Verification of Hardware IP Security and Trust

Prabhat Mishra

Professor

Computer and Information Science and Engineering

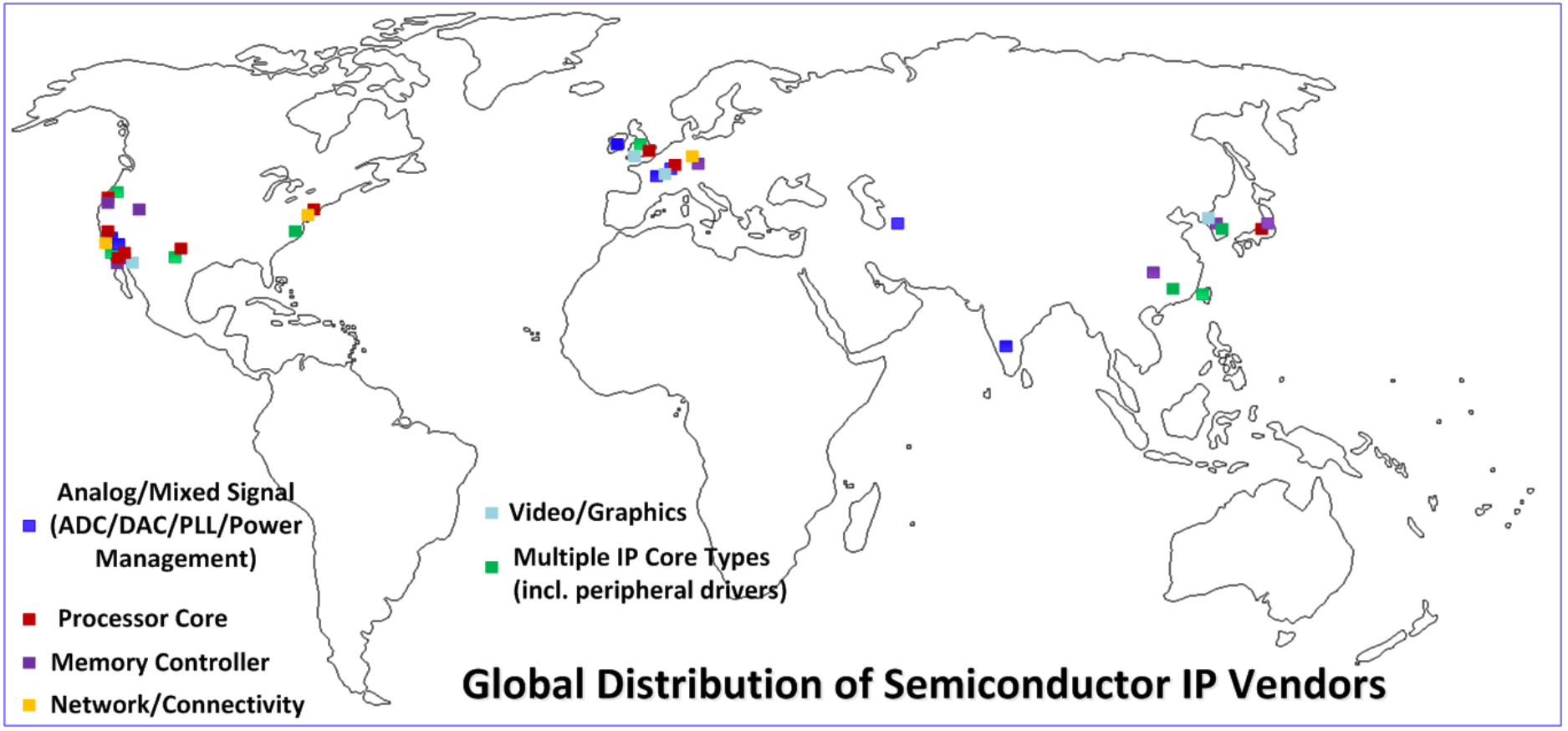
University of Florida, USA

UF UNIVERSITY of FLORIDA | EMBEDDED SYSTEMS LAB

Outline

- Introduction
- Design for Security
- IP Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Conclusion

