Challenges in Design and Verification of PCI Express® Cores

Ashwin Matta

Denali Software Inc.

Target Audience

- PCI Express® core developers
  - ✓ Single core (RC, EP, Switch)
  - ✓ Multiple similar cores or multi-use cores
- PCI Express IP providers
  - ✓ Multiple customized cores
- PCI Express verification teams
- IP managers and re-use experts

Agenda

- PCI Express design customization space
- Design customization techniques
- PCI Express verification challenges and techniques
- Metrics: design customization and verification coverage
- Conclusion

PCIe® Design Customization Space

- Root complex/end point, dual-mode, switch
- Lane width, payload size, read request size, etc.
- Number of PCI functions (up to 64 for PCIe 2.0)
- Configurable RAM sizes for replay buffer, PNP receive FIFO, completion FIFO
- ECC for RAMs (in addition to parity checking)
- Host/client interface protocol, data width, frequency
- Pipe version, frequency and data width
- Interrupt support (Legacy INTx, MSI/MSI-X)

PCIe Design Customization Space (cont'd)

- ECRC generation and checking
- Quality of service support
  - ✓ Multi-channel and traffic classes
- IO Virtualization support
  - ✓ Access control services
  - ✓ Function level reset
- Clock frequency, low power states L1/L2
- FPGA support
- Additional features such as AER, lane reversal, external tags, etc.

Design Customization Types

- Type I: Bus width-based
  - ✓ Examples:
    - Host/client datawidth
- Type II: Structure/topology-based configurability
  - ✓ Examples:
    - RAM sizes for receive/completion FIFOs and replay buffer
    - Number of virtual functions
- Type III: Boolean Feature-based (code exists or doesn't exist)
  - ✓ Examples:
    - ECC for RAMs
    - IO virtualization support

Agenda

- PCI Express design customization space
- Design customization techniques
- PCI Express verification challenges and techniques
- Metrics: design customization and verification coverage
- Conclusion

Verilog-95 Customization Support

- `defines
  - ✓ Limited to supporting Customization Type I
  - ✓ Example: Customizable maximum payload size

```
`define MAX_PAYLOAD_SIZE 256
```

- Parameters
  - ✓ Supports Type I and Type II customization types
  - ✓ Example: Customizable Client Receive FIFO

```
module client_rx_fifo #(
        parameter ADDR_WIDTH=6,
                  WORD_COUNT=64)
        output [ADDR_WIDTH-1:0] fifo_ram_write_address;
        reg [7:0] control_ram[WORD_COUNT-1:0];
```

- Require source code re-compilation

Verilog-2001 Customization Support

- "genvar" and "generate" blocks
  - ✓ Supports Type II customization
  - ✓ Example: Support for 64 virtual functions

```
genvar vf_instance;
generate for (vf_instance = 0; vf_instance < `MAX_PCI_FUNCTIONS; vf_instance = vf_instance +1)
    begin : pci_command_status_reg_vf_inst
    pci_command_status_reg_vf #(vf_instance) pci_command_status_reg_vf_mod
    (.clock            (clock),
     .reset            (reset),
     .hot_reset        (hot_reset),
                …
                …
     .master_data_parity_error(master_data_parity_error[vf_instance]),
    .compl_abort_sent(compl_abort_sent[vf_instance]),
    .target_abort_received(target_abort_received[vf_instance]),
    .master_abort_received(master_abort_received[vf_instance]),
    .signaled_system_error(signaled_system_error[vf_instance]),
    .detected_parity_error(detected_parity_error[vf_instance])
    end
endgenerate
```

SystemVerilog Techniques for Design Customization

- Method/architecture for a sum-of-all-features PCI Express core design called “Sigma Core”
  - ✓ Includes all interface types, advanced features
  - ✓ Sigma core is largest PCI Express core possible
    - User-specified fixed Type I, II customization values (bus widths, FIFO depths, etc.)
    - All Type III features included in single compilable source (even mutually exclusive features)
      - Configurability designed in using SystemVerilog
  - ✓ Special design architecture constraints

SystemVerilog Techniques for Design Customization (cont'd)

- Sigma core verified completely in SystemVerilog verification environment
  - ✓ Tremendous verification efficiency due to single RTL source
  - ✓ Randomization of PCI Express customization space
- Internal tool called 'Slimfast' to reduce the Sigma core into a subset that can be manufactured
  - ✓ Manufactured core MUST also be verified with the same environment

SystemVerilog: A Few New Keywords

- SV adds the following keywords to help you organize how things are connected on the RTL side
  - ✓ “typedef”: Just like the C “typedef”. This allows you to rename more complicated definitions.
  - ✓ “interface”:This allows you to group a bundle of signals, but also declare different “views” into the bundle, as well as behavioral code.
  - ✓ “package”: A collection of classes, structs, interfaces, functions, etc. that can be referenced using “::” operator
  - ✓ “user-defined attributes”: A user-defined value ignored by the simulator but usable by user’s own tools

