# Speedup Silicon Issue Debug with VC Formal

Wayne Yun

Advanced Micro Devices, Inc.

April 21, 2017

Canada

# Agenda

Resolving issues found at silicon bring-up

Introduction to formal verification

Adapting Formal Property Verification for silicon bring-up issues

Reproducing an issue with VC Formal

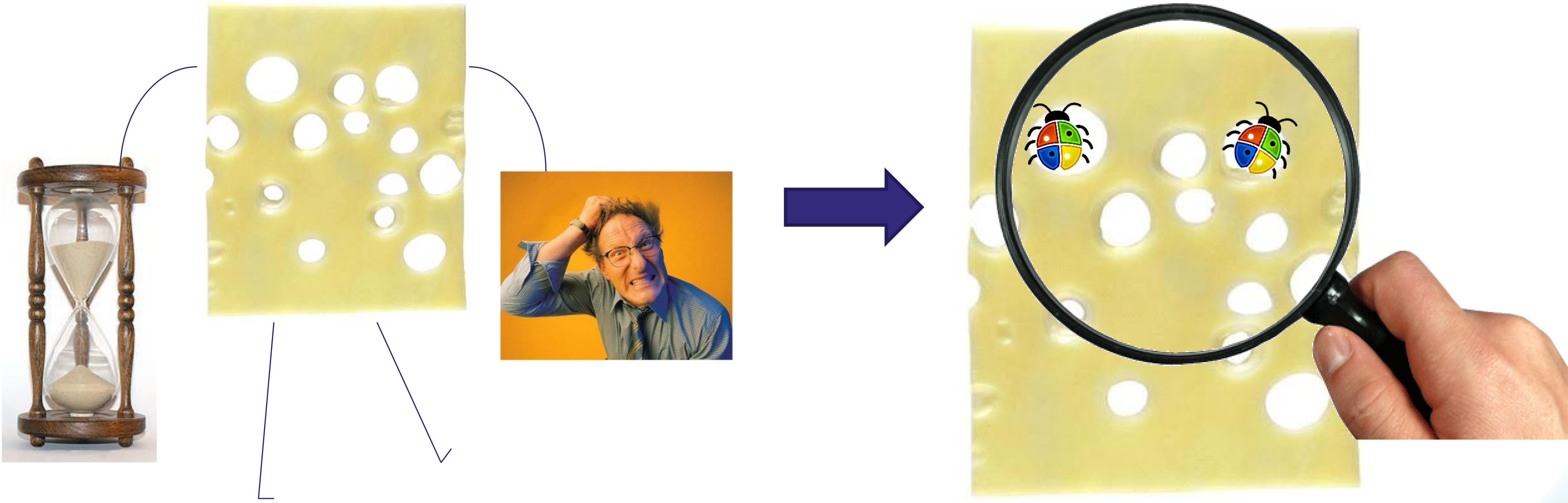
Comparing effort between formal and simulation

