# CEGAR+SMT: Formal Verification of Control Logic in the Reveal System

***Karem A. Sakallah****

University of Michigan, Ann Arbor

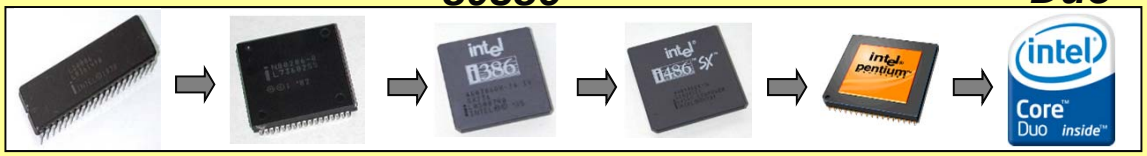
June 13, 2011

Presented at SAT/SMT Summer School

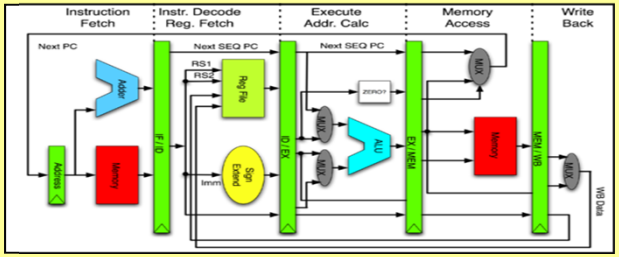
MIT

* Joint work with Zaher Andraus and Mark Liffiton

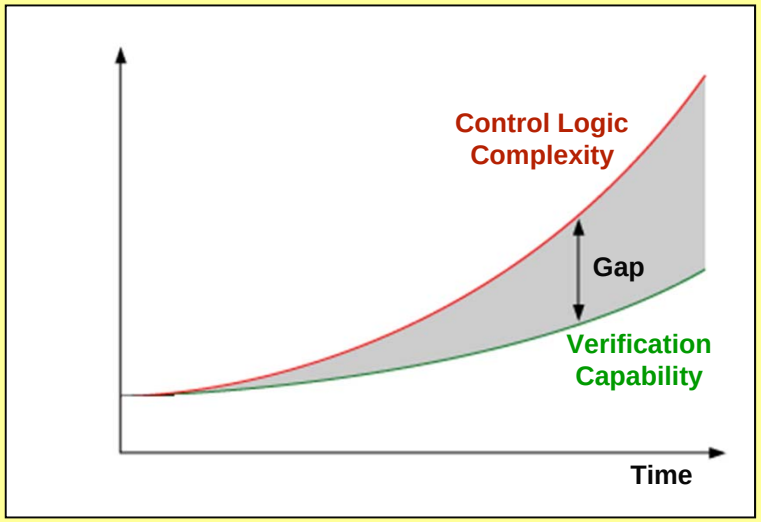
# Motivation




Requirements:
Power, Performance,
Reliability, Features, …




High-Level Control Optimizations:
Pipelining, Caching, Multi-Core,
Power Management, ..

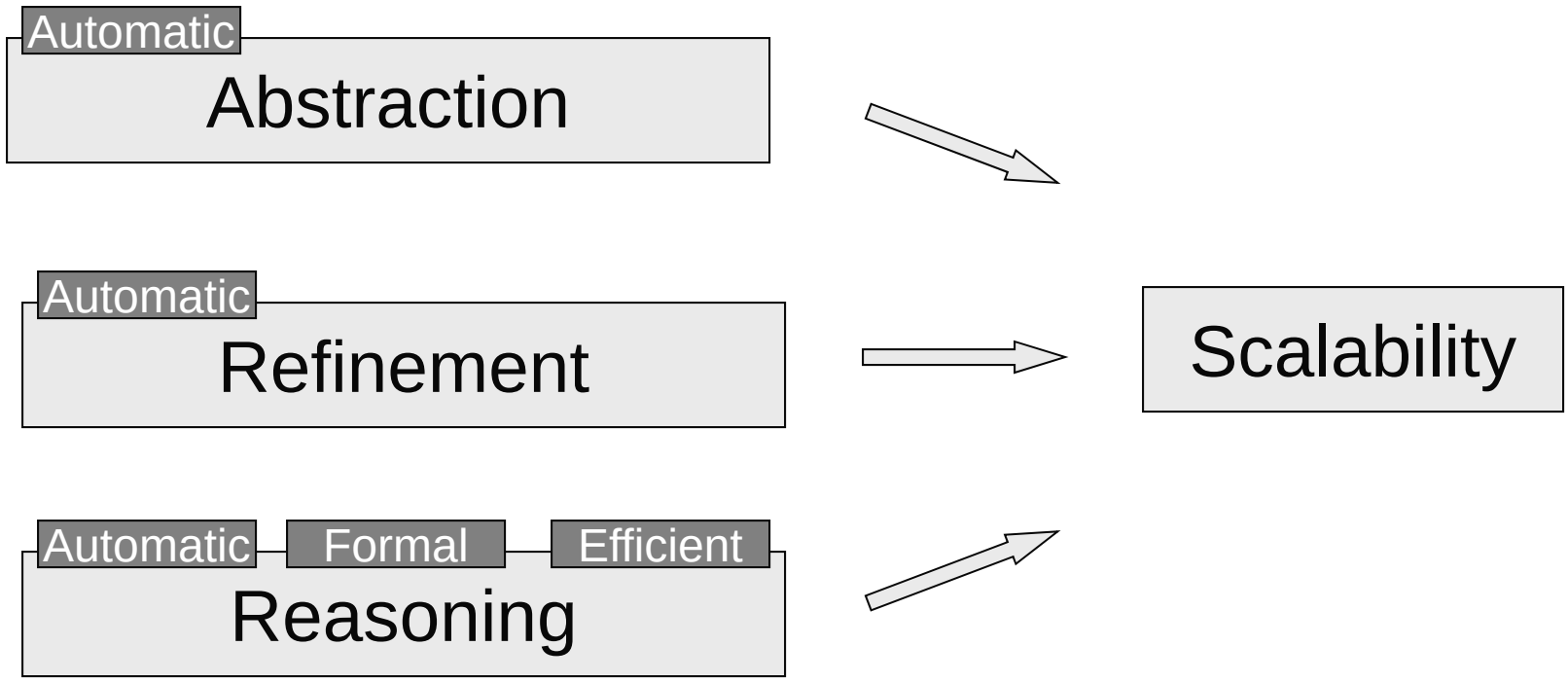



Design Control Logic:
Complex, Error-Prone, Non-reusable, ..

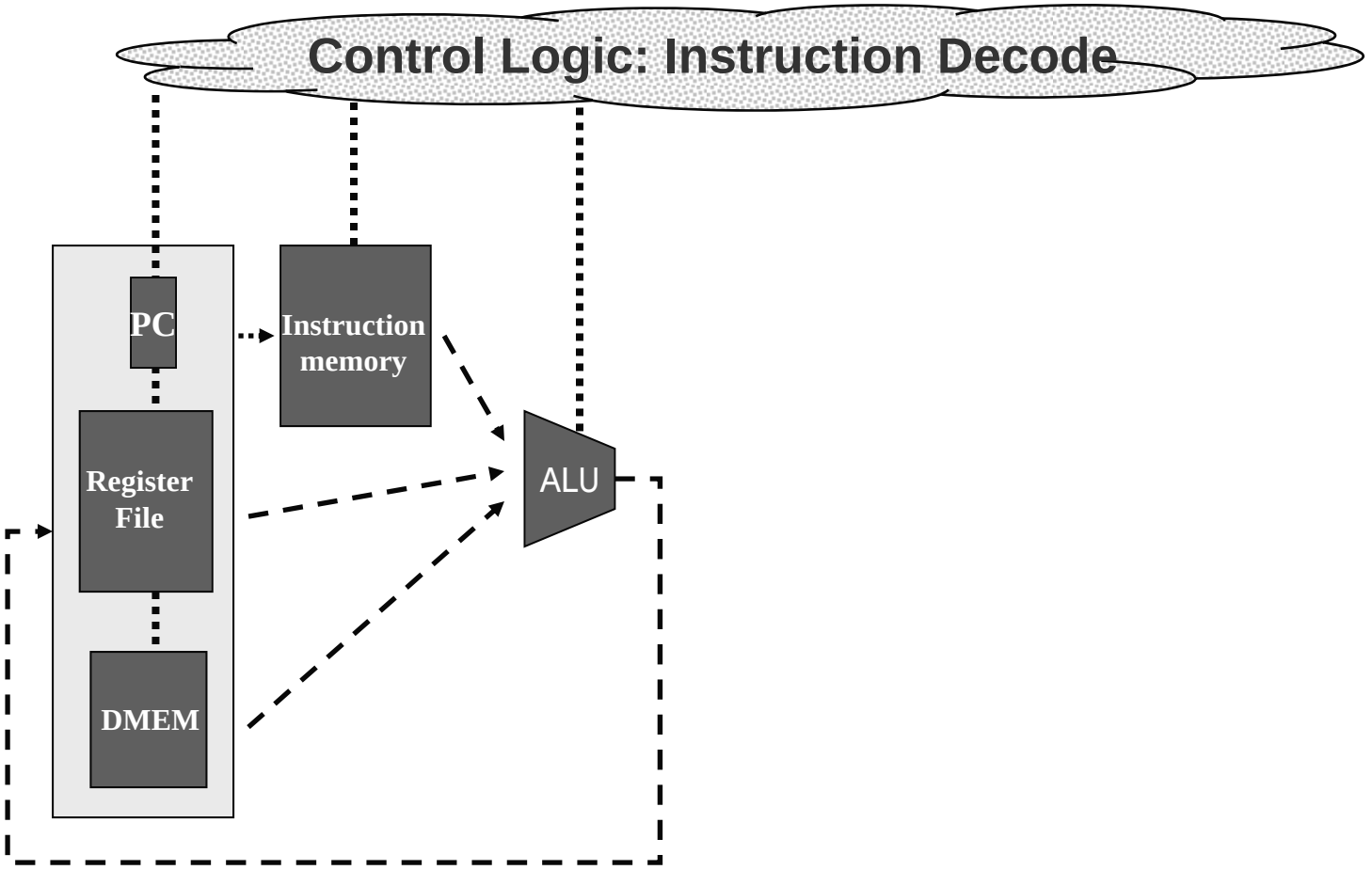
Lack of Automation Limits Scalability

**We Demonstrate a Fully Automatic Approach to Formal Verification of Control Logic**

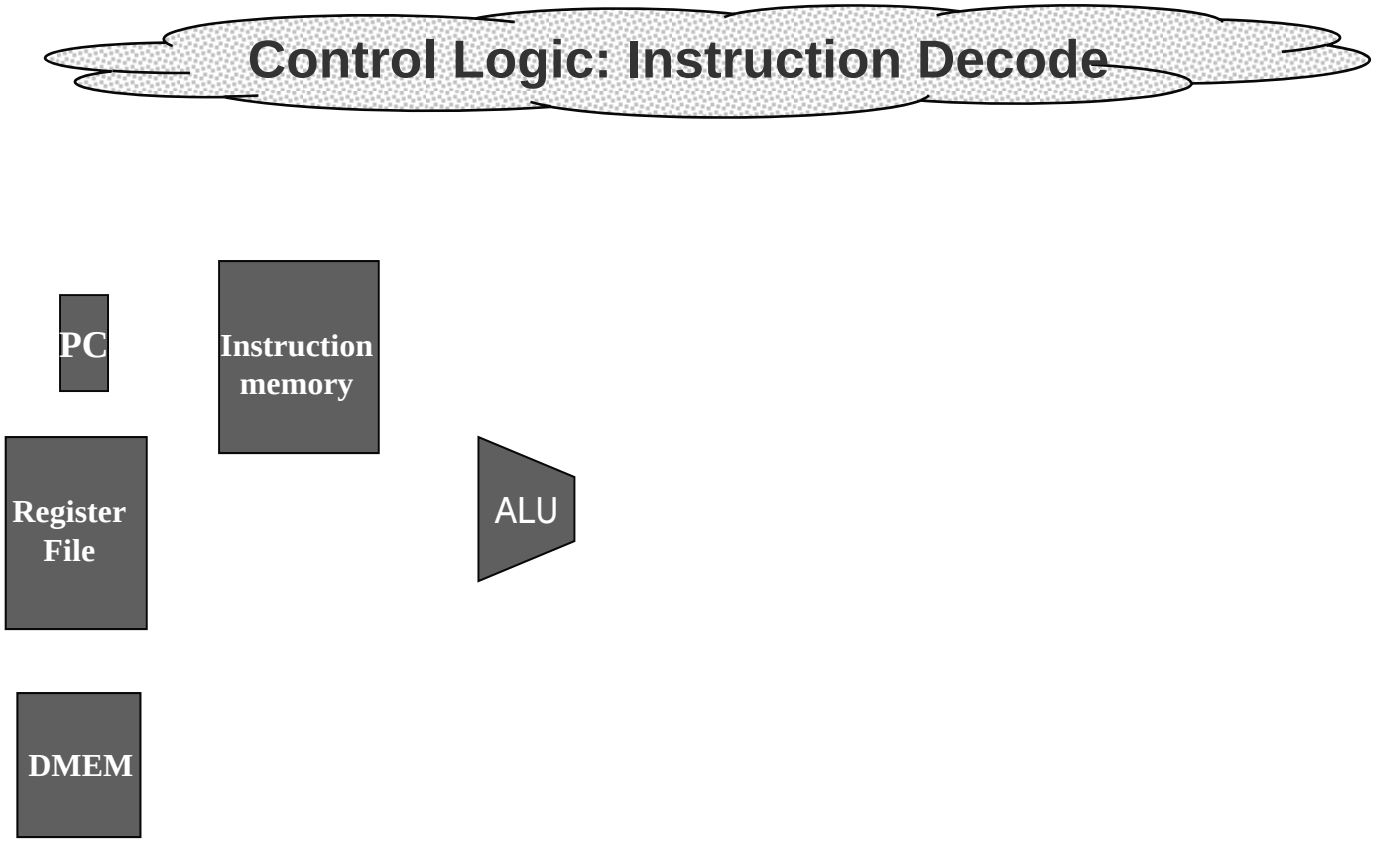
# Turn-Key Formal Verification of **Control Logic** in Digital Systems