Using SystemVerilog Interfaces

```
// Declaring 'interface'
interface denaliPCIE_top_if();
   // Clk
   logic CORE_CLK;
   logic AUX_CLK;
   // MSI Interrupt Interface
   logic [31:0] MSI_ASSERT_INT = 0;
   logic [31:0] MSI_CLEAR_INT = 0;
   logic [1:0] MSI_ENABLE;
   logic [2:0]  Fn_MSI_VECTOR_COUNT[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic [63:0] Fn_MSI_MSG_ADDRESS[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic [15:0] Fn_MSI_MSG_DATA[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic MSI_MSG_SENT;
   modport msi_interrupt_mp(output MSI_ASSERT_INT, MSI_CLEAR_INT, input
                MSI_ENABLE, Fn_MSI_VECTOR_COUNT, Fn_MSI_MSG_ADDRESS,
                Fn_MSI_MSG_DATA, MSI_MSG_SENT);
   …
   …
endinterface: denaliPCIE_top_if

module denaliPCIE_top(denaliPCIE_top_if top_if);
   always@(posedge top_if.CORE_CLK)
      if (top_if.MSI_ENABLE) …
endmodule:denaliPCIE_top
```

Customization with SystemVerilog Attributes

```
interface denaliPCIE_top_if();
   // Clk
   logic CORE_CLK;
   (* AUX_POWER_SUPPORT *) logic AUX_CLK;
   // MSI Interrupt Interface
   logic [31:0] MSI_ASSERT_INT = 0;
   logic [31:0] MSI_CLEAR_INT = 0;
   logic [1:0] MSI_ENABLE;
   logic [2:0]  Fn_MSI_VECTOR_COUNT[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic [63:0] Fn_MSI_MSG_ADDRESS[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic [15:0] Fn_MSI_MSG_DATA[`DENALIPCIE_NUM_PCI_FUNCTIONS-1:0];
   logic MSI_MSG_SENT;
   modport msi_interrupt_mp(output MSI_ASSERT_INT, MSI_CLEAR_INT, input
                MSI_ENABLE, Fn_MSI_VECTOR_COUNT, Fn_MSI_MSG_ADDRESS,
                Fn_MSI_MSG_DATA, MSI_MSG_SENT);
   …
   …
   (* HAL_TARGET_IO_SUPPORT *) modport hal_target_io(output HAL_IO_REQ, ….);
endinterface: denaliPCIE_top_if
```

- Attributes define conditions under which feature/signal/module exists or does not exist **(Type III Customization Type)**
- Attributes are ignored by RTL simulators but processed by internal Slimfast tool

Module Configurability Using Attributes

- Rules:
  - ✓ Use an attribute when instantiating modules that may be removed if a feature does not exist
  - ✓ Pay special attention to mutually exclusive module instantiations
- Example:

```
module fifo_mux();
….
…
// Instantiation of feature-based mutually exclusive FIFOs
   (* CfgFAST *) fast_fifo fifo ();
   (* NotCfgFAST *) norm_fifo fifo ();

endmodule:fifo_mux
```

SystemVerilog Configurability Package

- Customization SV class
- Constant functions
  - ✓ Example:

```
function integer getLog2(integer count);
    reg [pkg::getLog2(`FIFO_DEPTH)-1:0] head, tail;
```

- Type definitions
  - ✓ Example: typedef enum { RX_IDLE , RX_DATA } rx_hal_state
- “Exists” function to identify if a feature is supported or not

SystemVerilog Customization Class Contents

- Hash table for feature array
  - ✓ Example: feature[“AUX_POWER_SUPPORT”] = 1;
- Method to read user-specified customization information (in XML format) to populate the feature hash table
- Constraints on randomizing the customization space
  - ✓ Example:

    constraint c_type { ((core_type == END_POINT) | (core_type == ROOT_COMPLEX) | (core_type == DUAL_MODE));}
  - ✓ Allows randomization of customization space

