DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

accellera SYSTEMS INITIATIVE™

# UCIe based Design Verification

Short Workshop

# Anunay Bajaj
Sr Principal Product Engineer
Cadence Design Systems Pvt. Ltd.

*Anunay is a seasoned expert of Simulation and Acceleration based verification of peripheral interconnects like PCIe and CXL. He has worked on several broad market and turnkey projects from Unit Level to a complete System level testbench verification. Anunay is responsible for defining models and methodologies for UCIe design verification*

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

accellera SYSTEMS INITIATIVE™

# Sundararajan Ananthakrishnan

## Application Engineer Architect

## Cadence

Sundar has been at Cadence for 13 years, with an overall Experience of 23+ years in the field of SoC & IP Verification. Sundar leads the Cadence Advance Verification & Systems Technical field team for India region and drives Advanced ML/AI and Verification Solutions in the region.

# UCIe based Design Verification

Anunay Bajaj

Sundarajan Ananthakrishnan

Cadence Design Systems Pvt. Ltd.

accellera SYSTEMS INITIATIVE™

cadence®

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

# Agenda

Cadence Generic VIP Overview

Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

Getting started with Test sequences

Debug Aids

# Agenda

Cadence Generic VIP Overview

Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

Getting started with Test sequences

Debug Aids

# Cadence Generic VIP Overview – Key Features

- Cadence follows Consistent architecture for all its VIPs. (Verification IP)
- It ensures a similar user interface for all its VIPs and enables seamless integration of different VIPs.

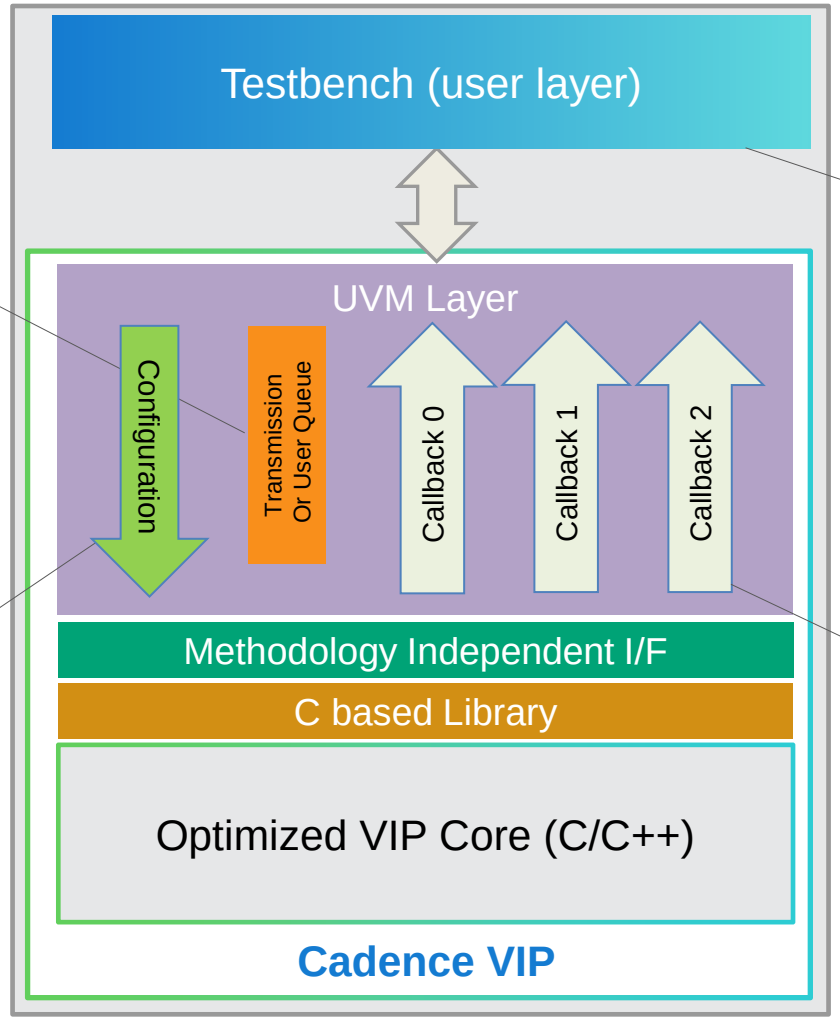
- Key Features
  - Optimized core-based architecture
  - VIP Configuration through PureView™ GUI tool using UVM configuration or SOMA (Specification of Modeling Architecture) files.
  - Queues for seamless movement of traffic.
  - Extensive Callbacks for precise control and monitoring.
  - Various debug utilities ranging from text files to waveform.

# Cadence Generic VIP Overview - Testbench and VIP core Interaction

User has full visibility into and controllability over VIP core by using:
- Configuration
- Queues
- Callbacks

Interaction, using transmission or user queue, receives transactions from the User Layer that are further transmitted by VIP core on bus

Interaction, using configuration, at the Testbench User Layer is passed to the core

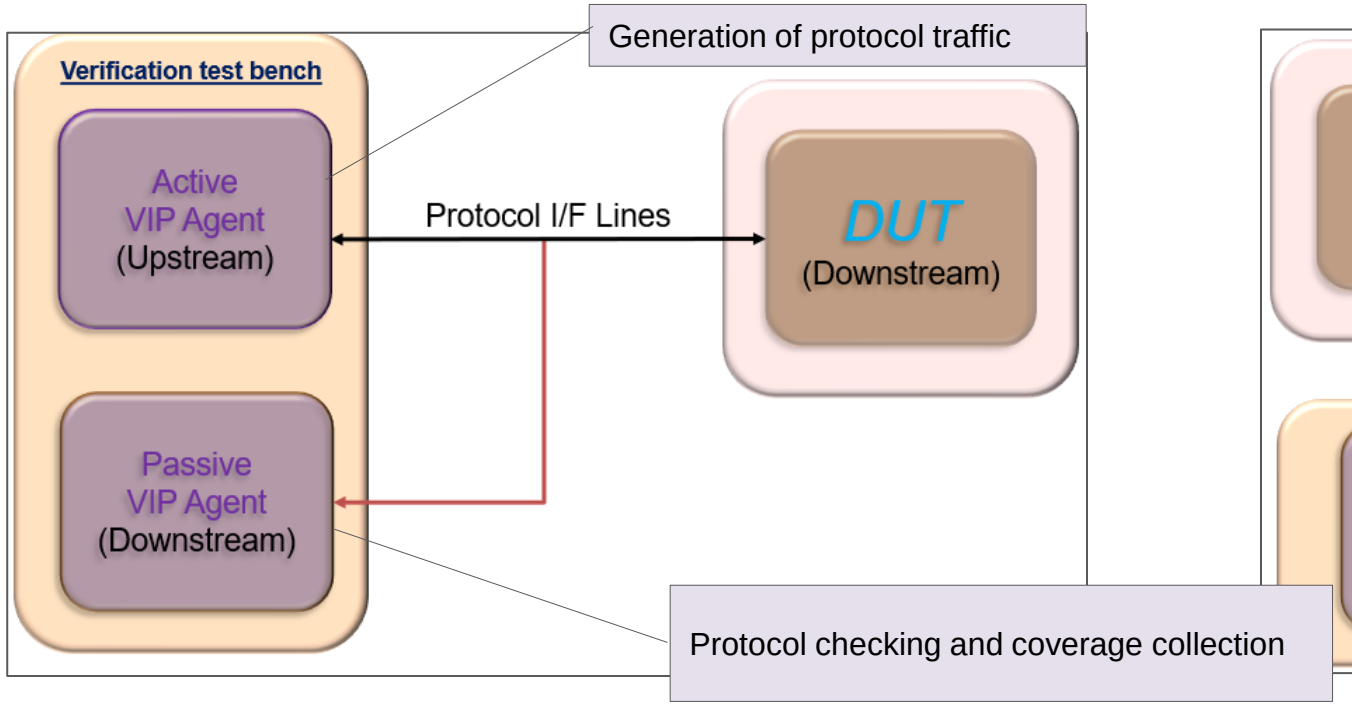



SV UVM Testbench User Layer automatically places transactions into Transmission/User Queue.

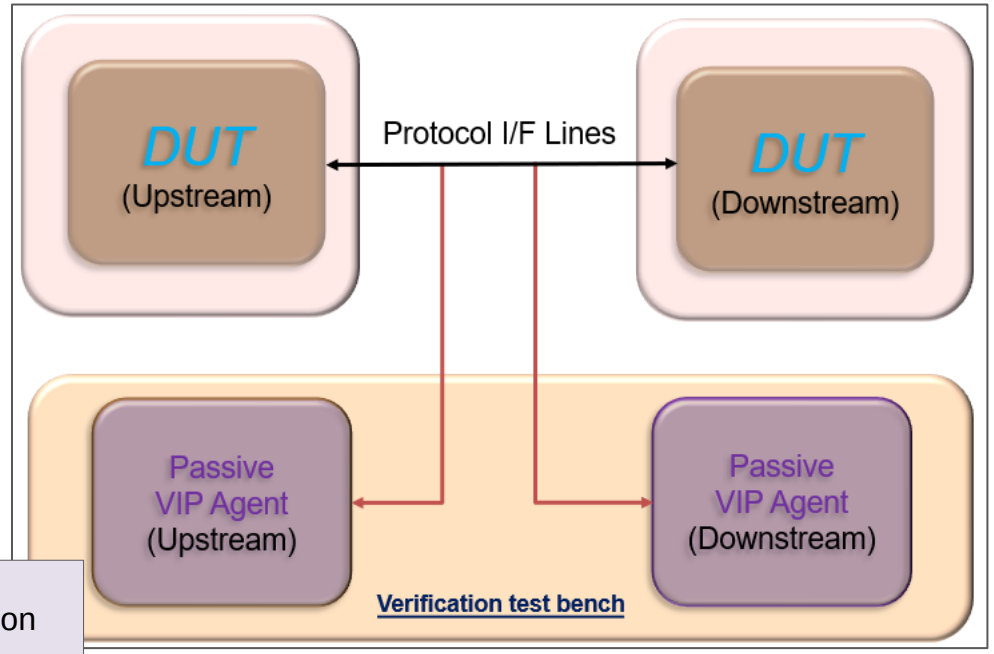
Interaction, using callbacks (0,1 and 2 here), is triggered by the VIP core and is accessible at the Testbench User Layer

# Cadence Generic VIP Overview

## Usage 1




## Usage 2



# Agenda

Cadence VIP Overview

Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

Getting started with Test sequences

Debug Aids

# Cadence UCIe VIP Key Features

- UCIe is a chiplet interconnect protocol which is used to connect two chiplets on a substrate.
- UCIe helps to improve overall chip yield by connecting multiple chiplets. It also enables chiplets of different size to interconnect and hence realize the vision of optimized system architecture.

