:
  - For Data, the FC Unit Size is 4 DW.
  - For Headers in Non-Flit Mode:
    - For Receivers that do not support TLP Prefixes, FC Unit Size is the sum of one maximum-size Header and TLP Digest.
    - For Receivers that support End-End TLP Prefixes, FC Unit Size is the sum of one maximum-size Header, TLP Digest, and the maximum number of End-End TLP Prefixes permitted in a TLP.
    - The management of FC for Receivers that support Local TLP Prefixes is dependent on the Local TLP Prefix type.
  - For Headers in Flit Mode:
    - For Switch Port Receivers, FC Unit Size is the sum of one maximum-size Base Header, OHC-A, OHC-B, OHC-C, OHC-E if supported, and one maximum-size TLP Trailer.
    - For Endpoint Upstream Port and Root Port Receivers, FC Unit Size is the sum of one Base Header of the largest supported size, OHC-A, OHC-B, OHC-C, OHC-E if supported, and one TLP Trailer of the largest size supported.
- For NFM and for dedicated credits in FM, each Virtual Channel has independent FC.

Spec Reference Continued…

3 H Error: Detected[cfg_0_0] (TX) [] **TL_IDE_INVALID_LINK_STREAM_ID** [PCISIG-PCIe IDE ECN Section 6.99.4]. [Port_0]: Detected non-Selective IDE pkt TLP for TC 2 with StreamID 0 which is associated with TC 0. Discard incoming TLP.

Denali Error: <uvm_test_top.sve.link_env.dev_env0.vip_wrap_env.pcie_ep_env.active_bfm>@207.112 us :: Detected[cfg_0_0] (RX) [] TL_IDE_INVALID_K_BIT_TOGGLING [PCISIG-PCIe IDE ECN Section 6.99.4]. [Port_0]: Detected Bit K toggled to 1 but KeySet 1 are invalid.

6.99.4 IDE TLPs

TLPs secured by IDE are called IDE TLPs. All IDE TLPs must use the IDE prefix (see Figure 7-17), and this prefix must precede all other End-End TLP prefixes.

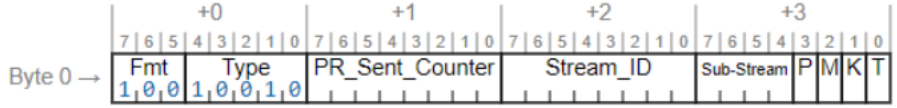


Figure 7-17 IDE TLP Prefix

The IDE Prefix includes:

- M bit – When Set, indicates this TLP includes a MAC
  - When aggregation is not used, the M bit must be Set for all IDE TLPs.
  - Rules for the use of aggregation are given below.
- K bit – Indicates the key set used for this TLP
  - After Transmitting a TLP with the K bit toggled for any Sub-Stream, subsequent TLP Transmissions for different Sub-Streams of the same Stream must also use the new value for the K bit.
  - After receiving a TLP with the K bit toggled, the Receiver must transition to the new key and *IV* set for that TLP and all subsequent TLPs associated with the Sub-Stream, and must mark the old key and *IV* set invalid until reprogrammed.
    - Such transitions must not affect other Sub-Streams of the IDE Stream at the Receiver.
- T bit – When Set, indicates the TLP originated from within a trusted execution environment (see Section 6.99.1)
  - It is permitted for IDE TLPs to originate from both trusted and non-trusted execution environments, and the value of the T bit does not modify the handling of TLPs within IDE per se; the rules for trusted execution environments are not defined in this document.
  - The T bit must be Clear unless the use of the T bit has been explicitly defined by TEE management mechanisms outside the scope of this document.
- P bit – When Set, indicates the TLP includes PCRC.