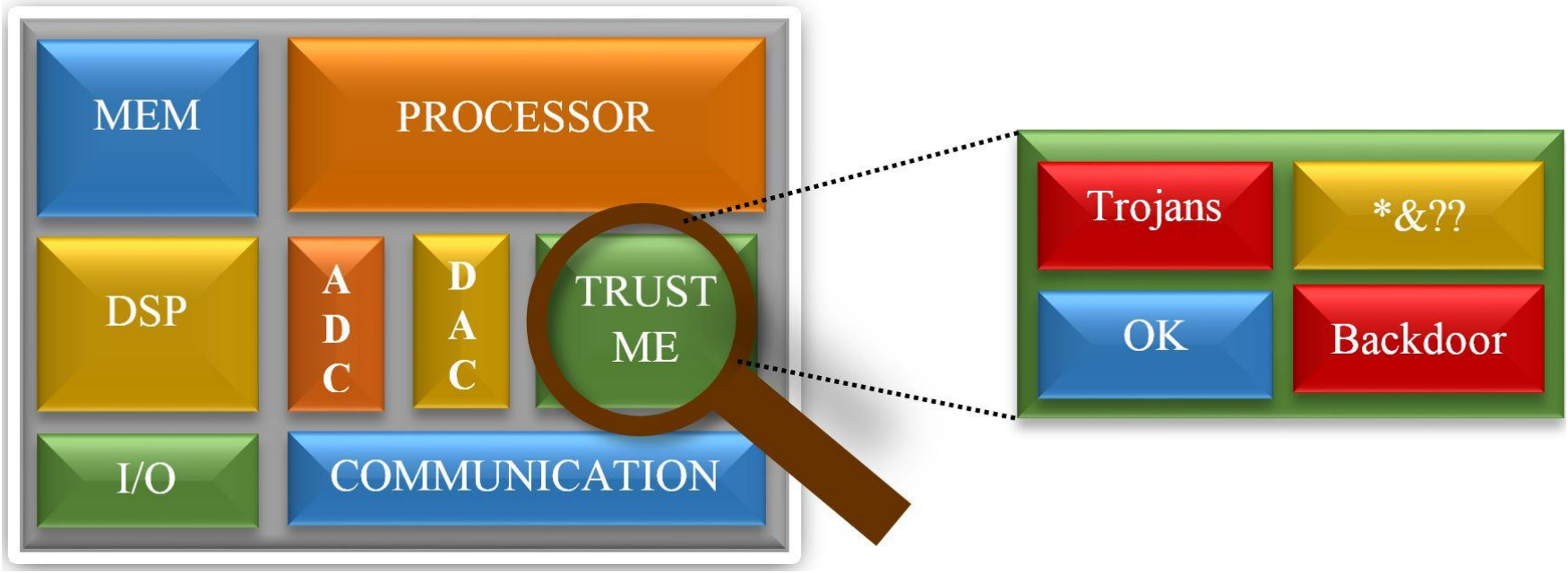
SoC Design using Intellectual Property (IP) Blocks



Long and globally distributed supply chain of hardware IPs makes SoC design increasingly vulnerable to diverse trust/integrity issues.

