Exploiting Sophisticated Static Analysis for Verilog

Qinlin Chen, Nairen Zhang, Jinpeng Wang, Jiacai Cui,
Tian Tan*, Xiaoxing Ma, Chang Xu, Jian Lu, and Yue Li*

PASCAL @ Institute of Computer Software
State Key Laboratory for Novel Software Technology, Nanjing University

Outline

- The Untapped Potential of Hardware Static Analysis
- **Qihe**: A General-Purpose Static Analysis Framework for Verilog
- What Hardware Static Analysis Looks Like
- Preliminary Results in Hardware Bug Detection, Security, Understanding, and Optimization

The Untapped Potential of Hardware Static Analysis

Software

Programming (C, Java)

Testing + Formal Verification

Increasing Software Diversity and Complexity

2000 —— 2026

Software Static Analysis

Thousands of analyses for bug detection, security, program understanding…

Basic Syntactic Checks

Naming, Style, Pattern, ...

Sophisticated Semantic Analysis

Logic, Property, Intention, ...

Hardware

Programming (Verilog)

Simulation + Formal Verification

Increasing Hardware Diversity and Complexity

2026 ——

Hardware Static Analysis?

Underexplored

The Untapped Potential of Hardware Static Analysis

Software

Programming (C, Java)

Testing + Formal Verification

Increasing Software Diversity and Complexity

2000 —— 2026

Software Static Analysis

Thousands of analyses for bug detection, security, program understanding…

Hardware

Programming (Verilog)

Simulation + Formal Verification

Increasing Hardware Diversity and Complexity

2026 ——

Hardware Static Analysis?