## Protocol Features

- Layered Architecture (layers - protocol, Die-to-die adapter, physical)
- High level protocol support (PCIe, CXL, Raw Data Stream)
- Supports Standard and Advanced packaging.
- Registers access defined by UCIe specification
- Standard Interface support
  - RDI (*Raw Die-to-Die Interface*)
  - FDI (*Flit Aware Die-to-Die Interface*)
  - UCIe Serial link (MainBand and SideBand)

## Verification Features

Used for Block level and Full Stack verification

Ease of Configuration

Active and Passive Agent Support

Extensive protocol checks and coverage

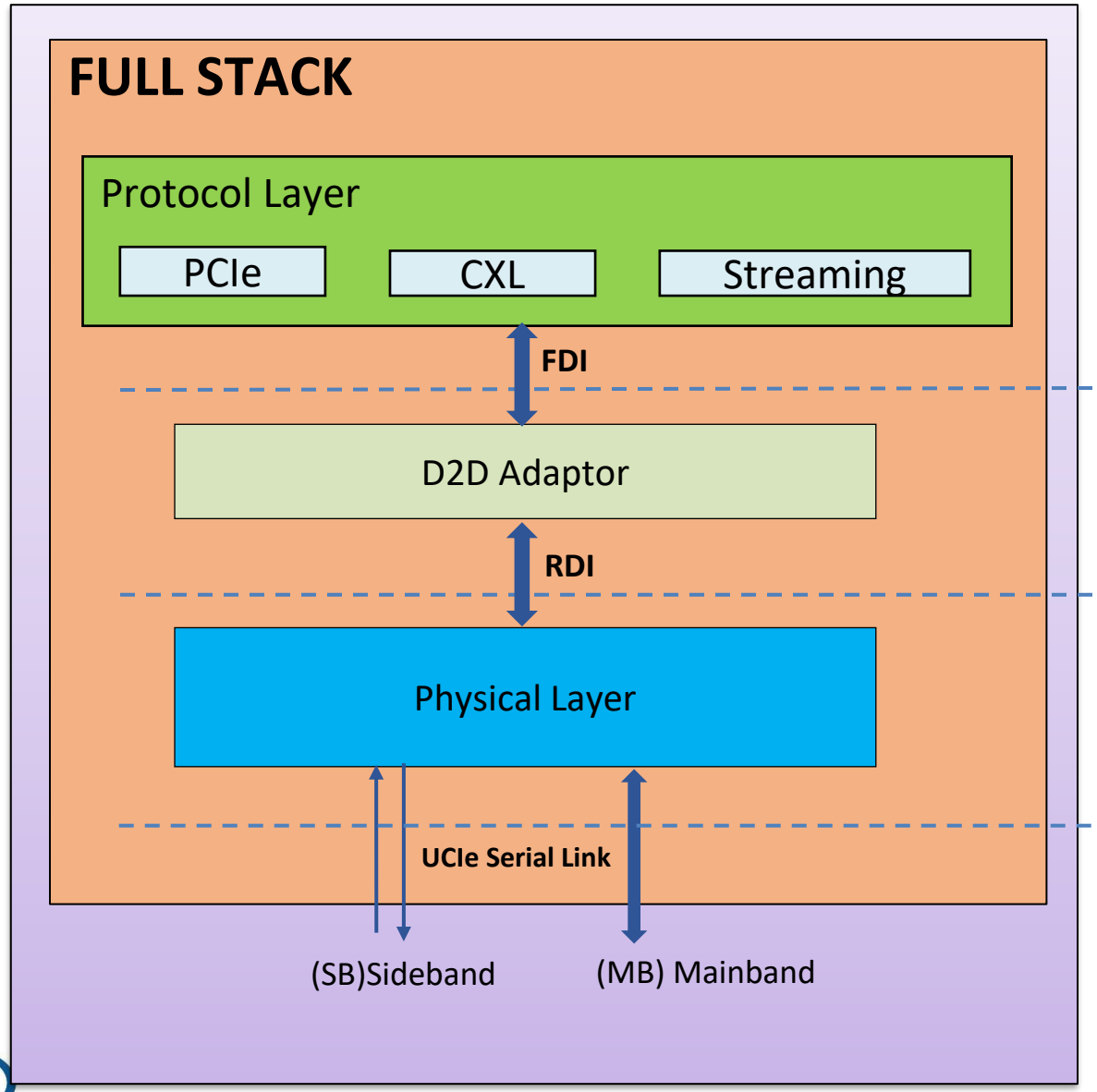
Easy testbench integration at IP, SoC, and System levels

Supports Verilog, SV and SV/UVM. Also supports all major simulators

Packet tracker and Waveform debugger for efficient debugging

Error Injection using callbacks

# VIP Architecture




- **PCIe 6.0,** (PCIe non-FLIT via CXL.io)
- **CXL 2.0 - 68B, CXL3.0 - 256B**
- **Streaming Protocol**

*(FDI) Flit Aware Die-to-Die Interface*

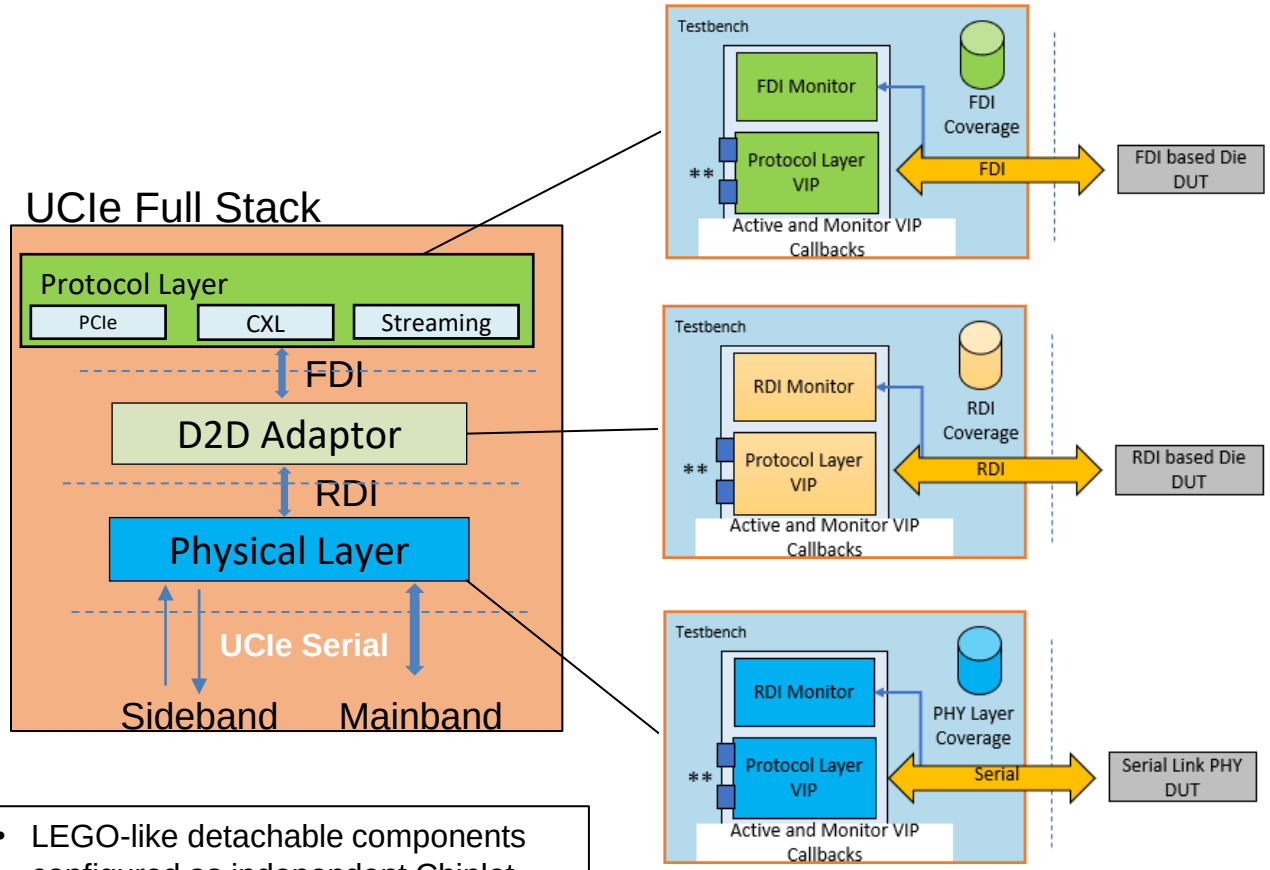
- **D2D Adapter coordinates with the Protocol Layer and the Physical Layer**
- **Link State Management and Parameter negotiation**

*(RDI) Raw Die-to-Die Interface*

- **Physical Layer** for Data Path, Link Training, Lane Repair, parameter exchanges, register access etc.
- Standard and Advanced Package
- Single Module & Multi module PHY

*UCIe Serial Link (Mainband & Sideband Lanes)*

# Die Disaggregated Verification




- LEGO-like detachable components configured as independent Chiplet interconnects of UCIe

**** Pseudo Ports are special Queues for Packet/Flit handling**

## Verification Focus

- Die-to-Die Detailed protocol verification
- Observability & Controllability
- Full Callback & Error Injection
- Layer-wise scoreboard
- Interface-wise Coverage

## Advantages

- No need for the entire design - Head start with verification prior to complete Design availability
- Expedite verification with Block level test cases
- IP design agnostic – Forward compatible
- Plug and Play at Subsystem/System level
- Bypass Link Training & Setup controls
- Bypass Interface dependency using Pseudo Ports

# Agenda

Cadence VIP Overview

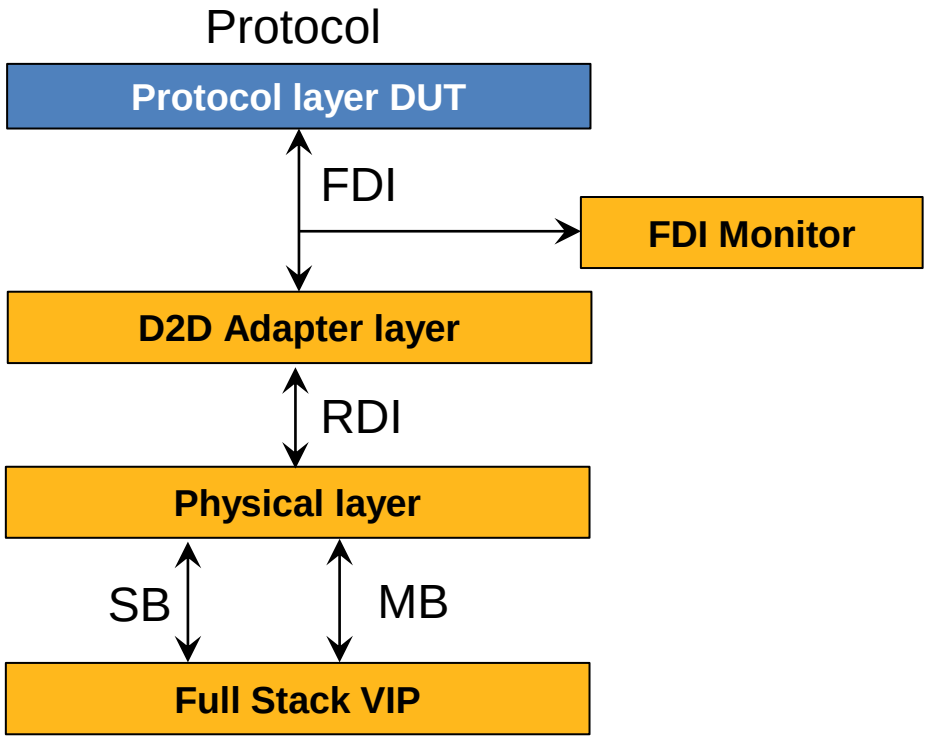
Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

