A Verification Synergy: Constraint-Based Verification

Carl Pixley
Advanced Technology Group
Synopsys, Inc.

John Havlicek
Motorola Inc., Austin

Verification Synergy

- **The object of (functional) verification is to**
  - Specify consistent (ideally, comprehensive & complete) model of behavior using
    - Golden model
    - Properties
  - Check compliance of implementation w. specification
    - Find bugs
      - Analyze/locate the cause of bugs
      - Correct bugs
    - Prove correctness
  - Measure coverage of verification plan and execution.

Verification Synergy

- **Cost-effective verification requires efficient use of resources:**
  - Information is a critical (the *most* critical) resource. Designer's time is very valuable!
    - "Capture once; use repeatedly."
  - Human resources
  - Compute resources
- **Tools should work together**
  - Why use two unrelated formats for expressing the same type of thing?
  - Example: why should simulation and formal verification use different formats for safety assertions? Why use different formats for constraints?

Verification Synergy

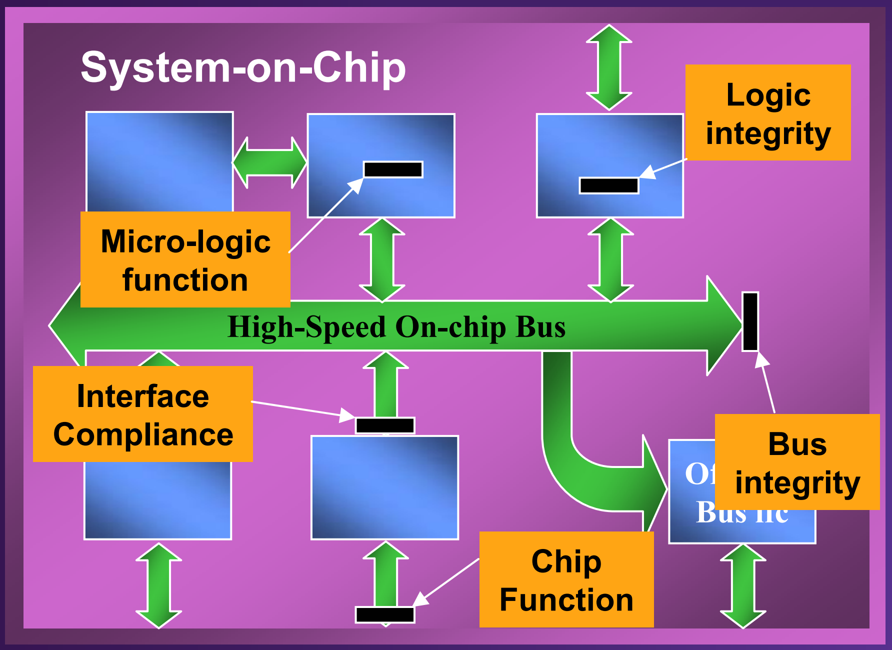
- **Bus functional model (BFM) / Testbench synergies**
  - Synthesizable: suitable for emulation
  - "Flip-able" I.e., equally suitable as drivers or monitors
  - Completeness possible.
  - Equally usable for simulation- / emulation-based verification as with formal verification (e.g., model checking)
  - Documentation / Formal description.
  - Support hierarchical (assume/guarantee) reasoning
  - Supports coverage analysis and simulation biasing.
  - Suitable for instruction/transaction level modeling?
  - Suitable for design synthesis optimization?

What is Constraint-Based Verification?

- Designers define constraints involving the inputs of their designs.
- They can immediately simulate their designs with constraints ONLY and debug wave forms. No testbench program is needed.
- Constraints and design mature incrementally.
- During integration constraints become monitors automatically. (Flipping) This supports assume/guarantee reasoning.

Constraint / Assertion-Based Methodology

Assertions (e.g., OVA, CBV) Verification



Use of Assertions

- Checking results
- Stimulus generation (Constraint assertions like SimGen)
- Proving correctness
- Measuring coverage
- Verification IP reuse

Reuse of Assertions Among Simulation, Semi-Formal, and Formal Verification

Constraint Examples

"Inputs 0, 1 & 2 are 0-1-hot"

```
In0 + In1 + In2 <= 1;
```

"A transaction start can only be asserted when the address state machine is in the idle state."

