Formal Verification for PCIe® 1.1 and 2.0 RTL designs

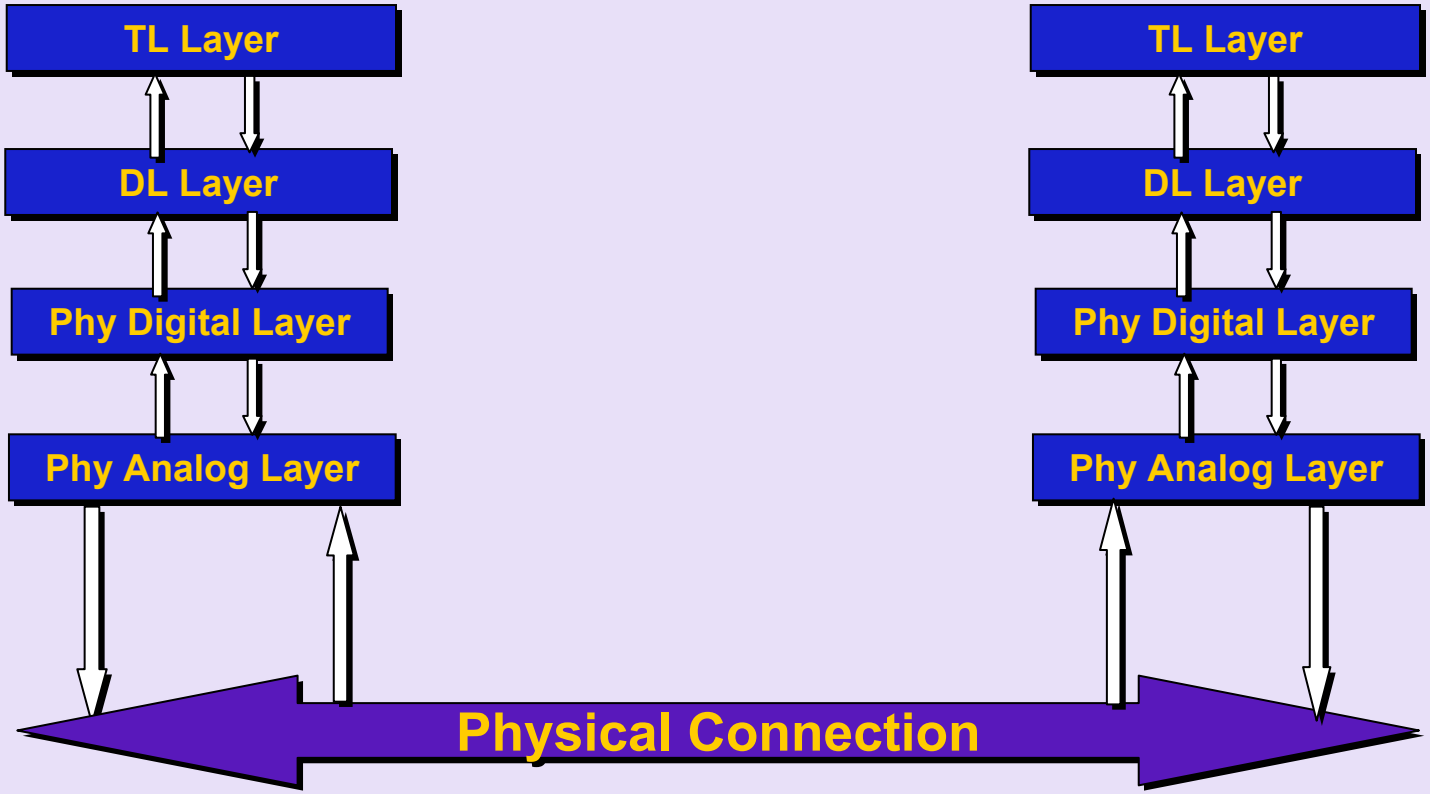
Vigyan Singhal

Oski Technology

Agenda

- PCIe architecture
  - ✓ Where formal applies, where not
- Formal verification introduction
  - ✓ Use model
  - ✓ Challenges
- Formal testplanning
  - ✓ Project planning
- Formal applied to PCIe RTL design
  - ✓ Strategies for different blocks
- Example (Transaction Layer)
  - ✓ Properties
  - ✓ Verification strategies
- PCIe as internal or 3rd party IP

PCIe Functional Layers



PCIe Complexity

- Concurrency
  - ✓ Interacting components
  - ✓ Synchronization
  - ✓ Arbitration
  - ✓ Long state machine traversal times
- Packet transfer
  - ✓ Packet ordering
  - ✓ Data integrity
  - ✓ Retry transmissions
  - ✓ Packet framing/encodings

⇒ **Lots of corner cases**

What's covered

- Functional Verification
  - ✓ Block interface checks
  - ✓ Protocol requirements
  - ✓ Data integrity checks
  - ✓ Flow Control checks
  - ✓ Ordering checks

What's not covered

- Timing Verification
  - ✓ Hold/setup time violations
  - ✓ Synchronization across clock domains
- Analog PHY verification
  - ✓ Jitter, BER requirements
  - ✓ …
- Performance verification
  - ✓ Statistical throughput

Software Simulation

- Traditional method for verification
- Easy to get started with pre-existing bus functional models
- Use model is familiar
- Coverage goals (structural and functional) serve as metrics for completeness

- Not complete
  - ✓ design with N inputs and M flops, needs ((2**M)**N) distinct input vectors for complete coverage
  - ✓ With design concurrency, opportunities to miss corner-case bugs
  - ✓ Coverage is only an approximate measure of completeness – 100% coverage does not mean no bugs

Formal Verification

- New radical approach to verification
- New use model
- Completeness metrics are evolving
  - ✓ Property coverage, proof radius
- Need new implementation of properties
  - ✓ SVA or OVL
- Formal sometimes blows up
  - ✓ Especially for data transform designs

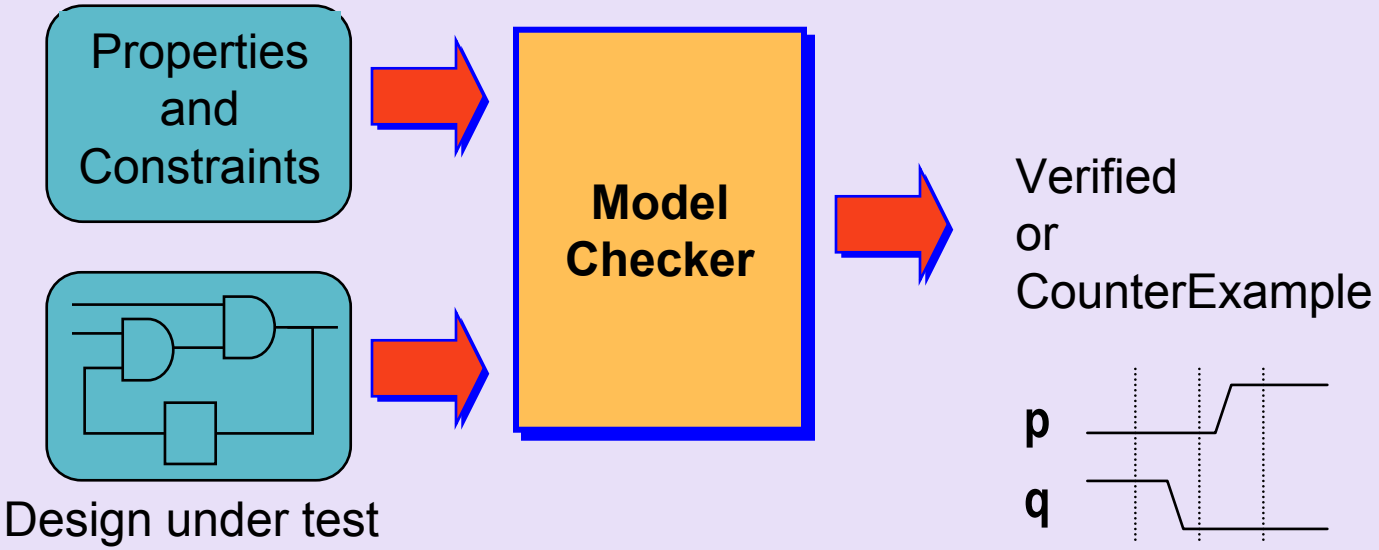
- If formal works, completeness is achieved

