Qihe in Action:

A Practical Guide to Hardware Static Analysis

Qinlin Chen

Ph.D. Advisors: Yue Li & Tian Tan

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

Outline

Part I: Context

- **Motivation:** Hardware Static Analysis and Qihe Framework
- **Overview:** Qihe's Use Cases and Underlying Workflow

Part II: Getting Started

- **Verilog Overview:** A Quick Example
- **Warm-Up Analysis:** Start with Printing the Module Hierarchy

Part III: In Action

- **The Problem:** Missing-Reset Bugs in Hardware
- **The Algorithm:** Detecting Missing-Reset Bugs
- **Step-by-Step Implementation:** Missing-Reset Analysis

Outline

Part I: Context

- **Motivation:** Hardware Static Analysis and Qihe Framework
- **Overview:** Qihe's Use Cases and Underlying Workflow

Part II: Getting Started

- **Verilog Overview:** A Quick Example
- **Warm-Up Analysis:** Start with Printing the Module Hierarchy

Part III: In Action

- **The Problem:** Missing-Reset Bugs in Hardware
- **The Algorithm:** Detecting Missing-Reset Bugs
- **Step-by-Step Implementation:** Missing-Reset Analysis

Motivation: Hardware Static Analysis and Qihe

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…

Motivation: Hardware Static Analysis and Qihe

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

Motivation: Hardware Static Analysis and Qihe

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

Support

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

Overview: Qihe's Use Cases

Analysis Users

Analysis Developers

Overview: Qihe's Use Cases

Analysis Users

```
> qihe compile buggy.v -o buggy.qh
> qihe run missing-reset -i buggy.qh
ERROR axis_frame_fifo.drop_frame needs resetting
ERROR axis_frame_fifo.wr_ptr_cur needs resetting
```

Analysis Developers

Overview: Qihe's Use Cases

Analysis Users

```
> qihe compile buggy.v -o buggy.qh
> qihe run missing-reset -i buggy.qh
ERROR axis_frame_fifo.drop_frame needs resetting
ERROR axis_frame_fifo.wr_ptr_cur needs resetting
```

Analysis Developers

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  @Analysis.Run
  public void run(Design design) {
      // Analysis Logic
  }
}
```

Overview: Qihe's Workflow

```
> qihe compile buggy.v -o buggy.qh
```

Analysis Users

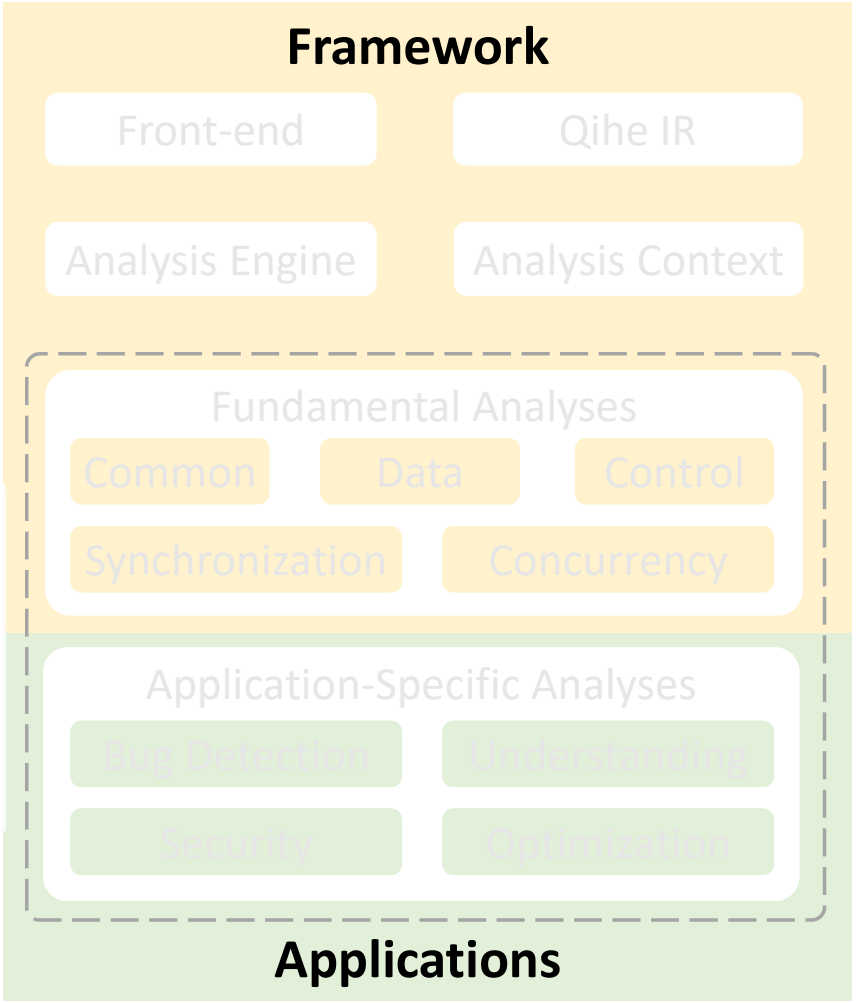
Analysis Developers

Overview: Qihe's Workflow

```
> qihe compile buggy.v -o buggy.qh
```

Analysis Users

Analysis Developers

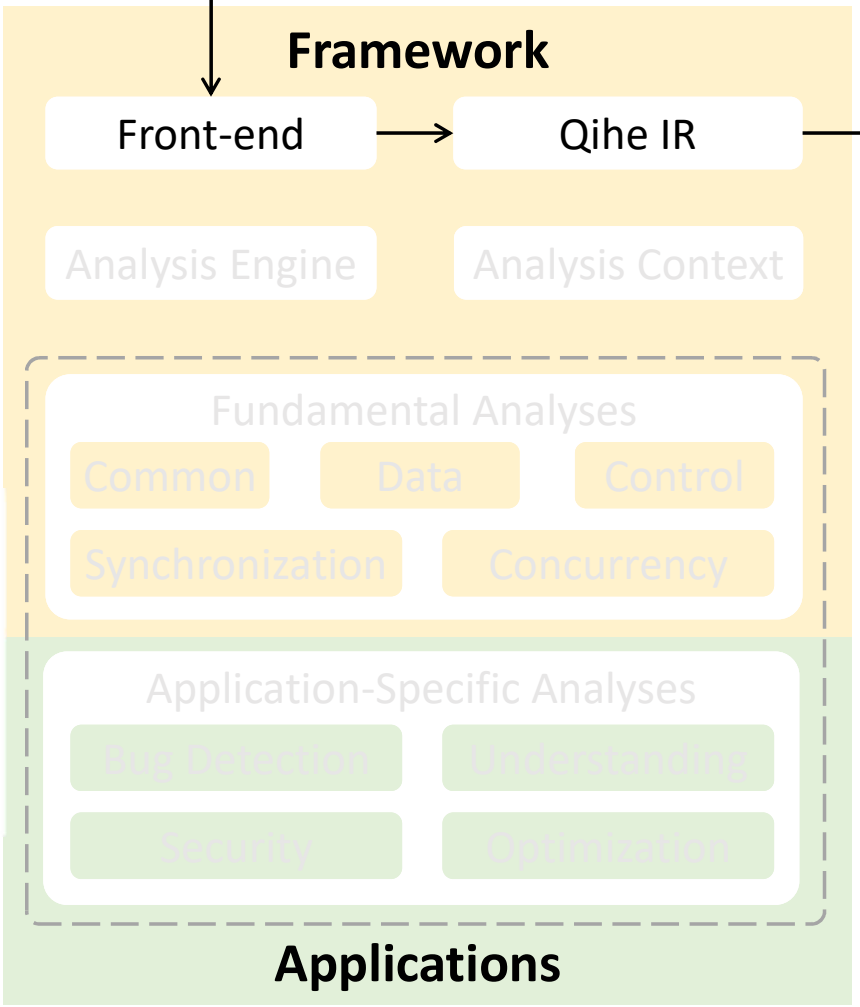


Overview: Qihe's Workflow

```
> qihe compile buggy.v -o buggy.qh
```

Analysis Users

Analysis Developers

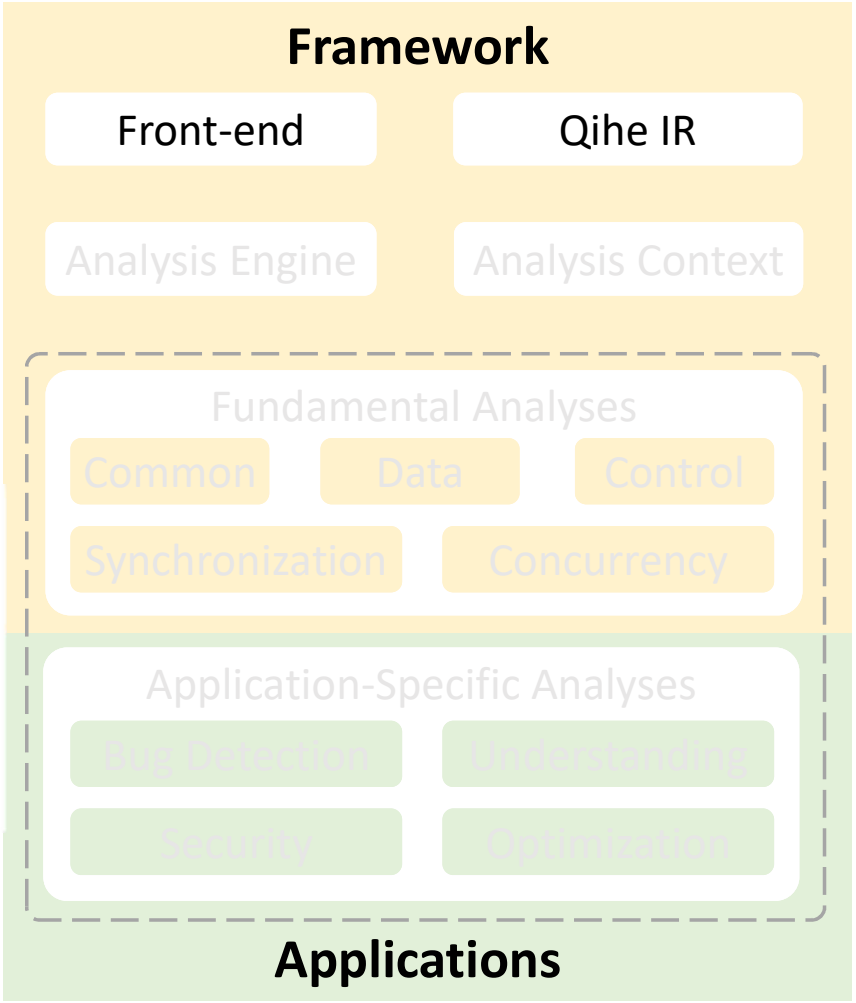


Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

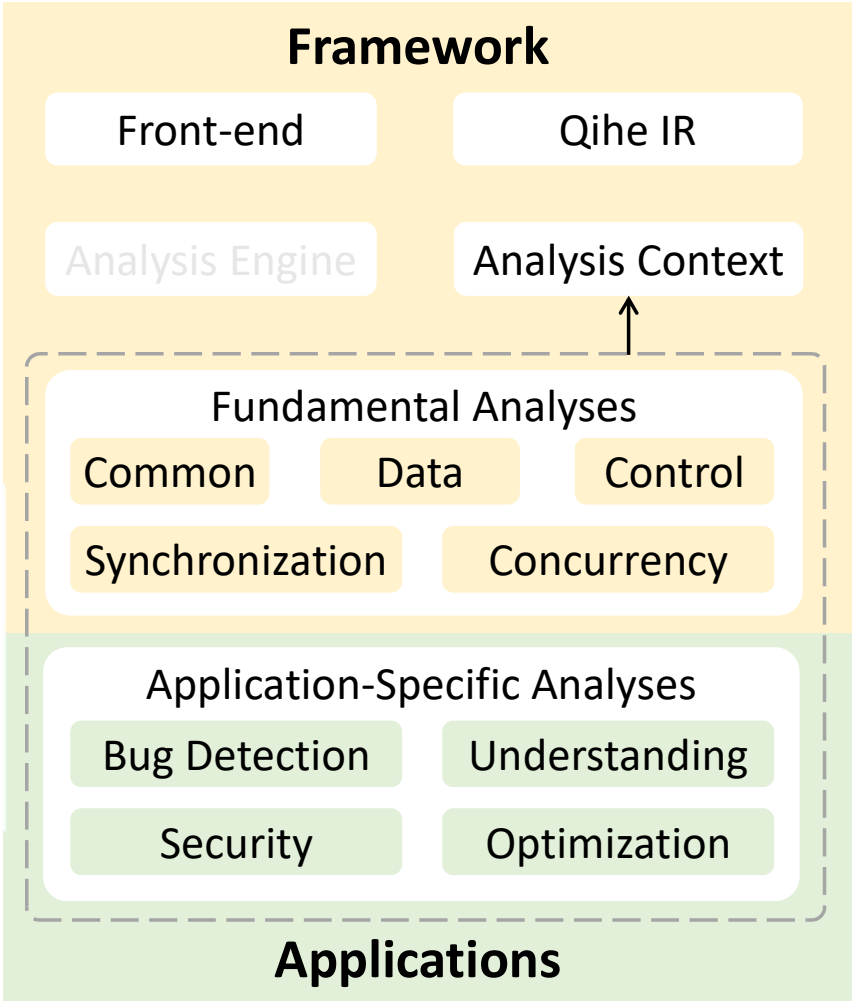


Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

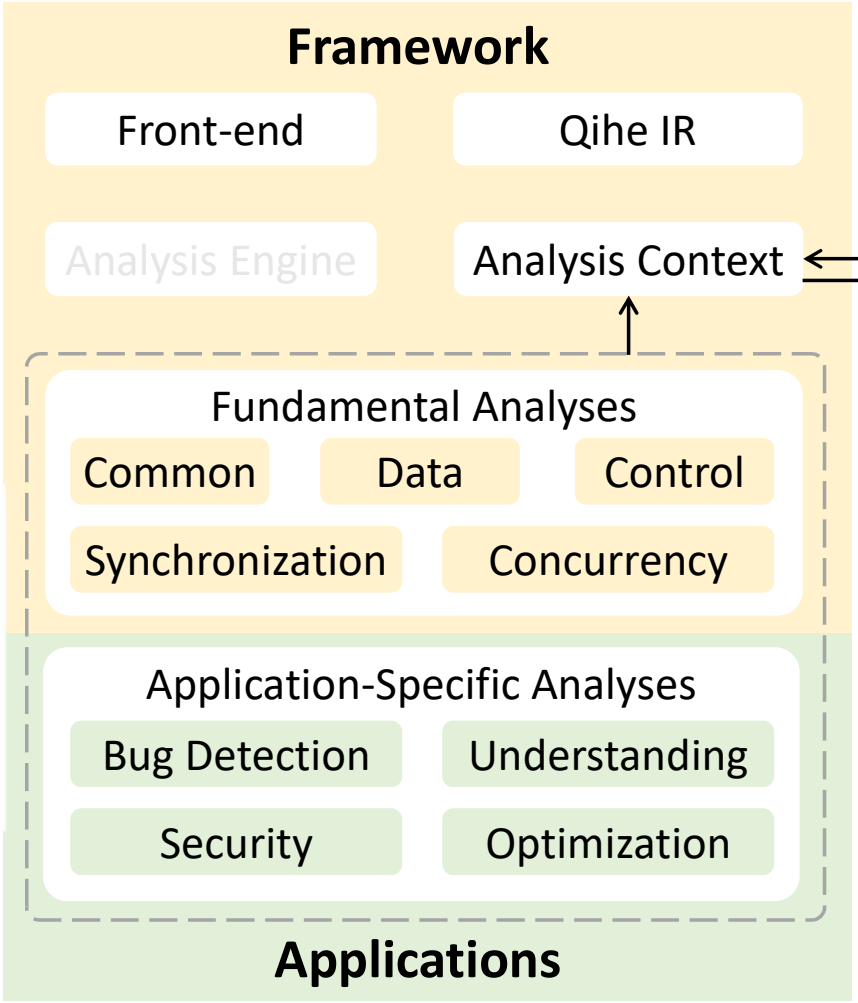


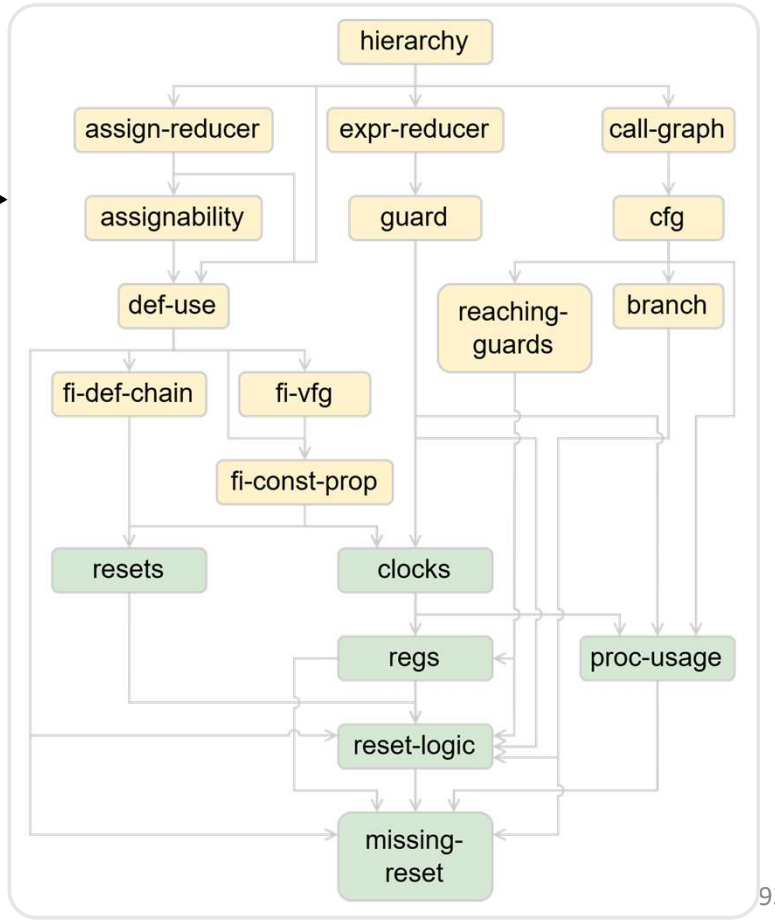
Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers





Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

Framework

Front-end

Qihe IR

Analysis Engine

Analysis Context

Fundamental Analyses

Common

Data

Control

Synchronization

Concurrency

Application-Specific Analyses

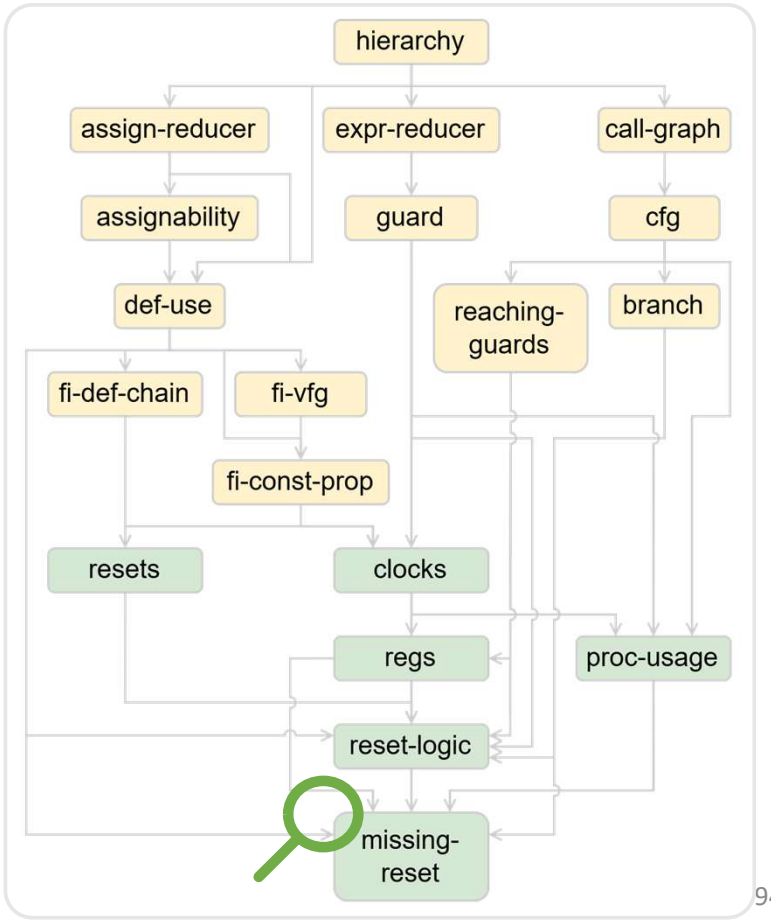
Bug Detection

Understanding

Security

Optimization

Applications



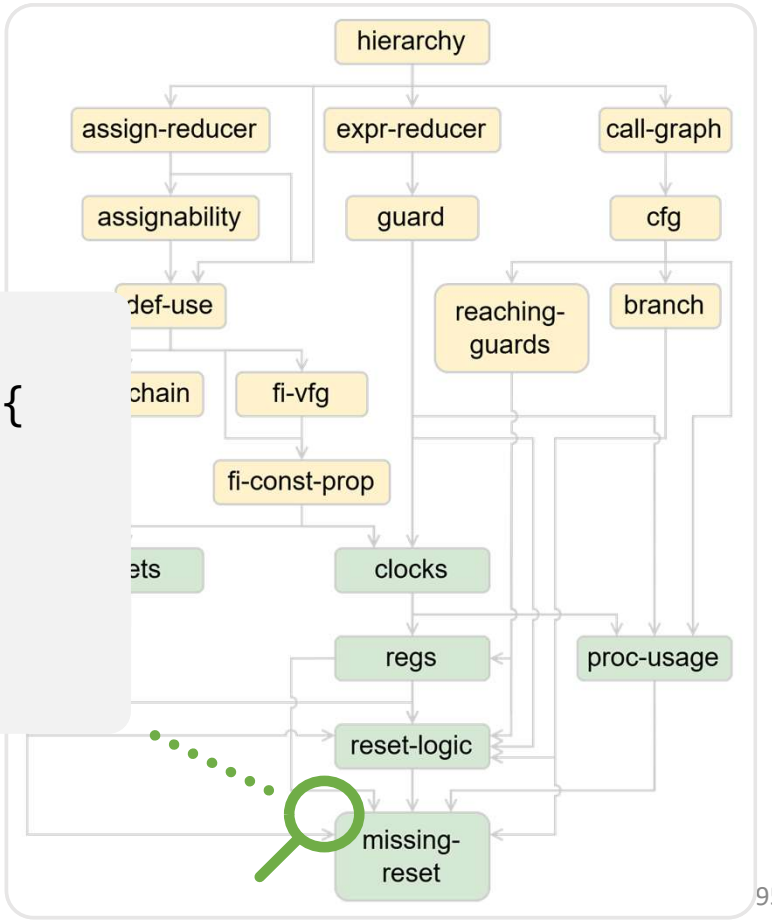
Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
  }
}
```



Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

Framework

Front-end | Qihe IR

Analysis Engine | Analysis Context

Fundamental Analyses

Common | Data | Control

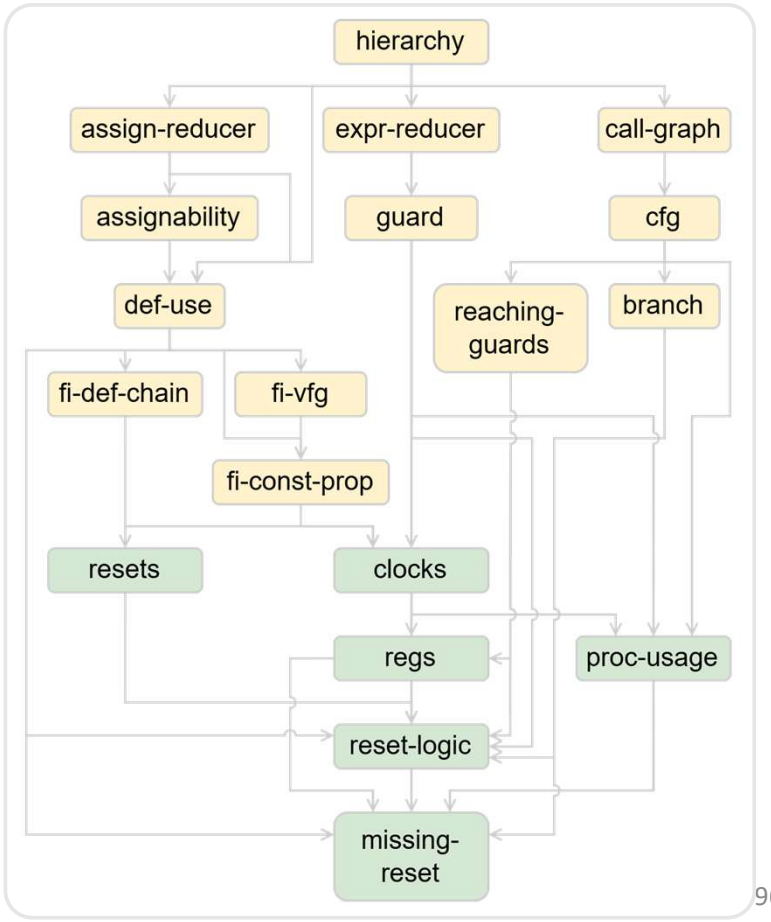
Synchronization | Concurrency

Application-Specific Analyses

Bug Detection | Understanding

Security | Optimization

Applications



Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

Framework

Front-end

Qihe IR

Analysis Engine

Analysis Context

Fundamental Analyses

Common | Data | Control

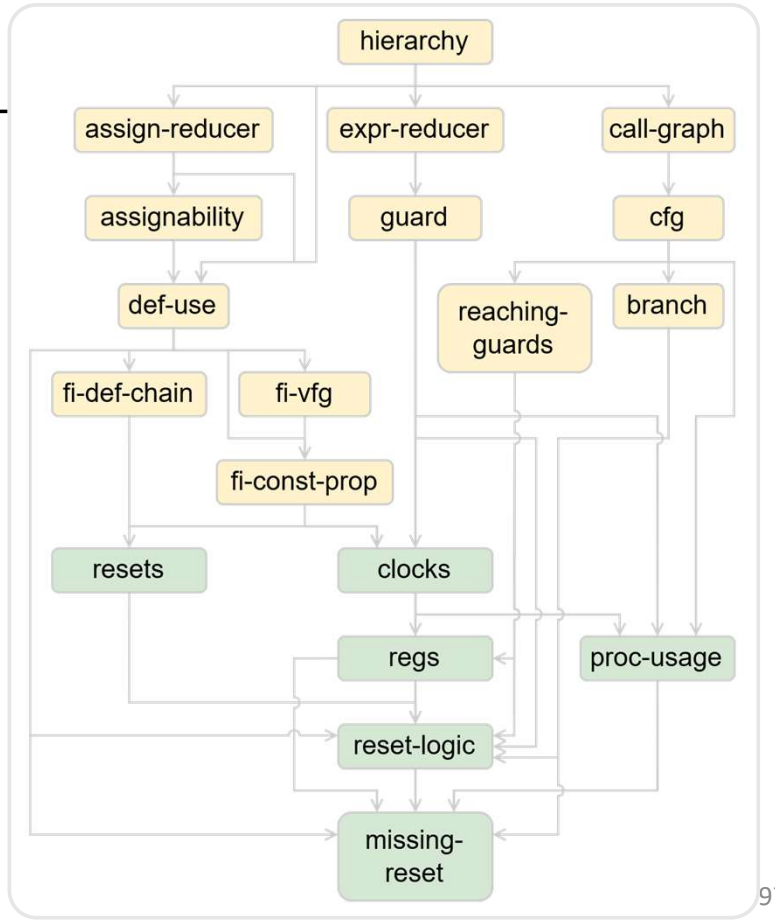
Synchronization | Concurrency

Application-Specific Analyses

Bug Detection | Understanding

Security | Optimization

Applications



Overview: Qihe's Workflow

```
> qihe run missing-reset -i buggy.qh
```

Analysis Users

Analysis Developers

Framework

Front-end

Qihe IR

Analysis Engine

Analysis Context

Fundamental Analyses

Common | Data | Control

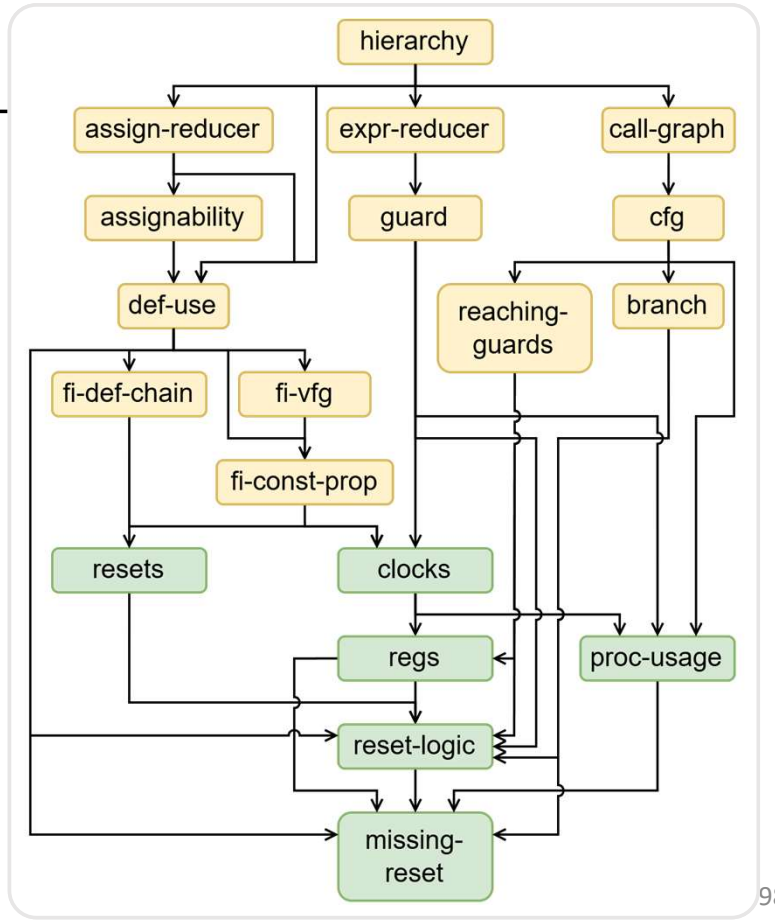
Synchronization | Concurrency

Application-Specific Analyses

Bug Detection | Understanding

Security | Optimization

Applications



Overview: Qihe's Workflow

Analysis Users

Analysis Developers

Framework

Front-end | Qihe IR

Analysis Engine | Analysis Context

Fundamental Analyses

Common | Data | Control

Synchronization | Concurrency

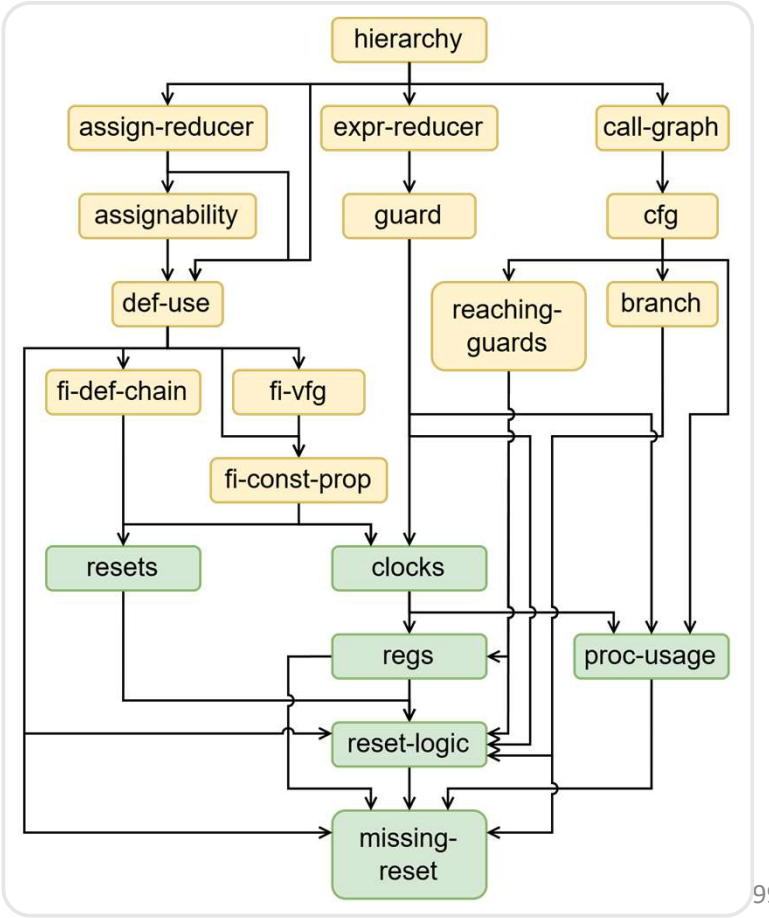
Application-Specific Analyses

Bug Detection | Understanding

Security | Optimization

Applications

```
> qihe run missing-reset -i buggy.qh
ERROR axis_frame_fifo.drop_frame...
ERROR axis_frame_fifo.wr_ptr_cur...
```



Today's Goal

Build a static analysis within the Qihe framework to detect a category of real-world hardware bugs

Analysis Developers

Java

```java
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  @Analysis.Run
  public void run(Design design) {
      // Analysis Logic
  }
}
```

Outline

Part I: Context

- **Motivation:** Hardware Static Analysis and Qihe Framework
- **Overview:** Qihe's Use Cases and Underlying Workflow

Part II: Getting Started

- **Verilog Overview:** A Quick Example
- **Warm-Up Analysis:** Start with Printing Module Hierarchy

