TESSOLVE

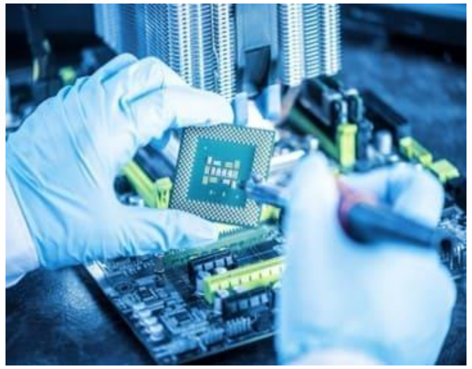
A HERO ELECTRONIX VENTURE

Formal Verification

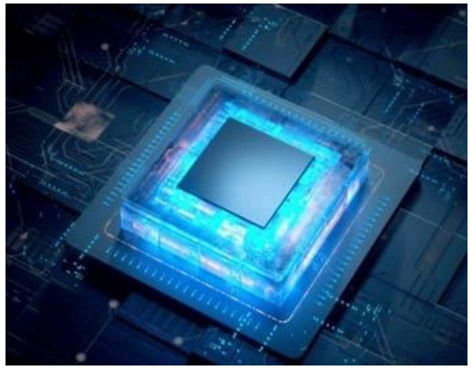
Savita Suresh Lohar

Eswaran Krishnan

Chip Design


Test Engineering

Hardware Design


Embedded Systems

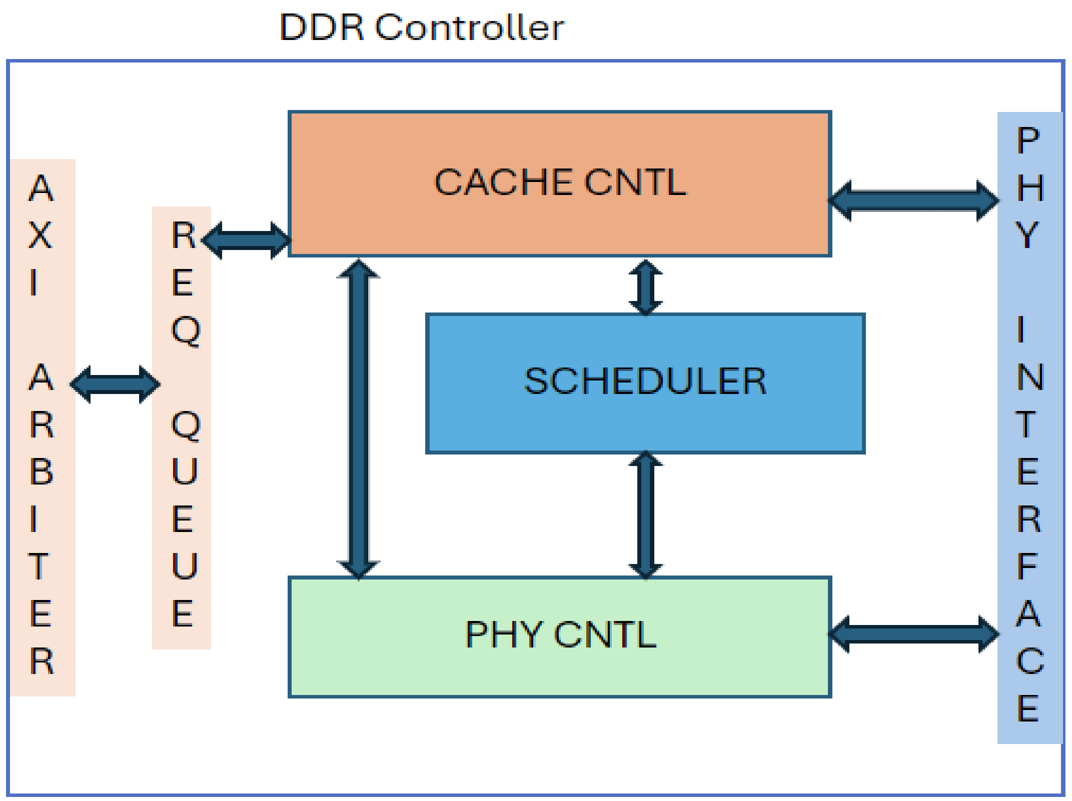
Agenda

- Top-Level Module Selection
- Lack of Assumption
- Simulation uncover scenarios covered
- Long run time assertions
- Multiple hierarchy used SVA divided into multiple SVA
- Catching RTL Bugs with Jasper
- Coverage
- Memory Issues with Jasper - Solution

Top-Level Module Selection

TESSOLVE
A HERO ELECTRONIX VENTURE

- **Top-Most Module**
- **Sub-Module**



Top-Level Module Selection

•**Top-Most Module**:
This approach involves selecting the highest-level module in your design hierarchy as the top-level module.
This module typically represents the entire system. Verifying properties at this level allows you to capture system-level behaviours and interactions.