Efficient Debug Hint

- Error: Detected[cfg_0_0] (RX) [NONFATAL-UnsupportedRequest] TL_ATS_DEVID_EN_UR [PCISIG].Translation Requests received as MWr_64 from Device ID 0x301 are not supported by TA, as ATS is not enabled for the device
- **DEBUG_HINT:** To avoid this error, set the ATS Control Register's Enable Bit in the DUT's PF. Test can do this by sending CfgWr packet as a front-door or by a backdoor DUT configuration on the ATS control register in config space. RC VIP shadow space will get updated with the actual value after it receives completion of the CfgRd packet from device ID 0x301

Robust Callbacks and Error Inject Approach

- **Packet Manipulation and Error Injection**
  - Modify specific fields within a packet to simulate various scenarios
  - Discard packets intentionally to test system resilience
  - Inject errors into packets after generation by the VIP but before transmission to the physical link
- **Data Synchronization and Verification**
  - Synchronize transaction data between the VIP and the DUT to ensure consistency
  - Connect to scoreboards to compare DUT responses against expected results
  - Synchronize with model events without relying on polling mechanisms
- **Enhanced Callback Mechanism**
  - Extend callback capabilities within the PCIe® Link environment to support advanced verification needs
  - Define a comprehensive set of UVM callbacks corresponding to each VIP callback
  - Dynamically activate or deactivate callbacks during simulation
  - Enable or disable callbacks at specific simulation points or stages in the packet flow

Additional Debug Info Support

Setting this in VIP simulation can add significant information to the log that can ease the further debug process.

- CDN_PCIE_PRINT_AFTER_SCRAMBLER: Data after scrambler
- CDN_PCIE_PRINT_LANE_SYMCOUNT: Symbol Count on the lanes
- CDN_PCIE_PRINT_RXMARGIN: RX Margin
- CDN_PCIE_PRINT_PIPE_PM: Pipe Power Management
- CDN_PCIE_PRINT_ACK_NAK: Ack/Nak Timer information
- CDN_PCIE_PRINT_SKEW: Skew data