Formal Verification

- Example corner-case bugs
  - ✓ Last DW of 1024-DW Memory Write packet arriving on VC1, from Application Layer to Transaction Layer
  - ✓ On same cycle, TL wants to send an Unsupported Request Completion
  - ✓ On same cycle, DLL Replay Buffer issues a wait cycle
  - ✓ DUT (Transaction Layer) overwrites last DW of Memory Write packet with Header of Completion
- Rare combination of events: unlikely to be caught with simulation
- Ideal for formal: through data integrity property

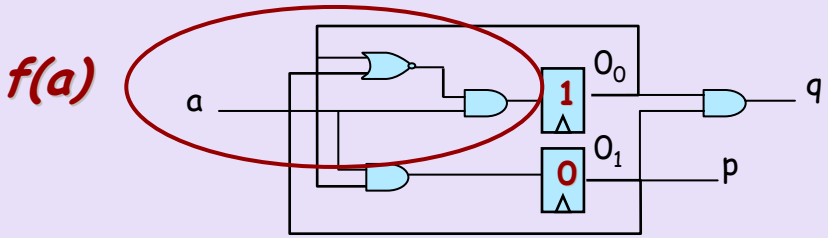
Formal verification

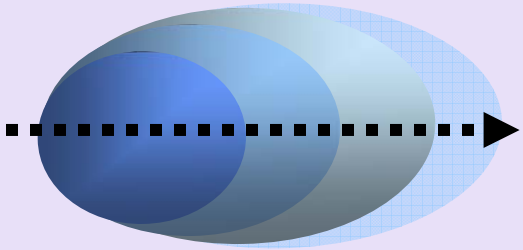
Does the design have a given desirable property?

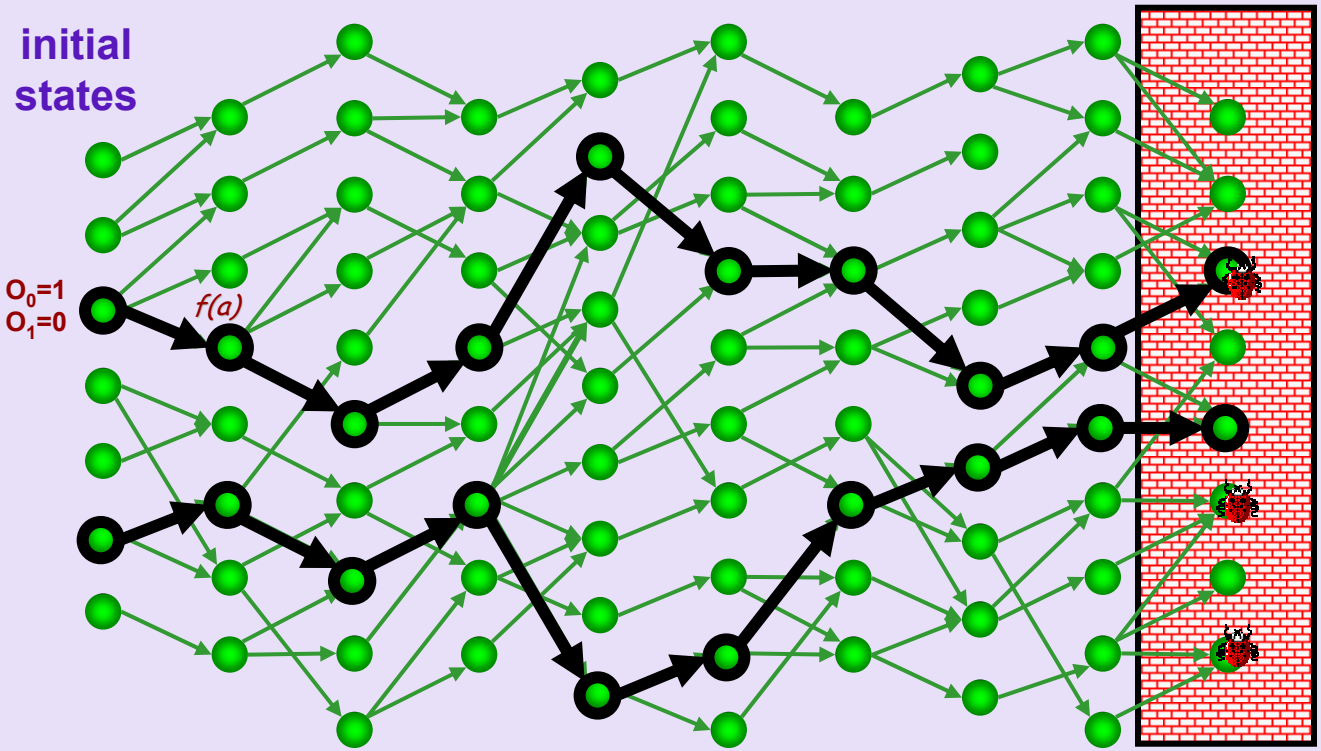


- Properties - Specification
- Constraints - Assumptions needed to prove Properties
  - ✓Properties and Constraints written in Assertion Languages
    - e.g. SVA or PSL

How is formal different from simulation?