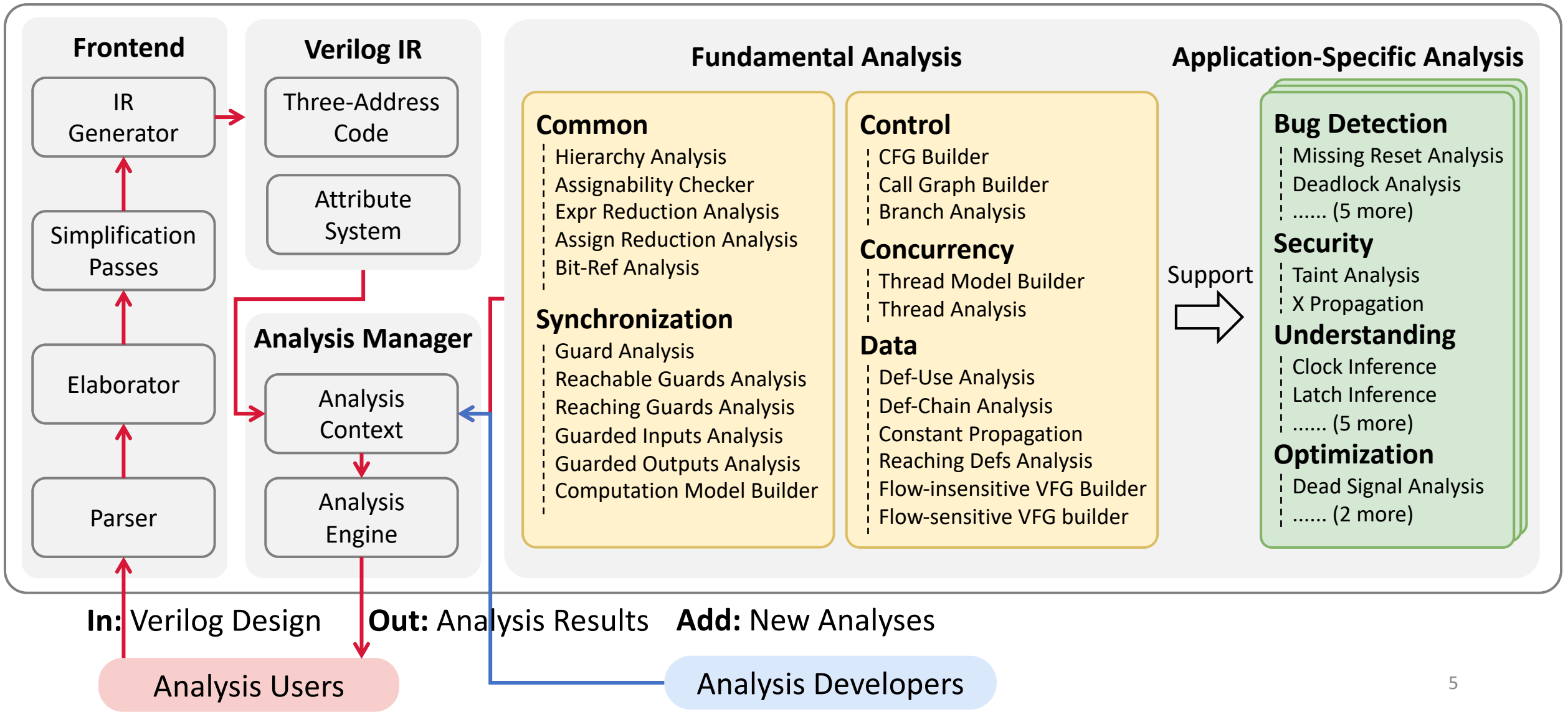
Underexplored

Exploit

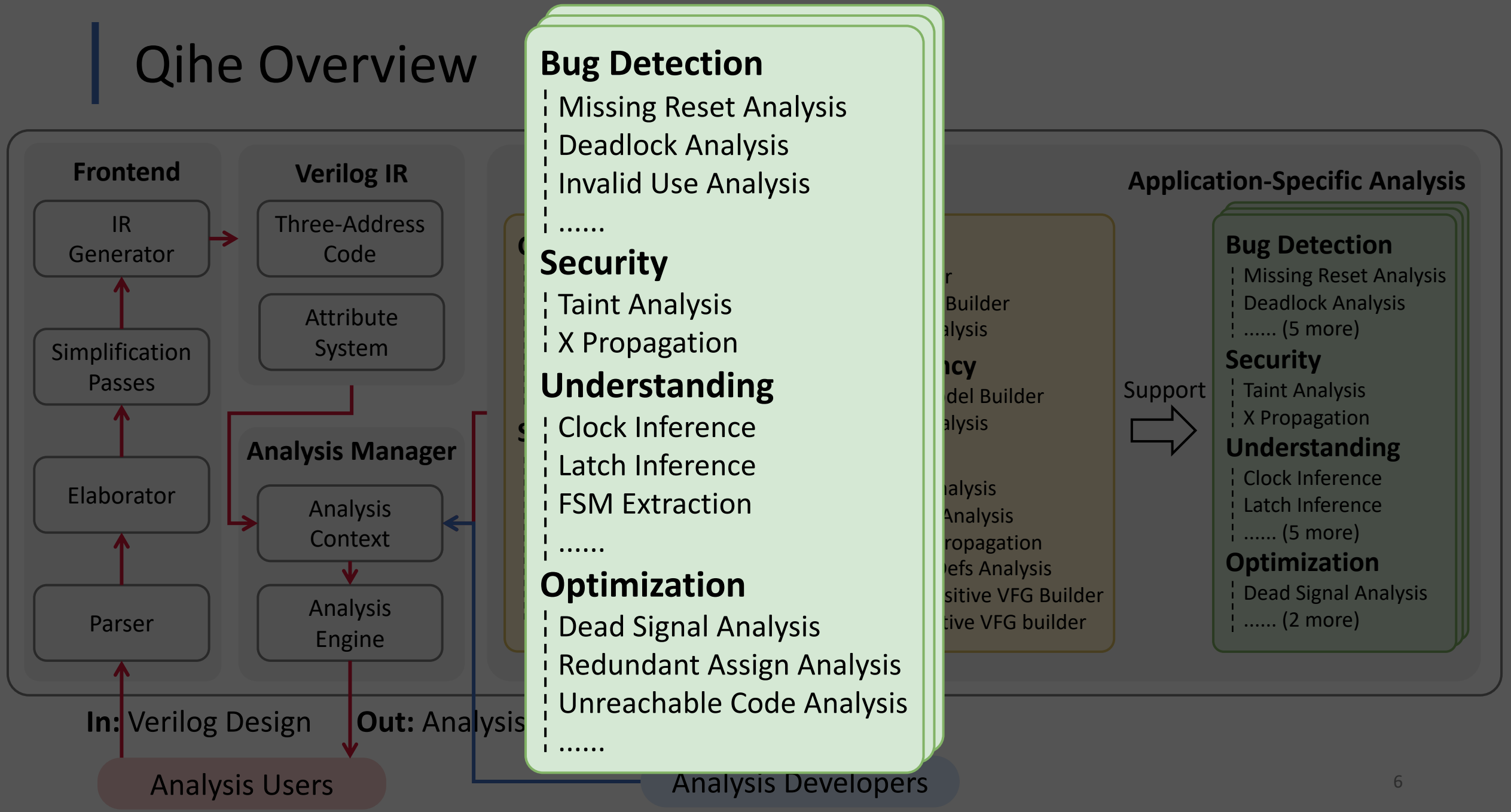
Qihe: The First General-Purpose Static Analysis Framework for Verilog

100K+ LoC, Open-Source

Qihe Overview



Qihe Overview



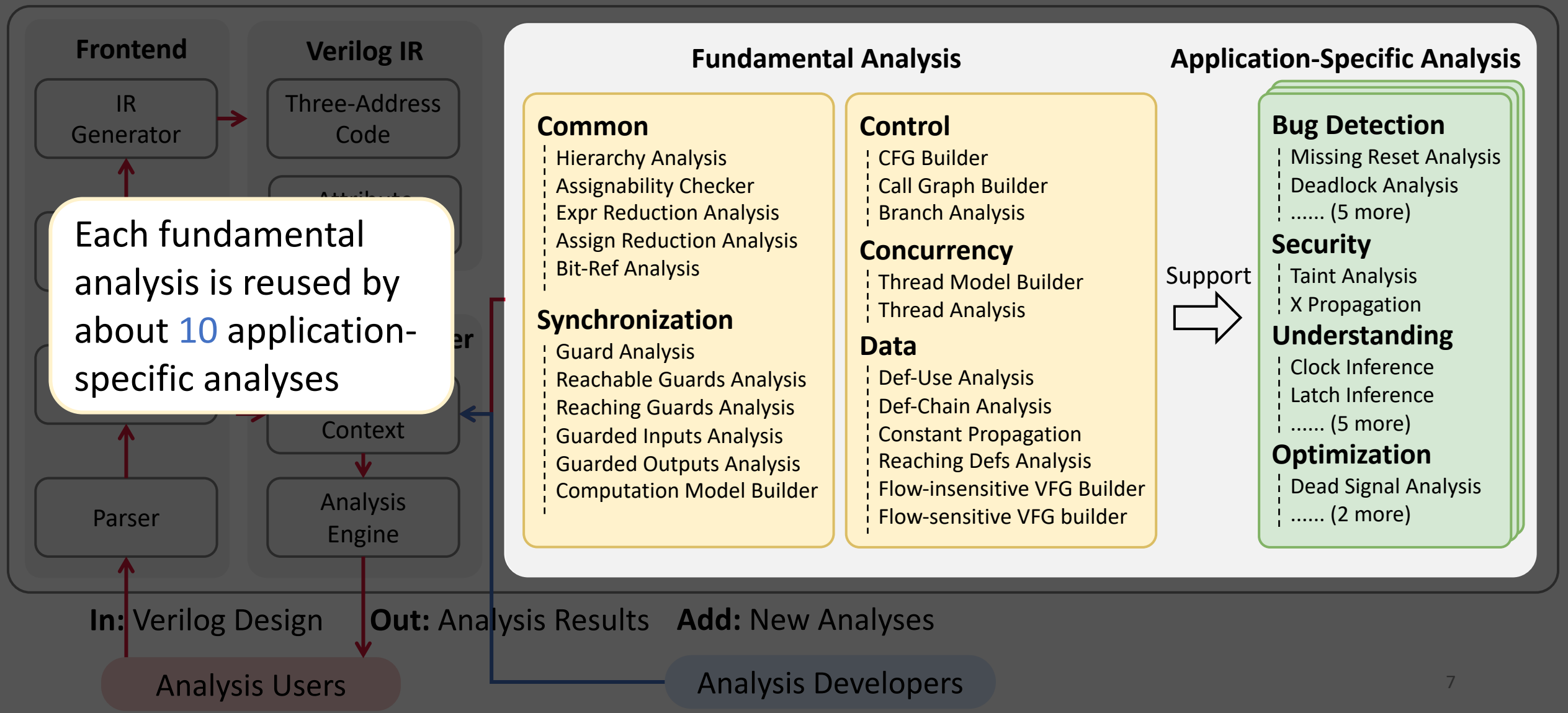
Bug Detection
- Missing Reset Analysis
- Deadlock Analysis
- Invalid Use Analysis
- .....

