A Formal Approach to Low Power Verification

Frédéric Rocheteau
STMicroelectronics

Barbara Jobstmann
Jasper Design Automation
Outline

• Low Power Design and Verification Challenges

• Overview Formal Apps

• Usage of Formal Apps within ST

• Low Power Verification (LPV) App
  • Power Formats in a Nutshell and the Missing Piece
  • Overview of LPV App
  • Generation of Auto-Checks
  • Integration with other Apps
  • Status and Next Steps

• Conclusion
Low Power Design and Verification Challenges
Context

• Design complex systems-on-chip for multiple market segments
  • Focus on multimedia convergence (smart devices, connectivity)
  • Sensing technologies

• General demand for devices with reduced power consumption
  • Mobile products on the edge of this trend (battery life …)
  • Technology urgent requirement as well (heat dissipation …)

• Develop flexible and re-usable platforms, subsystems and blocks
  • Assembly of numerous IP blocks from multiple providers
Power Consumption Reduction

- Technology process
  - FDSOI: very fine grain power management

- Generalized usage of low power design techniques
  - Device sizing, clock gating, power gating …
  - Retention
  - Cross power domains (feed-through) wires

- Additional verification challenges
  - Verification space explosion
  - New types of bugs (isolation bugs, voltage bugs…)

- Requirement for new verification technologies
  - Urgent for mobiles phones
  - Rapidly increasing for other types of systems (set top boxes …)
Formal Methods Status

• Formal Methods used within ST for 20 years
  • Theorem Proving (e.g. instruction set exploration / validation)
  • Combinational Equivalence Checking
  • Property Checking
  • Transistor Abstraction (e.g. transistor netlist vs RTL)
  • Test generation

• Combinational Equivalence Checking mainstream
  • Applied directly by designers

• Formal Techniques largely underexploited within ST
  • Unable to capitalize on experience
  • Property checking use remains very limited
  • No will to expand formal techniques scope (e.g. arch exploration)
Need for Formal Methods

• Breakthroughs obviously required to address design / verification challenges

• Successful large scale uses of formal methods in several major semiconductors companies

• Negative global perception of property checking
  • Difficult to use
  • Capacity limitations

• Lack of knowledge of what is possible
Formal Methods Resumption

- New dedicated formal design / verification tools
  - Simple to implement, due to latest process in formal techniques
  - Applicable to complete systems within specific contexts
  - Directly usable by designers

- Use of JasperGold Apps to provide tool for generic tasks
  - FPV
  - Coverage
  - Point to point connectivity checking
  - X propagation

- Collaboration with Jasper to implement more specific tools
  - On top of property checking and sequential equivalence checking solutions
  - Memory integration
Sample Application: GPU Subsystem

Main features:
- 3D graphics IP
- Dedicated HW power management
- Switchable 3D power
- DVFS
- Specific sensors

Size:
- >50M gates
Specific Design & Verification Tasks

• After graphics IP is received from external provider:
  • Replace original memories by ST Memories
  • Add BIST
  • Add DFT
  • Add ST specific low-power features (power switches and control IPs…)
  • Add ST specific IPs (sensors …)

• All these additional design steps have to be verified:
  • Does not break original IP functionality
  • New features are fully functional
Low Power Verification

• Check that behavior of a system remains correct after modifying it to reduce its power consumption

• Same behavior when operating in a stable power mode
  • Constrained sequential equivalence checking between original and power aware models

• Transition from one power mode to another is safe
  • No X propagation
  • No data corruption (e.g. FIFOs, Feed-through wires)

• Low power controls are correct
  • Right order of events during power-up / power down sequences

• Significantly reduced verification time with better coverage
  • And better vision of this coverage
Overview Formal Apps
What Is Possible with Formal Technology

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

X-Propagation Verification
- Unexpected X Detection and debugging

Intelligent Proof Kits and Verification IPs
- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

CSR Verification
- Automated register verification

Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

... and many more
What Is Possible with Formal Technology

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

CSR Verification
- Automated register verification

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Coverage Metrics
- Coverage Metrics Generation from Formal Verification
- Coverage metrics to establish sanity of formal testbench
- Coverage metrics for bounded/full proof result
- Interacting with coverage metrics from simulation via an external DB (e.g. UCDB)
What Is Possible with Formal Technology