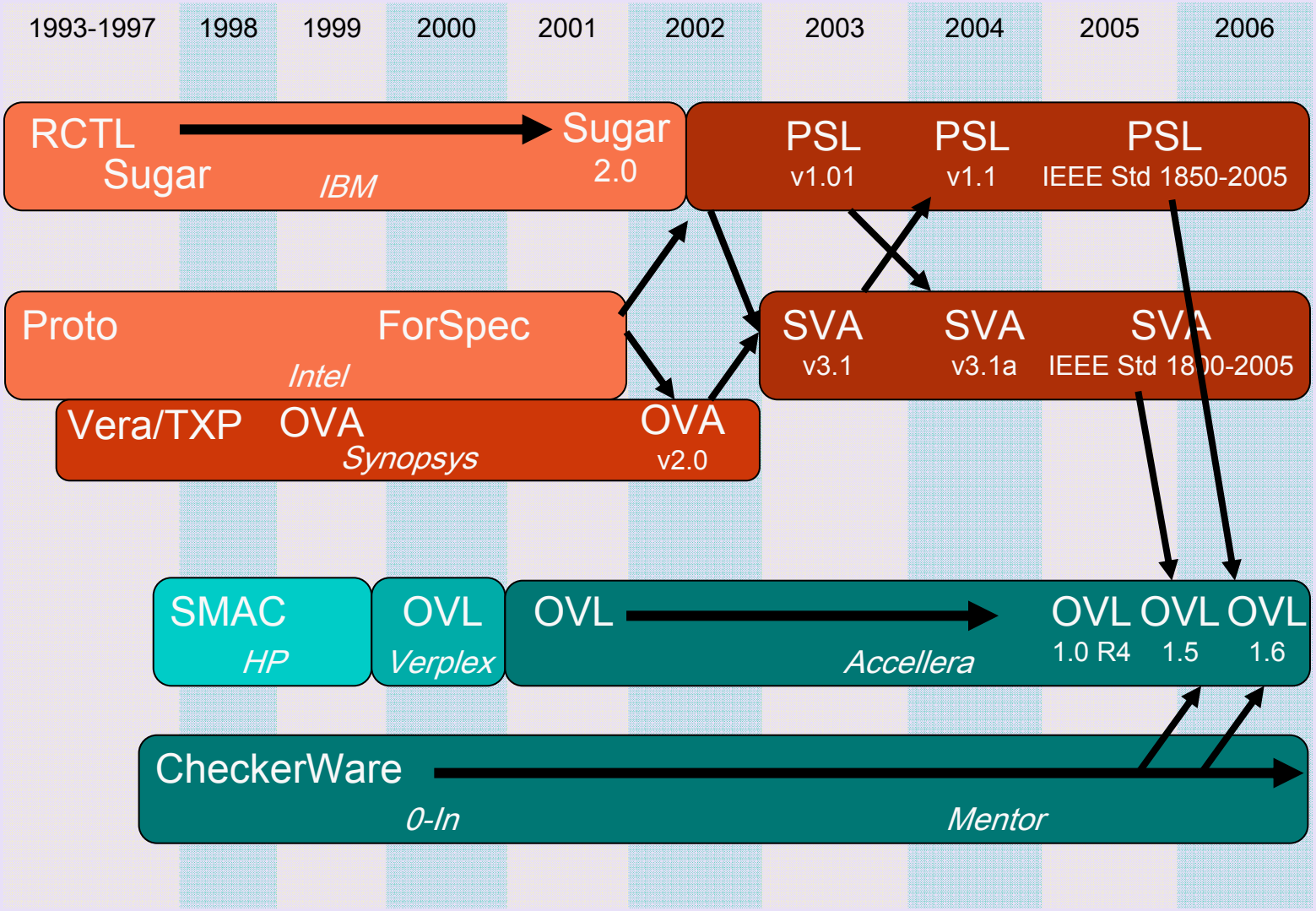




SystemVerilog Assertion

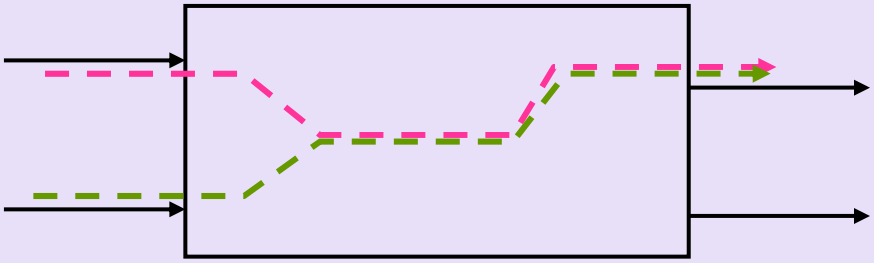
```
property arb;
  @(posedge clk)
    req |=> ##[0:2] gnt;
endproperty
assert property (arb);
```

The Evolution of Assertions



Where to apply formal

- Control, data transport: concurrency, multiple streams
- Arbiters of many kinds
- On-chip bus bridges
- Power management unit
- DMA controller
- Host bus interface unit
- Schedulers
- Interrupt controller
- Memory Controller
- Tag generator
- Credit manager block
- Standard interfaces (PCI Express®, USB)
- Clock disable unit



Multiple, concurrent streams

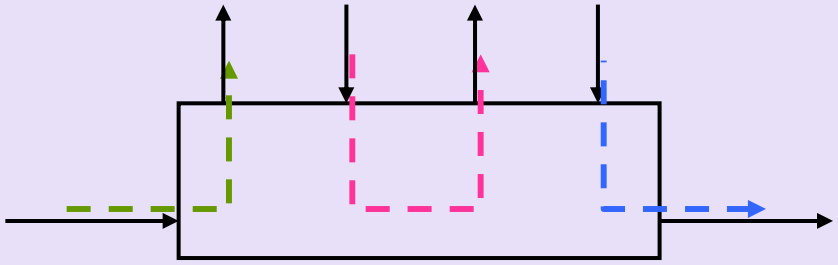
Hard to completely verify using simulation

"10 bugs per 1000 gates"

-Ted Scardamalia, IBM

Where not to apply formal

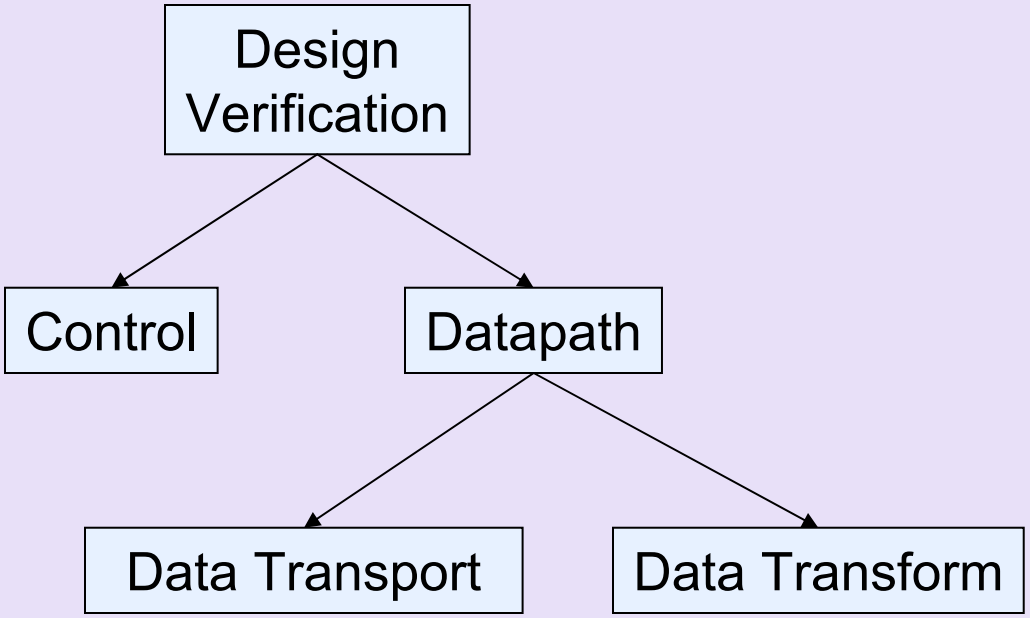
- Data transform: often sequential, math operations
- Floating point unit
- Graphics shading unit
- Convolution unit in a DSP chip
- MPEG decoder
- Classification search algorithm



Single, sequential functional streams

"2 bugs per 1000 gates"
-Ted Scardamalia, IBM

Control versus datapath



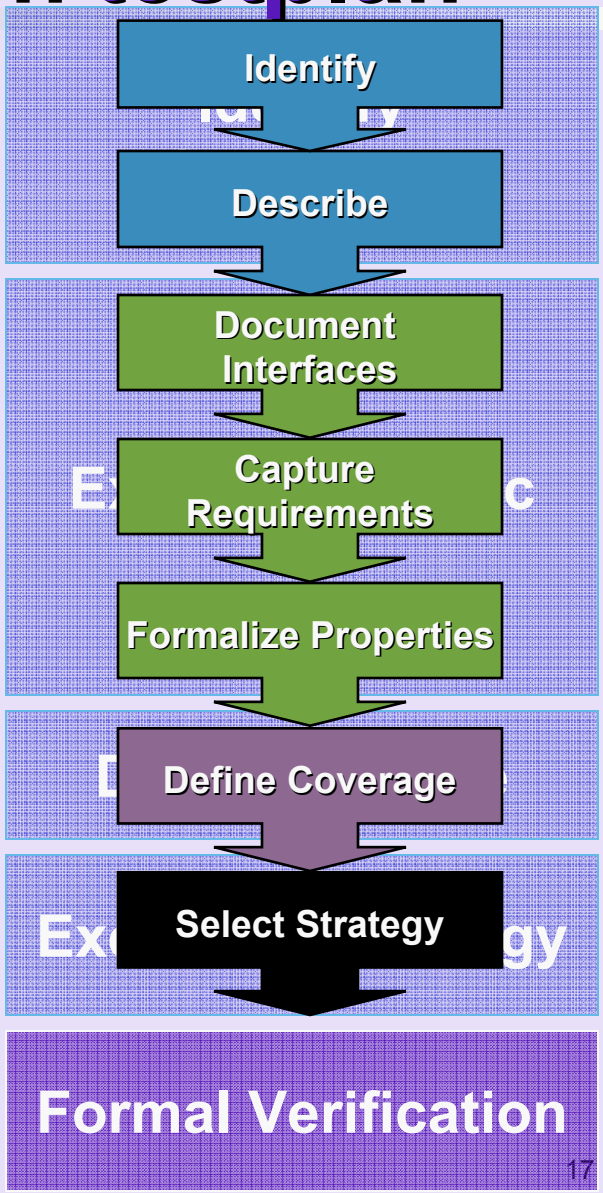
Design Characteristics

- Concurrency
  - ✓ High concurrency, e.g. a block interacting with multiple, asynchronous blocks
  - ✓ E.g. DMA controller
- Connectivity
  - ✓ A block connected with many others
  - ✓ E.g. Arbiter
- Concise specification of end-to-end properties
  - ✓ E.g. DLL blocks (vs. Packet decoder)
- Control (vs Computation)
- Flop count
  - ✓ Arbiters (vs. DLL, switch cores)