Security
- Taint Analysis
- X Propagation

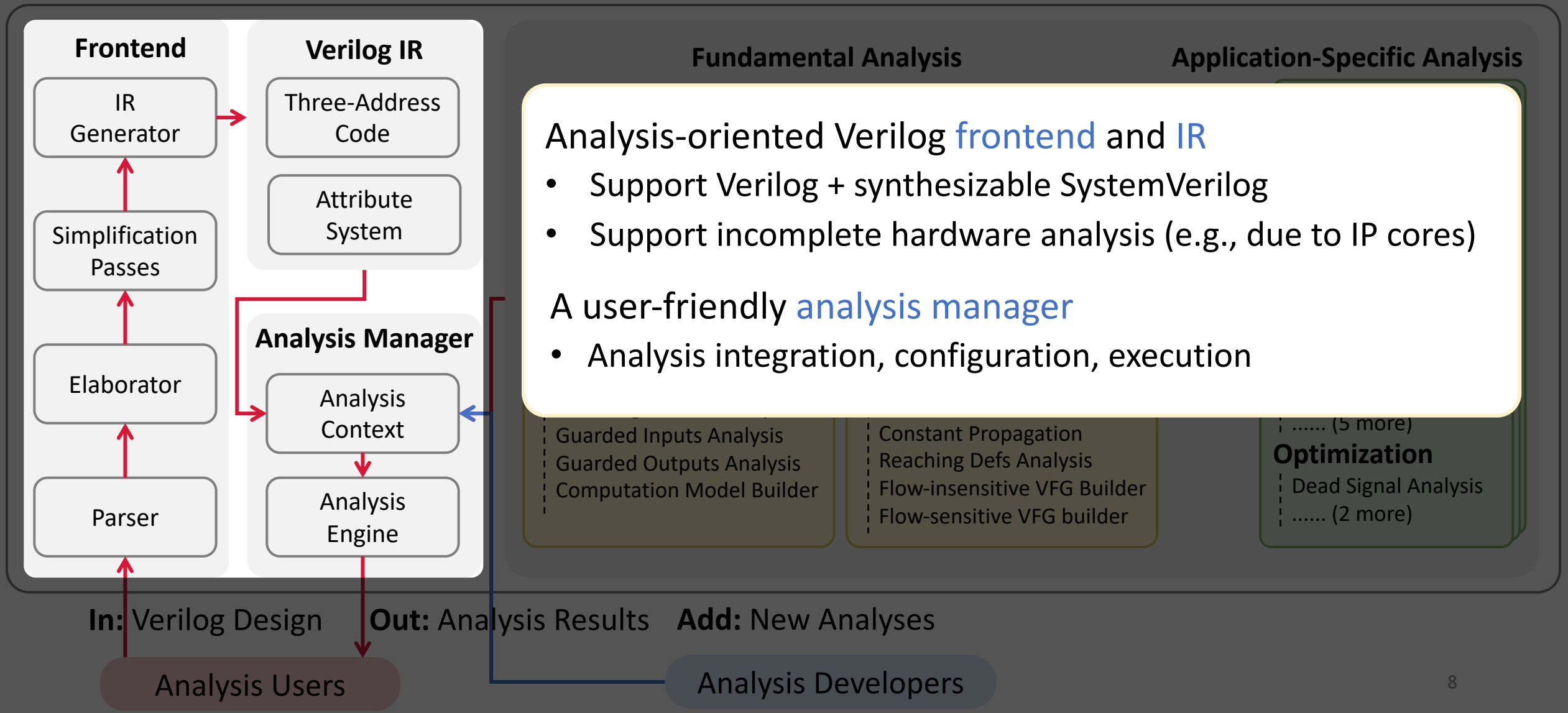
Understanding
- Clock Inference
- Latch Inference
- FSM Extraction
- .....

Optimization
- Dead Signal Analysis
- Redundant Assign Analysis
- Unreachable Code Analysis
- .....

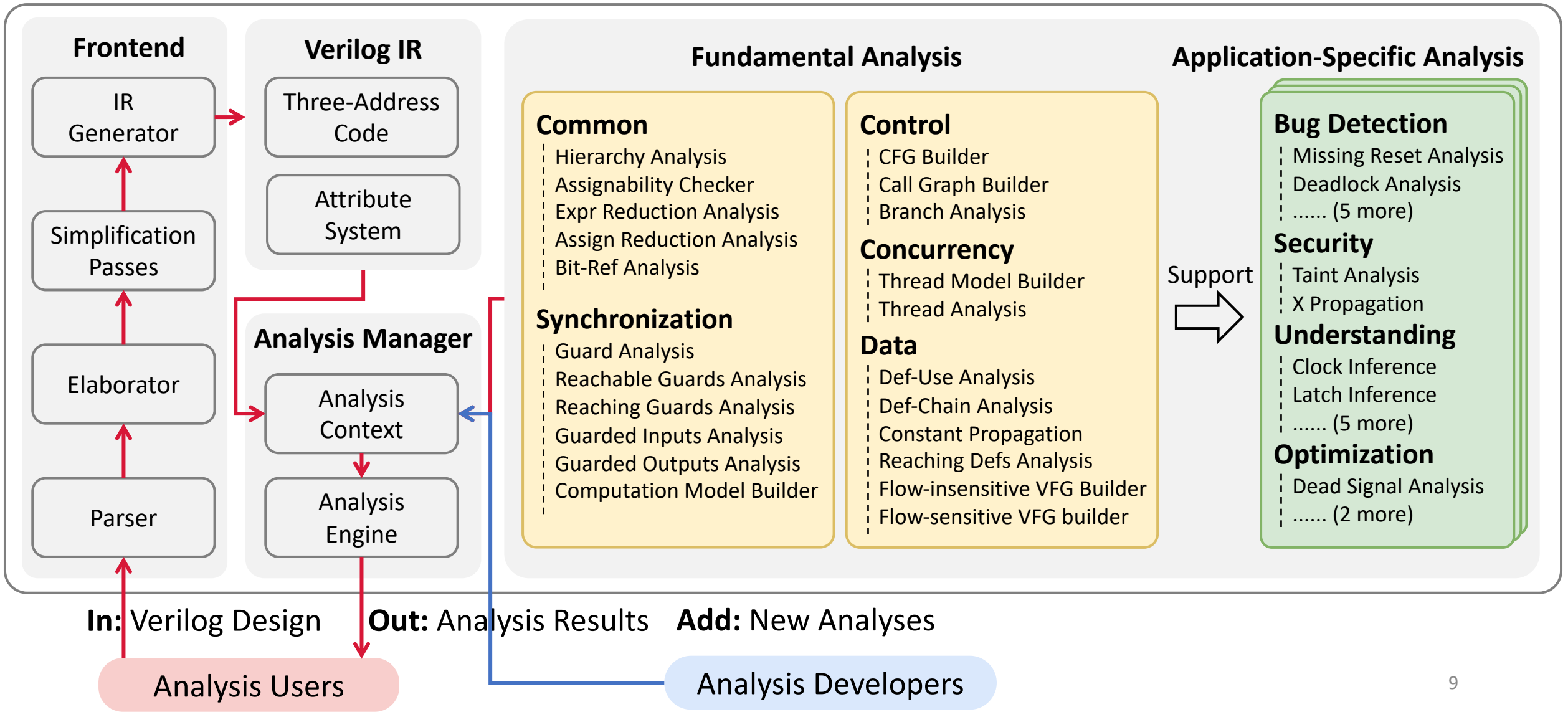
Qihe Overview



Qihe Overview



Qihe Overview



Questions on Developing Hardware Static Analysis

Q1: What kinds of hardware problems can static analysis handle, and how do they differ from software analysis problems?