Part III: In Action

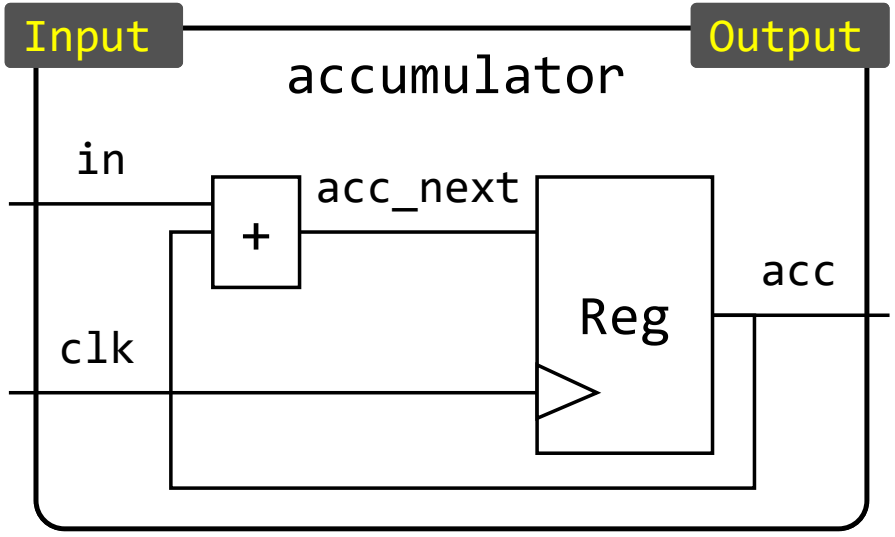
- **The Problem:** Missing-Reset Bugs in Hardware
- **The Algorithm:** Detecting Missing-Reset Bugs
- **Step-by-Step Implementation:** Missing-Reset Analysis

Verilog Overview: A Quick Example

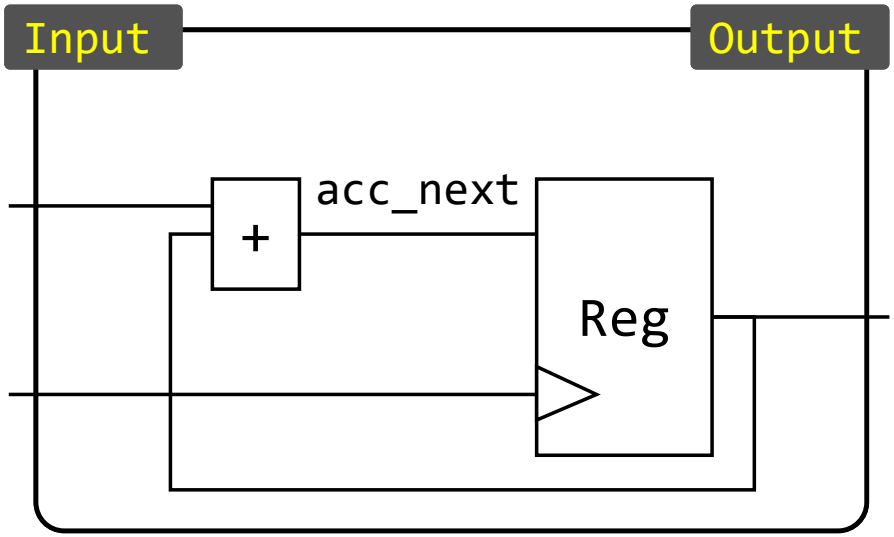
Verilog: a hardware description language

Verilog Overview: A Quick Example

Verilog: a hardware description language

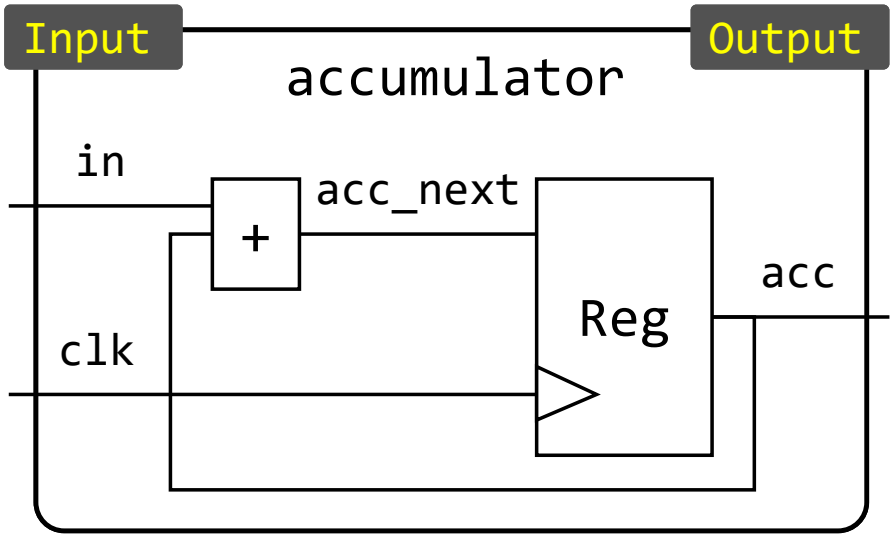


Verilog Overview: Describing A Circuit



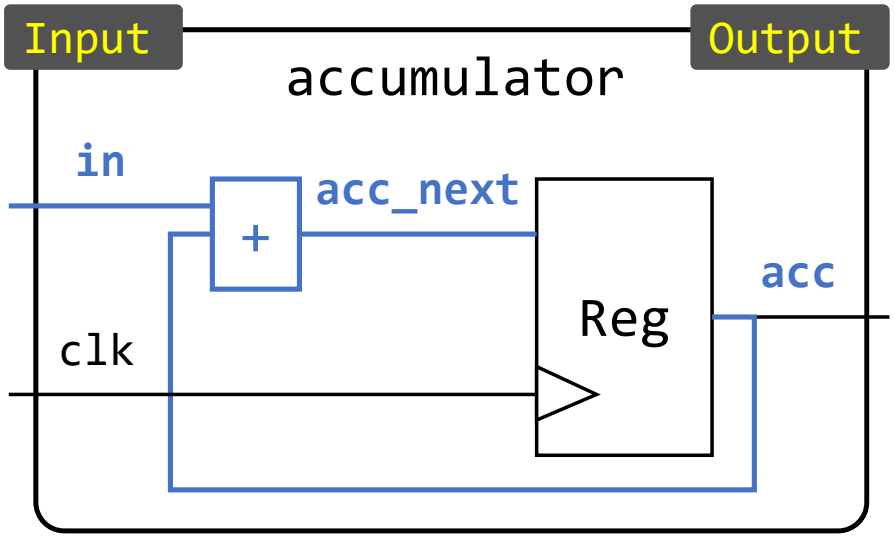
Verilog Overview: Describing A Circuit

```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);




endmodule
```



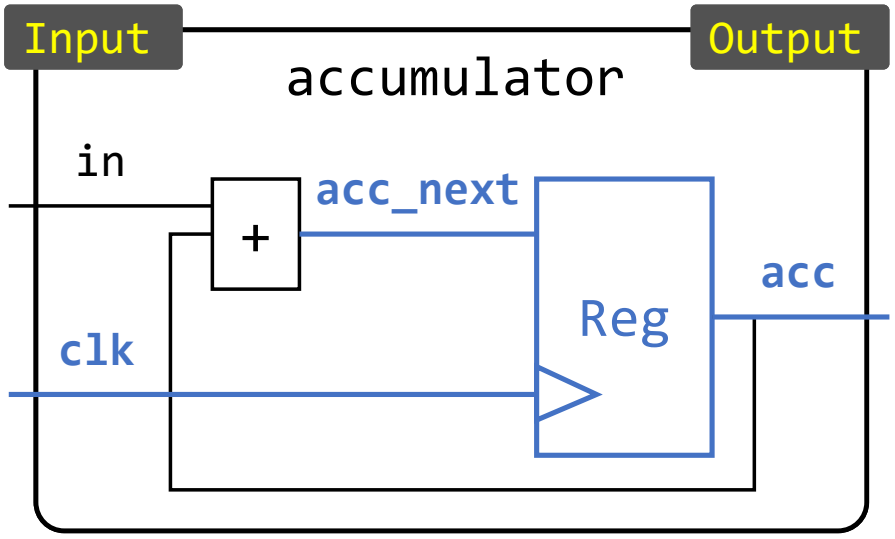
Verilog Overview: Describing A Circuit

```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);
  wire acc_next;
  assign acc_next = acc + in;



endmodule
```



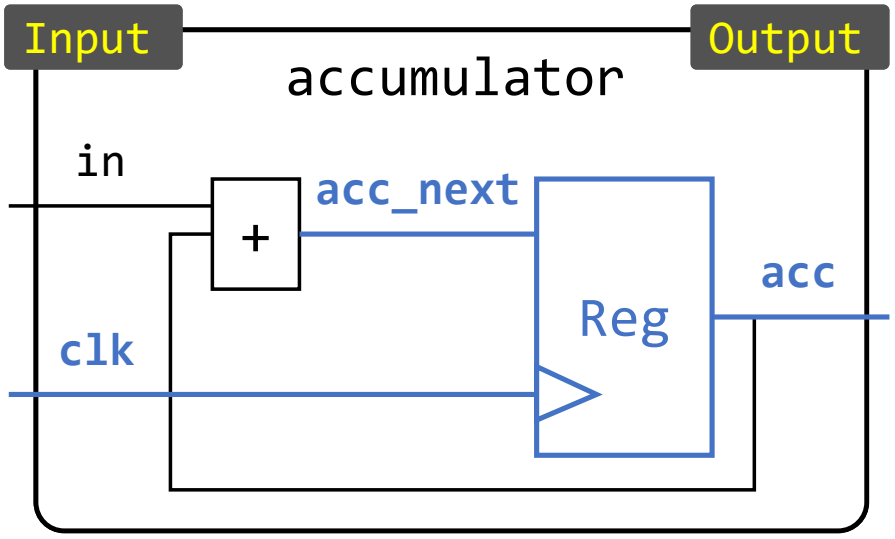
Verilog Overview: Describing A Circuit

```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);
  wire acc_next;
  assign acc_next = acc + in;

  always @(posedge clk)
      acc <= acc_next;
endmodule
```



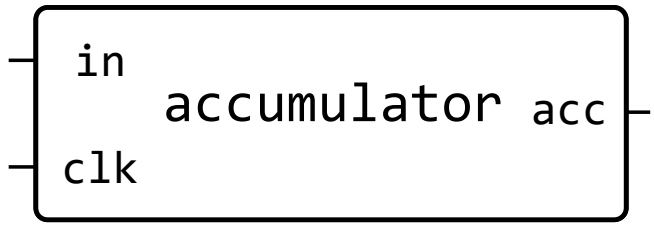
Verilog Overview: Describing A Circuit

```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);
  wire acc_next;
  assign acc_next = acc + in;

  always @(posedge clk)
      acc <= acc_next;
endmodule
```

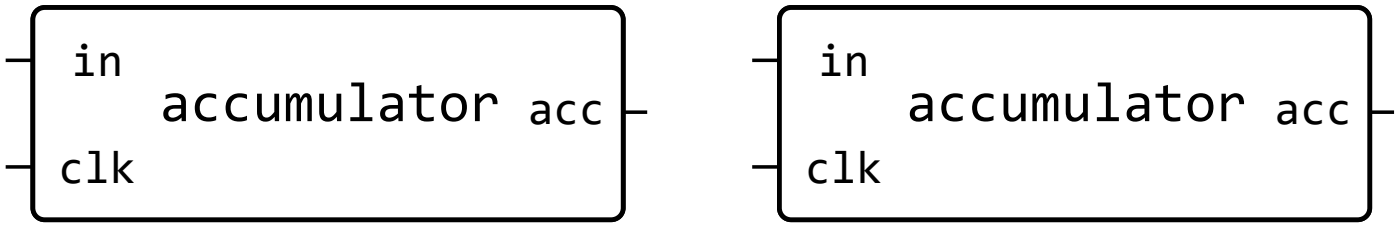


We omit posedge to save space in later examples

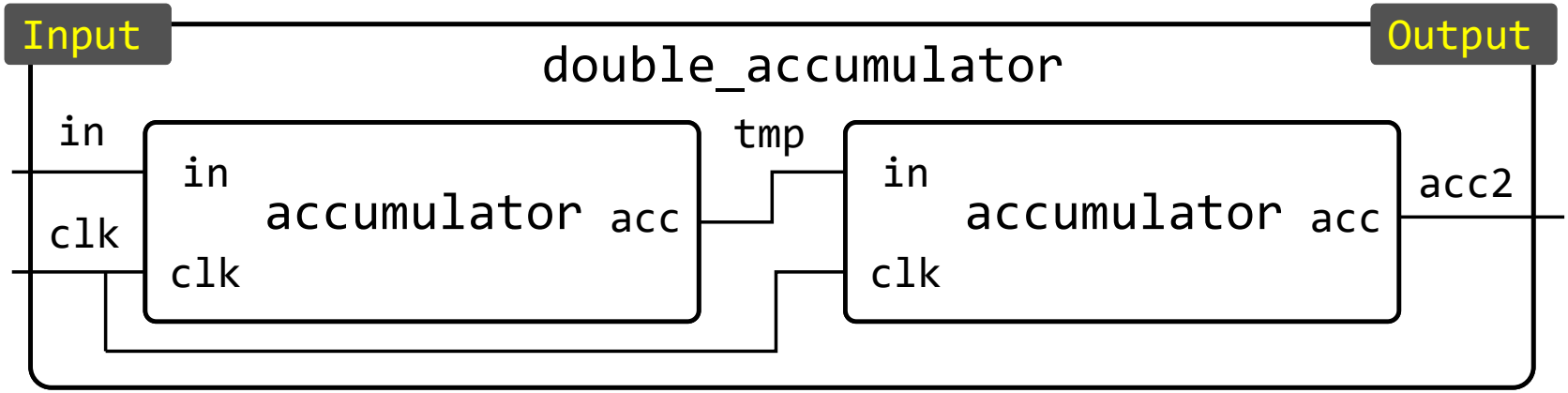
Verilog Overview: Reuse A Circuit



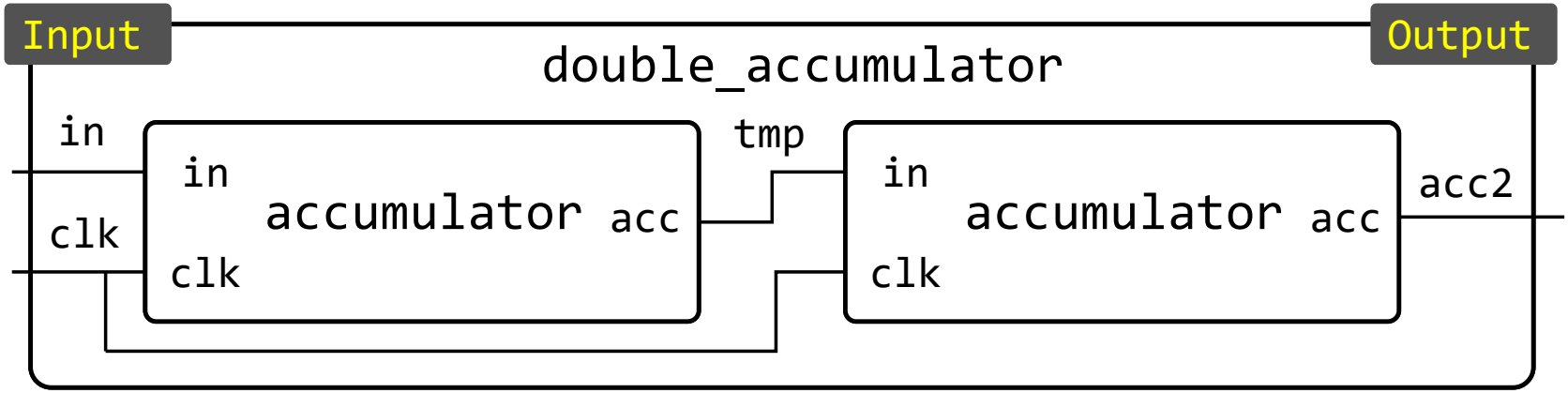
Verilog Overview: Reuse A Circuit



Verilog Overview: Reuse A Circuit

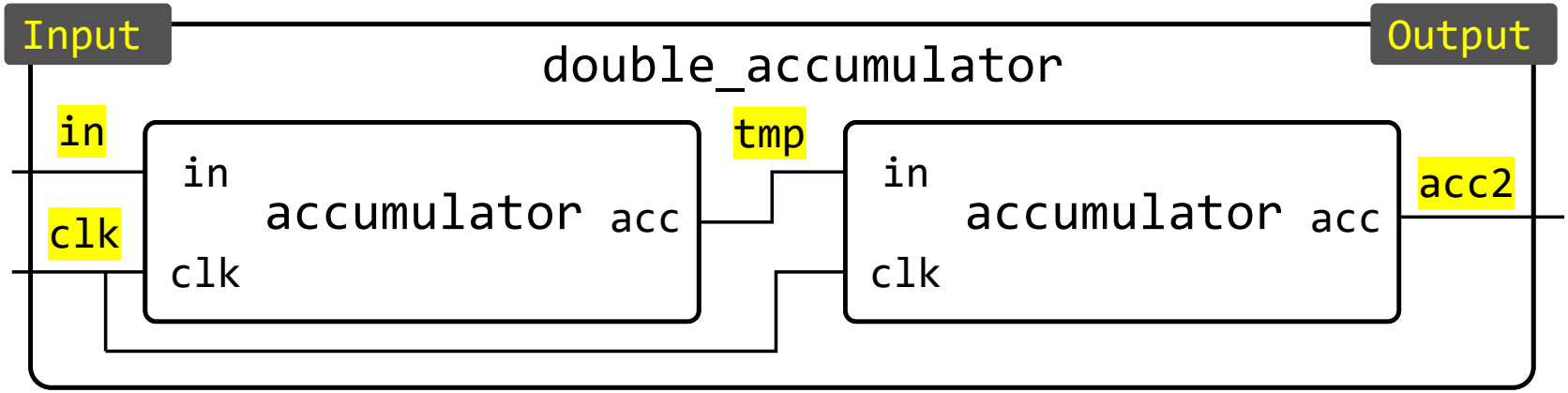


Verilog Overview: Reuse A Circuit




```
module double_accumulator(
  input wire in,
  input wire clk,
  output wire acc2
);
  wire tmp;
```

