2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

A Systematic Formal Reuse Methodology: From Blocks to SoC Systems

Hao Chen, Yi Sun, Ang Li, Dorry Cao

NVM Solutions Group

Intel Corporation

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

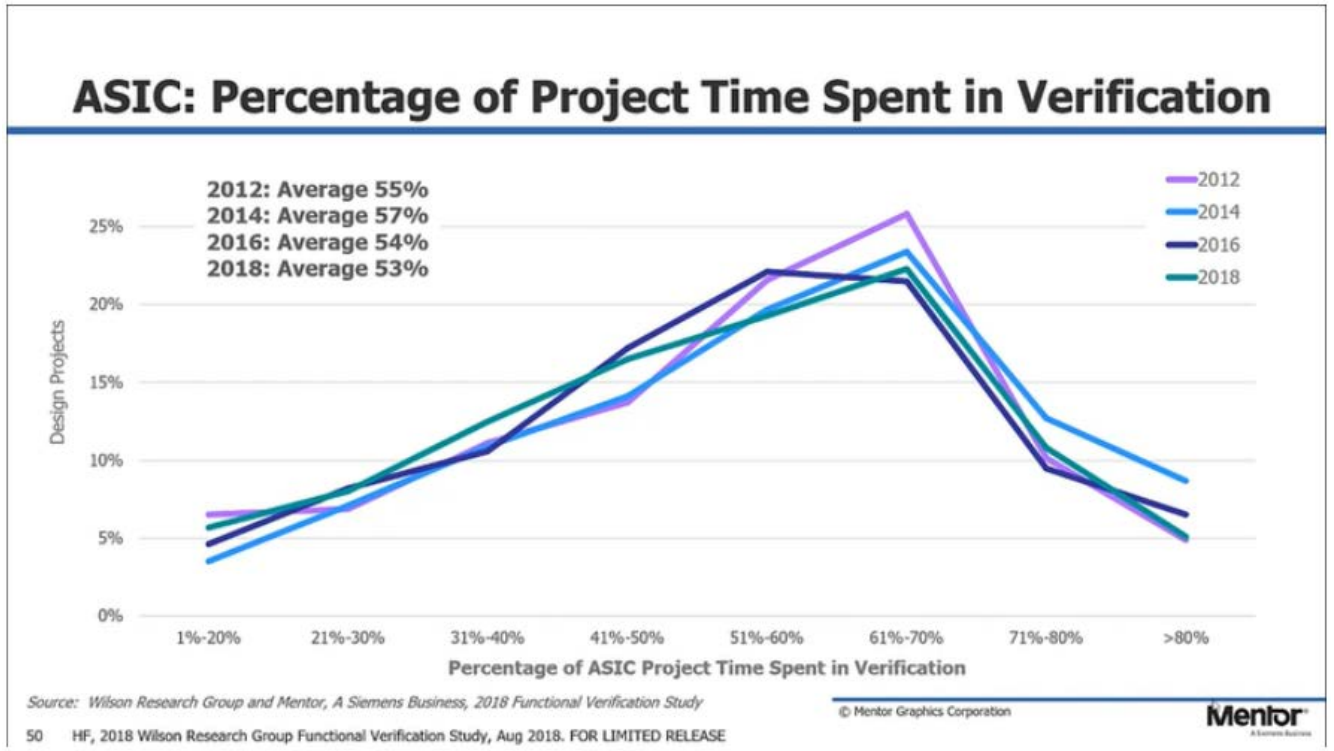
Agenda

- **Introduction**
- Reusable FV Testbench Structure
- Integrating FV Collaterals into Emulation
- Real-World Results: a SSD Controller SoC
- Conclusion

accellera SYSTEMS INITIATIVE

SoC Verification – Industry Trends

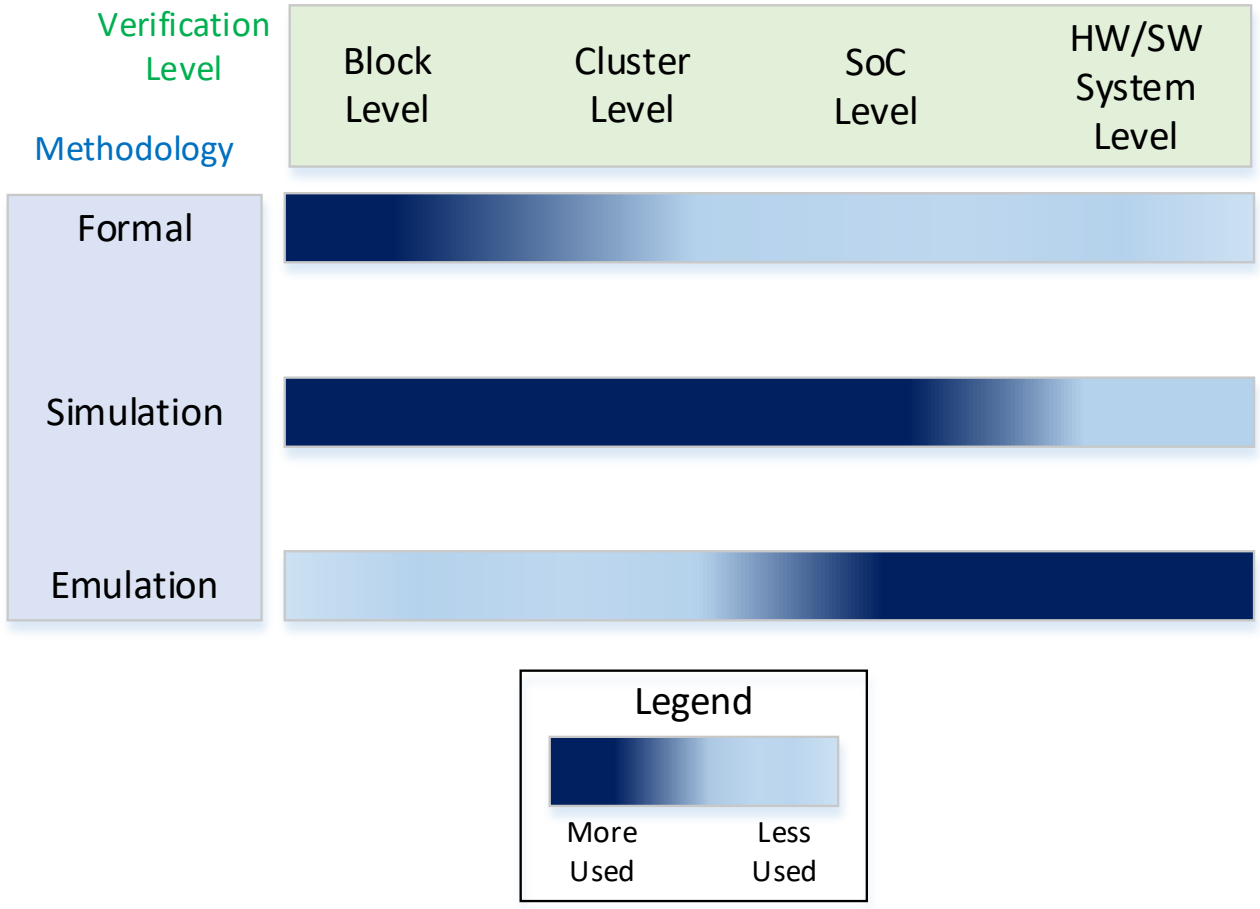
- SoC verification is full of hard problems:
  - HW/SW interactions at SoC top level
  - 3rd party IP integration at subsystem level
  - Exhaustive corner cases at block level
  - …
- These drive the continuous evolution of verification strategies/methodologies



Courtesy: Harry D. Foster “The 2018 Wilson Research Group ASIC and FPGA Functional Verification Study” [1]

Our Hybrid Verification Strategy

- Goal: To extend our capability to verify larger and more complex SoCs
  - Block-level: Simulation + Formal
  - Cluster-level: Simulation
  - SoC-level/System-level: Simulation + Emulation



Challenges

- Hybrid strategy ⇨ Maximize ROI in verification quality
- Challenges:
  - Verification gaps on boundaries
  - Duplicate effort

How to Close the Gap Between Schedule and Productivity?

