A Formal Verification App
Towards Efficient Chip-Wide Clock Gating Verification

Presenter: Syed Suhaib

Authors: Prosenjit Chatterjee, Scott Fields and Syed Suhaib
Problem Description

• Mobile Technology - Low Power is important

• Reduce Dynamic Power Consumption
  – Solution: Clock Gating

• Multiple units on a chip with various clock gated islands.
  – Divide and Conquer
Clock Gating Classification

• Functional Clock Gating
  – Turns off unit/functionality
  – Controlled by super-unit / external unit.
  – Used for functional + power saving.

• Non-functional Clock Gating (Our Focus)
  – Activity turns on the lights
  – Used only for power saving
  – Defeature bits used to turn off clock gating for late bugs.
Issues and Risks

• Clock gating usually come up late in the design process.

• Functional verification has higher priority.

• Bugs missed or come very late in ECO process.
  – Very Expensive

• Debugging bugs due to clock gating in silicon can be a nightmare.
Work-Arounds

• Work-around clock gating bugs can be very expensive.
  – Rely on software potentially increasing complexity.
  – “lights out” for large sections of the chip.
  – Big loss on power budget could be un-acceptable.
Prior Techniques

• Traditional simulation is necessary but not sufficient.

• Debugging failures are time-consuming.

• Large effort with traditional model checking
  – Lots of valid constraints needed.
Approach

Two instances of same DUT:
- SFV0 – Ungated Design
- SFV1 – Clock Gated Design.

• All inputs are mapped and tied together.
• Assertions are added on corresponding output signals to be equivalent.
• Identify all non-functional clock-gaters in design.
• Force defeature bits to be high in one instance, and random but stable in other.
Assertions on Outputs

- Control Signals

```assert property ( (sfv0.output_ctrl_a == sfv1.output_ctrl_a));```

- Data Signals

```assert property ( sfv0.output_ctrl_a ->
    (sfv1.output_data_a == sfv1.output_data_a));```
Reset Sequence

• Flops categorized into
  – Resettable flops (Reset by design)
  – Un-initialized flops (driven by some behavior)
• Setup also helps identify X-propagation issues.
• Option to initialize the un-initialized flops and run the flow (constrained setup).
FV Challenges

• Two instances of RTL doubles the size of design under verification.
  – Tricky for formal verification.
  – Use various tricks to help with the proofs.
    • Abstractions
    • Assume-guarantee
    • Intermediate cut-points
    • Proof engines take advantage of symmetry.
FV Challenges

- Identify units not involved with clock gating.
- Abstract units out by proving all signals are equal at its input.
- Add assumptions on output signals to be equal.
SEC App (Jasper)

- Ease of setup.
- Automatically map signals across the two instances.
- Automatically add assumptions to the inputs.
- Automatically add assertions to the outputs.
- Force defeature to be high in one instance, and random but stable in other.
SEC App

• Supports setups in “Wrapper Mode” (widely used at Nvidia).
  – Wrapper mode: Wrapper containing the two instances of DUT.

• Able to reuse assumptions/setup used for Formal Property Verification.
• Debugging:
  – Two instances side by side.
  – Ease of quickly identifying the mismatch and debugging.
  – Created a script which automatically plotted the mismatches between signals until the first failure was found.

• Reduced debugging time by about ~8-10 mins for some designs.
SEC App

• Better proof results:
  – **Blackboxing**: Automatically map I/O for blackboxed modules.
    • Add assertions on inputs of Bboxed modules.
    • Add assumptions on outputs of Bboxed modules.

```bash
check_sec -map -auto -spec dut0.a.ram -imp dut1.a.ram -type bbox_input -tag ram_input
check_sec -map -auto -spec dut0.a.ram -imp dut1.a.ram -type bbox_output -tag ram_output
```
SEC App

- **Cutpoints**: Add cutpoints to internal signals for assume-guarantee based approach.
- Once proven, use as assumptions in same setup.
- **Two features**: 
  - Single sided cutpoints – Remove logic from one partition when proven.
  - Double sided cutpoints – Remove logic from both partitions when proven.

```
check_sec -map -spec dut0.b.counter -imp dut1.b.counter -tag my_counter -with_attr cutpoint
```

*Feature of Jasper SEC App*
Application and Results

- Applied on multiple chips (GPUs, Tegras).

- Chip divided into multiple units/sub-units.

- Each unit/sub-unit evaluated for clock gating islands.
Applications and Results

- Application on “Tegra”:
  - 50 setups.
  - Setup per unit took 15~30 mins. (Wrapper mode)
  - Run by 20 engineers (FV, DV and Designers). Easy to use and were quickly trained.
  - Verification time: 1~3 weeks
    - Adding constraints
    - Debugging CEX
  - ~40 bugs found (~50% after high simulation coverage)
## SEC App vs FPV

<table>
<thead>
<tr>
<th>Name</th>
<th>Flops</th>
<th>Clock Gate Domains</th>
<th>FPV Proof Convergence</th>
<th>SEC Proof Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit A</td>
<td>25k</td>
<td>4</td>
<td>30%</td>
<td>100%</td>
</tr>
<tr>
<td>Unit B</td>
<td>35k</td>
<td>4</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Unit C</td>
<td>25k</td>
<td>8</td>
<td>70%</td>
<td>100%</td>
</tr>
<tr>
<td>Unit D</td>
<td>35k</td>
<td>9</td>
<td>40%</td>
<td>60%</td>
</tr>
<tr>
<td>Unit E</td>
<td>45k</td>
<td>14</td>
<td>25%</td>
<td>65%</td>
</tr>
</tbody>
</table>
Types of Bugs Found

- Missing terms in clock_enable used for clock gaters causing signals to not get flopped correctly.

```
Clk_en = vld_1 || vld_2 || vld_3
```

- Use of incorrect clock_enable signal.

```
 @(posedge clk_en_a2) vld_a1_data2 <= a1_data2;
  Instead of:
  @(posedge clk_en_a1) vld_a1_data1 <= a1_data1;
```
Types of Bugs Found (2)

• Hang Case: clock_enable is stuck due to bad logic driving it and not able to propagate valid value to output.

@ (posedge clk_en_stuck) vld_a3 <= vld_a2;
Signoff Checklist

• Resolve all cex
• Re-confirm all non-functional clock gaters part of setup.
• Re-confirm that all outputs have equivalence properties
• Remov ‘-nonResettableRegs 0’ for reset coverage
• Increase the per-property runtime to at least 10 h
• Review input constrains added with the designer. Enable in simulation (if possible).
• Prove all equivalent asserts to an acceptable depth.
Acknowlegement

• Jasper Design Automation
• Engineers in GPU, Tegra teams at Nvidia for driving this to closure.
Thank You