SVAs in IP

The Holy Grail or the Holy Snail?

Markus Pugi, Lawrence Said

Cisco Systems Inc.

Oct 1, 2015

SNUG Canada

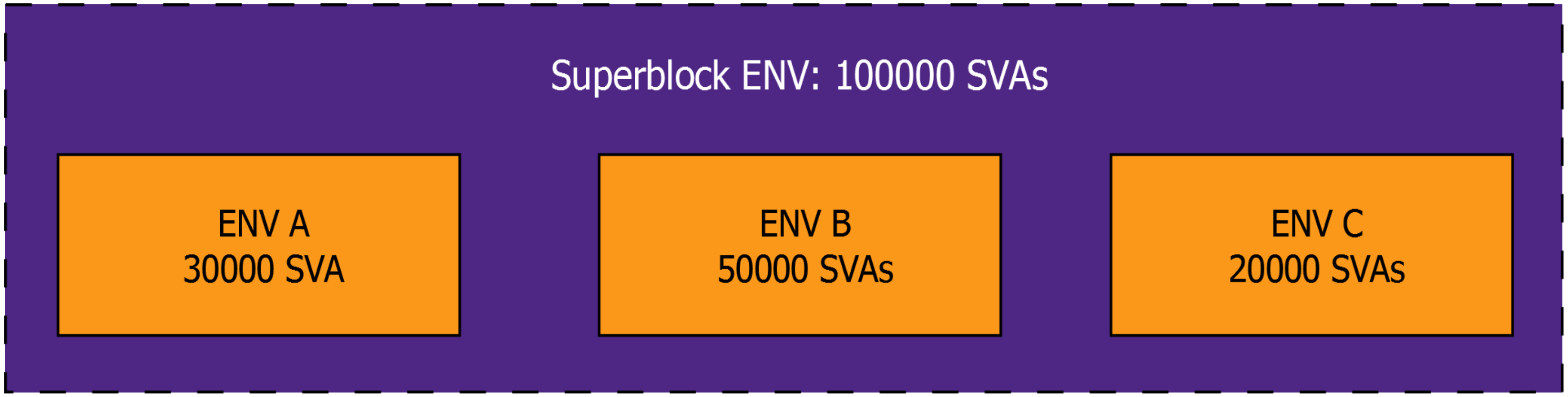
Agenda

SVAs: The Problem

- Assertion density definition: assertion lines per lines of RTL
- As assertion density increases, simulation performance becomes increasingly relevant
- Growing significantly from chip to chip
  - 2nd order effect with IP RTL reuse
  - Vertical Reuse suffers the most

Vertical Reuse Impact

SVAs in IP



- SVA performance IS relevant:
  - OLD mindset: % time in RTL vs. DV
  - NEW mindset: % time in RTL vs. DV vs. SVAs