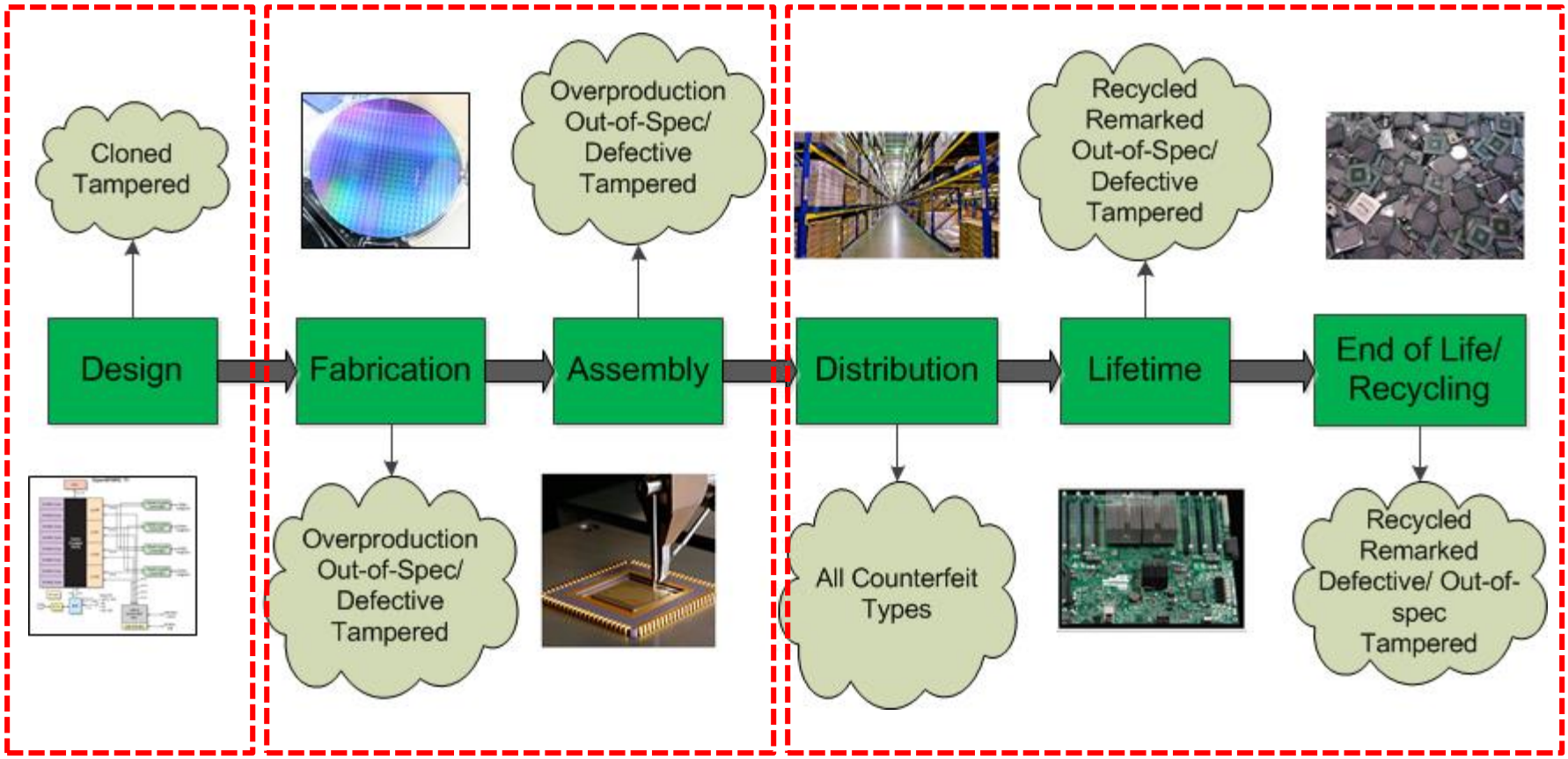
Prabhat Mishra, Swarup Bhunia and Mark Tehranipoor (Editors), Hardware IP Security and Trust, ISBN: 978-3-319-49024-3, Springer, 2017.

Trust Me!



Farimah Farahmandi, Yuanwen Huang and Prabhat Mishra, System-on-Chip Security Validation and Verification, Springer, ISBN: 978-3-030-30596-3, 2019.