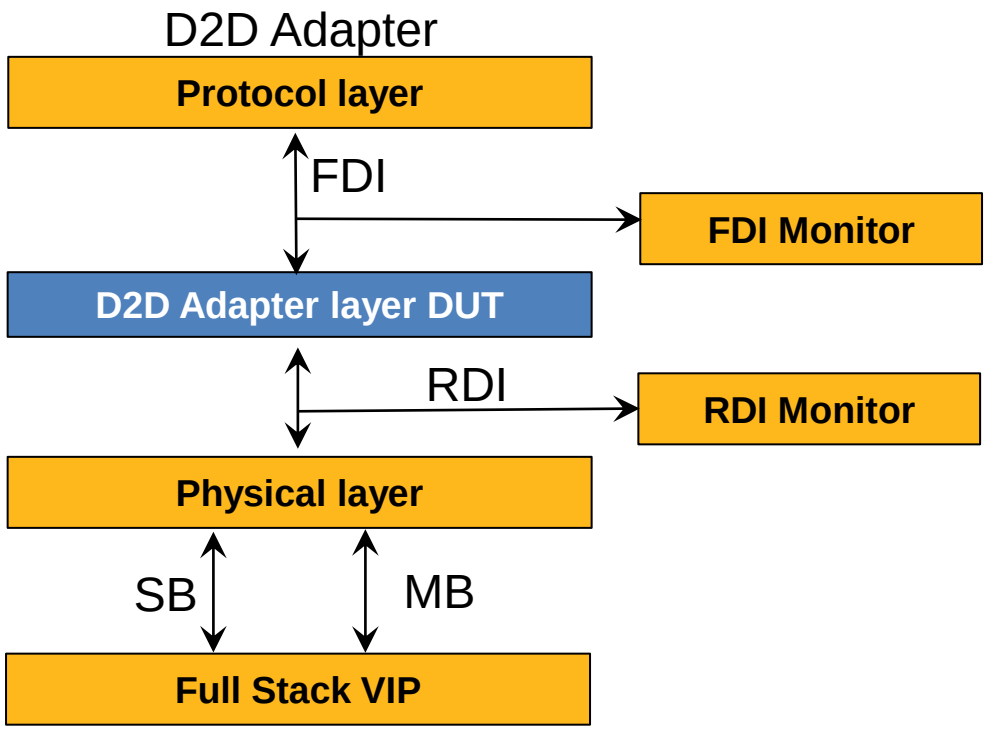
Getting started with Test sequences

Debug Aids

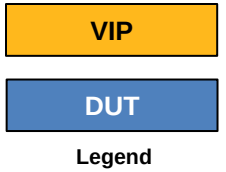
# VIP Usage Topology – Partial stack Verification



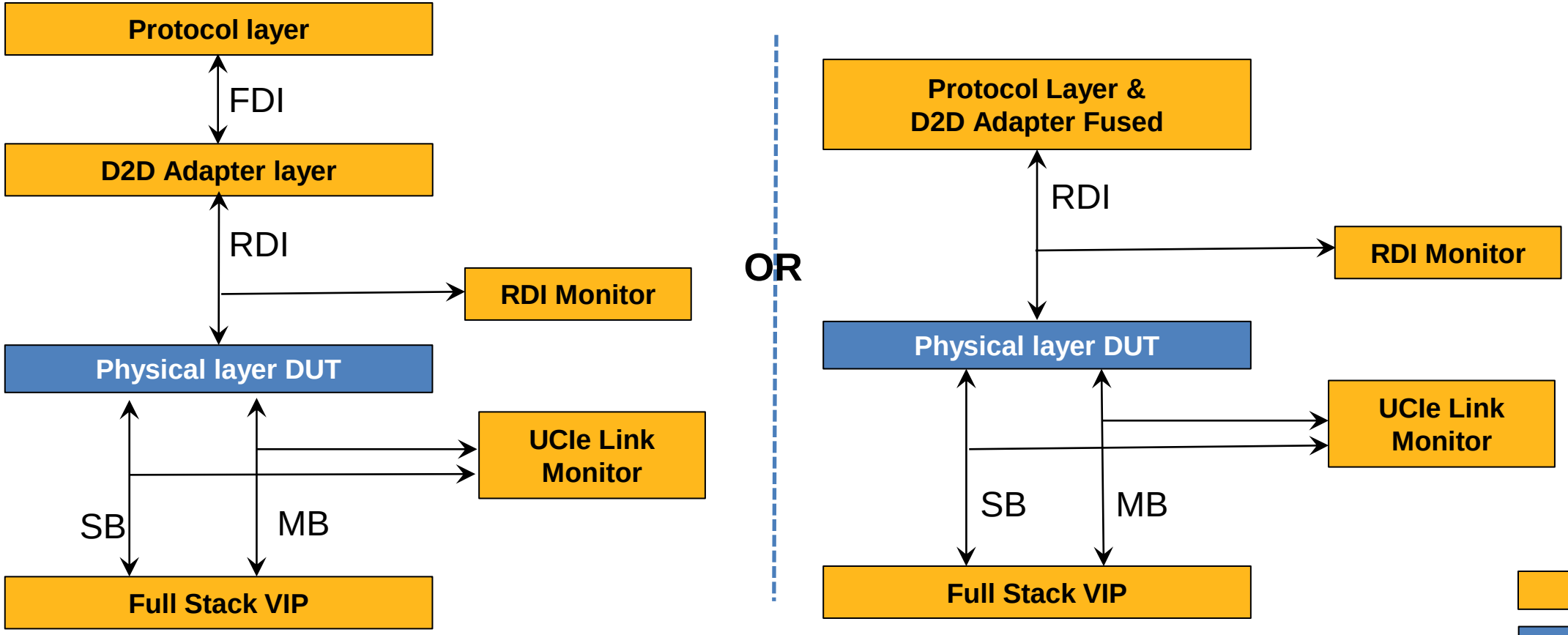

- Focus on FDI Protocol checks and Coverage
- Focus on Protocol layer Functionality




- Focus on FDI and RDI Protocol checks and Coverage
- Focus on D2D Adapter layer Functionality

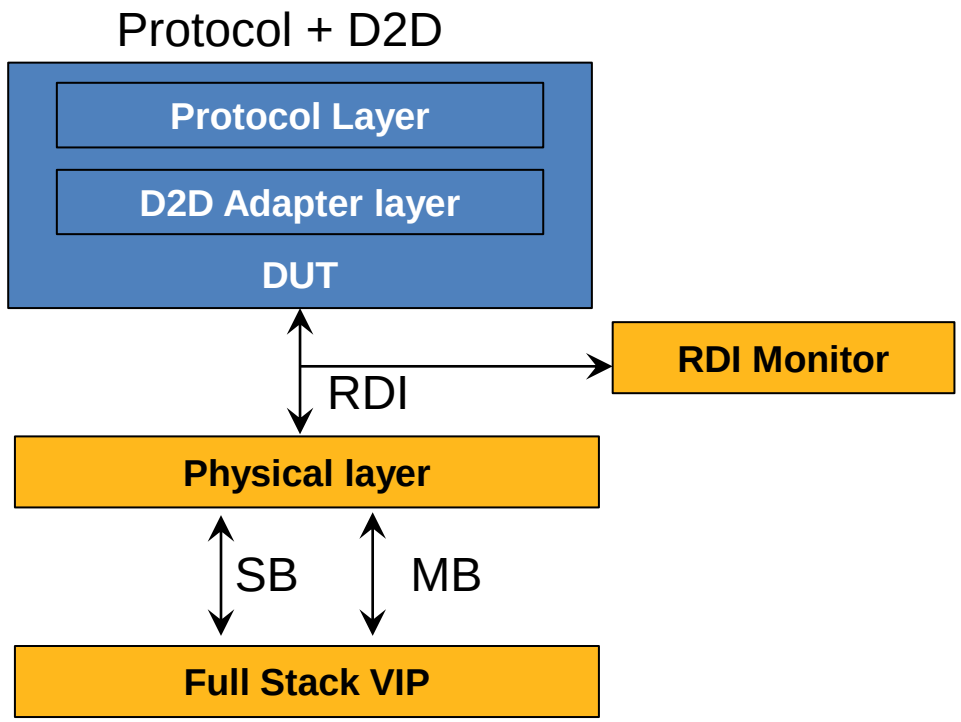


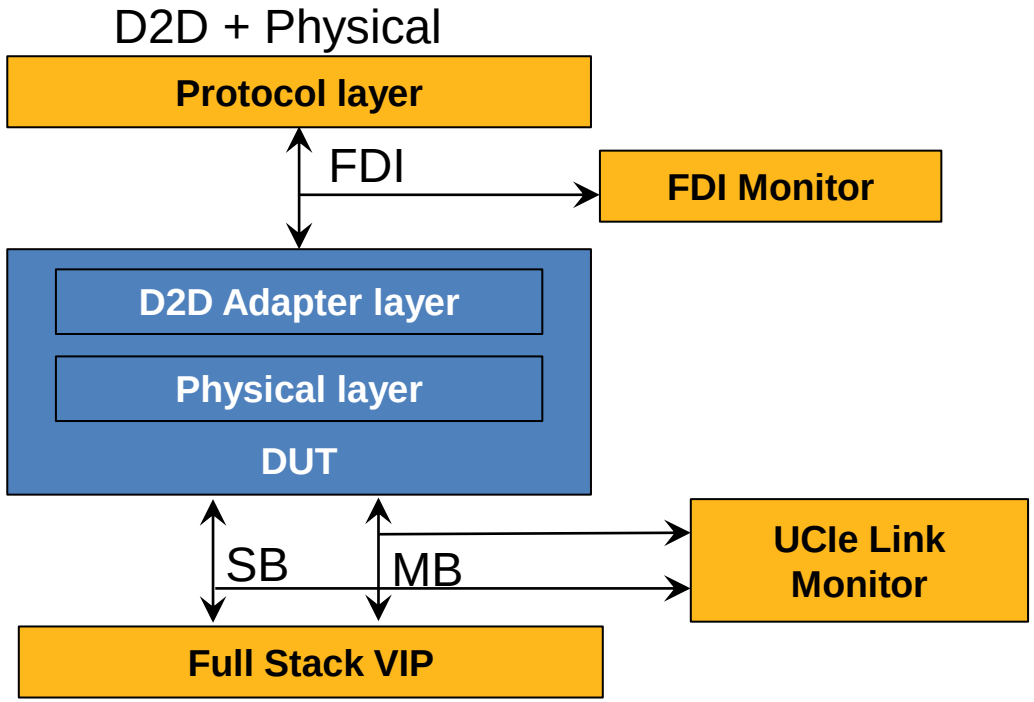
# VIP Usage Topology – Partial stack Verification (Physical)




- Focus on RDI and UCIe Serial link Protocol checks and Coverage
- Focus on Physical layer Functionality

# VIP Usage Topology – Partial stack Verification




- Focus on RDI Protocol checks and Coverage
- Focus on D2D Adapter layer Functionality

