Clock Gater Verification Using Formal Property Checking

Alfonso Urzua

Advanced Micro Devices, Inc.

April 27, 2017

Boston

Agenda

Formal Verification Overview

Clock Gater Design

The Challenge of Clock Gater Verification

How can Formal help with the Challenge?

Clock Gater Formal App

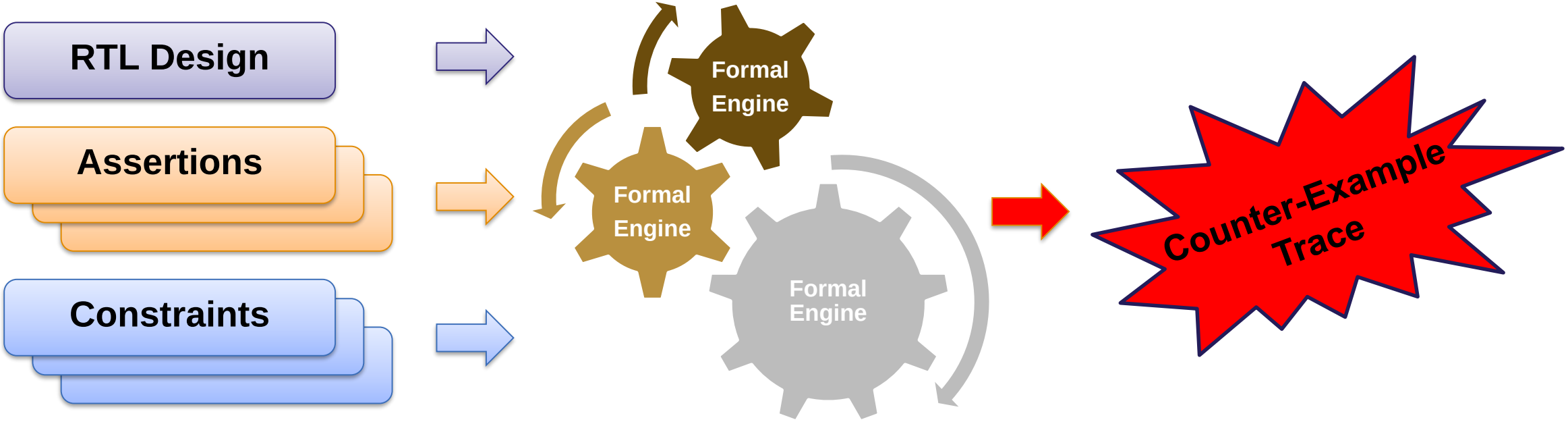
Results and Future Work

Summary

Formal Verification Overview

What is Formal Verification?

- ***Formal verification*** is a systematic process of ensuring, through exhaustive algorithmic techniques, that a design implementation satisfies the requirements of its specification.



Simulation vs Formal

- Comparing *simulation* vs *formal* with respect to *controllability* and *observability*

	Simulation	**Formal**
Controllability	Test vectors stimulate design Reach coverage goals	No input stimulus required Input constrains
Observability	Faulty behavior visible through assertions, checkers, monitors. Limited by controllability	Faulty behavior visible through assertions, checkers, monitors. Supported by controllability

Formal Verification Use Models

Not an exhaustive list!

- Data path vs control logic designs
- Early design debug (pre-testbench)
- General bug hunting
- Verification of "Hot" design areas
- Timing fixes
- Full IP verification
- Late bug finds (pre or post-silicon)
- Coverage closure
- Automated solutions (i.e., Formal Apps)