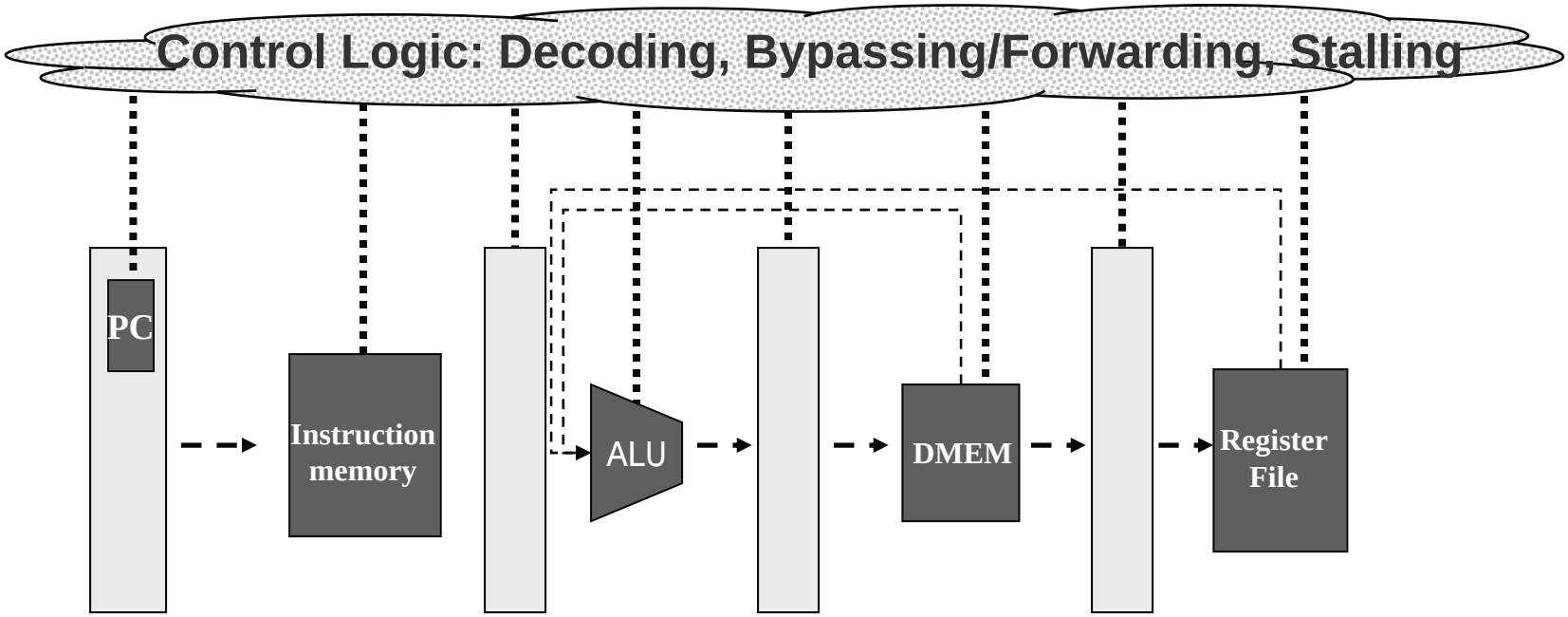


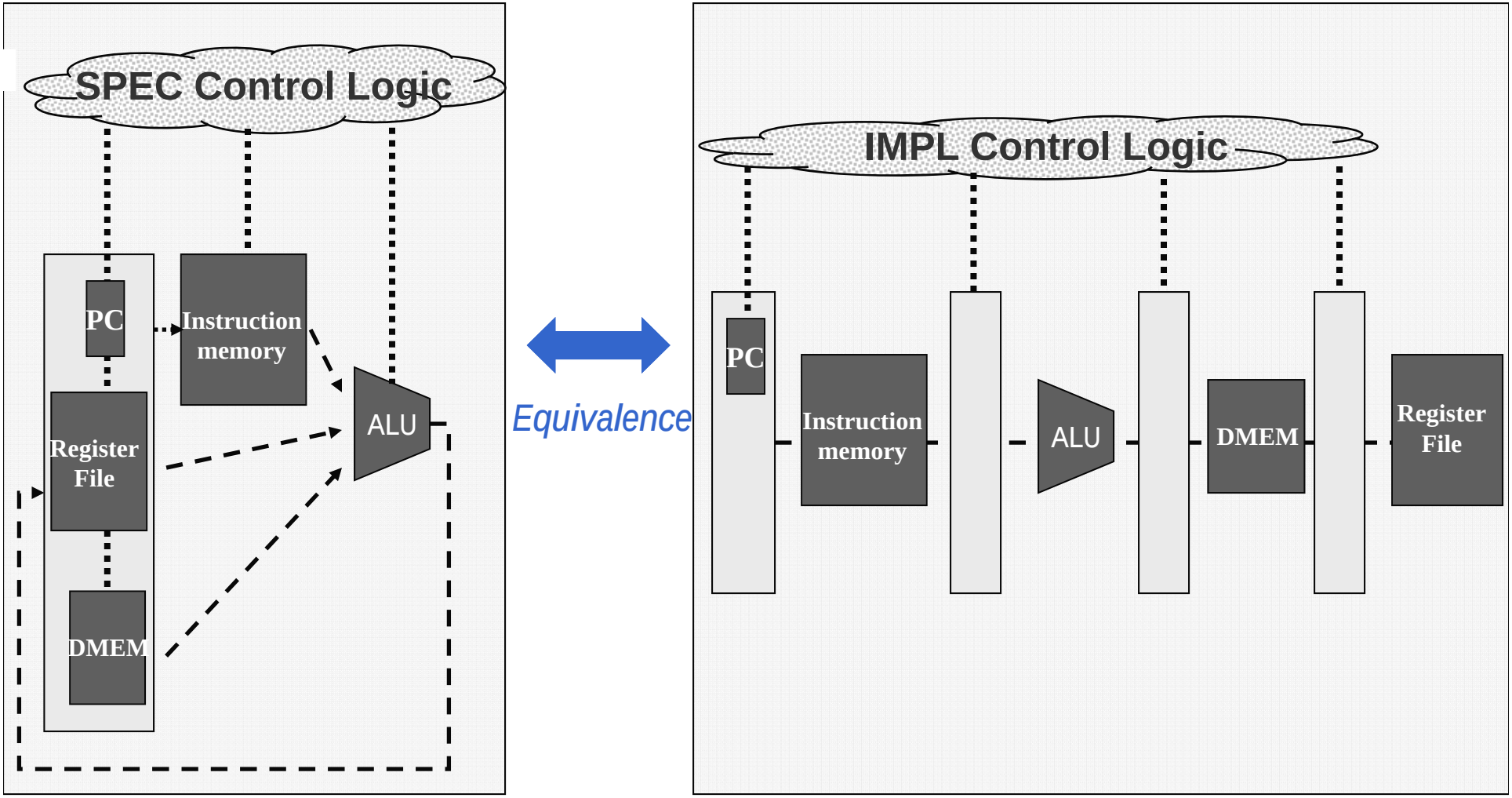
# Pipelining


Control Logic: Instruction Decode
PC
Instruction memory
Register File
ALU
DMEM

# Pipelining


Control Logic: Instruction Decode
PC
Instruction memory
Register File
ALU
DMEM

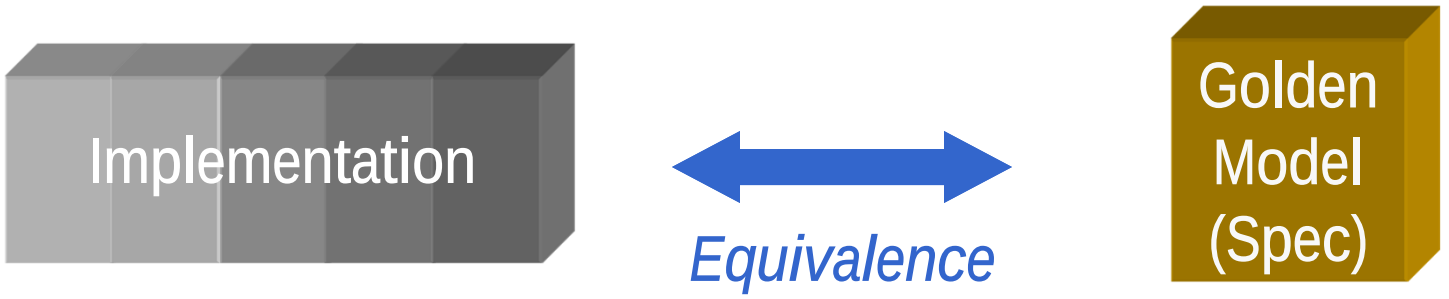

Control Logic: Decoding, Bypassing/Forwarding, Stalling
PC
Instruction memory
ALU
DMEM
Register File


SPEC Control Logic
PC
Instruction memory
Register File
ALU
DMEM
Equivalence
IMPL Control Logic
PC
Instruction memory
ALU
DMEM
Register File

# Outline

- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Conclusions

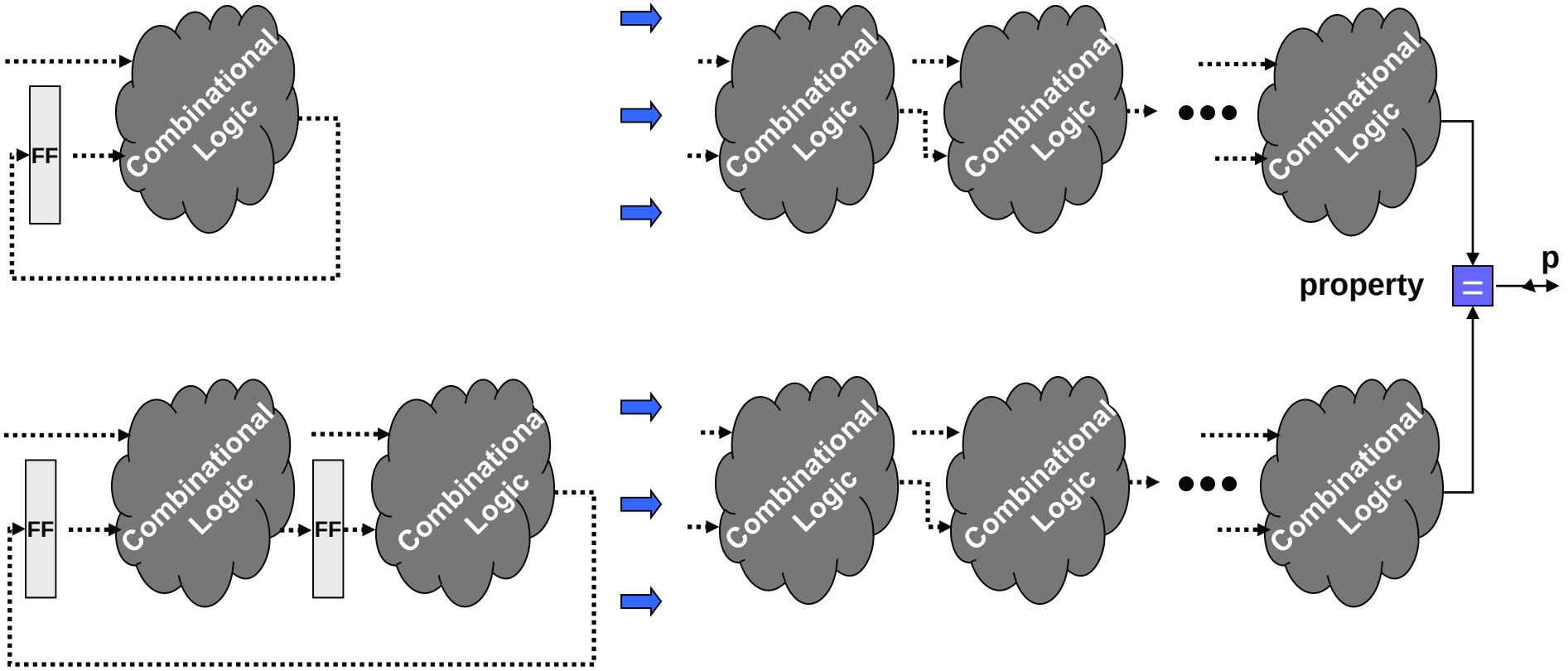
# Verification Framework



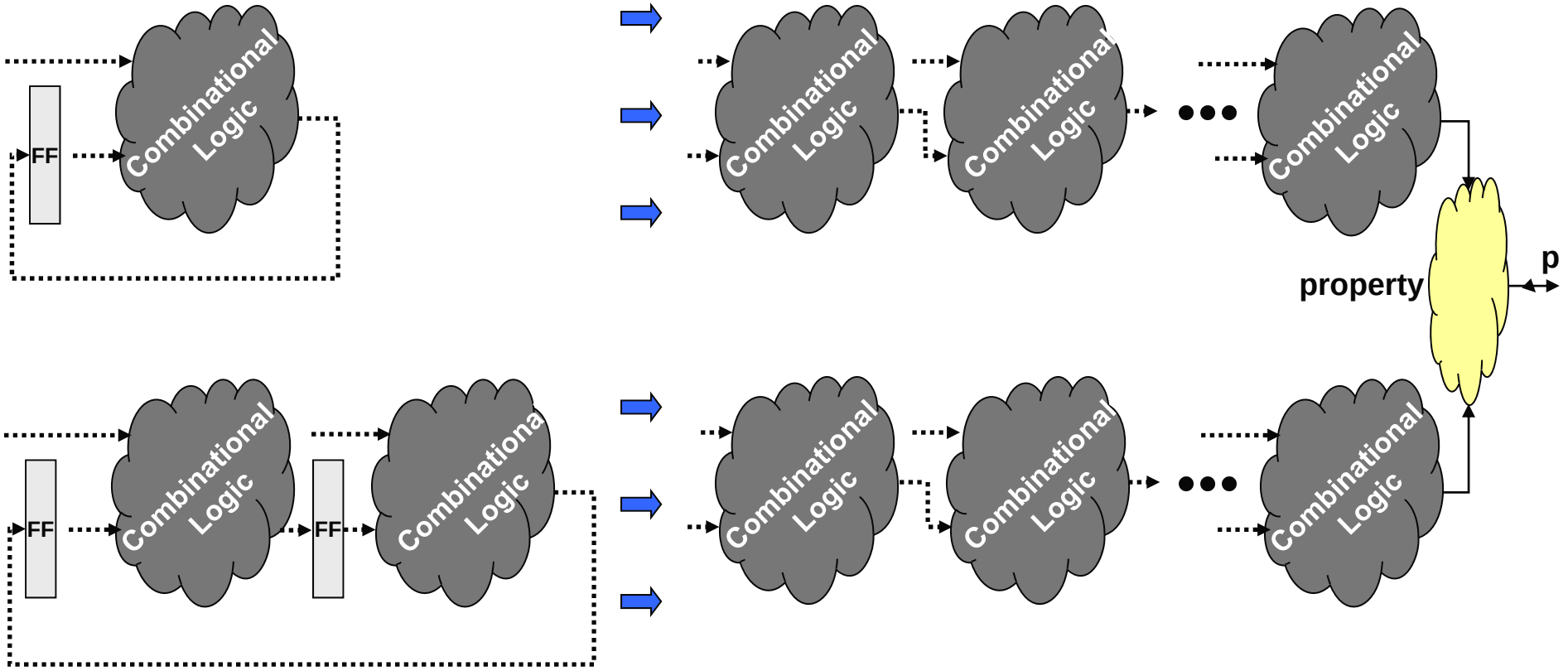

Register Transfer Level
Verilog

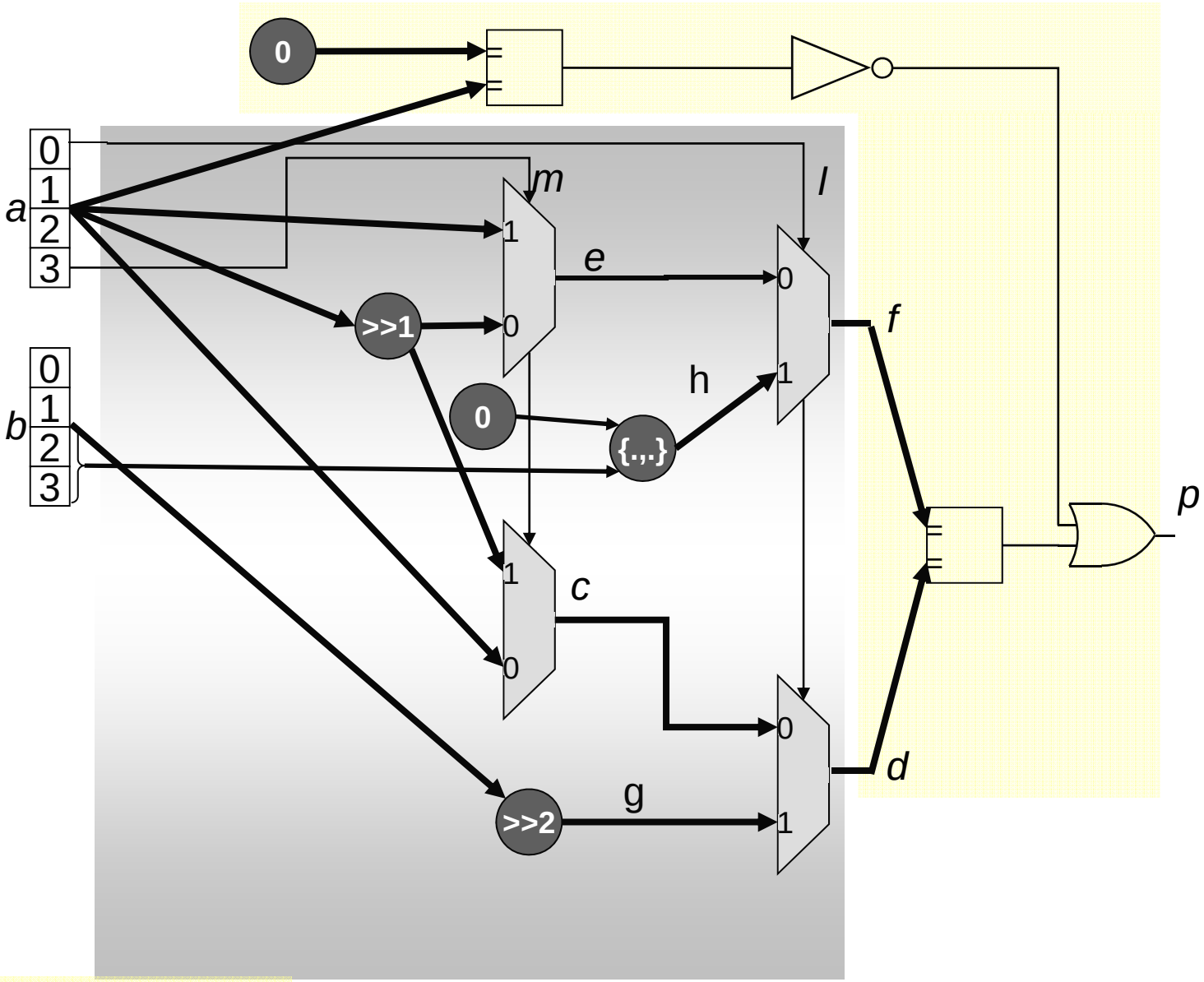
ISA (document)
Non-pipelined Verilog
Transaction C/C++ Module

# Unfolding


FF
Combinational Logic
FF
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
property
p