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

CSR Verification
- Automated register verification

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

X-Propagation Verification
- Unexpected X Detection and debugging

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

... and many more
What Is Possible with Formal Technology

**Property Synthesis (Structural / Behavioral)**
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

**Formal Property Verification**
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

**Executable Spec**
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

**X-Propagation Verification**
- Unexpected X Detection and debugging

**Connectivity Verification**
- Chip-level connectivity
- Conditional connection with latency

**Architectural Modeling**
- Executable spec
- Absence of deadlock
- Cache coherency

**Post-Silicon Debugging**
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

**RTL Development**
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

**CSR Verification**
- Automated register verification

**Other SoC-Related Applications**
- Glitch detection
- Multi-cycle path verification
- Low power verification

Formal increases SoC integration productivity

… and many more
What Is Possible with Formal Technology

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

CSR Verification
- Automated register verification

X-Propagation Verification
- Unexpected X Detection and debugging

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

Formal provides visibility into a design, isolating relevant areas effectively

Popular standard protocols
- Configurable, illustrative, optimized for formal

… and many more
What Is Possible with Formal Technology

**Property Synthesis (Structural / Behavioral)**
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

**Formal Property Verification**
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

**Executable Spec**
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

**X-Propagation Verification**
- Unexpected X Detection and debugging

**Connectivity Verification**
- Chip-level connectivity
- Conditional connection with latency

**RTL Development**
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

**Architectural Modeling**
- Executable spec
- Absence of deadlock
- Cache coherency

**CSR Verification**
- Automated register verification

**Post-Silicon Debugging**
- Failure signature matching
- Root cause isolation
- Candidate cause isolation
- Validation of fixes before re-spin

**Intelligent Proof Kits and Verification IPs**
- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

**Synergy from various sources of properties at various abstraction levels**
- ... and many more
What Is Possible with Formal Technology

- **Architecture Modeling**
- **RTL Development**
- **Functional Verification**
- **SoC Integration**
- **Post-Silicon Debugging**

**Formal Verification throughout the Entire Design Cycle**

- **Property Synthesis (Structural/Behavioral)**
  - Automated assertion generation
  - Functional pre-defined property generation
  - Inference & synthesis of properties from RTL & simulation
  - Identification of coverage holes

- **Inference & synthesis of properties from RTL & simulation**

- **Automated register verification**
  - Automated expected X detection and debugging
  - Automated register verification

- **Intelligent Proof Kits and Verification IPs**
  - Certification of AMBA 4/ACE checkers
  - Popular standard protocols
  - Configurable, illustrative, optimized for formal

- **SoC-Related Applications**
  - Glitch detection
  - Multi-cycle path verification
  - Low power verification

- **Other SoC-Related Applications**

- **Post-Silicon Debugging**
  - Failure signature matching
  - Root cause isolation
  - Candidate cause elimination
  - Validation of fixes before re-spin

- **Architecture Modeling**
  - Automatic synthesis
  - Functional verification
  - Inference & synthesis from intent
  - Designer-based verification w/o testbench
  - Design trade-off analysis

- **Functional Verification**
  - Protocol certification
  - End-to-end packet integrity
  - Asynchronous clocking effects
  - Assertion-based verification
  - Proofs for critical functionality
  - Debug isolation & fixes validation

- **SoC Integration**
  - Executable spec
  - Absence of deadlock
  - Cache coherency

- **Post-Silicon Debugging**
  - Failure signature matching
  - Root cause isolation
  - Candidate cause elimination
  - Validation of fixes before re-spin

- **Other SoC-Related Applications**

... and many more
Look & Feel of Jasper’s Apps

Example: RTL DEVELOPMENT
Typical Designer-Based Verification

- Testbench and input stimulus are required to explore and verify design behavior
  - Usually unavailable at early design stage or smaller block levels
  - Designers usually do not have time to create extensive tests (non-trivial effort)
  - No persistent ROI (throwaway effort)

- No systematic method for confirming RTL functional scenarios as each feature is added to the RTL code
  - Usually done by eye-balling the RTL

This usually means designer-based verification is not done.
Rethinking Designer Verification

- **Simulation**
  - More of an “input driven” method, may not exercise desired behavior
  - Wiggle the inputs to produce a desired behavior (trial and error)

- **Visualize**
  - Specify the target and let the formal engines generate the stimulus (“output driven” method)
  - Interactively add constraints to construct desired waveform
The ‘target’ is satisfied without the need of a testbench
Analyze Waveforms with ‘Why’-Feature

always @(posedge clk or negedge rstN)
    if (rstN == 1'b1) begin
        int_read_write <= 1'b0;
    end
else if (new_tran && trans_started && ~wr_rd_req && rd_ready)
    int_read_write <= 1'b1;
else if (ready && trans_started) int_read_write <= 1'b0;
always @(posedge clk or negedge rstN)
    if (rstN == 1'b1) begin
        int_addr <= 7'd0;
        int_valid <= 1'b0;
    end

A Formal Approach to Low Power Verification 12/10/12
Capture Properties from the Waveform

‘export -to_sva/psl’ to include captured properties in other verification flows
Complete Flow for RTL Designers

RTL

Visualize design behavior w/o testbench

Debug failing scenarios

Functional scenario A: assertion 5 violation
Functional scenario B: assertion 7 violation
Functional scenario C……
Functional scenario D…..