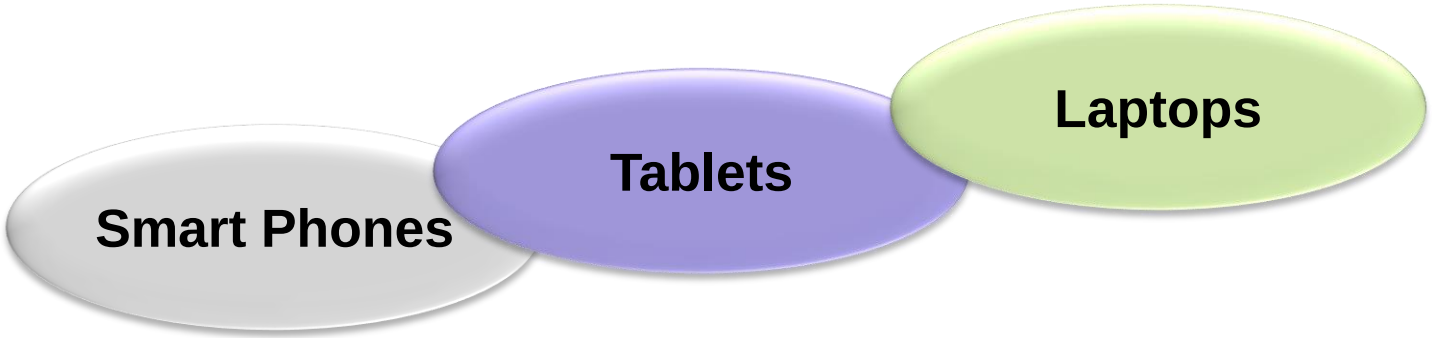
Limitations of Formal Verification

- Design state space size
- Proof times could be long unless proper constraints are applied
- Relative to simulation, formal is still an emerging technology
- However, formal can still be a practical complement to simulation for well-defined design targets

Clock Gater Design

Why is Clock Gating Important?

- Many of today's portable electronic devices depend on low power IC designs



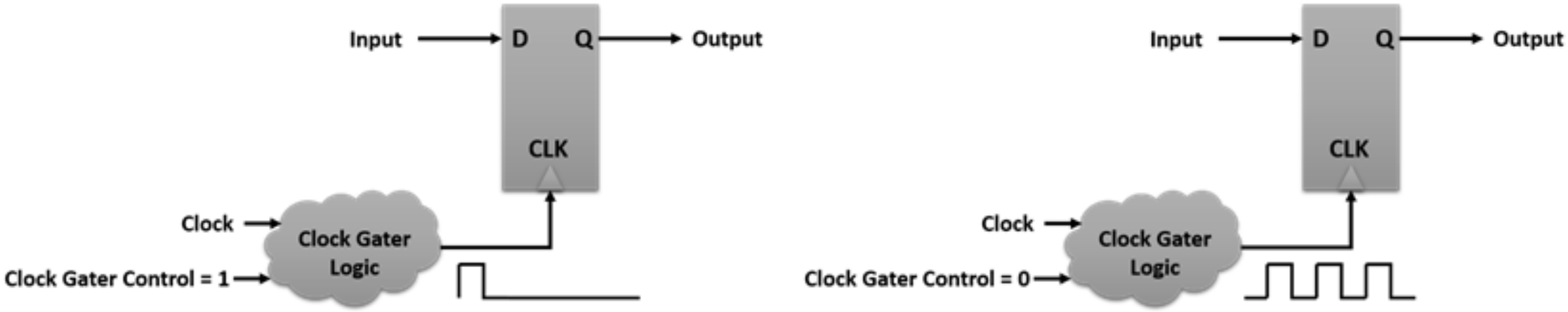
- RTL teams usually follow certain local power requirements
- Part of those requirements are achieved by the use of *clock gating*
- *Clock gaters* limit dynamic power dissipation by disabling clocks of selected flops

Types of Clock Gaters

- There are two types of *clock gaters*: *power* and *functional*
- **Power Clock Gaters**: only gate the clock when flop inputs are constant
- **Functional Clock Gaters**: gate flop inputs when we don't want downstream logic to consume
- **We only focus on *power clock gaters* on this presentation**

Power Clock Gaters

- **Power Clock Gaters**: only gate the clock when flop inputs are constant



Clock Gater Control = 1 gates the clock, assuming flop input is held constant
Clock Gater Control = 0 clock runs free when flop input is changing value