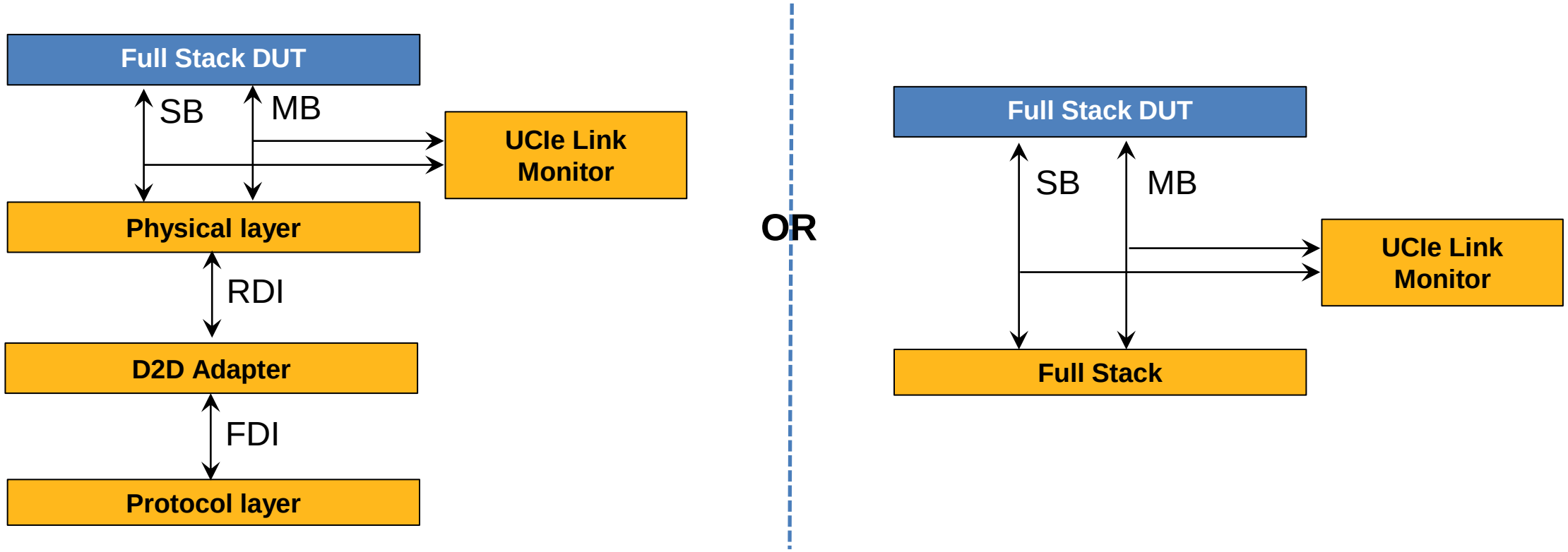
- Focus on FDI and UCIe Link checks and Coverage
- Focus on D2D Adapter and Physical layer Functionality

VIP

DUT

Legend

# VIP Usage Topology – Full stack Verification



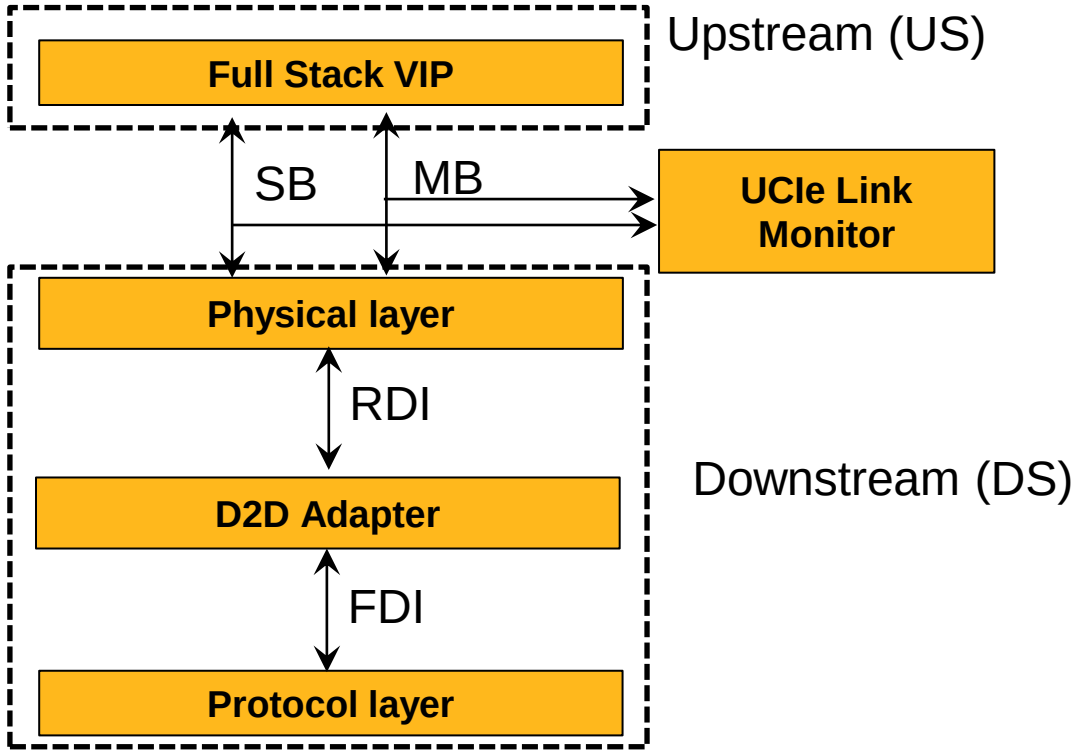

- Full Protocol checks and Coverage
- Focus on end-to-end Functionality
- Ideal for Sub-system and System verification

VIP

DUT

Legend

# Workshop– Full stack Verification
# Simplified Topology




- Both Upstream and Downstream are VIPs
- Upstream is a unified Full Stack with Serial Mainband & Sideband connections
- Downstream shows capability of VIP as LEGO-like blocks. Each layer serves as an independent UVM agent with dedicated interfaces

VIP

DUT

Legend

# Integrating Full Stack Environment

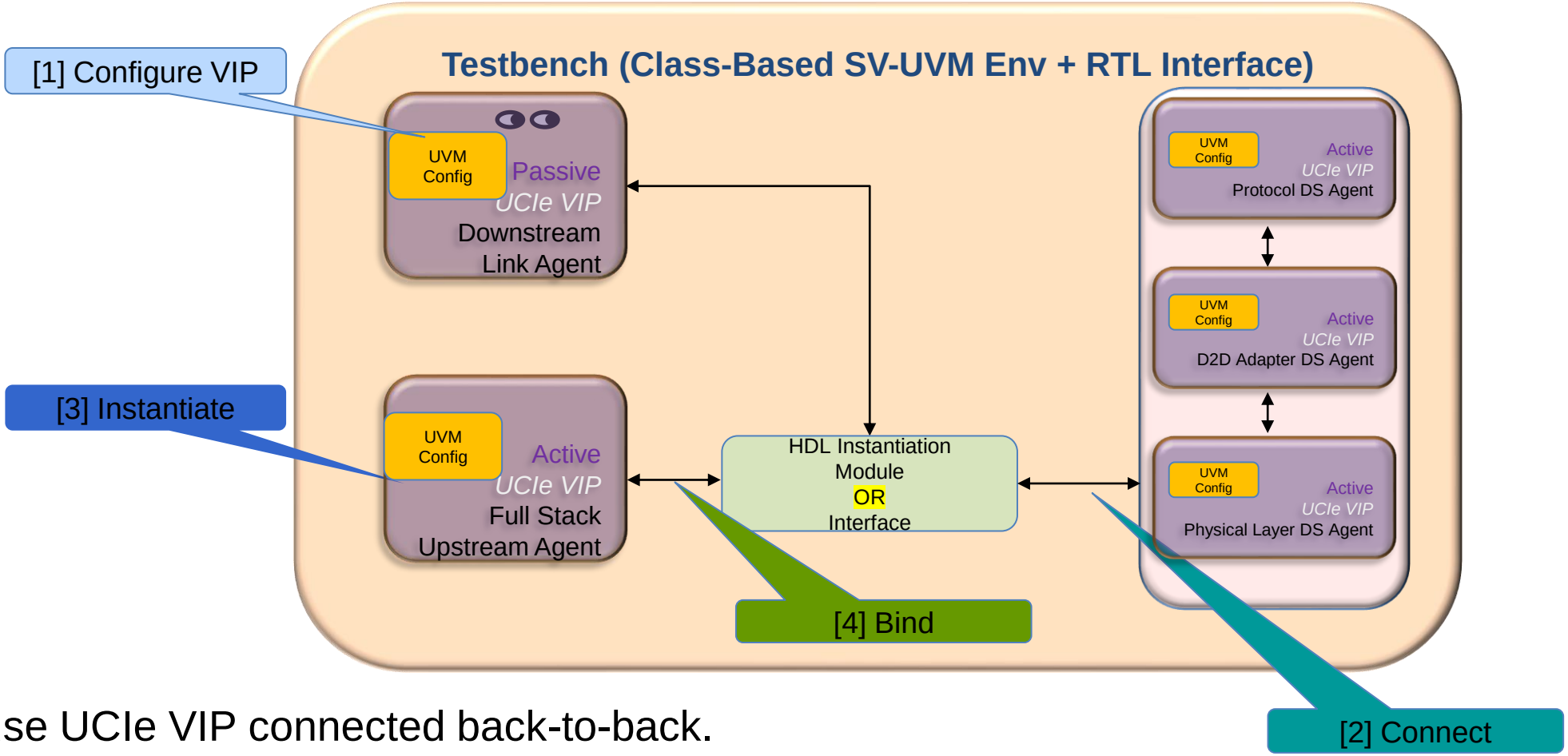
- Configure UCIe VIP as per DUT Verification requirements and then connect the interface with DUT.
- After that, we need to instantiate the interface in the testbench and then bind it with the DUT environment.
- At last, we need to align the scripts to simulate the verification environment.

1. Configure VIP
2. Connect VIP Interface with DUT
3. Instantiate VIP UVM Agent in Testbench
4. Bind Agents and Interfaces

**Notes:-**

- We will use UCIe VIP connected back-to-back.
- One UCIe VIP instance needs to be replaced with User Design.

# Full Stack Special Topology Diagram

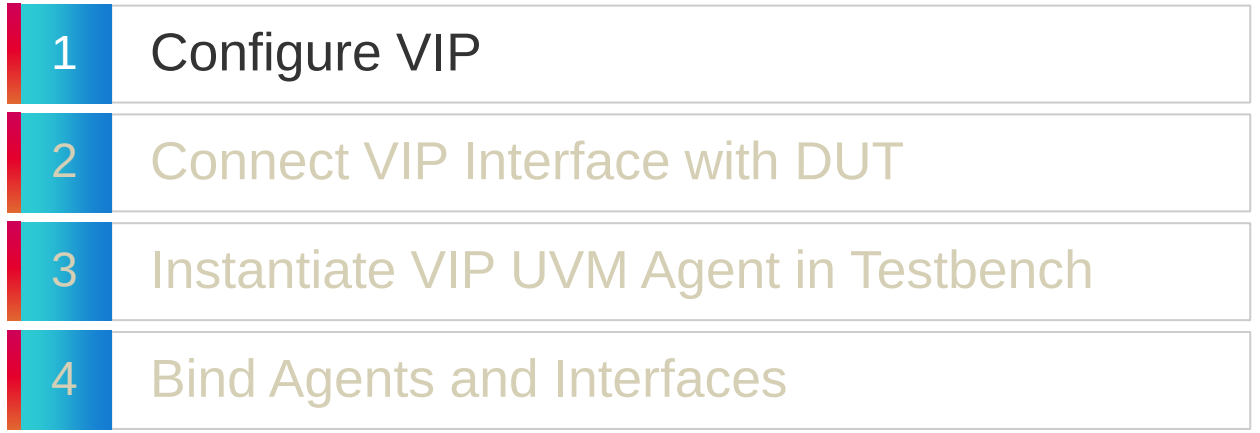



**Notes:-**

- We use UCIe VIP connected back-to-back.
- One UCIe VIP instance would be replaced with User Design.