Verilog Overview: Reuse A Circuit




```
module double_accumulator(
  input wire in,
  input wire clk,
  output wire acc2
);
  wire tmp;
  accumulator a1(
    .in(in), .clk(clk),
    .acc(tmp));
  accumulator a2(
    .in(tmp), .clk(clk),
    .acc(acc2));
endmodule
```

Verilog Overview: Synthesis and Simulation

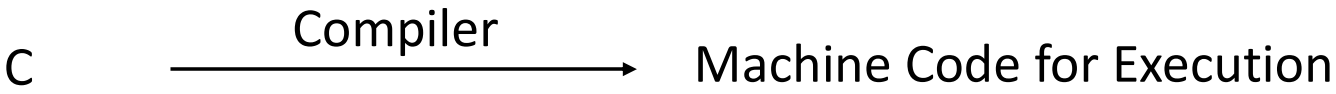
Verilog: a hardware description language

Verilog Overview: Synthesis and Simulation

Verilog: a hardware description language

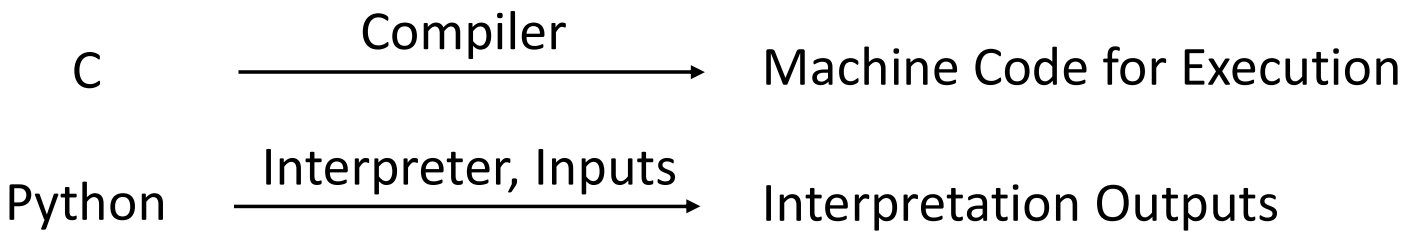
Verilog Overview: Synthesis and Simulation

Verilog: a hardware description language



Verilog Overview: Synthesis and Simulation

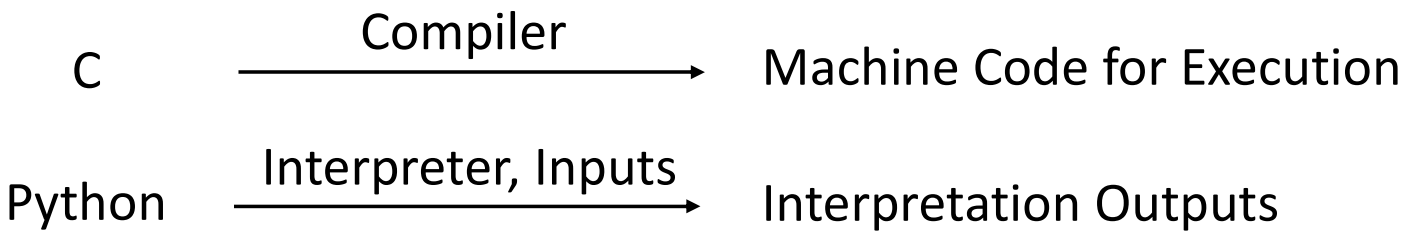
Verilog: a hardware description language



Verilog Overview: Synthesis and Simulation

Verilog: a hardware description language

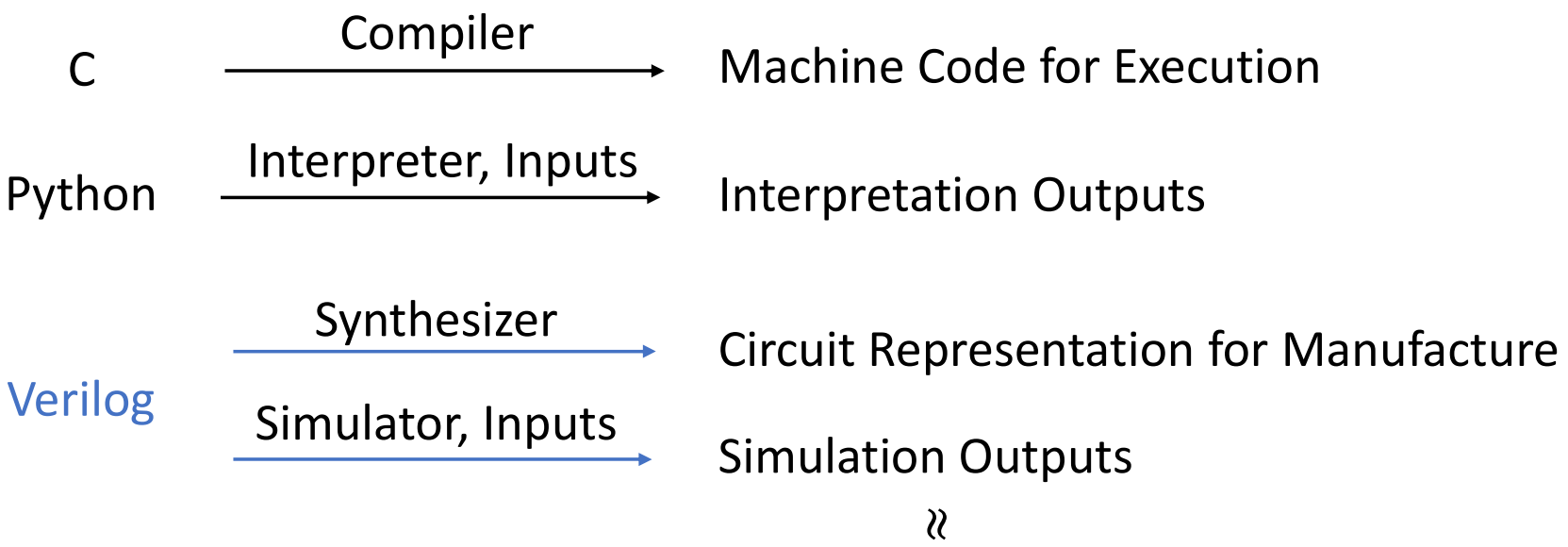
What does its language tool do?



Verilog Overview: Synthesis and Simulation

Verilog: a hardware description language

What does its language tool do?

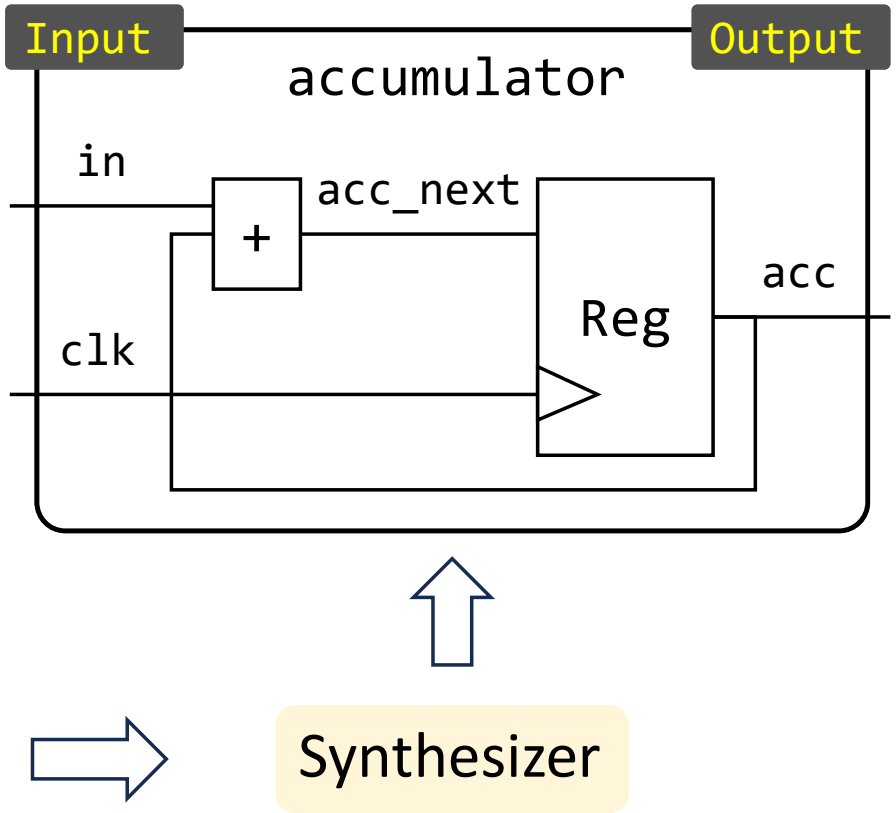


≈

The outputs of the synthesized circuit for identical inputs

Verilog Overview: Synthesis

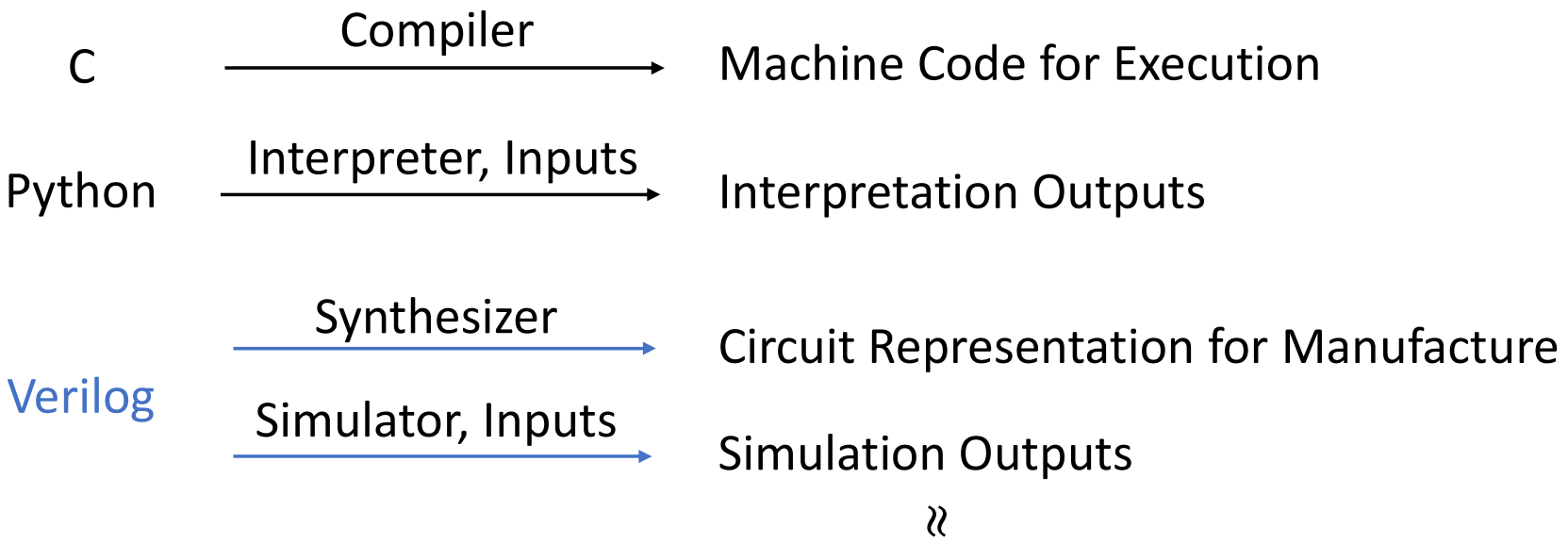
```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);
  wire acc_next;
  assign acc_next = acc + in;

  always @(posedge clk)
      acc <= acc_next;
endmodule
```



Verilog Overview: Synthesis and Simulation

Verilog: a hardware description language

What does its language tool do?

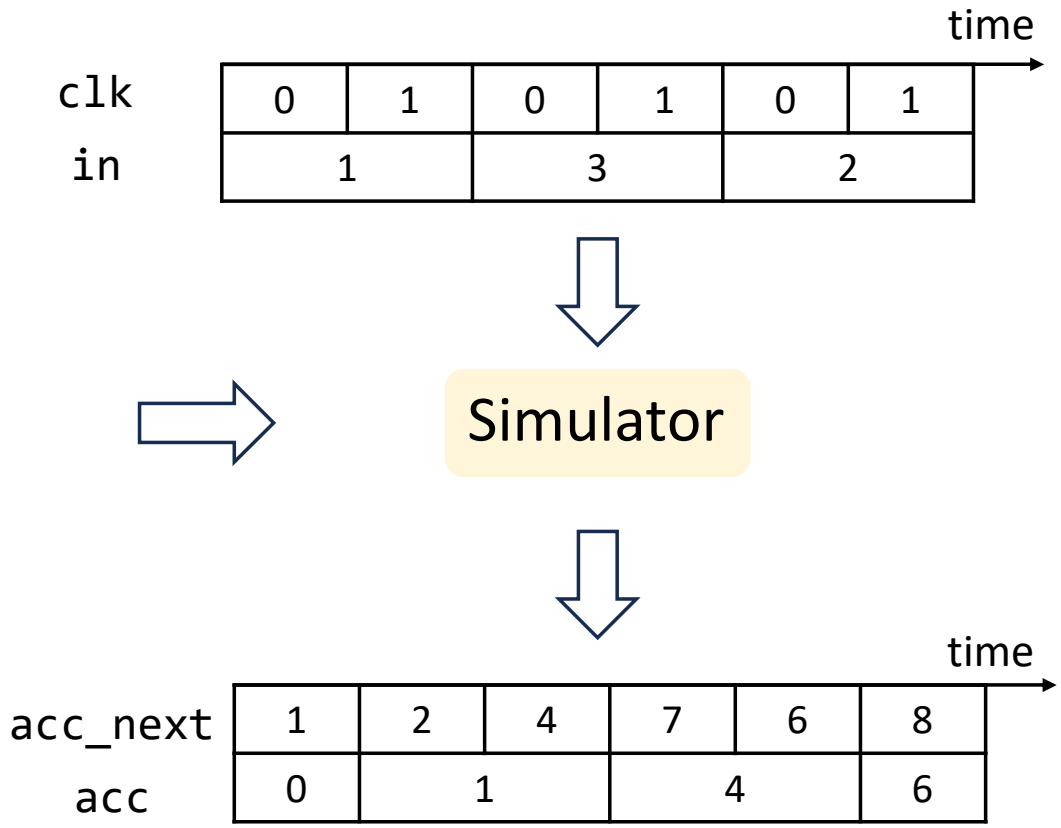


≈

The outputs of the synthesized circuit for identical inputs

Verilog Overview: Simulation

```
module accumulator(
  input wire in,
  input wire clk,
  output reg acc
);
  wire acc_next;
  assign acc_next = acc + in;

  always @(posedge clk)
      acc <= acc_next;
endmodule
```



What’s the Entry-Point of A Verilog Program?