The Challenge of Clock Gater Verification

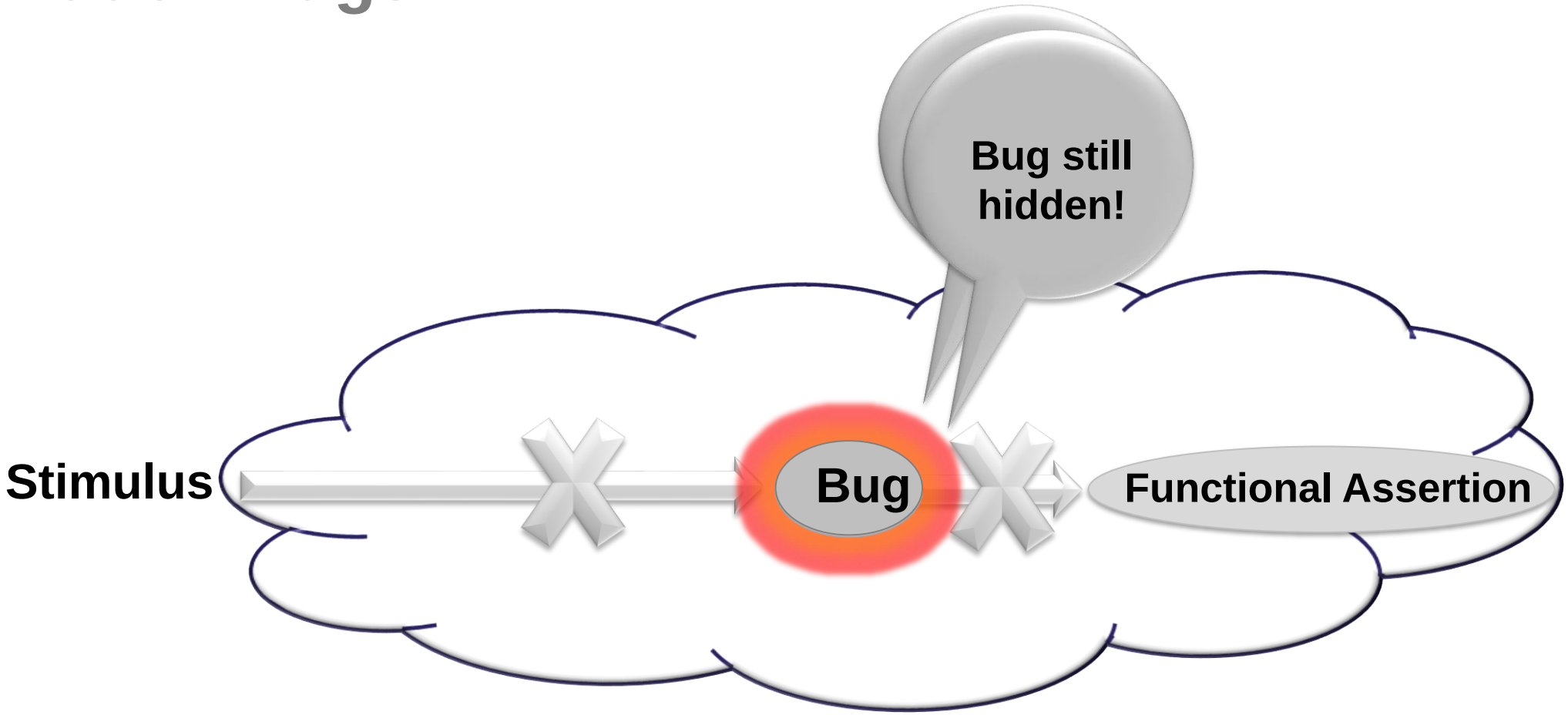
Clock Gater Verification Goal

- Qualifying clock gater design intent can be seen from different perspectives:
  1. Make sure local power requirements are met
  2. Make sure functional clock gater intent always holds true
  3. Make sure overall functional intent is preserved in the presence of power clock gaters
  4. Make sure power clock gater intent always holds true
- Our focus is on the fourth goal above, making sure the intent of power clock gaters always holds true

Hidden Bugs!

- There might be occasions when clock gater verification is not part of a testplan!
- Instead, it is usually addressed through featured-based verification infrastructure (i.e., assertions, checkers, stimulus, coverage).
- The potential problem of not having assertions that directly check clock gater intent:
  1. Hidden bugs!
  2. Larger debug effort