# Step 1: Configure VIP

- Configure the UCIe VIP using the PureView™ tool, which is a common tool for all Cadence VIPs.
- All the screenshots involved in the configuration along with their description are shown in subsequent slides.
- The VIP top wrapper is also generated along with the configuration file in this step.

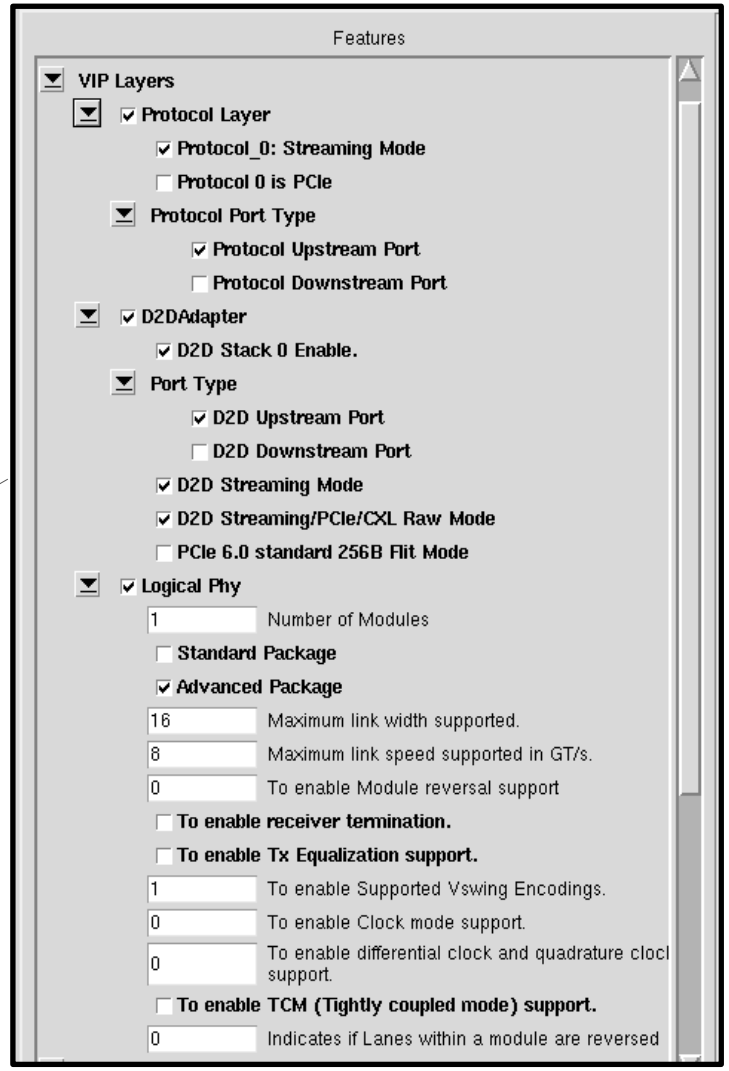


# Configure VIP Upstream Port (UP)

- Configuration Selection (Upstream port)
  - All VIP layers selected for Full Stack with Upstream Port
  - Individual layer specific configurations can be done
  - Advanced package selected for PHY layer
  - Other options are kept as default.

- Save the Final Upstream SINGLE configuration class as

```
cdnUcieUvmUserConfigFullStackUsAgent
```



# Configure VIP Downstream Port (DP)

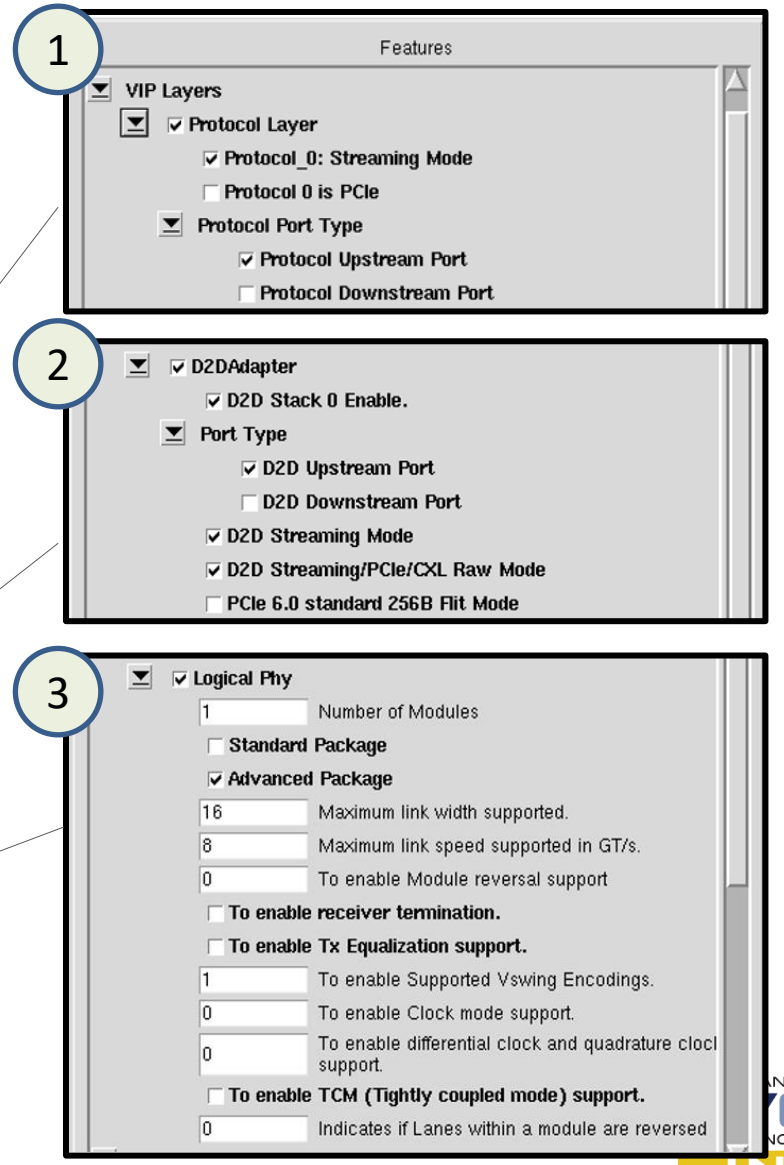
- 3-step process
- Configuration Selection (Downstream port)
  - Configure Physical, D2D Adapter and Protocol Layer separately
  - Individual layer specific configurations can be done
  - Advanced package selected for PHY layer
  - Other options are kept as default

- Final Downstream configurations are 3 standalone classes for 3 agents

`cdnUcieUvmUserConfig`**`ProtoDs`**`Agent`

`cdnUcieUvmUserConfig`**`AdpDs`**`Agent`

`cdnUcieUvmUserConfig`**`PhyDs`**`Agent`



# Consolidated VIP Configuration File

- Downstream has individual config classes mapped to Layers
  - Physical Layer
  - D2D Adapter Layer
  - Protocol layer
- Upstream has a common config class
- Multiple custom config classes are consolidated into one file for ease of use

`cdnUcieUvmUser`**`FullStackConfig`**`Inst.sv`

```
//Downstream (DS) agent
    class cdnUcieUvmUserConfigPhyDsAgent extends cdnUcieUvmUserConfigDsAgent;
        function new(string name = "cdnUcieUvmUserConfigPhyDsAgent");
            super.new(name);     //Set All Default Feature Values
            LogicalPhy = 1;      //Modify PHY configs
            . . .
        endfunction
    endclass
    class cdnUcieUvmUserConfigAdpDsAgent extends cdnUcieUvmUserConfigDsAgent;
        function new(string name = " cdnUcieUvmUserConfigAdpDsAgent ");
            super.new(name);     //Set All Default Feature Values
            D2DAdapter = 1;       //Modify D2D Adapter configs
            . . .
        endfunction
    endclass
    class cdnUcieUvmUserConfigProtoDsAgent extends cdnUcieUvmUserConfigDsAgent;
        function new(string name = "cdnUcieUvmUserConfigProtoDsAgent");
            super.new(name);     //Set All Default Feature Values
            Protocol = 1;         //Modify Protocol layer configs
            . . .
        endfunction
    endclass
//Upstream (US) agent
    class cdnUcieUvmUserConfigFullStackUsAgent extends cdnUcieUvmUserConfigUsAgent;
        function new(string name = " cdnUcieUvmUserConfigFullStackUsAgent ");
            super.new(name);     //Set All Default Feature Values
            LogicalPhy = 1;      //Modify all layers'configs here
            D2DAdapter = 1;
            Protocol = 1;
            . .  .
        endfunction
    endclass
```

Physical layer DS VIP Agent Configuration

D2D Adapter layer DS VIP Agent Configuration

Protocol layer DS VIP Agent Configuration

Full Stack US VIP Agent Configuration

# Step 2: Connect VIP Interface with DUT

- UCIe VIP Interface generated in the previous step can connect VIP with DUT.

- The Full Stack example present in the installation path used one configuration file. It contains the configuration of all the modules.

- Modules used in Full stack example:

  Full Stack upstream VIP

  Full Stack downstream VIP using explicit connections between

  -Protocol layer

  -D2D Adapter layer

  -Physical layer

1. Configure VIP
2. Connect VIP Interface with DUT
3. Instantiate VIP UVM Agent in Testbench
4. Bind Agents and Interfaces

# Interface connections

Instantiate Full Stack US agent and Protocol+D2D+Physical (acting as Full Stack DS agent) in the testbench module. Instantiations are shown below.

1

- Physical layer interface of DS Agent
- D2D Adapter layer interface of DS Agent
- Protocol layer interface of DS Agent

Downstream

2

- Full Stack interface of US Agent

Upstream

```
module testbench;
…
// Instantiate the DS agent.
  uciePhy_interface    dsAgent(
       .tx_ck_p              (tx_ck_p),
       .rx_ck_n              (rx_ck_p),
       .tx_vld               (rx_vld),
       .tx_data              (rx_data),
       .rx_vld               (tx_vld),
       .rx_data              (tx_data),
       …
       .rdi_lp_data          (dsAgent_rdi_lp_data),
  );
  ucieAdp_interface    adpDsAgent(
       .rdi_lp_data          (dsAgent_rdi_lp_data),
       …
       .fdi_lp_data0         (dsAgent_fdi_lp_data0),
  );
  ucieProtocol_interface     protoDsAgent(
       .fdi_lp_data0         (dsAgent_fdi_lp_data0),
  );

// Instantiate the US agent.
  ucieFullStack_interface    fullStackUsAgent(
     .tx_ck_p                (rx_ck_p),
     .rx_ck_p                (tx_ck_p),
     .tx_vld_rd              (rx_vld_rd),
     .tx_data_rd             (rx_data_rd),
     .rx_vld_rd              (tx_vld_rd),
     .rx_data_rd             (tx_data_rd),
     …
  );
endmodule
```

cdnUcieUvmUserFullStackTb.sv

3 Connection between UP and DS Agent. (Tx ports are connected to Rx ports and vice versa)

4 Internal connection between Phy layer and D2D Adapter layer of DS

5 Internal connection between Protocol layer and D2D Adapter layer of DS

# Step 3: Instantiate

- This will Instantiate UCIe agents and their UCIe configurations
- It also connects UCIe configurations to their respective UCIe agents
- All the screenshots involved in the configuration and their description are shown in subsequent slides

