

Formal Verification of a Multistage Arbiter

Using VCFormal,
New Formal Verification tool from Synopsys

Shahid Ikram
Cavium Inc.
Marlboro, MA, USA

September 22, 2015
SNUG Boston



Agenda

Overview

The Design

Formal Verification

Are we Done?

Epilogue

Overview

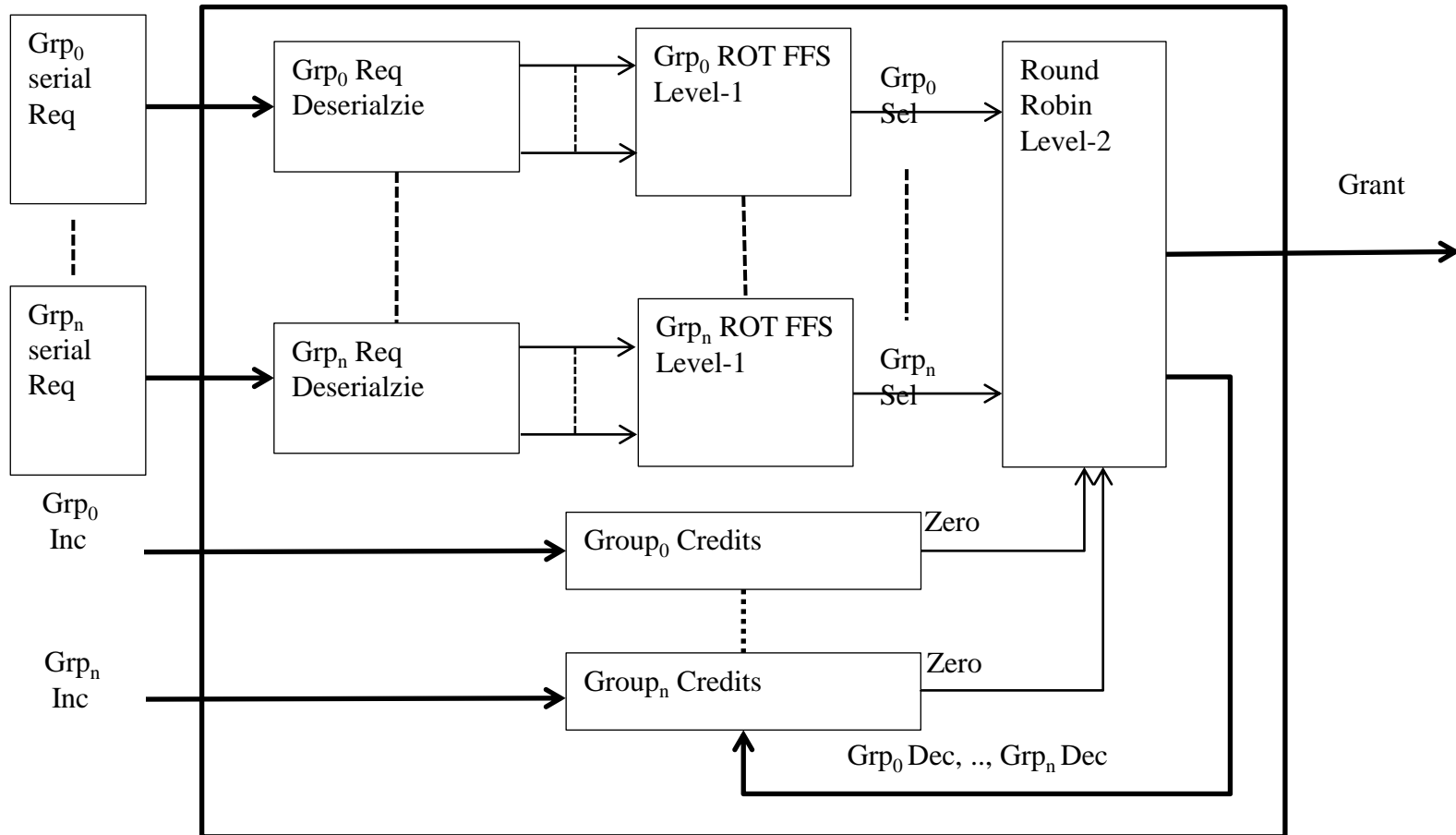
- Arbitration schemes/algorithms are commonly occurring elements in the Digital Systems.
- Generally these schemes and the algorithms are pretty complex to fathom.
- Simulation cannot verify them exhaustively.
- Design of the conditions(constraints etc.) required to catch the cases like “ request starvation” is next to impossible.
- This is a scenario where Formal Verification (FV) can help as it is exhaustive and can formulate starvation but only if:
 - The design can fit into formal space.
 - Formal constraints to the design are known.
 - Design specifications are available in a readable format.