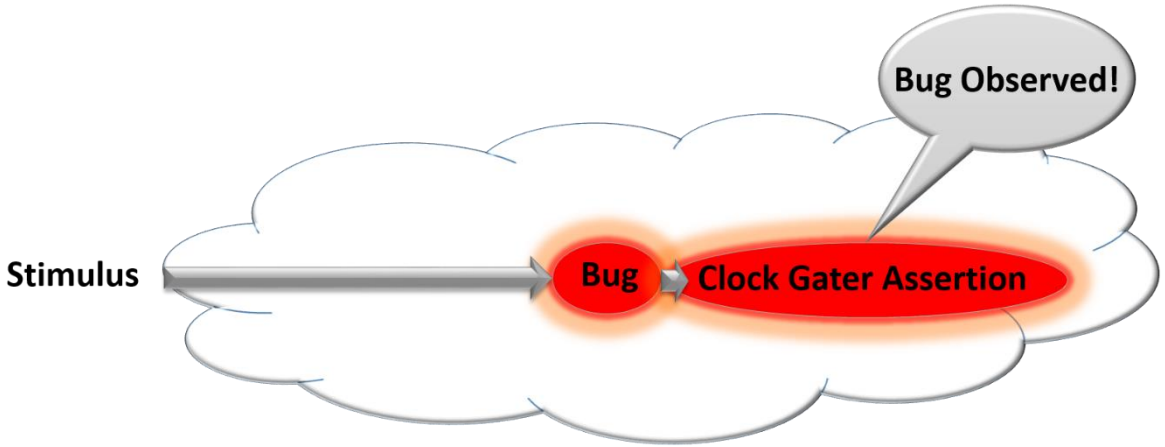
Hidden Bugs!



Clock Gater Assertions

What would a power clock gater assertion look like?

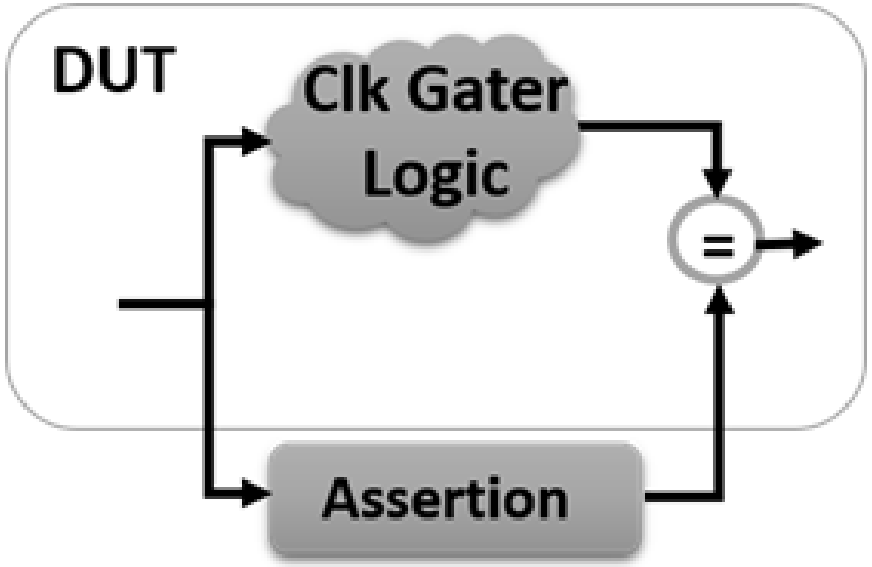
```
// Instance of a clock gated flop
FFG Flop(.C(CLK),.R(RESET),.D(IN),.Q(OUT),.CLKEN(CLKEN));

// Clock gater assertion
asrt_cg: assert property(IN != OUT |-> CLKEN == 1'b1);
```



How can Formal help with the Challenge?

Formal Property Checking

- **Formal Property Checking**, also called ***Model Checking***
- Formal technique that uses ***properties/assertions*** as a way to specify an ***intent***



What are Properties?

- *Properties* express a design intent
- *Properties* are made of temporal logic expressions. Ex:

```
Antecedent |-> Consequent
```