Q2: How to design sophisticated hardware static analyses that leverage hardware-specific insights to address these problems?

Q3: What challenges arise in building sophisticated hardware static analyses and how can Qihe help address it?

Next, Motivating Example: Missing-Reset Analysis

- Software Analogue: Use-Before-Initialization (UBI)
- Both lead to serious consequence due to undefined value

Background: Registers and Verilog

Registers: store values representing the circuit state and updates on clock edges (i.e., when the clock signal changes).

```
1 reg acc;
2 input wire in, clock;
3 always @(posedge clock)
4   acc <= acc + in;
```

Background: Registers and Verilog

Reset: Since registers power up in an undefined state, hardware developers design a reset signal which users can pull up to activate reset logic forcing registers into a predefined known state.

```
1 reg acc;
2 input wire in, clock;
3 always @(posedge clock)
4   acc <= acc + in;
```

Background: Registers and Verilog

Reset: Since registers power up in an undefined state, hardware developers design a reset signal which users can pull up to activate reset logic forcing registers into a predefined known state.

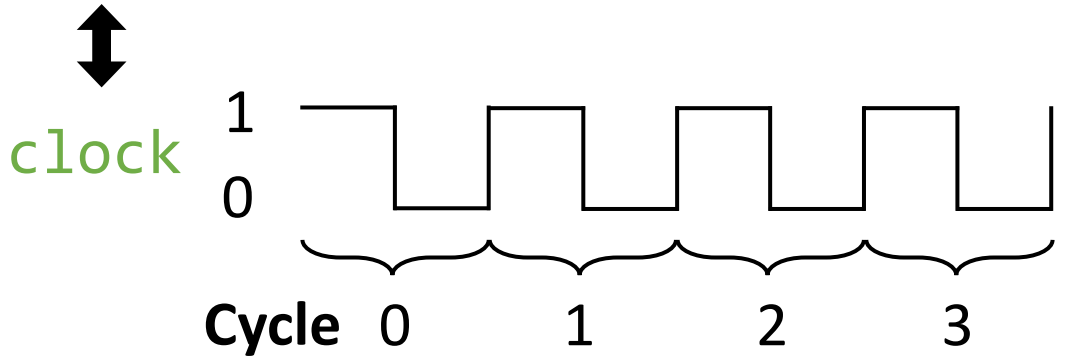
```
1 reg acc;
2 input wire in, clock;
3 input wire reset;
4 always @(posedge clock)
5   if (reset)
6     acc <= 0;
7   else
8     acc <= acc + in;
```

Background: Runtime

```
1  reg acc;
2  input wire in, clock;
3  input wire reset;
4  always @(posedge clock)
5    if (reset)
6      acc <= 0;
7    else
8      acc <= acc + in;
```

Cycle	Input		State
	reset	in	acc
0	1	1	undef
1	0	2	0
2	0	3	2
3	…	…	5

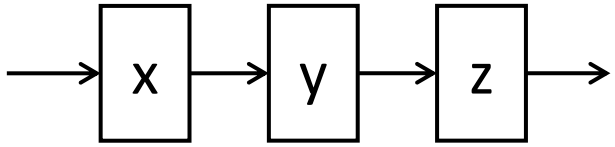
power-up



Missing-Reset Problem

Definition: A register has the missing-reset problem if its undefined value is infinitely-often used after reset

Finitely-often is OK: propagating defined value needs time (e.g. pipelined circuit)




```
1 always @(posedge clock)
2   if (reset)
3     x <= 1;
4   else begin
5     y <= x;
6     z <= y;
7 end
```

Cycle	Input	State		
	reset	x	y	z
0	1	undef	undef	undef
1	0	1	undef	undef
2	0	1	1	undef
3	…	1	1	1

Missing-Reset Problem

Definition: A register has the missing-reset problem if its undefined value is infinitely-often used after reset

```
1  always @(posedge clock)
2    if (reset)
3      x <= 1;
4    else begin
5      x <= y;
6      y <= x ? in : 0;
7    end
```

Cycle	Input		State	
	reset	in	x	y
0	1	0	undef	undef
1	0	1	1	undef
2	0	2	undef	1
3	0	1	1	undef
4	…	…	undef	1

Missing-Reset vs. Use-Before-Initialization

Seem similar, but:

UBI: a variable whose undefined value is once used in any possible software execution

Missing-Reset: a register whose undefined value is infinitely often used after reset

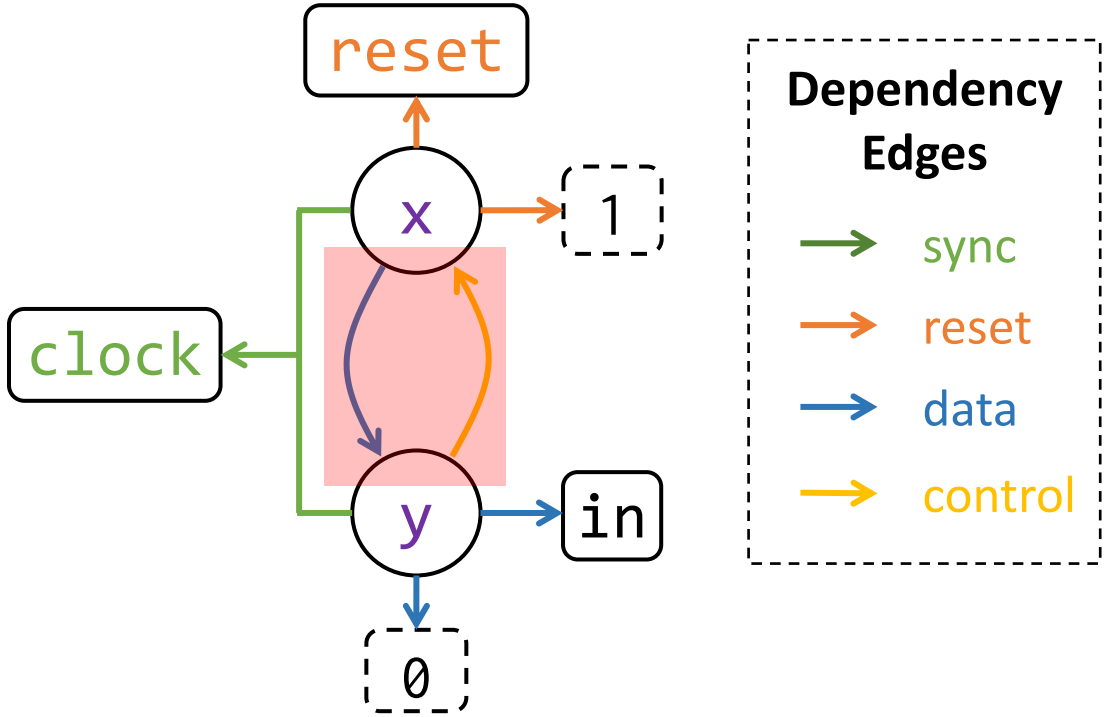
Naturally, addressing the missing-reset problem needs some hardware-specific insights, rather than simply copying approaches designed for UBI

Insight of Our Missing-Reset Analysis

Our Insight: only non-reset registers involved in circular dependencies can have missing-reset problems.