Type III Customizability using "EXISTS" Function

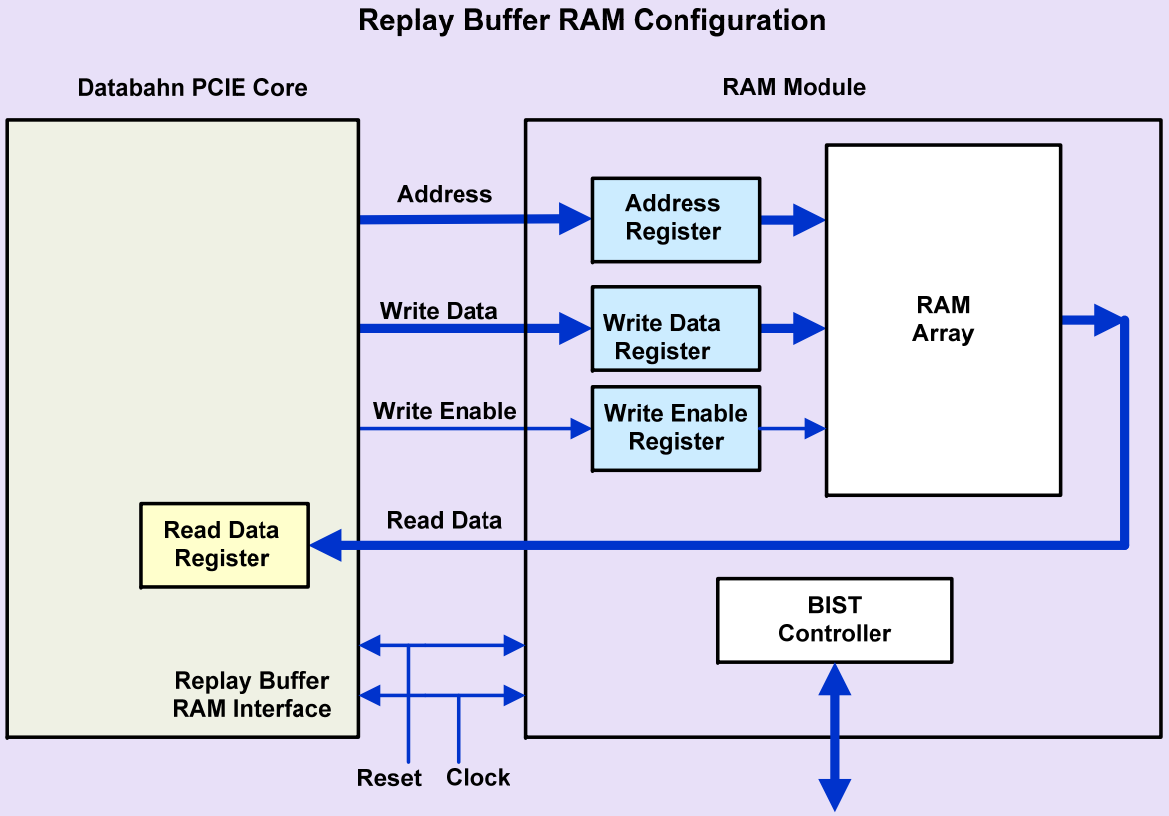
- EXISTS function provides mechanism for dynamic muxing of Type III features
- "EXISTS" function declaration:
  - ✓ Example:

```
function logic EXISTS (string index, input true_val, false_val);
    return (feature.exists(index) ? true_val : false_val); // Hash lookup
endfunction:EXISTS
```

- "EXISTS" function usage:
  - ✓ Example:

```
always @( posedge clk ) begin
   dataout <= pkg::EXISTS("PILL", datain_happy, datain_grumpy);
end
```

Miscellaneous Customization



- RAM module extracted from PCI Express core
  - ✓ Customizable size and implementation
  - ✓ Customizable BIST and ECC

