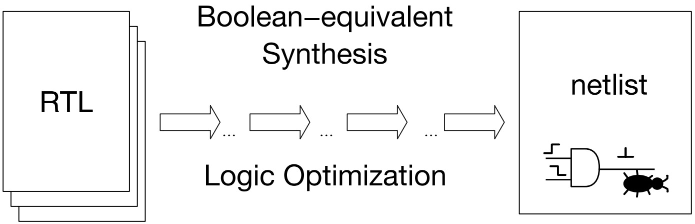
Finding Glitches Using Formal Methods

Yan Peng[1] Ian W. Jones[2] Mark R. Greenstreet[1]

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

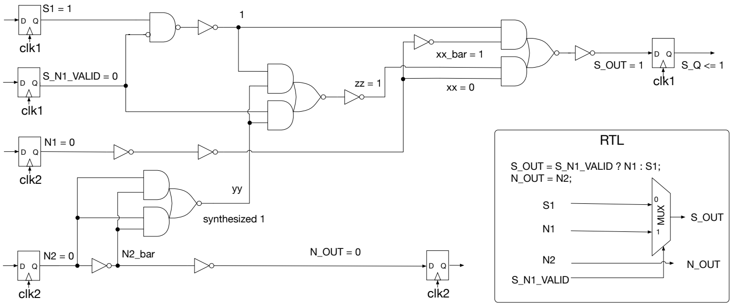
May 9th 2016

Outline



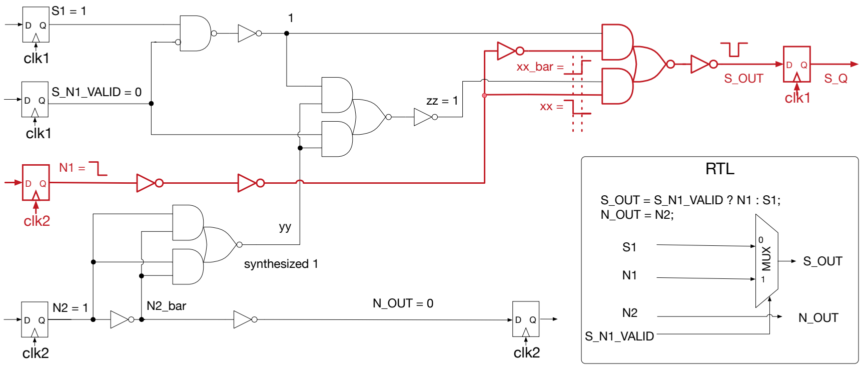
- **Warming up**
  - **A Small Example**
  - **Glitch Detection Using Ternary Simulation**
- Our Glitch Hunting Tool
- Experimental Results
- Conclusion

Is the netlist equivalent to the RTL?



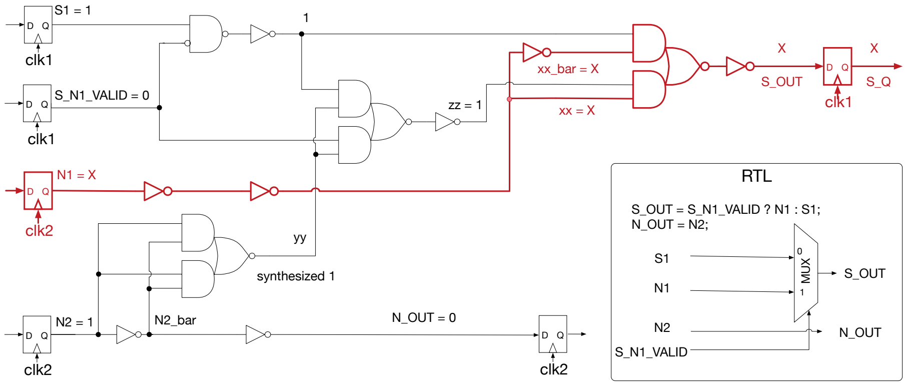
- **YES!** When using standard logical-equivalence checking
- Logical equivalence formulation:
  "For every input from {T, F}, the netlist produces the same output as the RTL."
- Signal naming:
  - S – signals Synchronous to output clock domain
  - N – signals Non-synchronous to output clock domain