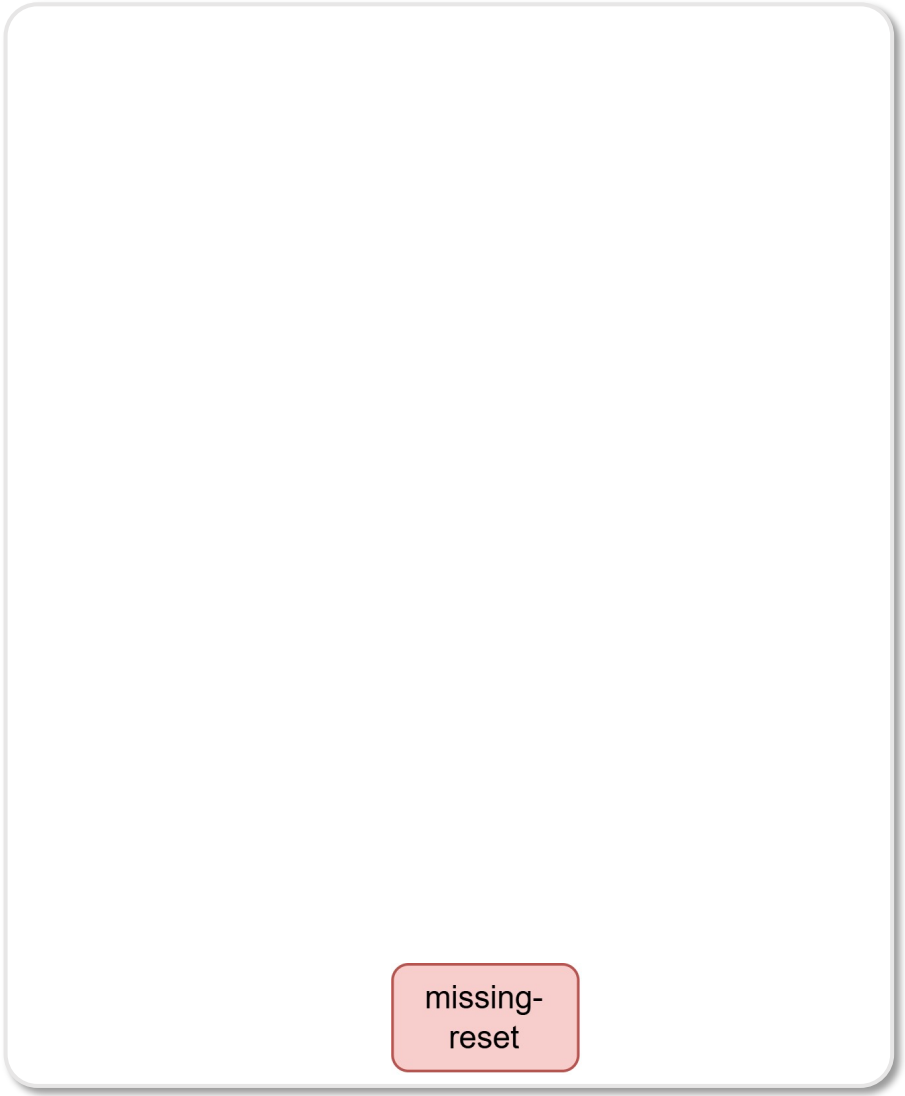
```
1  always @(posedge clock)
2    if (reset)
3      x <= 1;
4    else begin
5      x <= y;
6      y <= x ? in : 0;
7    end
```

Graph



See our paper for details on the soundness

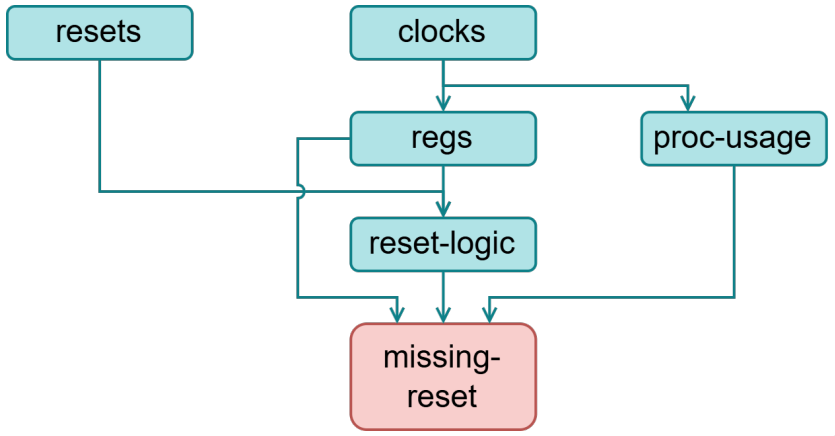
Challenges in Building Missing-Reset Analysis