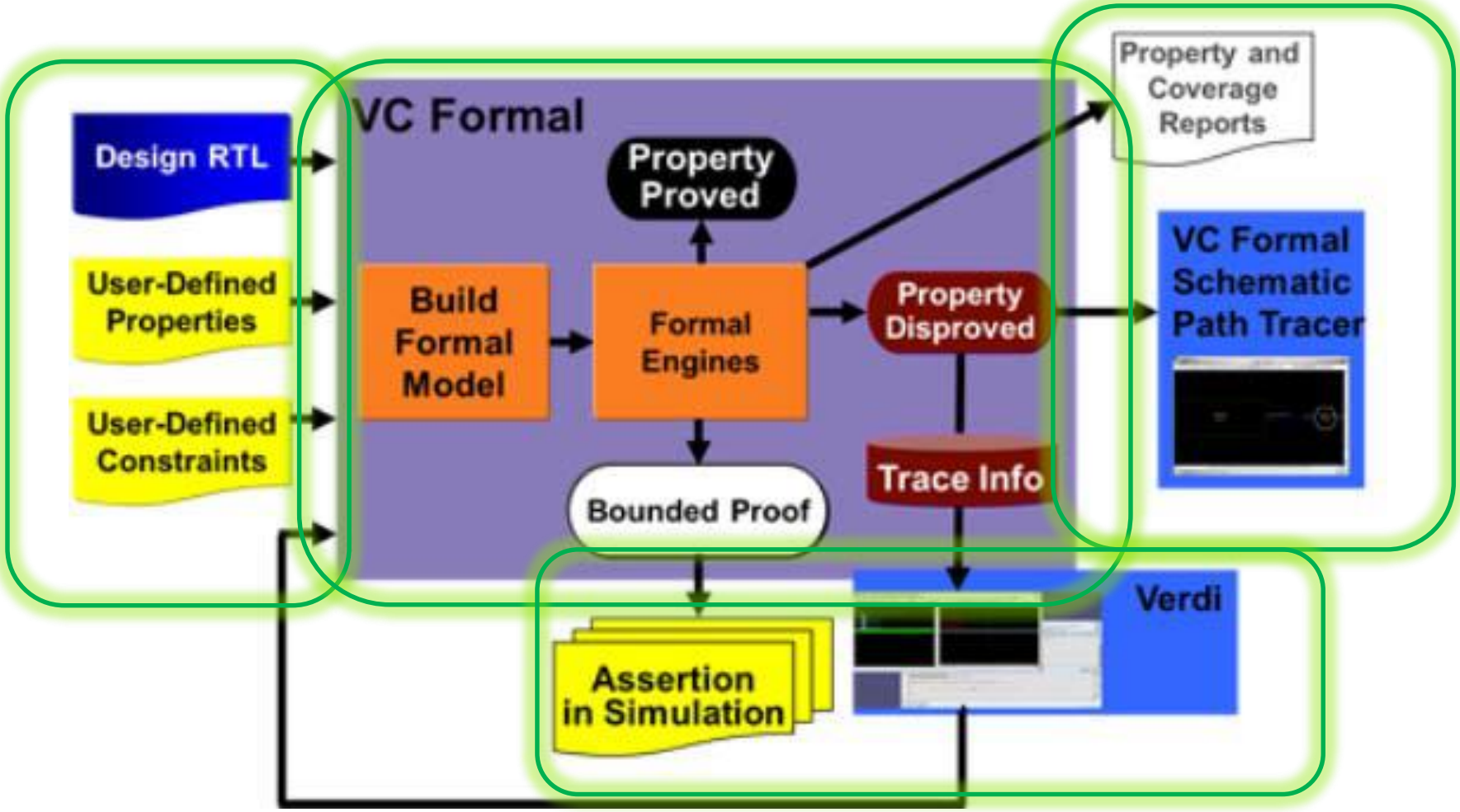
- Which is similar to our clock gater assertion:

```
// Clock gater assertion
asrt_cg: assert property(IN != OUT |-> CLKEN == 1'b1);
```

- A *Formal Property Checking* tool will help make sure *properties* hold true, otherwise counter-example traces are generated

