DVCon 2013
Design & Verification Conference & Exhibition

February 25-28, 2013
DoubleTree, San Jose

accellera
SYSTEMS INITIATIVE

Using Formal Verification to Exhaustively Verify SoC Assemblies

by

Mark Handover
Applications Engineer
Mentor Graphics Corp.

Kenny Ranerup
ASIC Consultant
ST-Ericsson

Sponsored By:

SYSTEMS INITIATIVE

Agenda

- Introduction
- SoC Assembly Verification
- ST-Ericsson’s SoC Assembly Verification Approach
- Types of Checking
- Connectivity Specifications
- Keys to success
- Results & Conclusions

Sponsored By:

Introduction

- We will discuss a project at ST-Ericsson for which Formal verification was successfully deployed in the task of SoC Assembly verification
- We will describe the methods used and the types of checking employed
- We will present results and conclusions including the elements that led to successful deployment

Sponsored By:

SoC Assembly Verification

- SoC assembly verification is the process of checking that blocks of logic are correctly connected

- Traditionally verified using Dynamic Approach
  - Constructing a set of simulation stimuli to toggle source nodes and observe the behavior at the destination
  - Challenges
    - Likely requires directed approach not Constrained Random
    - Requires huge number of tests and potentially many test environments
    - Creating and managing test suites
    - Observability & debug

Sponsored By:

SoC Assembly Verification

- Static Approach
  - Using Formal tools to exhaustively check all connectivity in the SoC
  - Challenges
    - Requires the creation of the property set
    - Running Formal on SoC
  - Benefits
    - Simplified debug
    - No requirement to create tests
    - Time to Results

Sponsored By:

ST-Ericsson Project

- ST-Ericsson modem subsystem
  - ~ 50 IP blocks, VHDL and Verilog implementation
  - Several processors, bus interconnect, 400 memory instances
  - Multi-Power domains, clock gating
- General Verification Approach
  - Constrained Random and software driven verification
- Connectivity checking Verification
  - Traditionally has used directed tests
    - Bugs seen here have a large impact on other verification tasks
    - Time consuming and considerable effort involved

Sponsored By:

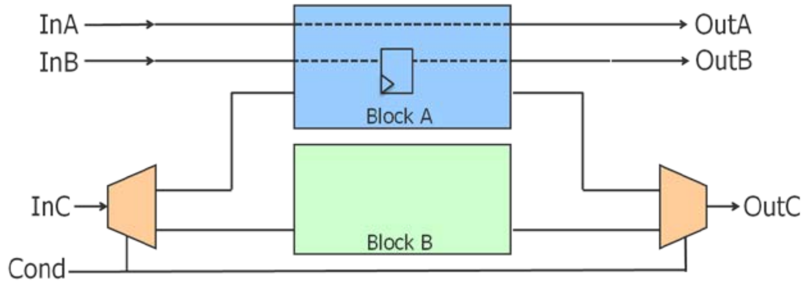
ST-Ericsson Project

- Looked to deploy formal for connectivity checking to improve throughput and results
  - IP connectivity, Reset, Clocking and DFT checks
- Connectivity Specifications
  - Many types
    - Integration spreadsheet
    - Port lists
    - Memory lists
    - Architecture specification
  - Created and used for both SoC Construction/generating RTL and for verification
  - Scripted for auto-generation of properties

Sponsored By:

Types of Checks

- Many types of connectivity checks can be performed
  - IP Connectivity
    - Unconditional Point to Point
    - Point to Point With Delay
    - Point to Point With Condition
  - Resets
    - Reset source correctly controls the correct block reset
    - Correct reset values appear on outputs of blocks/subsystems
  - DFT
    - Memory BIST control & BIST status checking, Clocking, Resets
  - Clocks
    - Clock is successfully applied to destination when enabled



Sponsored By:

Specifications and property generation

- Automation of the property set is a key requirement
  - Specification has to facilitate automation
  - Scripts written to create properties from specification
- Specification Example 1

```
{check}        CDR connection N0-N4,    Main PLL
{src}          $syscon_main_pll_cdr_entity
{srcports}     tst_pll_pf_n0,
               tst_pll_pf_n1,
               tst_pll_pf_n2,
               tst_pll_pf_n3,
               tst_pll_pf_n4,
               tst_pll_pf_enable
{dst}          $syscon_main_pll_entity
{dstports}     N0,N1,N2,N3,N4,ENABLE
{tag}          main_pll_n_ctrl
```