Challenges in Building Missing-Reset Analysis

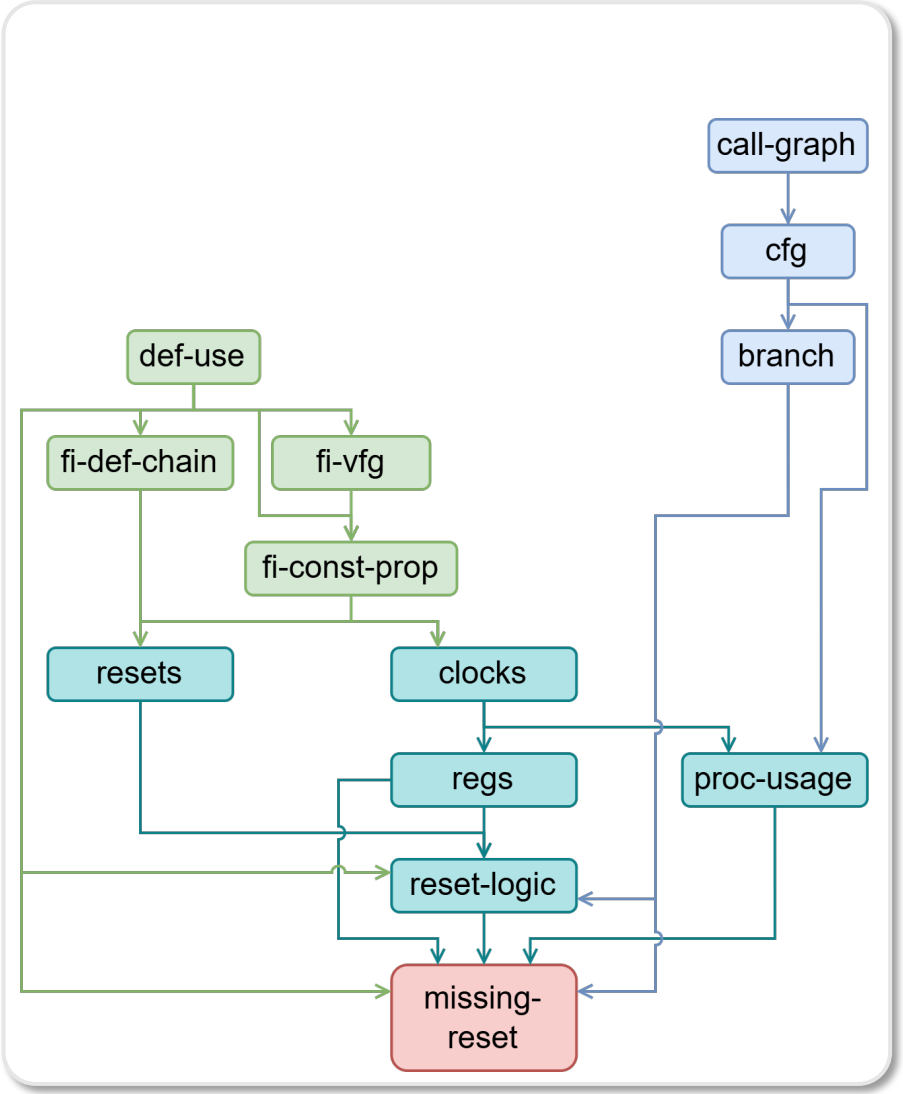
Need to understand the synthesis process, i.e., how Verilog variables are mapped to physical registers, clocks, and resets

- invoking a synthesizer for such information requires > 1h for a design of **1.8M LoC**
- Qihe provides such analyses finishing in 3 seconds



Challenges in Building Missing-Reset Analysis

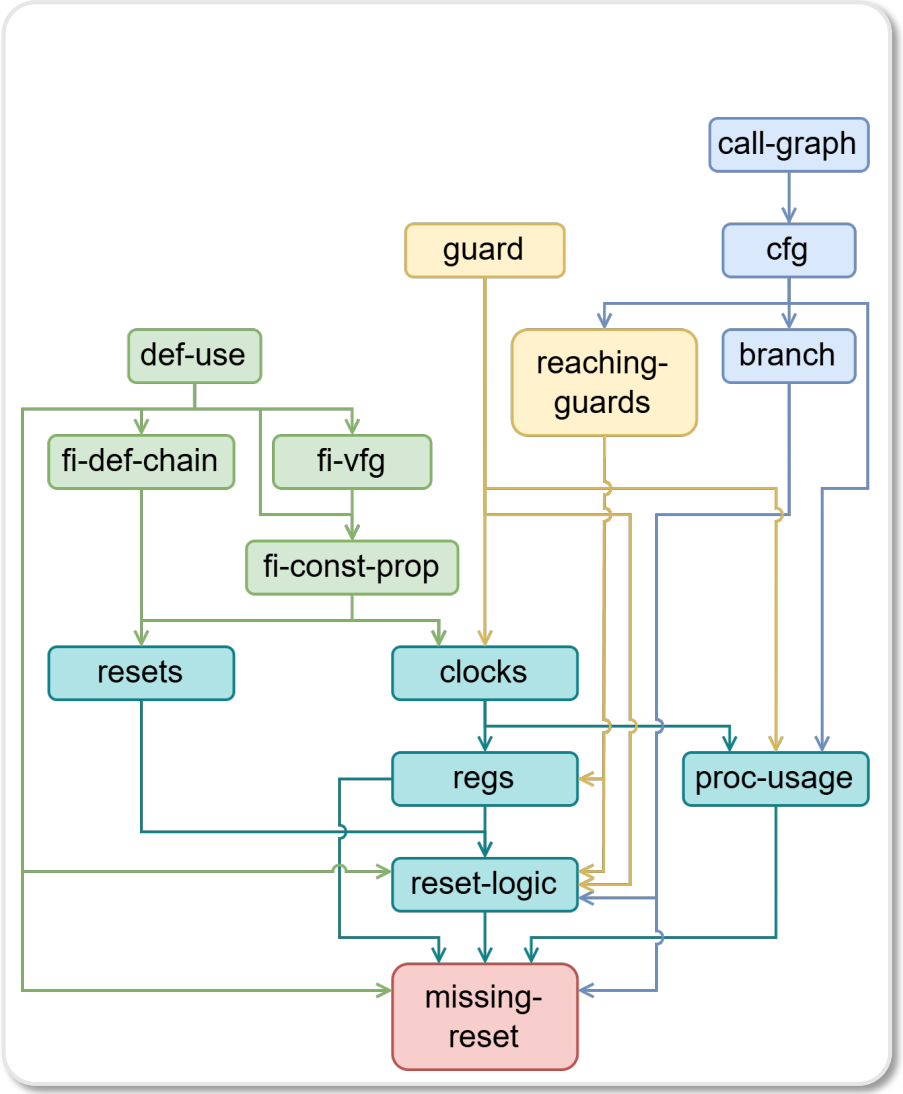
Need hardware data-flow and control-flow analyses for constructing the graph structures



Challenges in Building Missing-Reset Analysis

Need hardware data-flow and control-flow analyses for constructing the graph structures

Need analyses for capturing Verilog's synchronization characteristics, which are essential for reasoning about behavioral semantics in hardware understanding