Solution:

A Systematic formal reuse methodology to enable **code reuse** and **cross proof** from FV and improve interoperability between FV and other platforms

Agenda

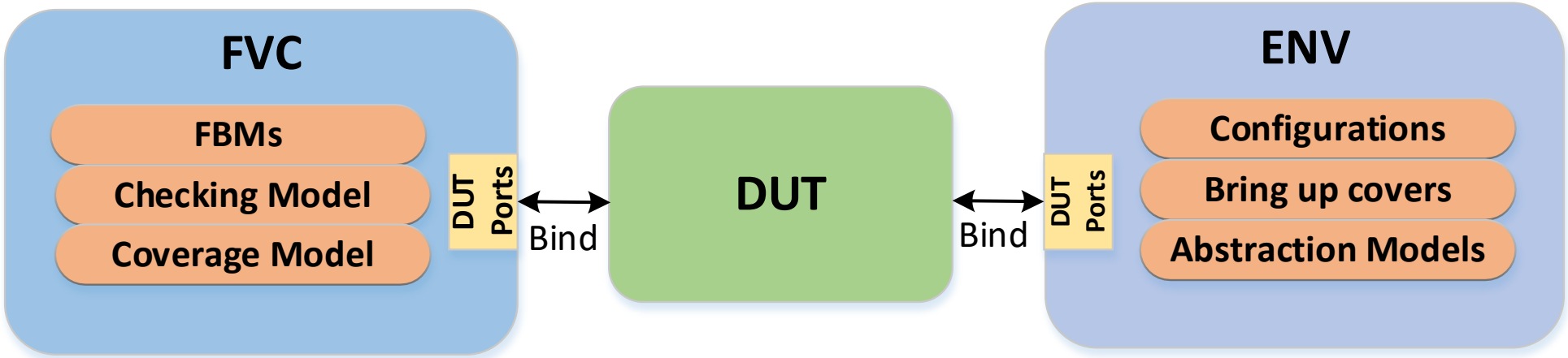
- Introduction
- **Reusable FV Testbench Structure**
- Integrating FV Collaterals into Emulation
- Real-World Results: a SSD Controller SoC
- Conclusion

Principles for Formal Reuse

- **Modularity**
  - Modularized components
  - Easy to reuse in different platforms and projects
- **Usability**
  - Simple and clean interfaces
  - Easy to be integrated by other verification environments
- **Consistency**
  - Follows a consistent, easy-to-communicate form

Complete Formal Testbench

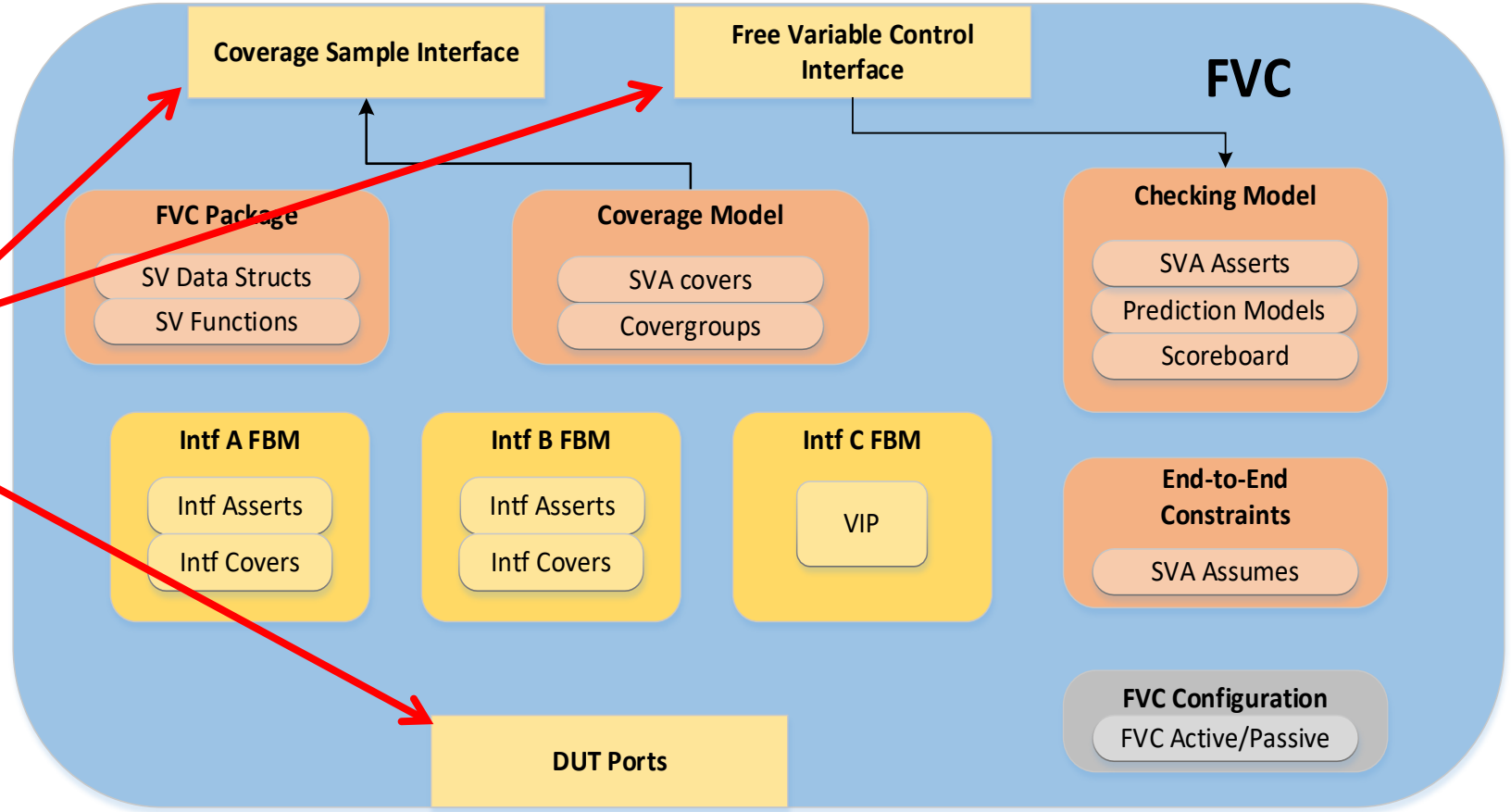
- A complete formal testbench consists of
  1. A Formal Verification Component (**FVC**)
  2. A formal testbench environment (**ENV**)



Formal Verification Component (FVC)

- An FVC provides:

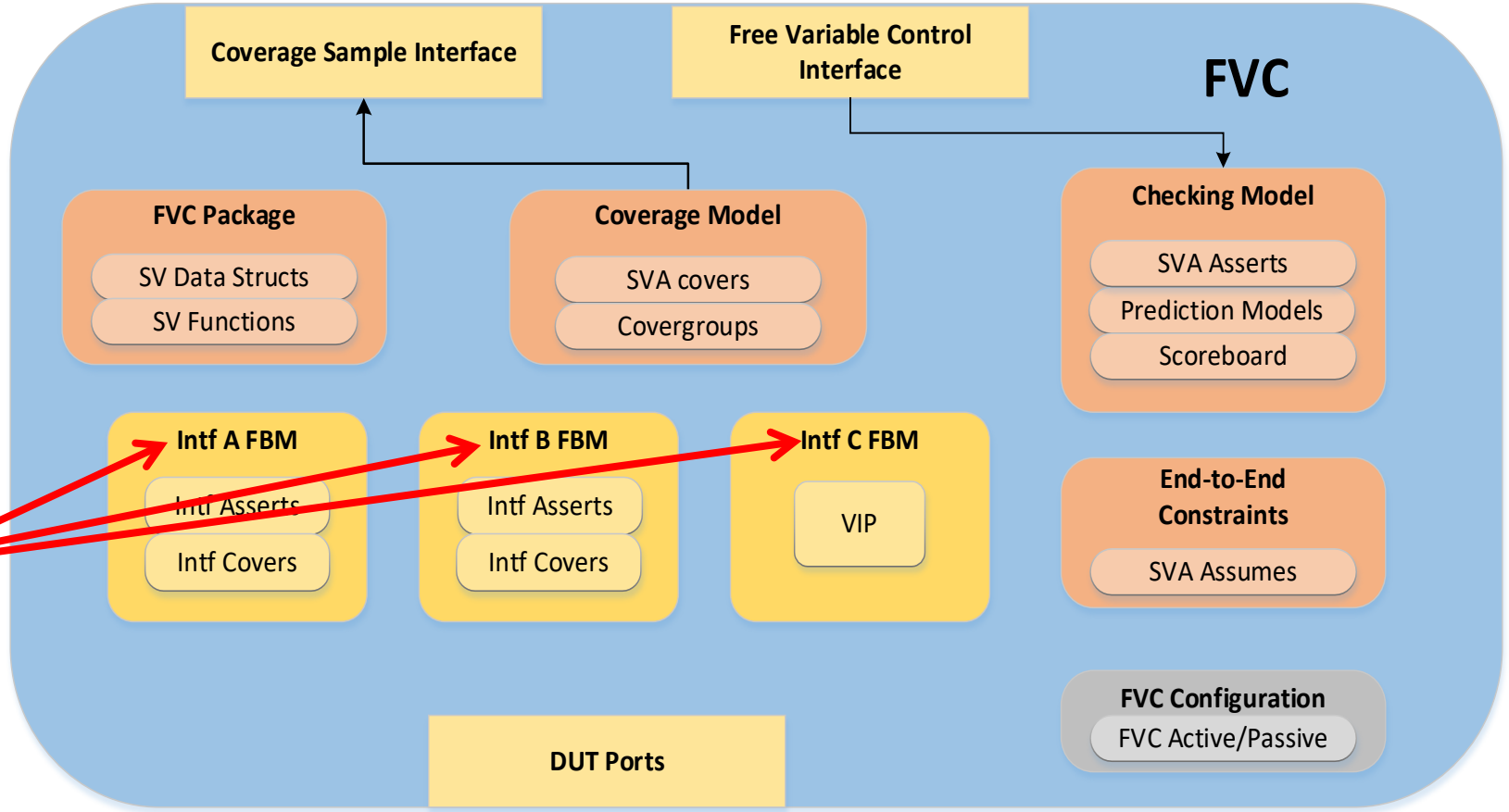
Simple and clean interfaces



Formal Verification Component (FVC)

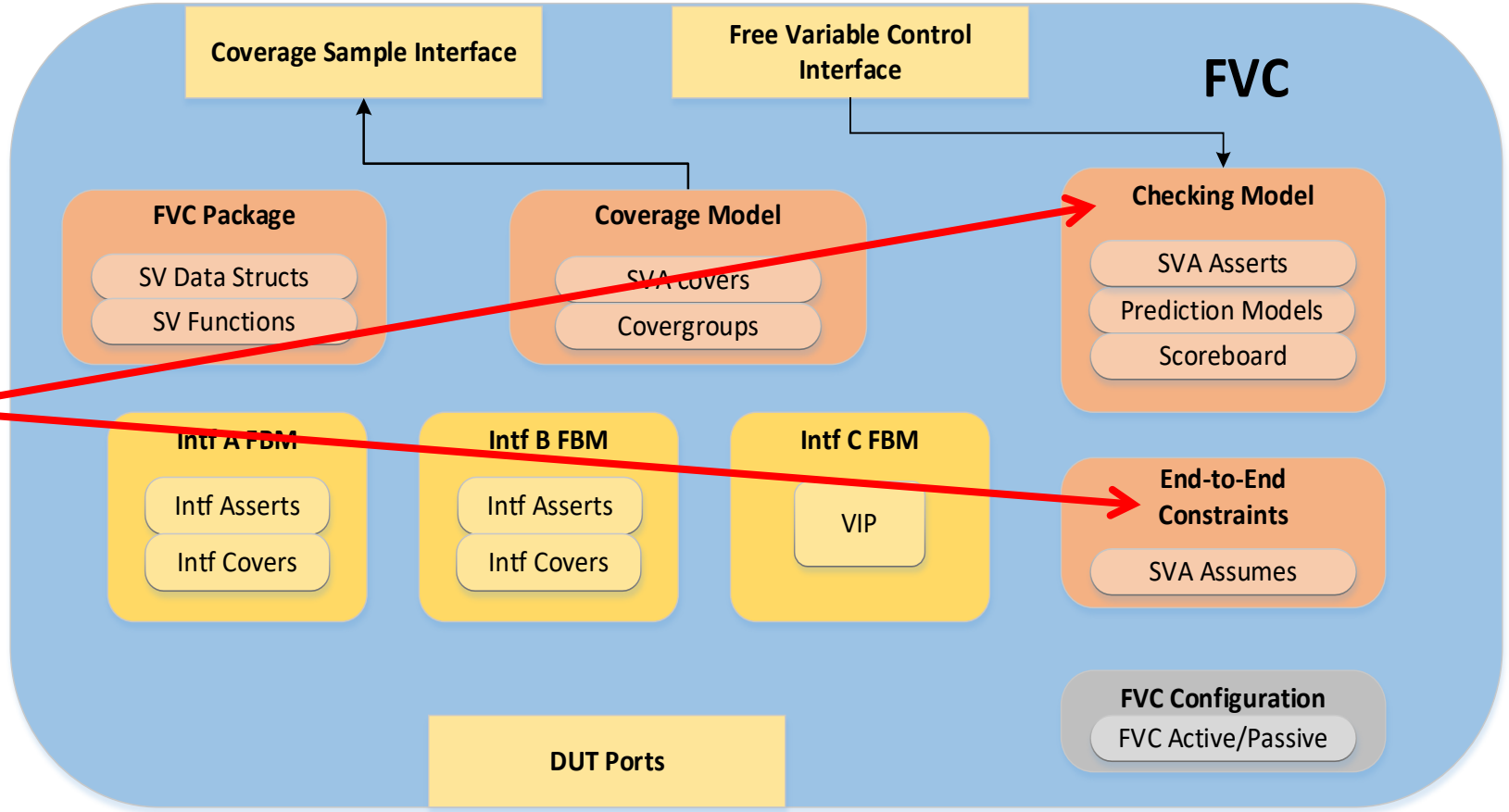
- An FVC provides:

Interface protocol modeling and checking



Formal Verification Component (FVC)