Compare saved scenarios against modified RTL

Modified RTL

What-if analysis

Combine and save multiple functional scenarios

Database

Scenario A
Scenario B
Scenario C
Scenario D
RTL Development Summary

• Conduct early RTL exploration w/o a testbench

• Store expected functional scenarios and validate against modified RTL

• Perform design trade-off analysis while RTL is being developed

• Properties developed at this stage live with the RTL and are leveraged throughout the verification flow
Usage of Formal Apps within ST
Formal Property Verification

• Verification of control dominated blocks
  - Clock and reset generators
  - Power controllers
  - Sensor controllers

• Complete specification required

• Reliability improved with bounded proofs

• Deadlock detection highly desirable but difficult
  - System level verification
Coverage

- Required complement to FPV
  - Must be sure that verification environment and properties are themselves correct

- How complete is the verification?
  - Measure out of Cone of Influence to find holes

- Is the environment over constrained?
  - All properties to be checked must be covered

- How good are the bounded proofs?
  - Dedicated bounded coverage to answer to that
Coverage

• Dead code analysis
  • Are assumptions disabling behaviors to be checked?

• Combination of dynamic and formal verification
  • Merge coverage results

• Complement dynamic verification
  • Are some tests, covered by properties, missing?

• Replace dynamic verification
  • Is formal verification strictly checks more behaviors than dynamic verification?
Design Exploration

- Bug analysis
  - Formal query engine
  - Quiet traces

- Features understanding
  - Discover certain aspects of the third-party IPs
  - Rebuild specification for legacy blocks

- Design assistant
  - Help to define / optimize reset / power down / power up sequences
Connectivity

- Point-to-point connectivity checking
  - 2500 reference connections generated from key project document

- Once everything is verified at the unit / block-level …

- Provides a first check that blocks have been assembled into the subsystem correctly

- Eliminates wiring errors; useful before functional system testing

- Useful for low-power:
  - Can check connectivity rules for changing power states
  - Rule validity can be tied to power states
Sequential Equivalence Checking

• Simplest to use
  • No state correspondence required

• Capacity limitations
  • Property checking + dedicated solvers

• Huge number of applications
  • Memories integration verification
  • Original vs power aware model comparison
  • Incremental verification
  • …
Memories Integration

• Check that system behavior is not altered by replacing instances of a generic memory model by instances of real memory models

• Real memory models are wrapped to adapt interfaces and provide assemblies for non native sizes

• Several hundreds memory instances with numerous different configurations
  
  e.g. 92 single port and 50 double port memory configurations in a GPU design

Memory functional model

Memory Wrapper

ST memory

ST memory
Memories Integration Result

- Single port memory with 128 bits data and 12 bits address
  - 5.7M gates and 1M flip-flops
  - 2h time out

<table>
<thead>
<tr>
<th>Output bit</th>
<th>FPV</th>
<th>SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q0</td>
<td>proven</td>
<td>1158 s</td>
</tr>
<tr>
<td>Q1</td>
<td>proven</td>
<td>1083 s</td>
</tr>
<tr>
<td>Q2</td>
<td>time out</td>
<td>N/A</td>
</tr>
<tr>
<td>Q3</td>
<td>time out</td>
<td>N/A</td>
</tr>
<tr>
<td>Q4</td>
<td>time out</td>
<td>N/A</td>
</tr>
<tr>
<td>Qx (x &gt; 4)</td>
<td>time out</td>
<td>N/A</td>
</tr>
</tbody>
</table>
CSR / Security

• Verify that a design matches the CSR spec
  • A register can’t be modified after end of reset sequence

• Verify data protection schemes
  • A protected data (memory or register) always traverse an encryption block before reaching an output interface
Low Power Verification App
Outline

• Low Power Design and Verification Challenges

• Overview Formal Apps

• Usage of Formal Apps within ST

• Low Power Verification (LPV) App
  • Power Formats in a Nutshell and the Missing Piece
  • Overview of LPV App
  • Generation of Auto-Checks
  • Integration with other Apps
  • Status and Next Steps

• Conclusion
Power Formats in a Nutshell
Power Formats in a Nutshell

• Two Standards: UPF/CPF

• Define additional functionality

• They allow user to define:
  • Power and Voltage domain
  • Supply Network
  • Power Switches
  • Isolation Cells and Level Shifters
  • Retention Cells
  • Power States/Modes (= configuration of switches)
  • Power State Table (defines legal configurations)
Example Functionality not in RTL

- **Power Domains**: group of blocks that can be turned off together. “Turn-off” means all elements in these blocks lose their values and drive X.