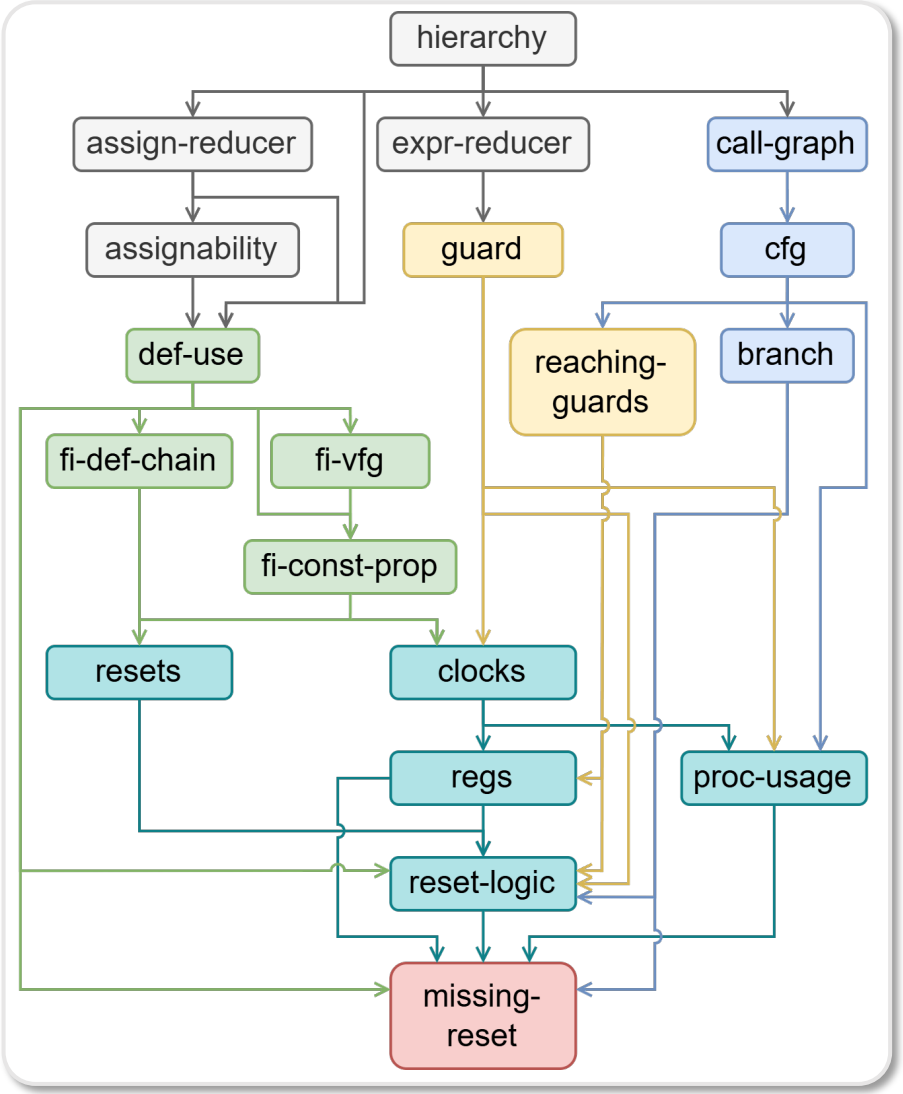
Challenges in Building Missing-Reset Analysis

Need hardware data-flow and control-flow analyses for constructing the graph structures

Need analyses for capturing hardware's synchronization characteristics, which are essential for reasoning about behavioral semantics in hardware understanding

Need module hierarchy analysis for inter-module reasoning and more ...

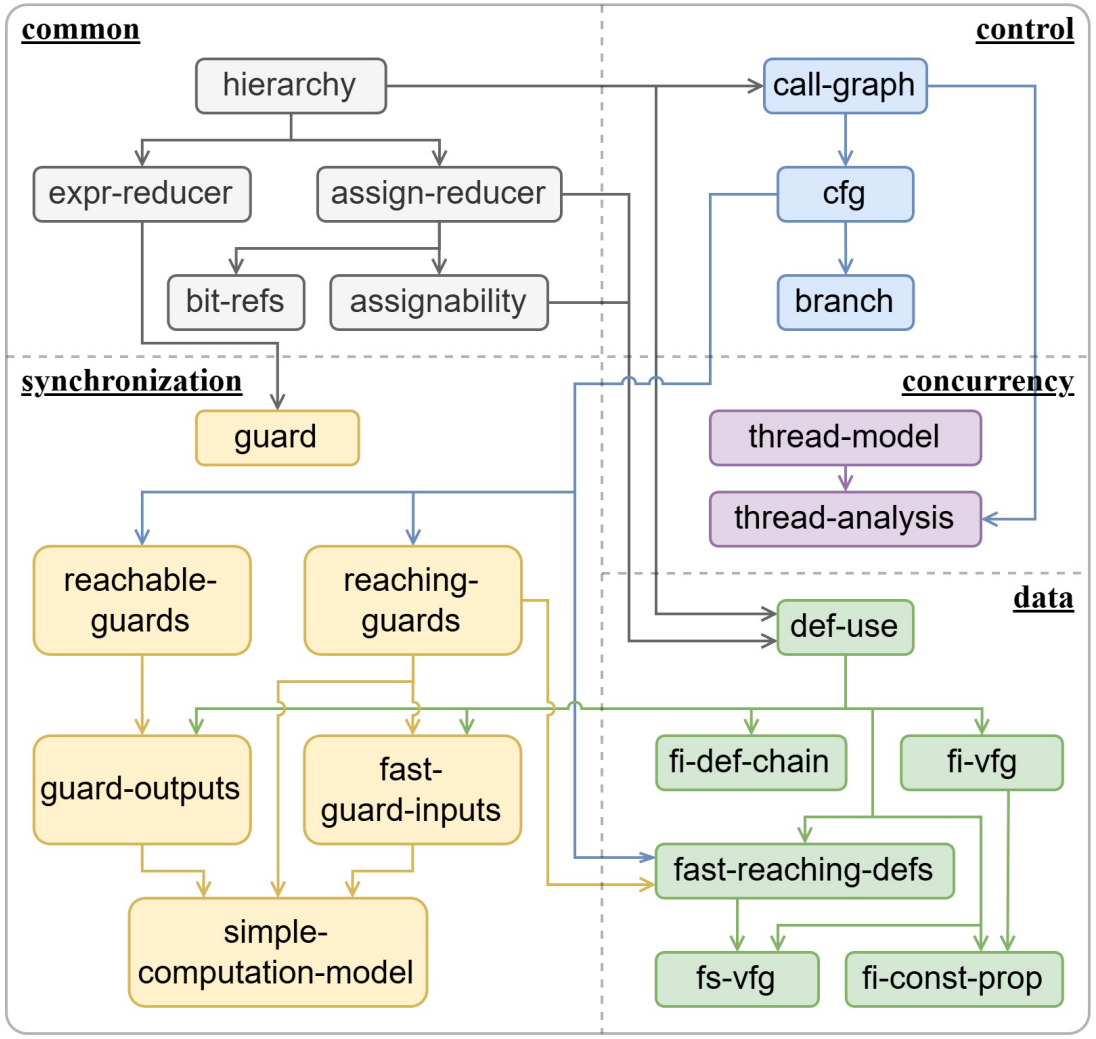
Qihe comes with all of these analyses and more, ready to be reused across a wide range of clients.