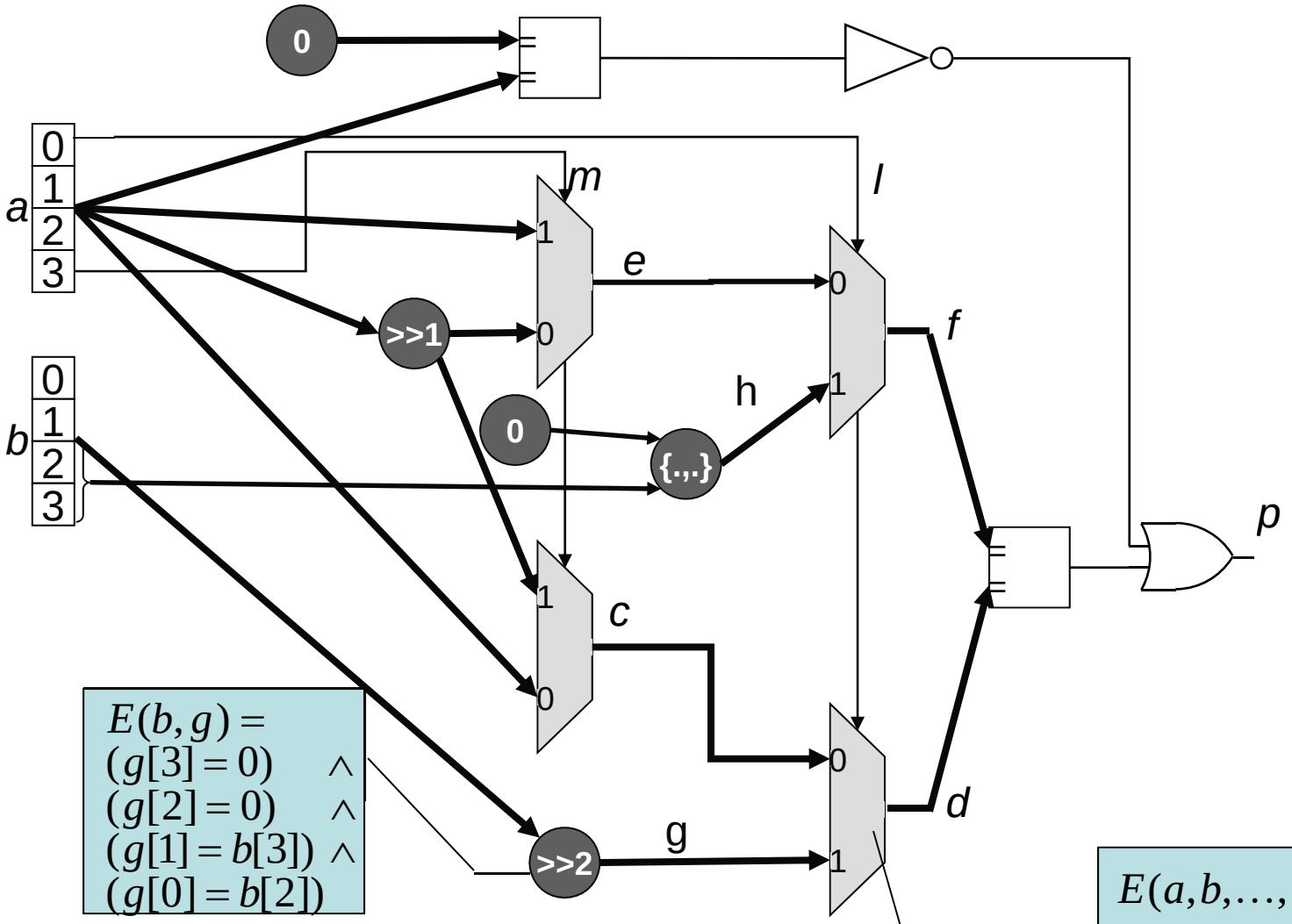
# Unfolding


FF
Combinational Logic
FF
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
Combinational Logic
property
p



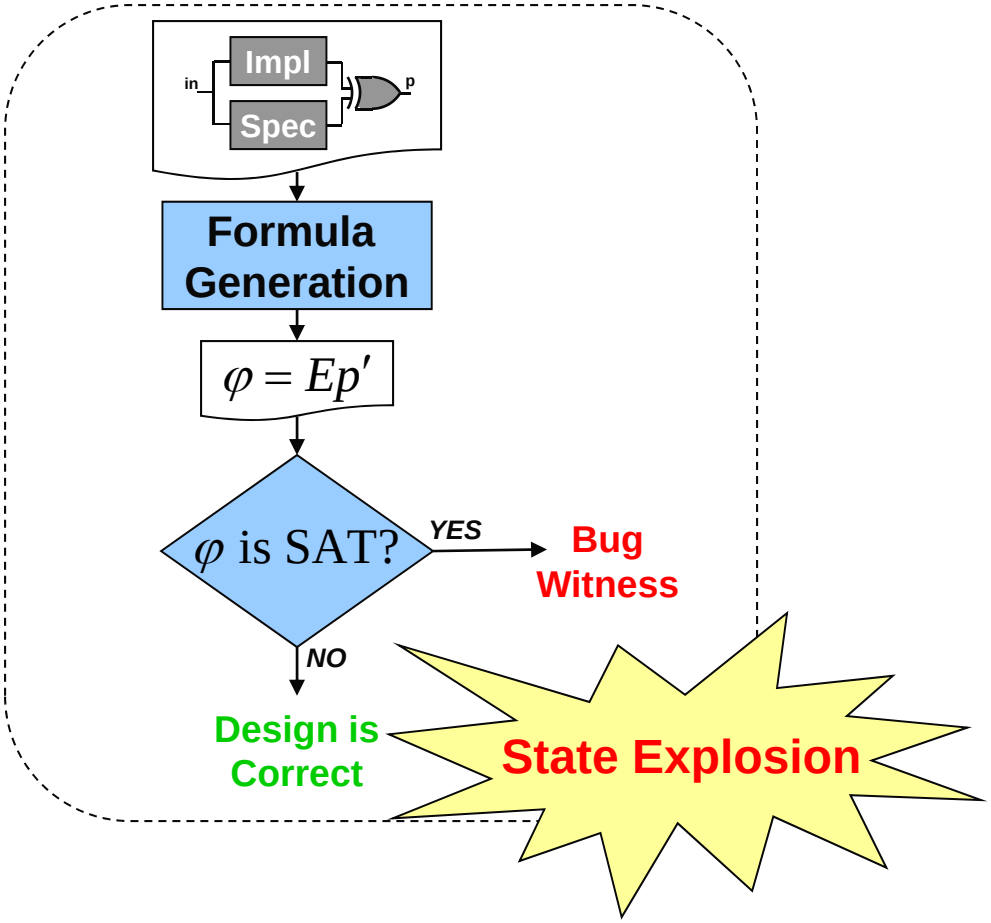

Does the property hold?

Is $p$=1 for all assignments to circuit inputs?




Is $p$=1 for all assignments to circuit inputs? ⇨ Is $E \rightarrow p$=1 for all variable assignments?

# SAT-based Verification



# Approximation

- Replace *exact* behavior of the design with a less precise behavior to speed up verification
  - Compromise Accuracy for Speed

- Sound Approximation

  [Approximation Correct → Design Correct]

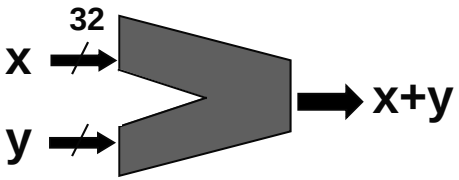
- Complete Approximation

  [Approximation Buggy → Design Buggy]

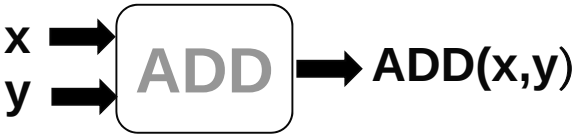
# Outline

- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Conclusions

# Datapath Abstraction



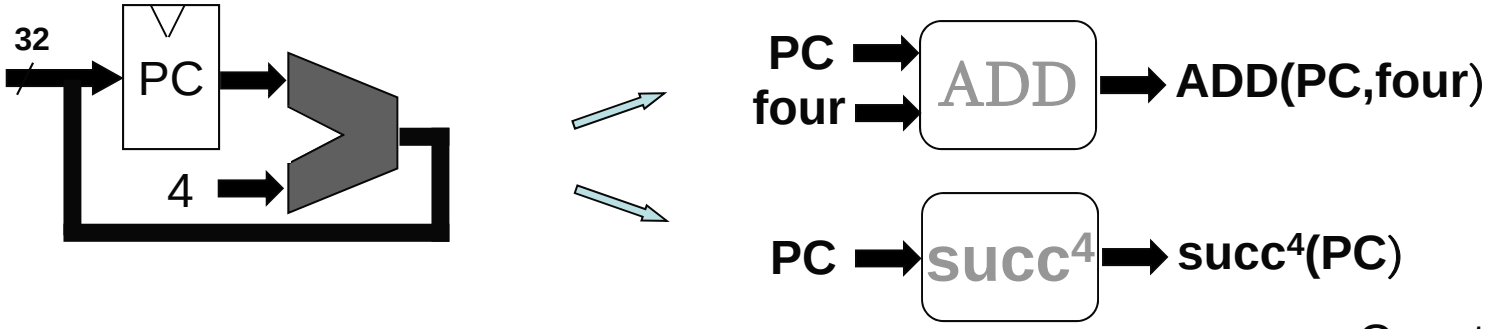

**Adder**




**Un-interpreted Function**

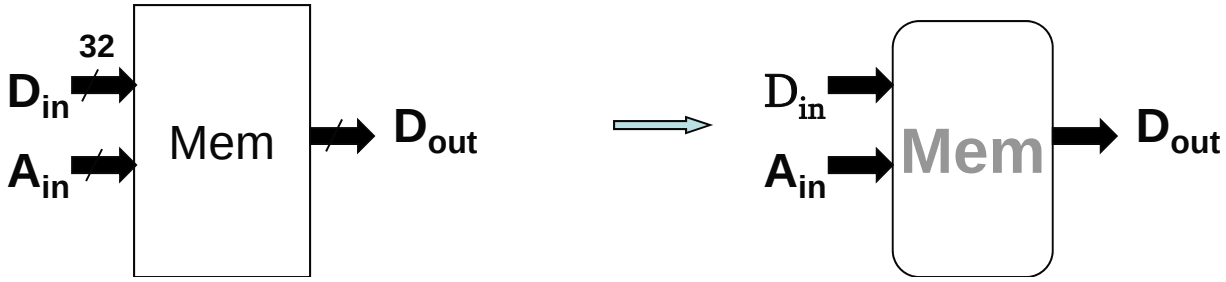
Functional Consistency:

$x_1=x_2$ & $y_1=y_2$ → $ADD(x_1,y_1)=ADD(x_2,y_2)$



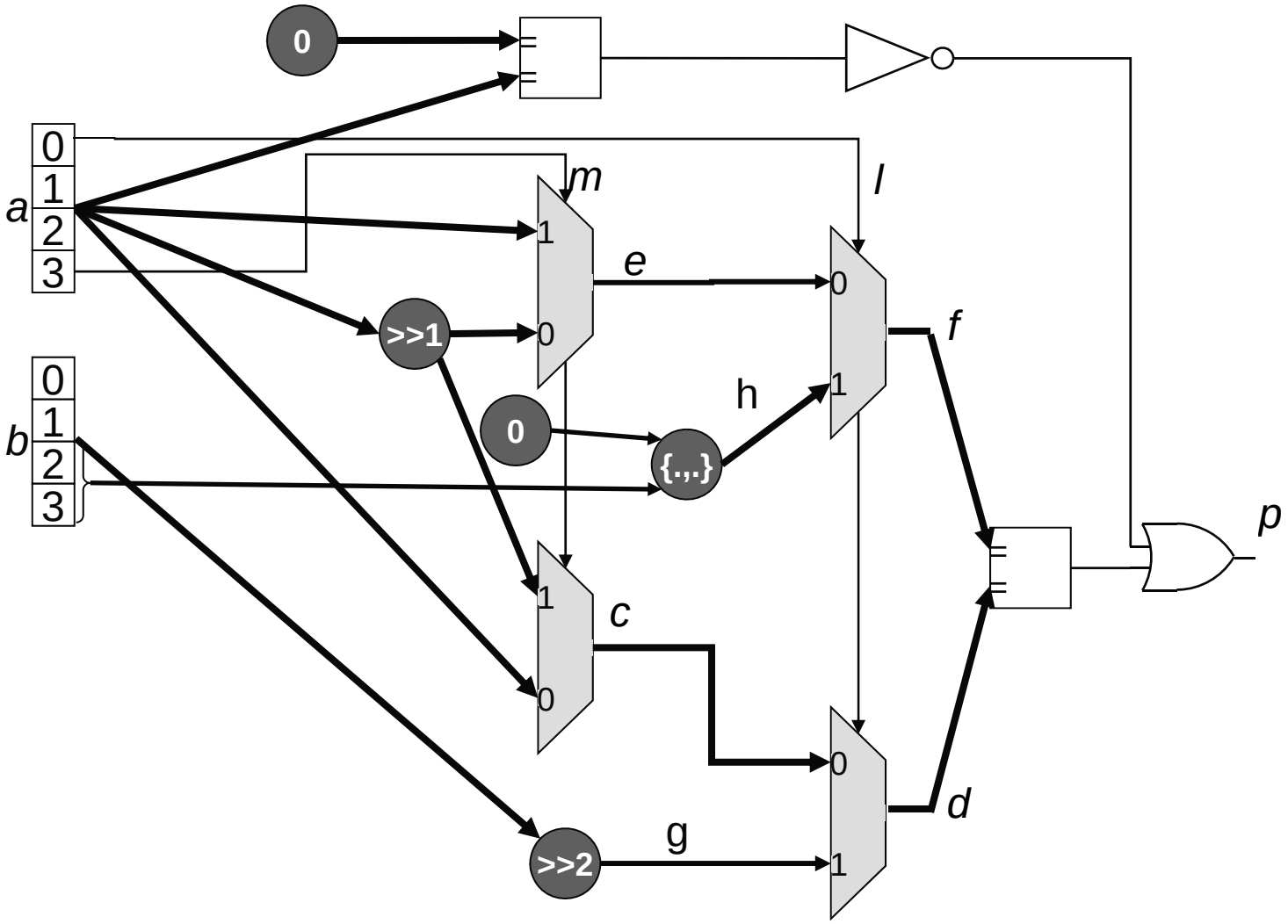

Counting Aritmetic:

**pred(succ (PC))=PC**

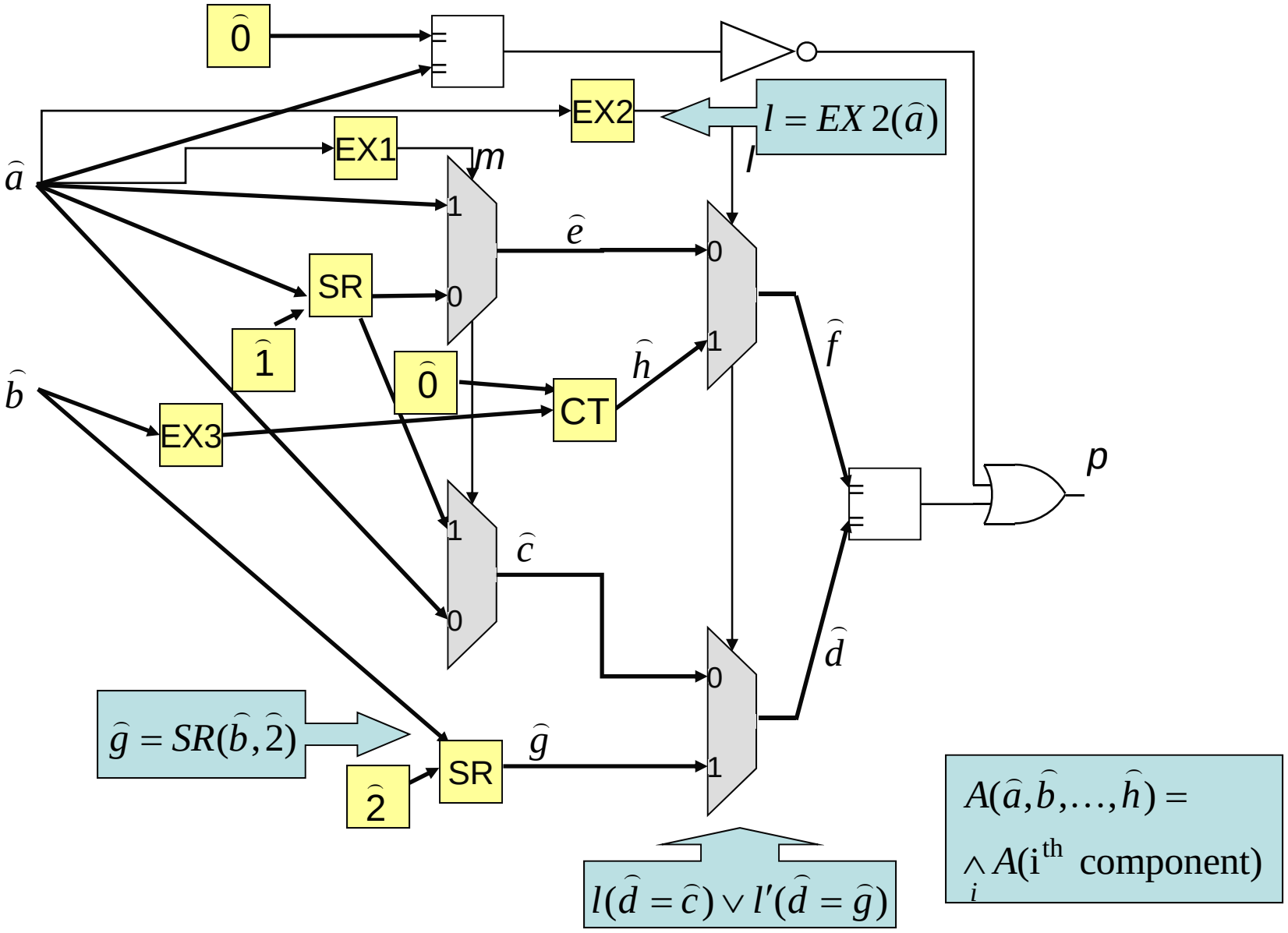



Memory Consistency:

**$D_{out}$ presents the last $D_{in}$ written to same address**

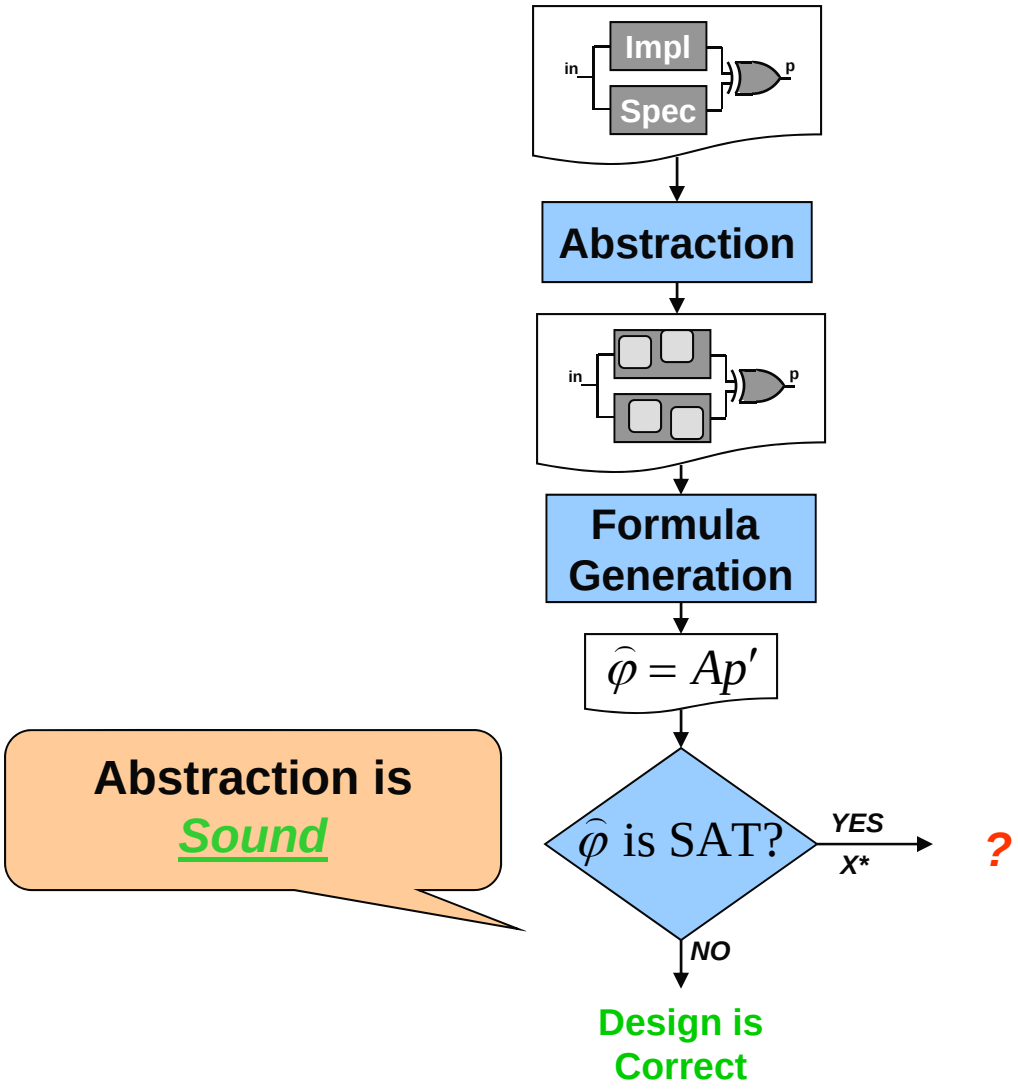


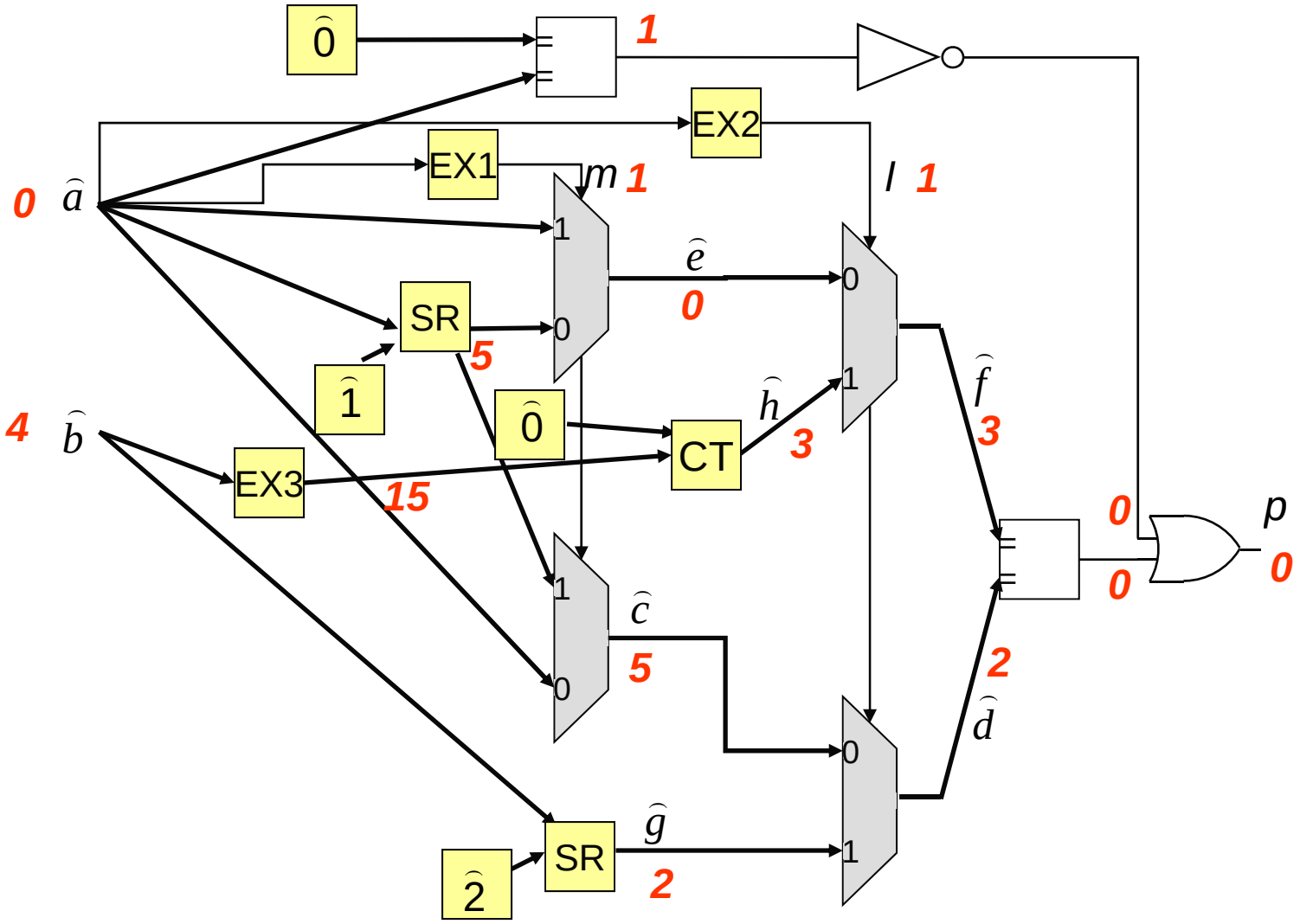

Is $E \rightarrow p=1$ for all variable assignments?



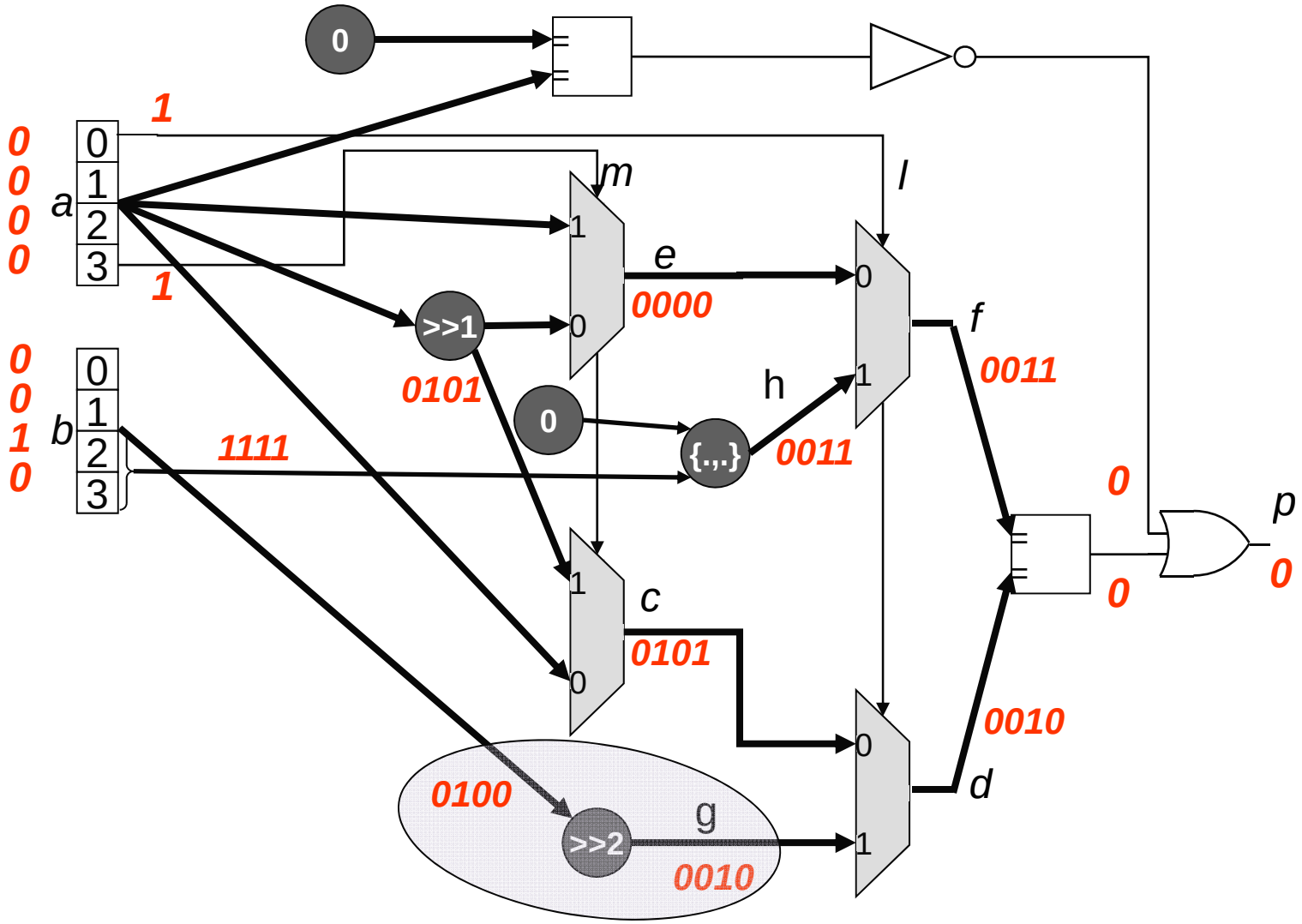

Is $E \rightarrow p=1$ for all variable assignments? ⇨ Is $A \rightarrow p=1$ for all variable assignments?

# Abstraction-based Verification





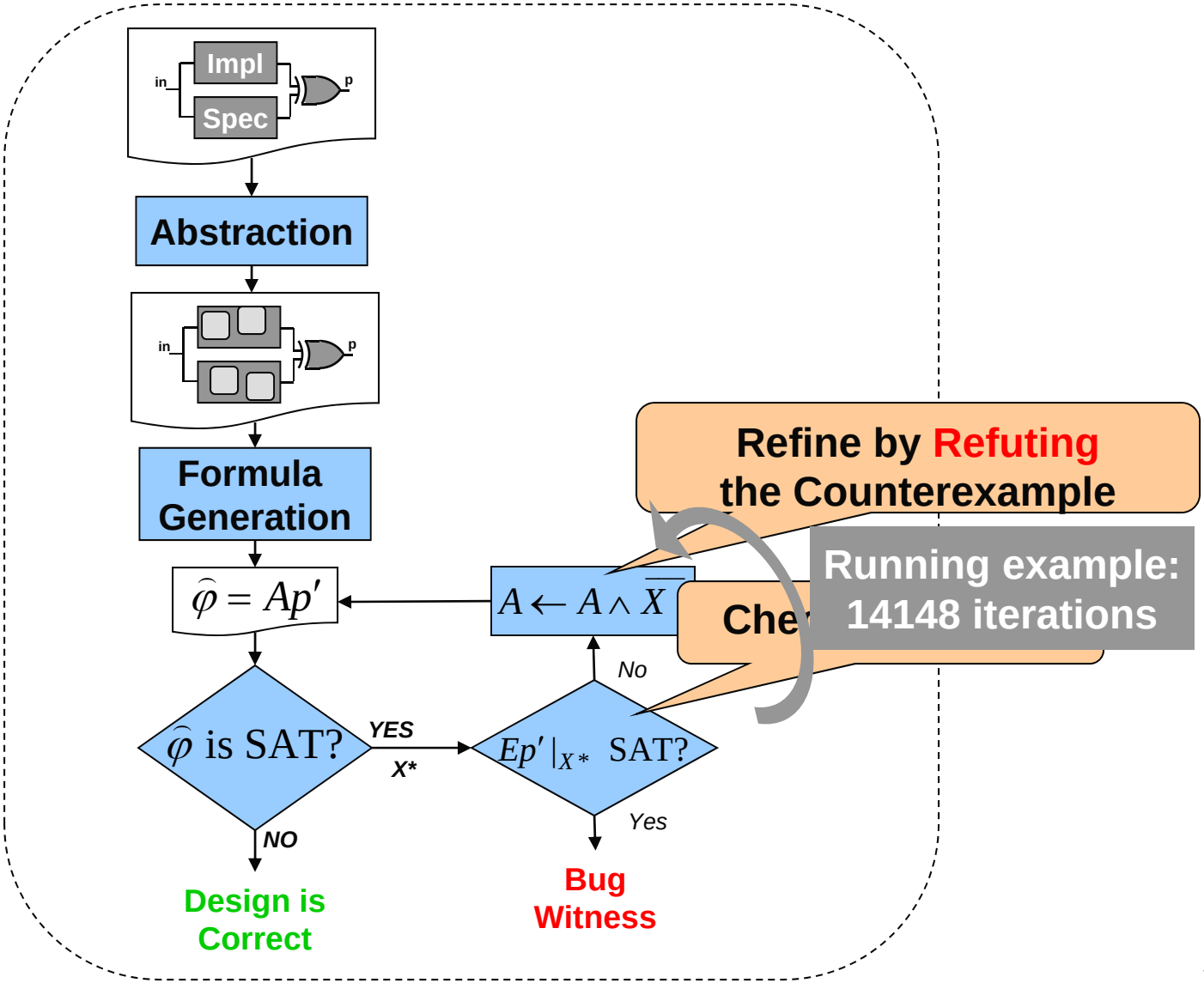

Property is violated on the abstract design $Ap'|_{X*}=1$ ⇨ Is property violated on the concrete design? Is $Ep'|_{X*}=1$?