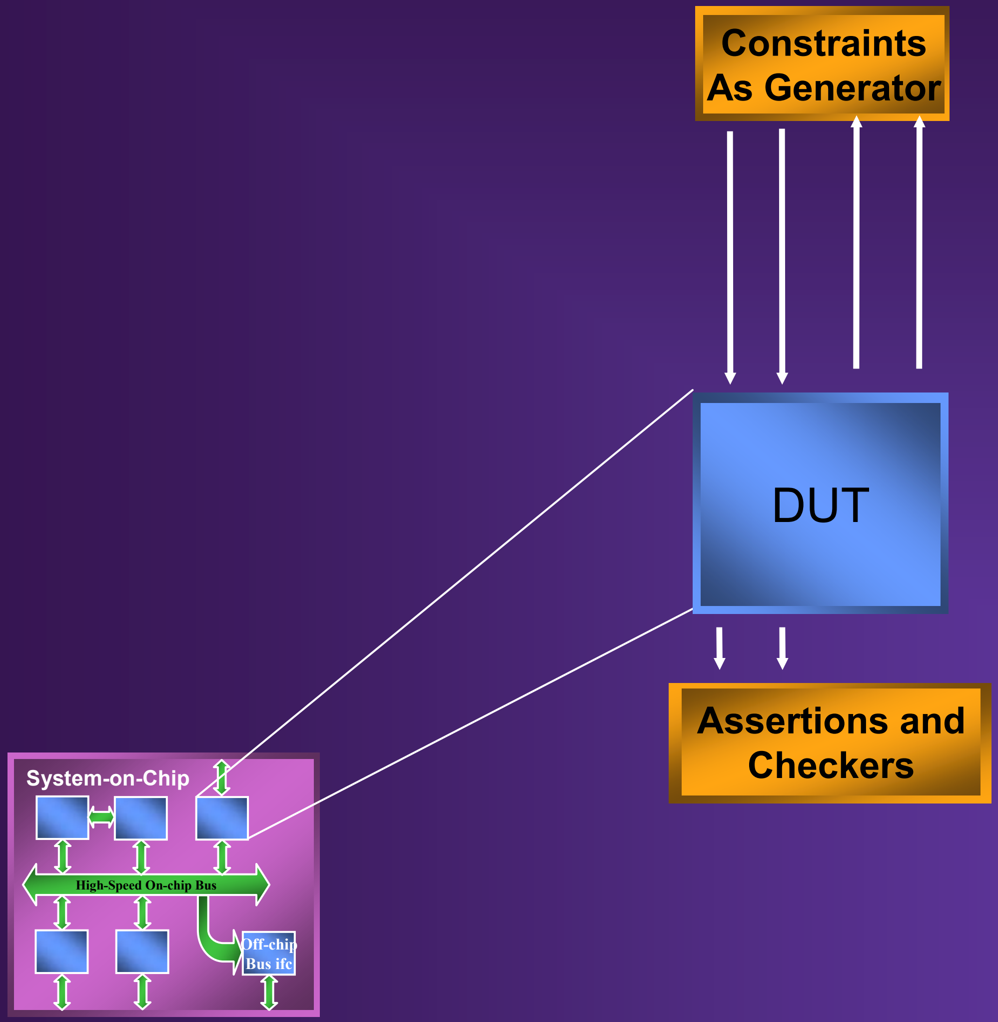
```
ts -> (addr_state = `ADDR_IDLE));
```

Constraints are just Verilog formulas. It works fine with OVA, TSP, Verilog or almost any assertion language.