Sponsored By:

Specifications and property generation

- Specification Example 2
  - Allows for generating RTL and Verification of the connectivity
  - Compact formal allows for greater readability

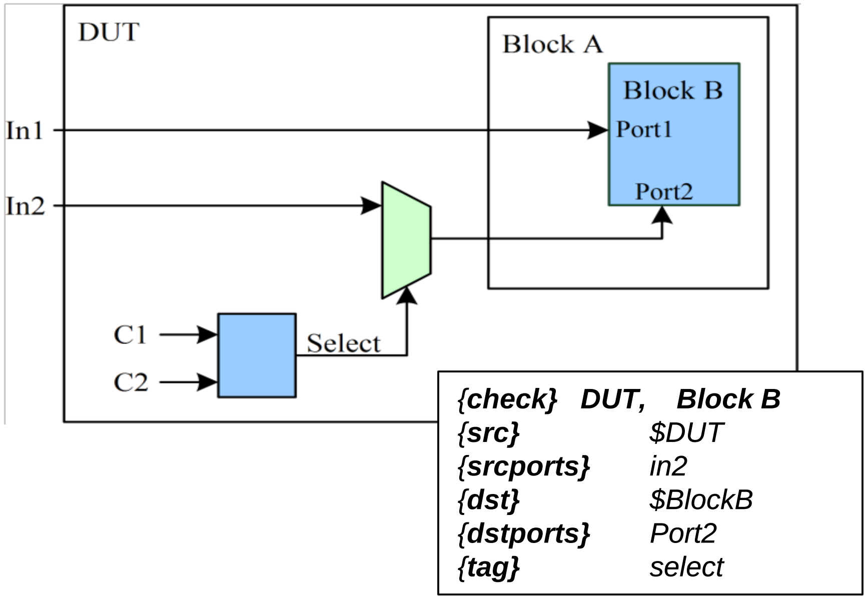
```
cpu.paddrdbg[11..2]     ;db.paddr[11..2]
cpu.paddrdbg31          ;db.paddr[31]
ac.prdata[31..0]        ;pb0.ac_prd[31..0]
ac.psel_vec_a           ;pb0.ac_evec_psel
cpu.rstreq              ;_to_open
cpu.nopwrdwn            ;_to_open
cpu.addr[31..12]        ;_to_constant ;(OTHERS => '0')
cpu.addrv               ;_to_constant ;'0'
```

Connectivity Property Examples

Sponsored By:

- Property examples
  - Port connectivity

```
assert_in1__blockb_port1:
    assert property (
     dut.in1 == dut.a.b.port1
    );
```



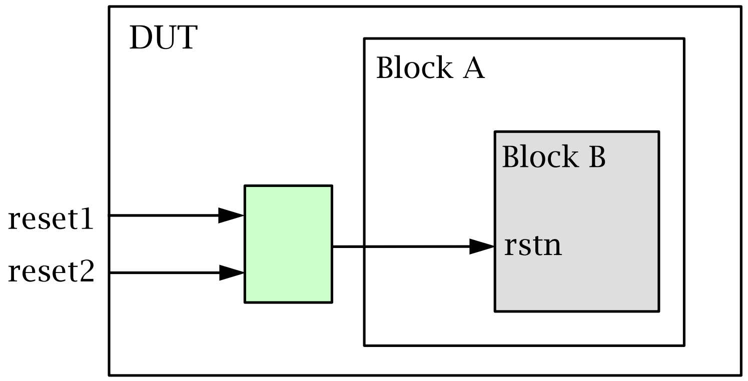
{check}	***DUT, Block B***
{src}	*$DUT*
{srcports}	*in2*
{dst}	*$BlockB*
{dstports}	*Port2*
{tag}	*select*

```
assign select = ((dut.c1 == 1) && (dut.c2 == 1));
assert_select_in2__b_port2:
    assert property
        select |-> dut.in2 == dut.a.b.port2
    );
```