What's the Entry-Point of A Verilog Program?

```
void bar() { }
void foo() { bar(); }
int main() {
  foo();
  foo();
  return 0;
}
```

By convention: the main function

What's the Entry-Point of A Verilog Program?

```
void bar() { }
void foo() { bar(); }
int main() {
  foo();
  foo();
  return 0;
}
```

```
module accumulator (...);
endmodule

module double_accumulator(...);
  accumulator a1(...);
  accumulator a2(...);
endmodule
```

By convention: the main function

What's the Entry-Point of A Verilog Program?

```
void bar() { }
void foo() { bar(); }
int main() {
  foo();
  foo();
  return 0;
}
```

By convention: the main function

```
module accumulator (...);
endmodule

module double_accumulator(...);
  accumulator a1(...);
  accumulator a2(...);
endmodule
```

User-specified, called a top module

Verilog Overview: Summary

- Module
  - The abstraction of a circuit with IO ports
- Module Instantiation
  - Reuse a circuit
- `wire` & `reg`
  - Physical wires and registers (usually)
- `assign` & `always`
  - The computational logic between wires and registers

Verilog Overview: Summary

- Module
  - The abstraction of a circuit with IO ports
- Module Instantiation
  - Reuse a circuit
- `wire` & `reg`
  - Physical wires and registers (usually)
- `assign` & `always`
  - The computational logic between wires and registers

```
module A(input wire in,
         output wire out);
endmodule
```

Verilog Overview: Summary

- Module
  - The abstraction of a circuit with IO ports
- Module Instantiation
  - Reuse a circuit
- `wire` & `reg`
  - Physical wires and registers (usually)
- `assign` & `always`
  - The computational logic between wires and registers

```
module A(input wire in,
         output wire out);
endmodule
```

```
A instance(.in(...), .out(...));
```

Verilog Overview: Summary

- Module
  - The abstraction of a circuit with IO ports
- Module Instantiation
  - Reuse a circuit
- `wire` & `reg`
  - Physical wires and registers (usually)
- `assign` & `always`
  - The computational logic between wires and registers

```
module A(input wire in,
         output wire out);
endmodule
```

```
A instance(.in(...), .out(...));
```

```
wire a, b, clk; reg x;
```

Verilog Overview: Summary

- Module
  - The abstraction of a circuit with IO ports
- Module Instantiation
  - Reuse a circuit
- `wire` & `reg`
  - Physical wires and registers (usually)
- `assign` & `always`
  - The computational logic between wires and registers

```
module A(input wire in,
         output wire out);
endmodule
```

```
A instance(.in(...), .out(...));
```

```
wire a, b, clk; reg x;
```

```
assign b = a + 1;
always @(clk)
   x <= b + 1;
```

Warm-Up Analysis: Printing Module Hierarchy

Given a top module, it is interesting to print which submodules it instantiates, i.e., the module hierarchy.

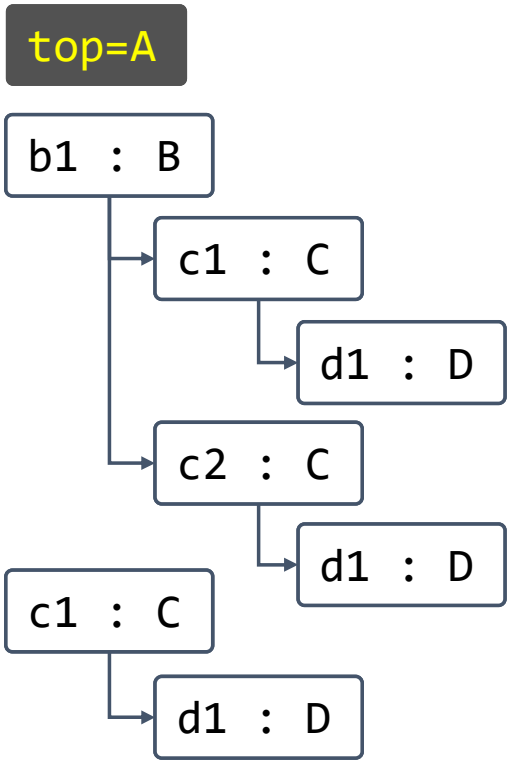
Warm-Up Analysis: Printing Module Hierarchy

Given a top module, it is interesting to print which submodules it instantiates, i.e., the module hierarchy.

```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
module C;
  D d1();
endmodule
module D;
endmodule
```

Warm-Up Analysis: Printing Module Hierarchy

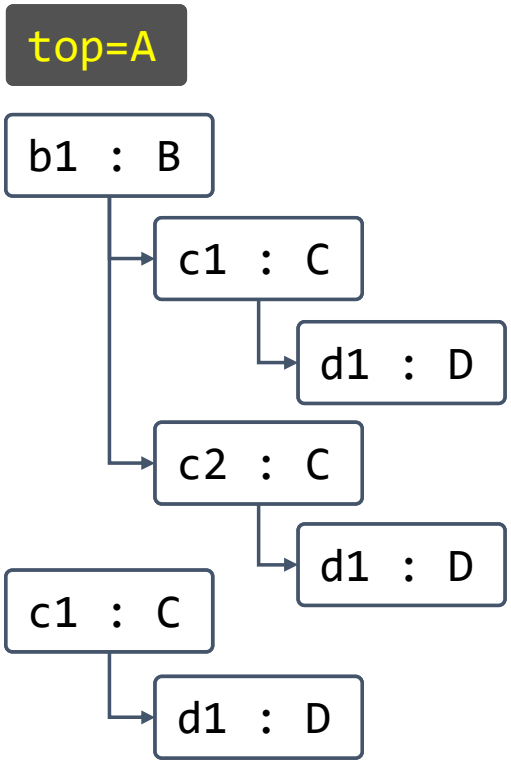
Given a top module, it is interesting to print which submodules it instantiates, i.e., the module hierarchy.

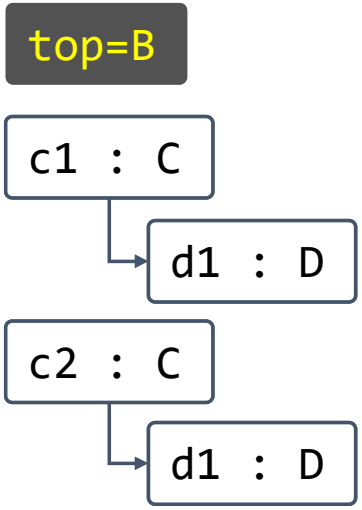



```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
module C;
  D d1();
endmodule
module D;
endmodule
```

Warm-Up Analysis: Printing Module Hierarchy

Given a top module, it is interesting to print which submodules it instantiates, i.e., the module hierarchy.






```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
module C;
  D d1();
endmodule
module D;
endmodule
```

Module Hierarchy Printer: Implementation

Declare an analysis

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

An analysis option for user's external inputs

Module Hierarchy Printer: Implementation

Declare an analysis

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

The entry point of the analysis

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
...
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

Load

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
...
```

```
> qihe compile
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
module B;
  C c1(...);
  C c2(...);
endmodule
...
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  @Analysis.Option
  public String top;

  @Analysis.Run
  public void run(Design design) {
    // Analysis Logic
    if (top == null) return;
    var topModule = design.getModule(top);
    if (topModule == null) return;
    printHierarchy(topModule);
  }

  public void printHierarchy(Module module) { /* TODO */ }
}
```

Declare an analysis

"A"

```
> qihe run hierarchy-printer -i a.qh \
    -c hierarchy-printer.top=A
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

Module Hierarchy Printer: Implementation

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

Console

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "
          + inst.getInstantiatedModule().getName());
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "
          + inst.getInstantiatedModule().getName());
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

`B b1(...);`

Console

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  b1
          + inst.getInstantiatedModule().getName());
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

```
B b1(...);
```

Console

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  b1
          + inst.getInstantiatedModule().getName()); B
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

```
B b1(...);
```

Console

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  b1
          + inst.getInstantiatedModule().getName()); B
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

B b1(...);

Console

```
b1: B
```

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  b1
          + inst.getInstantiatedModule().getName());
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

B b1(...);

B

```
module B;
  C c1(...);
  C c2(...);
endmodule
```

Console

```
b1: B
```

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  b1
          + inst.getInstantiatedModule().getName()); B
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

B b1(...);

```
module B;
  C c1(...);
  C c2(...);
endmodule
```

Console

```
b1: B
  c1: C
    d1: D
  c2: C
    d1: D
```

Module Hierarchy Printer: Implementation

```java
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  c1
          + inst.getInstantiatedModule().getName()); C
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

C c1(...);

```
module C;
  D d1(...);
endmodule
```

Console

```
b1: B
  c1: C
    d1: D
  c2: C
    d1: D
```

Module Hierarchy Printer: Implementation

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(Module module) {
    for (var inst : module.getInstModules()) {
      println(inst.getName() + ": "  c1
          + inst.getInstantiatedModule().getName()); C
      indentLevel++;
      printHierarchy(inst.getInstantiatedModule());
      indentLevel--;
    }
  }
  private int indentLevel = 0;
  private void println(String s) {
    System.out.println("  ".repeat(indentLevel) + s);
  }
}
```

```
module A;
  B b1(...);
  C c1(...);
endmodule
```

C c1(...);

```
module C;
  D d1(...);
endmodule
```

Console

```
b1: B
  c1: C
    d1: D
  c2: C
    d1: D
c1: C
  d1: D
```

Reusing Module Hierarchy Printer

Qihe encourages leveraging existing analyses for building a new one.

Reusing Module Hierarchy Printer

Qihe encourages leveraging existing analyses for building a new one.

Here is a simple example (a more practical example will be presented in the next part)

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(
      Module module) {
    ...
  }

}
```

```
@Analysis(name="another")
public class AnotherAnalysis {









}
```

Reusing Module Hierarchy Printer

Qihe encourages leveraging existing analyses for building a new one.

Here is a simple example (a more practical example will be presented in the next part)

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(
      Module module) {
    ...
  }

}
```

```
@Analysis(name="another")
public class AnotherAnalysis {
  HierarchyPrinter hp;
  @InjectAnalysis
  AnotherAnalyis(HierarchyPrinter hp) {
    this.hp = hp;
  }



}
```

Reusing Module Hierarchy Printer

Qihe encourages leveraging existing analyses for building a new one.

Here is a simple example (a more practical example will be presented in the next part)

```
@Analysis(name="hierarchy-printer")
public class HierarchyPrinter {
  ...
  public void printHierarchy(
      Module module) {
    ...
  }

}
```

```
@Analysis(name="another")
public class AnotherAnalysis {
  HierarchyPrinter hp;
  @InjectAnalysis
  AnotherAnalyis(HierarchyPrinter hp) {
    this.hp = hp;
  }
  @Analysis.Run
  public void run(Design design) {
    hp.printHierarchy(...)
  }
}
```

Warm-Up Analysis: Summary

```
@Analysis(name="id")
public class SomeAnalysis {
  @Analysis.Option
  public String opt;

  @InjectAnalyses
  public SomeAnalysis(...) { ... }

  @Analysis.Run
  public void run(Design design) { ... }

  public Result getResult() { ... }
}
```

User-Specified Options

Import Dependencies

Analysis Logic

Export Results

Outline

Part I: Context

- **Motivation:** Hardware Static Analysis and Qihe Framework
- **Overview:** Qihe's Use Cases and Underlying Workflow

Part II: Getting Started

- **Verilog Overview:** A Quick Example
- **Warm-Up Analysis:** Start with Printing Module Hierarchy

Part III: In Action

- **The Problem:** Missing-Reset Bugs in Hardware
- **The Algorithm:** Detecting Missing-Reset Bugs
- **Step-by-Step Implementation:** Missing-Reset Analysis

Introduction

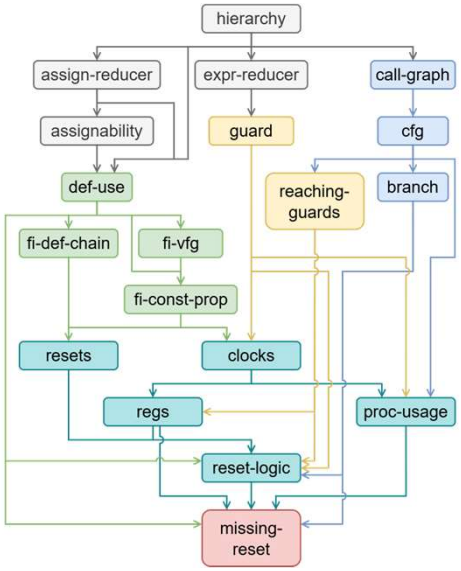
Qihe's Design Philosophy: encourages leveraging existing analysis results to build more sophisticated ones.

This allows us to: with minimal code, implement useful analyses that can find real-world bugs beyond the reach of existing hardware static analysis tools.

Introduction

Qihe's Design Philosophy: encourages leveraging existing analysis results to build more sophisticated ones.

This allows us to: with minimal code, implement useful analyses that can find real-world bugs beyond the reach of existing hardware static analysis tools.

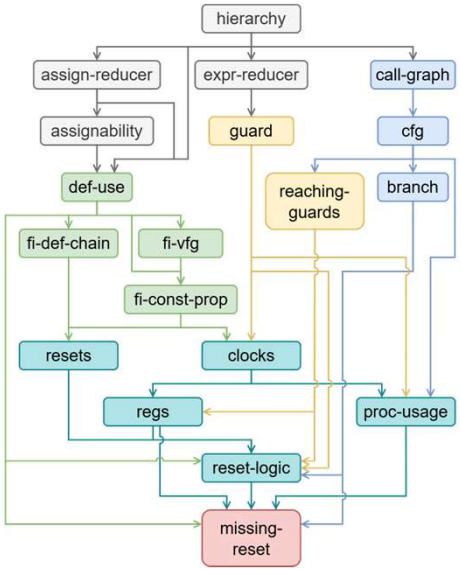


An example: missing-reset analysis

Introduction

Qihe's Design Philosophy: encourages leveraging existing analysis results to build more sophisticated ones.

This allows us to: with minimal code, implement useful analyses that can find real-world bugs beyond the reach of existing hardware static analysis tools.



An example: missing-reset analysis

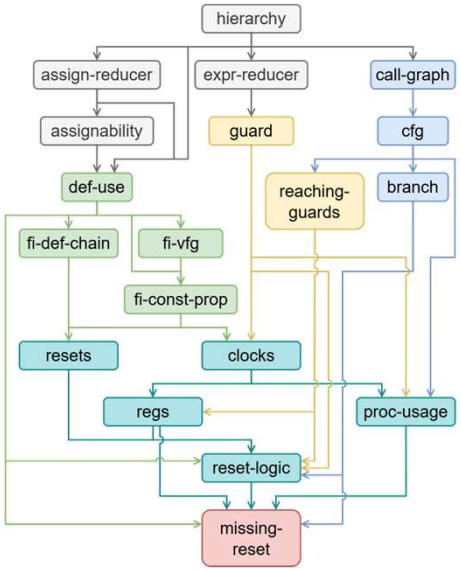
Implemented within Qihe
~450 LoC

Implemented from scratch
~9000 LoC (including dependent analyses)

Introduction

Qihe's Design Philosophy: encourages leveraging existing analysis results to build more sophisticated ones.

This allows us to: with minimal code, implement useful analyses that can find real-world bugs beyond the reach of existing hardware static analysis tools.



An example: missing-reset analysis

Implemented within Qihe
~450 LoC

Implemented from scratch
~9000 LoC (including dependent analyses)

Find six missing-reset bugs in real-world projects undetectable by existing static analysis tools

Introduction

Qihe's Design Philosophy: encourages leveraging existing analysis results to build more sophisticated ones.

This allows us to: with minimal code, implement useful analyses that can find real-world bugs beyond the reach of existing hardware static analysis tools.

Today's Goal: Build a missing-reset analysis

1. Understand the problem
2. Design the algorithm
3. Implement the algorithm

Missing-Reset Bugs: Background

A core concept: a digital circuit is a finite state machine.
The current state is determined by its initial state and input history.

Missing-Reset Bugs: Background

A core concept: a digital circuit is a finite state machine.
The current state is determined by its initial state and input history.

Upon power-on, a circuit's state is undefined.
So, **what sets its initial state?**

Missing-Reset Bugs: Background

A core concept: a digital circuit is a finite state machine.
The current state is determined by its initial state and input history.

Upon power-on, a circuit's state is undefined.
So, **what sets its initial state?**

Typically, a circuit has a dedicated input, called `reset`.
When users activate it (e.g., set it to a logic 1), this signal triggers internal reset logic that forces the circuit into its predefined initial state.

Missing-Reset Bugs: Background

A core concept: a digital circuit is a finite state machine.
The current state is determined by its initial state and input history.

Upon power-on, a circuit's state is undefined.
So, **what sets its initial state?**

Typically, a circuit has a dedicated input, called `reset`.
When users activate it (e.g., set it to a logic 1), this signal triggers internal reset logic that forces the circuit into its predefined initial state.

How is the reset logic designed?

Missing-Reset Bugs: Background

How is the reset logic designed?

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

Directly-reset:

```
input clk, rst;
reg x;
always @(clk)
  if (rst)
    x <= 0
  else
    // ...
```

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

An example for directly-reset:

```
module accumulator(...);
  wire acc_next;
  assign acc_next = acc + in;
  always @(clk)
    acc <= acc_next;
endmodule
```

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

An example for directly-reset:

```
module accumulator(...);
  wire acc_next;
  assign acc_next = acc + in;
  always @(clk)
    acc <= acc_next;
endmodule
```

No initial value

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

An example for directly-reset:

❌

```
module accumulator(...);
  wire acc_next;
  assign acc_next = acc + in;
  always @(clk)
    acc <= acc_next;
endmodule
```

No initial value

✅

```
module accumulator(...,
  input wire rst);
  wire acc_next;
  assign acc_next = acc + in;
  always @(clk)
    if (rst) acc <= 0;
    else acc <= acc_next;
endmodule
```

Missing-Reset Bugs: Background

How is the reset logic designed?

Since a circuit's state is stored in its registers, the reset logic forces these registers reset to a known initial value, either directly or indirectly.