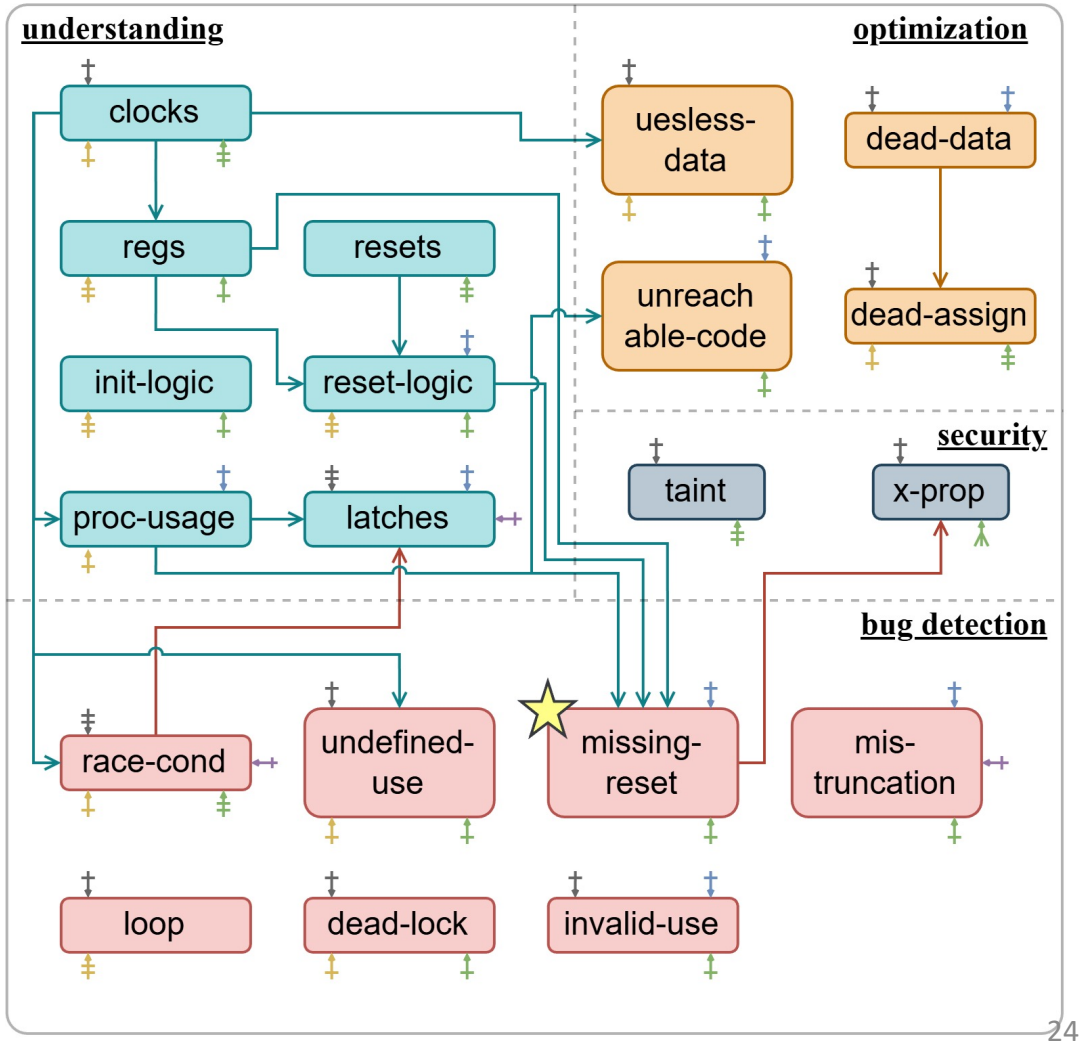
Qihe's Analysis Suite

Each fundamental analysis is reused by about 10 application-specific analyses

Fundamental Analyses



Application-Specific Analyses



Preliminary Results

1. Hardware Bug Detection (11 analyses)

- Unlike software, hardware tends to contain fewer but harder-to-detect bugs
- Detected 18 bugs across 8 categories from 11 popular open-source hardware projects
  - Projects: avg. 1.5K+ GitHub stars — CPU, AXI Protocol, SHA512 Encryption, ...
  - Bugs: Missing-Reset Regs, Unreachable States, Deadlocks, Invalid Use, ...
- 9/18 bugs were previously unknown, all confirmed by developers
- Beyond the capabilities of existing Verilog static analyses (i.e., linters)

2. Hardware Security Analysis (2 analyses)

- 15 information leaks found in 19 hardware designs from TrustHub
  - e.g. encryption keys to leakage points, such as capacitance, antenna, and shift reg
- A Hardware Trojan exploiting Verilog's X-value

Preliminary Results

3. Hardware Program Understanding (7 analyses)

- Representative understanding tasks: inferring clock signals, reset signals, and registers
- On 13 real hardware programs (averaging 146K lines)
  - 97.1% precision and 96.9% recall on average;
  - 62% (=24/39) cases can achieve both 100% precision and recall

4. Hardware Code Optimization (4 analyses)

- Optimize 7000+ useless signal on the same 13 real programs as above

Preliminary Results

3. Hardware Program Understanding (7 analyses)

- Representative understanding tasks: inferring clock signals, reset signals, and registers
- On
  -
  -

4. Hardwa

- Optimize 7000+ useless signal on the same 13 real programs as above

Low Development Cost

- ~300 LoC for each of the client analyses
- vs. ~5000 LoC from scratch

Lightweight Runtime Cost

- 40+ analyses, on a 1.8M LoC code base, 6 minutes

Preliminary Impact

Since its release 4 months ago, Qihe has received registrations for use from nearly 50 institutions worldwide

- Peking University
- Tsinghua University
- Georgia Institute of Technology
- University of California, Los Angeles
- Fudan University
- Zhejiang University
- Johns Hopkins University
- The Ohio State University
- University of Science and Technology of China
- Chinese Academy of Science (all its related institutes)
- The University of Hong Kong
- Max Planck Institute for Security and Privacy
- Taiwan Cheng Kung University
- Ivannikov Institute for System Programming of Russian Academy of Sciences

… …

Where to Go?

qihe.pascal-lab.net

Homepage (qihe.pascal-lab.net)
- Obtain Qihe's source code (100K+ LoC)

Documentation (qihe-docs.pascal-lab.net)
- Learn how to use Qihe
- Try developing new analyses in Qihe!

Javadoc (qihe-docs.pascal-lab.net/javadoc/)
- Detailed API Reference

Questions or Discussions?
Feel free to contact us!

qinlinchen@smail.nju.edu.cn
jiacaicui@smail.nju.edu.cn
tiantan@nju.edu.cn
yueli@nju.edu.cn

Thank You!
Q&A