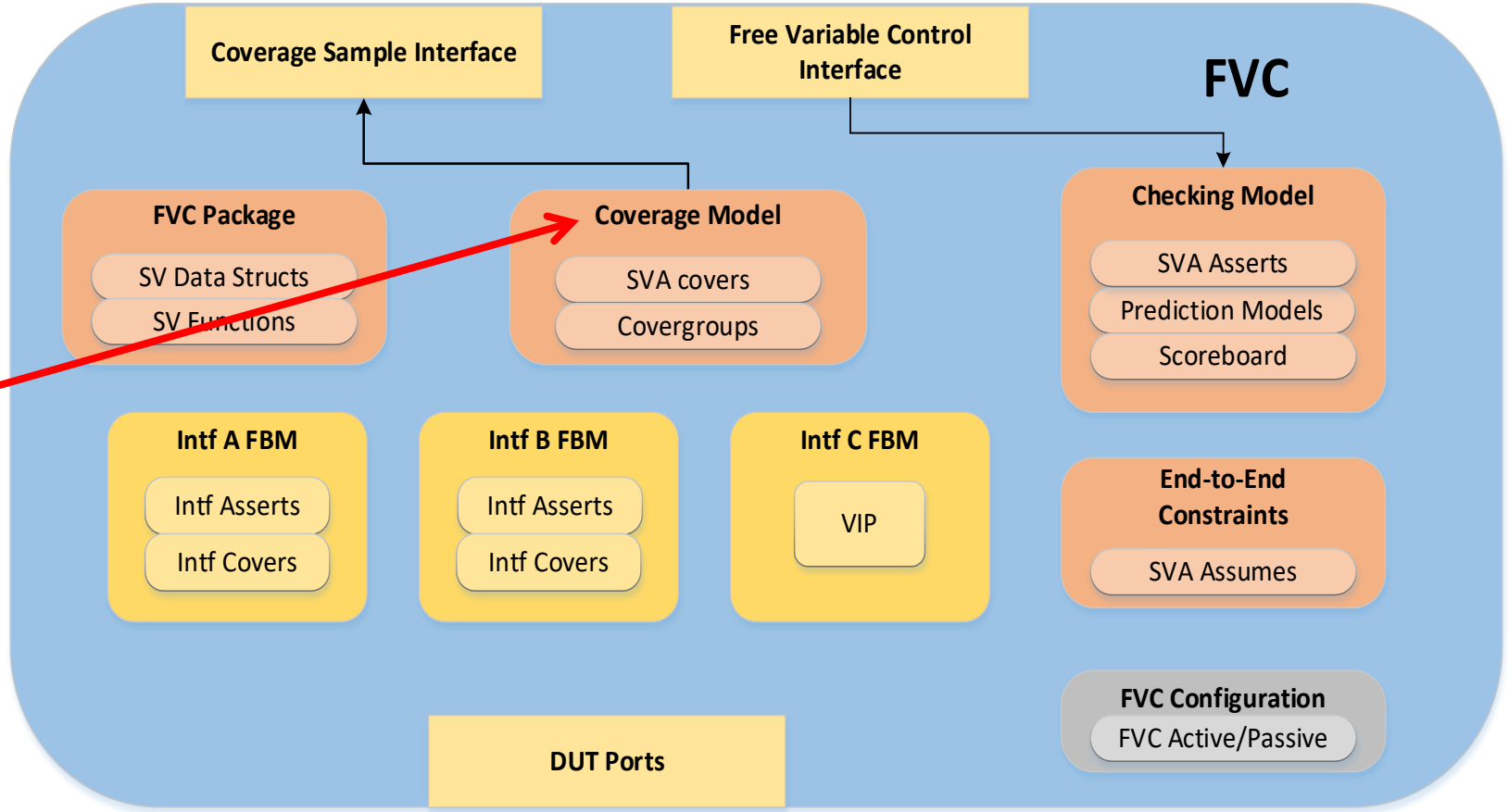
- An FVC provides:
  - **End-to-end checkers**
  - **End-to-end constraints**



Formal Verification Component (FVC)

- An FVC provides:

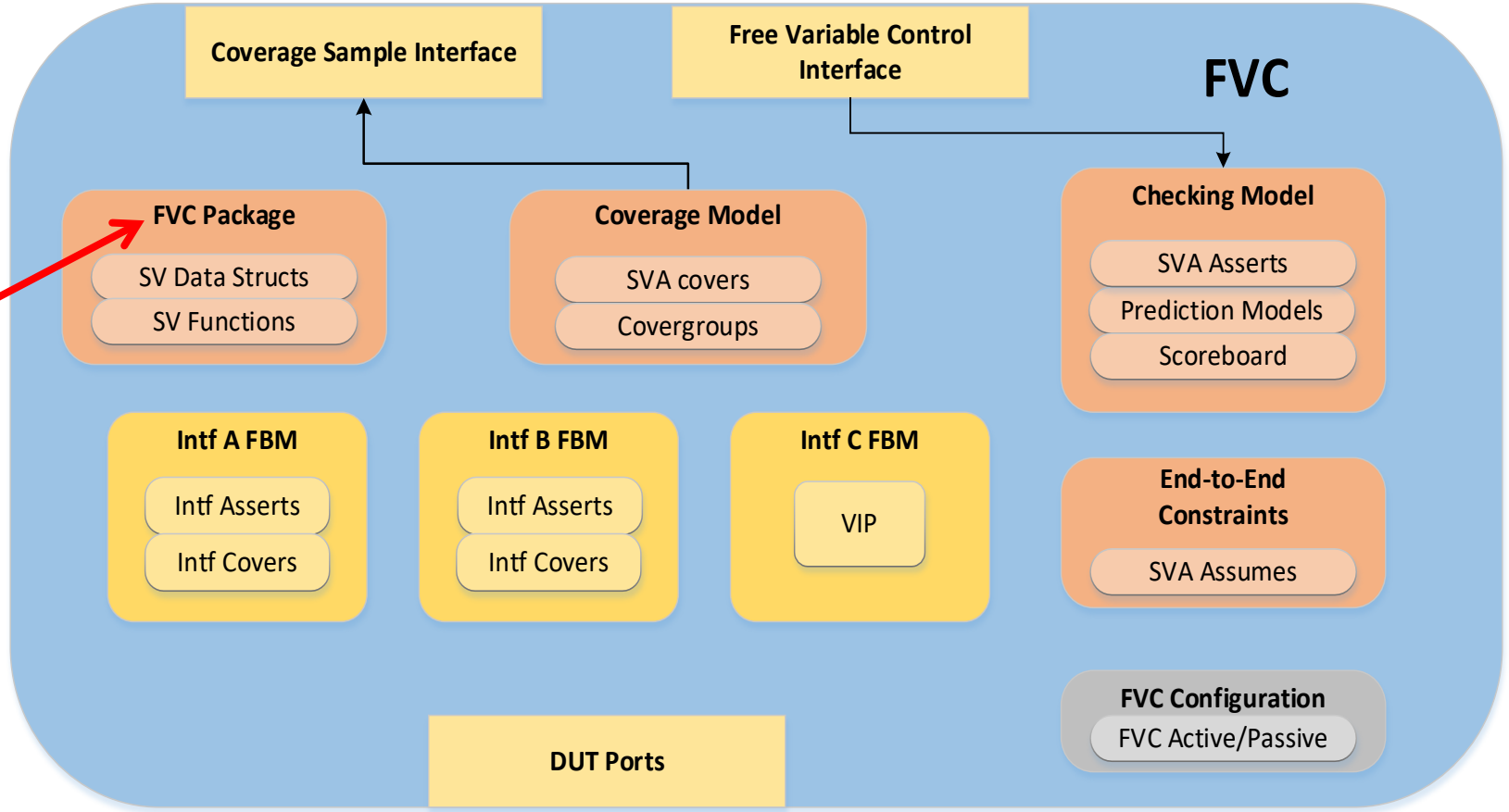
Functional coverage



Formal Verification Component (FVC)