1. Configure VIP
2. Connect VIP Interface with DUT
3. Instantiate VIP UVM Agent in Testbench
4. Bind Agents and Interfaces

# Base User Agent

The base common user agent (`cdnUcieUvmUserAgent`) later configured as any of the below UCIe agents

- Full Stack for Upstream
- Physical Layer for Downstream
- D2D Adapter Layer for Downstream
- Protocol Layer for Downstream

1. Reference to model memory
2. User Agent Build Phase
3. User Agent Connect Phase
4. Return State identification number based on state input.
5. Set the severity of an error.

```
class cdnUcieUvmUserAgent extends cdnUcieUvmAgent;

    cdnUcieUvmUserMemInstance memInst;

    virtual function void build_phase(uvm_phase phase);
    ...
    endfunction : build_phase

    virtual function void connect_phase(uvm_phase phase);
    ...
    endfunction : connect_phase

    virtual function int getPhyErrLogStateFromEnum(denaliUcieLtsmStateT st);
    ...
    endfunction : getPhyErrLogStateFromEnum

    ...

    function void setErrorSeverity(denaliUcieErrorTypeT errId, denaliUcieErrCtrlSeverityT serv,
                                   denaliUcieErrCtrlDirectionT dir = DENALI_UCIE_ERR_CTRL_DIRECTION_TRx);

    ...
    endfunction
endclass
```

cdnUcieUvmUserAgent.sv

# Full Stack Environment

Full stack environment contains

- UCIe Agents and Configuration Class Instances
- Mapping of configuration classes to their respective UCIe agents
- Setting necessary callbacks

1. Upstream and Downstream Instances
2. UCIe Configuration Class Instances
3. Mapping of configuration classes to their respective UCIe agents
4. Setting necessary callbacks

```
class cdnUcieUvmUserFullStackEnv extends uvm_env;

    //UCIe Agents Instances
    cdnUcieUvmUserAgent                    fullStackUsAgent;    //US full stack
    cdnUcieUvmUserAgent                    dsAgent;             //DS Physical layer
    cdnUcieUvmUserAgent                    adpDsAgent;          //DS D2D Adapter layer
    cdnUcieUvmUserAgent                    protoDsAgent;        //DS Protocol layer

    //UCIe Configuration Class Instances
    cdnUcieUvmUserConfigFullStackUsAgent   fullStackUsAgentCfg;//US full stack
    cdnUcieUvmUserConfigPhyDsAgent         dsAgentCfg;          //DS Physical layer
    cdnUcieUvmUserConfigAdpDsAgent         adpDsAgentCfg;       //DS D2D Adapter layer
    cdnUcieUvmUserConfigProtoDsAgent       protoDsAgentCfg;     //DS Protocol layer
    ...
    ...
    virtual function void build_phase(uvm_phase phase);
        //Mapping of configuration classes to their respective UCIe agents
        uvm_config_object::set(this,"dsAgent"          , "cfg", dsAgentCfg);
        uvm_config_object::set(this,"adpDsAgent"       , "cfg", adpDsAgentCfg);
        uvm_config_object::set(this,"protoDsAgent"     , "cfg", protoDsAgentCfg);
        uvm_config_object::set(this,"fullStackUsAgent", "cfg", fullStackUsAgentCfg);
        ...
    endfunction : build_phase
    ...
    function void end_of_elaboration_phase(uvm_phase phase);
        //Setting necessary callbacks
        void'(dsAgent.inst.setCallback         (DENALI_......);
          ...
    endfunction
      ...
endclass
```

cdnUcieUvmUserFullStackEnv.sv

# Step 4: Bind

- This step will Bind the UCIe VIP Interface with the Agent path.
- All the screenshots involved in the configuration along with their description are shown in subsequent slides.

1. Configure VIP
2. Connect VIP Interface with DUT
3. Instantiate Interface with Testbench
4. Bind Agents and Interfaces

# System Verification Environment (1/2)

The binding steps perform the below actions in system verification environment.

- Binding of UCIe VIP agents to respective Interface in HDL Instances
- Setting agents to be Active and Enabling coverage
- Connect agents and their sequencers at virtual sequencer levels

1 Full Stack environment as Sub environment & Full Stack Virtual sequencer

2 Optional Type overrides

```
class cdnUcieUvmUserFullStackSve extends uvm_env;

  //Instantiate full stack environment and virtual sequencer
  cdnUcieUvmUserFullStackEnv                  env;
  cdnUcieUvmUserFullStackVirtualSequencer     virSeqr;

  function new(string name = "cdnUcieUvmUserFullStackSve", uvm_component parent);
      //Overwrite UVM Agent configuration with user components to give flexibility to user for modifications
      set_type_override_by_type(cdnUcieUvmUserConfigDsAgent::get_type(), cdnUcieUvmUserConfigPhyDsAgent::get_type());


      //Overwrite UVM components with user components to give flexibility to user for modifications
      set_type_override_by_type(cdnUcieUvmSequencer::get_type(),              cdnUcieUvmUserSequencer::get_type());
      set_type_override_by_type(cdnUcieUvmDriver::get_type(),                 cdnUcieUvmUserDriver::get_type());
      set_type_override_by_type(cdnUcieUvmMonitor::get_type(),                cdnUcieUvmUserMonitor::get_type());
      ...

      set_type_override_by_type(cdnUcieUvmUserVirtualSequenceBase::get_type(),cdnUcieUvmUserFullStackVirtualSequenceBase::get_type());
      set_type_override_by_type(cdnUcieUvmUserVirtualSequencer::get_type(),   cdnUcieUvmUserFullStackVirtualSequencer::get_type());
      ...
  endfunction
```

cdnUcieUvmUserFullStackSve.sv

# System Verification Environment (2/2)

```
virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    //Binding of UCIe Agents to respective Interface in HDL Instances
    uvm_config_db #(string)::set(this, "env.dsAgent          ",   "hdlPath", "testbench.dsAgent");          //DS Physical layer
    uvm_config_db #(string)::set(this, "env.adpDsAgent       ",   "hdlPath", "testbench.adpDsAgent");       //DS D2D Adapter layer
    uvm_config_db #(string)::set(this, "env.protoDsAgent     ",   "hdlPath", "testbench.protoDsAgent");     //DS Protocol layer
    uvm_config_db #(string)::set(this, "env.fullStackUsAgent",   "hdlPath", "testbench.fullStackUsAgent");//US full stack

    //Setting UCIe Agents to be Active
    uvm_config_db #(int)::set(this, "env.dsAgent",          "is_active", UVM_ACTIVE);
    . . .
    uvm_config_db #(int)::set(this, "env.fullStackUsAgent","is_active", UVM_ACTIVE);

    //Enabling coverage in UCIe Agents
    uvm_config_int::set(this, "env.dsAgent.monitor", "coverageEnable", 1);
    uvm_config_int::set(this, "env.adpDsAgent.monitor", "coverageEnable", 1);
    uvm_config_int::set(this, "env.protoDsAgent.monitor", "coverageEnable", 1);
    uvm_config_int::set(this, "env.fullStackUsAgent.monitor", "coverageEnable", 1);
    . . .
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    //connect environment agents to respective agents in virtual sequencer
    $cast(virSeqr.dsAgent, env.dsAgent);                          //DS Physical layer
    $cast(virSeqr.adpDsAgent, env.adpDsAgent);                    //DS D2D Adapter layer
    $cast(virSeqr.protoDsAgent, env.protoDsAgent);                //DS Protocol layer
    $cast(virSeqr.fullStackUsAgent, env.fullStackUsAgent);        //US full stack

    //connect environment sequencer to respective sequencers in virtual sequencer
    $cast(virSeqr.dsSeqr, env.dsAgent.sequencer);
    $cast(virSeqr.adpDsSeqr, env.adpDsAgent.sequencer);
    $cast(virSeqr.protoDsSeqr, env.protoDsAgent.sequencer);
    $cast(virSeqr.fullStackUsSeqr, env.fullStackUsAgent.sequencer);
endfunction
endclass : cdnUcieUvmUserFullStackSve
```

1. UCIe Agents associated with hdl path of testbench module instances
2. Set Agents to UVM_ACTIVE
3. Enable coverage
4. Env agents and sequencer mapping with Virtual sequencer

cdnUcieUvmUserFullStackSve.sv

# Agenda

Cadence Generic VIP Overview

Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

Getting started with Test sequences

Debug Aids

# Virtual Sequencer

1 Top Virtual Sequencer

2 Upstream and Downstream Agents instantiated here

3 Upstream and Downstream Sequencers instantiated here

```
class cdnUcieUvmUserFullStackVirtualSequencer extends uvm_sequencer;

  // ****************************************************************
  // Agent handles for all the instances
  // ****************************************************************

  cdnUcieUvmUserAgent    fullStackUsAgent;  // US Full Stack
  cdnUcieUvmUserAgent    dsAgent;           // DS Physical layer
  cdnUcieUvmUserAgent    adpDsAgent;        // DS D2D Adapter layer
  cdnUcieUvmUserAgent    protoDsAgent;      // DS Protocol layer

. . .
  // ****************************************************************
  // This driver-sequencers
  // ****************************************************************

  cdnUcieUvmSequencer fullStackUsSeqr ; // US Full Stack Sequencer
  cdnUcieUvmSequencer dsSeqr;          // DS Physical layer Sequencer
  cdnUcieUvmSequencer adpDsSeqr;        // DS D2D Adapter layer Sequencer
  cdnUcieUvmSequencer protoDsSeqr;      // DS Protocol layer Sequencer

. . .

endclass : cdnUcieUvmUserFullStackVirtualSequencer
```

# Virtual Sequence Writing (1/2)

1 Virtual Sequence

2 Get Downstream and Upstream Agent instances

3 Access UCIe Link Control Register to start UCIe Link Training

4 Set RAW Mode for flits

```
class cdnUcieUvmFullStackVirSeq extends cdnUcieUvmUserFullStackVirtualSequenceBase;

  // ****************************************************************
  // the driver sequence
  // ****************************************************************
  . . .
   virtual task body();

   cdnUcieUvmUserAgent fullStackUsAgent; // US Full Stack
   cdnUcieUvmUserAgent dsAgent;          // DS Physical layer
   cdnUcieUvmUserAgent adpDsAgent;       // DS D2D Adapter layer
   cdnUcieUvmUserAgent protoDsAgent;     // DS Protocol layer
   denaliUcieTransaction tr;

  . . .
  // Start UCIe link training is being written
    dsAgent.regInst.maskedWriteReg(DENALI_UCIE_REG_DVSEC_LINK_CONTROL,
         1 << DENALI_UCIE_Rpos__DVSEC_LINK_CONTROL_StartUCIeLinktraining,
         DENALI_UCIE_Rmask__DVSEC_LINK_CONTROL_StartUCIeLinktraining);

    fullStackUsAgent.regInst.maskedWriteReg(DENALI_UCIE_REG_DVSEC_LINK_CONTROL,
         1 << DENALI_UCIE_Rpos__DVSEC_LINK_CONTROL_StartUCIeLinktraining,
         DENALI_UCIE_Rmask__DVSEC_LINK_CONTROL_StartUCIeLinktraining);
  . . .
  // Writing Raw mode to be used for Param negotiation in adapter
    adpDsAgent.regInst.maskedWriteReg(DENALI_UCIE_REG_DVSEC_LINK_CONTROL,
        1 << DENALI_UCIE_Rpos__DVSEC_LINK_CONTROL_RawMode,
        DENALI_UCIE_Rmask__DVSEC_LINK_CONTROL_RawMode);

    fullStackUsAgent.regInst.maskedWriteReg(DENALI_UCIE_REG_DVSEC_LINK_CONTROL,
        1 << DENALI_UCIE_Rpos__DVSEC_LINK_CONTROL_RawMode,
        DENALI_UCIE_Rmask__DVSEC_LINK_CONTROL_RawMode);

   . . .
    endfunction
```