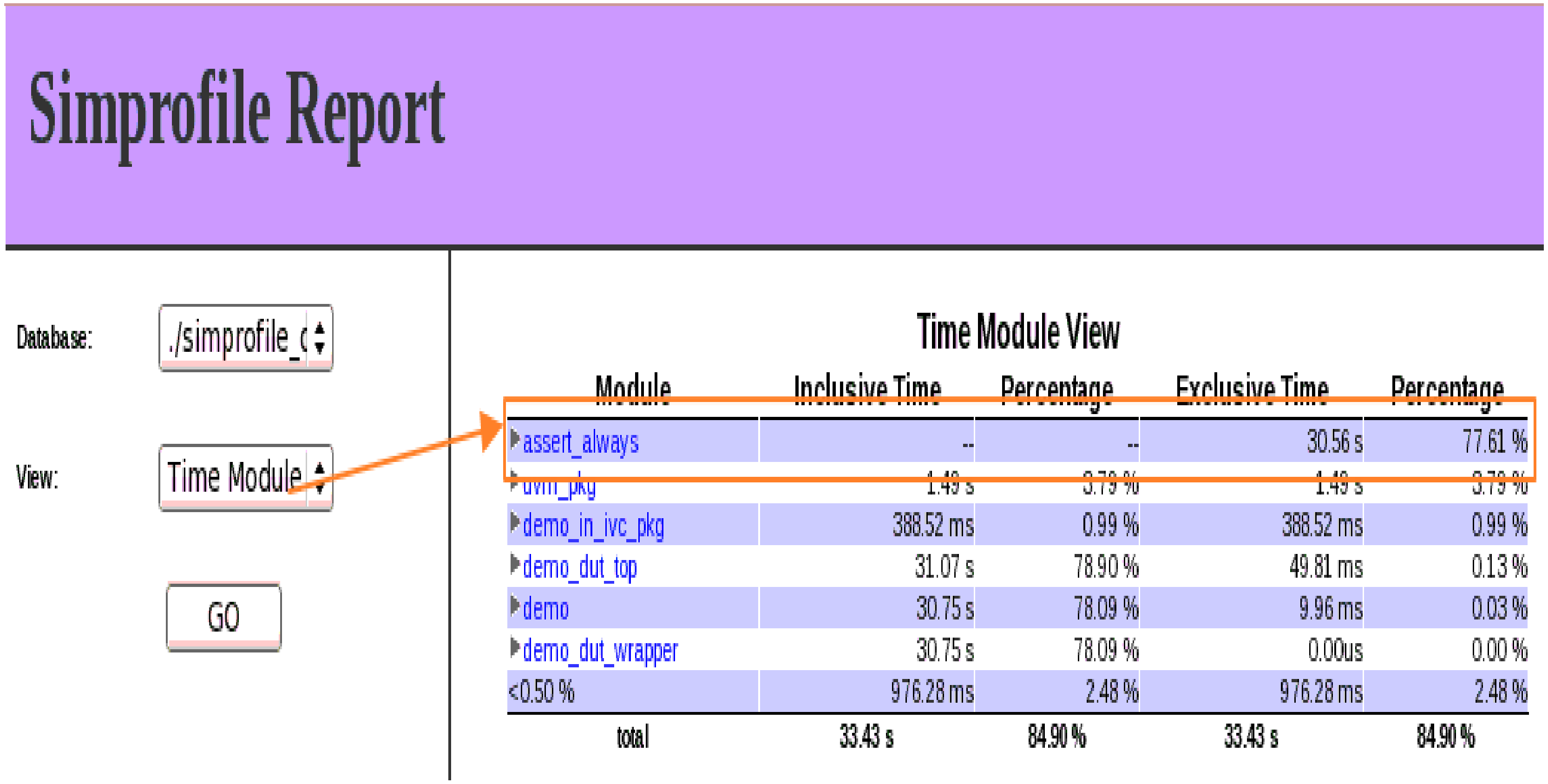
Identifying SVA Performance Impact

First steps

- Run with/without SVAs compiled in
  - obtain SVA impact
- What if SVA contribution was more than DV+RTL combined?
  - Profiler works on modules
  - Groups RTL & SVA together
    - Can pinpoint “low hanging fruit”
    - Recent profiler improvements now show SVA Library

Identifying SVA Performance Impact

Using profiler



Simprofile Report

Database:	./simprofile_d
View:	Time Module

GO

Time Module View

Module	Inclusive Time	Percentage	Exclusive Time	Percentage
assert_always	--	--	30.56 s	77.61 %
uvm_pkg	1.49 s	3.79 %	1.49 s	3.79 %
demo_in_ivc_pkg	388.52 ms	0.99 %	388.52 ms	0.99 %
demo_dut_top	31.07 s	78.90 %	49.81 ms	0.13 %
demo	30.75 s	78.09 %	9.96 ms	0.03 %
demo_dut_wrapper	30.75 s	78.09 %	0.00us	0.00 %
<0.50 %	976.28 ms	2.48 %	976.28 ms	2.48 %
total	33.43 s	84.90 %	33.43 s	84.90 %