Violation is inconsistent with concrete design

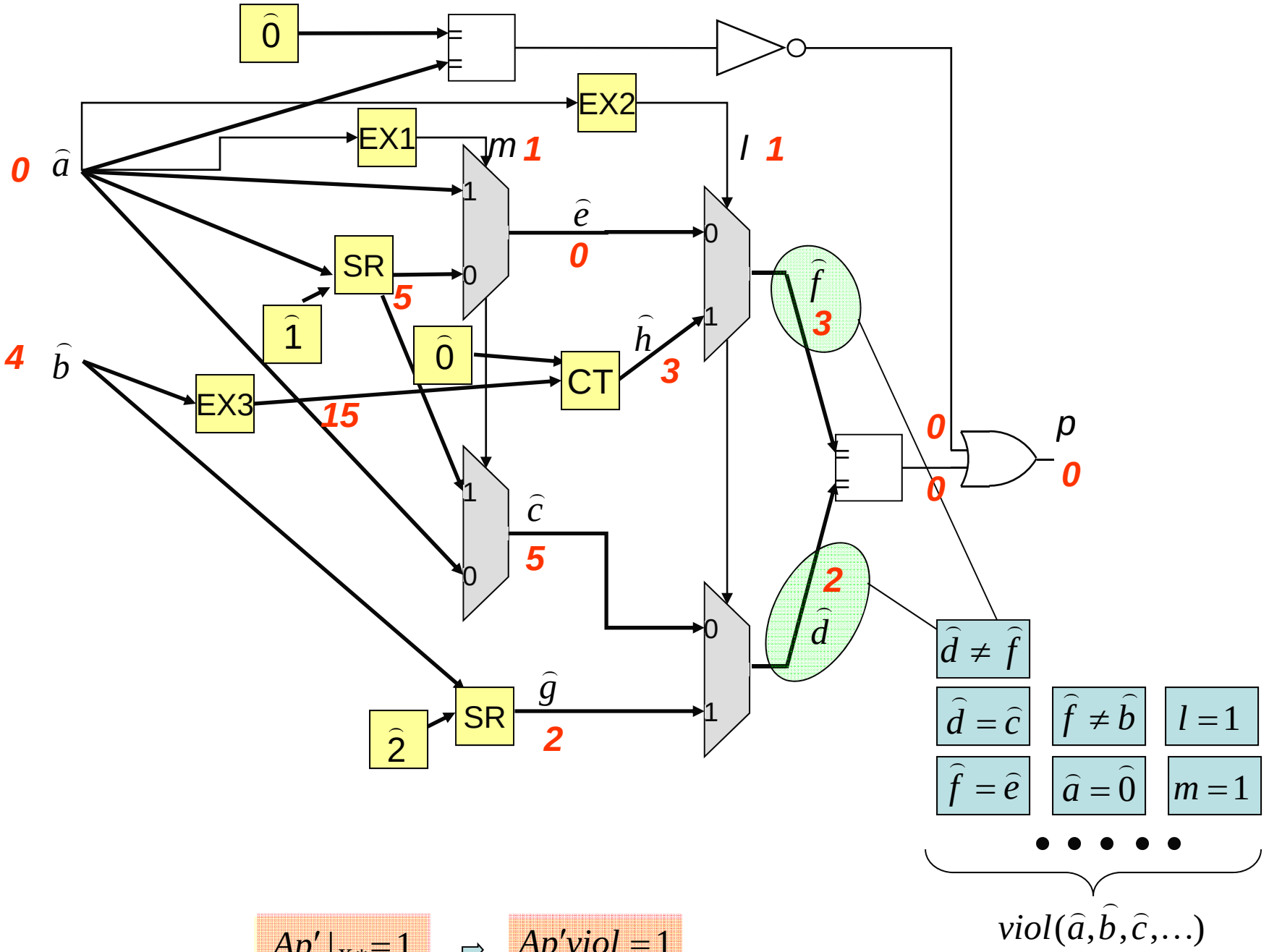
$$Ep'|_{X*}=0$$

# Counterexample Guided Abstraction Refinement



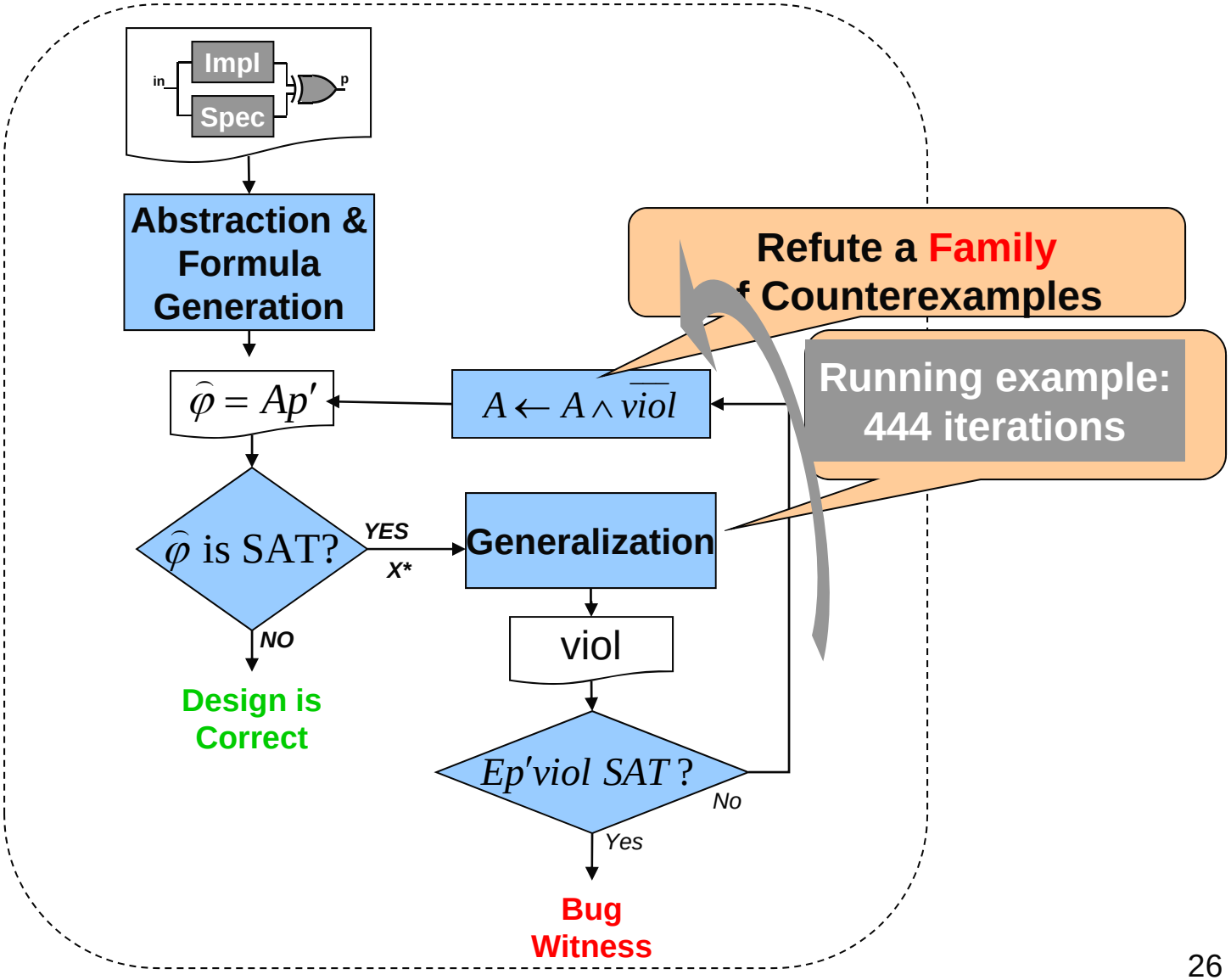
# Outline

- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Conclusions



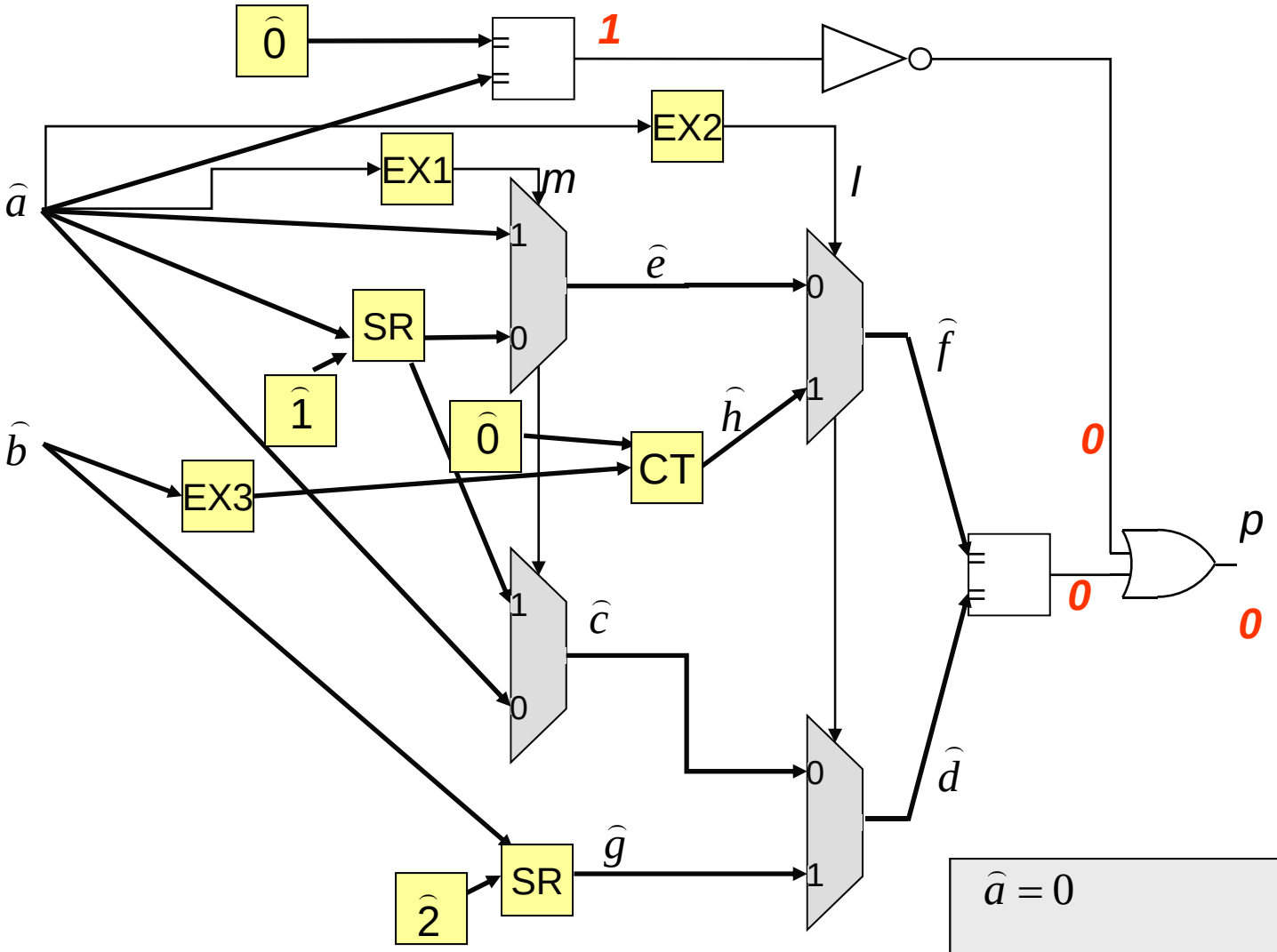

$Ap'|_{X*}=1 \Rightarrow Ap'viol=1$

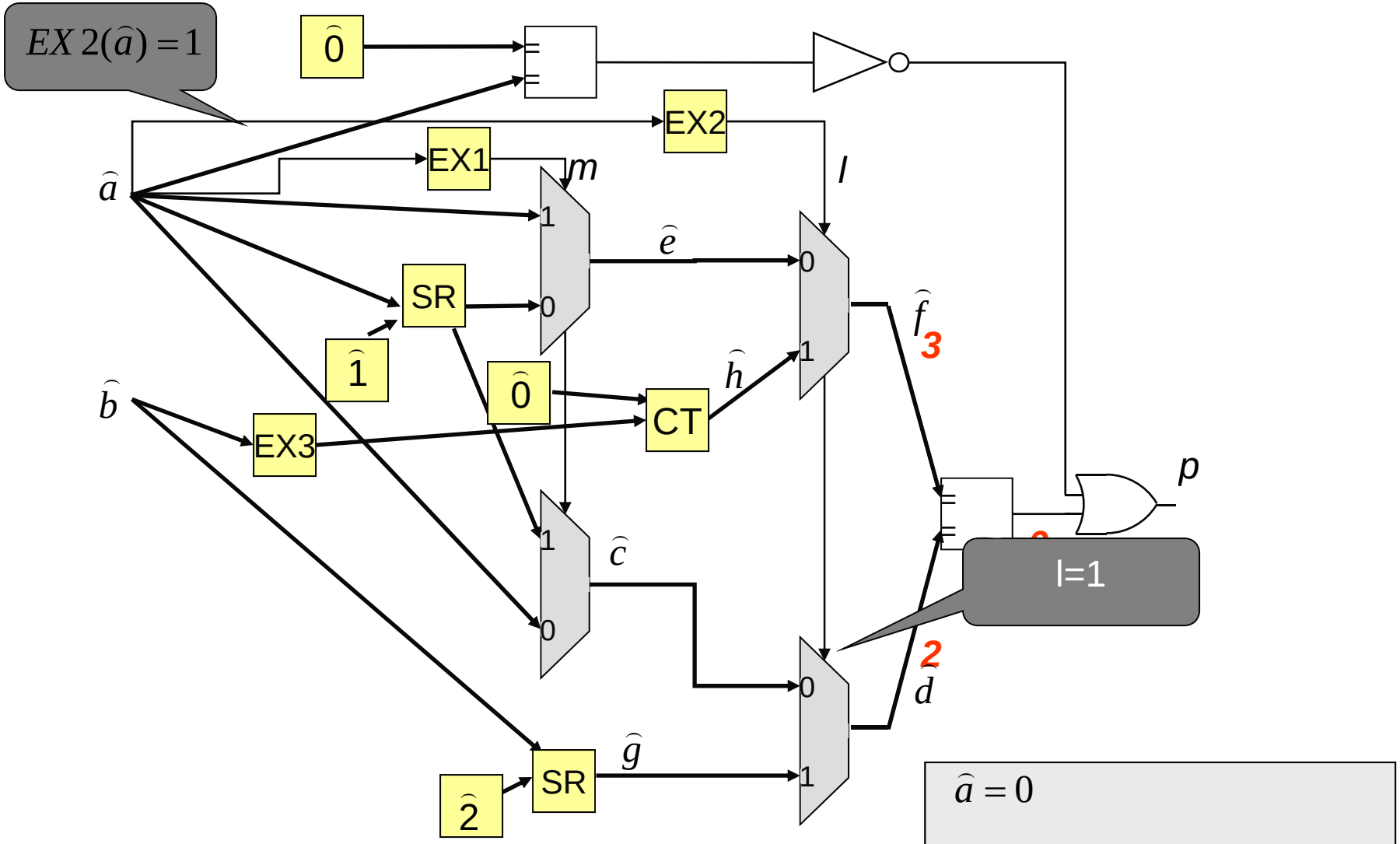
# Counterexample Guided Abstraction Refinement

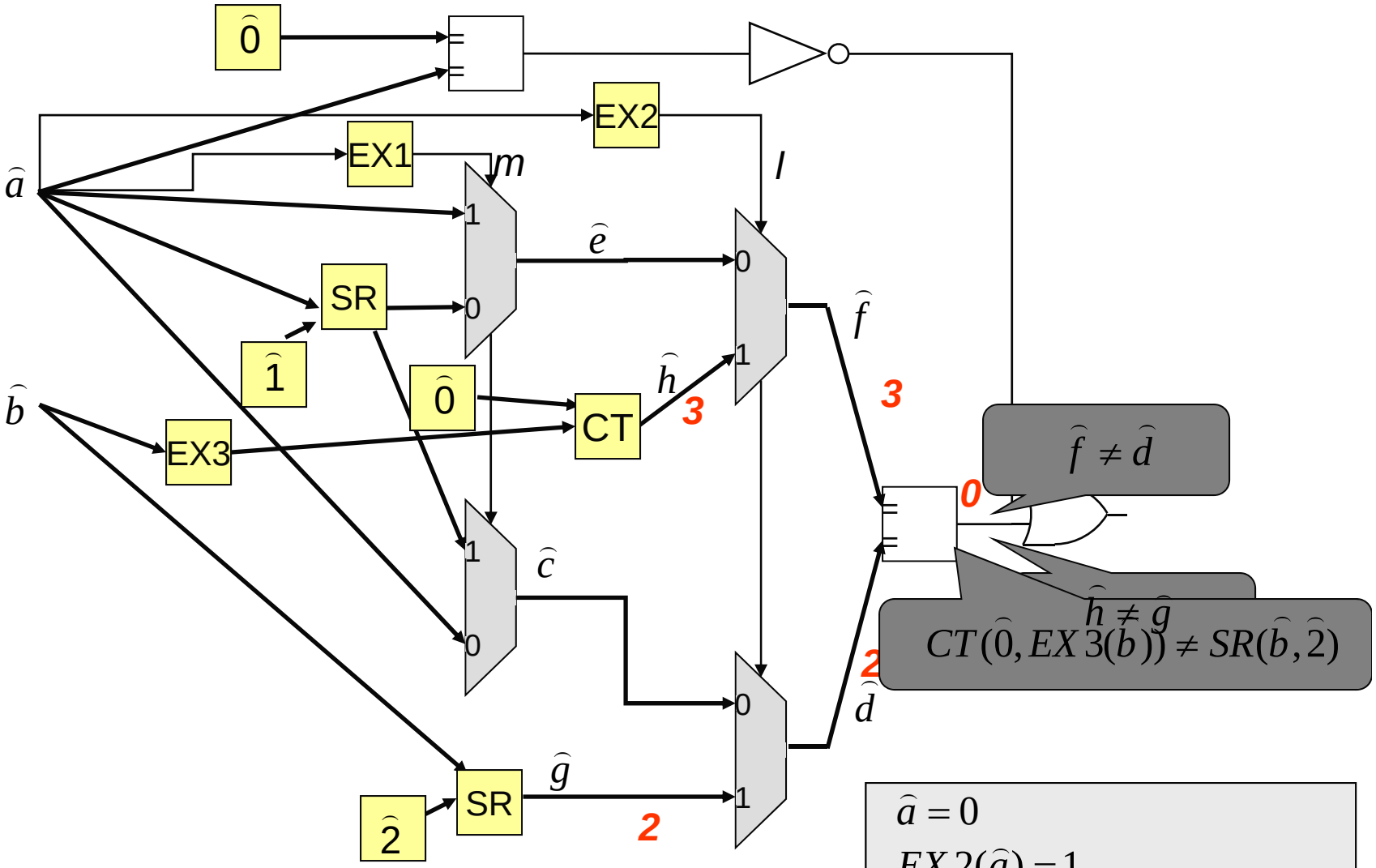


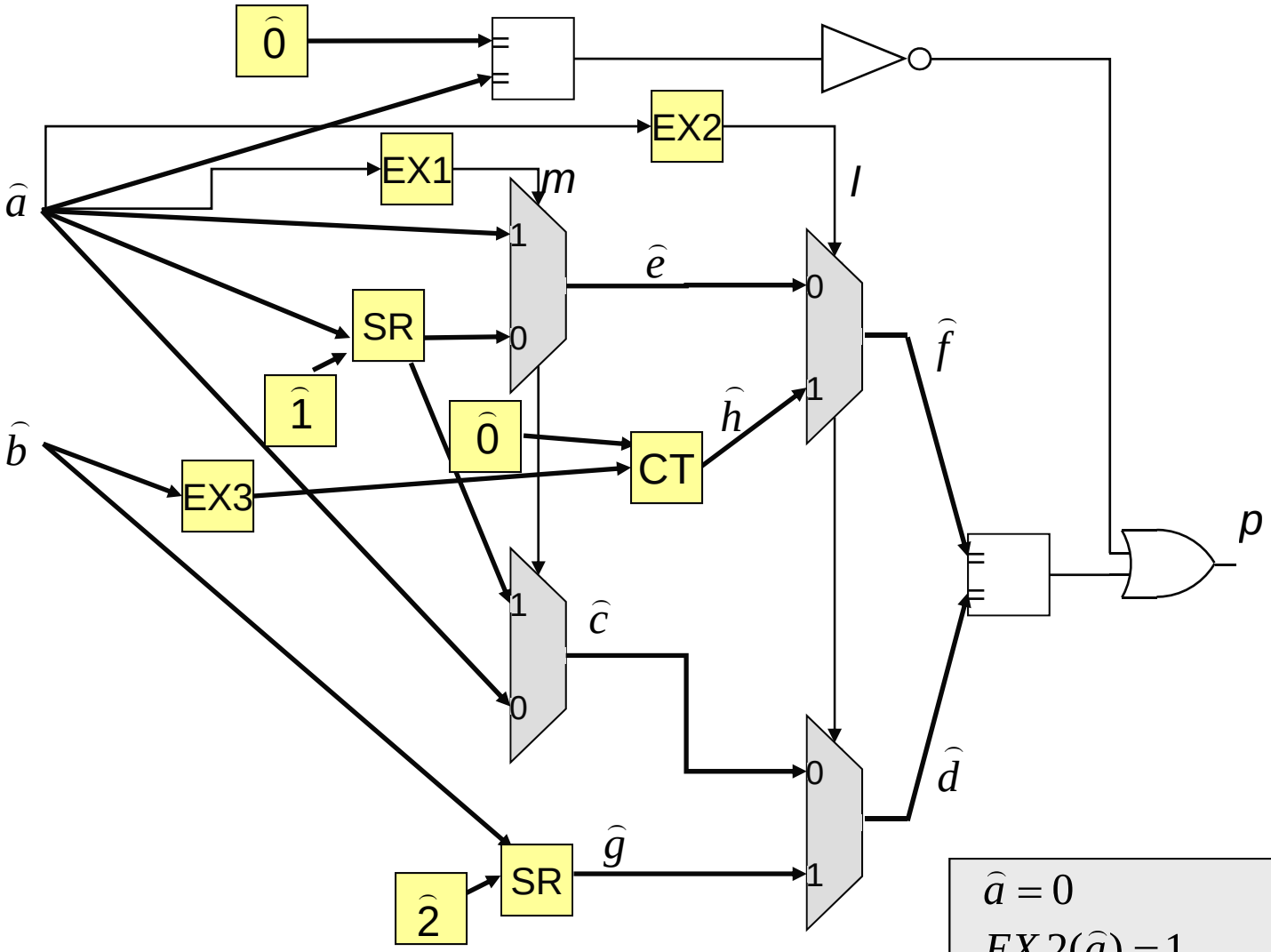
# Generalization

- Pros
  - Resolves the issue of out-of-bound constants
  - Captures and refines a family of counterexamples
- Cons
  - Expensive feasibility check on the circuit itself
  - Many refinement iterations due to unnecessary (dis-) equalities
    - Only cone-of-influence (w.r.t. counterexample) is relevant in each refinement iteration
    - Only a subset of the (dis-)equalities is needed in each iteration
  - A one-time only lemma → cannot be reused