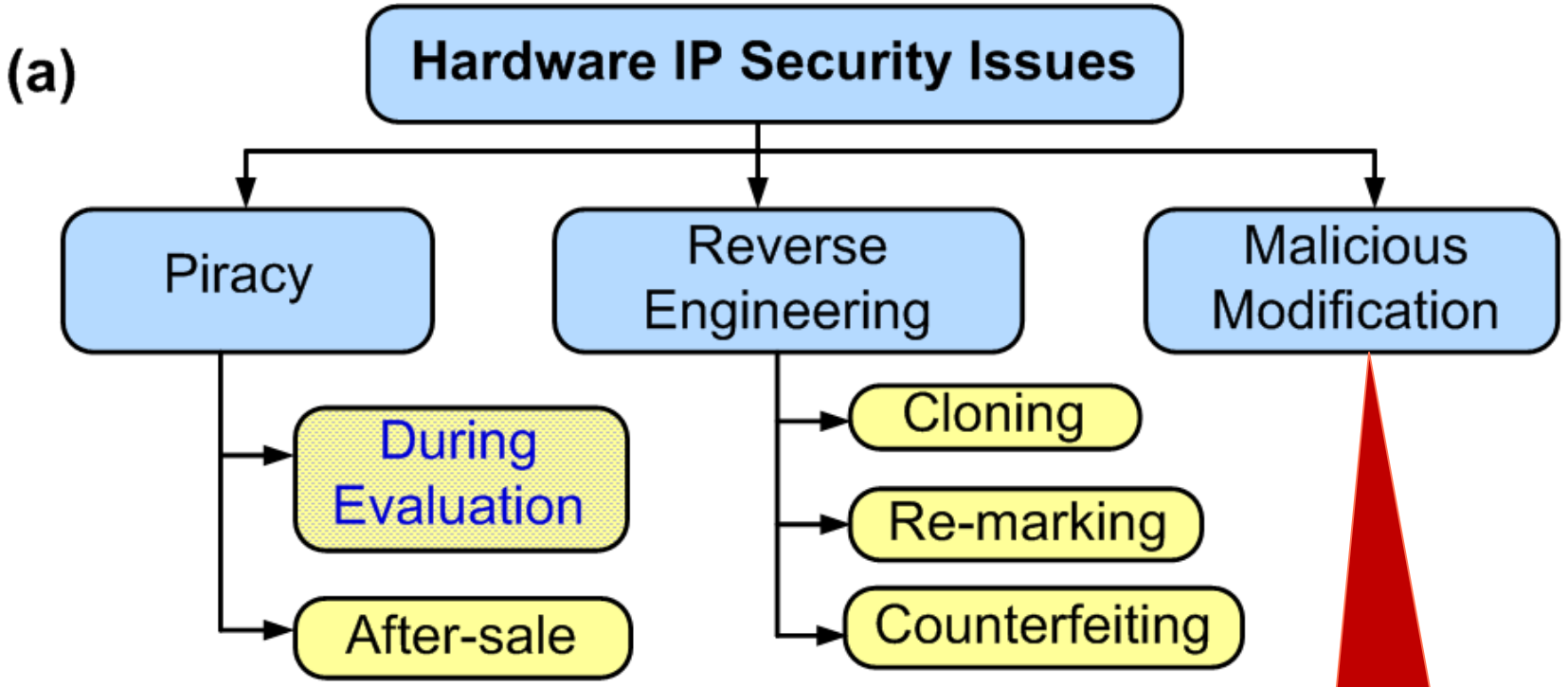
Electronics Supply Chain Security



Maximum Flexibility