VC Formal Property Checker

- An example of Formal Property Checking tool is VC Formal Property Checker

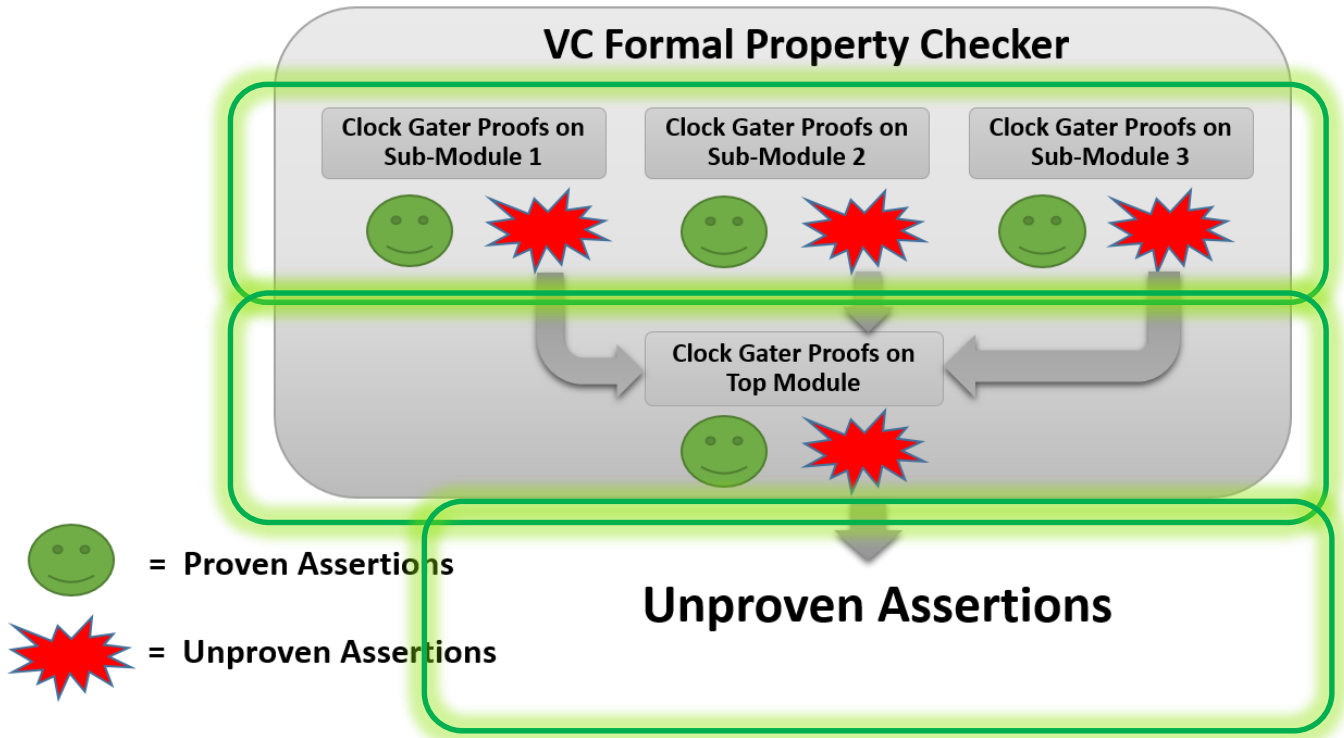


Clock Gater Formal App

Approach

- Create an assertion describing the intent of each clock gater
- Divide and conquer: Split DUT into sub-modules
- For each sub-module, use VC Formal Property Checker to run *unconstrained* formal proofs on each clock gater assertion
- Unresolved or failing proofs are run at a higher level module (new neighbor modules help further constrain these proofs)
- The remaining unresolved or failing proofs become the real work
  - requiring deeper understanding of the design implementation
  - and most probably some input constraints!

The App Flow



- Threads are spawn, each running VC Formal Property checker on a sub-module
- Unresolved and failing proofs are fed to another script running at a higher module level
- The app generates a detailed list of unproven assertions that need further analysis

Scripting the Flow

- In order to automate the flow, script it out using Ruby and Tcl
- Run the script as a property checker with fml_mode_on set to true

```
# Initial Setup and Configuration
set_app_var fml_mode_on true

# Read in the design and set top level module
set top sub_module_1
read_file -top $top -sva -format sverilog -vcs …
```

- Define clock and reset

```
# Reset
create_reset reset -high

# Clock definition
create_clock CLK -initial 0 -period 10
```

Scripting the Flow

- Simulate until signal activity settles

```
# Run the simulation
sim_run -stable
```

- Run proofs and generate a report of unproven assertions to be used for further processing

```
# Run proof
check_fv -property [get_props "*power_clk_gating_check*" -usage assert]

# Report passing and unproven assertions and create output file.
report_fv -list > unproven_assertions_submodule_1.txt
```

Results

Low Hanging Fruit: 80% Proofs!