Summary

# Resolving Issues Found at Silicon Bring-up

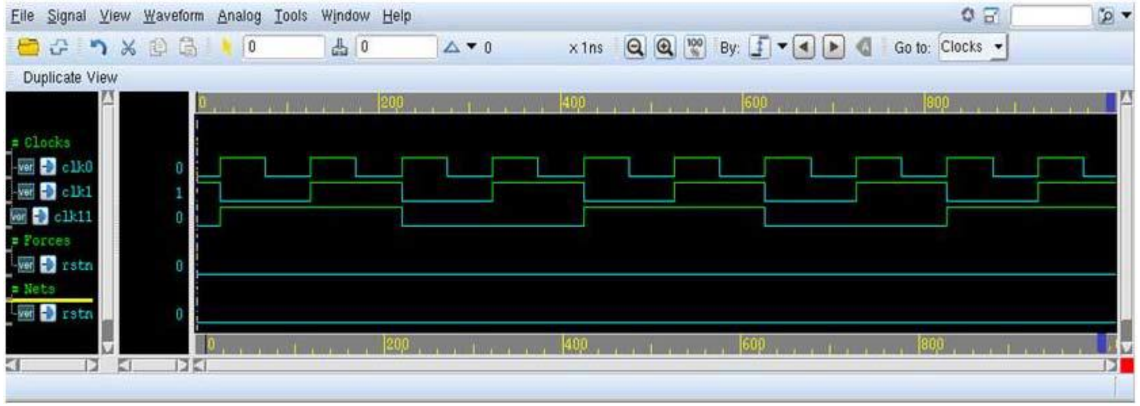
# A Story of A Silicon Issue

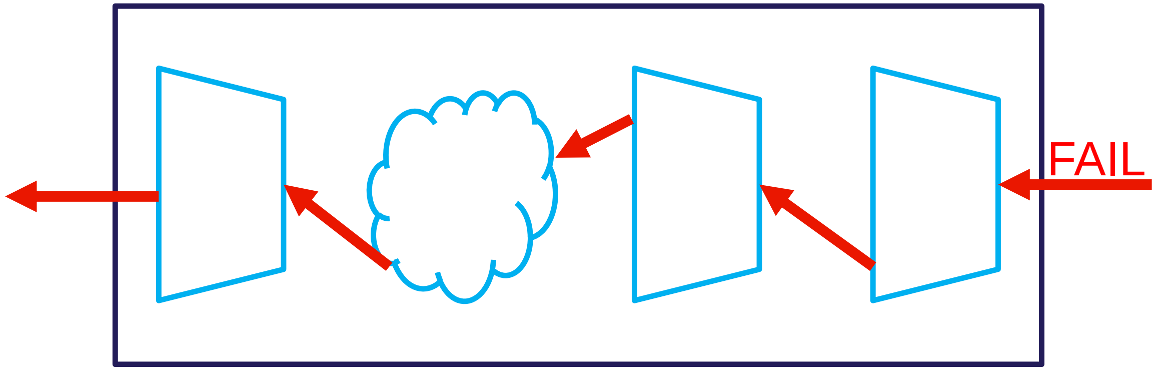
## It Takes Time and Especially Hair to Find Silicon Bugs



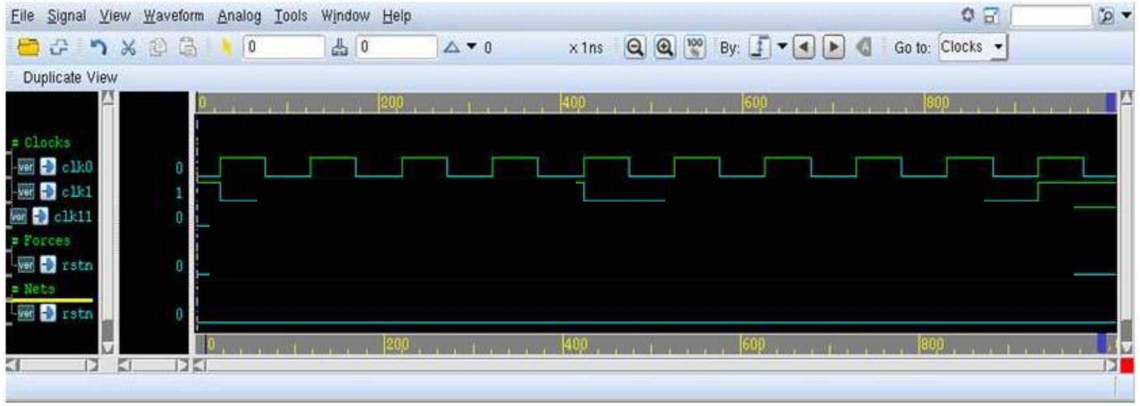
# The Difficulty of Debugging a Silicon Issue