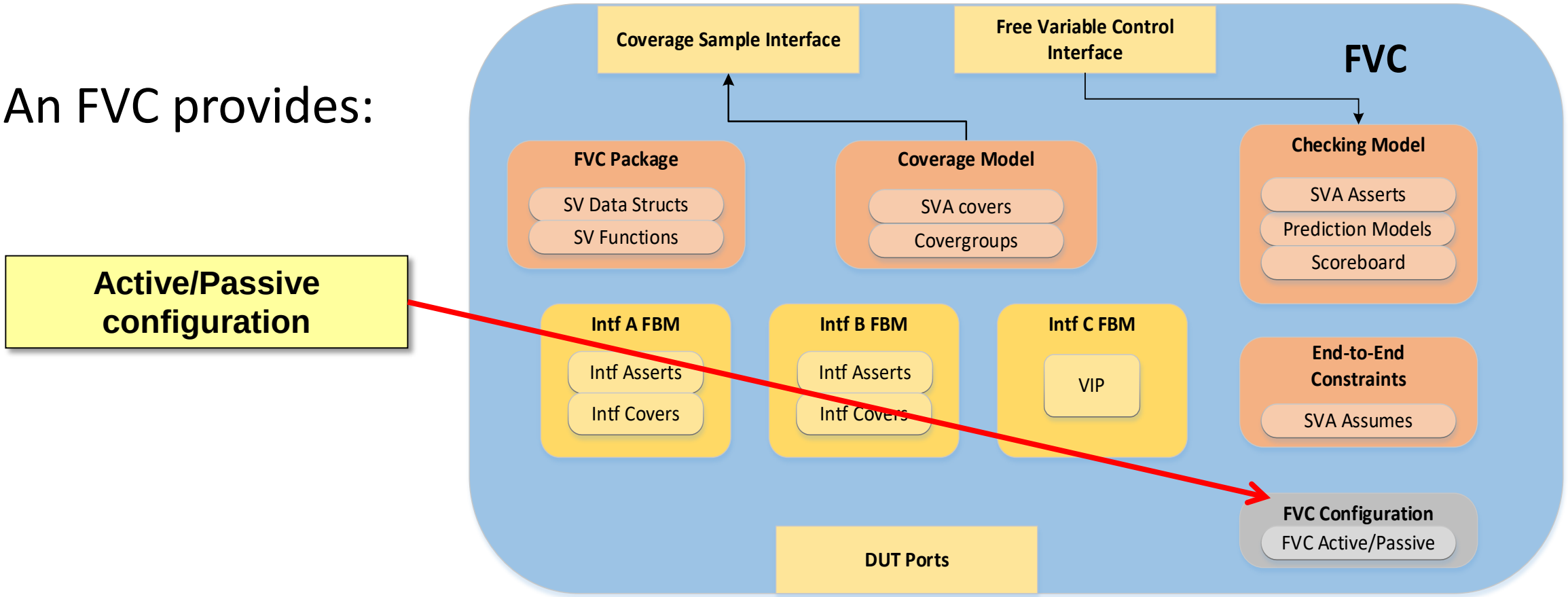
- An FVC provides:

A package with reusable data structs, functions, properties, etc.



Formal Verification Component (FVC)

- An FVC provides:



FVC Interfaces

- **DUT Ports**
  - Identical to design interfaces
  - Connect FVC to its DUT
- **Coverage Sample Interface**
  - A verification interface that enables coverage reuse
  - Propagate important block-level events to upper layer testbench
- **Free Variable Control Interface**
  - A verification interface that controls FV free variables
  - Free variables are commonly used in FV
  - Extra driver logic in simulation or emulation is needed

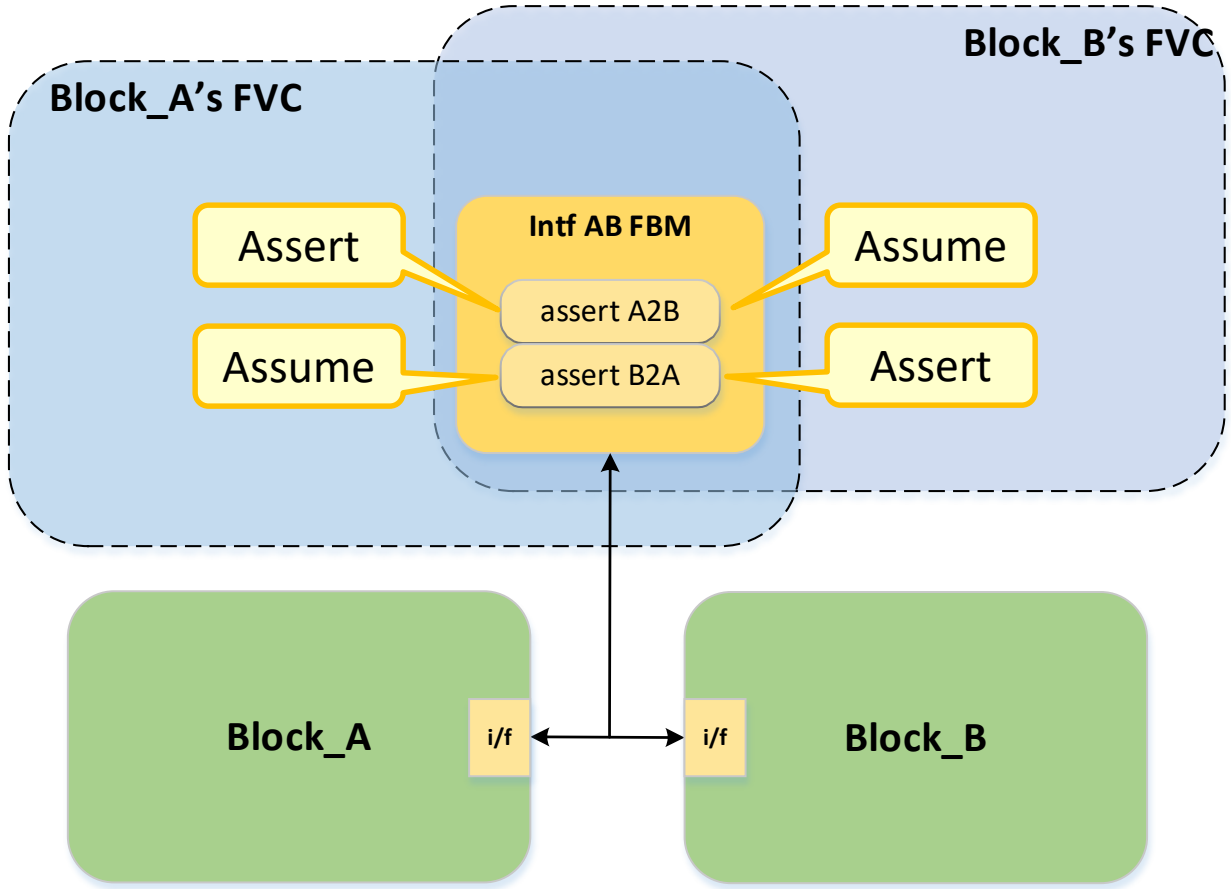
Free Variable Control Interface

- When *FVC_ACTIVE* = 1: this assume is in effect
- When *FVC_ACTIVE* = 0: this assume is disabled

```
interface blockA_free_var_ctrl_if
#(
    parameter bit FVC_ACTIVE        = 1,
    parameter int NUM_CLIENTS       = 8,
    parameter int NUM_CLIENTS_WIDTH = 3
);

  logic [NUM_CLIENTS_WIDTH-1:0]   client_idx;

  if (FVC_ACTIVE == 1) begin
     client_idx_stable: assume property (
        @ (posedge clk) disable iff (!rst_n)
        1 |-> ##1 ($stable(client_idx) && (client_idx < NUM_CLIENTS)));
  end

endinterface
```