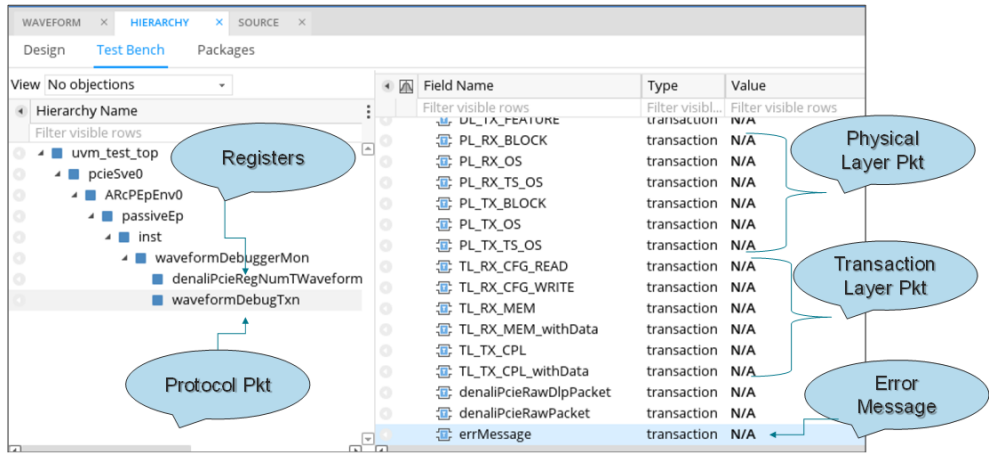
Efficient Debugging With Waveforms

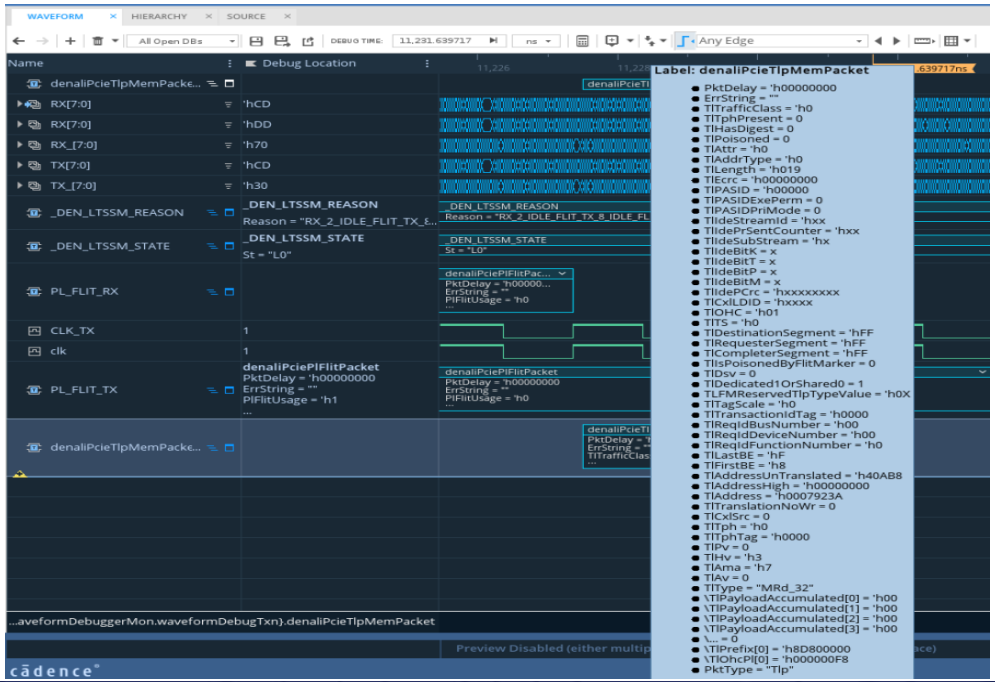
- **Graphical Visualization of PCIe® Protocol**
  View PCIe protocol transactions and packets in a graphical waveform format for intuitive debugging
- **In-Waveform Protocol Specification Error Display**
  Protocol specification errors are directly visualized within the waveform, enabling quick identification and resolution
- **Layered Transaction View**
  Access a per-layer breakdown of PCIe transactions and packets for detailed protocol analysis
- **ASCII Representation of Values**
  Wherever applicable, values such as FSM states and enum variables are displayed in ASCII for better readability
- **Graphical PCIe Register Representation**
  PCIe registers are visualized graphically, simplifying register-level debugging

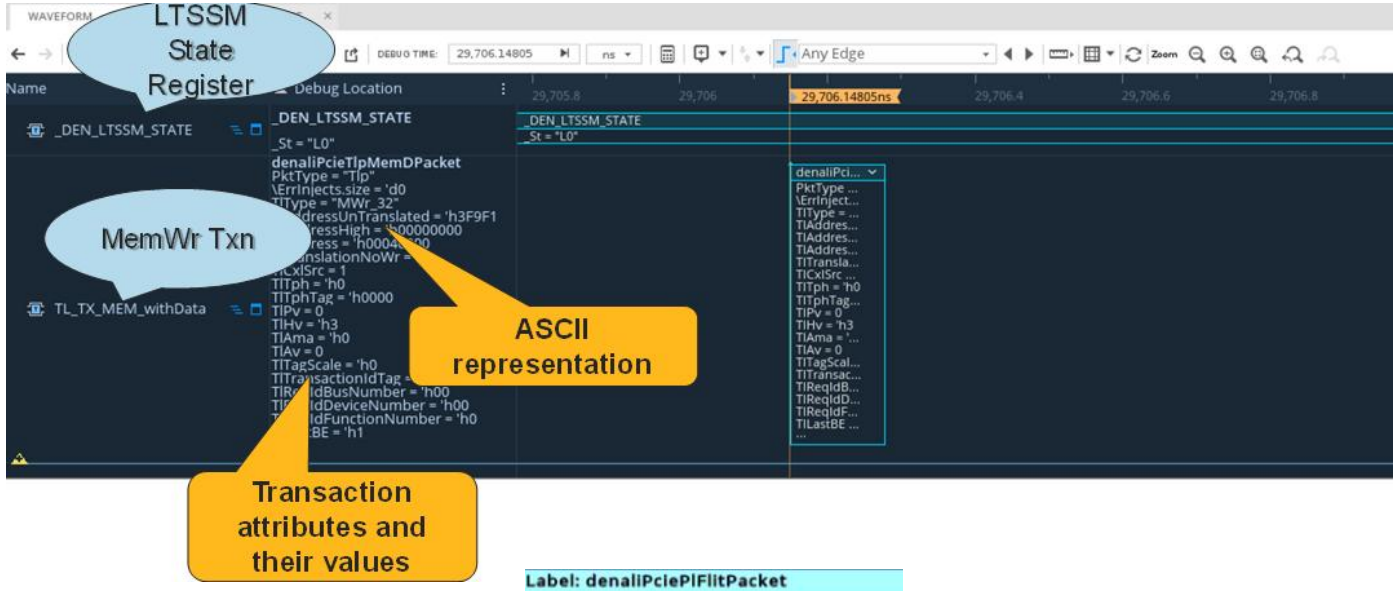
Continued…

- **Transaction Attribute Viewer**
  Inspect packet contents through a dedicated transaction attributes view to streamline debugging
- **Unified Debugger Window**
  Signals and transactions can be logged and viewed together in the same waveform debugger window
- **FSM State Transition Visualization**
  Finite State Machine (FSM) transitions are clearly visualized on the waveform for easier state tracking
- **Multi-Protocol Debugging Support**
  Simultaneously debug multiple protocols in a single SoC setup
- **Smart Log Integration**
  Smart logging highlights UVM_ERROR messages along with their corresponding debug timestamps