# Virtual Sequence Writing (2/2)

```
// Wait for LTSM
    fork
      fullStackUsAgent.waitForPhyLtsmState(DENALI_UCIE_LTSM_STATE_ACTIVE, 0);
      dsAgent.waitForPhyLtsmState(DENALI_UCIE_LTSM_STATE_ACTIVE, 0);
    join

    // Wait for RDI SSM
    fork
      adpDsAgent.waitForAdpRdiSsmState(DENALI_UCIE_SSM_STATE_Active);
      fullStackUsAgent.waitForAdpRdiSsmState(DENALI_UCIE_SSM_STATE_Active);
    join

    // Wait For Adp LSM
    fork
      adpDsAgent.waitForAdpLsmState(DENALI_UCIE_SSM_STATE_Active,0);
      fullStackUsAgent.waitForAdpLsmState(DENALI_UCIE_SSM_STATE_Active,0);
    join
    ...
     //Sending packet from Protocol Layer
   repeat(numPkt)
    begin
        `uvm_info(get_type_name(), "++ Generating Protocol mainband packet ++", UVM_LOW);
        `uvm_do_on_with ( tr, protoDsSeqr,{
           Type            == DENALI_UCIE_TYPE_Protocol;
           SubType         == DENALI_UCIE_SUBTYPE_Mainband;
           StreamId        == DENALI_UCIE_STREAM_ID_Stack0StreamingProtocol;
           MbPayload.size  == 64;
        })
    end
. . .
 endtask : body
endclass  //cdnUcieUvmFullStackSbVirSeq
```

1

Layer wise APIs for state machine changes

2

Send Mainband Protocol Streaming packets

3

Protocol Layer Sequencer part of Top Virtual Sequencer shown earlier

# Agenda

Cadence Generic VIP Overview

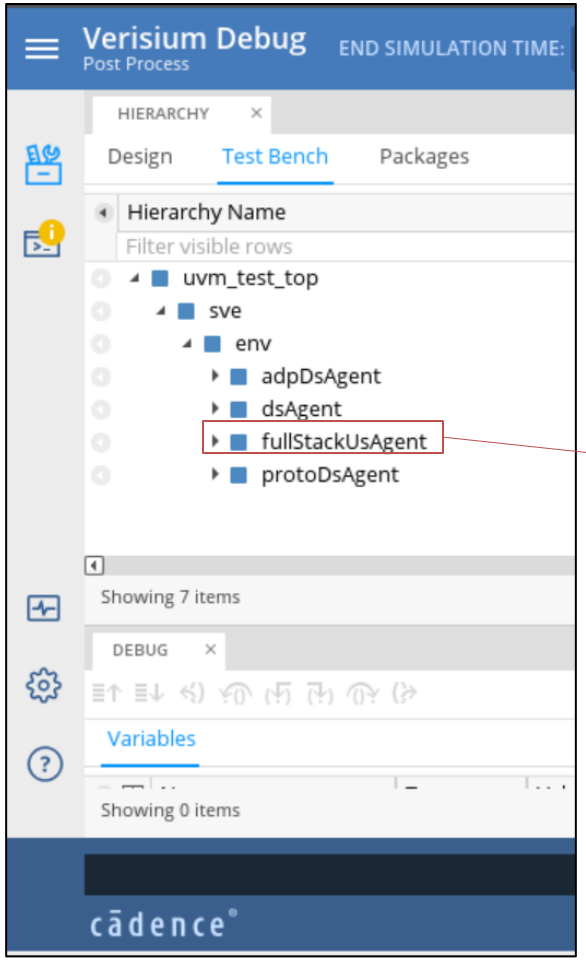
Cadence UCIe VIP Key Features & Architecture

Integrating Full Stack Environment

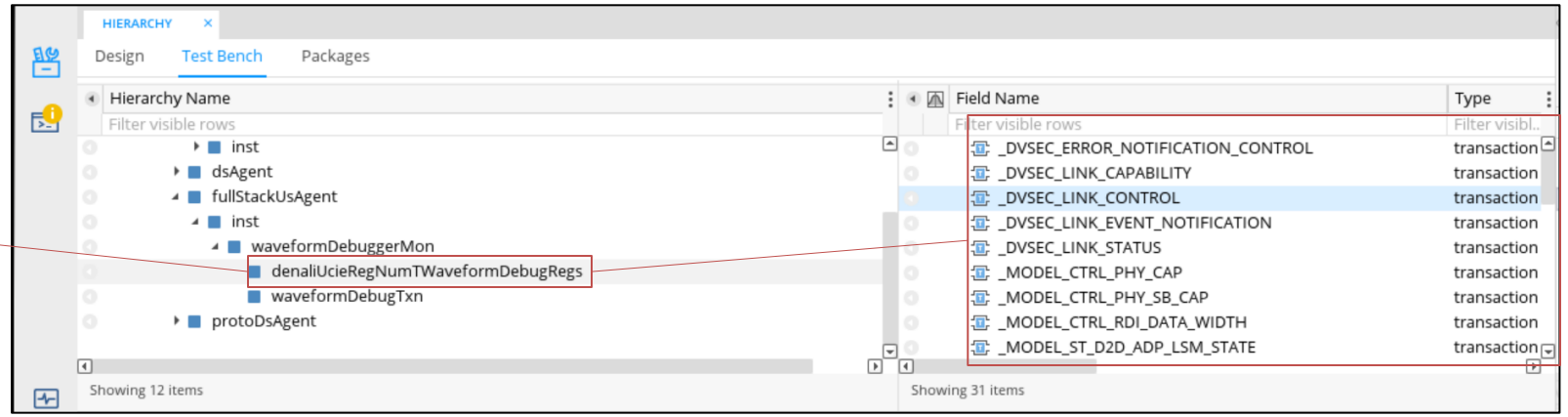
Getting started with Test sequences

Debug Aids

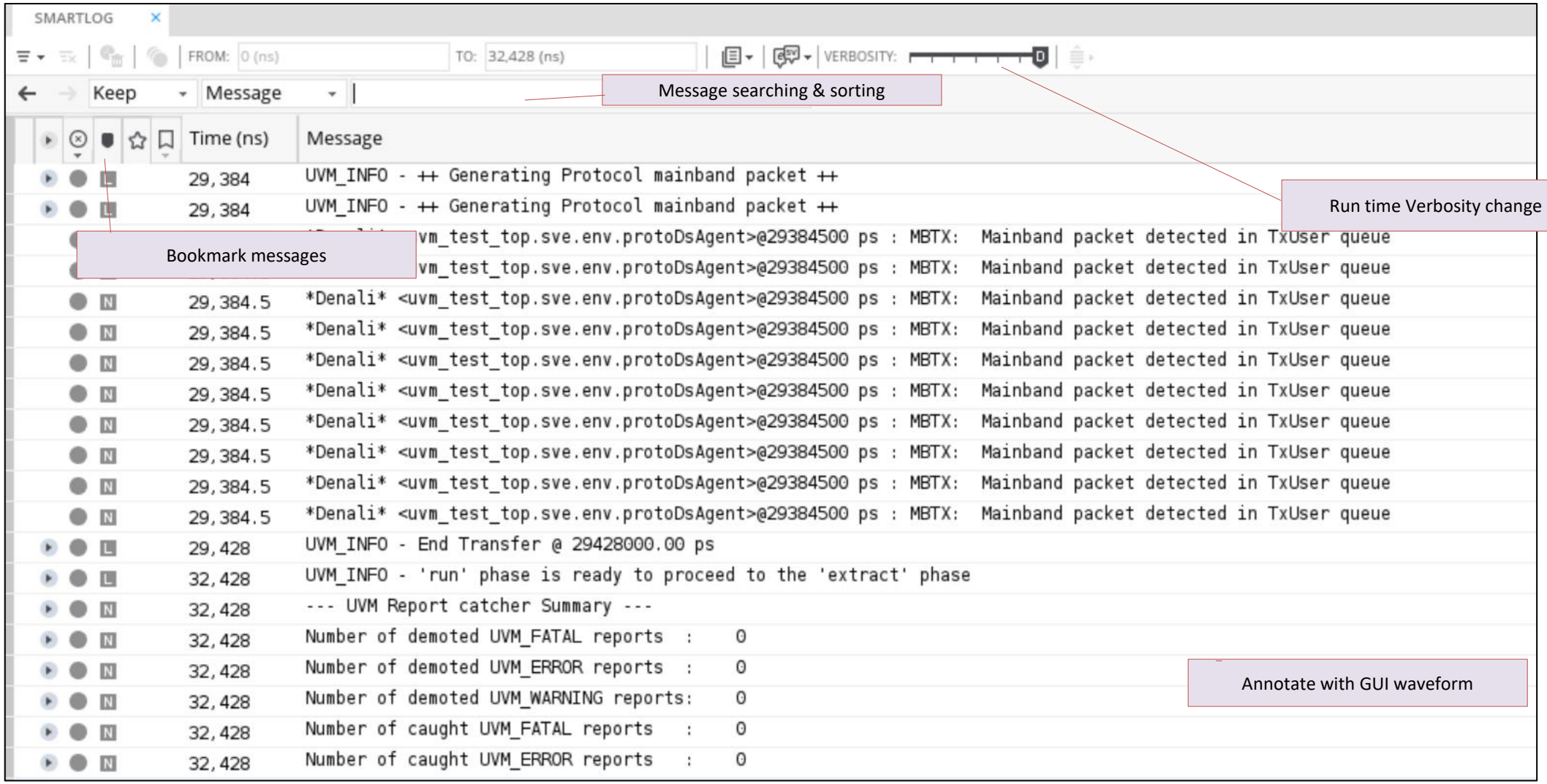
# Verisium Debug




- After running the simulation, open Verisium and,
- Load the database to find the hierarchical Testbench path