The formal verification testplan

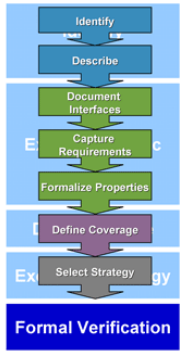
- Seven steps
  - ✓ Step 1: Identify potential candidate
  - ✓ Step 2: Describe intended behavior
  - ✓ Step 3: Document interface signals
  - ✓ Step 4: Document functional requirements
  - ✓ Step 5: Formalize requirements description
  - ✓ Step 6: Document coverage criteria
  - ✓ Step 7: Select strategy for each requirement

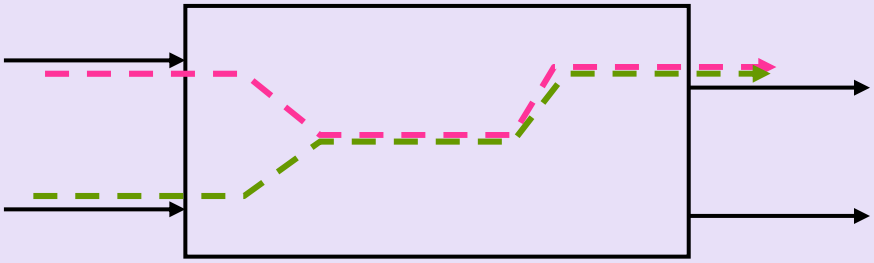
- H. Foster, L. Loh, B. Rabii, V. Singhal. "Guidelines for creating a formal verification testplan", DVCon 2006



Step 1. Identify candidates

- Real examples include:
- Arbiters of many kinds
- On-chip bus bridges
- Power management unit
- DMA controller
- Host bus interface unit
- Schedulers
- Interrupt controller
- Memory Controller
- Tag generator
- Credit manager block
- Standard interfaces (PCI Express, USB)
- Clock disable unit





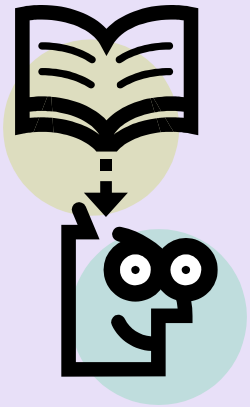
Multiple, concurrent streams

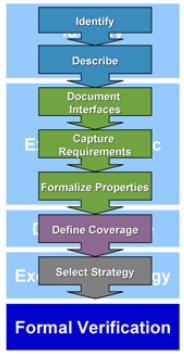
Hard to completely verify using simulation

"10 bugs per 1000 gates"

-Ted Scardamalia, IBM

Step 2. Describe intent



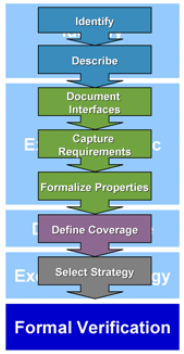


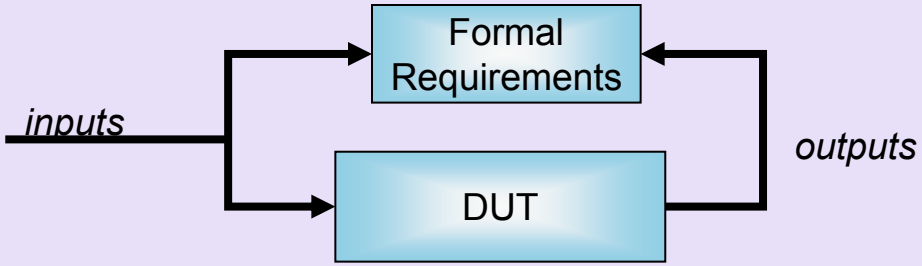
- Write down a high-level description of design Intent
  - ✓ Natural language description
  - ✓ Helps define the final goal
  - ✓ Does not need to be a formal process
    - – Can be a video taped whiteboard discussion
  - ✓ Leverage existing specification if possible

Step 3. Define interface signals

- Defines the signal grouping for each interface
- Will be used later to determine if additional requirements are needed
  - ✓ Each signal should appear in the requirements

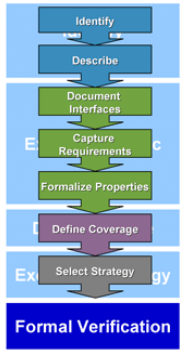
Signal Name	Description	Size	Direction
PCLK	PCIe Core clock	1-bit	In
PRESETn	Master Reset (active low)	1-bit	In
DLL_VALID	TLP from DLL is value	1-bit	In
DLL_SOP	TLP is starting	1-bit	In
DLL_EOP	TLP is ending	1-bit	In
DLL_TLP	TLP packet	32-bit	In
DLL_DISCARD	Discard TLP from DL	1-bit	In
TL_VALID	TLP from TL is valid	1-bit	Out
TL_SOP	TLP is starting	1-bit	Out
TL_EOP	TLP is ending	1-bit	Out
TL_TLP	TLP packet	32-bit	Out
DLL_REPLAYBUF_WAIT	Wait cycle from Replay Buffer	1-bit	In





Step 4: Requirements checklist

- Capture what needs to be verified in a natural language
  - ✓ Clarifies the exact requirements you need to cover
  - ✓ Can be interface related, end-to-end requirements, or internal



Transaction Layer requirements

A. Interface requirements

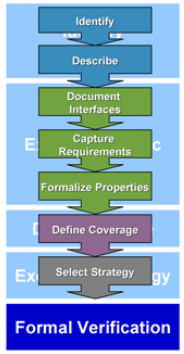
1. **DLL must repeat transaction if TL inserts a wait cycle**
2. **SOP for a transaction must be followed by EOP for same transaction**
3. **. . .**

B. End-to-end requirments

1. **For any VC, for any Type, the TLP sequence from AL must equal the sequence to DLL**
2. **TLP packets must follow the PCI Express packet ordering rules**
3. **…**

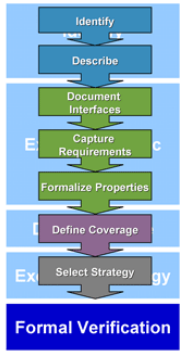
Step 5 : Formalize requirements

- Capture all requirements using assertions
  - ✓ Use existing verification IP (monitors) if available
  - ✓ Convert additional requirements to formal assertions



Step 5 : Formalize requirements

- Example
  - ✓ ***There should not be another*** dll_sop ***after a*** dll_sop ***occurs, until a*** dll_eop ***occurs on the DLL-TL interface***