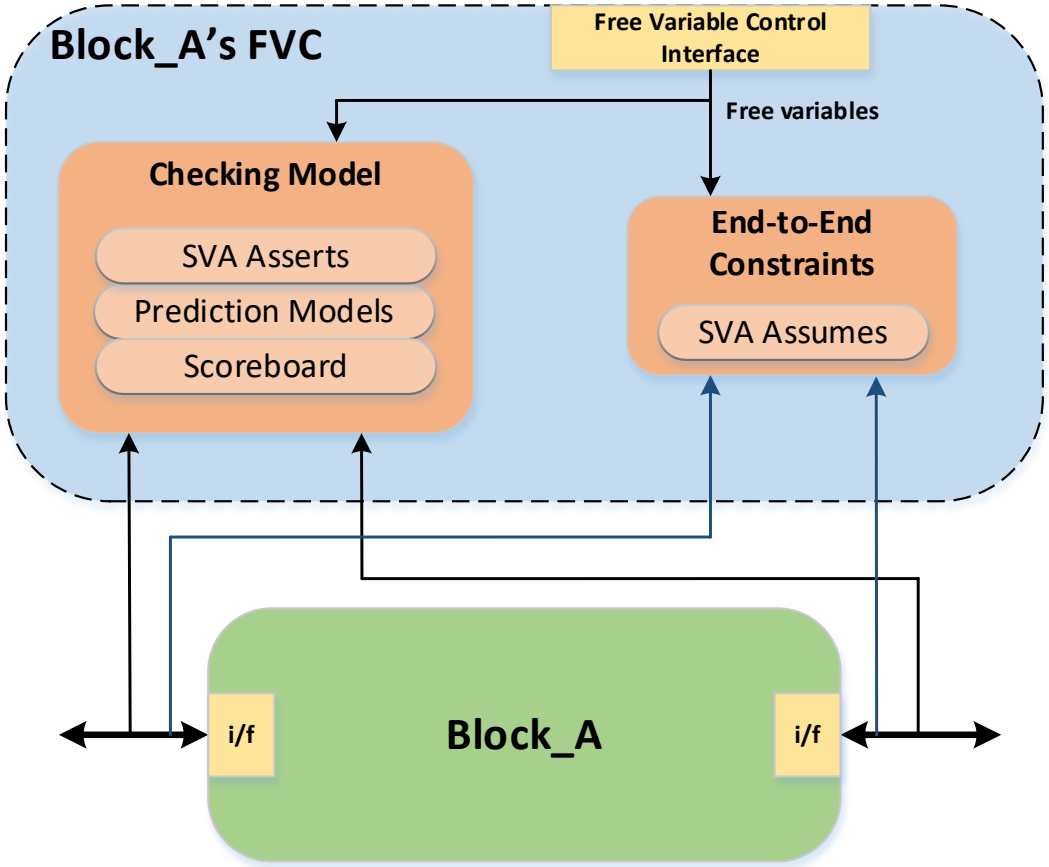
Formal Bus Model (FBM)

- Self-contained and reusable Assertion Based Verification IPs (ABVIPs)
- Bi-directional assertions and covers
- Cross proof at the interface



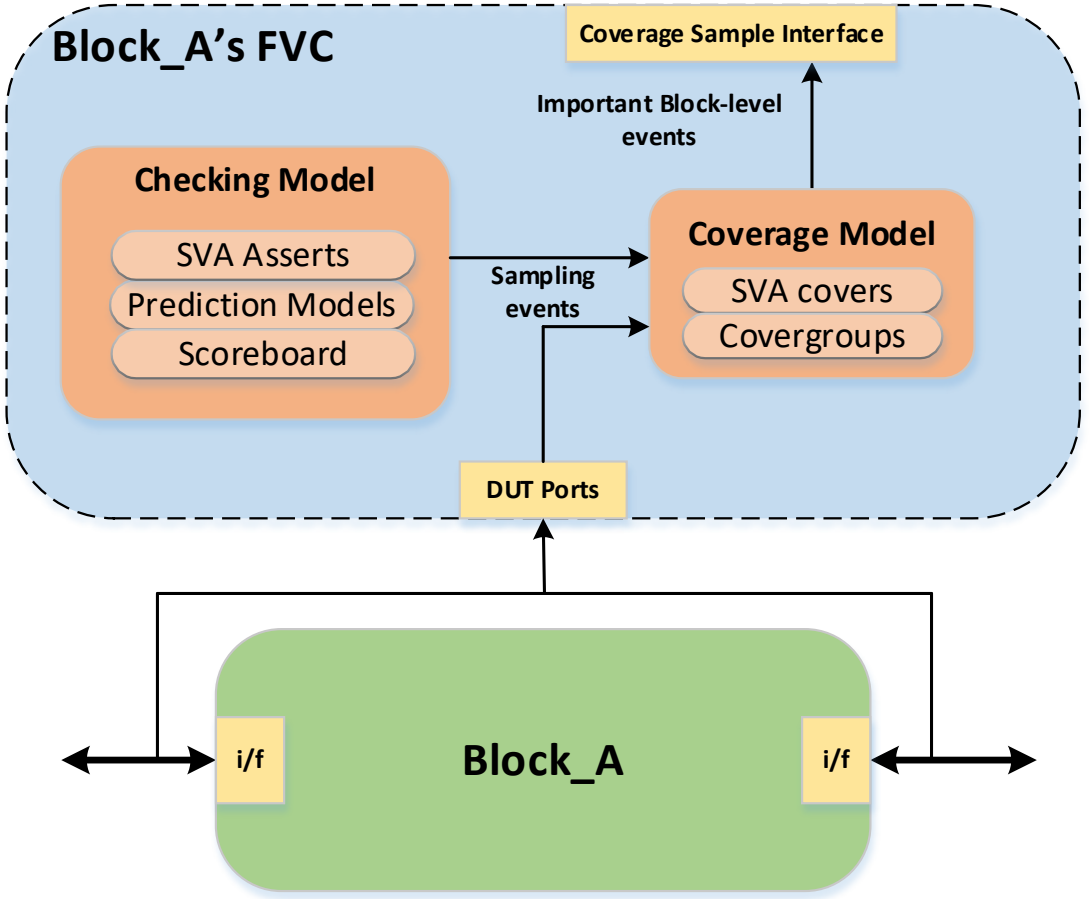
End-to-End FV Properties

- SVA assumes across multiple interfaces
- End-to-end modeling and SVA asserts
- Free variables in the verif interface



Formal Coverage Model

- Measure verification completeness
- Sampled when corresponding checkers trigger
- Certain events propagated for reuse



2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

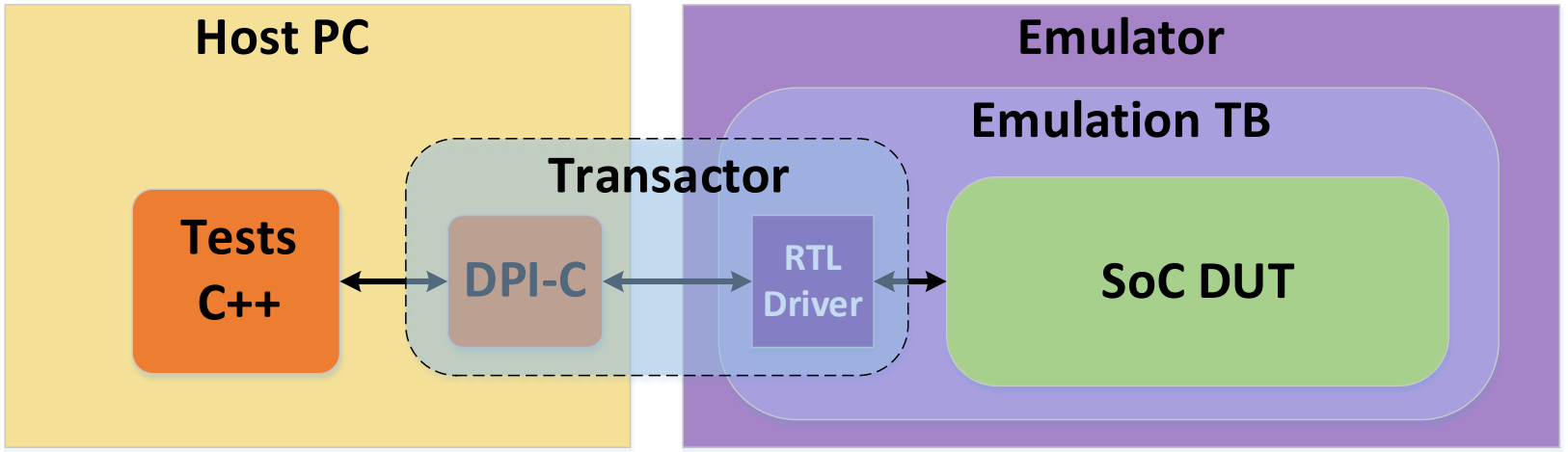
Agenda

- Introduction
- Reusable FV Testbench Structure
- **Integrating FV Collaterals into Emulation**
- Real-World Results: a SSD Controller SoC
- Conclusion