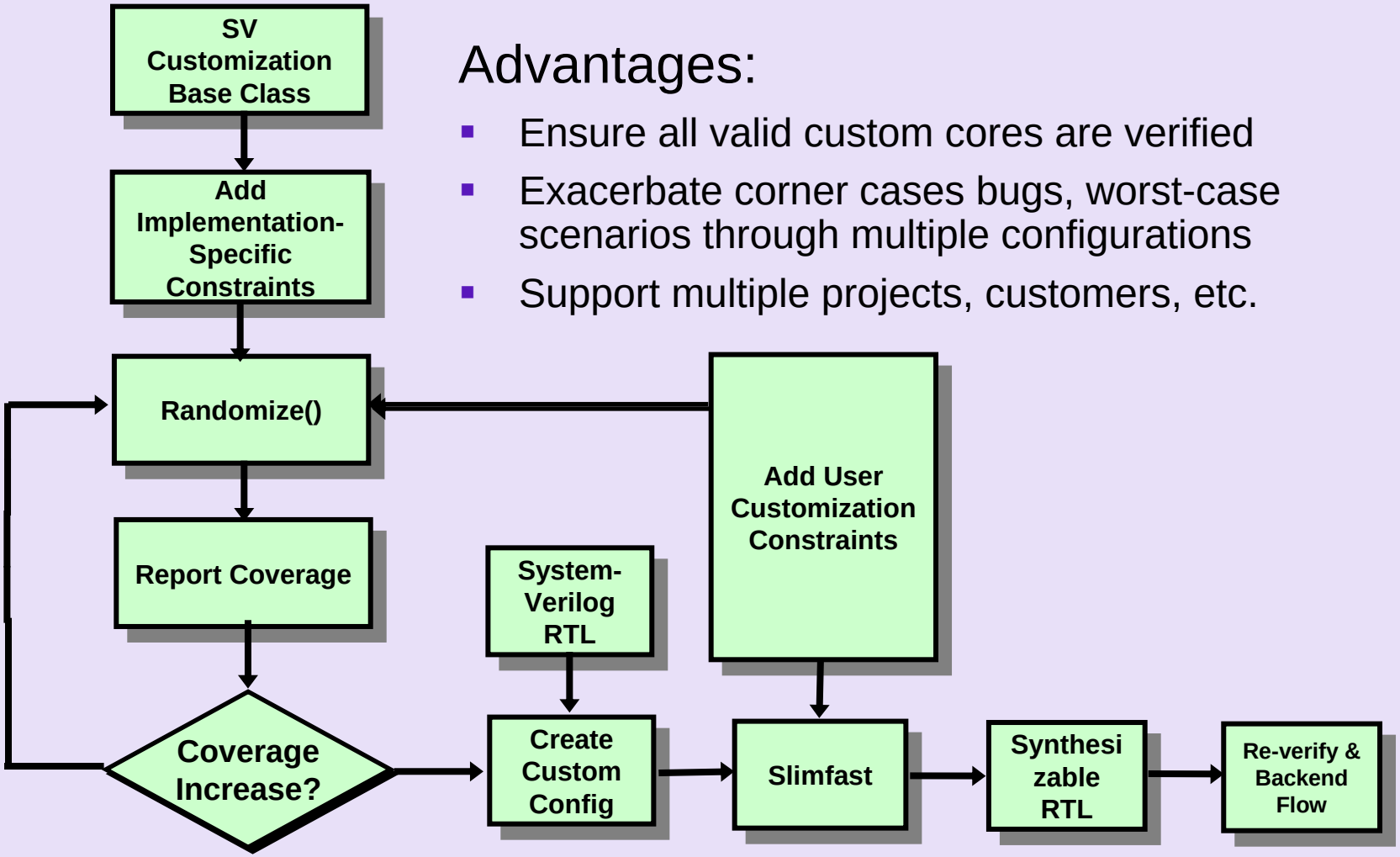
Design/Verification Flow

- All code snippets shown are perfectly legal, although non-synthesizable SystemVerilog code
- Simulate and verify Sigma core represented in SystemVerilog, randomizing configuration as needed
  - ✓ Verification environment uses the same packages used by design
    - No restrictions on usage of SystemVerilog constructs
  - ✓ Verification environment needs to be designed intelligently to dynamically connect components to test randomized configuration
- When ready, use Slimfast for generating synthesizable RTL

Slimfast Operation

- When design is ready for backend flow, Slimfast does the following:
  - ✓ Performs configurability syntax checking to ensure that configurability rules for packages, interfaces, generate-endgenerate statements, etc. have been adhered to
  - ✓ Evaluates constant function values based on specified parameters or defines (unless the function is synthesizable)
  - ✓ Strips out all attributes from interfaces, removes unwanted modules and ports and their connections
  - ✓ Outputs clean, synthesizable RTL ready for full re-verification and backend flow

Customizable Design Flow



Advantages:

- Ensure all valid custom cores are verified
- Exacerbate corner cases bugs, worst-case scenarios through multiple configurations
- Support multiple projects, customers, etc.

Other Options for Customizable Code

- Configure RTL code using a templating language
  - ✓ Template toolkit (http://template-toolkit.org/)
- Advantages
  - ✓ Easy to get started, no special tools required
- Disadvantages
  - ✓ Difficult to read RTL due to inserted templating directives
  - ✓ Difficult to verify all valid configurations

Agenda

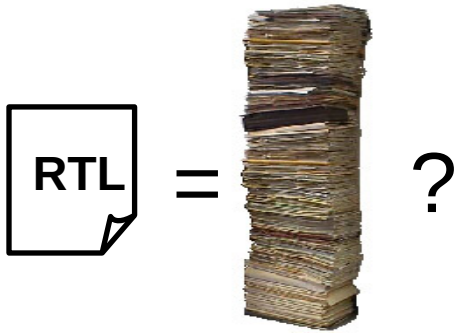
- PCI Express design customization space
- Design customization techniques
- PCI Express verification challenges and techniques
- Metrics: Design customization and verification coverage
- Conclusion

Market Risks for All PCIe-Based Chips

- Chips come out, but don't talk to each other
  - ✓ Due to complexity: serial, new protocol, switch fabric …