SVA

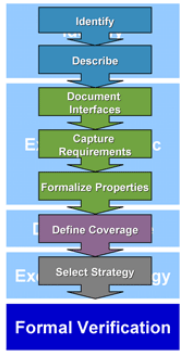
```
property P_no_sop_sop;
  @(posedge PCLK) disable iff (~PRESETn)
     dll_sop |=> ~dll_sop[*0:$]  ##1  dll_eop;
endproperty

A_no_sop_sop: assert property (P_no_sop_sop);
```

PSL

```
default  clock = PCLK;
A_no_sop_sop: assert (always dll_sop ->
                                  next(~dll_sop until dll_eop))
                            abort (~PRESETn);
```

Step 6: Coverage

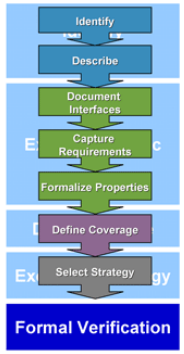


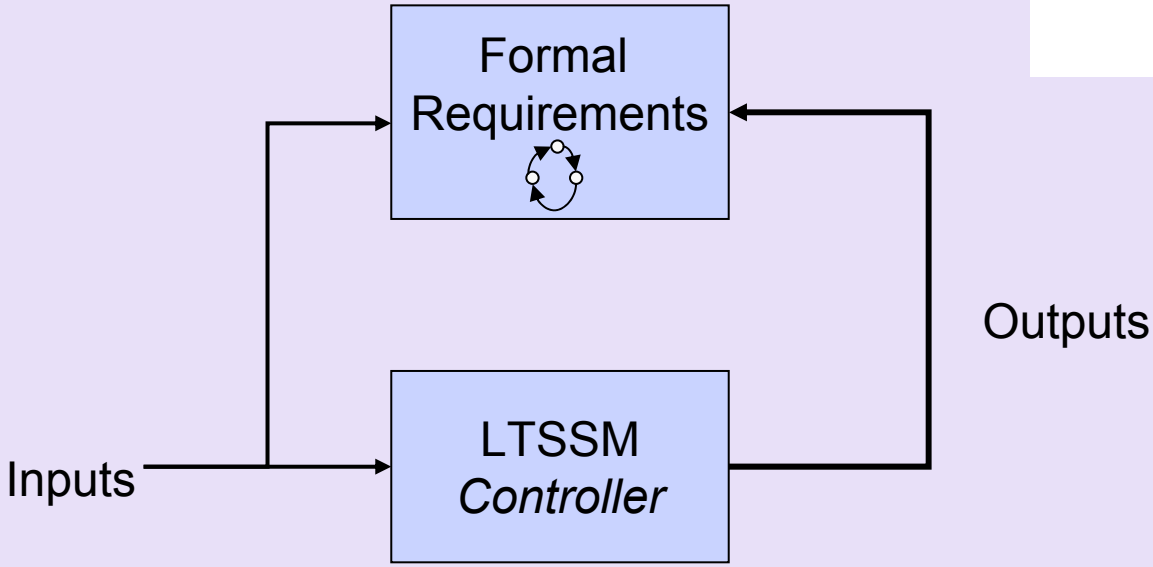
- Identify coverage goals per block
- Use to identify constraint problems
- Merge coverage between formal and simulation

- **Set 1: Input coverage – All possible pairs of Type of consecutive TLPs**
- **Set 2: Output coverage – Posted TLPs passed earlier Non-Posted TLPs**
- **Set 3: Internal main state-machines – Arbitration state machines**

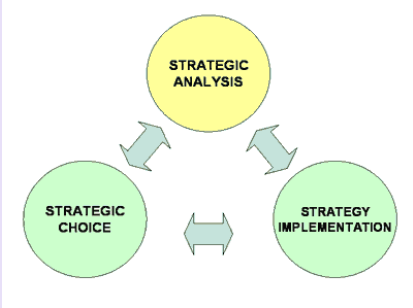
Step 6: Coverage

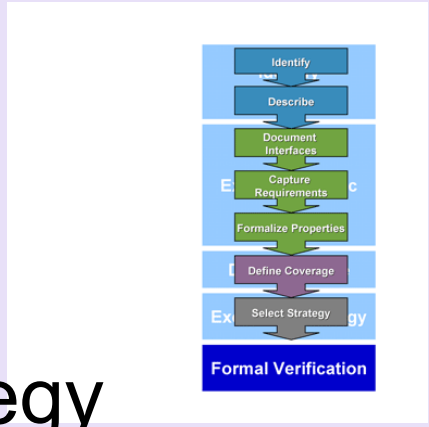
- Did the proper sequence transitions occur in FSMs for my requirements model?





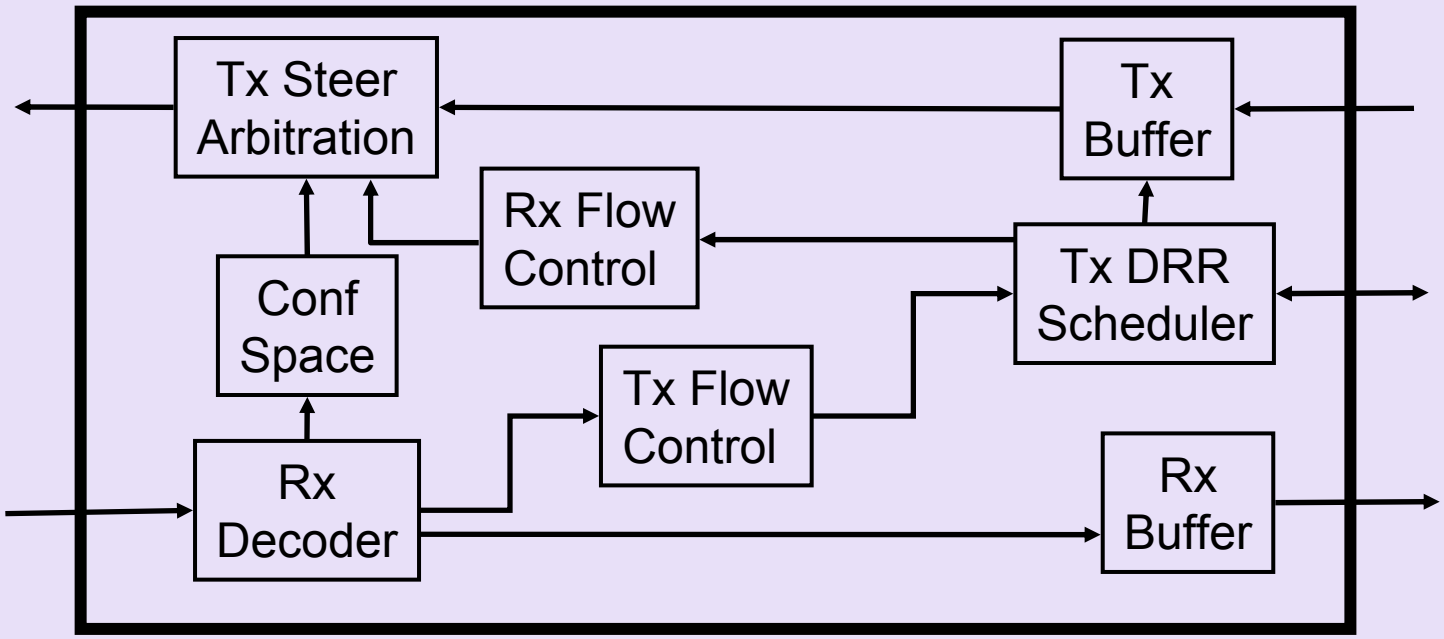
Step 7: Strategy





- Verification technology and strategy will depend on the goal for each formalized requirement.
  - ✓ Proofs
  - ✓ Bug hunting
  - ✓ Interface formalization
  - ✓ Coverage enhancement

Transaction Layer block



- 128-bit datapath
- 8 VCs
- History, policy-dependent DRR scheduler
- Conf responses arbitrated with TLPs and FC DLLPs
- Tx, Rx Buffers can store multiple TLPs (upto 32kb each)