0
1
EX2
EX1
m
l
a
1
e
0
SR
0
1
f
h
1
0
0
b
CT
EX3
p
1
c
0
0
0
d
g
SR
2
a = 0


EX2(â)=1
0̂
EX2
EX1
m
l
â
1
ê
SR
0
f̂3
1̂
ĥ
0̂
b̂
CT
EX3
p
ĉ
l=1
2
d̂
ĝ
SR
2̂
â = 0


0
EX2
EX1
m
l
a
1
e
0
SR
0
f
1
3
1
h
b
0
CT
3
EX3
f ≠ d
0
1
c
h ≠ g
CT(0, EX3(b)) ≠ SR(b, 2)
2
0
d
g
SR
1
2
2
a = 0
EX2(a) = 1




$\hat{a} = 0$

$EX2(\hat{a}) = 1$

$CT(\hat{0}, EX3(\hat{b})) \neq SR(\hat{b}, \hat{2})$

# Feasibility/Refinement based on Explaining the Abstract Counterexample

- Distill a simplified expression
  - Include *only* equalities/dis-equalities
  - Exclude logic that is not in the cone-of-influence
  - Exclude numeric values
  - Exclude Control Logic
  - Based on Primary Inputs

    → allow independent feasibility checking

# Improved Feasibility/Refinement

$$(\hat{a} = 0) \wedge (EX\,2(\hat{a}) = 1) \wedge (CT(\hat{0}, EX\,3(\hat{b})) \neq SR(\hat{b}, \hat{2}))$$

$$\{a_3a_2a_1a_0 = 0000\} \wedge \{a_0 = 1\} \wedge \{00b_3b_2 \neq 00b_3b_2\}$$

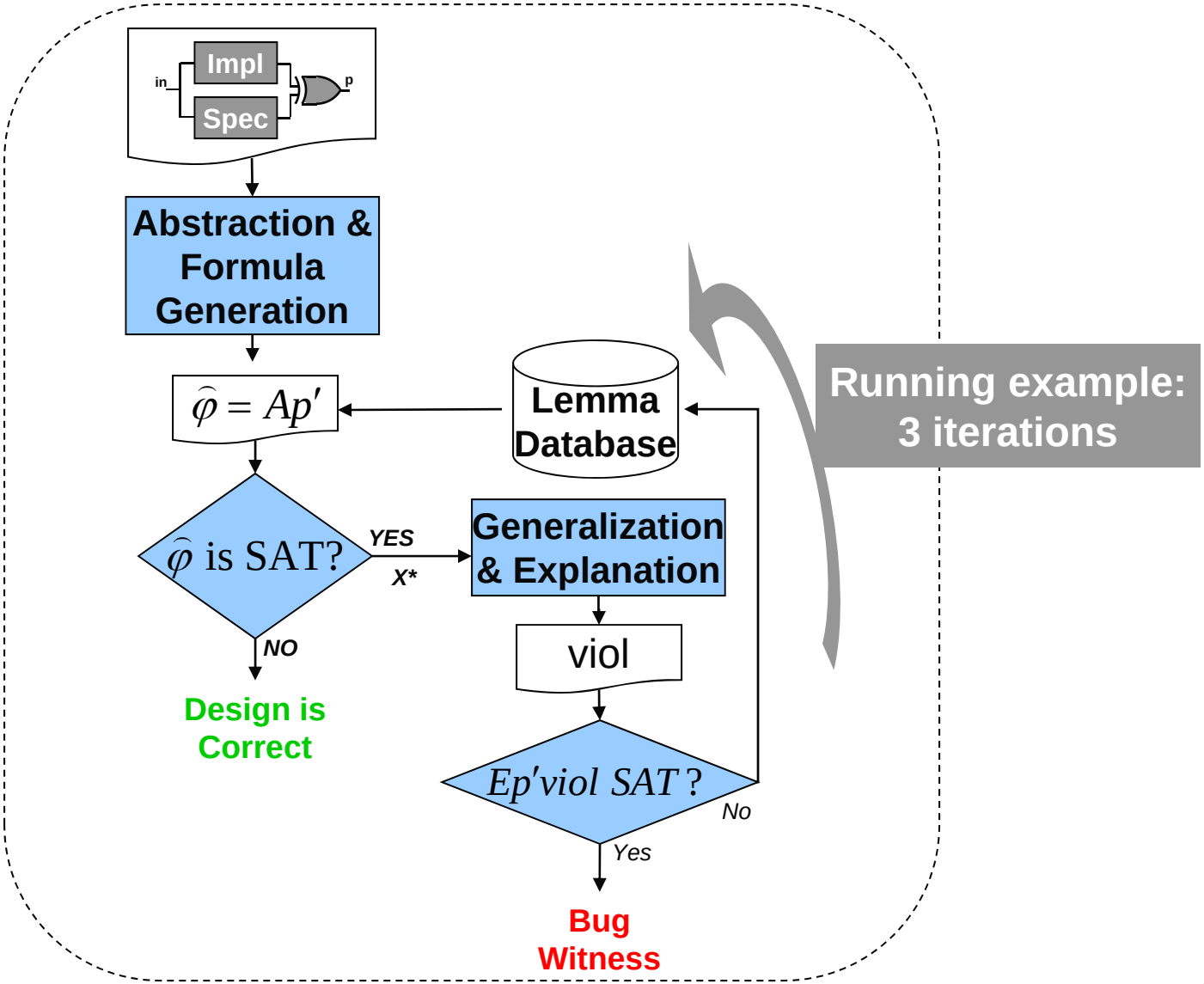
**UNSAT**

# Improved Feasibility/Refinement

$$\neg([\hat{a} = 0] \wedge [EX\,2(\hat{a}) = 1] \wedge [CT(\hat{0}, EX\,3(\hat{b})) \neq SR(\hat{b}, \hat{2})])$$

- Lemma: A high-level universal truth
  - Refutes a family of spurious counterexamples
  - Can be reused whenever relevant
    - Across refinement iterations
    - Across various invocations of the verification on modifications of the design/property

# Refinement based on Lemmas



# Minimization

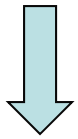
$$\{a_3a_2a_1a_0 = 0000\} \wedge \{a_0 = 1\} \wedge \{00b_3b_2 \neq 00b_3b_2\}$$

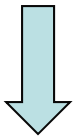
$$\{a_3a_2a_1a_0 = 0000\} \wedge \{a_0 = 1\}$$

**(UNSAT)**

$$\{00b_3b_2 \neq 00b_3b_2\}$$

**(UNSAT)**





$$\neg[(\hat{a} = 0) \wedge (EX\,2(\hat{a}) = 1)]$$

**(Lemma)**

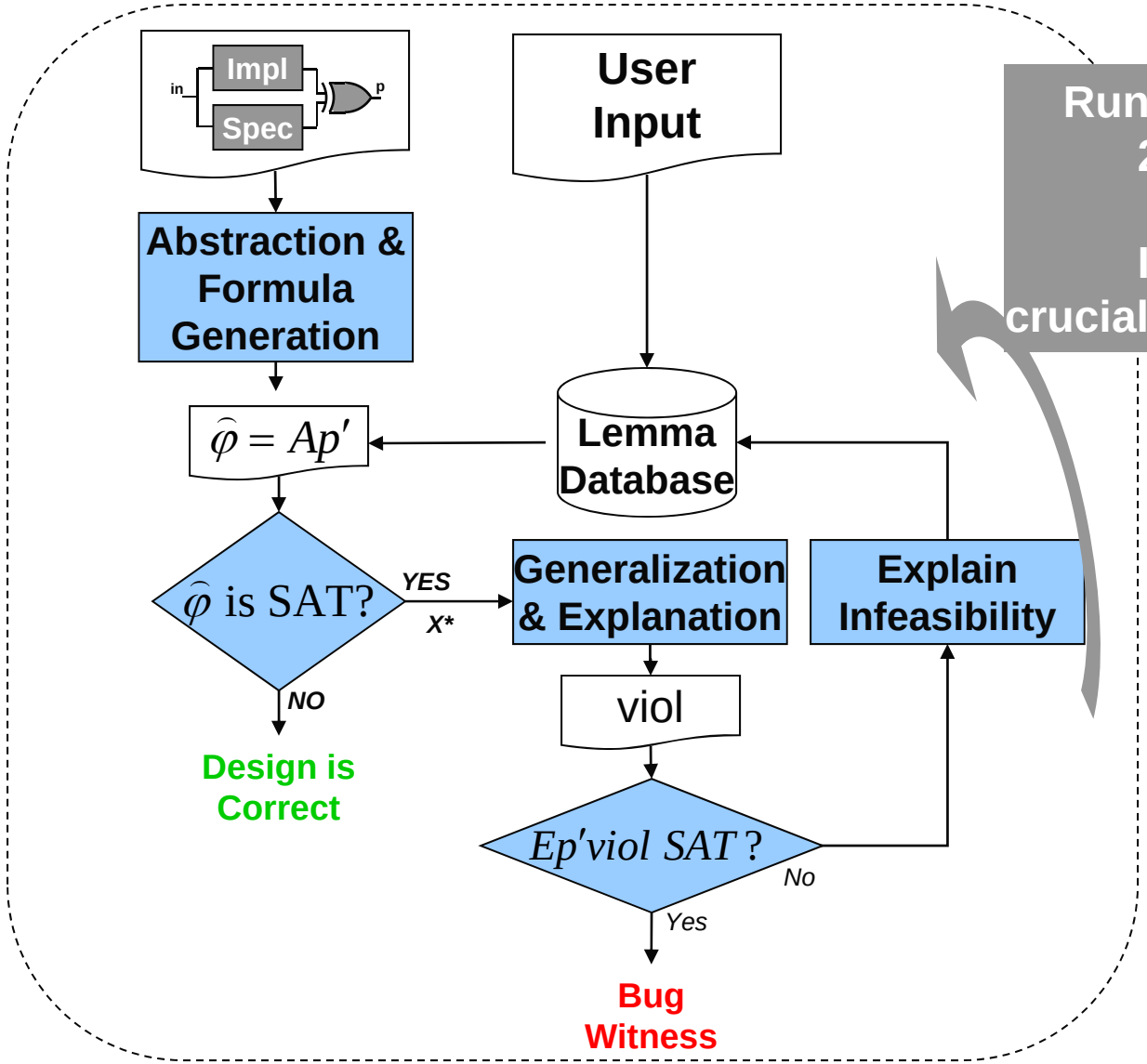
$$\neg(CT(\hat{0}, EX\,3(\hat{b})) \neq SR(\hat{b}, \hat{2}))$$

**(Lemma)**

An All-MUS algorithm can generate lemmas

- As many as possible
- As compact as possible

# Refinement based on Lemmas



# Trade-Offs

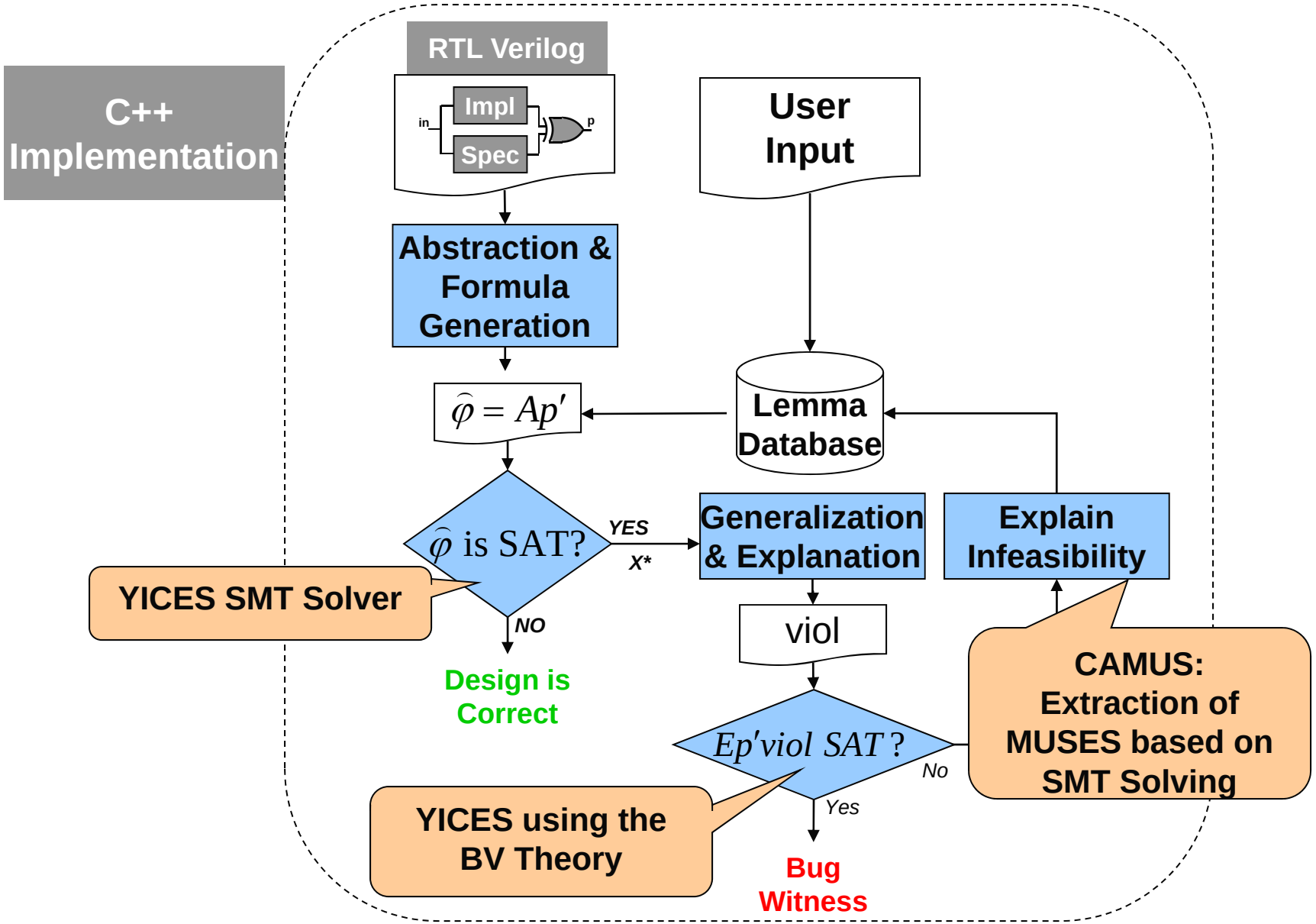
- Efficiency in each Step
  - Validity Check
  - Feasibility Check
  - Refinement
- Preciseness of Initial abstraction
  - Precise (detailed) versus Impresice (coarse)
- Refinement Convergence

➔ Overall Performance

# Outline

- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Conclusions

# The Reveal System



# Outline

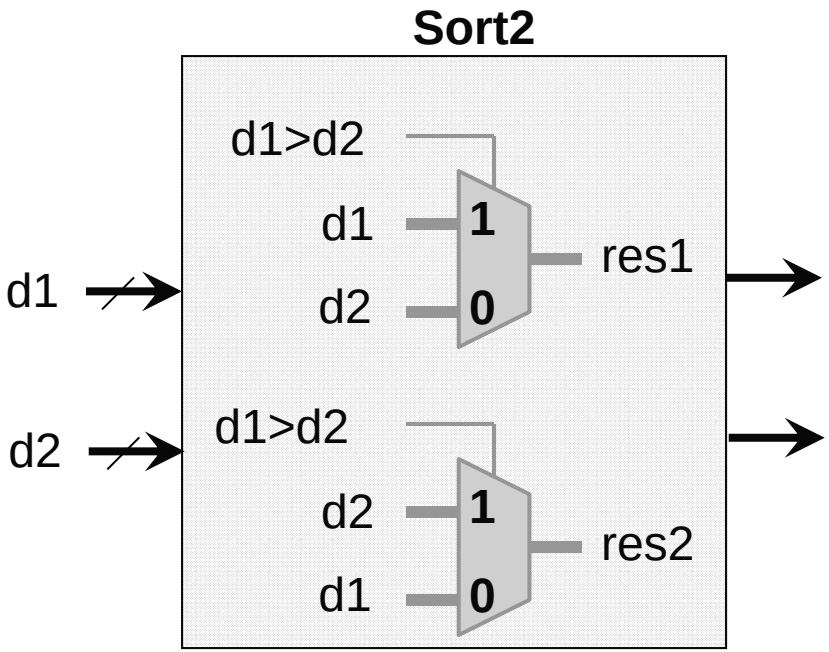
- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Conclusions