accellera SYSTEMS INITIATIVE

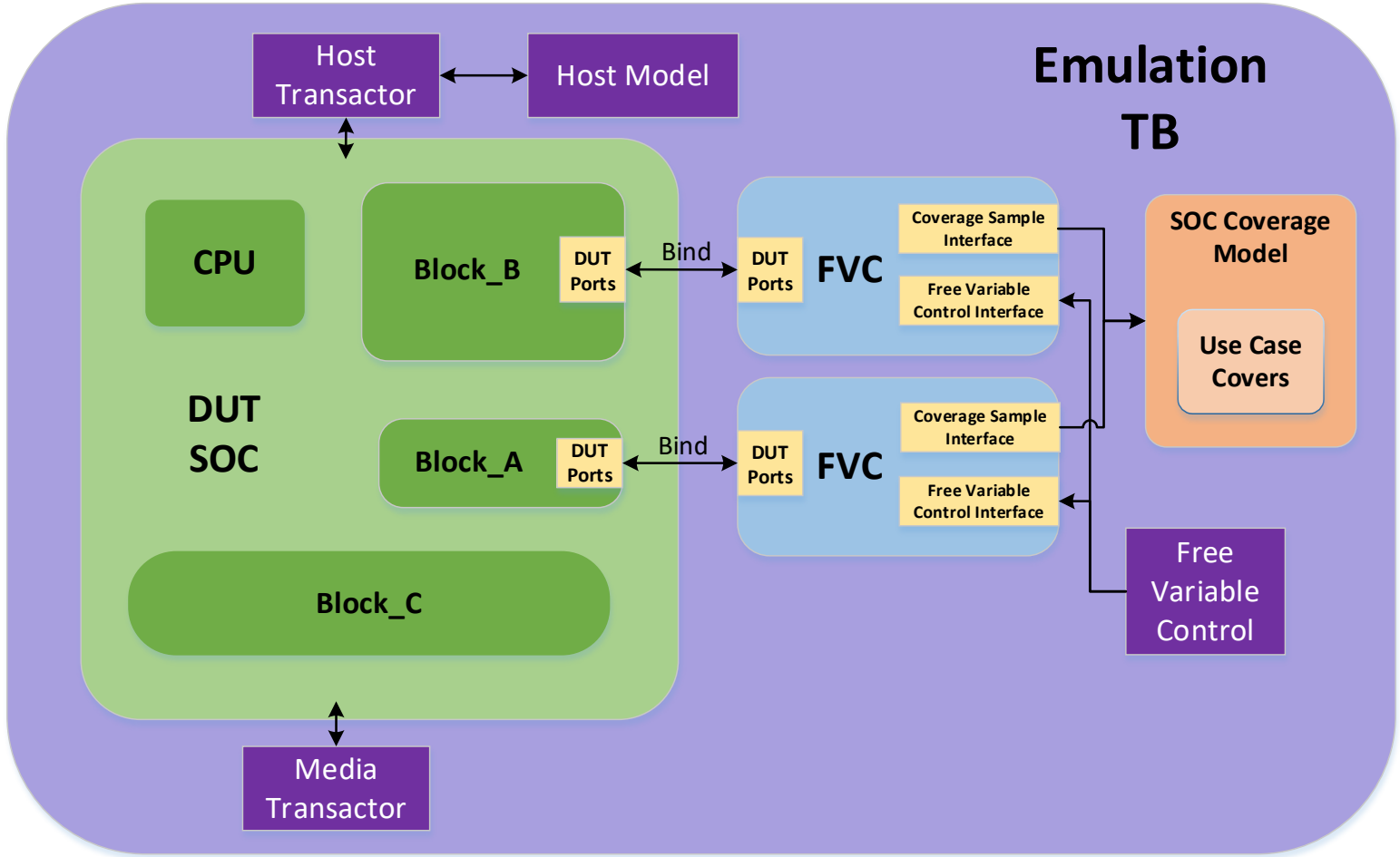
Transactor-Based Emulation

- Emulation enables system-level HW/SW co-verification
- Our transactor-based emulation platform contains:
  - **A SystemVerilog testbench** synthesized together with the SoC DUT
  - **System-level tests** in C++ language
  - **Transactors** that communicate information between the emulator and its host



Integrating FVC into Emulation

1. Bind each FVC (with *FVC_ACTIVE* = 0) to its DUT
2. Connect each FVC's coverage sample interface
3. Use a transactor to drive free variables



2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- Reusable FV Testbench Structure
- Integrating FV Collaterals into Emulation
- **Real-World Results: a SSD Controller SoC**
- Conclusion

accellera SYSTEMS INITIATIVE

Proof of Concept (PoC) Results

- PoC Experiment: Reusing two block FVCs in a SoC emulation platform

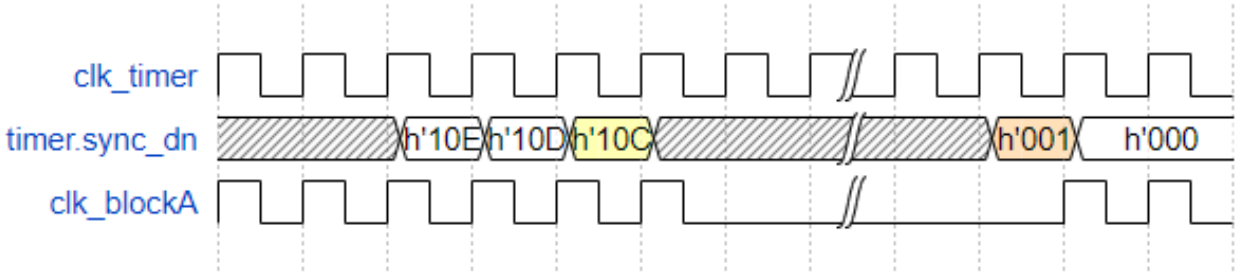
Quantitative Metrics	Return on Investment (RoI)		
	Block_A	Block_B	Total
Line of FV code reused	5392	1714	7106
# Assert/assume props reused	assert: 632 assume: 293	assert: 6 assume: 39	assert: 638 assume: 332
# Failed assert/assume props in emulation	assert: 10 assume: 2	assert: 1 assume: 2	assert: 11 assume: 4
# Block-level cover points reused	82	13	95
# System-level inter-block covers added		18	
DUT gate count (percentage)	0.8%	6%	6.8%
Cost: FVC gate count (percentage)	0.6%	0.02%	0.62%