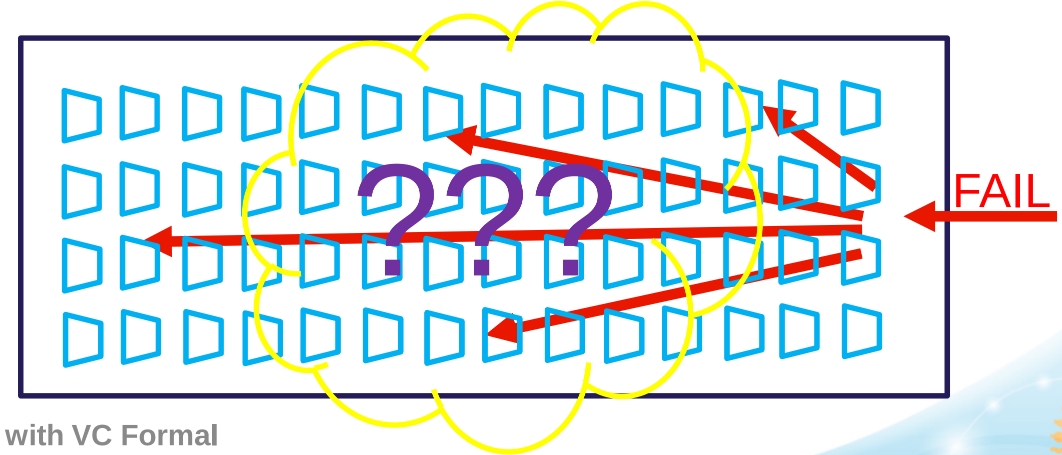
## Functional Simulation





## Silicon Issue Report





# Introduction to Formal Verification

# Formal Verification

- Formal verification is the process of mathematically checking whether a design satisfies some requirements
  - The design usually is the DUT in simulation, or a portion, or a transformation of it
  - Requirements are almost identical to checkers in simulation
  - No need to create any stimulus manually or by programming, all possible and legal stimuli are exhaustively considered
- If the design doesn’t satisfy a requirement, a counter example is found
  - A waveform can be generated from the counter example
  - The debugging process with the waveform is same as simulation
- Logic equivalence check is a formal EDA technology widely deployed
- The branch of formal verification used for silicon issue debug is called formal property verification

# Formal Property Verification

- Formal Property Verification (FPV) is also called property checking or model checking
- A requirement is a property, FPV checks if the design is a model satisfying the property
- If it satisfies, the property is proven
- If it doesn’t, the property is falsified
  - A counter example is generated as a waveform
  - Reason can be found in the waveform

# Formal Properties

- Formal properties could be written in different languages, examples are in SVA
- There are 3 types of properties, all are used in reproducing silicon issue

| Property Example | Formal Semantics | Simulation Semantics |
|---|---|---|
| Assertion:<br>assert property<br>(out_a && out_b); | Requires the expression to be always true for all legal stimuli | Fails the test if the expression is not true |
| Assumption:<br>assume property<br>(in_a && in_b); | Requires all stimuli to satisfy the expression | Same as assert property above |
| Coverage:<br>cover property<br>(sig_a && sig_b); | Checks if there is a stimulus can satisfy the expression | If a scenario of the expression happened, increase the coverage counter |