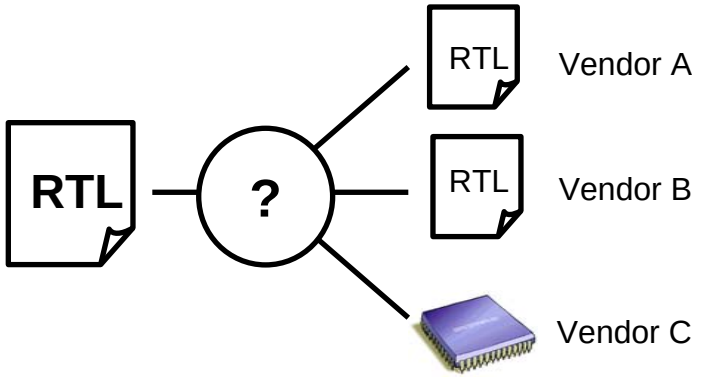
Compliance

– a measure of adherence to the specification



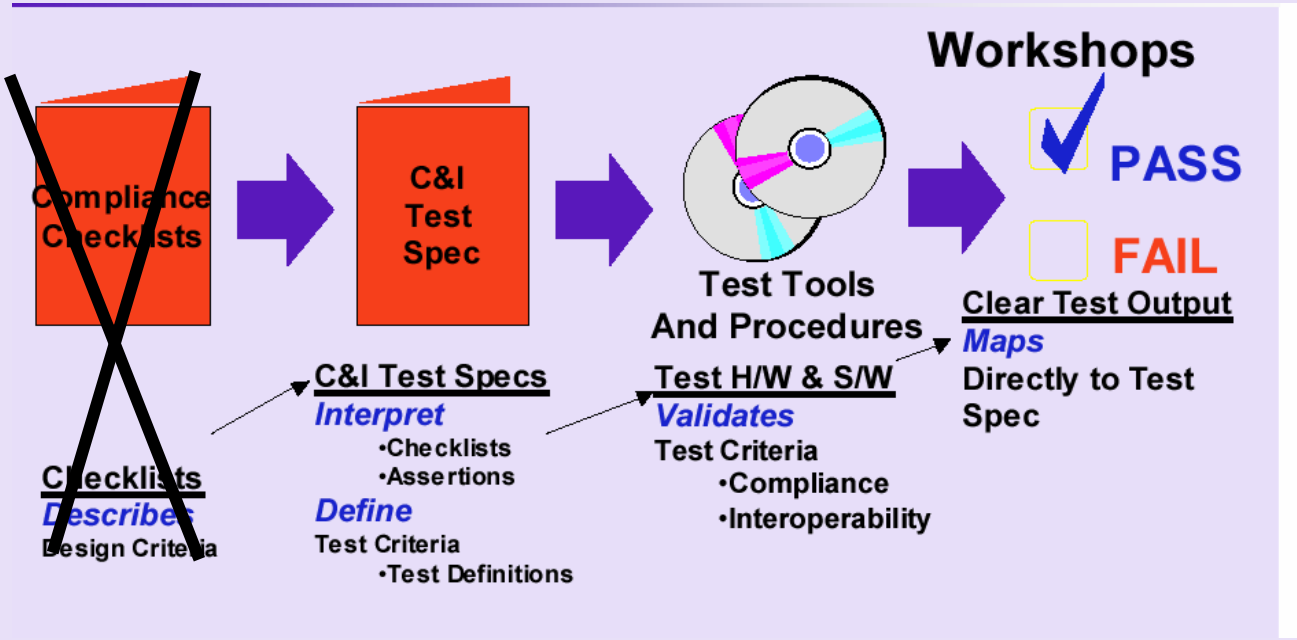
Interoperability

– a measure of compatibility with other systems



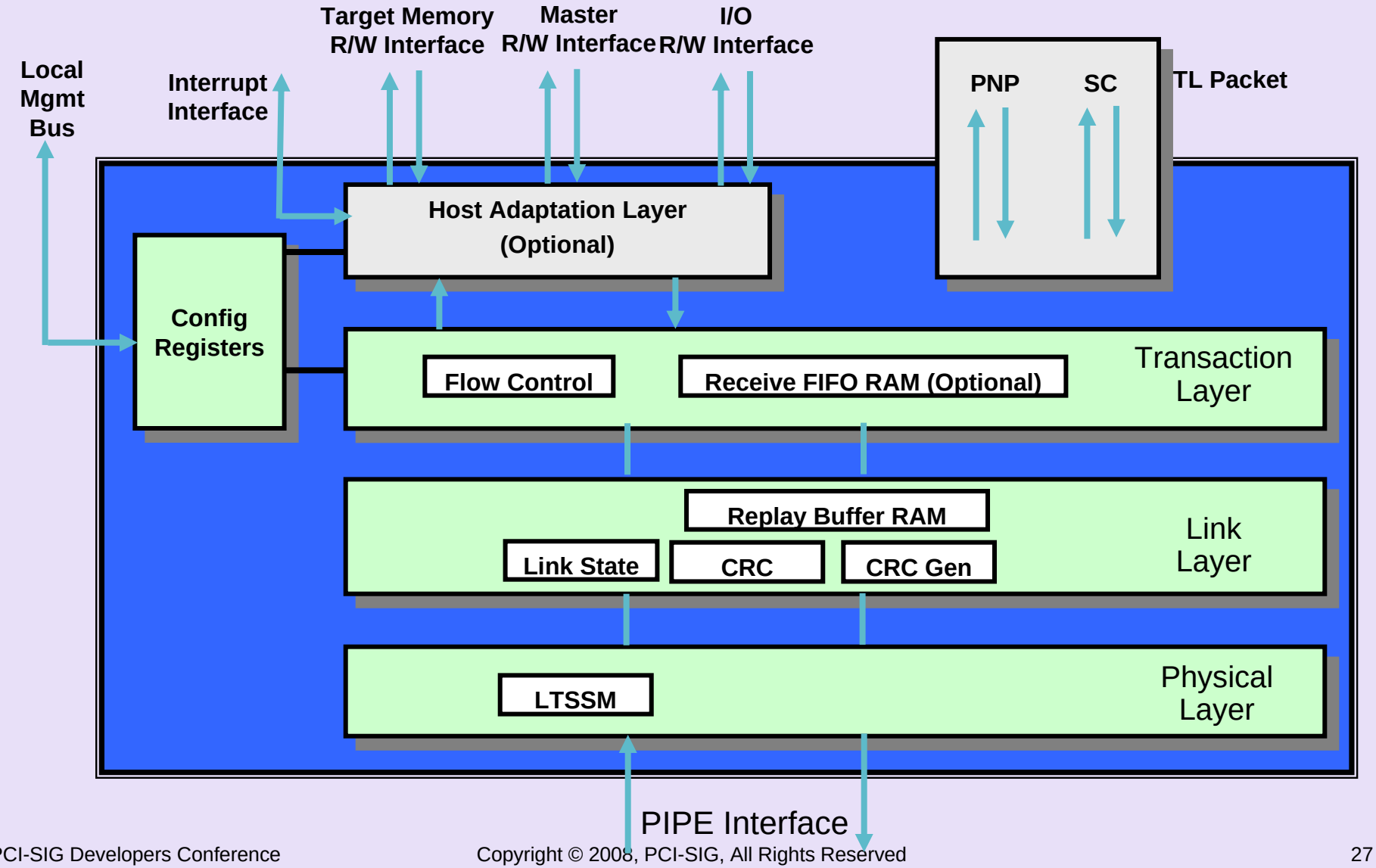
Guaranteeing Compliance and Interoperability

- Currently, there is an infrastructure/process in place to tackle post-silicon interoperability



- Pre-Silicon verification requires strong verification methodology

High-Level Architecture for Sigma PCI Express Core



Pre-Silicon Verification Strategy

- Collect specifications and understand ecosystem limitations
- Build and review a coverage-driven verification testplan
- Architect your verification environment
- Perform block and system-level verification
  - ✓ Facilitate driving application traffic and scenarios
- Analyze code and functional coverage of design
  - ✓ Close-loop with the testplan

Verification Requirements

- Comprehensive testplan matching DUT specifications
- Robust and comprehensive compliance testsuite
  - ✓ Example: PCI-SIG PCI Express Compliance testlist
- Pseudo-random verification environment using an advanced verification methodology (AVM/OVM, VMM, eRM, etc.)
- Industry-standard PCIe Verification IP
  - ✓ Comprehensive assertion library
  - ✓ Compatible with your simulator and verification methodology
  - ✓ Proven in multiple designs

What Constitutes the Testplan

- Feature from design spec being tested
- SOMA feature under which feature is applicable (eg. Max_Payload_Size)
- Method/description how this feature is tested
  - ✓ Directed test sequence
  - ✓ Pseudo-random test
  - ✓ Assertions (OVL/SVA)
  - ✓ Testbench infrastructure (scoreboard/monitor)
- Link/tag connecting feature to actual implementation
- Support fields (description, owner, etc.)
- Coverage Status – ideally back-annotated from your measured functional coverage