Minimum Flexibility

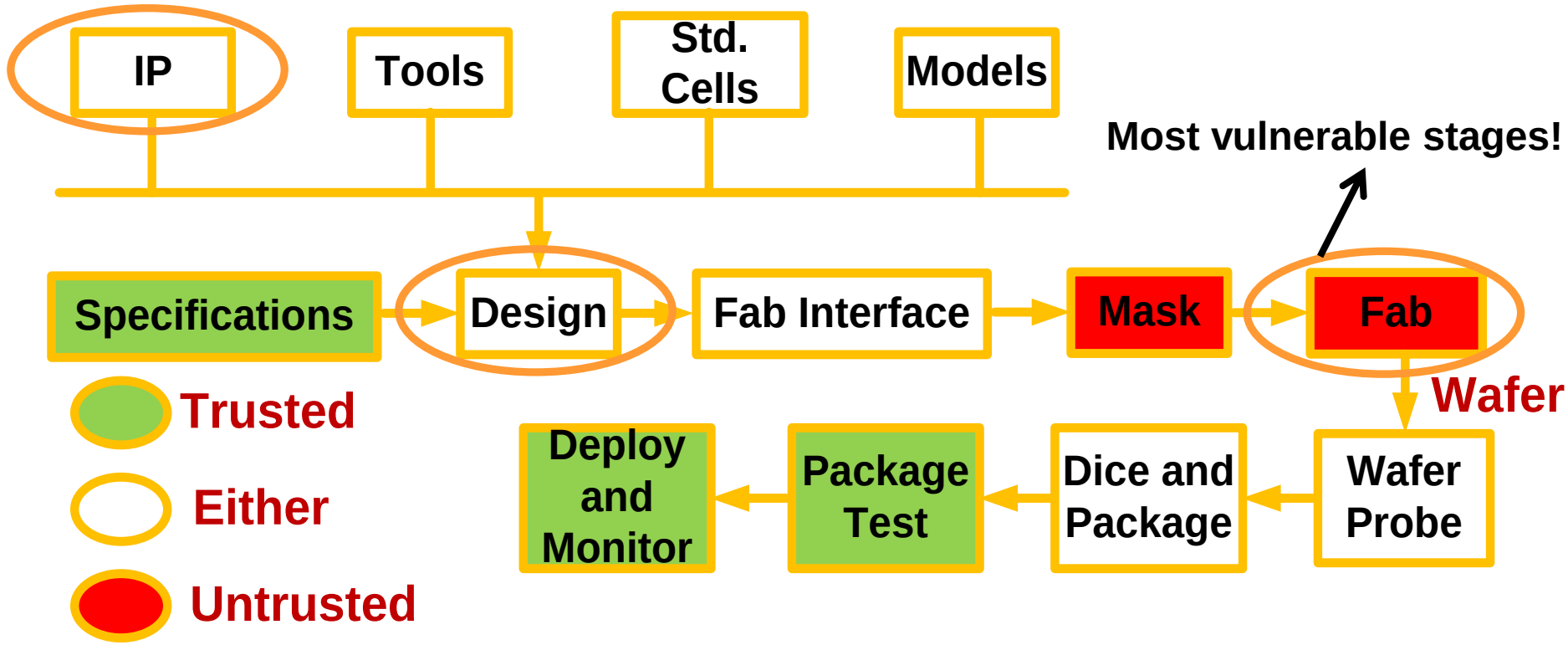
What are the challenges?