# Test Cases and Setup

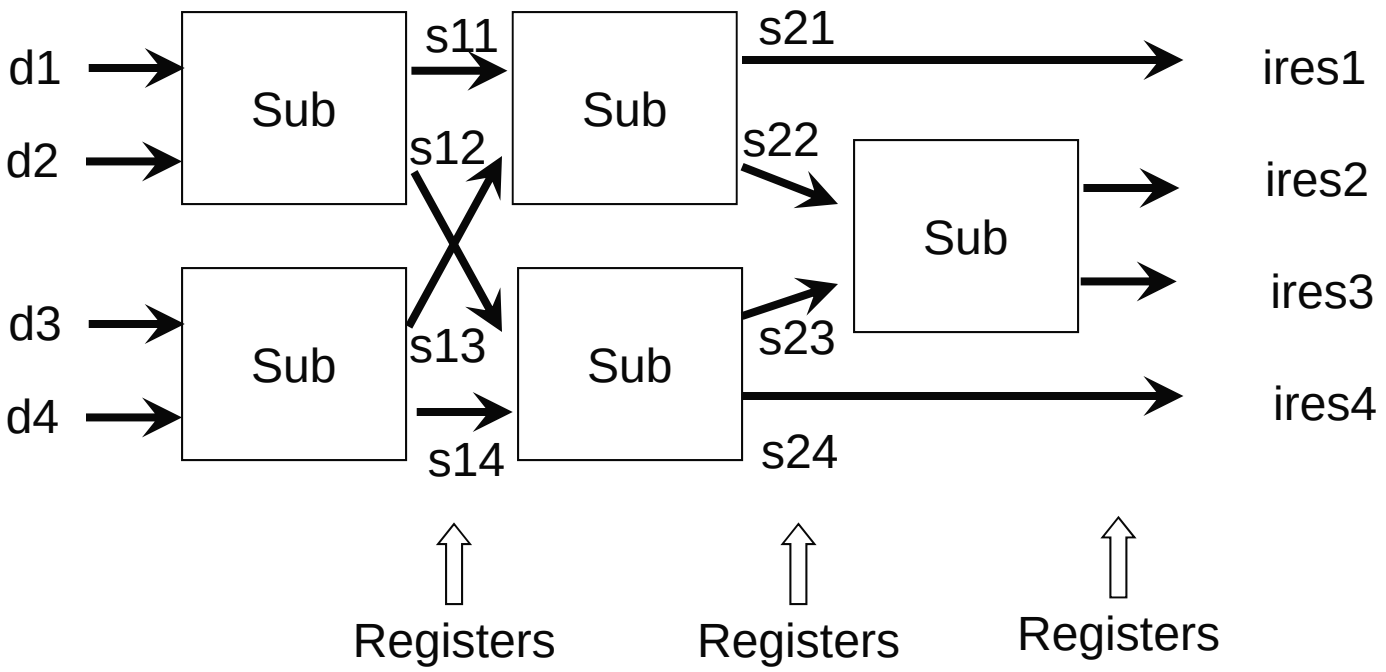
| Name | Verilog Lines | Verilog Signals | State Bits |
|---|---|---|---|
| Sorter | 79 | 30 | 35 to $1x10^3$ |
| DLX | $2.4x10^3$ | 399 | $1x10^{11}$ |
| Risc16F84 | $1.2x10^3$ | 169 | $1x10^5$ |
| X86 | $1.3x10^4$ | $1x10^3$ | $5.8x10^3$ |

Setup: 2.2 GHz AMD Opteron Processor, 8GB RAM, Linux

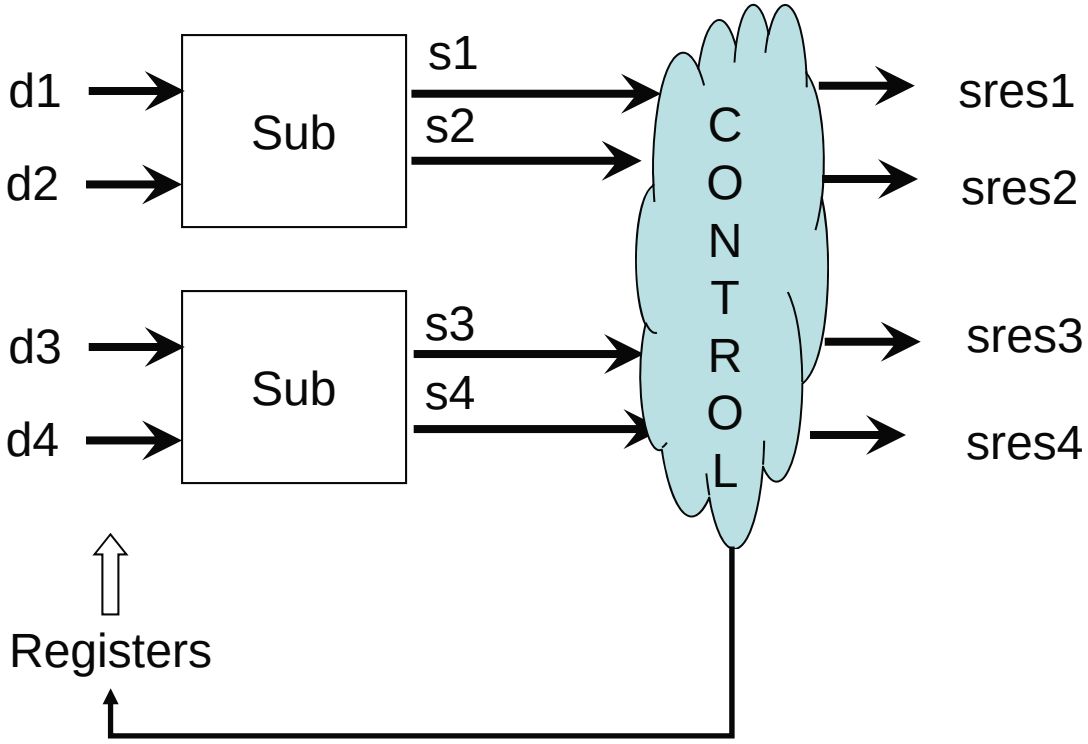
# Test Case1: Sorter



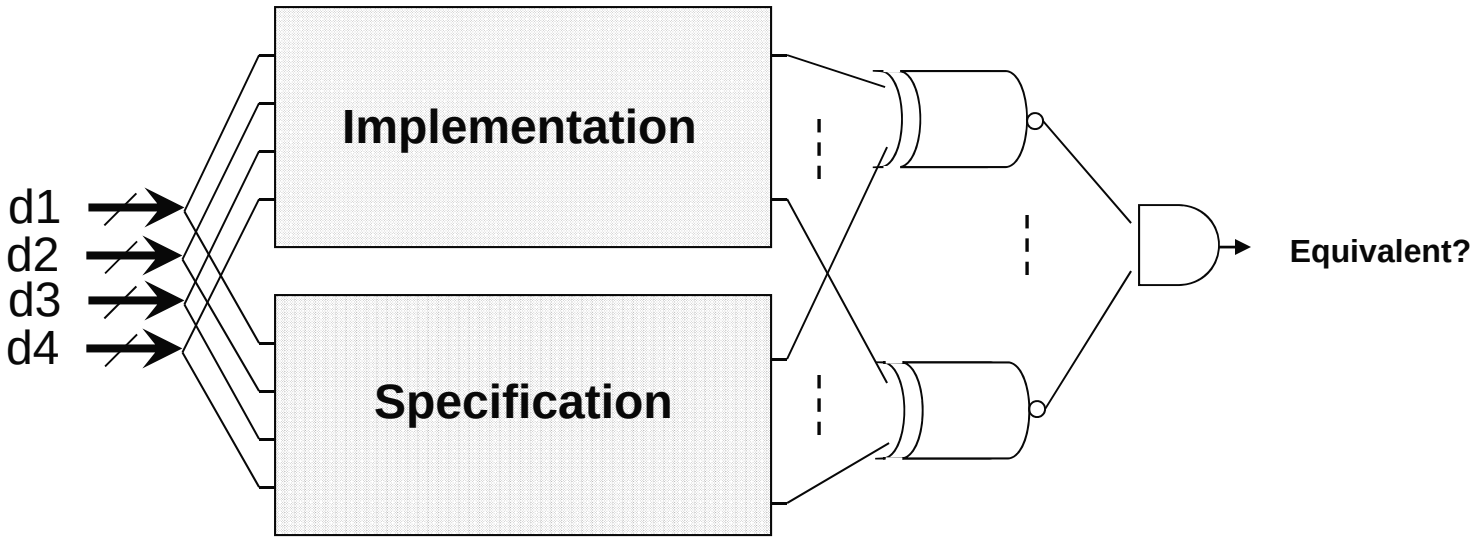
# Sorter: Implementation



# Sorter: Specification



# Sorter Equivalence



# Test Case 1: Sorter

- A Sort Algorithm for 4 *W*-bits vectors
- Equivalence between 2 Implementations

# Test Case2: RISC16F84

- A Microcontroller from OpenCores.org
- $2^{13}$x14-bit I-Mem, $2^{9}$x8-bit D-Mem
- 34 Op-codes
- 4-stage pipeline
- Property: Equivalence to a 1-stage Spec starting from an arbitrary state:

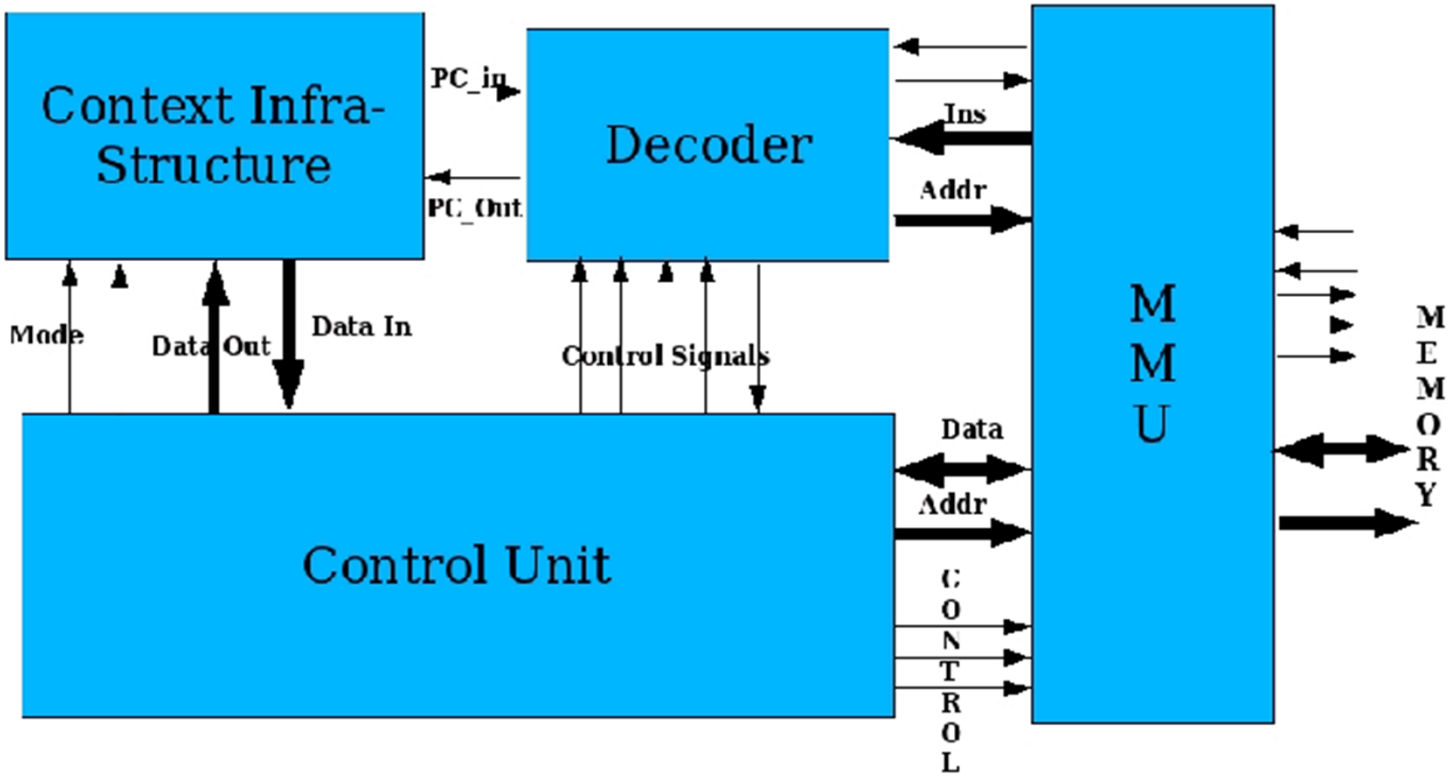
$$\bigwedge_j (I_0^j = S_0^j) \rightarrow \bigwedge_j (I_4^j = S_1^j)$$

# Test Case3: DLX

- DLX is a 32-bit RISC microprocessor*
- 32-bit address space, I-memory, D-memory
- 32-word register file, 2 read ports, 1 write port
- 38 op-codes
- 5-stage Pipeline
- Property: Pipeline to Non-pipeline Equivalence Starting from Reset:

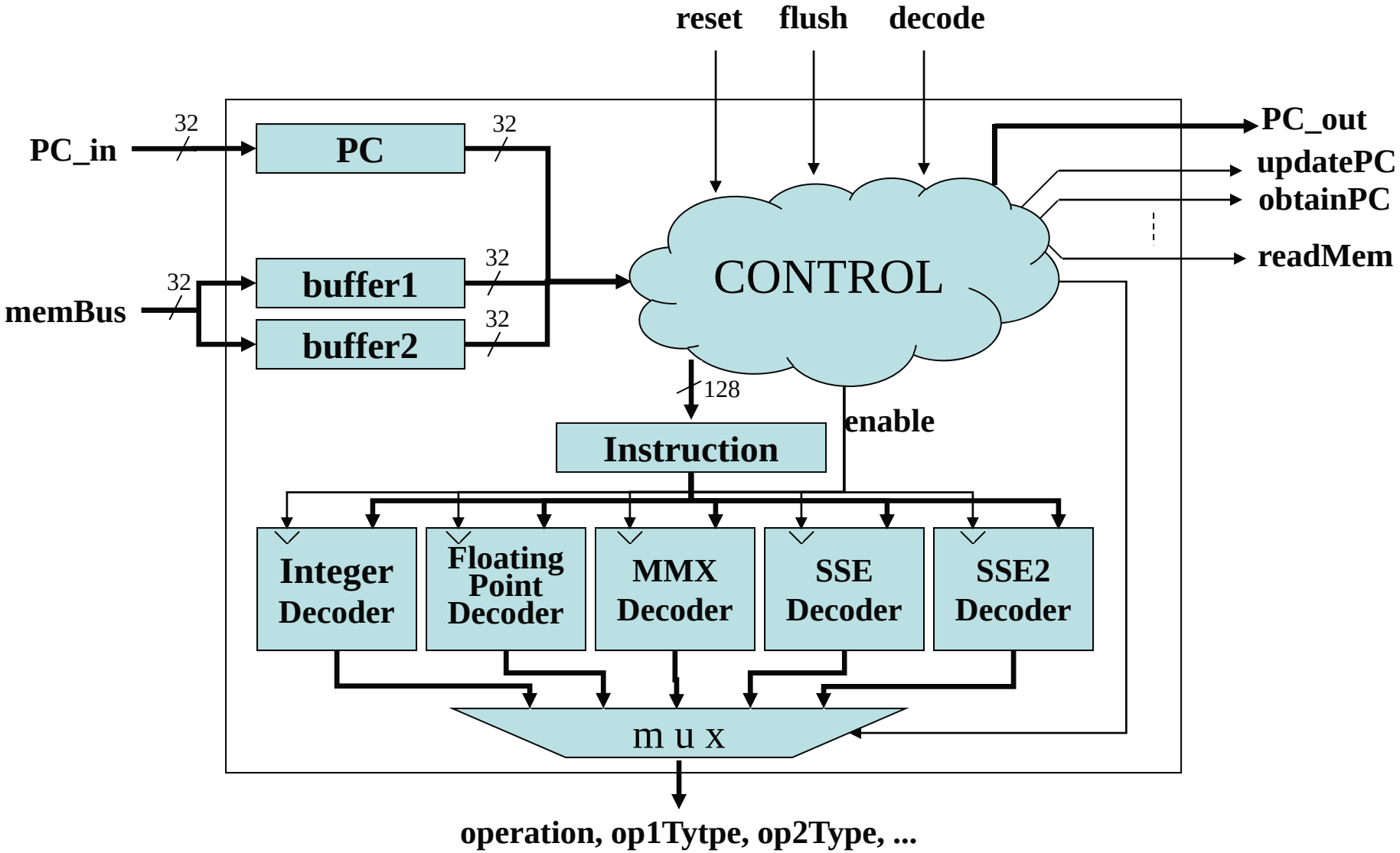
$$(E_1^S = E_1^I) \vee (E_1^S = E_2^I) \vee \ldots \vee (E_1^S = E_5^I)$$