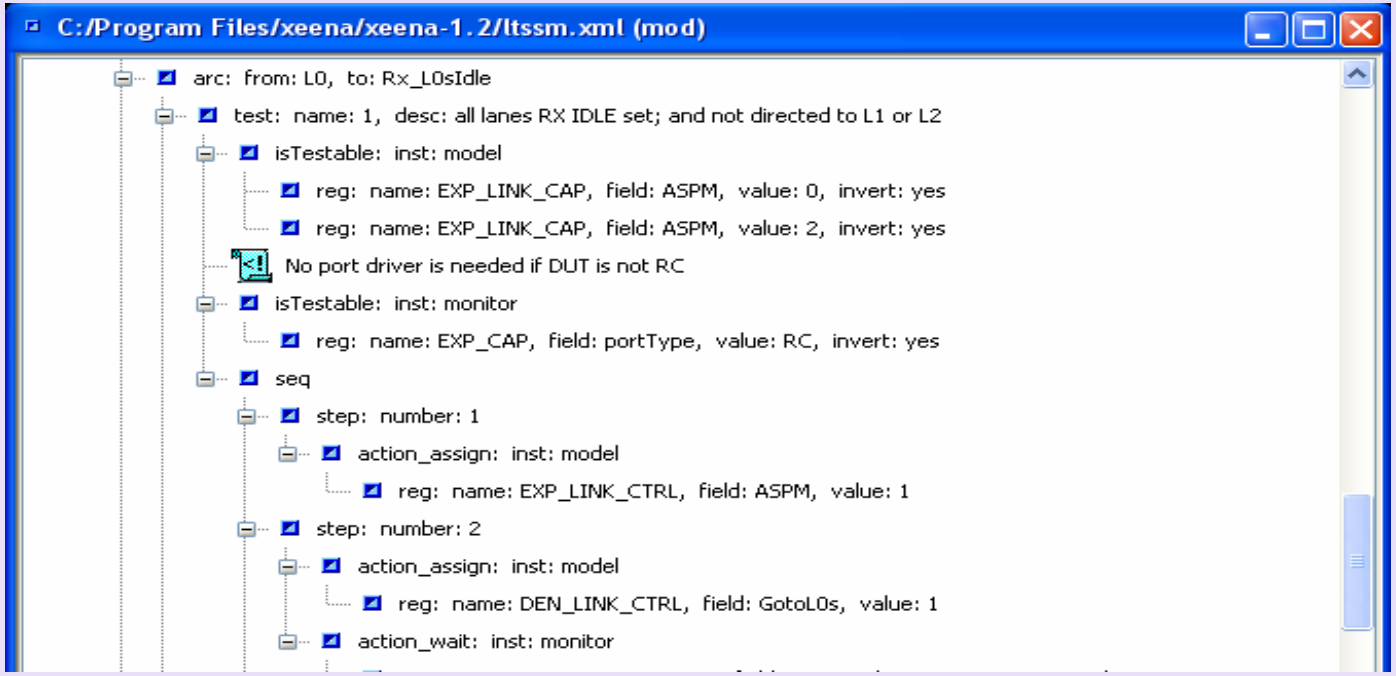
Sigma PCI Express Core Verification Testplan Example

Table of Contents

- Testplan source in XML DocBook Format
  - ✓ Easily configurable
- Testplan can be imported inside UCDB (Universal Coverage Database)
  - ✓ Easy to back-annotate measured coverage with testplan

PCI Express Compliance Suite

- Test purpose and assumptions
- Scenario and criteria to pass or fail
- Expected result and the related error messages
- PCI-SIG® checklist number (if applicable)