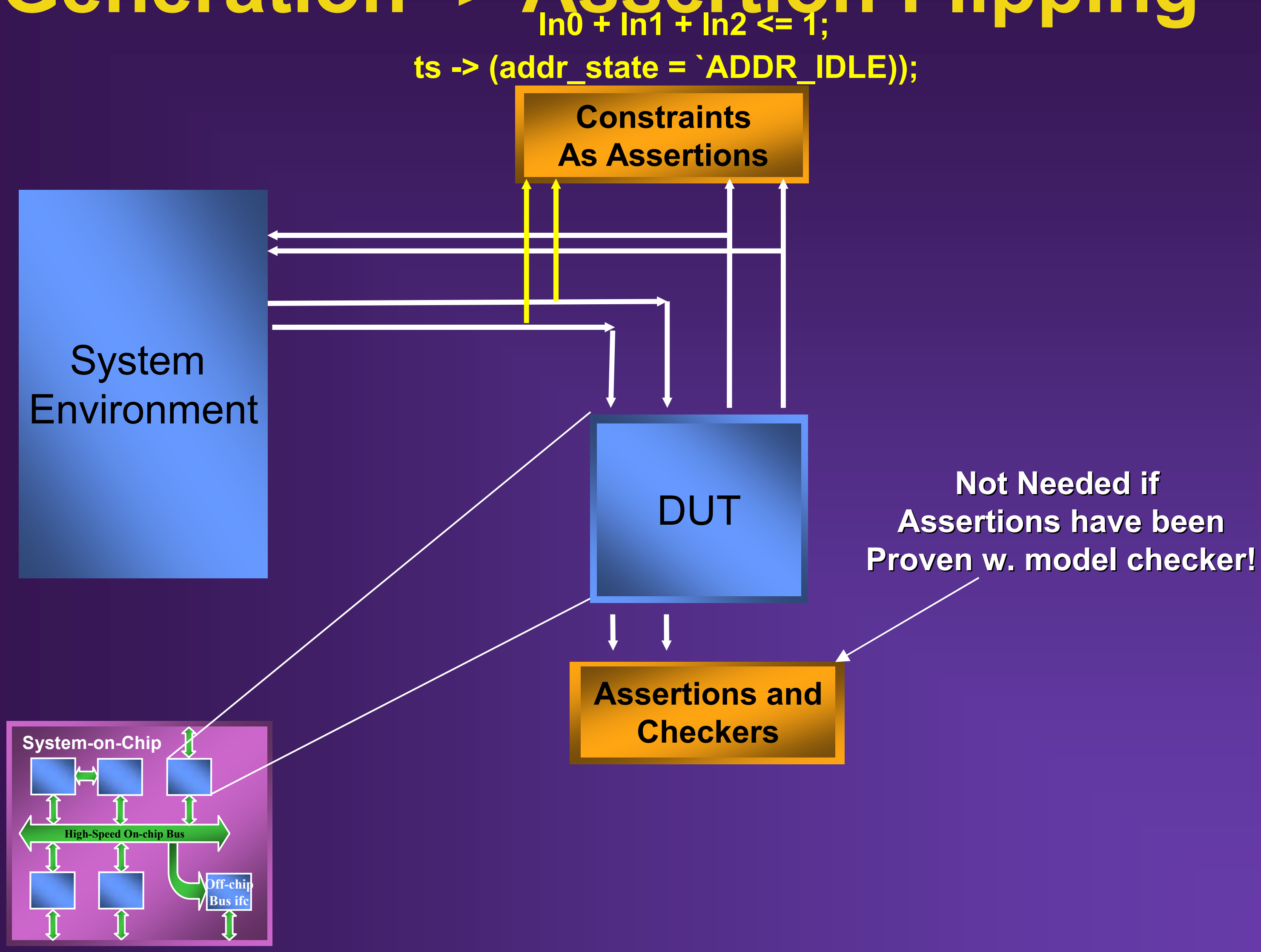
Generation

In0 + In1 + In2 <= 1;

ts -> (addr_state = `ADDR_IDLE));



Generation -> Assertion Flipping



Constraint-Based Verification

- **Enables early, more extensive use of assertion–based simulation at the unit level by designers!**
  - -- by lowering the effort to animate a design block and
  - by incrementally refining the logic and constraints

Constraint-Based Verification

- Design Manager:

"My proposal is for designers to test their logic before releasing it to the verification team. This will guarantee that we're not fighting careless/silly errors when the blocks are integrated in a system environment.

There are two reasons why I would like to follow the CBV [*SimGen*] route: 1) all the support you and your group have provided this past year and a half, and 2) I believe it would be easier for designers to use this tool than trying to learn the [conventional directed-random simulation] environment along with C++ and everything else."

Constraint-Based Verification

Low-effort, early animation of design blocks. The cost of getting started is low.

Designers don't have to write an elaborate test-bench to begin animating and debugging a block.

Because the development of environments for designs is incremental, the cost of developing constraint-based environments is amortized over time.

Constraint-Based Verification

Constraint-based verification integrates well with other, existing simulation approaches.

It can be integrated incrementally into a verification flow.

Constraints can be developed to monitor inputs in a directed or directed random approach. As constraints mature, they become simulation drivers (E.g., Automotive at Motorola).

Simulation & Formal methodology

- Constraints can be used both in simulation and formal verification (model checking).
- Constraint-based verification reinforces assertion-based verification (e.g., OVA – because constraints ARE assertions.
- Constraint-based simulation is unexpectedly effective in finding corner cases. (See slides below.)