# Adapting Formal Property Verification for Silicon Bring-up Issues

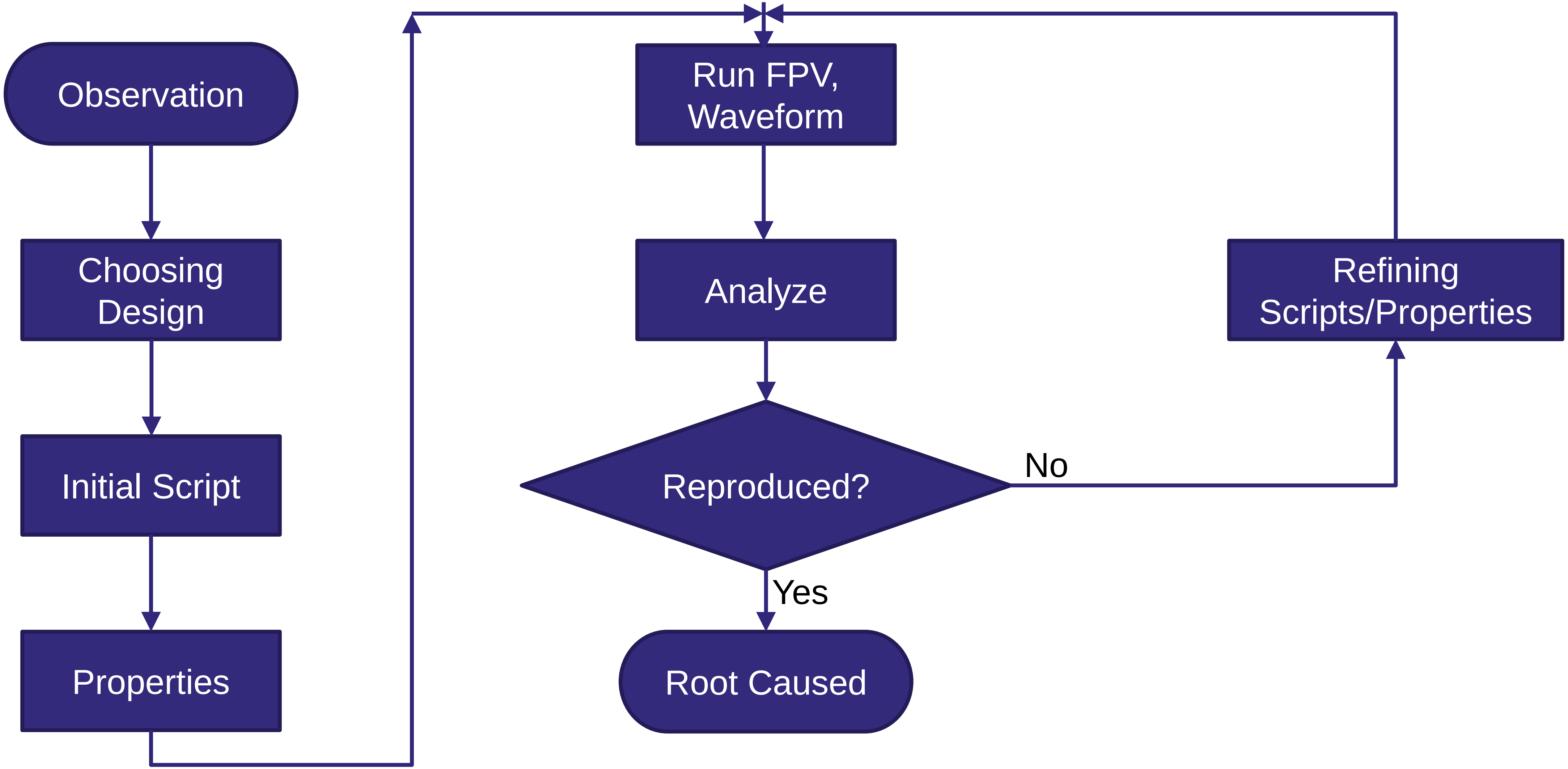
# The Design

- The amount of logic for a chip is too large for an FPV tool
  - The complexity of a formal problem grows exponentially
  - An FPV tool will run out of memory, or cannot solve within reasonable time
- All logic related to the issue should be cut from the chip and used as the design
  - The ideal design should only include logic contributing to the issue to minimize the size
  - Since the cause is unknown, it is impossible to accurately isolate the ideal design
  - The design should be a small portion of the chip with high confidence of covering the issue