Identifying SVA Performance Impact
Detective work required

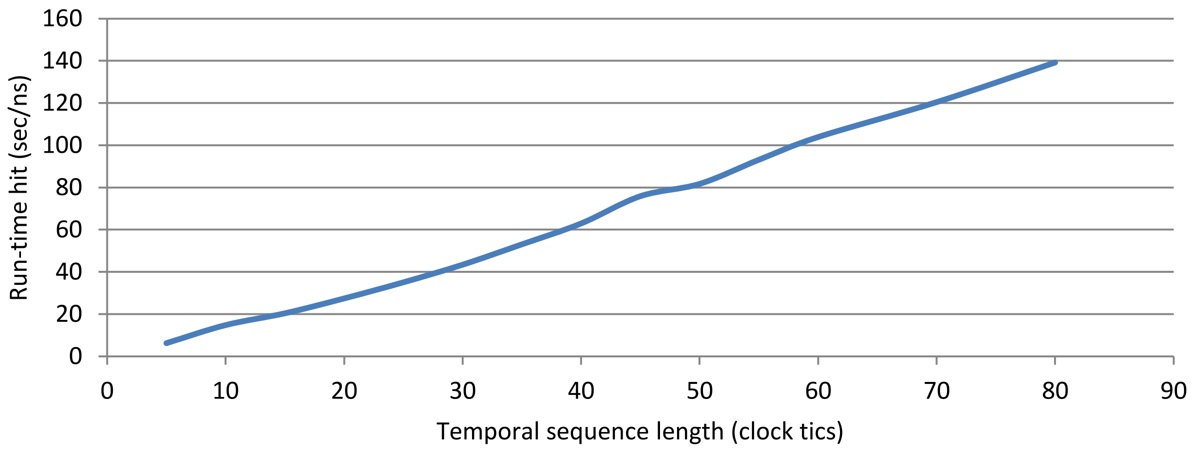
- Investigate VCS switches
  - -assert enable_diag is performance impacting (2x hit!)
    - enables certain run-time support of assertions
- Investigate IP blocks:
  - possibility of runtime disable
- Investigate Non-IP blocks:
  - Collapse assertions
  - Adhere to assertion coding guidelines
  - Don’t compile unnecessary assertions
    - Use categories!
- Recommendation: focus on blocks with highest assertion density!

SVA Performance Impact Guide

Custom “User” Assertions

- With great power comes great responsibility!
  - Ample online resources to create coding guidelines
- Temporal assertions create parallel threads
  - Possibility of large performance impact:

```
asrt_safe: assert property
            (@(posedge clk) a |-> 1’b1 ##`N b);
```



Vendor Assertion Analysis

Built in cover properties

- Not all assertions are created equal!
  - VCS SVA library contain additional coverage properties
- Controlled via coverage_level_* parameters:

```
<assertion module> #(...(),
                      .coverage_level_1(0),
                      .coverage_level_2(0),
                      .coverage_level_3(0)
                      ) asrt_inst_name(...);
```