Formal property list

- Categories:
  - ✓ Design-internal assertions
    - E.g. do not write to FIFOs when full
  - ✓ Inferface handshake checks
    - TL-DLL interface
    - TL-AL interface
  - ✓ End-to-end checks
    - Integrity of TLPs going across blocks

Formal property list

- Interface to DLL:
  - ✓ Rx TLP handshake, Tx TLP handshake
    - E.g. if DLL Tx inserts wait cycle, TL Tx respects the wait
  - ✓ Tx FC credit reception, Rx FC credit transmission
    - For transmission, FC protocol rules: no more than 127 headers outstanding, no more than 2047 data outstanding
- Interface to Application Layer:
  - ✓ Rx TLP handshake, Tx TLP handshake
  - ✓ Other application-specific checks
    - Respect per-VC back-pressure

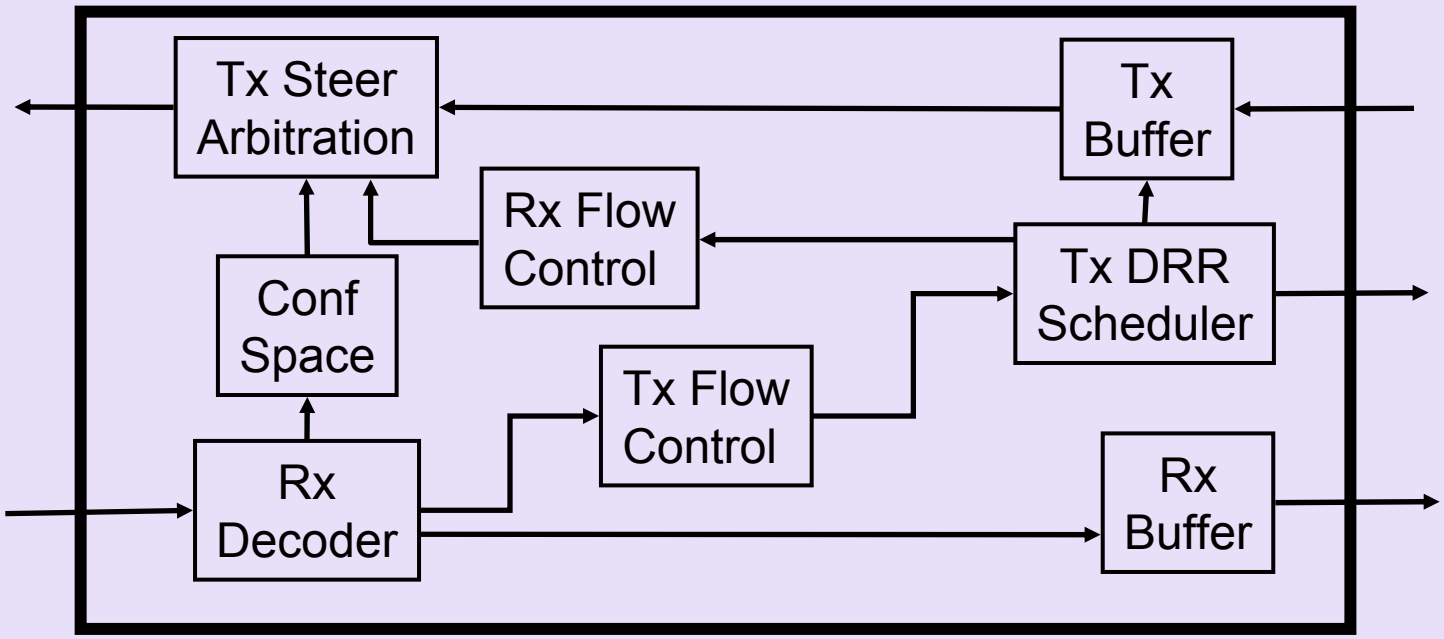
Formal property list

- Rx end-to-end checks
  - ✓ Rx issues the correct error signals or Completion responses for
    - Malformed packets
    - Unsupported Request TLPs
    - Poisoned TLPs
    - ECRC error
    - TLPs that should be dropped
  - ✓ For each VC, for each non-Error TLP, for each Type (Posted, Non-Posted, Completion)
    - TLP order remains 1:1
  - ✓ For each VC, packet passing respects PCI Express ordering requirements

Formal property list

- TL Tx end-to-end (more concurrency than Rx)
  - ✓ Multiple input streams of data (for TLPs) and only one stream of output data
    - Internal Completions from Rx path
    - Normal TLPs, but maybe one stream for each VC * each Type
    - TLPs after Message framing, Config framing, Completion framing
- Rx Tx end-to-end (less concurrency, but more computation)
  - ✓ Multiple input streams of data (for TLPs) and only one stream of output data
    - Internal Completions from Rx path
    - Normal TLPs, but maybe one stream for each VC * each Type
    - TLPs after Message framing, Config framing, Completion framing

TL example: formal challenges

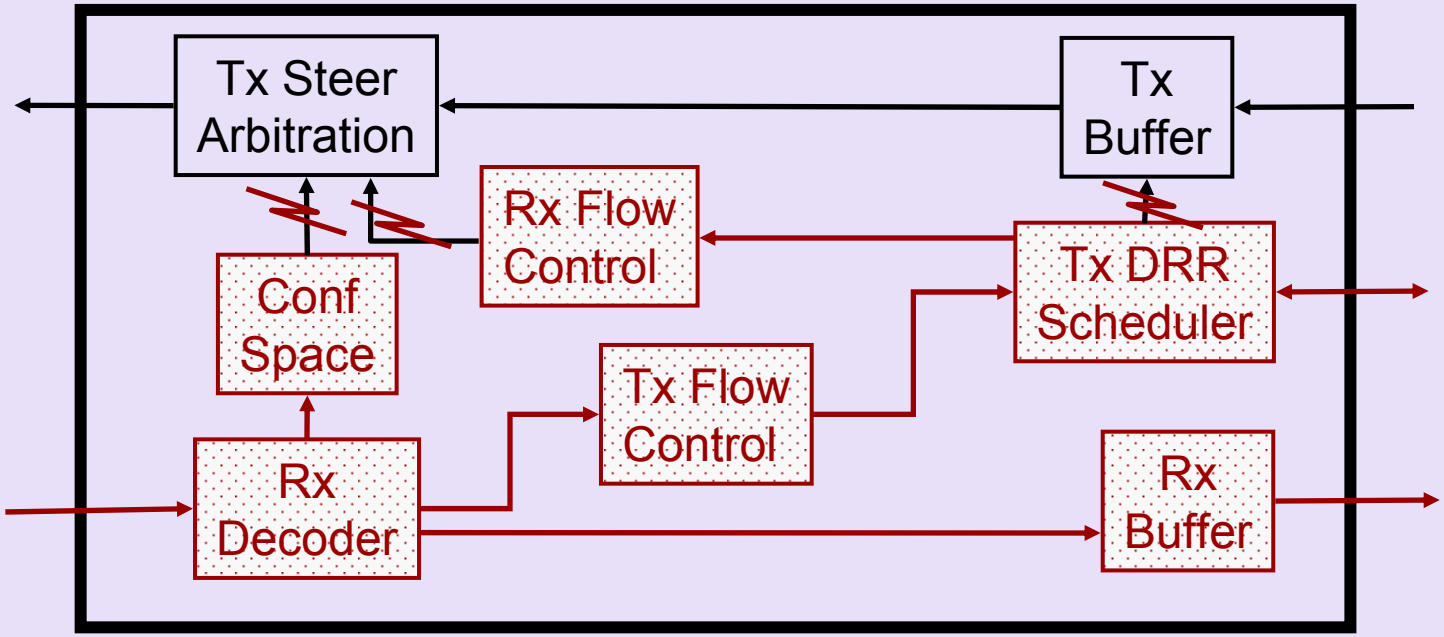


- Data-width large
- Supports upto 8 requestors
- Mathematical scheduling algorithm
- Large data buffers