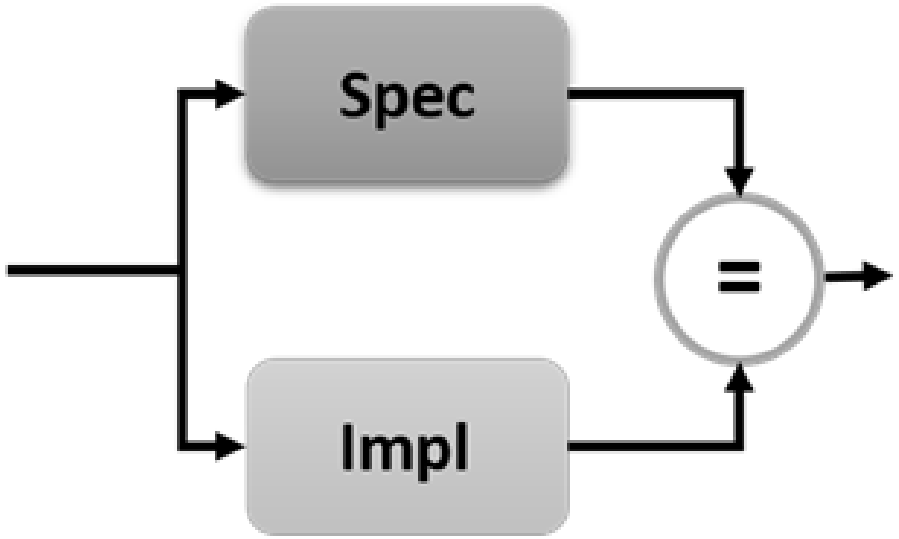
- Successfully proved an average of 80% of power clock gater assertions, without constraints
- Which means, verification is complete, and can look into running without them in simulation, or at least partially!
- Focus on the smaller set of unproven assertions

Module	Clock gaters	Property Checker Proofs	Proof time
Sub-module 1	1.2K	60%	2.7hr
Sub-module 2	1.3K	70%	2.5hr
Sub-module 3	1K	99%	3hr
Top module	0.8K	20%	4hr

Future Work

Closing the Gap Using VC SEQ App

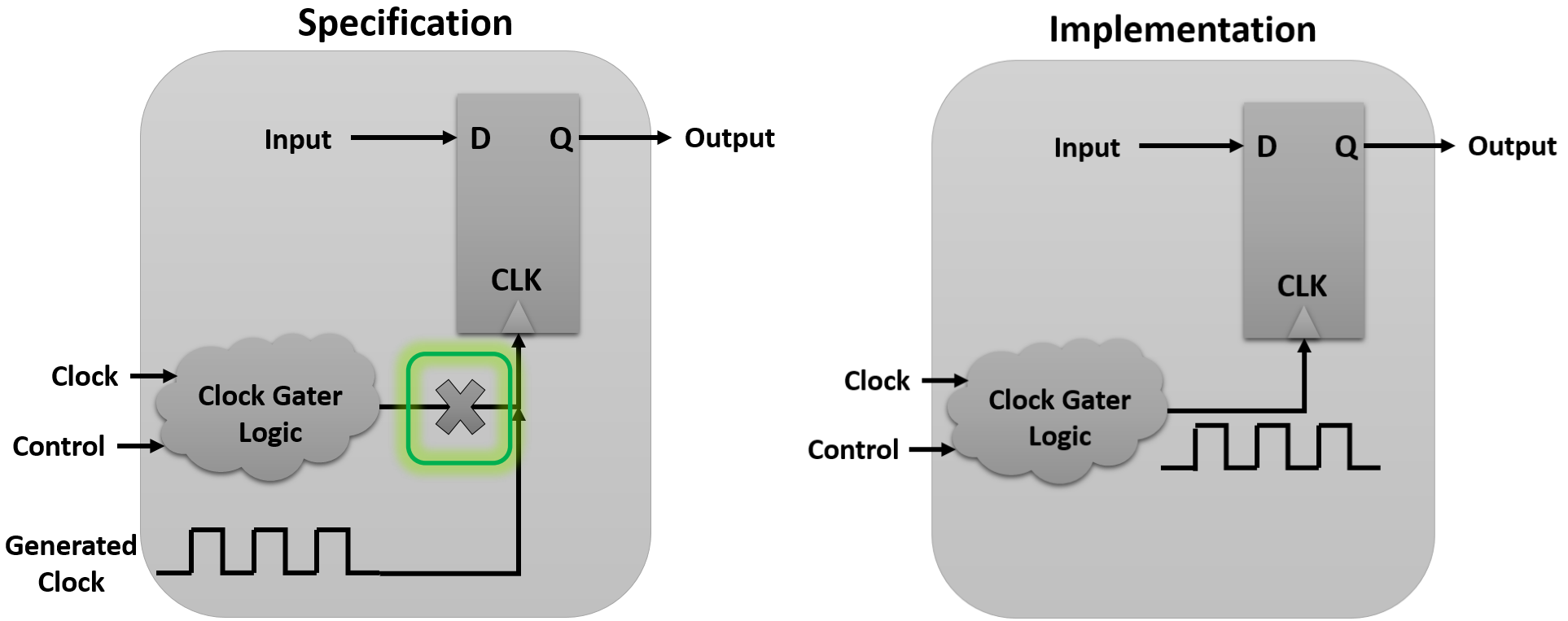
- *Sequential Equivalence Checking* compares two designs with the same primary inputs, outputs, and sequential depth