Vendor Assertion Analysis

Built in cover property example

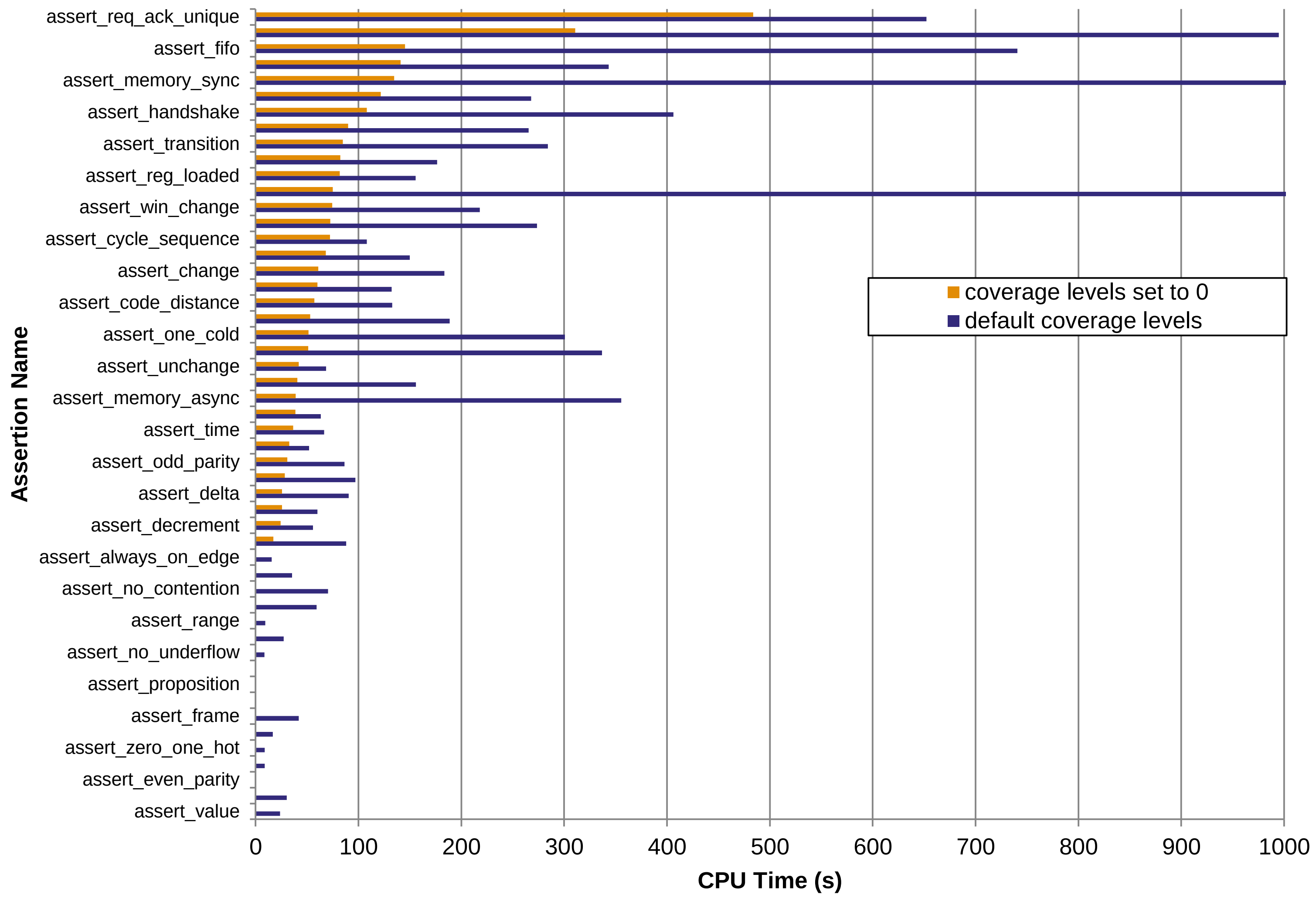
- assert_always:
  - single cover property:

```
Coverage modes :
cov_level_1_0 (bit 0 set in coverage_level_1) :
 Cover property, cover_always, indicates that the
 test_expr was asserted when enabled by reset_n
```

- Recommendation: always set coverage_level_* to 0!