Constraint-Based Verification

Reuse of constraint verification IP at the SoC level

1. Constraints can be used with model checking as environments.
2. Constraint-based generators can be easily **converted into checkers** during system integration.

Constraint-Based Verification

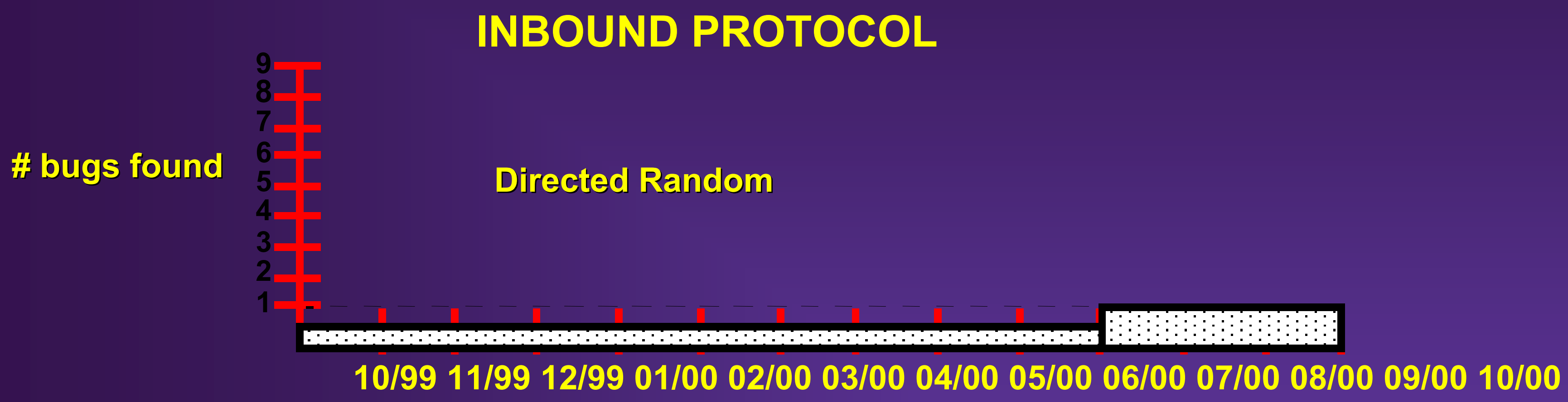
Constraint-based verification simulates **corner cases** of designs more effectively than other methods.

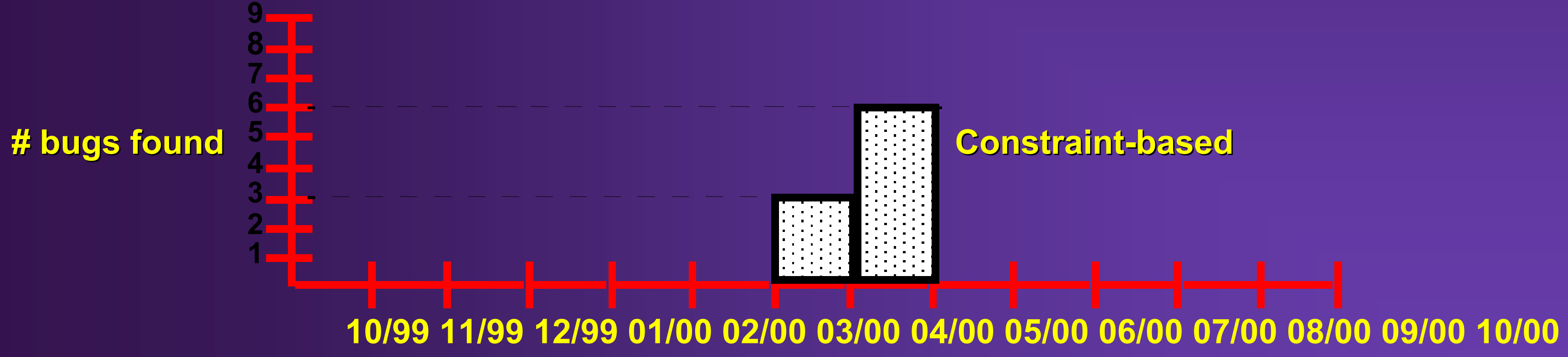
Constraint-based simulation **finds bugs earlier!**

Another **PPC** Design Manager:

> "The kind of bugs [CBV/SimGen user] has found in my logic are difficult to find in simulation. I do not believe we can guarantee a high quality first tapeout without [t]his work."