The Design

A Multistage Arbiter
Verification Challenge



A Multistage Arbiter



Verification Challenge

- Arbiters are used to provide access to the shared resources.
- We generally want an Arbiter to be fair in providing access unless it is purposefully designed in a different way.
- There are three definitions of fairness:
 - Weak fairness, Strong fairness, FIFO fairness.
- A multistage arbitration scheme can be locally fair but globally unfair.
- Here is what we tried to prove:
 - *“As long as the system is eventually returning credits and making requests infinitely, our multi-stage arbiter issues grants for same priority requests with strong fairness.”*

Formal Verification

Setup

Environment

AEP

Arbiter Properties

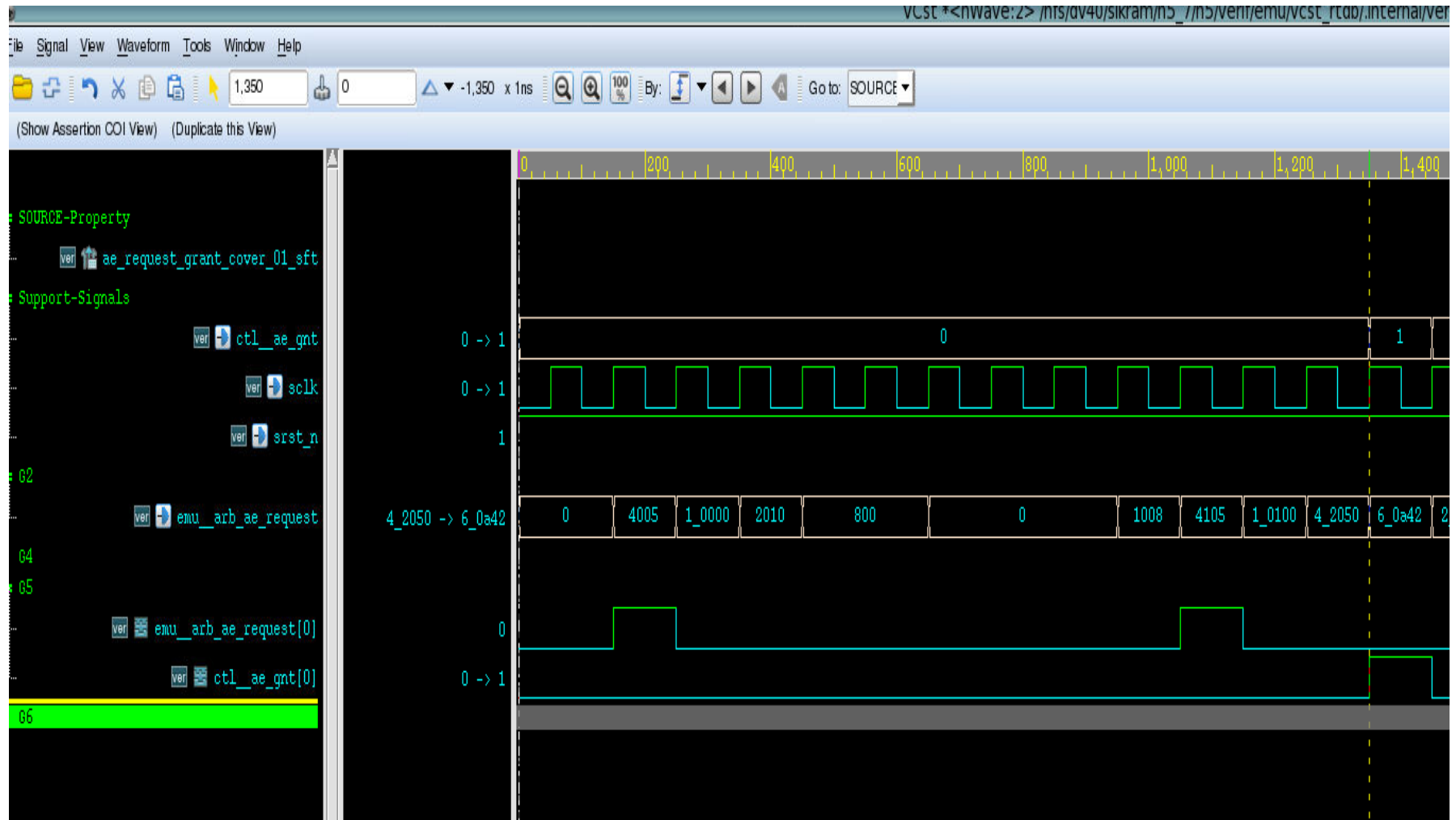


- The design-setup puts a simplified but correct design in a valid legal state from where we can start formal verification without problem of false failures. For this we need to know:
 - **Design Clocks:** An essential component for analysis.
 - **Reset Signals:** When these signals are asserted, analysis is not valid.
 - **Constants during reset simulation:** Unchanged signals during reset simulation.
 - **Constants during formal analysis:** Signals not needed in formal analysis like BIST, scan, fusechain et.c
 - **Set of abstractions:** A set of behaviors that is not needed for formal analysis and can be ignored.
 - Methods for generating and saving initial state.