# Properties

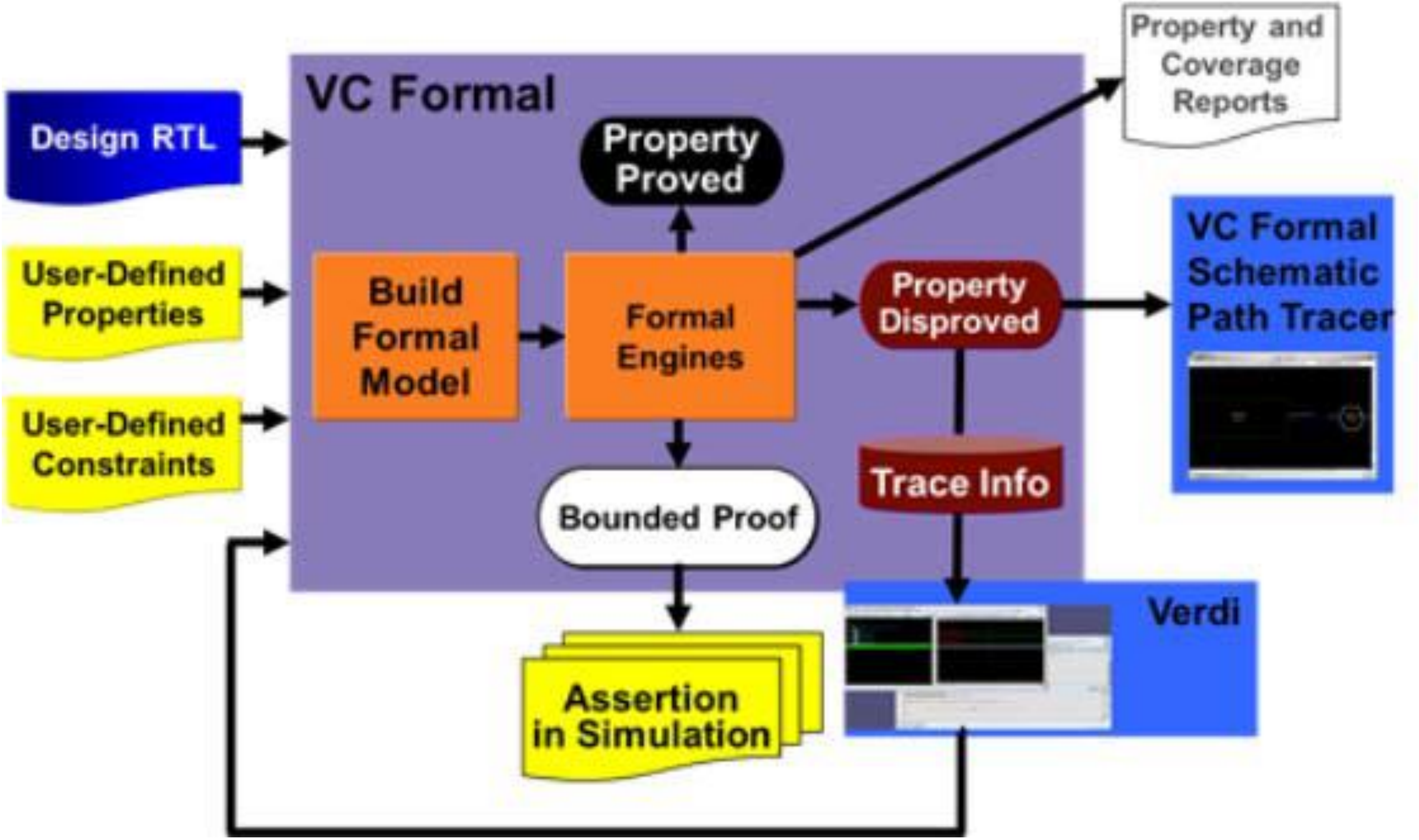
- An assume property
  - Should be defined for clock, reset, behavior of neighbor block, etc.
- A cover property
  - Can be used to describe an **abnormal** behavior of the design
  - In formal, the tool will find out if it is possible to hit the described scenario and how to hit it. The expression of the property is the unintended but observed silicon behavior.
  - In simulation, a hit of a cover property usually shows an intended scenario happened
- An assert property
  - Can be used to describe an **expected** behavior of the design
  - The counter example, waveform, is an exception to the expectation
- Either cover or assert property can be used
  - Choose the one that fits the description provided by the issue report

# The Flow


Observation
Choosing Design
Initial Script
Properties
Run FPV, Waveform
Analyze
Reproduced?
No
Yes
Root Caused
Refining Scripts/Properties