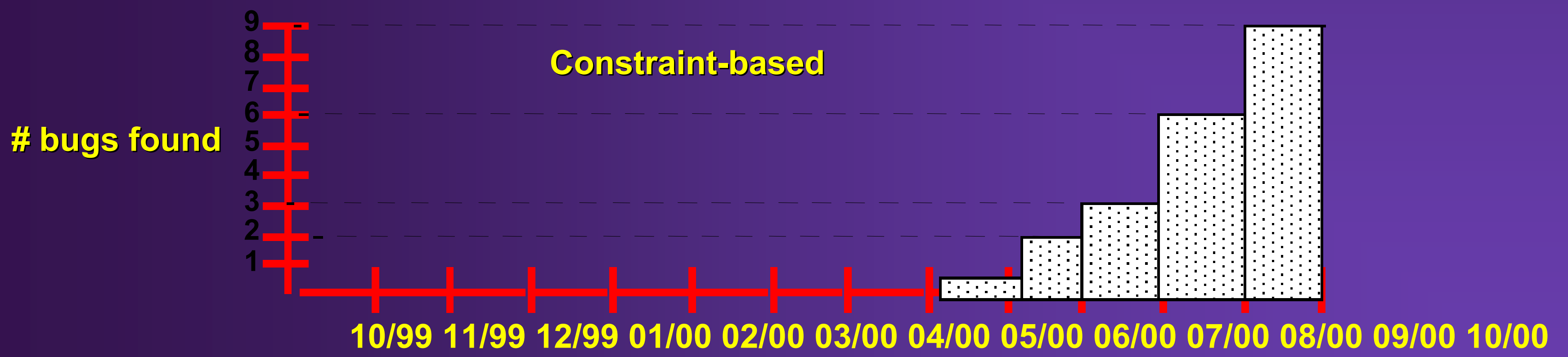
Directed-Random vs. Constrained-Random





Constrained-random vs. directed random

OUTBOUND - LOGIC LAYER



Summary

- **Verification Synergy is important for cost-effective verification, example:**
- **Constraint-Based Verification**
  - Provides early/easy animation of DUVs by designers -- without checkers, without stimulus driver programs, ….
  - Provides robust stimulus to exercise corner cases of design
  - Inputs can be “weighted” to bias simulation
  - Stimulus generation and checkers are dual concepts.

Summary (cont.)

- **Constraint-Based Verification**
  - Incrementally integrates into existing simulation environment.
  - Works with both simulation (VCS & Vera), formal tools and OVA.
  - Constraints can be used by designers directly and incrementally – broader market.
  - Constraint-based verification finds bugs faster than other methods.

End of Talk

Benefits

. Constraint-based verification can be put in the hands of designers at the module, block and unit levels of design. This implies a much broader user-base for formal and simulation tools.

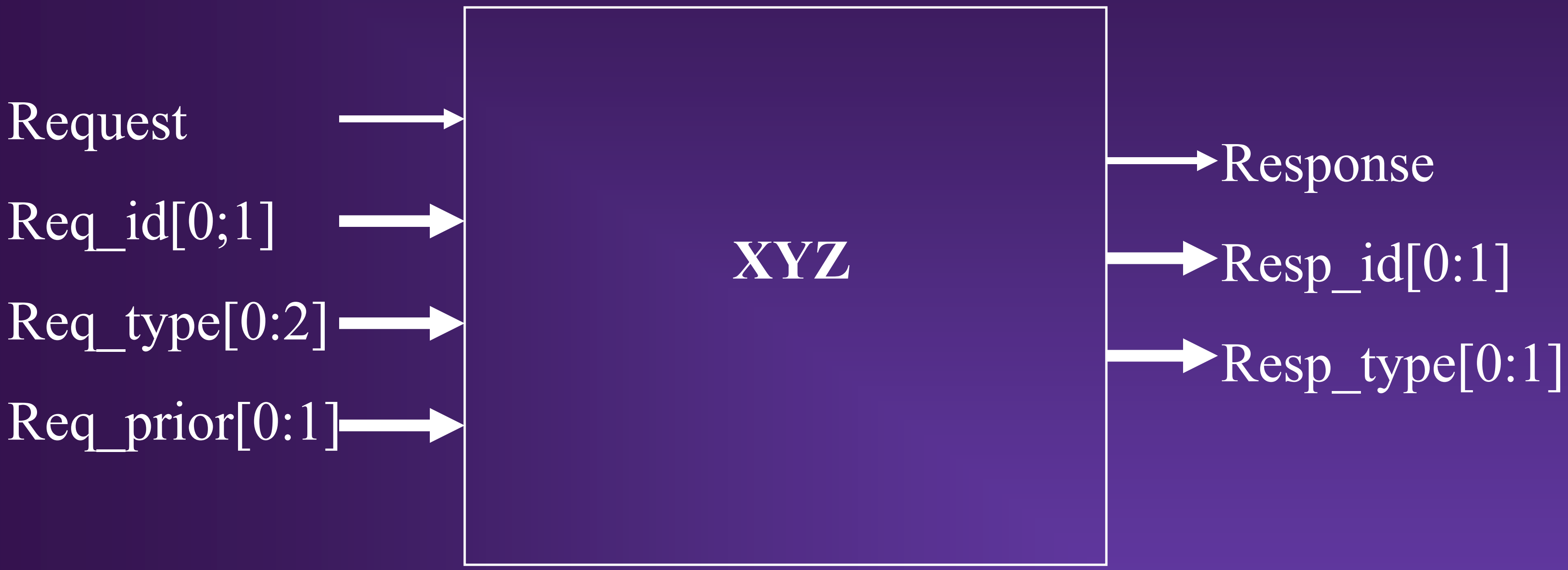
. Verification checkers are left all over the design to locate and isolate problems near the bug site.