- One design is called *specification* while the second is the *implementation*
- Produce identical outputs given identical inputs on a cycle by cycle basis
- A sequential equivalence tool like VC SEQ App would create a counter-example trace when a mismatch is found

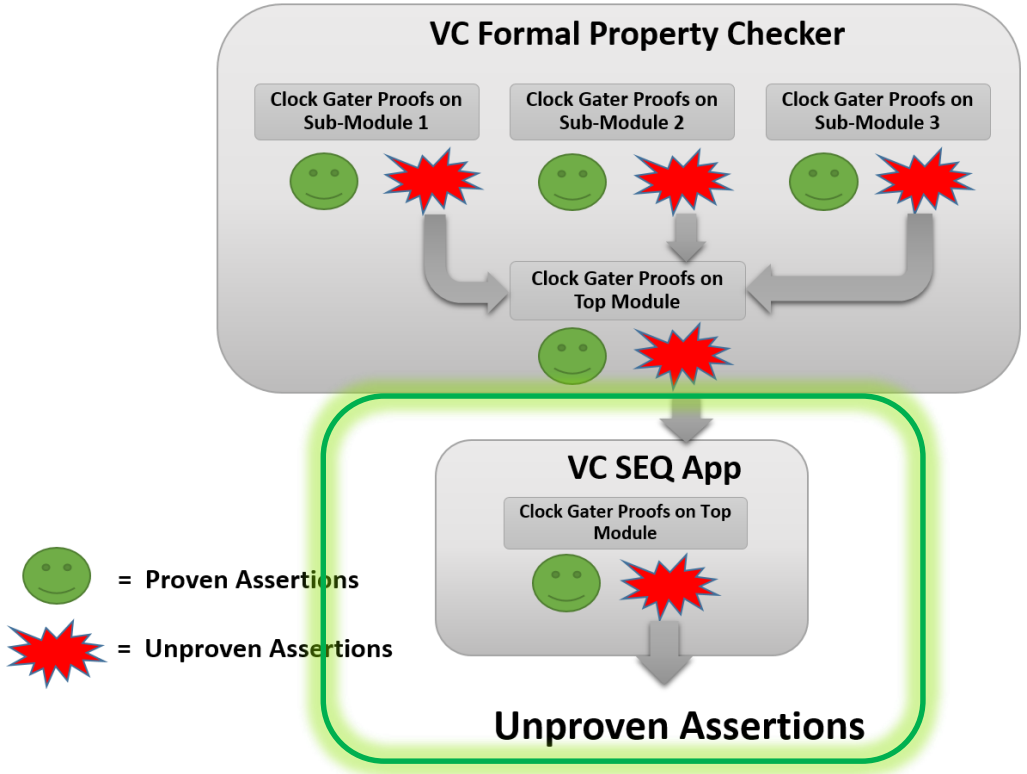
Specification vs Implementation

- To verify clock gaters using sequential equivalence checking, *specification* and *implementation* are defined as follows:



Adding VC SEQ App to the Flow

- To integrate into the App flow:
  - The VC SEQ App would consume the unproven assertions fed by the VC Formal Property Checker
  - Generating a list of potentially smaller unproven assertions



Summary

Summary

- Formal verification can be a powerful complement to simulation
- It can be applied via different use models
- Formal Apps is a use model that helps automate a formal verification flow
- An example where automation can be applied is clock gater verification
- *Formal Property Checking* can be used in the automated flow to verify clock gaters
- Running unconstrained proofs on all clock gater assertions showed an average of 80% proof success
- An opportunity is created to fully or partially exclude assertions from simulation and/or emulation models, once a formal proof exists
- *Sequential Equivalence Checking* could offer a way to close the unproven clock gater assertion gap

AMD

snug
Synopsys Users Group
Boston

Thank You

Disclaimer and Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.
The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.
AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.
AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
ATTRIBUTION