•**Sub-Module**:
To focus on a specific sub-module within your design hierarchy as the top-level module.
This approach is useful when you want to verify properties related to a particular functional block or subsystem in isolation.
It helps analyzing specific behaviours or interactions without the complexity of the entire system.

Lack of Assumption

- Assertions represent properties that the design should satisfy under specific conditions or scenarios.
- Assumptions represent characteristics of the design, inputs to the design, or behavior of the environment in which the design operates
- When assertions fail, it means that the properties being verified rely on certain conditions or behaviors that are not explicitly assumed or verified

Lack of Assumption

- Let consider below Assumption was not provided by Designer

```
assume_cachedirty:     assume property (
                         rg_cachedirtymin < rg_cachedirtymax  );

assume_onehotack:      assume property (
                         $onehot(ack_ch1,ack_ch2,ack_ch3)  );
```

- Most of the checkers related to this cache functionality will fail in jasper FPV.
- This was not designing issue; we have to check all the input signals and condition related to cache dirty functionality then we come to know that we were lacking an assumption.

Lack of Assumption

- To address assertions failing due to a lack of assumption

  Identify the assumptions implicitly required by the properties being verified.

  Explicitly define these assumptions as part of the verification constraints.

  Ensure that the assumptions accurately reflect the expected behavior of the design or environment.

  Verify that the design satisfies these assumptions under all relevant scenarios.

Simulation uncover scenarios covered

- Let's consider a scenario where the cache is being accessed by multiple CPUs concurrently

```
property cache_coherence;
  @(posedge clk) disable iff (reset)
  // If a write operation occurs (write == 1),
  // then any subsequent read operation (read == 1) on the same address
  // should see the updated data (data_out == write_data).
  (write) |-> (read ##1 (data_out == write_data));
endproperty
```

- during simulation, we may have a test scenario where CPU 1 writes data to a specific memory location, followed by CPU 2 reading from the same location.

- Simulation might verify this successfully if the timing is right and the testbench stimuli trigger the necessary events in the correct order

Simulation uncover scenarios covered

- There could be scenarios that simulation might miss, such as rare timing-related race conditions or interleaving between CPU accesses that are not covered by the testbench

- These scenarios could lead to potential cache coherence violations that may go unnoticed during simulation.

- For example, FPV identify scenarios where CPU 1 and CPU 2 access the cache simultaneously, leading to unexpected cache states or data inconsistencies that violate the cache coherence property

- By leveraging FPV alongside simulation, verification engineers can complement the strengths of both methodologies, ensuring comprehensive verification coverage and exposing the potential issues that might be undetected

Long run time assertions

- Analyzing the interaction and behavior of different hierarchies requires the FPV tool to explore a vast state space, contributing to longer verification times.

- Let consider typical DDR memory controller, consists of high-performance bus arbiter, controller core with cache support, scheduler and phy controller.

- Example: Cache miss request

```
property ddr_read_req_check;
  @(posedge clk) disable iff (reset)
  (Axi_cmd_queue.rd_valid && cache.nohit ##[0:1] cache_ctrl.miss) |-> ##[0:2] scheduler.ddr_access_req
endproperty
```

Multiple hierarchy used SVA divided into multiple SVA

- By dividing the properties into smaller SVAs targeting specific hierarchies or components, we can simplify the verification process and focus on verifying one aspect of the design at a time.
- This approach allows us to manage the complexity of verifying properties across multiple hierarchies more effectively and lead to faster verification times.
- Example: Cache miss request

```
property ddr_read_req_miss_check1;
  @(posedge clk) disable iff (reset)
  (Axi_cmd_queue.rd_valid && cache.nohit |-> ##[0:1] cache_ctrl.miss)
endproperty

property ddr_read_req_miss_check2;
  @(posedge clk) disable iff (reset)
  (cache_ctrl.miss) |-> ##[0:2] scheduler.ddr_access_req
endproperty
```