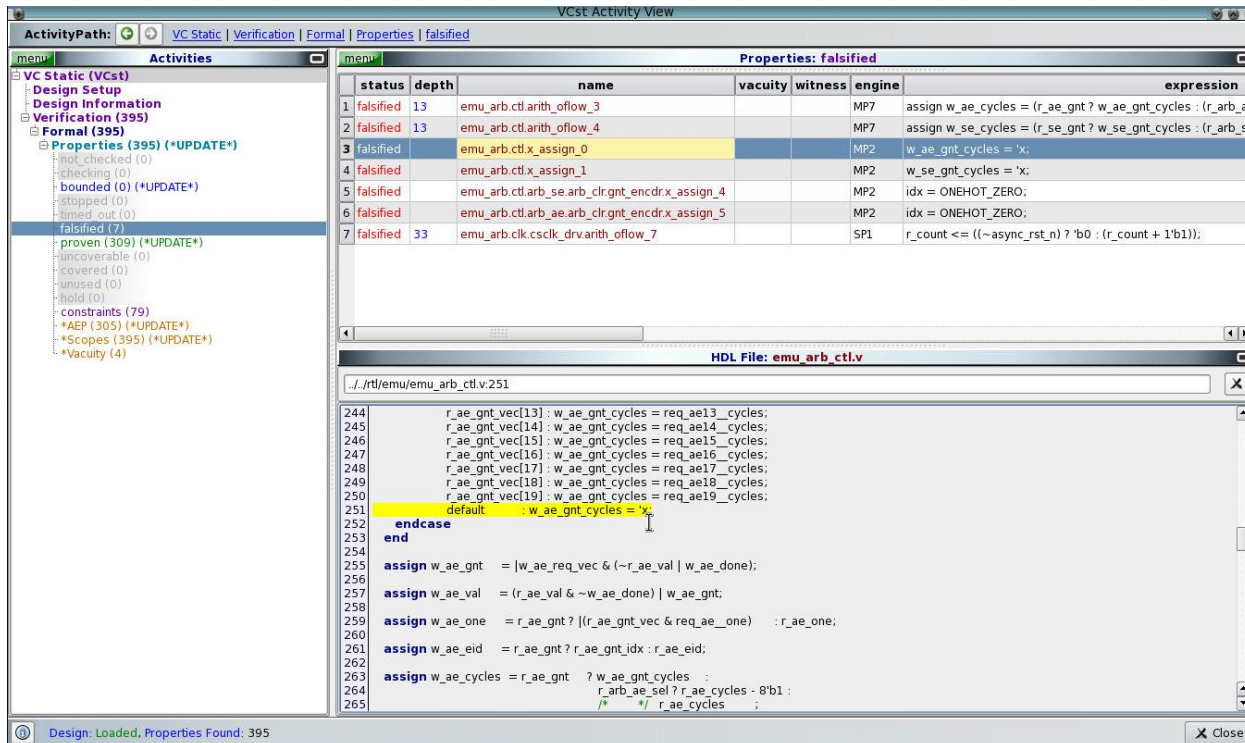
Environment

- Provides the legal stimuli to the design.
- Similar to random constraints in simulation.
- Challenges are:
 - Over constraints
 - Under constraints
- Here is what we force through our environment
 - Randomly generated requests for all groups with random number of credits.
 - Only one pending request per agent at any time.
 - Consume grants and credits.
 - Return consumed credits after a random delay.
 - Properly formatted requests.

Environment



- Automatically generated properties through the structural analysis of the design.
- Ideal for the testing and debugging a new environment.
- Good initial insight of the design.



The screenshot shows the VCst Activity View window. The left pane displays a tree view of activities, with 'Properties (395) (*UPDATE*)' selected. The right pane shows a table of properties with columns: status, depth, name, vacuity, witness, engine, and expression. The table lists 7 properties, all marked as 'falsified'. The bottom pane shows the HDL file 'emu_arb_ctl.v' with lines 244 to 265 visible.

status	depth	name	vacuity	witness	engine	expression
falsified	13	emu_arb_ctl.arith_oflow_3			MP7	assign w_ae_cycles = (r_ae_gnt ? w_ae_gnt_cycles : (r_arb_ae
falsified	13	emu_arb_ctl.arith_oflow_4			MP7	assign w_se_cycles = (r_se_gnt ? w_se_gnt_cycles : (r_arb_se