TL: verification strategies

- Localization
- Datapath abstraction
- Assume-guarantee
- Sequence abstraction
- Restrictions

Localization - example



- Property: for any VC, sequence of Posted packets arriving to Tx Buffer go out in the same order from Tx Steer Arbitration logic

- Scheduler algorithm, Rx DLLP packets, Conf Packets can all be arbitrarily chosen to prove this property

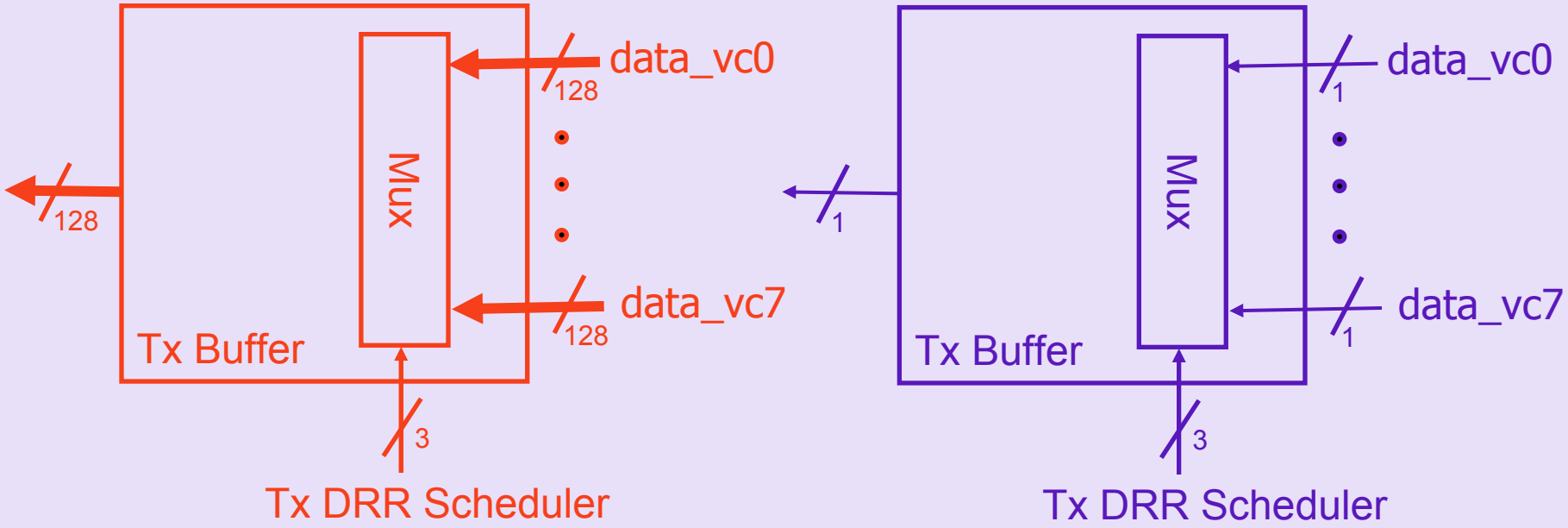
Localization

- Free design logic that is irrelevant to property proof
- Increases legal space (removed logic is arbitrary)
- But decreases complexity

- Safe:
  - ✓ If removed logic is irrelevant, proof will go through
  - ✓ Else, trace (counterexample) shows relevance of logic

- Many tools allow user to perform localization
  - ✓ JasperGold, Synopsys Magellan, Cadence ISV, Mentor 0-In

Datapath abstraction - example



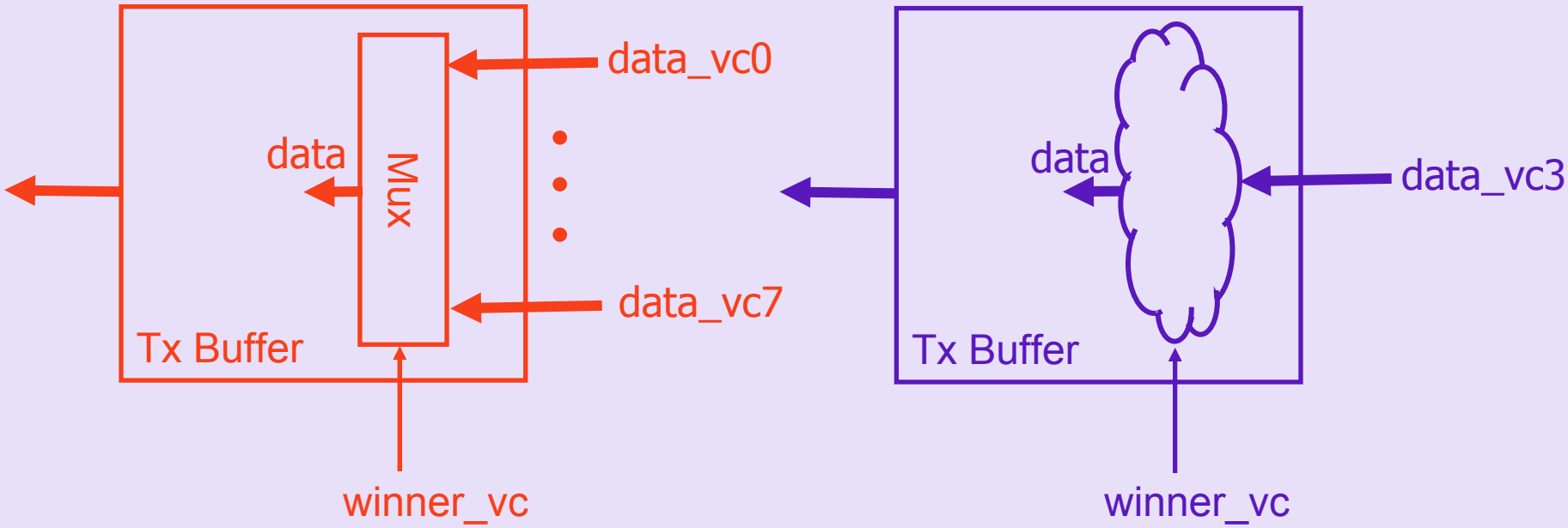
- Property: for any VC, sequence of Posted packets arriving to Tx Buffer go out in the same order from Tx Steer Arbitration logic

- Pick any arbitrary bit from the 128-bit datapath, say data[29]
- Prove correctness, one bit at a time

Datapath abstraction

- Have to assume, or prove, design control is independent of data bits
- To be complete, have to prove
  - ✓ For each bit
  - ✓ All bits travel together