Connectivity Property Examples

- Reset Connectivity
  - Multiple reset sources to a block

```
assert_reset1_blocka_rst:
  assert property (
    dut.reset1 == 0 |->
      !dut.a.b.c.blocka.rstn
  );
assert_reset2_blocka_rst:
  assert property (
    dut.reset2 == 0 |->
      !dut.a.b.c.blocka.rstn
  );
```



Sponsored By:

Connectivity Property Examples

- Reset Value check

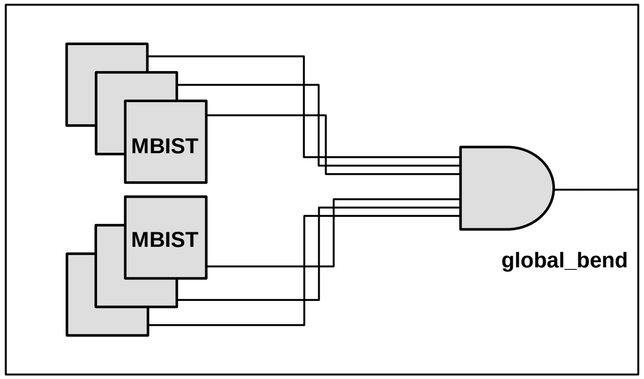
```
module reset_checker ( input ref_clk );
   default clocking ref_clock @(posedge ref_clk);
   endclocking

   assume property (dut.core_rst_n == 0);
   assume property (dut.dbg_rst_n == 0);

   assert_rst_mem_m0_awsize:
      assert property ( dut.mem_m0_awsize == 0);
   assert_rst_mem_m0_awvalid:
      assert property ( dut.mem_m0_awvalid == 0);
 endmodule

bind dut reset_checker i_reset_checker (.*);
```

Sponsored By:

Connectivity Property Examples

- Verifying a global logic function
  - MBIST status signals are combined from all controllers to a global output port using AND or OR logic.
    - This is very difficult to verify in simulation.

```
assert_global_bend: assert property (
        dut.global_bend == (
            dut.a.b.c.bend  &&
            dut.a.x.i1.bend &&
            dut.b.bend
        )
 );
```

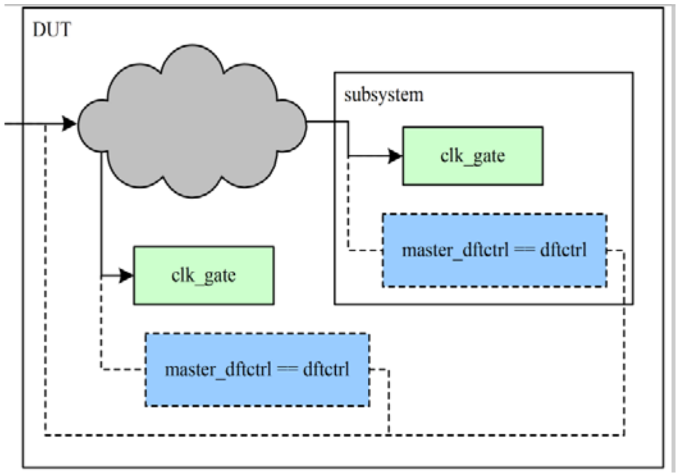


Sponsored By:

Connectivity Property Examples

- Design Regularity
  - Can take advantage of regularity to greatly simplify property generation
  - Use SystemVerilog bind statement to bind checker module to all instances thereby avoid listing all instance paths.

```
– bind clk_gate checker i_checker(.*);
```



Results and Conclusions

- ST- Ericsson have successfully developed an approach and deployed connectivity checking using formal verification
  - Exhaustive verification of SoC assemblies
- New design project
  - 100+ bugs were found
    - Simple connectivity to interface bugs
  - Property set for project included
    - 3500+ automatically generated properties
    - 300+ semi automated properties
    - 100+ manually created constraints
- Subsequent project, same structural design but with some new functionality, 40 bugs found

Sponsored By:

Results and Conclusions

- Keys to success
  - Automation
    - Its essential that the Connectivity specifications lend themselves to automation of properties
      - SoC's will generate thousands of checks to be verified
  - Design regularity and consistency
  - Considerations for Formal
    - Application doesn't require high level of Formal competence
      - Properties are typically simple
    - SoC's will contain blocks which are non-friendly to formal tools
- ST-Ericsson continue to use and develop formal based connectivity checking

Sponsored By:

THANK YOU