# Verisium Smart Log

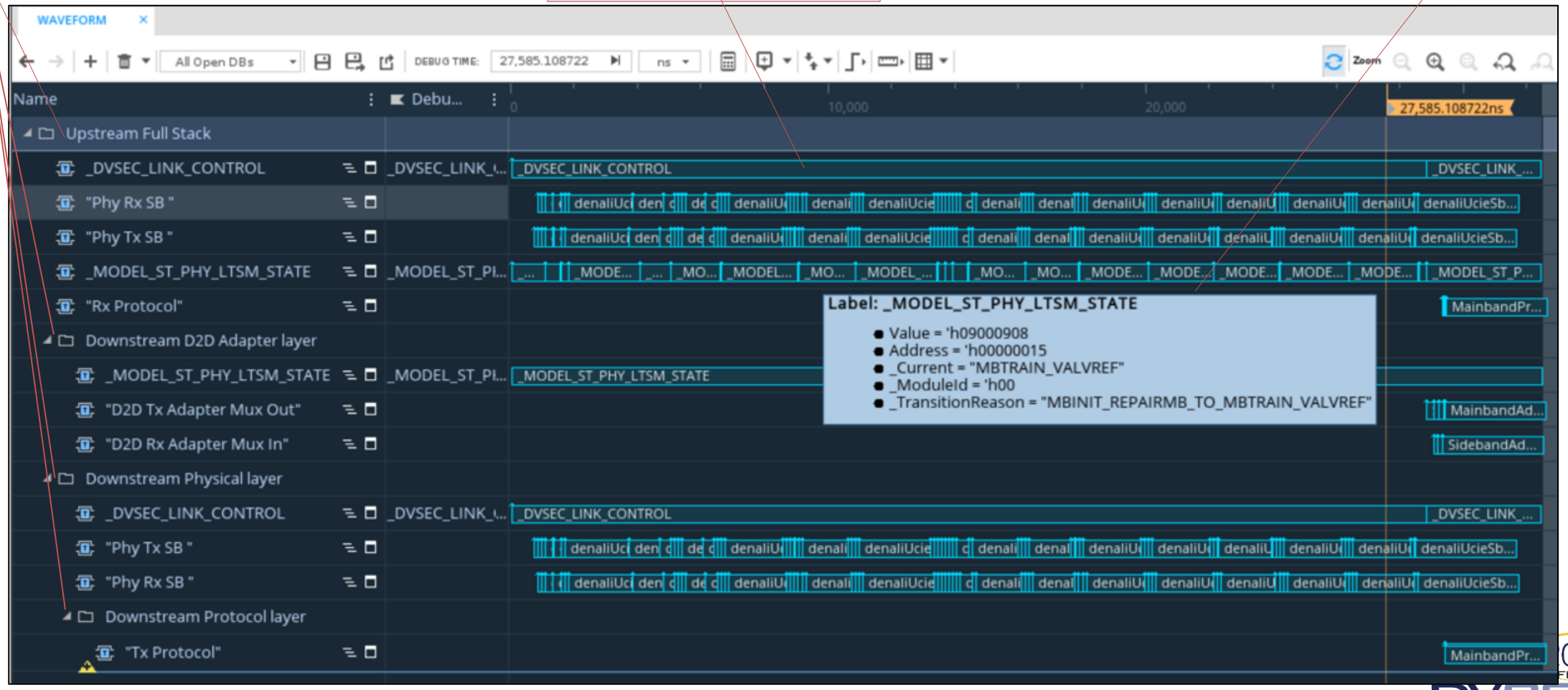


# Verisium Waveform Debugger View

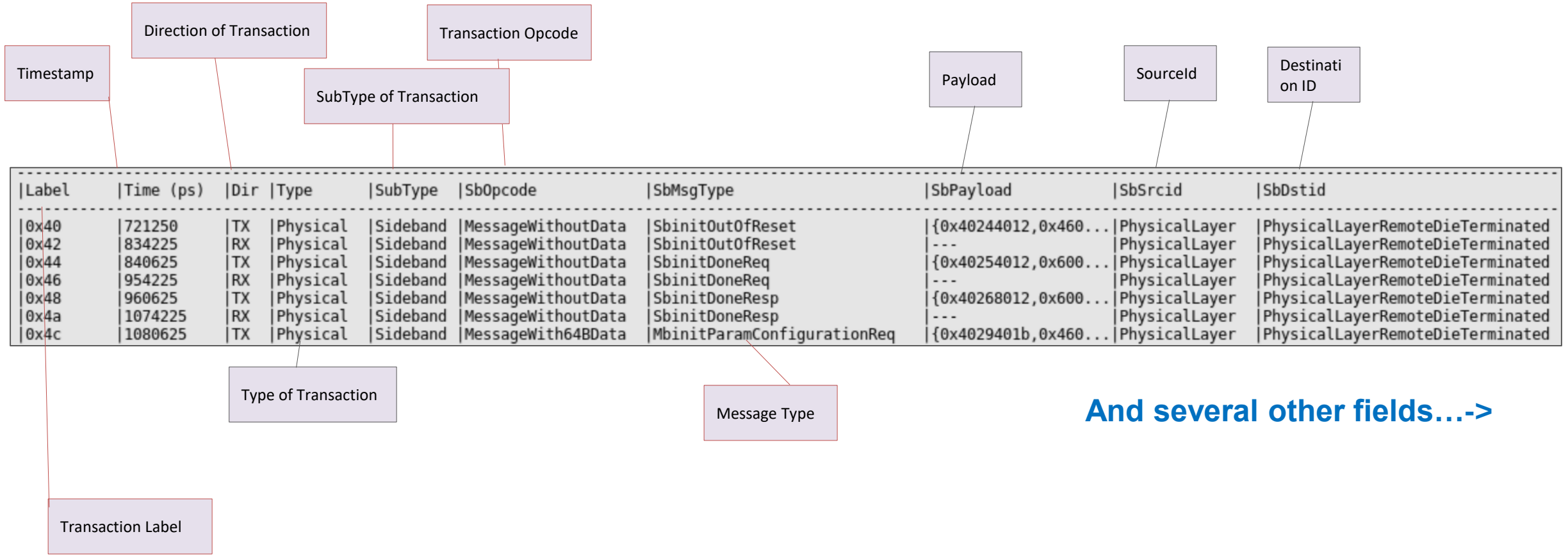
Upstream & Downstream agents in groups

GUI view
- Registers
- Packets
- State Machine Changes

Labels with detailed information



# Packet Tracker




| Label | Time (ps) | Dir | Type | SubType | SbOpcode | SbMsgType | SbPayload | SbSrcid | SbDstid |
|---|---|---|---|---|---|---|---|---|---|
| 0x40 | 721250 | TX | Physical | Sideband | MessageWithoutData | SbinitOutOfReset | {0x40244012,0x460... | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x42 | 834225 | RX | Physical | Sideband | MessageWithoutData | SbinitOutOfReset | --- | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x44 | 840625 | TX | Physical | Sideband | MessageWithoutData | SbinitDoneReq | {0x40254012,0x600... | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x46 | 954225 | RX | Physical | Sideband | MessageWithoutData | SbinitDoneReq | --- | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x48 | 960625 | TX | Physical | Sideband | MessageWithoutData | SbinitDoneResp | {0x40268012,0x600... | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x4a | 1074225 | RX | Physical | Sideband | MessageWithoutData | SbinitDoneResp | --- | PhysicalLayer | PhysicalLayerRemoteDieTerminated |
| 0x4c | 1080625 | TX | Physical | Sideband | MessageWith64BData | MbinitParamConfigurationReq | {0x4029401b,0x460... | PhysicalLayer | PhysicalLayerRemoteDieTerminated |

**And several other fields…->**

# Trace File – Deep debug

LTSM State Changes

Upstream & Downstream UVM Agents

```
M *Denali* <uvm_test_top.sve.env.dsAgent>@1250 ps : [Module0 TX] LTSM state change : RESET to SBINIT
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@1076250 ps : [Module0 TX] LTSM state change : SBINIT to MBINIT_PARAM
M *Denali* <uvm_test_top.sve.env.dsAgent>@1556250 ps : [Module0 TX] LTSM state change : MBINIT_PARAM to MBINIT_CAL
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@1796250 ps : [Module0 TX] LTSM state change : MBINIT_CAL to MBINIT_REPAIRCLK
M *Denali* <uvm_test_top.sve.env.dsAgent>@4046250 ps : [Module0 TX] LTSM state change : MBINIT_REPAIRCLK to MBINIT_REPAIRVAL
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@5016250 ps : [Module0 TX] LTSM state change : MBINIT_REPAIRVAL to MBINIT_REVERSALMB
M *Denali* <uvm_test_top.sve.env.dsAgent>@6602500 ps : [Module0 TX] LTSM state change : MBINIT_REVERSALMB to MBINIT_REPAIRMB
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@6602500 ps : [Module0 TX] LTSM state change : MBINIT_REVERSALMB to MBINIT_REPAIRMB
M *Denali* <uvm_test_top.sve.env.dsAgent>@9028750 ps : [Module0 TX] LTSM state change : MBINIT_REPAIRMB to MBTRAIN_VALVREF
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@9028750 ps : [Module0 TX] LTSM state change : MBINIT_REPAIRMB to MBTRAIN_VALVREF
M *Denali* <uvm_test_top.sve.env.dsAgent>@10838750 ps : [Module0 TX] LTSM state change : MBTRAIN_VALVREF to MBTRAIN_DATAVREF
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@10838750 ps : [Module0 TX] LTSM state change : MBTRAIN_VALVREF to MBTRAIN_DATAVREF
M *Denali* <uvm_test_top.sve.env.dsAgent>@13417500 ps : [Module0 TX] LTSM state change : MBTRAIN_DATAVREF to MBTRAIN_SPEEDIDLE
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@13417500 ps : [Module0 TX] LTSM state change : MBTRAIN_DATAVREF to MBTRAIN_SPEEDIDLE
M *Denali* <uvm_test_top.sve.env.dsAgent>@13657500 ps : [Module0 TX] LTSM state change : MBTRAIN_SPEEDIDLE to MBTRAIN_TXSELFCAL
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@13657500 ps : [Module0 TX] LTSM state change : MBTRAIN_SPEEDIDLE to MBTRAIN_TXSELFCAL
M *Denali* <uvm_test_top.sve.env.dsAgent>@13897500 ps : [Module0 TX] LTSM state change : MBTRAIN_TXSELFCAL to MBTRAIN_RXCLKCAL
...
...
...
M *Denali* <uvm_test_top.sve.env.dsAgent>@20048750 ps : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINCENTER1 to MBTRAIN_DATATRAINVREF
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@20048750 ps : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINCENTER1 to MBTRAIN_DATATRAINVREF
M *Denali* <uvm_test_top.sve.env.dsAgent>@22115 ns : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINVREF to MBTRAIN_RXDESKEW
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@22115 ns : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINVREF to MBTRAIN_RXDESKEW
M *Denali* <uvm_test_top.sve.env.dsAgent>@24181250 ps : [Module0 TX] LTSM state change : MBTRAIN_RXDESKEW to MBTRAIN_DATATRAINCENTER2
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@24181250 ps : [Module0 TX] LTSM state change : MBTRAIN_RXDESKEW to MBTRAIN_DATATRAINCENTER2
M *Denali* <uvm_test_top.sve.env.dsAgent>@26367500 ps : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINCENTER2 to MBTRAIN_LINKSPEED
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@26367500 ps : [Module0 TX] LTSM state change : MBTRAIN_DATATRAINCENTER2 to MBTRAIN_LINKSPEED
M *Denali* <uvm_test_top.sve.env.dsAgent>@28553750 ps : [Module0 TX] LTSM state change : MBTRAIN_LINKSPEED to LINKINIT
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@28553750 ps : [Module0 TX] LTSM state change : MBTRAIN_LINKSPEED to LINKINIT
M *Denali* <uvm_test_top.sve.env.dsAgent>@28795 ns : [Module0 TX] LTSM state change : LINKINIT to ACTIVE
M *Denali* <uvm_test_top.sve.env.fullStackUsAgent>@28795 ns : [Module0 TX] LTSM state change : LINKINIT to ACTIVE
```

# Demo

# Conclusion

- ✓ There is an Explosion of Design topologies
- ✓ Multiprotocol verification compounds verification complexity
- ✓ Need of LEGO-like verification components
- ✓ Need to build Scalable verification components to Subsystem/System level
- ✓ Prudent to start early verification before full design availability

---

## Contact Us!


https://www.cadence.com/en_US/home.html

Email:

anunay@cadence.com

sundara@cadence.com