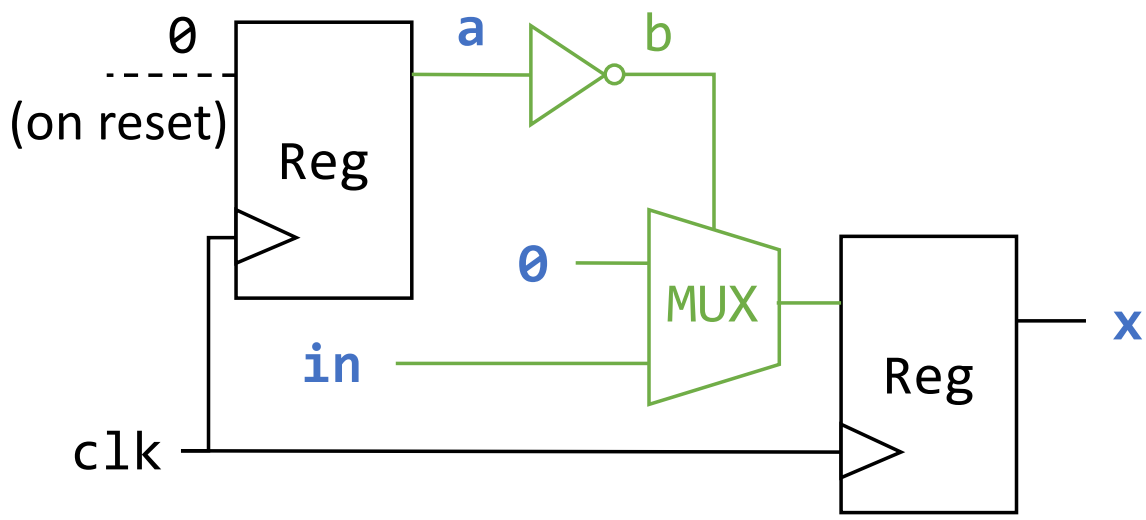
Indirectly-reset: the register is assigned a value computed from the initial states of registers it depend on or inputs by the circuit's combinational logic.

Missing-Reset Bugs: Background

Indirectly-reset: the register is assigned a value computed from the initial states of registers it depend on or inputs by the circuit's combinational logic.

An example:

```
input wire in;
reg a, x; wire b;
always @(clk)
  if (rst) a <= 0;
assign b = ~a;
always @(clk)
  if (b) x <= 0;
  else x <= in;
```



Missing-Reset Bugs

Missing-Reset Bugs

Registers fail to have defined initial value after the circuit is reset, leading to subsequent undefined circuit states.

Analyze: When A Register May Miss Reset?

A Naïve Approach

Report all registers not directly reset as missing-reset.

Analyze: When A Register May Miss Reset?

A Naïve Approach

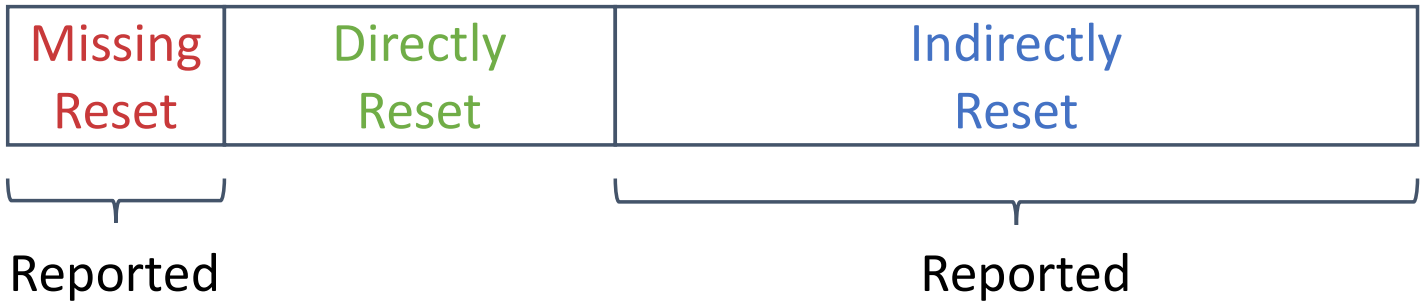
Report all registers not directly reset as missing-reset.

Missing Reset	Directly Reset	Indirectly Reset

Analyze: When A Register May Miss Reset?

A Naïve Approach

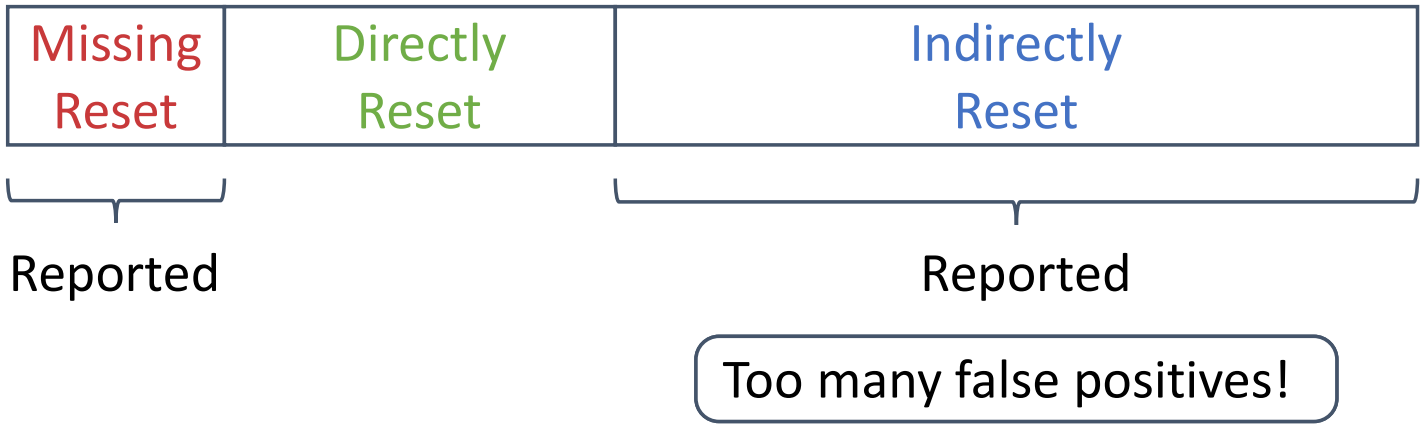
Report all registers not directly reset as missing-reset.



Analyze: When A Register May Miss Reset?

A Naïve Approach

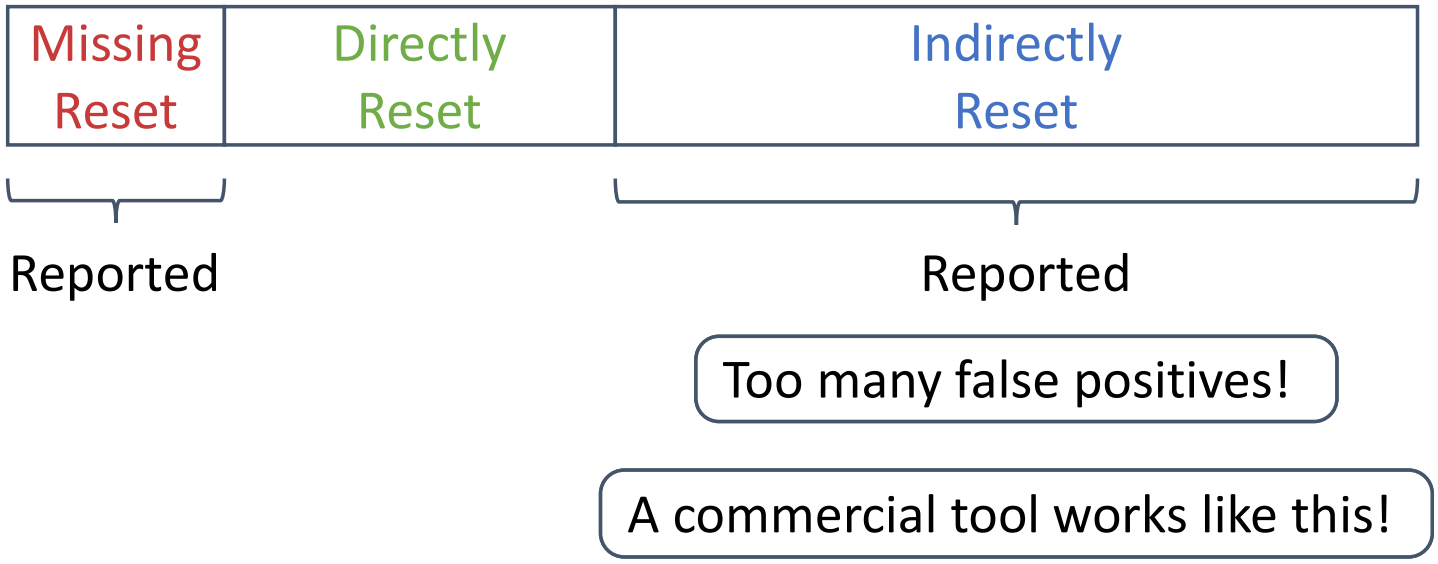
Report all registers not directly reset as missing-reset.



Analyze: When A Register May Miss Reset?

A Naïve Approach

Report all registers not directly reset as missing-reset.



Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers forms a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers forms a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Dependency: a register x depends on y if x's value in next clock cycle is computed from the y's value in current clock cycle.

Analyze: When A Register May Miss Reset?

Our Approach
If a set of registers forms a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Dependency: a register x depends on y if x's value in next clock cycle is computed from the y's value in current clock cycle.

Why we restrict reported registers to those forming a dependency cycle?

Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers forms a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Dependency: a register x depends on y if x's value in next clock cycle is computed from the y's value in current clock cycle.

Why we restrict reported registers to those forming a dependency cycle?

Registers form a dependency cycle is the source of missing-reset bugs

Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers forms a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Dependency: a register x depends on y if x's value in next clock cycle is computed from the y's value in current clock cycle.

Why we restrict reported registers to those forming a dependency cycle?

Registers form a dependency cycle is the source of missing-reset bugs

Let us explain it.

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

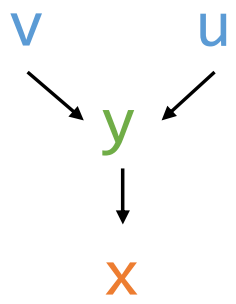
A circuit without registers forming a dependency cycle is naturally (indirectly) reset

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

> A circuit without registers forming a dependency cycle is naturally (indirectly) reset

- These registers can be topologically ordered
- 0-indgree Registers depend on no other registers and driven by inputs

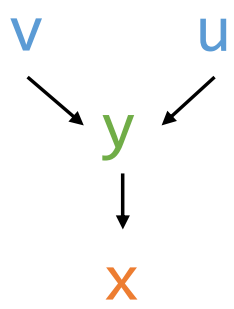


Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

A circuit without registers forming a dependency cycle is naturally (indirectly) reset

- These registers can be topologically ordered
- 0-indgree Registers depend on no other registers and driven by inputs



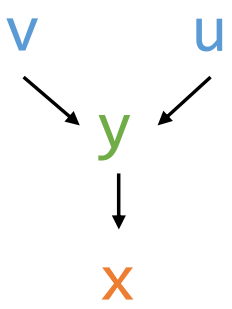
0-indgree registers are firstly indirectly reset by inputs

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

A circuit without registers forming a dependency cycle is naturally (indirectly) reset

- These registers can be topologically ordered
- 0-indgree Registers depend on no other registers and driven by inputs



0-indgree registers are firstly indirectly reset by inputs

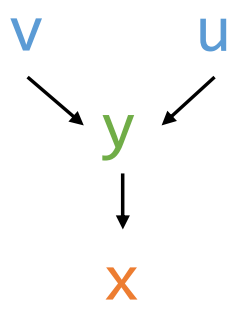
y is (indirectly) reset by u and v

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

A circuit without registers forming a dependency cycle is naturally (indirectly) reset

- These registers can be topologically ordered
- 0-indgree Registers depend on no other registers and driven by inputs



0-indgree registers are firstly indirectly reset by inputs

y is (indirectly) reset by u and v

x is (indirectly) reset by y

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

Why we highlight the word "source"

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

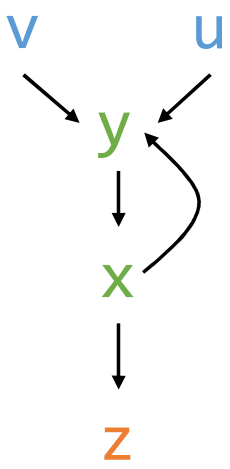
Why we highlight the word "source"

Registers not in the cycle can also miss reset

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

Why we highlight the word "source"



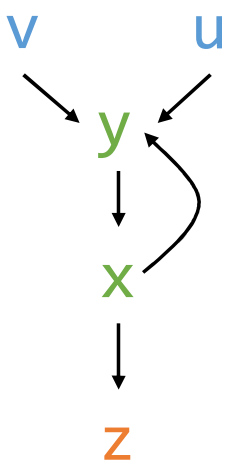
Registers not in the cycle can also miss reset

z is not in a cycle, but if xy miss reset, undefined value will be generated and propagated to z, making z miss reset.

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

Why we highlight the word "source"



Registers not in the cycle can also miss reset

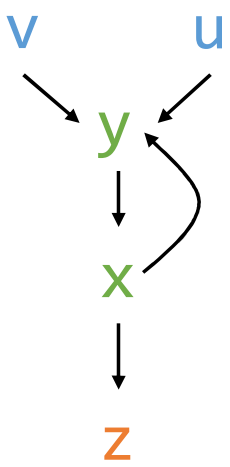
z is not in a cycle, but if xy miss reset, undefined value will be generated and propagated to z, making z miss reset.

We don't report these registers (z) as missing-reset because the root cause is those in the cycle (xy).

Analyze: When A Register May Miss Reset?

Registers forming a dependency cycle is the source of missing-reset bugs

Why we highlight the word "source"



Registers not in the cycle can also miss reset

z is not in a cycle, but if xy miss reset, undefined value will be generated and propagated to z, making z miss reset.

We don't report these registers (z) as missing-reset because the root cause is those in the cycle (xy).

Although this makes our algorithm not sound, it is useful in practice as users only need to check key registers.

Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers form a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Missing Reset	Directly Reset	Indirectly Reset

Analyze: When A Register May Miss Reset?

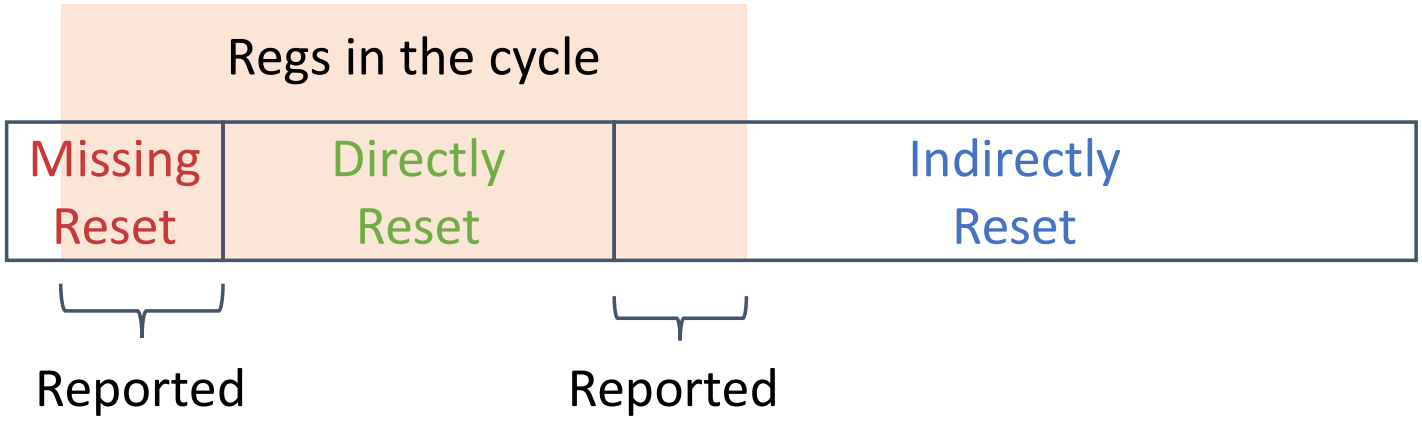
Our Approach

If a set of registers form a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.

Analyze: When A Register May Miss Reset?