# VC Formal

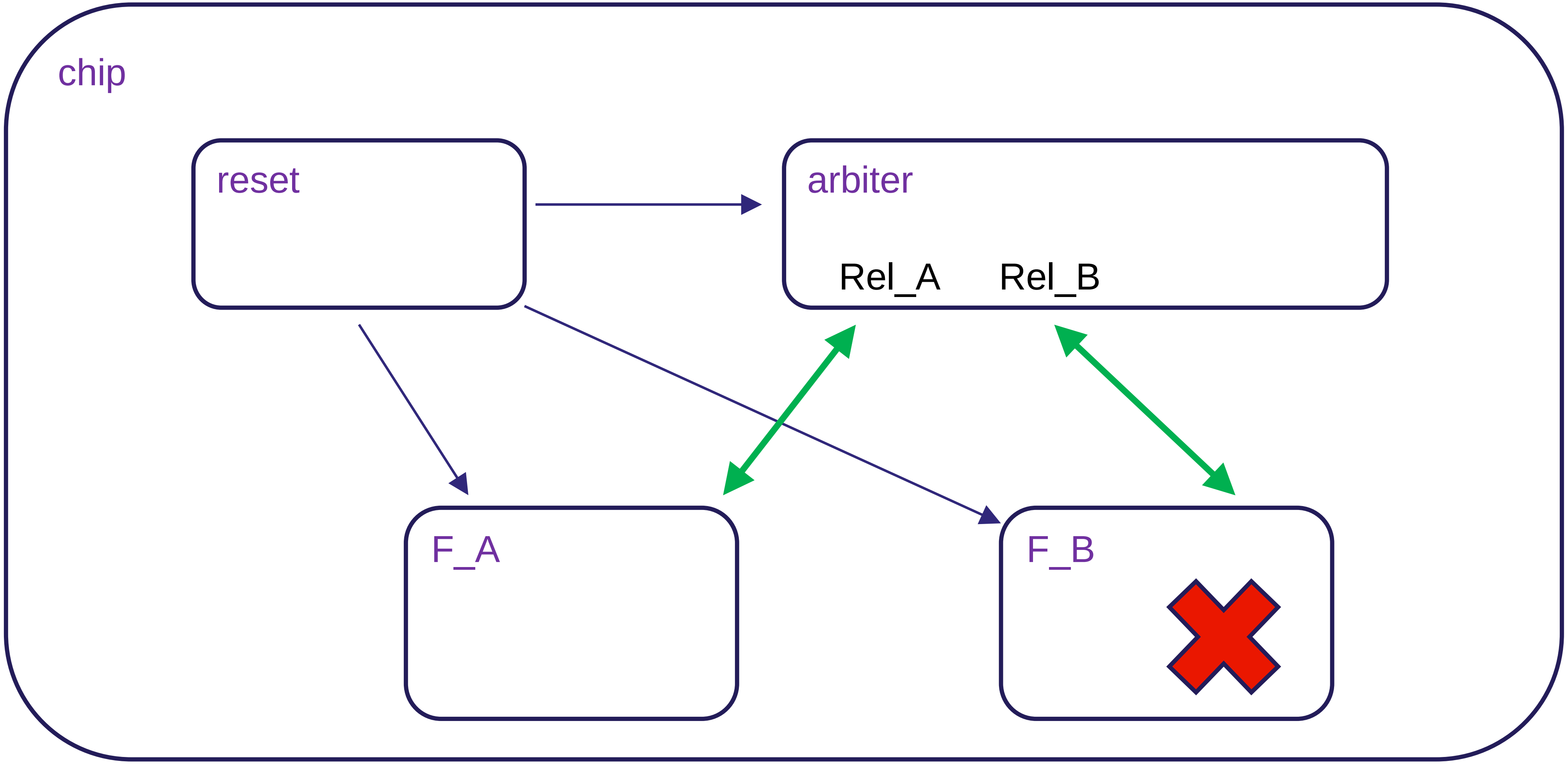
## The FPV Tool from Synopsys



# Reproducing an Issue with VC Formal

# The Issue

## F_A Went to Working State after Reset, however, F_B didn’t



# Which to Use, RTL or Netlist?

- Both of  RTL and netlist are legitimate input for VC Formal
- Netlist
  - More accurate than RTL
  - There are cases only reproducible with netlist, even if equivalence checking was run
  - For these cases, if RTL is started with, another round of effort with netlist is required
  - May take less time to setup because of less number of files
- RTL
  - Has word level logic, formal tool can leverage to improve performance
  - However, control type logic most likely is of single bit
  - More humanly readable than netlist
- RTL or Netlist, roughly is a question of optimizing execution time
  - To target better average or max
- Netlist was chosen for this task

# The First Setup

- Issue Review
  - The design and a waveform from a successful simulation were studied
- Design Constraint Script
  - Constraints for clocks, resets, tie-offs were defined in the VC Formal script
- Write Assertion
  - Assert property was created for Rel_B which should be set at the middle of startup

    **Assertion_Rel_B: assert property ($fell(firmware_reset) |-> ##3 Rel_B);**
- Black Box
  - F_A and F_B were black boxed since they are not needed by startup sequences to the point Rel_B is set
- Results
  - It took too long for the tool to create formal model.  The setup needs to be refined.

# Refined Setup

- Changes
  - The design was still too large
  - Black-boxed all large modules not directly related to the issue
- Results
  - The runtime was reduced to 20 minutes
  - The assert property was falsified for first several runs
  - Each waveform was analyzed and constraints were improved
  - Eventually the assertion was proven
  - The design always sets Rel_B as expected
- Conclusion
  - The issue is not in the current reduced design, more logic should be included

# Un-black-boxing

- Changes
  - F_B actively participates in the later half of the startup sequence
  - F_B was un-block-boxed, so it can be included
  - The assertion was also changed to check the end of the startup sequence

```
Assertion_F_B: assert property ($fell(firmware_reset) |-> ##6 F_B_up);
```

- Results
  - The runtime was increased to about one hour