Taxonomy of hardware IP security issues

Hardware Trojan Attacks!

HW Trojan Threats

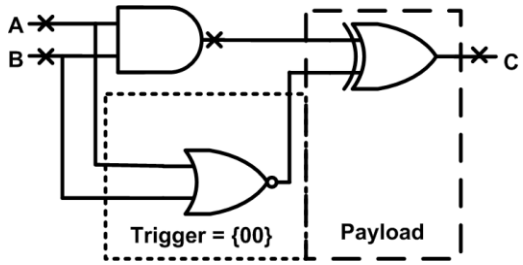


Modern SoC Design and Manufacturing Flow*

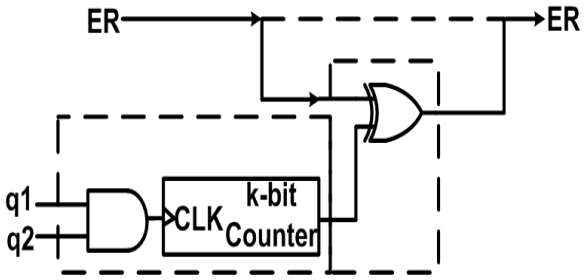
**http://www.darpa.mil/MTO/solicitations/baa07-24/index.html*

HW Trojan Examples/Models

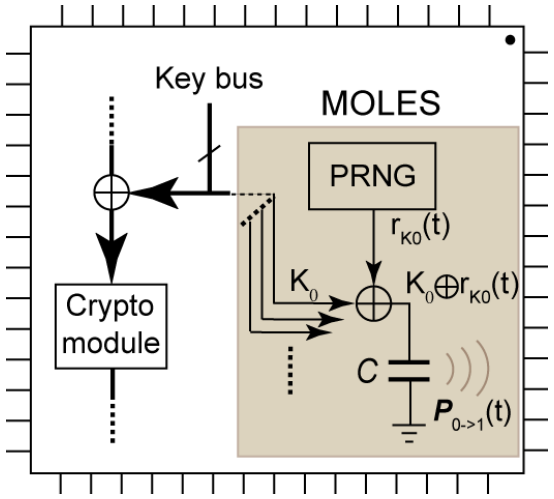
Comb Trojan Example



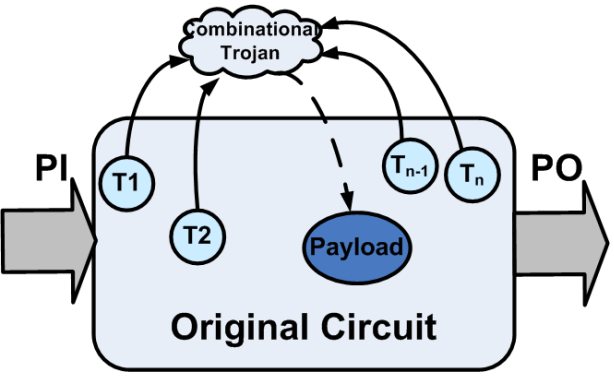
Seq Trojan Example



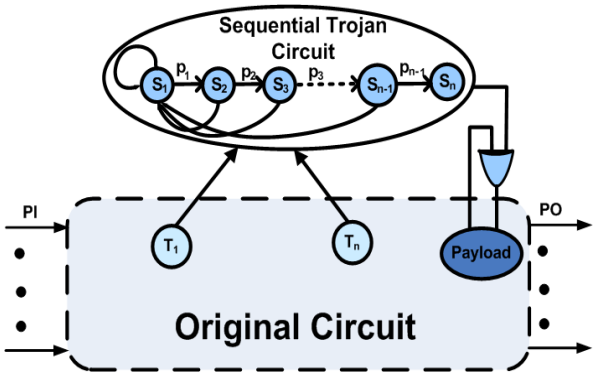
MOLES: Info Leakage Trojan*

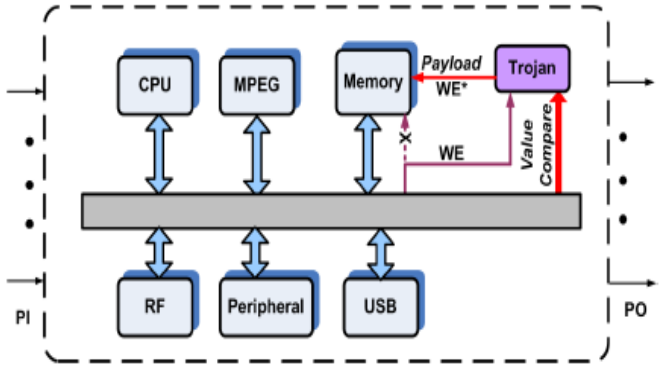


Comb Trojan model



Seq. Trojan Model



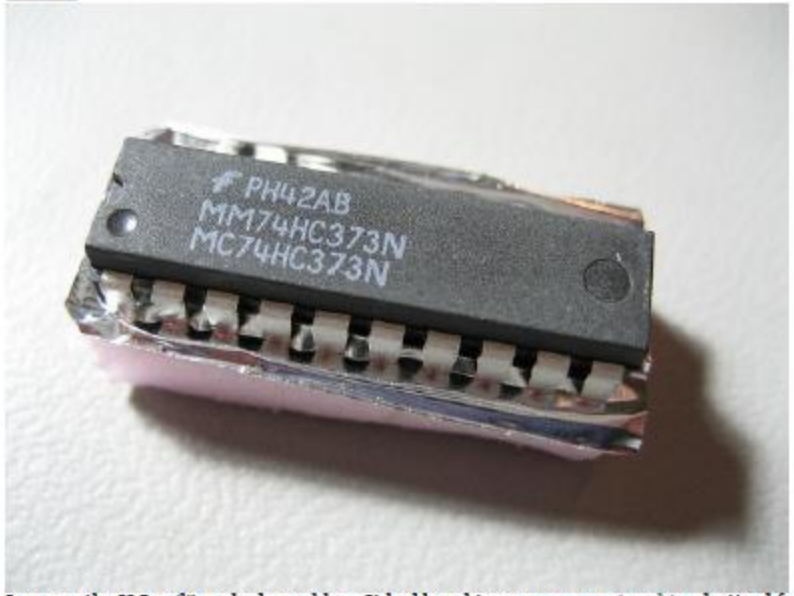


System level view

HW Trojan: in the News

Fishy Chips: Spies Want to Hack-Proof Circuits

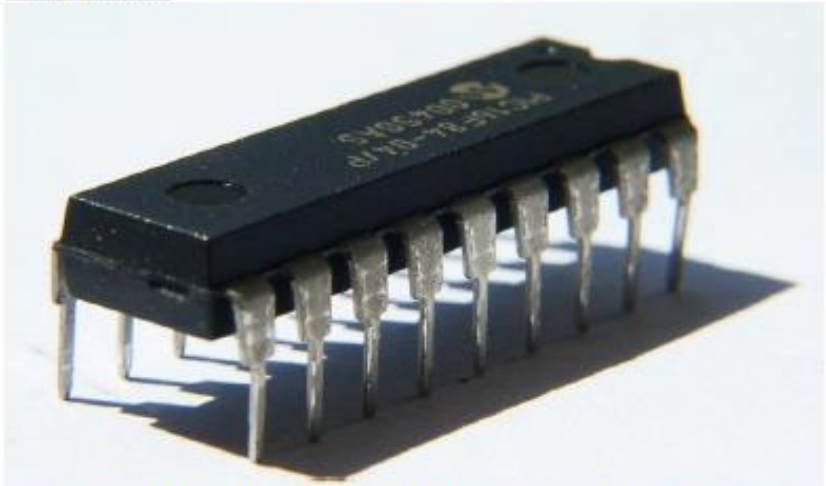
By Adam Rawnsley
06.24.11
12:00 PM
Follow



In 2010, the U.S. military had a problem. It had bought over 59,000 microchips destined for installation in everything from missile defense systems to gadgets that tell friend from foe. The chips turned out to be counterfeits from China, but it could have been