Finding Glitches Using Formal Methods

Yan Peng¹  Ian W. Jones²  Mark R. Greenstreet¹

¹University of British Columbia, Vancouver, BC, Canada
²Oracle Labs, Redwood City, California, USA

May 9th 2016
Warming up

- A Small Example
- Glitch Detection Using Ternary Simulation

- Our Glitch Hunting Tool
- Experimental Results
- Conclusion
Is the netlist equivalent to the RTL?

- **YES!** When using standard logical-equivalence checking
- Logical equivalence formulation:
  “For every input from \{T, F\}, the netlist produces the same output as the RTL.”
- Signal naming:
  - S – signals Synchronous to output clock domain
  - N – signals Non-synchronous to output clock domain
Glitches caused by non-synchronous signals

- Standard logical-equivalence is not enough, e.g., when S_N1_VALID is 0:
  - RTL: permits only S1 to pass to the MUX output, S_OUT
  - netlist: allows a glitch to propagate from N1 to S_OUT
Ternary logic values \{T, F, X\} facilitate detection of glitch paths
Our Formal Methods Glitch Hunting Tool

• Warming up

• A Formal Methods Glitch Hunting Tool using ACL2
  ▶ Tool Architecture and Work Flow
  ▶ The Formal Definition

• Experimental Results

• Conclusion
For a state-bit, $q$, let $S_q$ denote the synchronous inputs to the combinational logic for the next-state of $q$, and $N_q$ denote the non-synchronous inputs. Let $\mathbb{B} = \{0, 1\}$, and $\mathbb{B}^X = \{0, 1, X\}$.

$$\text{glitchFree}(q) = \forall S_q \in \mathbb{B}^*. \forall N_q \in \mathbb{B}^{X*}. \ (\text{next}_q, \text{net}(S_q, N_q) = X) \Rightarrow (\text{next}_q, \text{RTL}(S_q, N_q) = X)$$

(1)
Outline

- Warming up
- Our Glitch Hunting Tool

**Experimental Results**
- Real Designs
- Performance

- Conclusion
## Experimental Results: Real Designs

<table>
<thead>
<tr>
<th>Description</th>
<th>Module A</th>
<th>Module B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RTL file size</strong></td>
<td>0.7M</td>
<td>2.5M</td>
</tr>
<tr>
<td><strong>netlist file size</strong></td>
<td>8.2M</td>
<td>5.4M</td>
</tr>
<tr>
<td># state-bits</td>
<td>22473</td>
<td>4439</td>
</tr>
<tr>
<td># state-bits w N&lt;sup&gt;1&lt;/sup&gt;</td>
<td>1253 (5.6%)</td>
<td>957 (21.6%)</td>
</tr>
<tr>
<td># Glitches found</td>
<td>0</td>
<td>148</td>
</tr>
</tbody>
</table>

- Modules have multiple clock domains
- Found all previously known glitches
- Discovered glitch paths that were benign due to unstated assumptions in the RTL

<sup>1</sup>N stands for non-synchronous input
Theorem checking is compute intensive, but each fan-in tree can be run in parallel.

Preprocessing overhead expected to grow linearly with size of netlist and RTL Verilog.
Conclusion and future work

- Implemented a tool using **SAT solving** and **theorem proving** to detect synthesis inserted glitches
- Provide a **formal definition** of the required glitch-free property
- Successfully demonstrated our tool on **real industrial designs**
- Future work:
  - Automatically generate simulation scripts for glitch found
  - Larger designs
  - Integrate the method into chip design flow
Thank You! Questions?
Glitches caused by non-synchronous signals

- **RTL:** \( N_2 \) is specified to generate only \( N_{\text{OUT}} \)
- **Netlist:** even when \( S_{\text{N1 VALID}} = 0 \), a posedge on \( N_2 \) can cause a glitch to propagate to \( S_{\text{OUT}} \)
Combinational logic fan-in trees often have 100+ inputs

Challenge:

- succinctly present glitch path results