  - After a few rounds of tuning, the assertion was falsified and the waveform was similar to the scenario of the silicon issue
- Conclusion
  - The bug was in the arbiter, it is only triggered when F_B is started later than F_A for more than a certain number of cycles

# After Thoughts

- Well Done
  - The decision of starting with Rel_B saved time at the later refining constraint stage
    - A smaller design and simpler assertion reduced turnaround time for each iteration
  - Using more intuitive assertion rather than coverage saved debugging time
  - Internal signals were used in antecedents, avoided constructing detailed register accessing sequences at ports

    **Assertion: assert property (reg_out |=> result);**
- If I Can Do It Once More ….
  - More modules should have been black-boxed at the very first run
  - Better understanding of DUT, more experience with assertions and formal verification will generate positive impact
  - Save/Restore might save compile time, however, assertion would not be able to be written in SVA

# Comparing Effort Between Formal and Simulation

# Formal Could Have Saved Time and Effort

## Formal and simulation were run in parallel, time and effort are below

| Item | Formal | Simulation |
|---|---|---|
| Calendar Time | 2.5 days | 5 days |
| Total Effort | 3 man-days | 7 man-days |
| Verification Effort | 2.5 man-days | 5 man-days |
| Design Effort | 0.5 man-days | 2 man-days |
| Skill of Verification | Formal | Simulation |

## Observations from comparing time and effort of formal and simulation

- 50% calendar time saving if both were started at same time
- About same total effort as simulation if both were started at same time
- 57% reduction of total effort if formal were run alone
- 75% design effort saved if formal were run alone
- Formal skills are design/project independent, formal enables more flexible resources allocation

# Summary

# Summary

## Silicon Issue Debug Requires more Formal Skills

- Either RTL or netlist could be used, netlist covers more possibilities
- Both of assert and cover properties work, choose the one that fits known facts
- Using internal signals to avoid complicated sequences at design ports
- Starting with a simple property if uncertain about constraints
- Black-boxing as many modules as possible at the beginning
- Be prepared, practice before deployment

# Thank You

# Disclaimer & 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