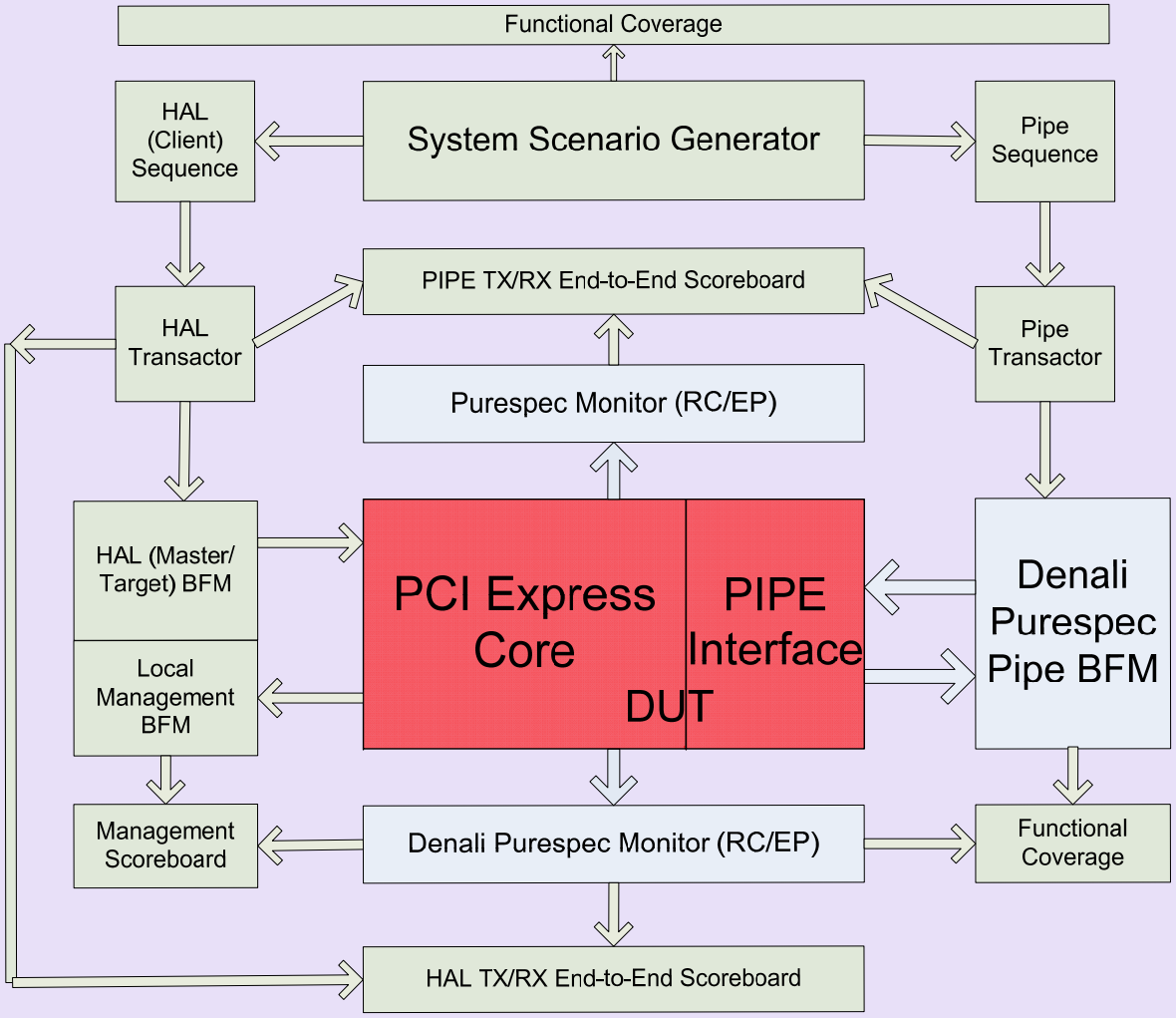


Intelligent Assertion Library

- PCI Express requires hundreds of configuration and runtime checks
  - ✓ Correlated to Specification
- Assertions coverage reporting

Checklist Item	Description	Denali Puresuite Error ID
DLL.5.3#2	If a normal TLP (one with END framing symbol) is received and its LCRC doesn't match calculated CRC, discard the TLP, free any storage associated with it, schedule a NAK DLLP for transmission if one is not already scheduled and report an error associated	PCIE_DL_NONFATAL_NAK_LATE, PCIE_DL_NONFATAL_UNEXP_PKT, PCIE_TL_NONFATAL_TLP_COR_ERR_MISMATCH, PCIE_TL_NONFATAL_TLP_FATAL_ERR_MISMATCH, PCIE_TL_NONFATAL_TLP_NONFATAL_ERR_MISMATCH
DLL.5.3#3.1	If the sequence number doesn't match with expected value and if the sequence number is within the last 2048 TLPs outstanding, this is a duplicate TLP and an ACK DLLP must be scheduled.	PCIE_DL_NONFATAL_ACK_LATE
DLL.5.3#3.2	If the sequence number doesn't match with expected value and if the sequence number is not within the last 2048 TLPs, schedule a NAK DLLP for transmission if one is not already scheduled, report TLP missing and report an error associated with the Port.	PCIE_DL_NONFATAL_NAK_LATE

Psuedo-Random PCI Express Verification Environment Architecture



Agenda

- PCI Express design customization space
- Design customization techniques
- PCI Express verification challenges and techniques
- Metrics: design customization and verification coverage
- Conclusion

Measuring Functional and Assertion Coverage

Sigma PCI Express Core	Num Cover-groups	Num Cover-points	Num Coverage Bins	Coverage Bin Examples
RX Side (from PIPE Interface)	11	25	~700	bins PollingActiveToDetect = (PCIE_LTSSM_STATE_PollingActive => PCIE_LTSSM_STATE_DetectEntry); bins CompletionStatusSuccess; bins CompletionStatusCfgRetry;
TX Side (from client Interface)	15	40	~1300	bins HALMasterReadyWaitLow; bins HALTargetAddr32 [16] = { [$: 'hffffffff] }; bins RetryBufferFull;
Configuration and Programmability	2	8	~200	bins LaneReversal; bins Gen1_Gen2_Switch; bins LaneGoingDown;

- Total functional coverage bins = ~2200
- Total number of assertions = ~1700

PCIe Configurability Scope

- Total number of customizable options = 47
  - ✓ Type I = 17
  - ✓ Type 2 = 9
  - ✓ Type 3 = 21
- Total number of functional coverage bins for customizability class = ~50
- Total number of unique PCI Express cores = >1000
- Total number of unique Type2/Type3 configurations for 100% customization space coverage = ~60

Closing the Loop with Coverage

1. Ensure adequate code coverage
2. Back-annotate measured functional coverage data with testplan
3. Review coverage reports
   - ✓ Sign-off on uncovered bins
4. Repeat for each custom configuration

Agenda

- PCI Express design customization space
- Design customization techniques
- PCI Express verification challenges and techniques
- Metrics: design customization and verification coverage
- Conclusion

Conclusion

- PCI Express offers plethora of challenges for designing customizable cores
- Complexity of protocol and features demand sophisticated design and verification techniques
- SystemVerilog-based solutions address both customization and verification needs

Thank you for attending the
PCI-SIG Developers Conference 2008

For more information please go to
www.pcisig.com