falsified		emu_arb_ctl.x_assign_0			MP2	w_ae_gnt_cycles = 'x';
falsified		emu_arb_ctl.x_assign_1			MP2	w_se_gnt_cycles = 'x';
falsified		emu_arb_ctl.arb_se_arb_clr_gnt_encdr_x_assign_4			MP2	idx = ONEHOT_ZERO;
falsified		emu_arb_ctl.arb_ae_arb_clr_gnt_encdr_x_assign_5			MP2	idx = ONEHOT_ZERO;
falsified	33	emu_arb_clk_cclk_drv.arith_oflow_7			SP1	r_count <= ((~async_rst_n) ? 'b0 : (r_count + 1'b1));

```

244 r_ae_gnt_vec[13] : w_ae_gnt_cycles = req_ae13_cycles;
245 r_ae_gnt_vec[14] : w_ae_gnt_cycles = req_ae14_cycles;
246 r_ae_gnt_vec[15] : w_ae_gnt_cycles = req_ae15_cycles;
247 r_ae_gnt_vec[16] : w_ae_gnt_cycles = req_ae16_cycles;
248 r_ae_gnt_vec[17] : w_ae_gnt_cycles = req_ae17_cycles;
249 r_ae_gnt_vec[18] : w_ae_gnt_cycles = req_ae18_cycles;
250 r_ae_gnt_vec[19] : w_ae_gnt_cycles = req_ae19_cycles;
251 default : w_ae_gnt_cycles = 'x';
252 endcase
253 end
254 assign w_ae_gnt = |w_ae_req_vec & (~r_ae_val | w_ae_done);
255
256 assign w_ae_val = (r_ae_val & ~w_ae_done) | w_ae_gnt;
257
258 assign w_ae_one = r_ae_gnt ? (r_ae_gnt_vec & req_ae_one) : r_ae_one;
259
260 assign w_ae_eid = r_ae_gnt ? r_ae_gnt_idx : r_ae_eid;
261
262 assign w_ae_cycles = r_ae_gnt ? w_ae_gnt_cycles :
263   r_arb_ae_sel ? r_ae_cycles - 8'b1 :
264   /* */ r_ae_cycles ;
265
  
