IP Design and Integration Verification
Utilizing Formal Technologies

TVS
Verification Futures
November 2013
OneSpin Solutions
The Pioneers of Advanced Verification Solutions

Pioneering, Leading Technology
Broad Range of Solutions
User Oriented Approach

- Award winning, comprehensive verification product line
- Validated by hundreds of design projects & tape-outs
- Well financed global engineering and support team
- Incubation at Infineon/Siemens, 300+ years development

Incubation @ Infineon
Product Commercialization
Lead Customer Adoption
Global Expansion


From Automated Solutions To Advanced Verification

Rapid, Exhaustive Coverage-Driven Property Verification

Quick & Easy, Automated, Comprehensive Design Analysis

ASIC & FPGA Tool Sign-off Accuracy Sequential Equivalency Checking

OneSpin Advanced Formal Proof Engine

High Performance, Easy to use, Accessible Technology Platform
OneSpin & Renesas
IP Integration Verification Application

• OneSpin Solutions’ technology is being utilized in many static verification scenarios by leading companies

• Today we are presenting a relevant example application where static verification drove significant improvements in an SoC integration scenario

High Coverage Block/IP Integration Verification Example with Renesas
Example: Renesas’ Micro Controller Unit Platform (MCU-PF) for the efficient development of new products within the same series.
SoC Platform Based Design
Selection of IPs, Connectivity, Parameters,…

Verification becomes a big challenge!

Selection of memory size
Change of interrupt factors
Change of bus bridge
Selection of peripheral IP
Selection of port and package
Verification Challenges in Platform Based Design

Verification of IP blocks

- Multiple configurations & high coverage
- Difficult to achieve with simulation alone

Verification of SoC Integration

- Logical Connectivity & Combinations of Communication Channels
- Each SoC needs specific testbench to achieve high coverage
  - Directed tests for all required scenarios
  - Time-consuming
  - Product derivatives require new tests
Formal Verification of Platform Components
Parameterized Assertions allow Re-Use

Components are Platform (PF) Modules, IP Modules, Connection Rules

Formal Verification of Components by Parameterized PF and IP Assertions

- PF-Assertions
- Connection Rules
- CPU
- DMAC
- Function Modules
- IP-Modules
  - IPa
  - IPb
  - IPC
  - IPd
- Specific to Product Type
- Common to a Series
- Parameterized Assertions allow Re-Use
## Transaction Level Operational Assertions

**Capture and Verify IP Behaviour at Interface**

<table>
<thead>
<tr>
<th>req_i</th>
<th>t_gnt</th>
<th>t_free</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_r</td>
<td>idle</td>
<td>idle</td>
</tr>
<tr>
<td>grant_o</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>free_i</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Sequence

```vsl
def sequence t_gnt; nxt(t,1); endsequence
def sequence t_free; nxt(t_gnt,3); endsequence
```

### Property

```vsl
def property arbiter_prop1;
  t ##0 state_r == idle and
  t ##0 req_i[0] and
  t_free ##0 free_i
  implies
  t_gnt ##0 grant_o == 3'b001 and
  t_free ##1 state_r == idle;
defendproperty
```

### Transaction Level Assertions

- Achieve an unprecedented productivity in assertion writing
- Capture complex transactions in one compact and intuitive assertion.
- Cover more functionality with less assertions
- Be productive after a single day of training
- Optimized proof engines and debugging capabilities allow high productivity
- Based on OneSpin SVA library

### Cause / Condition

### Effect / Expectation
Several formal techniques are applicable

- Formal Assertion Based Verification for each IP
- Operational assertions provide high functional coverage
- Assertions for logical connectivity and register maps can be derived automatically

What about functional connectivity?

- “Can IP-c raise an interrupt at CPU-a under each operation mode of each other IP”
Verification of operation combinations of multiple IP-Modules is a significant task.

If IPc is the verification target, its operation should be verified on EVERY combination of parallel operations of IPa, IPb and IPd.

*The number of operation combinations becomes huge!*
Traditional: Simulation Verification
Time Consuming & No Verification Re-Use

Verification is done by simulating test-cases on a testbench.
Each test-case controls all IPs, including the target and related IP.
All test-cases and the testbench must be revised for a different MCU configuration, even if most of the IP is reused.
The verification coverage depends on the test-cases.