Smart Effective Debugging on Waveform







Label: _EXP_LINK_ST

- _LkSpd = 'h6
- _LkWidth = 'h08
- _TrainErr = 0
- _LkTrain = 0
- _slotClk = 1
- _DLActive = 0
- _mgmtWidth = 0
- _autoWidth = 0

Label: _DEN_LTSSM_STATE

- St = "RecoveryRcvrCfg"

Label: denaliPciePlFlitPacket

- PlFlitType = "Nop"
- PlFlitUsage = 'h0
- PlFlitPriorFlitType = 0
- PlFlitDllpPayloadType = 0
- PlFlitReplayCmd = "Ack"
- PlFlitSeqNum = 'h003
- PlFlitDllpPayload = 'h31000000
- PlFlitOptUpdateFC0_Or_FlitMarker1 = 0
- PlFlitOptUpdateFCvc = 'h0
- PlFlitOptUpdateFCSharedNPH = 'h00
- PlFlitOptUpdateFCSharedPH = 'h00
- PlFlitOptUpdateFCSharedPD = 'h000
- PlFlit_FlitMarker_FlitStatus = 'h0
- PlFlit_FlitMarker_PTMcontained = 0
- PlFlitCrc = 'h79AD8128B70F4CE7
- PlFlitEcc1 = 'hBF28
- PlFlitEcc2 = 'hF994
- PlFlitEcc0 = 'h1E68
- PktType = "PlFlit"
- PktDelay = 'h00000000
- PlType = "flit"
- ErrString = ""

Label: denaliPcieTrainingSet

- " PlTsLinkNumber[0]" = 'h000
- " PlTsLaneNumber[0]" = 'h000
- " PlTsNFTS[0]" = 'h000
- " PlTsDataRate[0]" = 'h0EF
- " PlTsTrainingControl[0]" = 'h000
- " PlTsIdentifier[0]" = 'h045
- PktType = "OrderedSet"
- PktDelay = 'h00000000
- PlType = "ts2_set"
- ErrString = ""