```

Arbiter Properties

- Exclusive Grants

```
property mutex_grant_p(gnt);  
    @(posedge clk) disable iff (reset)  
        $onehot0(gnt);  
endproperty
```

- Minimum Latency

```
property request_grant_latency_property(req,gnt,i);  
    @(posedge clk) disable iff (reset )  
        !( req[i]&& gnt[i] ) ;  
endproperty
```

Arbiter Properties

- Fairness

```
property fairness_property(req0,pri0,gnt0,req1,pri1,gnt1);  
  @(posedge clk) disable iff (reset)  
    req0 && req1 && ( pri0 == pri1) | =>  
        (gnt1 | => ~gnt1 until_with gnt0);  
endproperty
```

- Grant without Request

```
property no_grant_without_request_property(req,gnt);  
  @(posedge clk) disable iff (reset)  
    Gnt | => (~gnt throughout (request) [->1]);  
endproperty
```

Arbiter Properties

- Starvation and liveness
 - It is hard to prove a liveness property(== Anti-starvation property) with out a fairness constraint.
 - Our fairness constraint is:
 - “Every requester generates non-zero credits requests infinitely often.”

```
property fairness_assumption;  
  @(posedge clk) disable iff (reset)  
  s_eventually  
  ((start_of_req && (credits!=0 )) != 0 );  
endproperty
```

```
property request_grant_liveness_property(req,gnt);  
  @(posedge clk) disable iff (reset)  
  req | => s_eventually grant;  
endproperty
```

```
property request_grant_bounded_liveness_property(req,gnt);  
  @(posedge clk) disable iff (reset)  
  req |-> ##eventually[1:100] grant;  
endproperty
```


Are we Done?

Assertion Coverage

Assertion Completeness



Assertion Coverage

- A metric to give some sense of the quality of the work.
- Provides a measure if the design is sufficiently exercised by our assertions.
- For our design, here is the report from Synopsys tool called VCFormal.

analyze_fv_coverage report summary:

Number of instances analyzed: 213
Number of registers analyzed: 542
Overall coverage score: 89/100

- More similar to code coverage in simulation than functional coverage.

Assertion Completeness

- Formal verification is exhaustive but not necessarily complete.
- First and the most important method is “designer review”.
- Second method is proving the completeness theorem.
 - Needs too many man-hours to furnish.
- Another method is using fault injection like Certitude.
 - We did fault injection manually.
 - We played with arbitration and priority logic to see if we are catching these bugs.
 - It was not comprehensive but gave us a higher level of confidence.
 - We are trying Certitude now.

Epilogue

Results

Conclusions



Results

- We found a starvation case.
 - “If arbiter gets a low-priority request in a mix of the high-priority requests, its round-robin will wait until all the high-priority requests are serviced.”
 - Designer added a timer to elevate the priority of any low-priority request if it is starving for too long.
 - We validated the fix.
 - The reason it was never found before because the arbiter was being used in an environment with few high-priority requests.
- We found an issue with the priority scheme.
 - Priority scheme does not work if a high and a low priority request come at the same time.
 - It takes a cycle to priority scheme become effective.

Conclusions

- A complete case study of formal verification of an arbitration scheme.
- We found cases that were never found by the simulation.
- The effort enhanced our understanding of the design.
- Formal verification is not a replacement to the simulation but there are places where it can add value beyond what we can do with simulation.

Thank You