# Test Case4: X86*

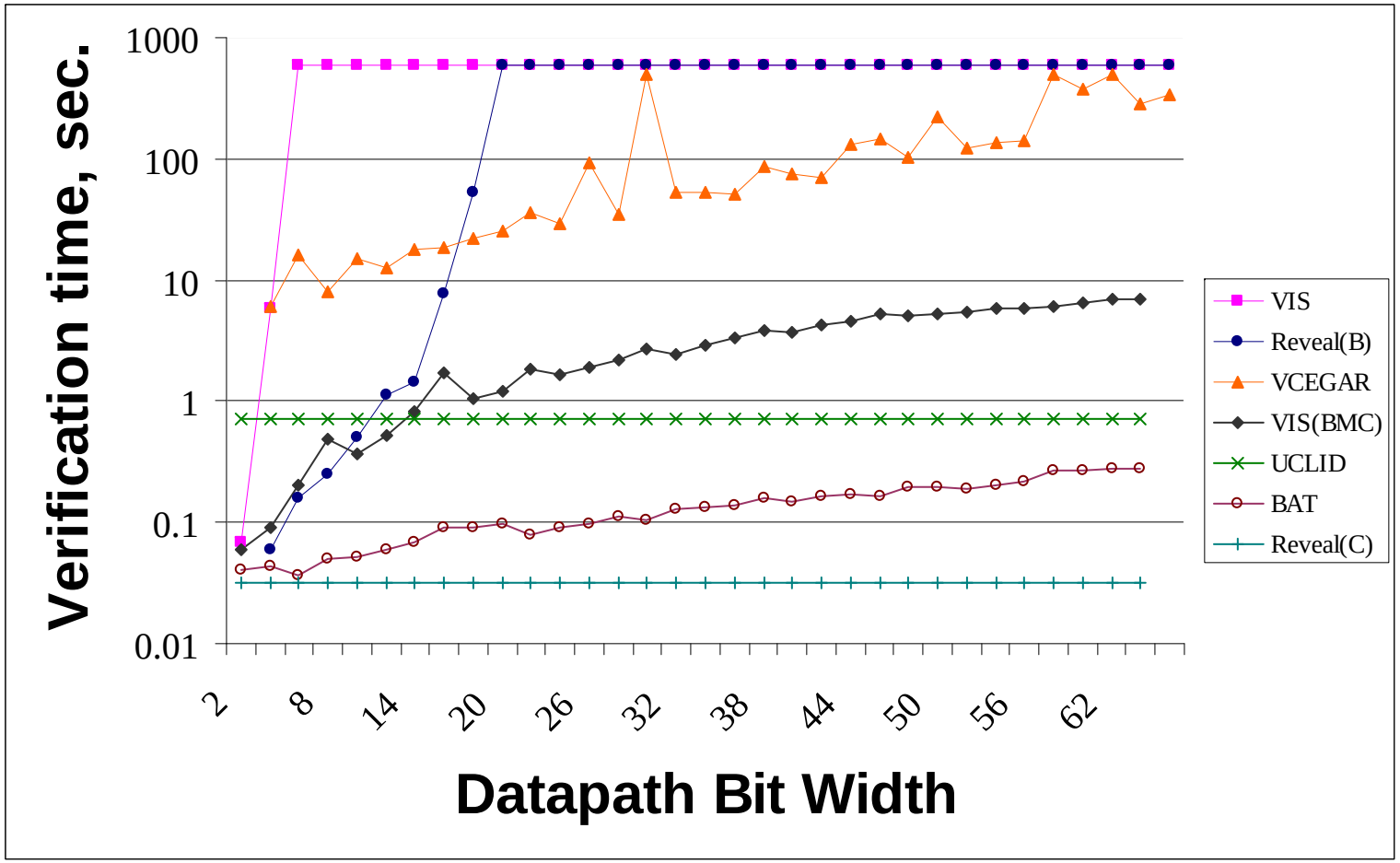



* http://vlsi.cs.iitm.ernet.in/x86_proj/x86HomePage.html

# IA-32 Decoder Unit



# Runtime Comparison on Sorter



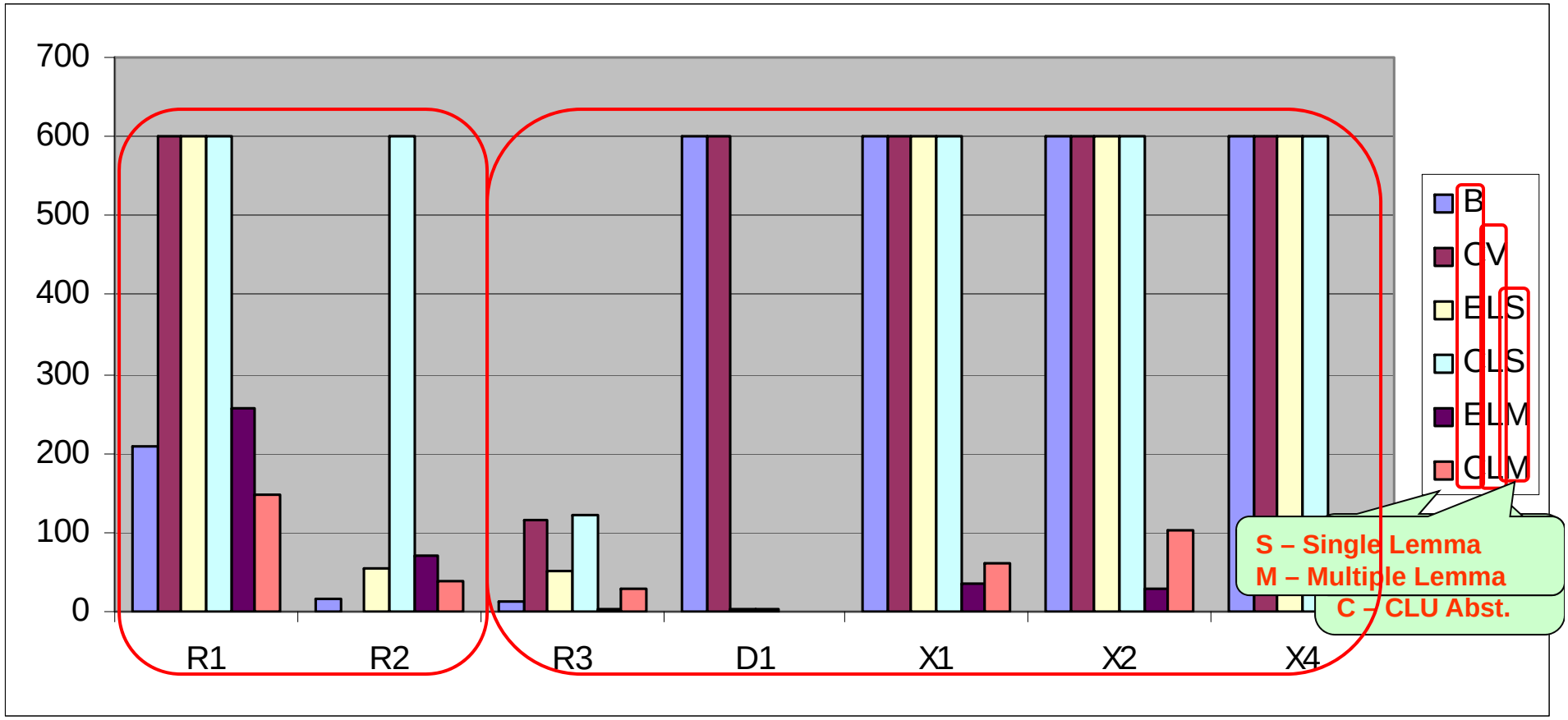
# Variations of Test Cases 2, 3, 4

| | |
|---|---|
| D1 | Bug-free $DLX_{Spec}$ and $DLX_{Impl}$ |

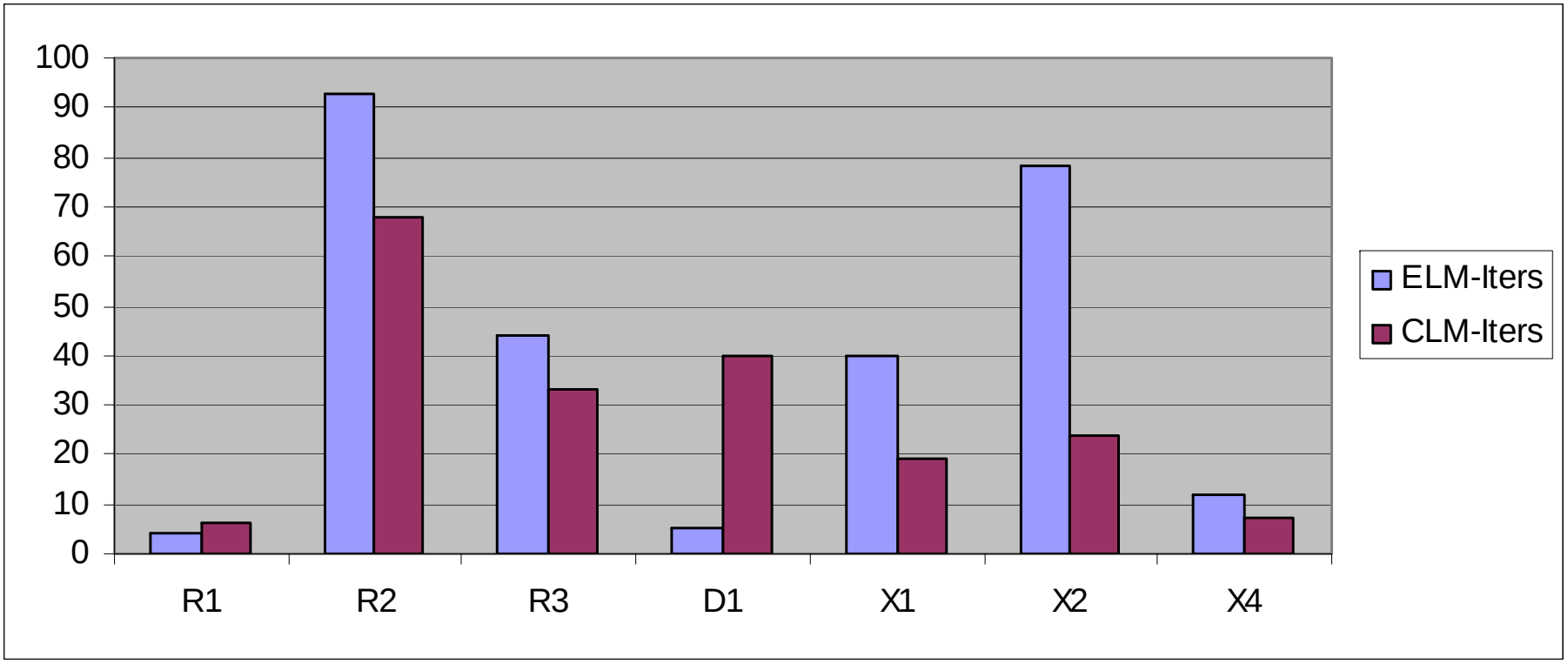
| | |
|---|---|
| R1 | Bug-free $OC_{Spec}$ and $OC_{Impl}$ |
| R2 | Floating ‘carry-in’ Signal for Addition |
| R3 | ‘aluout_zero_node’ is stuck-at-1 in $OC_{Impl}$ |
| X1 | Bug-free X86 Design and Property |
| X2 | The Property Swaps $en_{Integer}$ and $en_{FloatingPoint}$ |

| | |
|---|---|
| X4 | The Opcode for CMP Activates the FP Unit |

# Reveal's Performance



# Refinement Iterations for EUF and CLU Abstractions



# Genuine Bug Discovered

- OpenCores/RISC16F84:
  - A bug due to a floating c_in signal in:

```
// risc16f84_lite.v
reg c_in; // line 223
{add_node,temp} <= {1'b0,aluinp1_reg,1'b1}+
                   {1'b0,aluinp2_reg,c_in}; // line 519
```

# Lemmas

Lemma

$$(\mathrm{IR3} = 32'\mathrm{d0}) \rightarrow (\mathrm{IR3}[31{:}26] \neq 6'\mathrm{d4})$$

Source

```
define BEQ 4
define op 31:26
initial IR3 = 32'd0;
case IR3[`op] `BEQ: ...
```

$$(IR[13{:}k_1] = v_1) \rightarrow (IR[13{:}k_2] \neq v_2)$$
$$v_1 \neq v_2$$
$$k_1 \neq k_2$$

```
assign inst_call = (inst_reg[13:11] == 3'b100);
assign inst_goto = (inst_reg[13:11] == 3'b101);
assign inst_bcf = (inst_reg[13:10] == 4'b0101);
...
```

# Experimental Conclustions

- Memory Abstraction reduces the size of the verification condition significantly ➔ speeds up runtime UCLID, BAT, and Reveal
- Datapath Abstraction + Counterexample-Guided Refinement + Multiple Lemmas per Iteration ➔ Scalable
- SMT-based Solving (YICES) is very scalable
- Integrating Counting Arithmetic Speeds up Convergence, but might slow down each iteration

# Outline

- Verification Framework
- Datapath Abstraction and Basic Refinement
- Advanced Refinement
- The Reveal System
- Results
- Related Work
- Conclusions

# Related Work

- Earlier efforts in verifying microprocessor control logic (past two decades)
  - How to formulate the correctness criterion?
  - What mathematical model to use?

- Two Abstraction Approaches
  - Datapath Abstraction
  - Property-driven Abstraction

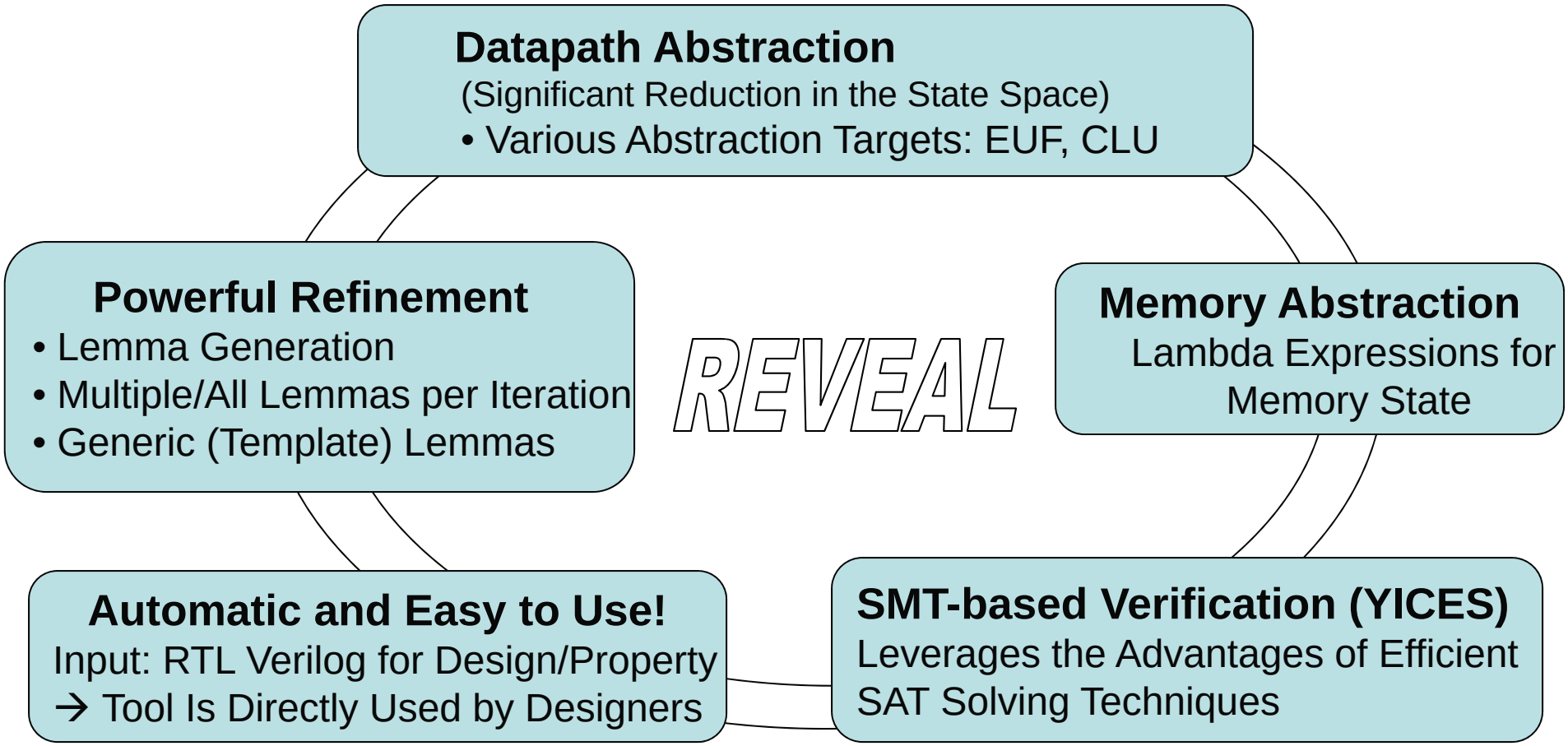
# Datapath Abstraction

- Theorem Proving
  - For two decades (PVS, HOL, ACL2, etc.)
  - Representing the datapath with integers, rationales, high-order logic relations, black-boxes, arrays, lists, etc.
  - Both the modeling and formulation of proofs require continuous user intervention
- Towards Automation
  - The work of Burch and Dill ‘94: EUF
  - The work of Bryant *et al.* ‘02: UCLID
  - The work of Andraus and Sakallah ’04: Vapor

# Property-driven Verification and Abstraction

- Based on generic formal methods
- Property is closely integrated in the abstraction and verification
- Abstraction methods:
  - Predicate abstraction
  - Localization reduction
  - Interpolants

# Conclusions




**Can be Used by Designers**

**Effective Discovery of Genuine Bugs**

**Establishes Correctness within Short Time-to-Market**