Glitches caused by non-synchronous signals



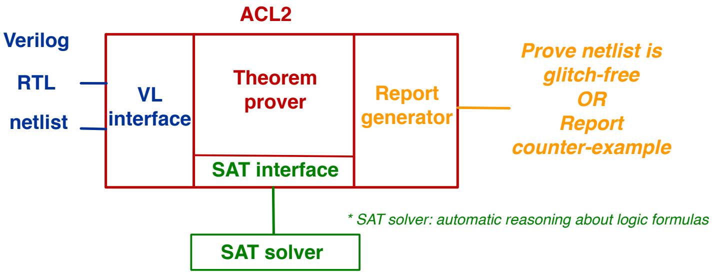
- Standard logical-equivalence is not enough, e.g., when `S_N1_VALID` is 0:
  - RTL: permits only `S1` to pass to the MUX output, `S_OUT`
  - netlist: allows a glitch to propagate from `N1` to `S_OUT`

Using ternary simulation to detect glitch



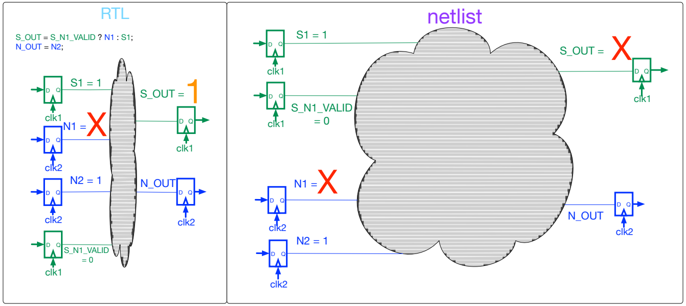
- Ternary logic values {T, F, X} facilitate detection of glitch paths

Our Formal Methods Glitch Hunting Tool


ACL2
Verilog
RTL
netlist
VL interface
Theorem prover
SAT interface
Report generator
Prove netlist is glitch-free OR Report counter-example
* SAT solver: automatic reasoning about logic formulas
SAT solver

- Warming up
- **A Formal Methods Glitch Hunting Tool using ACL2**
  - **Tool Architecture and Work Flow**
  - **The Formal Definition**
- Experimental Results
- Conclusion

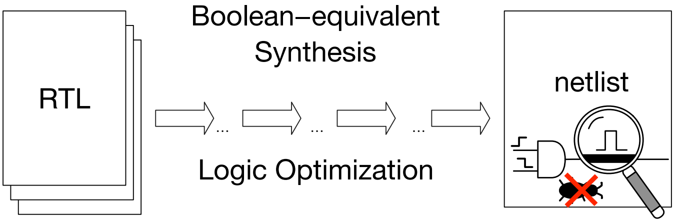
The Formal Definition



- For a state-bit, q, let $\mathcal{S}_q$ denote the synchronous inputs to the combinational logic for the next-state of q, and $\mathcal{N}_q$ denote the non-synchronous inputs. Let $\mathbb{B} = \{0, 1\}$, and $\mathbb{B}^{\mathbf{X}} = \{0, 1, \mathbf{X}\}$

$$\begin{aligned} \text{glitchFree(q)} \ = \ & \forall \mathcal{S}_q \in \mathbb{B}^*. \ \forall \mathcal{N}_q \in \mathbb{B}^{\mathbf{X}*}. \\ & (\text{next}_{q,\text{net}}(\mathcal{S}_q, \mathcal{N}_q) = \mathbf{X}) \Rightarrow (\text{next}_{q,\text{RTL}}(\mathcal{S}_q, \mathcal{N}_q) = \mathbf{X}) \end{aligned} \tag{1}$$

Outline



- Warming up
- Our Glitch Hunting Tool
- **Experimental Results**
  - Real Designs
  - Performance
- Conclusion