Assume-guarantee - example

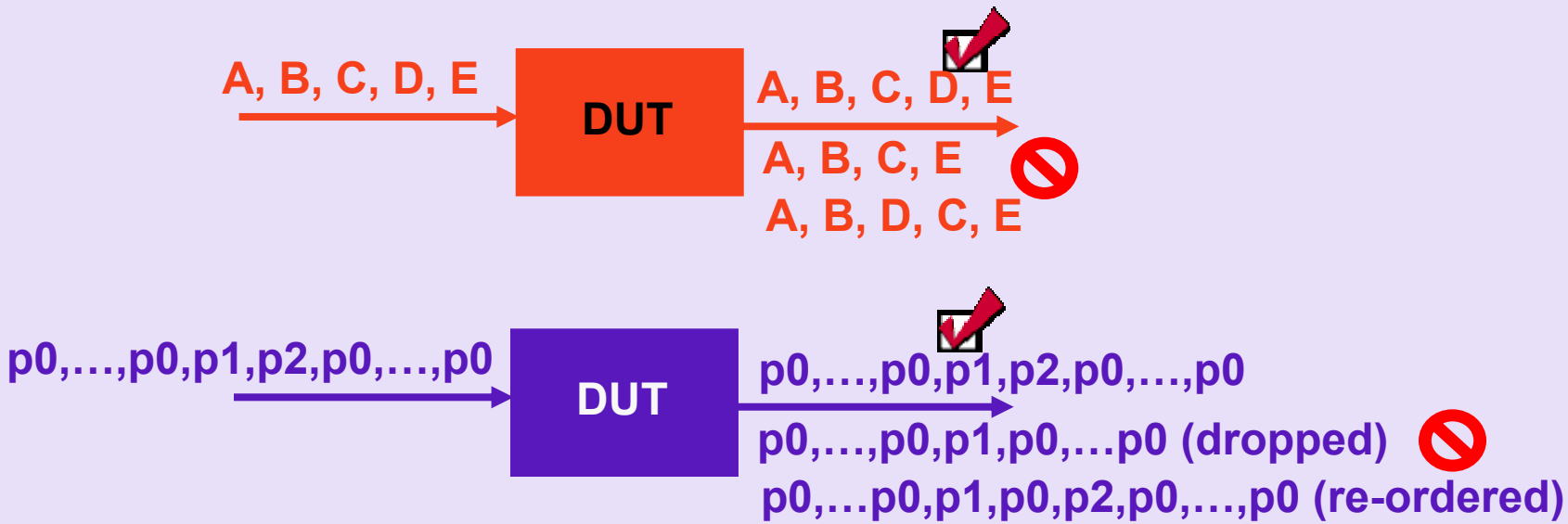


- Property: for any VC, sequence of Posted packets arriving to Tx Buffer go out in the same order from Tx Steer Arbitration logic
- Add assumptions: "**stopat data**"; "**assume (winner_vc == 3) ➔ (data = data_vc3)**"

Assume-guarantee

- User commands may differ from tool to tool
- May require some modeling logic
- Decomposes total verification complexity
  - ✓ Remember, complexity is exponential
- To be complete,
  - ✓ Every assumption should be proven independently

Sequence abstraction - example



- Property: for any VC, any (infinite) sequence of Write packets arriving into DUT go out in the same order

- Pick two arbitrary, consecutive packets. Mark them using a data bit. If DUT has a bug, it will show up in one of these two packets on the output.

Restrictions

- Useful when the complexity is expected to be unmanageable
- Restrictions give partial proofs
- Restrict to
  - ✓ Certain modes: e.g. MEM, IO transactions
  - ✓ Certain sequences: No back-to-back
  - ✓ Disallow exceptions: no Malformed, UR
  - ✓ Get around known bugs: No 3-DW MSG followed by 3-DW MRD
- Planned restrictions can help manage complexity, staged verification, schedule/resources

Property Sets

- Property Set 0:
  - ✓ Design-internal assertions, e.g. state machine is one-hot encoded
- Constraint Set 1:
  - ✓ Allow TLPs from AL on only VC0
  - ✓ Allow only Posted TLPs
- Property Set 1:
  - ✓ Data integrity of TLPs on Posted, VC0

Property Sets

- Constraint Set 2:
  - ✓ Allow Non-Posted and Completion TLPs
- Property Set 2:
  - ✓ Prove PCI Express ordering rules
- Constraint Set 3:
  - ✓ Allow TLPs on other VCs
- Property Set 3:
  - ✓ Prove VC arbitration is fair

Staged verification

	Property Set 0	Property Set 1	Property Set 2	Property Set 3	Property Set 4
Constraint Set 1		3 weeks			
Constraint Set 2					
Constraint Set 3					
Constraint Set 4					

Staged verification

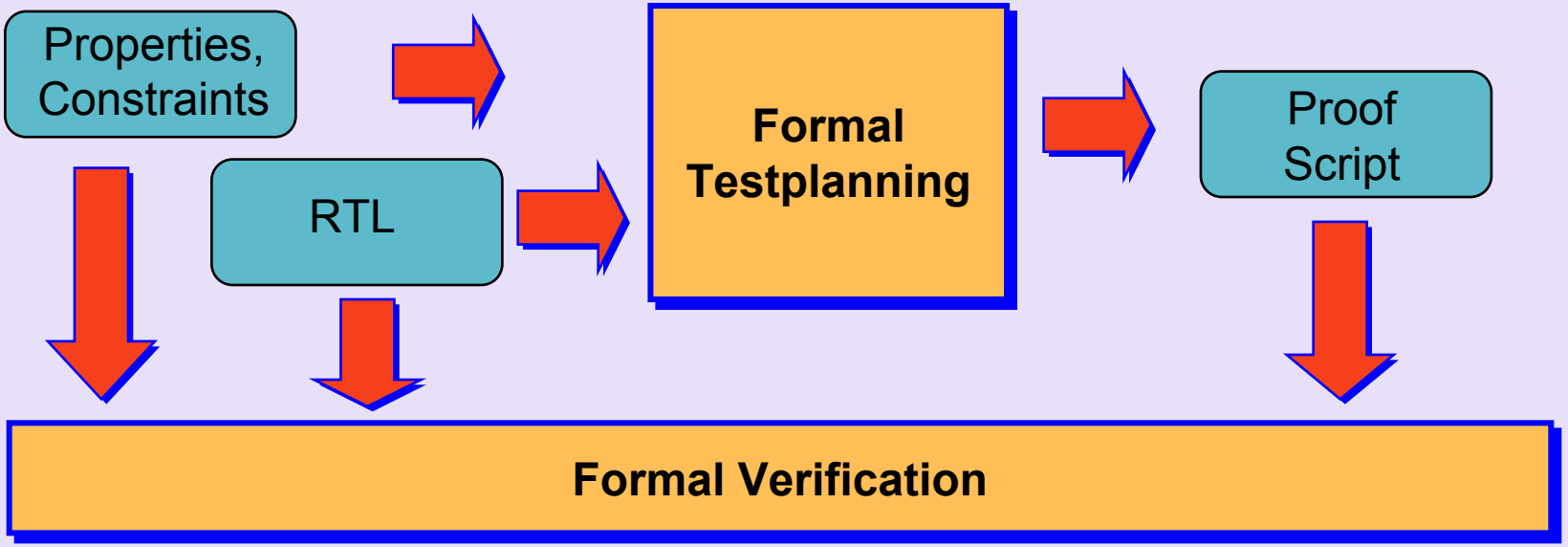
	Property Set 0	Property Set 1	Property Set 2	Property Set 3	Property Set 4
Constraint Set 1		3 weeks			
Constraint Set 2			4.5 weeks		
Constraint Set 3					
Constraint Set 4					

Staged verification

	Property Set 0	Property Set 1	Property Set 2	Property Set 3	Property Set 4
Constraint Set 1		3 weeks			
Constraint Set 2			4.5 weeks		
Constraint Set 3				5.5 weeks	
Constraint Set 4					

Proof script

- Executable verification strategy
  - ✓ E.g. localization: “stopat arb_winner”
- Regression advantage
  - ✓ Verification strategy may takes multiple man-weeks to evolve
  - ✓ Once finalized, can be implemented in a script
  - ✓ With a given script, run-time may be only a few machine-hours



Proof scripts in PCIe IP

- Design IP
  - ✓ RTL
  - ✓ Documentation
  - ✓ Synthesis scripts
- Verification IP
  - ✓ Property list (English)
  - ✓ Proof scripts
- Applicable for
  - ✓ Internal IP
    - Enables re-use, maintenance, modification
  - ✓ 3rd party IP
    - Facilitates evaluation: replay of formal proofs

Conclusions

- Formal verification is exhaustive verification, but…
  - ✓ Needs careful selection of blocks
  - ✓ Abstraction and reduction methods
- PCIe data transport blocks are ideal fit (TL, DLL, PL)
- 3rd party PCIe IP (RTL) should come with formal certificate of correctness
  - ✓ Replayable during evaluation at customer site

- Contact: vigyan@oskitech.com

Thank you for attending the
PCI-SIG Developers Conference 2006.

For more information please go to
www.pcisig.com

Formal Verification for PCIe 1.1 and 2.0 RTL designs

Vigyan Singhal

Oski Technology