Test-Cases:
- Target IP Control (IPc)
- Related IP Control (IPA & IPb)
- PF-Module Control
- Other Non-related IP Control
- Expected Results

"Can IP-c raise an interrupt at CPU-a under each operation mode of each other IP?"
Alternative: Formal Verification of MCUs
High Coverage and Re-Use of Existing Assertions

- Each set of formal assertions includes only descriptions of the specified IP.
- It is reusable, in the same way the IP is.
Simple Example of Formal Verification
Interrupt Function

- Interrupt function by MCU
  - Checking that an interrupt from “Function IP” occurs in “PF-module” via “Connection IP”.
- Assertions covering interrupt function are *created from existing assertions* in IP-Library.
Working Principle of Assertion Creation
Example for Interruption Function

Steps:
① Extraction of proven IP-Assertions from IP-Library
② From each assertion extract required condition part and expectation parts
③ Automatically create new assertion from conditions and expectation to check connection

IP-Library
- INT request property (R1)
  - Setting of request (R1)
  - implies
  - Checking of request (I/F signals)
- INT controller property (I1)
  - Setting T1 from R1
  - Setting of request from R1 (I/F signals)
  - implies
  - Checking of request to T1 (I/F signals)
- INT accept property (T1)
  - Setting of request (I/F signals)
  - implies
  - Checking of request (T1)

PF-Module

Definition of assertion
- property
- condition
- implies
- expectation

Definition (Operational ABV)
- property
- {condition} implies
- {expectation}
- endproperty
Example

Formal Assertion for Interrupt Function

```verilog
`define Bv_level 5
`define T_INTNUM top.pf_module.cpu.intnum
:

////////// property /////////////

property int_cpu_peri;
    disable iff(`intc.RESET_BUS == 1'b0)

    // Setting of Request R1
    tr_b ##0 set_freeze(ir_num,`R_ir_num) and
    tr_b ##0 `R_IREQ == 0 and
    tr_r ##0 `R_IREQ == 1 and
    tr_n ##0 `R_IREQ == 0 and

    // Setting T1 from R1
    tb_b ##0 set_freeze(level,`Bv_level) and
    during(tb_b,tb_e,`B_iер == `Bv_iер) and
    during(tb_b,tb_e,`B_ipr == level) and
    during(tb_b,tb_e,`B_irqmd == 2'b01) and
    during(tb_b,tb_b,`B_ir == 1'b0) and

    implies

    // Arrival of request at T1
    tt_req ##0 `T_INTNUM[7:0]==ir_num and
    tt_req ##0 `T_INTRQLV[3:0]==level;

endproperty
```
Evaluation of Formal Approach
Full Coverage, Lower Effort, Less Run Time

- Test development and execution data for “INTC”
- The formal method achieves full verification coverage for any combination of interrupt sources within a practical time.

<table>
<thead>
<tr>
<th>Test Spec. of Each Test-case or Assertion</th>
<th>1. Simulation</th>
<th>2. Formal Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Timing x Interrupt Req. Src. x Interrupt Ack. Dest.</td>
<td>1 timing x 1 source x 5 destination x 547 test-cases</td>
<td>1 timing x 1 source x 5 destinations, on PF-assertion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development Test-case or Assertion</th>
<th>Language</th>
<th># Tests</th>
<th># Lines</th>
<th>Development</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assembly Lang.</td>
<td>547</td>
<td>131,280</td>
<td>40.8 days</td>
<td>Total Run Time</td>
</tr>
<tr>
<td></td>
<td>Operational ABV</td>
<td></td>
<td></td>
<td></td>
<td>547.0 Hour</td>
</tr>
</tbody>
</table>

- An interrupt source is selected from 547 request events in the INTC.
- An interrupt destination is selected from CPU and 4 channels of DMAC in the PF-Module.

Order Magnitude Improvement!
Summary

Verification of IP blocks

- High coverage formal verification of each IP block
- Re-Use of assertions for different parameter settings

Verification of SoC Integration

- Exhaustive formal check of all combinations of communication channels
- Re-Use of block level assertions on SoC level

Tremendous saving on time and budget

- Order of magnitude saved on creating tests, as well as run-time
- Higher verification quality