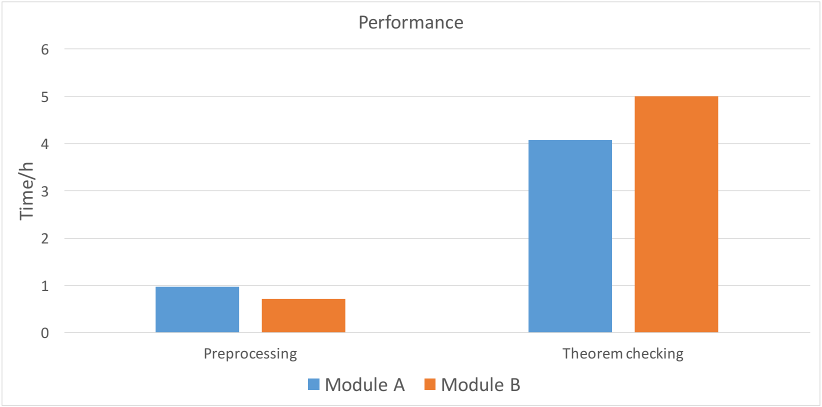
Experimental Results: Real Designs

	Module A	Module B
Description	control module	interface module
RTL file size	0.7M	2.5M
netlist file size	8.2M	5.4M
# state-bits	22473	4439
# state-bits w N[1]	1253 (5.6%)	957 (21.6%)
# Glitches found	0	148

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

[1] N stands for non-synchronous input

Some remarks about performance



- 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

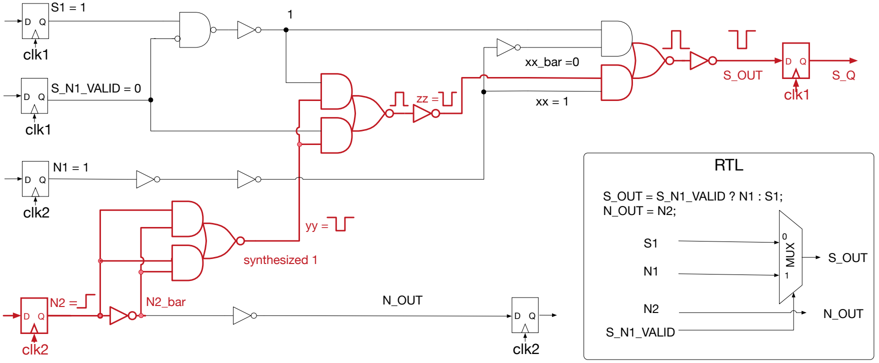
Conclusion and future work

- 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

Finding Glitches Using Formal Methods

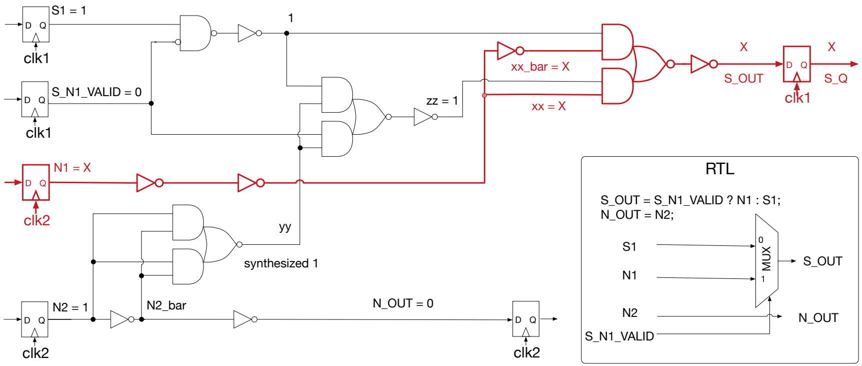
Thank You! Questions?

Glitches caused by non-synchronous signals



- RTL: `N2` is specified to generate only `N_OUT`
- netlist: even when `S_N1_VALID` is 0, a posedge on `N2` can cause a glitch to propagate to `S_OUT`

Glitch Demonstration - Glitch Path Extraction



- Combinational logic fan-in trees often have 100+ inputs
- Challenge:
    - succinctly present glitch path results