- **Isolation Cells**: isolated signal has a different driver (than stated in the RTL) if isolation condition is true.

In functional verification it is necessary to take functionality described in power format file into account!
Overview LPV App
Power-Aware Formal Analysis with JasperGold Lower Power Verification App

1. Bugs
2. Assertions
3. Status
4. Optimization

- Power-aware internal model generation
- Assertion inference

JG-LPV

User-Defined Assertions

JG-FPV
JG-XPROP
JG-CONN
JG-RTLD
...

RTL
CPF
UPF
Power-Aware Model

• Read RTL and power intents from UPF / CPF file
  • Power domains
  • Supply networks and power switches
  • Isolation rules
  • Retention rules

• Power aware model generation
Auto-Assertions
Auto-Generated Assertions

• Property inference from power intents and low power design guidelines

• Property inference from ST specific needs
Auto-Assertions: Examples (1)

• If the power supply of a power domain is changing (i.e., a power switch changes), then clock of this block is disabled.

• If a power supply net has resolution semantics onehot, then there is never more than one active driver.

• Whenever the value of an element is restored from the retention logic, the power supply of the corresponding retention logic is on.
Auto-Assertions: Examples (2)

- Whenever a power domain is powered down, all the isolation conditions related to this power domain will be true strictly before, during, and strictly after the power is turned off.

- No signal is isolated twice with contradictory clamp values.

- Clamp values coincide with reset values.
Integration with Other Apps
Motivation

• In Low-Power Design, many verification tasks that are executed on the non-power aware model verification need to be repeated with the power aware model using UPF/CPF

• This makes combining the LPV App with other Apps and Jasper technologies very compelling

• We will describes 4 such combinations:
  • LPV-CONN
  • LPV-CSR
  • LPV-Scoreboard
  • LPV-XProp

• Other relevant combinations that exist but not covered here are:
  • LPV-SEC
CONN-App with LPV-App
CONN-App: Review

Top-level of SoC

Connectivity proofs (assertions and covers)

Waveforms with connectivity conditions

Connectivity map

RTL

A Formal Approach to Low Power Verification 12/10/12
Non-Power-Aware Connectivity

Classical connectivity verification checks that each connection in connectivity table exist in RTL.
Power-Aware Connectivity: UPF/CPF applied

- Parts of the design belongs to different power domains
- Different domains can be in different on-off state
- PDA_A is always on
- PDA_B and PD_C can be on or off
- Not all connections are valid now, their validity depends on the Power State
Power Condition in Connectivity Table

- True when PD_B==ON
- True when PD_B==ON && PD_C==ON
- True when PD_C==ON
- Always True (PD_A is always ON)

Power conditions are simply added to the “condition” field in the table.
Connectivity Verification App

• Simply use the CONDITION field

<table>
<thead>
<tr>
<th>Compact Format:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIAS</td>
<td>top</td>
<td>a.b.c.d</td>
<td></td>
</tr>
<tr>
<td>CONNECTION</td>
<td>a3</td>
<td>out_a3</td>
<td>dum</td>
</tr>
<tr>
<td>CONNECTION</td>
<td>cntl_conn0</td>
<td>out_cntl</td>
<td>e</td>
</tr>
<tr>
<td>CONDITION(STABLE)</td>
<td>e</td>
<td>1'b1</td>
<td></td>
</tr>
<tr>
<td>CONDITION</td>
<td>PD_B</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>CONNECTION</td>
<td>cntl_conn1</td>
<td>out_cntl</td>
<td>some_reg</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>a==b</td>
<td></td>
<td>sdfj</td>
</tr>
<tr>
<td>CONNECTION</td>
<td>ab</td>
<td>top</td>
<td>top_ab</td>
</tr>
<tr>
<td>LATENCY</td>
<td>(1..3)</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Potential Counter Example

CEX: Path of this connection goes through a mux controlled from PD_B

True when PD_C==ON
CSR-App and LPV-App
Motivation

• Given a DUV with Register space accessed by:
  • Standard interface (AHB, OCP, etc.)
  • Proprietary interface (parallel, serial)

• Power Aspect
  • The Register space sits in always-on Power-Domain
  • ➞ It is suppose to have valid access from the AHB bus regardless of the Power State