Label: _DEN_LTSSM_REASON

- Reason = "ALL_LANES_RX_8_MATCHING_TS1_OR_TS2_WITH_SPEED_CHANGE_BIT_EQUALS_DIRECTED_SPEED_CHANGE

Enhancing Debugging by Tracking Packets

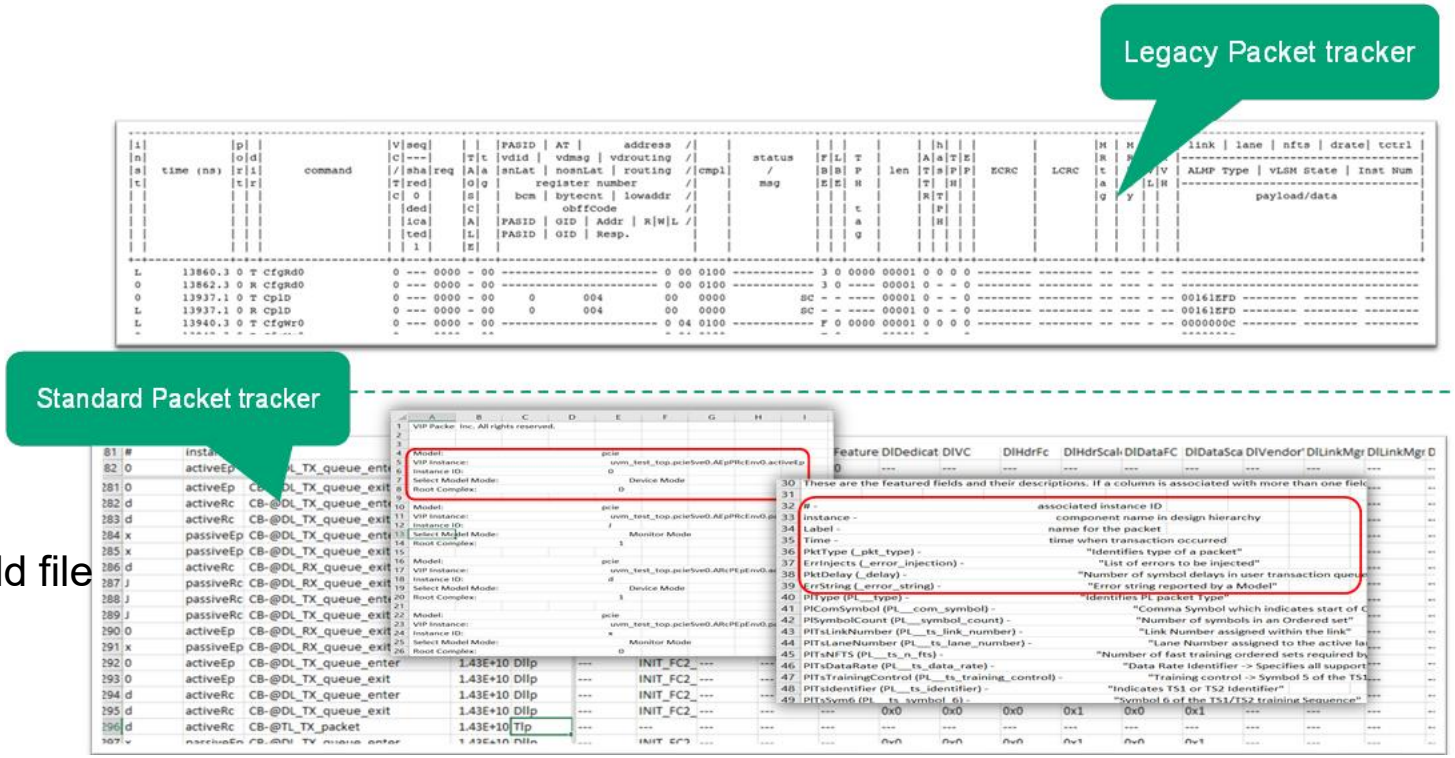
Enable VIP Packet Tracker

Run Simulation

Master Packet Tracker Generated (~350 fields)