Our Approach

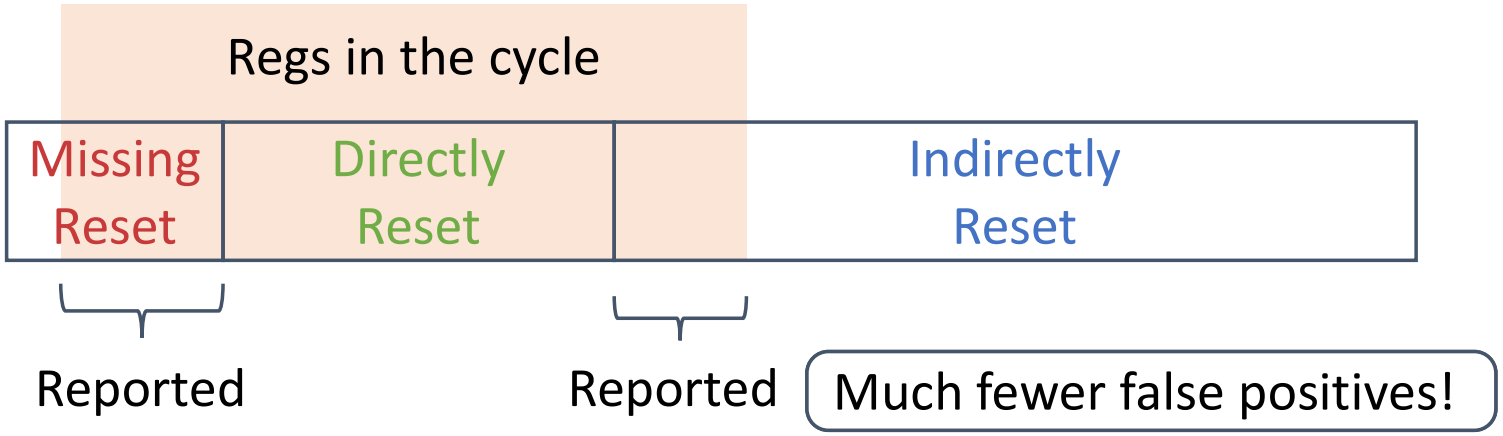
If a set of registers form a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.



Analyze: When A Register May Miss Reset?

Our Approach

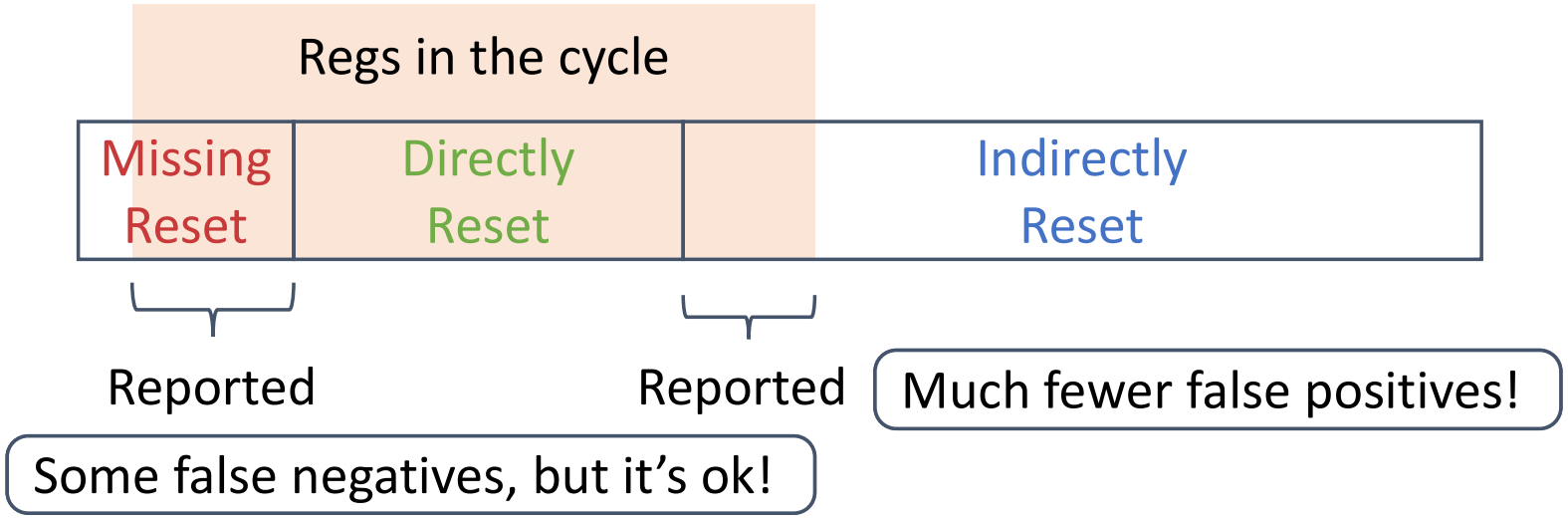
If a set of registers form a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.



Analyze: When A Register May Miss Reset?

Our Approach

If a set of registers form a dependency cycle (X depends Y, Y depends Z, Z depends X), report registers in the cycle not directly reset as missing-reset.



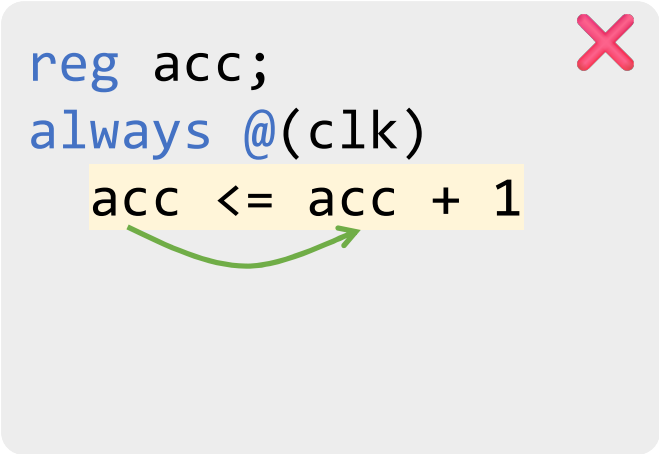
Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 1



Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 1

```
reg acc;
always @(clk)
  acc <= acc + 1
```

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 1

```
reg acc;
always @(clk)
  acc <= acc + 1
```


```
 reg acc;
 always @(clk)
+  if (rst)
+    acc <= 0
+  else
     acc <= acc + 1
```

Missing-Reset Analysis: Algorithm

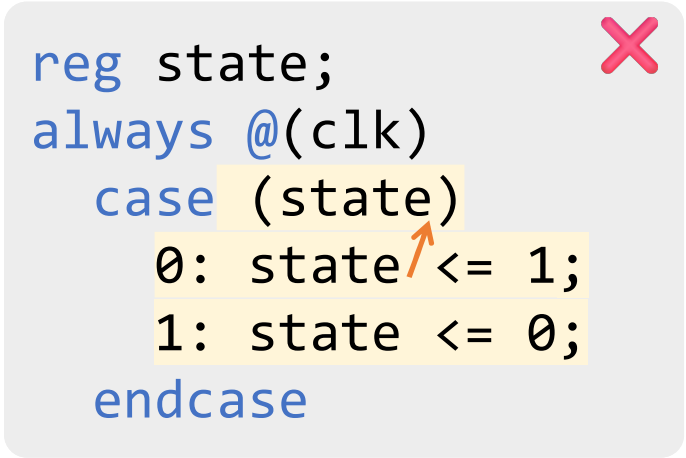
1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 2

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 2

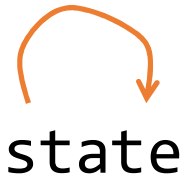


Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 2

```
reg state;
always @(clk)
  case (state)
    0: state <= 1;
    1: state <= 0;
  endcase
```

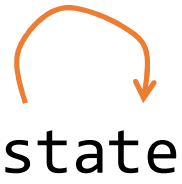


Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 2

```
reg state;
always @(clk)
  case (state)
    0: state <= 1;
    1: state <= 0;
  endcase
```




```
 reg state;
 always @(clk)
+  if (rst)
+    state <= 0
+  else
     case (state)
       0: state <= 1;
       1: state <= 0;
     endcase
```

Missing-Reset Analysis: Algorithm

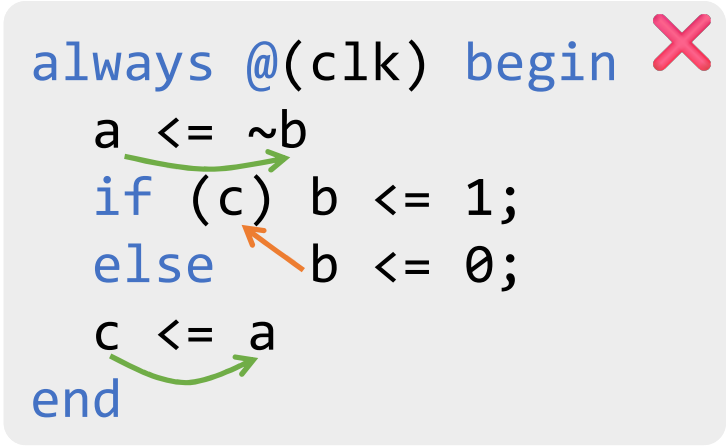
1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 3

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 3

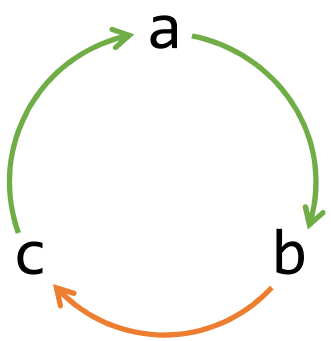


Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 3

```
always @(clk) begin
  a <= ~b
  if (c) b <= 1;
  else   b <= 0;
  c <= a
end
```

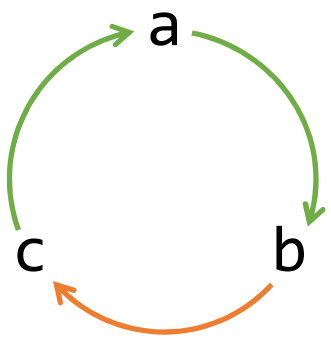


Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Example 3

```
always @(clk) begin
  a <= ~b
  if (c) b <= 1;
  else   b <= 0;
  c <= a
end
```




```
 always @(clk)
+  if (rst) begin
+    a <= 0; b <= 0;
+    c <= 0;
+  end else begin
     a <= ~b
     if (c) b <= 1;
     else   b <= 0;
     c <= a
+  end
```

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

Lacks details for implementation

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Dependency Graph

- Nodes: variables (registers marked blue)
- Edges: data or control dependency

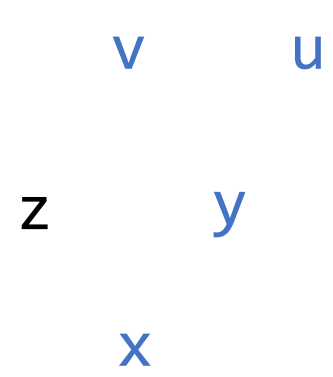
Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Dependency Graph

- Nodes: variables (registers marked blue)
- Edges: data or control dependency



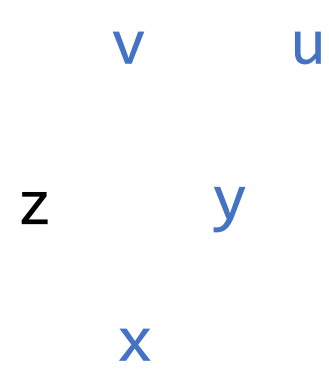
Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Dependency Graph

- Nodes: variables (registers marked blue)
- Edges: data or control dependency



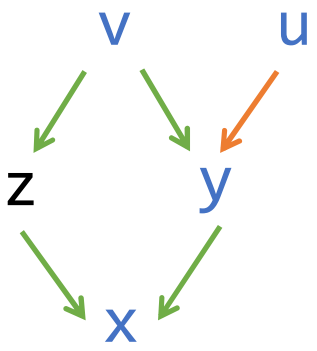
Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Dependency Graph

- Nodes: variables (registers marked blue)
- Edges: data or control dependency



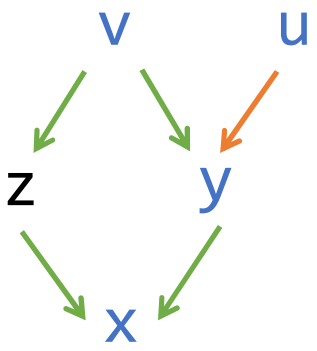
Missing-Reset Analysis: Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

```
always @(clk)
  if (u)
    y <= v;
  z = v + 1;
  x <= z + y;
```

Dependency Graph

- Nodes: variables (registers marked blue)
- Edges: data or control dependency



Find registers in SCCs of the dependency graph

↔ Find registers form a dependency cycle

Missing-Reset Analysis: Refined Algorithm

1. Find registers form a dependency cycle;
2. Report registers in the cycle not directly reset as missing-reset;

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Missing-Reset Analysis: Refined Algorithm

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Missing-Reset Analysis: Refined Algorithm

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Qihe's analyses greatly facilitate these tasks

Missing-Reset Analysis: Refined Algorithm

`DefUseManager` `BranchAnalysis`

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Qihe's analyses greatly facilitate these tasks

Missing-Reset Analysis: Refined Algorithm

`DefUseManager` `BranchAnalysis`

1. Build a graph of variables via data or control dependency;
2. Find `RegInference` ph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Qihe's analyses greatly facilitate these tasks

Missing-Reset Analysis: Refined Algorithm

`DefUseManager` `BranchAnalysis`

1. Build a graph of variables via data or control dependency;
2. Find `RegInference` oh; `ResetLogicInference`
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Qihe's analyses greatly facilitate these tasks

Missing-Reset Analysis: Refined Algorithm

`DefUseManager` `BranchAnalysis`

1. Build a graph of variables via data or control dependency;
2. Find `RegInference` oh; `ResetLogicInference`
3. Report registers in the SCCs that are not directly reset as missing-reset;

Non-trivial to implement

Qihe's analyses greatly facilitate these tasks

Let's implement it!

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  @InjectAnalyses
  public MissingResetAnalysis(...) {

  }

  @Analysis.Run
  public void analyze(Design design) {
    // Analysis Logic
  }

}
```

Declare an analysis

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  @InjectAnalyses
  public MissingResetAnalysis(DefUseManager defUseManager,
                              RegInference regInference,
                              ResetLogicAnalysis resetLogicAnalysis,
                              BranchAnalysis branchAnalysis) {
  }

  ...
}
```

Import dependencies

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  private final DefUseManager defUseManager;
  private final RegInference regInference;
  ...
  @InjectAnalyses
  public MissingResetAnalysis(DefUseManager defUseManager,
                              RegInference regInference,
                              ResetLogicAnalysis resetLogicAnalysis,
                              BranchAnalysis branchAnalysis) {
    this.defUseManager = defUseManager;
    this.regInference = regInference;
    ...
  }
```

Set fields for later use

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  @InjectAnalyses
  public MissingResetAnalysis(...) {
    ...
  }

  @Analysis.Run
  public void analyze(Design design) {
    // Analysis Logic
  }

}
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  @InjectAnalyses
  public MissingResetAnalysis(...) {
    ...
  }

  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    ...
  }
```

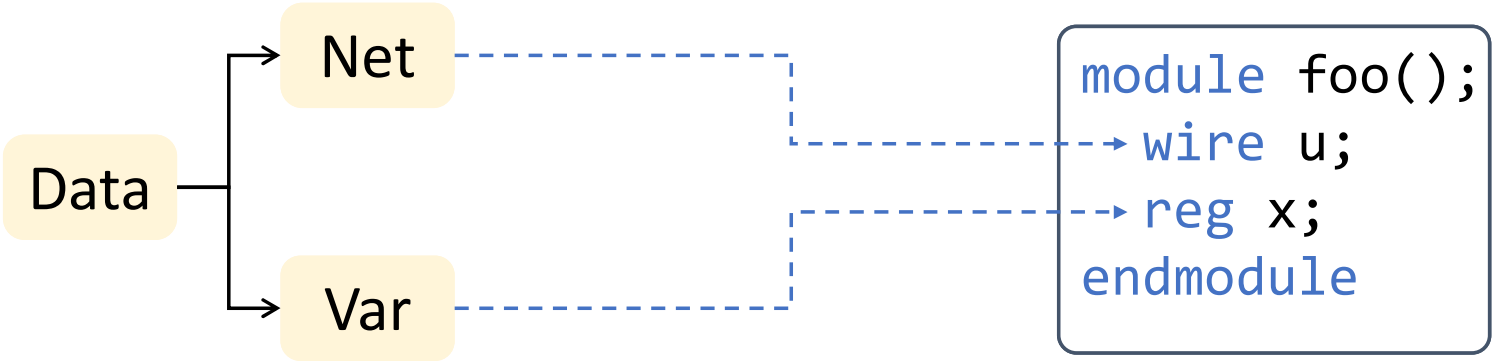
Initialize result set

Missing-Reset Analysis: Implementation

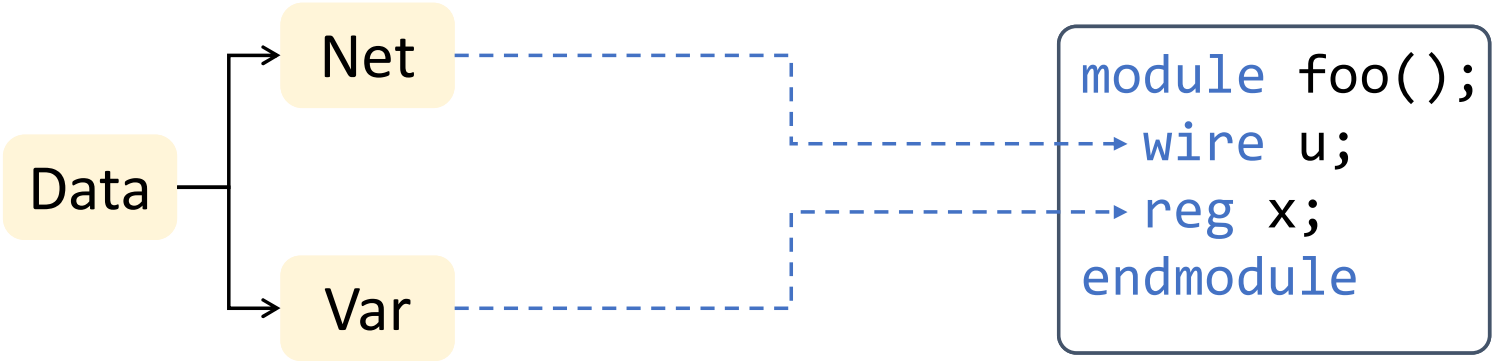
```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {

  @InjectAnalyses
  public MissingResetAnalysis(...) {
    ...
  }

  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    ...
  }
```