• To prove without the Low-Power aspect:
  • Data integrity of register fields
    • i.e. Data read from a register equals previously written data
  • Reset values
    • Data read from a register equals reset value till it is written to

• To prove with the Low-Power aspect:
  • All the above, but while the DUT can do different Power-Sequences
CSR App Overview

- Register Definition
- DUT
- Glue Logic
- Checks

JasperGold CSR App

Jasper CSR PA
CSR App: Sample Template

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR_OFFSET</td>
<td>0x0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR_WIDTH</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_WIDTH</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>DATA WIDTH / LINDEX</td>
<td>ADDRESS / RINDEX</td>
<td>ACCESS TYPE</td>
<td>RESET VALUE</td>
<td>DESCRIPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSR</td>
<td>reg1</td>
<td>8</td>
<td>0x0010</td>
<td>RO</td>
<td></td>
<td>Config reg 1</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>lower</td>
<td>1</td>
<td>0</td>
<td>RO</td>
<td>0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>upper</td>
<td>7</td>
<td>2</td>
<td>RW</td>
<td>0x04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSR</td>
<td>config2</td>
<td>0x0011</td>
<td></td>
<td>RO</td>
<td>0x20</td>
<td>Config reg 2</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>full</td>
<td>15</td>
<td>0</td>
<td>RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSR</td>
<td>config3</td>
<td>16</td>
<td>0x0100</td>
<td>ROX</td>
<td>0x1f</td>
<td>Config reg 3</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>full</td>
<td>15</td>
<td>0</td>
<td>RS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSR</td>
<td>config4</td>
<td>0x0010</td>
<td></td>
<td>RS</td>
<td></td>
<td>Config reg 1</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>low</td>
<td>1</td>
<td>0</td>
<td>RS</td>
<td>0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>upp</td>
<td>15</td>
<td>2</td>
<td>RR</td>
<td>0x04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CSR App: Automated Register Verification

- Formal proofs are exhaustive
  - Checks for all possible sequences of RD/WRs in any order
  - Checks for all register addresses

- Conceptually, the following non-deterministic trace is considered by formal for proving address A

Non-deterministic # (zero to infinite) of Rd/Wr access to any address except A
Read from address A
Write D to address A
Potential Bugs Discovered

CEX: Part of the AHB Slave went in the wrong PD

CEX: Status from Ingress passing through other domain

CEX: User did not clamp output of scheduler; X got to the CSR
LPV-App with Scoreboard
Cross Power Domain FIFOs

- Popular in LP designs
- The main way to pass data and commands between domains
- FIFO is split into 2 parts, each lives in a different Power Domain
- When one half is turned off, the other half should not be disturbed or generate garbage

Different methods:
- One half can go off with data in the FIFO, resume sending the data when its back on, need to retain the memory and pointers
- Either half is turned off only when the FIFO is totally empty, can do reset after power on
- Push side can go off when it is empty, but the pop side still sending its data out
LPV Requirements:

- Maintain data integrity while going through legal power cycles:
  - No packet-lose,
  - No change of order
  - No Extra packets
  - No data modification
  - We need Scoreboard
An Example of End-to-End Properties

- Scoreboard, just like in simulation, can be very powerful

- Formal can exhaustively prove that data is not dropped, duplicated, or swapped
Example: Cross Power Domain FIFO

- Glue Logic
- Jasper Formal Scoreboard
- Power State Machine
- Push side
- Pop side
- Glue Logic
LPV-App with X-Prop
LPV-App with X-Prop

- During power-up and power-down sequences, flops and signals without power drives X.
- These Xs are not supposed to reach any input of a power domain that is on.
- LP methodology allows adding clamps to prevent that.

**Issues:**
- Are we putting the clamp on all the need places?
- Are we choosing the right clamp-value?

**X-prop**
- Any unclamped output will propagate X, X-Prop flow can detect these and report them as bugs.
Status

• LPV App available as EA version
  • Includes all power transformations described in power format file
  • Includes 80% of the auto-checks
  • Integration with three Apps (FPV, X-Prop, Conn) completed

• Exercised on 3 ST projects (STB block, GPU, CPU)
  • On-going work, power aware models and running clamp checks
  • Final result expected by Q1’2013
Next Steps

- UPF/CPF source level debug
  - Extend visualize technology to UPF/CPF

- Automatic design optimizations
  - Safe reduction of the number of clamps
  - Reduction of retention logic
Conclusion
Summary

• ST and ST’s verification challenges
• Jasper’s Formal Apps
• Formal App for Low Power Verification
Conclusion

• Formal Technology has a variety of application areas

• Formal Technology can be used by everyone: just find the right App for you

• Low Power Verification is a new and interesting challenge, which we are tackling jointly with success