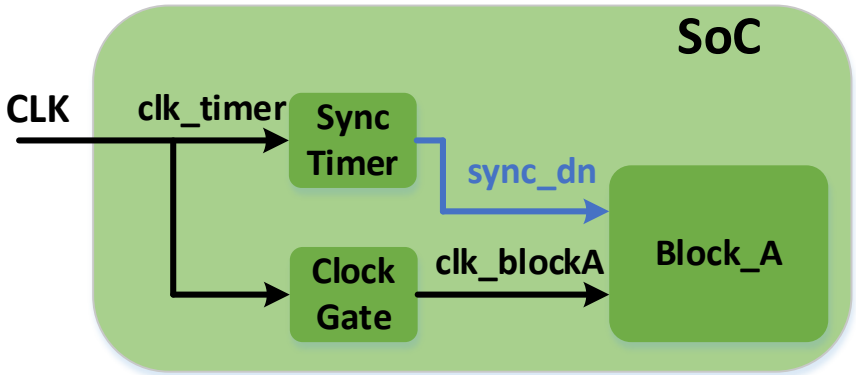
- ❑ **Overall reuse statistics:**
  - ✓ **~7000 LoC**
  - ✓ **~1000 Props**
- ❑ **Quality improvement (Block-level)**
- ❑ **Efficiency improvement (System-level)**
- ❑ **Manageable cost**

FVC Quality Improvement: An Over-constraint Case

- Very important to prove assumptions made at block boundaries
  - Pay attention to your clocks and resets!

```
Timer_decr: assume property (
   (sync_dn != `0 |-> ##1 sync_dn == $past(sync_dn)-1);
```





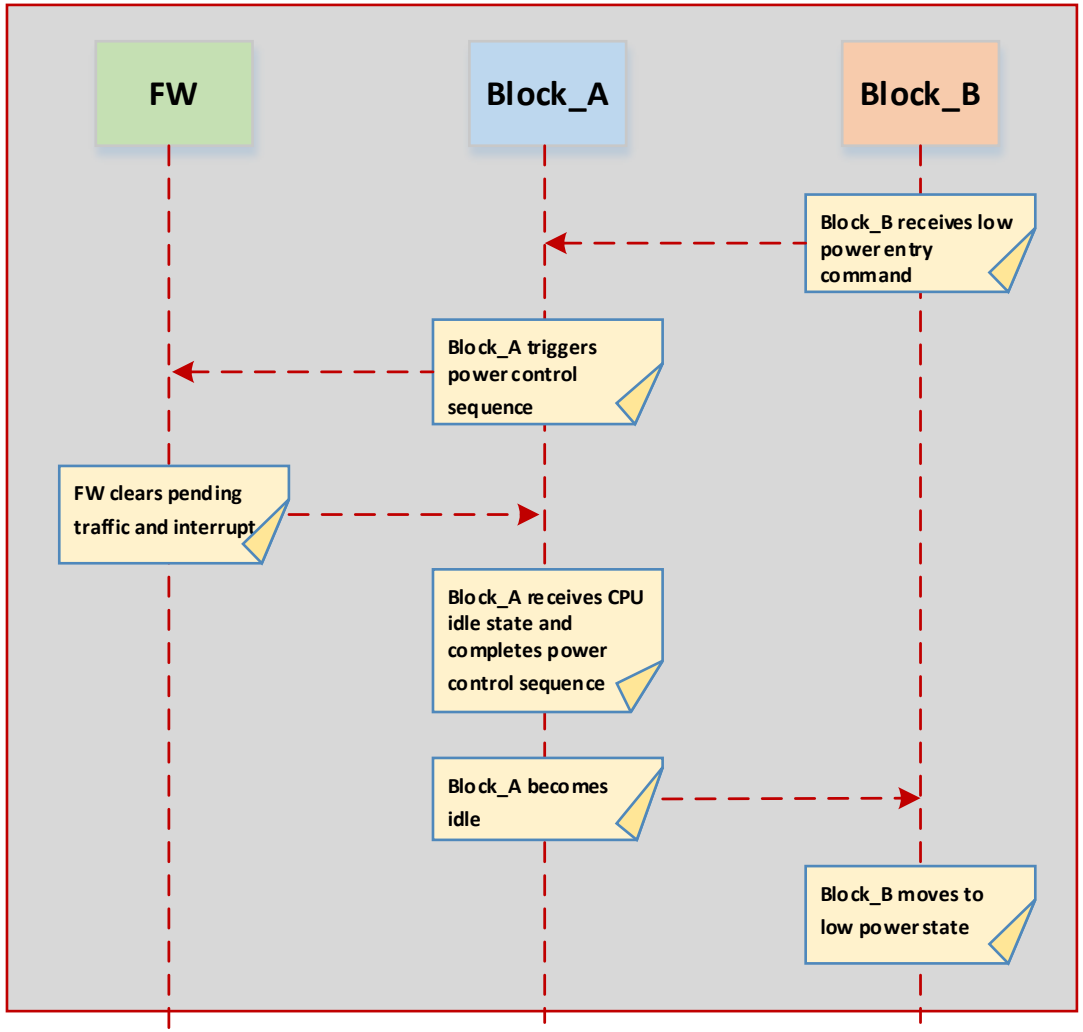
FVC Quality Improvement: FV Model Synthesized as Intended?

- Assertion *Instr_ptr_update*
  - passed in formal
  - failed in emulation
- Root cause: inferred latch!
- Pay attention to tool warnings!

```
Instr_ptr_update: assert property (
   (decode_fetch |-> ##1 instr_ptr ==
$past(instr_ptr_exp));

…

always_comb begin: model_ptr_gen

  if(!rst_n)
    instr_ptr_exp = 0;

  else if(decode_fetch) begin
    if(condition_1)
      instr_ptr_exp = '1;
    else if(condition_2)
      instr_ptr_exp = pending_ptr;
    else if(condition_3)
      instr_ptr_exp = fetch_address[13:0];
    end

end
```

Emulation Efficiency Improvement: A System-level Use Case Cover

- FVCs enable use case coverage:
  - Important block events
  - Inter-block event sequences
  - System-level use cases



Emulation Efficiency Improvement: Debuggability

- Embedded FVC checkers can greatly reduce debug effort!

Case 1: Reproduce A Post-Silicon Bug in Emulation

- Created an assertion for the RTL bug
- Reproduced system failure with the target assertion failed
- Rapid root cause analysis in Block_B

Case 2: Root Cause An Emulator Tool Bug

- Random system hanging due to a tool bug
- Root caused by an assertion in Block_A within a few days
- Saved weeks of debug time

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- Reusable FV Testbench Structure
- Integrating FV Collaterals into Emulation
- Real-World Results: a SSD Controller SoC
- **Conclusion**

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Conclusion

- Modern complex SoCs require a greater rigorous verification signoff methodology
  - Simulation
  - Formal
  - Emulation
- Our systematic formal reuse methodology helps to increase productivity
  - Efficiency: avoiding duplicate effort
  - Quality: cross-proof between different platforms

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Future Work

- Guidelines on how to create system-level use case covers
- Emulation-friendly scoreboard
- Building FVCs for more IPs/blocks

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

References

[1] Harry D. Foster, “The 2018 Wilson Research Group ASIC and FPGA Functional Verification Study”

[2] A. Li, H. Chen, J. K. Yu, E.L. Teoh, I. P. Anand, “A Coverage-Driven Formal Methodology for Verification Sign-off”, DVCon 2019

[3] J. Bromley, “Double the Return from your Property Portfolio: Reuse of Verification Asserts from Formal to Simulation”, DVCon 2015

[4] Erik Seligman, Tom Schubert, and M V A. Kiran Kumar, “Formal Verification, An Essential Toolkit for Modern VLSI Design,” Morgan Kaufmann, 2015

[5] N. Kim, J. Park, H. Singh, V. Singhal, “Sign-off with Bounded Formal Verification Proofs”, DVCon 2014

accellera SYSTEMS INITIATIVE