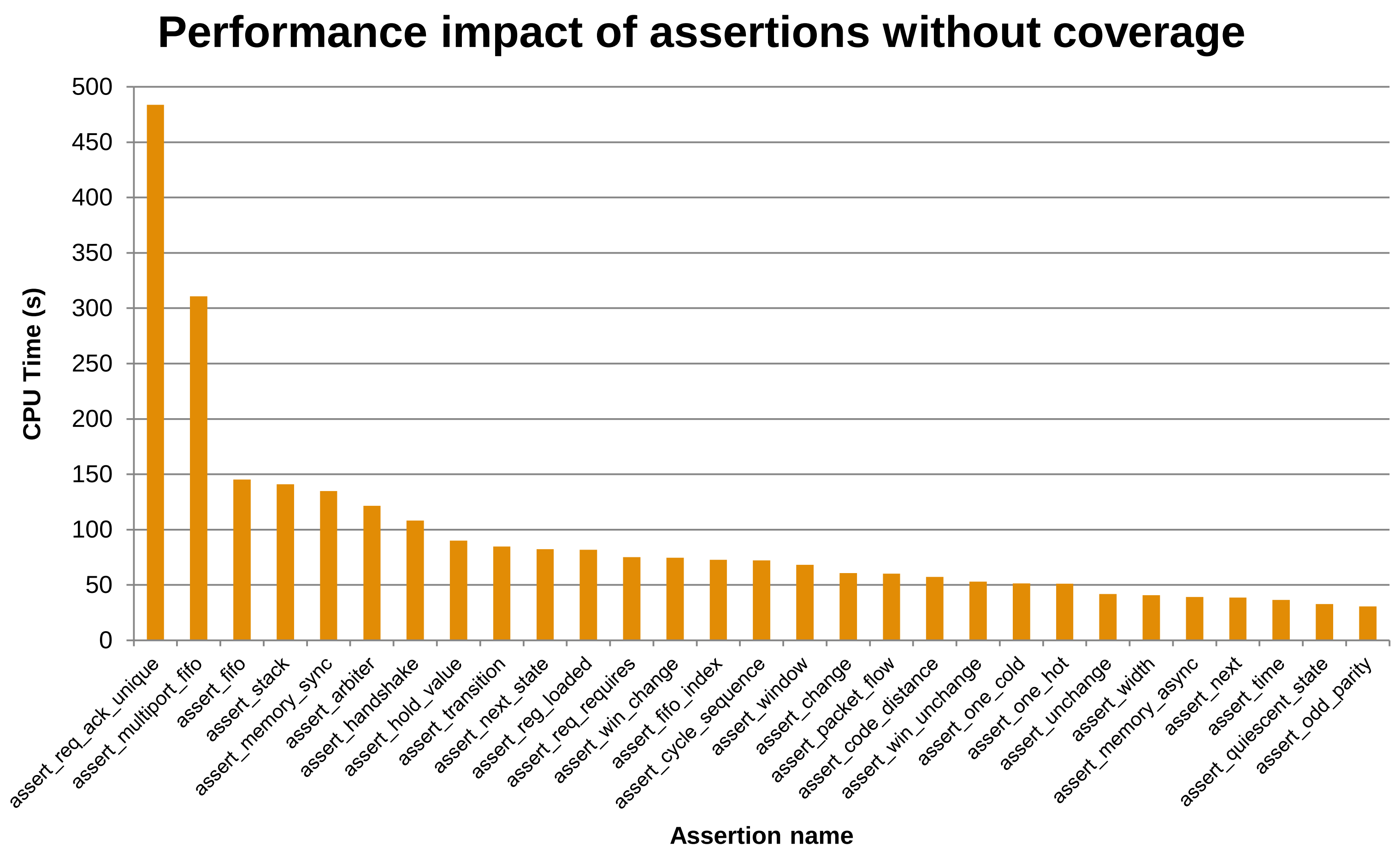
Vendor Assertion Analysis

Performance difference with coverage level 0



Vendor Assertion Analysis

Inefficient assertions with coverage level 0



SVA Performance Impact Guide

Summary

- Custom “user” assertions
    - Avoid long temporal sequences
- VCS SVA Library assertions
    - Turn off coverage parameter
    - Carefully select which assertion
    - Large variation in performance
- Reduce assertion density
    - Utilize SVA categories of “bring-up/Debug” vs “tapeout” SVAs
        - Compile out
    - Disable unnecessary assertions at run-time

Assertion manipulation

Controlling Assertions
Path based

- Without regex mechanism:

```
function void sva_stop_module(string a_module_name);
   bit [4095:0] l_path_array;
   $cast(l_path_array, a_module_name);
   $assertkill(0,l_path_array);
endfunction
```

- Full path required:

```
sva_stop_module("dut_top.dut_inst.block_inst.dummy_assert")
```

- Solution!? Defines:

```
`define BLK_PATH dut_top.dut_inst.block_inst
`define TO_STR(x) `"x`"
sva_stop_module(sformatf("%s.dummy_assert",`TO_STR(`BLK_PAT
H)));
```

Controlling Assertions

Cisco - Regex Mechanism

- Utilize VPI extensions to SystemVerilog
  - vpi_scan()
    - recursively traverse the entire DUT module hierarchy
  - vpi_iterate(vpiAssertion, child)
    - extract the assertion modules
  - vpi_get_str(vpiFullName,)
    - extract the assertion name as a string
- Create string database
  - custom functions can then manipulate SVAs using regex

Vertical Reuse Benefits

Utilizing Cisco - Regex Mechanism

- Simple regex call:

```
sva_stop_module_regex("*block_inst.dummy_assert");
```

- Performs:
  - a regex pattern match on the string database
  - calls sva_stop_module using the match from the database

- Benefits of regex:
  - Vertical reusable
  - Assertion database useful in determining assertion density

Summary

- Assertion overhead relevant with IP/reuse model
- Identifying assertion performance issues is difficult
  - Limited tools, requiring manual work
- Take care creating custom user assertions
- Carefully select checkers from vendor SVA library
- Avoid -assert enable_diag
- Regex mechanism excellent for SVA maintenance/manipulation
  - Benefits vertical reuse

Future Directions

- Test new VCS version with additional optimizations
- Profiler improvements?
  - Per property, module SVA contribution
- Regex mechanism built into VCS?
  - VPI calls into VCS
  - Perhaps instead build DB at compile time
- VCS SVA library efficiency improvements?

Thank You