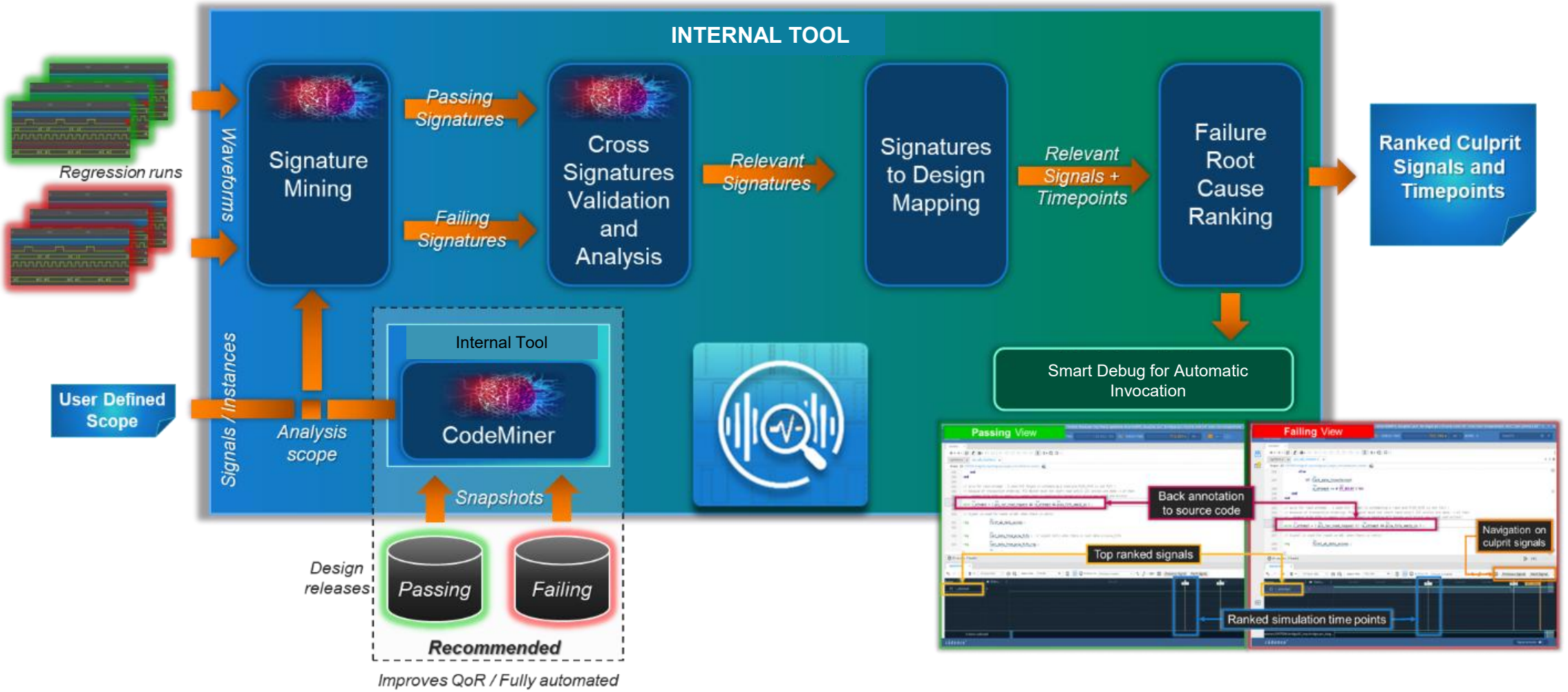
Perform filtering via inbuild file

Extended Packet Tracker generated (Reduced fields)



User Centric Configurable Debug

Automated Result Debug

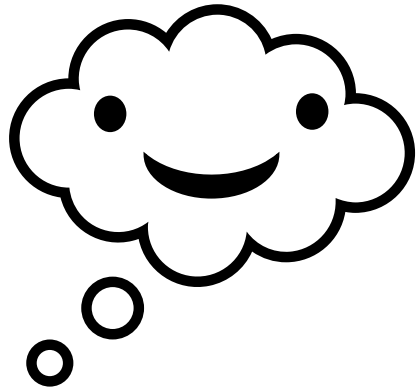


Summary and Best Practices

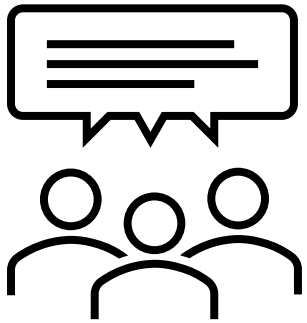
- **Comprehensive PCIe Gen6 Debug Strategy**: The methodology addresses the growing complexity of PCIe® 6.0/6.1 verification by combining protocol-aware VIP monitoring with smart trace and efficient visual-driven debugging
- **Smart Passive Monitoring Architecture**: Embedding PCIe Gen6 VIP as a passive monitor enables non-intrusive, real-time protocol checks across TL, DL, and PHY layers
- **FLIT Mode and Protocol Awareness**: The approach effectively handles FLIT mode intricacies, ensuring accurate validation of transactions and data link layer behaviors
- **Meaningful Error Messaging**: Carefully designed error messages, backed by real-world examples, enhance clarity and reduce time-to-debug
- **Robust Callbacks and Error Injection**: Callback mechanisms and targeted error injection strategies validate corner cases and stress-test the DUT under edge conditions
- **Extensive Debug Information**: Rich trace logs and protocol event correlation provide deep visibility into system behavior, aiding faster root-cause analysis

Continued…

- **Waveform-Based Debugging**: Visual waveform techniques, combined with traceability tools, accelerate issue localization and resolution
- **Packet-Level Tracking**: Tracking packets across layers and correlating them with protocol events improves traceability and debugging accuracy
- **Automated Debug Frameworks**: Automation scripts reduce manual effort, enabling scalable and repeatable debug cycles
- **Improved Verification Efficiency**: The combined strategy significantly reduces debug time, improves coverage, and ensures compliance with PCIe® Gen6 specifications
- **Scalable to Other Protocols**: The methodology's modular and automation-friendly nature makes it adaptable to other high-speed interface protocols



THANK YOU



ANY QUESTIONS ?