Initialize result set

Missing-Reset Analysis: Implementation



Missing-Reset Analysis: Implementation



Verilog reg variables are not necessarily physical registers:

```
reg x, t;
@(clk) x <= ~a;    true
t = ~a;            false
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  ...
  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    ...
  }

}
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  ...
  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    for (var module : design.getModules()) {
        var analyzer = new ModuleAnalyzer(module);
        missingResetRegSet.addAll(analyzer.findMissingResetRegs());
    }
    reportResult();
  }

  private class ModuleAnalyzer { ... }
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  ...
  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    for (var module : design.getModules()) {
        var analyzer = new ModuleAnalyzer(module);
        missingResetRegSet.addAll(analyzer.findMissingResetRegs());
    }
    reportResult();
  }

  private class ModuleAnalyzer { ... }
```

```
module A();
endmodule

module B();
endmodule;
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  ...
  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    for (var module : design.getModules()) {
        var analyzer = new ModuleAnalyzer(module);
        missingResetRegSet.addAll(analyzer.findMissingResetRegs());
    }
    reportResult();
  }

  private class ModuleAnalyzer { ... }
```

per-module analysis

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  ...
  private Set<Data> missingResetRegSet;

  @Analysis.Run
  public void analyze(Design design) {
    missingResetRegSet = new HashSet<>();
    for (var module : design.getModules()) {
        var analyzer = new ModuleAnalyzer(module);
        missingResetRegSet.addAll(analyzer.findMissingResetRegs());
    }
    reportResult();
  }

  private class ModuleAnalyzer { ... }
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private final Module module;

  public ModuleAnalyzer(Module module) {
      this.module = module;
  }

  public Set<Data> findMissingResetRegs() {
    buildDepGraph(collectModuleData());
    var sccs = findSCCs();
    return collectNotDirectlyResetRegs(sccs);
  }
}
```

Set the field for later use

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private final Module module;

  public ModuleAnalyzer(Module module) {
      this.module = module;
  }

  public Set<Data> findMissingResetRegs() {
    buildDepGraph(collectModuleData());
    var sccs = findSCCs();
    return collectNotDirectlyResetRegs(sccs);
  }
}
```

The algorithm

1. Build a graph of variables via data or control dependency;

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private final Module module;

  public ModuleAnalyzer(Module module) {
      this.module = module;
  }

  public Set<Data> findMissingResetRegs() {
    buildDepGraph(collectModuleData());
    var sccs = findSCCs();
    return collectNotDirectlyResetRegs(sccs);
  }
}
```

The algorithm

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private final Module module;

  public ModuleAnalyzer(Module module) {
      this.module = module;
  }

  public Set<Data> findMissingResetRegs() {
    buildDepGraph(collectModuleData());
    var sccs = findSCCs();
    return collectNotDirectlyResetRegs(sccs);
  }
}
```

The algorithm

1. Build a graph of variables via data or control dependency;
2. Find SCCs in the graph;
3. Report registers in the SCCs that are not directly reset as missing-reset;

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private final Module module;

  public ModuleAnalyzer(Module module) {
      this.module = module;
  }

  public Set<Data> findMissingResetRegs() {
    buildDepGraph(collectModuleData());
    var sccs = findSCCs();
    return collectNotDirectlyResetRegs(sccs);
  }
}
```

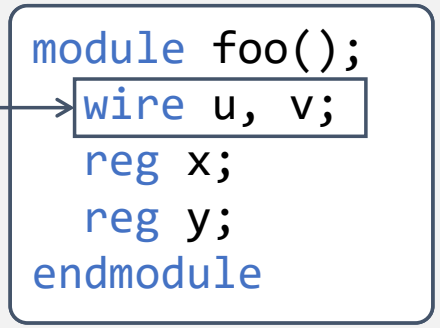
Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private Set<Data> collectModuleData() {
    Set<Data> moduleData = new HashSet<>();
    moduleData.addAll(module.getNets());
    moduleData.addAll(module.getVars());
    return moduleData;
  }

}
```

```
module foo();
  wire u, v;
  reg x;
  reg y;
endmodule
```

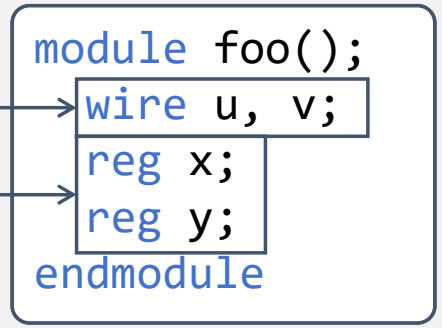
Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private Set<Data> collectModuleData() {
    Set<Data> moduleData = new HashSet<>();
    moduleData.addAll(module.getNets());
    moduleData.addAll(module.getVars());
    return moduleData;
  }

}
```



Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private Set<Data> collectModuleData() {
    Set<Data> moduleData = new HashSet<>();
    moduleData.addAll(module.getNets());
    moduleData.addAll(module.getVars());
    return moduleData;
  }

}
```



Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private MutableDirectedGraph<Data, DefaultEdge> depGraph;

  private void buildDepGraph(Set<Data> moduleDataSet) {
    depGraph = new MutableDirectedGraph<>(DefaultEdge.class);
    depGraph.addNodes(moduleDataSet);
    // Add edges
  }

}
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    // Add edges
  }

}
```

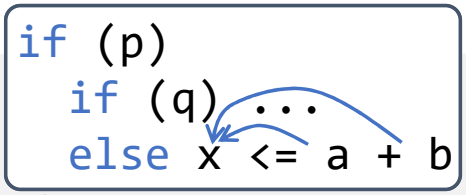
Missing-Reset Analysis: Implementation

```
if (p)
  if (q) ...
  else x <= a + b
```

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```



Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```

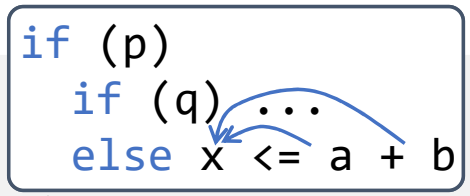
```
if (p)
  if (q) ...
  else x <= a + b
```

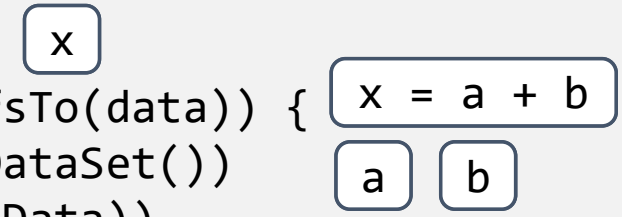
x

x = a + b

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```





Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        for (Data usedData : def.getUsedDataSet())
          if (moduleDataSet.contains(usedData))
            depGraph.addEdge(usedData, data);
        ...
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

x = a + b

a

b

a -> x

b -> x

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        ...
        for (Branch branch : branchAnalysis
                .getBelongingBranches(def.getSource())) {
          var decisionData = branch.getDecisionData();
          if (moduleDataSet.contains(decisionData))
            depGraph.addEdge(decisionData, data);
        }
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

x = a + b

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        ...
        for (Branch branch : branchAnalysis
                .getBelongingBranches(def.getSource())) {
          var decisionData = branch.getDecisionData();
          if (moduleDataSet.contains(decisionData))
            depGraph.addEdge(decisionData, data);
        }
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

x = a + b

if (p), true

if (q), false

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        ...
        for (Branch branch : branchAnalysis
              .getBelongingBranches(def.getSource())) {
          var decisionData = branch.getDecisionData();
          if (moduleDataSet.contains(decisionData))
            depGraph.addEdge(decisionData, data);
        }
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

x = a + b

if (p), true

if (q), false

p

q

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {
  ...
  private void buildDepGraph(Set<Data> moduleDataSet) {
    ...
    for (Data data : moduleDataSet) {
      for (Def def : defUseManager.getDefsTo(data)) {
        ...
        for (Branch branch : branchAnalysis
              .getBelongingBranches(def.getSource())) {
          var decisionData = branch.getDecisionData();
          if (moduleDataSet.contains(decisionData))
            depGraph.addEdge(decisionData, data);
        }
      }
    }
  }
```

```
if (p)
  if (q) ...
  else x <= a + b
```

x

x = a + b

if (p), true

if (q), false

p

q

p -> x

q -> x

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private List<Set<Data>> findSCCs() {
    var conn = new StrongConnectivity<>(depGraph);
    return conn.getConnectedComponents().stream()
        .filter(scc -> hasLoop(scc, depGraph))
        .map(Set::copyOf)
        .toList();
  }

}
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private List<Set<Data>> findSCCs() {
    var conn = new StrongConnectivity<>(depGraph);
    return conn.getConnectedComponents().stream()
        .filter(scc -> hasLoop(scc, depGraph))
        .map(Set::copyOf)
        .toList();
  }

}
```

Qihe's graph algorithm lib

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  private List<Set<Data>> findSCCs() {
    var conn = new StrongConnectivity<>(depGraph);
    return conn.getConnectedComponents().stream()
        .filter(scc -> hasLoop(scc, depGraph))
        .map(Set::copyOf)
        .toList();
  }

}
```

SCC without loops?
The single node without self-loop

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  public Set<Data> collectNotDirectlyResetRegs(
    List<Set<Data>> sccs) {
    Set<Data> result = new HashSet<>();
    for (Set<Data> scc : sccs) {
      for (Data data : scc) {
        if (regInference.isReg(data)
            && !resetLogicAnalysis.hasResetLogic(data))) {
          result.add(data);
        }
      }
    }
    return result;
  }
}
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  public Set<Data> collectNotDirectlyResetRegs(
    List<Set<Data>> sccs) {
    Set<Data> result = new HashSet<>();
    for (Set<Data> scc : sccs) {
      for (Data data : scc) {
        if (regInference.isReg(data)
            && !resetLogicAnalysis.hasResetLogic(data))) {
          result.add(data);
        }
      }
    }
    return result;
  }
}
```

Traverse each node in SCCs

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  public Set<Data> collectNotDirectlyResetR
    List<Set<Data>> sccs) {
    Set<Data> result = new HashSet<>();
    for (Set<Data> scc : sccs) {
      for (Data data : scc) {
        if (regInference.isReg(data)
            && !resetLogicAnalysis.hasResetLogic(data))) {
          result.add(data);
        }
      }
    }
    return result;
  }
}
```

Verilog reg variables are not necessarily physical registers:

```
reg x, t;
@(clk) x <= ~a;    true
t = ~a;            false
```

Missing-Reset Analysis: Implementation

```
private class ModuleAnalyzer {

  public Set<Data> collectNotDirectlyResetRegs(
    List<Set<Data>> sccs) {
    Set<Data> result = new HashSet<>();
    for (Set<Data> scc : sccs) {
      for (Data data : scc) {
        if (regInference.isReg(data)
            && !resetLogicAnalysis.hasResetLogic(data)) {
          result.add(data);
        }
      }
    }
    return result;
  }
}
```

```
if (rst)
  x <= 0
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  private final DiagnosticRecorder recorder;

  public MissingResetAnalysis(..., DiagnosticManager manager) {
    ...
    this.recorder = manager.getRecorder(getClass());
  }

  private void reportResult() {
    for (var data : missingResetRegSet) {
      recorder.error(data, "{} needs resetting", data);
    }
  }
}
```

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  private final DiagnosticRecorder recorder;

  public MissingResetAnalysis(..., DiagnosticManager manager) {
    ...
    this.recorder = manager.getRecorder(getClass());
  }

  private void reportResult() {
    for (var data : missingResetRegSet) {
      recorder.error(data, "{} needs resetting", data);
    }
  }
}
```

Import a diagnostic reporter

- For human
  - print diagnostics to the console
- For machines like language servers
  - dump json

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  private final DiagnosticRecorder recorder;

  public MissingResetAnalysis(..., DiagnosticManager manager) {
    ...
    this.recorder = manager.getRecorder(getClass());
  }

  private void reportResult() {
    for (var data : missingResetRegSet) {
      recorder.error(data, "{} needs resetting", data);
    }
  }
}
```

Import a diagnostic reporter

- For human
  - print diagnostics to the console
- For machines like language servers
  - dump json

Help locate source

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  private final DiagnosticRecorder recorder;

  public MissingResetAnalysis(..., DiagnosticManager manager) {
    ...
    this.recorder = manager.getRecorder(getClass());
  }

  private void reportResult() {
    for (var data : missingResetRegSet) {
      recorder.error(data, "{} needs resetting", data);
    }
  }
}
```

Import a diagnostic reporter
- For human
  - print diagnostics to the console
- For machines like language servers
  - dump json

Help locate source

Formatted string

Missing-Reset Analysis: Implementation

```
@Analysis(name = "missing-reset")
public class MissingResetAnalysis {
  private final DiagnosticRecorder recorder;

  public MissingResetAnalysis(..., DiagnosticManager manager) {
    ...
    this.recorder = manager.getRecorder(getClass());
  }

  private void reportResult() {
    for (var data : missingResetRegSet) {
      recorder.error(data, "{} needs resetting", data);
    }
  }
}
```

```
> qihe run missing-reset -i buggy.qh
ERROR (31:1) axis_frame_fifo.drop_frame needs resetting
ERROR (29:1) axis_frame_fifo.wr_ptr_cur needs resetting
```

Missing-Reset Analysis: Implementation

Congratulations! We made it.

Missing-Reset Analysis: Implementation

Congratulations! We made it.

The missing-reset analysis here only takes about 150 lines of code based on the Qihe's infrastructure.

Missing-Reset Analysis: Implementation

Congratulations! We made it.

The missing-reset analysis here only takes about 150 lines of code based on the Qihe's infrastructure.

Why Qihe's missing-reset analysis takes about 450 lines of code?

Missing-Reset Analysis: Implementation

Congratulations! We made it.

The missing-reset analysis here only takes about 150 lines of code based on the Qihe's infrastructure.

Why Qihe's missing-reset analysis takes about 450 lines of code?

To make it more useful in practice, we use some techniques to reduce false positives, though this comes at the cost of potentially missing some true bugs.

Missing-Reset Analysis: Implementation

Congratulations! We made it.

The missing-reset analysis here only takes about 150 lines of code based on the Qihe's infrastructure.

Why Qihe's missing-reset analysis takes about 450 lines of code?

To make it more useful in practice, we use some techniques to reduce false positives, though this comes at the cost of potentially missing some true bugs.

e.g. ... report registers in the cycle not directly reset as missing-reset.

→ ... report registers in the cycle only if none of them are directly reset.

Where to Go?

Visit our website to:

- Get all Qihe source code (100K+ LoC)
- Read our paper for research details

  Qihe: A General-Purpose Static Analysis Framework for Verilog
  Qinlin Chen, Nairen Zhang, Jinpeng Wang, Jiacai Cui,
  Tian Tan, Xiaoxing Ma, Chang Xu, Jian Lu, Yue Li

- Explore detailed documentation to master Qihe
- Try developing new hardware analyses!

qihe.pascal-lab.net

Where to Go?

Visit our website to:

- Get all Qihe source code (100K+ LoC)
- Read our paper for research details

  Qihe: A General-Purpose Static Analysis Framework for Verilog
  Qinlin Chen, Nairen Zhang, Jinpeng Wang, Jiacai Cui,
  Tian Tan, Xiaoxing Ma, Chang Xu, Jian Lu, Yue Li

- Explore detailed documentation to master Qihe
- Try developing new hardware analyses!

qihe.pascal-lab.net

Read our Zhihu post to explore the story behind Qihe:

「骑河」诞生记——被学生带飞的南大教授 By Yue Li
https://zhuanlan.zhihu.com/p/1958947785744311914

Where to Go?

Visit our website to:

- Get all Qihe source code (100K+ LoC)
- Read our paper for research details

  Qihe: A General-Purpose Static Analysis Framework for Verilog
  Qinlin Chen, Nairen Zhang, Jinpeng Wang, Jiacai Cui,
  Tian Tan, Xiaoxing Ma, Chang Xu, Jian Lu, Yue Li

- Explore detailed documentation to master Qihe
- Try developing new hardware analyses!

Read our Zhihu post to explore the story behind Qihe:

「骑河」诞生记——被学生带飞的南大教授 By Yue Li
https://zhuanlan.zhihu.com/p/1958947785744311914

qihe.pascal-lab.net

Questions or Discussions?
Feel free to contact us!
qinlinchen@smail.nju.edu.cn
tiantan@nju.edu.cn
yueli@nju.edu.cn