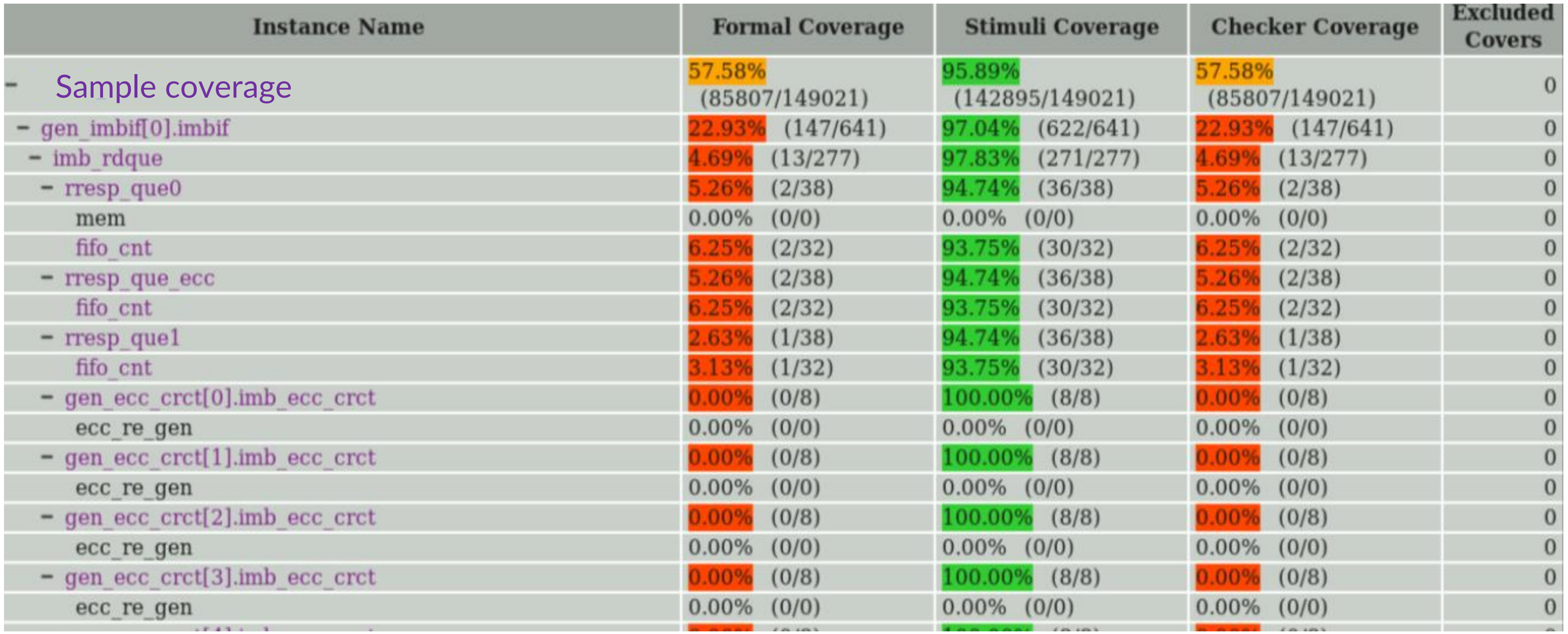
Catching RTL Bugs with Jasper

- Formal verification ensures correctness by mathematically proving that a system or software meets its specifications, eliminating ambiguity and interpretation errors.
- It can uncover subtle bugs and vulnerability that may not be easily detectable through traditional testing methods, leading to more robust and reliable systems.
- Insert assertions into RTL code to capture design bugs, these assertions act as checks during formal verification to detect violations of desired properties.
- Analyze coverage metrics to ensure that your formal verification tests cover sufficient portion of the design's functionality. This helps in detecting corner-case bugs that might otherwise go unnoticed during the simulations.
- When formal verification detects a violation or fails to prove a property, use debugging features provided by the verification tool i.e. **hunt command**

Coverage

- Coverage gives formal users metrics to measure progress and achieve signoff
- Extracts and displays valuable progress metrics, even for bounded (incomplete) proofs
- Helps avoid over-constraining the DUT
- Like simulation coverage, Jasper formal coverage provides support for both code and functional coverage, including covergroups.
- Formal-specific coverage types provide data on what coverage is being exercised by the formal testbench (“Stimuli Coverage”), and well as the completeness of the formal checks (“Checker Coverage”).
- Jasper coverage has ability to generate custom reports and file formats such as HTML and Text.