. Constraints formally document interfaces to DUVs in a machine-readable way.

Observation

- **Complex** temporal assertions (full CTL, LTL) CANNOT be easily reused as stimulus generators.

Constraint Example



Assume: A request may be given only if its identifier is not equal to the identifier of any active transaction.

Constraint Example

```
module xyz;
function activate(id[0:1])[0:0]   = request  &
   (req_id == id) ;
function deactivate(id[0:1])[0:0] = response
   & (resp_id == id) ;
function active_next(id[0:1])[0:0] =
     (deactivate(id) ? 1'b0       :
       activate(id)   ? 1'b1       :
                    active[id]) ;
```

Constraint-based Verification

```
var active[0:3] =
    {active_next(0),
     active_next(1),
     active_next(2),
     active_next(3),
      } ;
constraint(request ? ~active[req_id] : 1'b1) ;
```

Constraint-based Verification

- User provides constraints as Boolean expressions involving state and inputs.
- User provides biasing for each variable.
- SimGen generates input vectors to simulator on each clock cycle by solving constraints -- all together.
- SimGen is non-backtracking!
- SimGen is constant cost for each cycle. The cost is linear data structures representing constraints (e.g. BDDs).

SimGen technical issues

- Keeping BDD size low
- Automatic identification of special constraints that can be handled separately
- Constraint fracturing
- Variable ordering
- Constraint prioritization
- Run-time constraint solving (e.g., Shimizu/Dill)

References

- C. Pixley, K. Shultz, J. Yuan, “Integrated Formal and Informal Design Verification of Commercial Integrated Circuits“, PDPTA, pp. 1061-1067, June 28, 1999.
- J. Yuan, K. Shultz, C. Pixley, H. Miller, A. Aziz, “Modeling Design Constraints and Biasing in Simulation Using BDDs”, ICCAD 1999
- J. Kukula and T. Shiple, "Building Circuits from Relations" CAV 2000
- K. Shimizu, D. L. Dill, and A. J. Hu. "Monitor-Based Formal Specification of PCI", FMCAD 2000, Austin, Texas.
- K. Shimizu, D. L. Dill, C-T. Chou, "A Specification Methodology by a Collection of Compact Properties as Applied to the Intel Itanium Processor Bus Protocol", CHARME 2001, Livingston, Scotland.
- M. Kaufmann, A. Martin, C. Pixley, “Design Constraints in Symbolic Model Checking”, CAV 1998: 477-487

References

- J. Yuan , A. Aziz, K. Albin, C. Pixley, "Faster Boolean Constraint Solving for Random Simulator-Vector Generation", ICCAD, 2002.
- K.Shimizu, D. Dill, "Deriving a Simulation Input Generator and a Coverage Metric from a Formal Specification", DAC 2002.
- J.Yuan, K. Albin, A.Aziz, C.Pixley, "Constraint Synthesis for Environmental Modeling in Functional Verification", (accepted) DAC 2003.

Common User Assertion Examples

- One-hot buses
- Full and parallel case synthesis pragmas
- Array accesses
- Bus contention
- Valid data not lost in stalled pipelines
- Low priority events eventually processed
- Requests handled within spec'd window
- Packet Valid signal asserted correctly