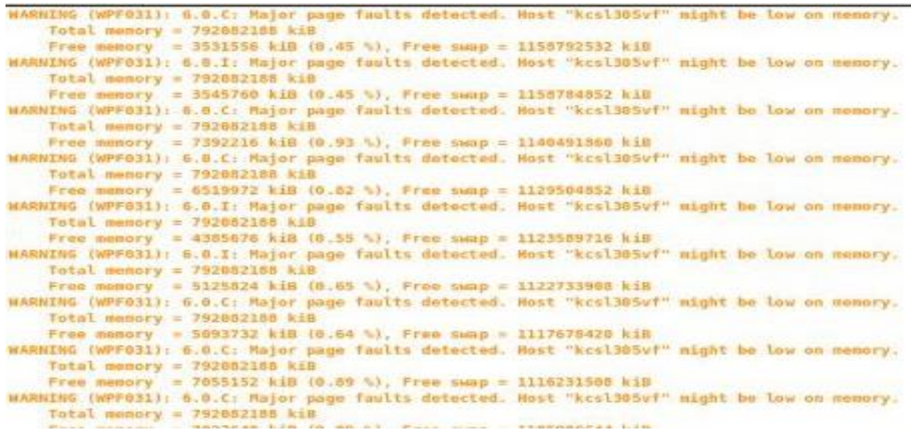
Coverage



Instance Name	Formal Coverage	Stimuli Coverage	Checker Coverage	Excluded Covers
− Sample coverage	57.58% (85807/149021)	95.89% (142895/149021)	57.58% (85807/149021)	0
− gen_imbif[0].imbif	22.93% (147/641)	97.04% (622/641)	22.93% (147/641)	0
− imb_rdque	4.69% (13/277)	97.83% (271/277)	4.69% (13/277)	0
− rresp_que0	5.26% (2/38)	94.74% (36/38)	5.26% (2/38)	0
mem	0.00% (0/0)	0.00% (0/0)	0.00% (0/0)	0
fifo_cnt	6.25% (2/32)	93.75% (30/32)	6.25% (2/32)	0
− rresp_que_ecc	5.26% (2/38)	94.74% (36/38)	5.26% (2/38)	0
fifo_cnt	6.25% (2/32)	93.75% (30/32)	6.25% (2/32)	0
− rresp_que1	2.63% (1/38)	94.74% (36/38)	2.63% (1/38)	0
fifo_cnt	3.13% (1/32)	93.75% (30/32)	3.13% (1/32)	0
− gen_ecc_crct[0].imb_ecc_crct	0.00% (0/8)	100.00% (8/8)	0.00% (0/8)	0
ecc_re_gen	0.00% (0/0)	0.00% (0/0)	0.00% (0/0)	0
− gen_ecc_crct[1].imb_ecc_crct	0.00% (0/8)	100.00% (8/8)	0.00% (0/8)	0
ecc_re_gen	0.00% (0/0)	0.00% (0/0)	0.00% (0/0)	0
− gen_ecc_crct[2].imb_ecc_crct	0.00% (0/8)	100.00% (8/8)	0.00% (0/8)	0
ecc_re_gen	0.00% (0/0)	0.00% (0/0)	0.00% (0/0)	0
− gen_ecc_crct[3].imb_ecc_crct	0.00% (0/8)	100.00% (8/8)	0.00% (0/8)	0
ecc_re_gen	0.00% (0/0)	0.00% (0/0)	0.00% (0/0)	0

Memory Issues with Jasper - Solution

- In Formal verification, based on design size formal tool will exercise the design exhaustively with required more memory which leads to memory issues
- While capturing the coverage percentage of bigger design there are possibilities to get the memory issues and tool will not be able to collect the entire coverage database, again which leads to incomplete of data.



```
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 3531556 kiB (0.45 %), Free swap = 1158792532 kiB
WARNING (WPF031): 6.0.I: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 3545760 kiB (0.45 %), Free swap = 1158784852 kiB
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 7392216 kiB (0.93 %), Free swap = 1140491860 kiB
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 6519972 kiB (0.82 %), Free swap = 1129504852 kiB
WARNING (WPF031): 6.0.I: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 4385676 kiB (0.55 %), Free swap = 1123589716 kiB
WARNING (WPF031): 6.0.I: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 5125824 kiB (0.65 %), Free swap = 1122733908 kiB
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 5093732 kiB (0.64 %), Free swap = 1117678420 kiB
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
    Free memory  = 7055152 kiB (0.89 %), Free swap = 1116231508 kiB
WARNING (WPF031): 6.0.C: Major page faults detected. Host "kcsl305vf" might be low on memory.
    Total memory = 792082188 kiB
```

Solution

- User can avoid memory issue by specifying the required memory size based on the design's complexity as follow,

- **set_proofgrid_shell {/common/N1GEtool/RBS/bin/bs -n 8 -J %jg_session_id -M 32000}**

  where –M specifies the memory size

THANK YOU

Any Questions?