even worse. Instead of crappy Chinese fakes being put into Navy weapons systems, the chips could have been hacked, able to shut off a missile in the event of war or lie around just waiting to malfunction.

Can Darpa Fix the Cybersecurity 'Problem From Hell?'

By Adam Rawnsley
08.05.11
9:40 AM
Follow @arawnsley



There are computer security threats — and then there are computer security nightmares. Put sabotaged circuits firmly in the second category. Last week, retired Gen. Michael Hayden, the former CIA and NSA chief, called the hazard of hacked hardware "the problem from hell." "Frankly, it's not a problem that can be solved," he added. "This is a condition that you have to manage."

Why is Trojan Detection Challenging?

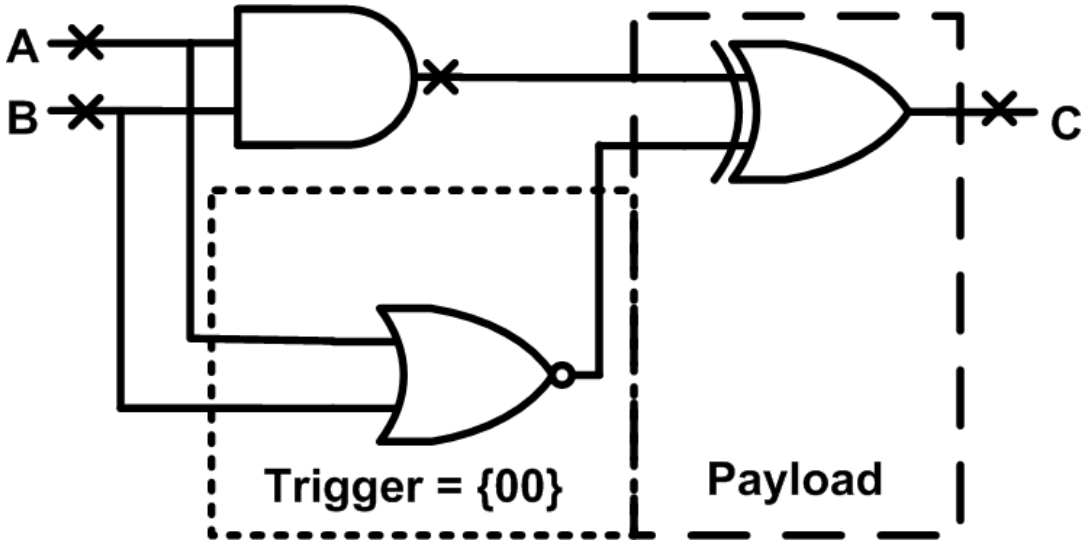
Trojans are stealthy

- Conventional ATPG is not effective

Inordinately large number of possible Trojan instances

- Combinatorial dependence on number of circuit nodes
- 8-bit ALU (*c880*) with 451 nodes → ~10^{11} possible 4-input Trojans!

Sequential Trojans extremely hard to detect

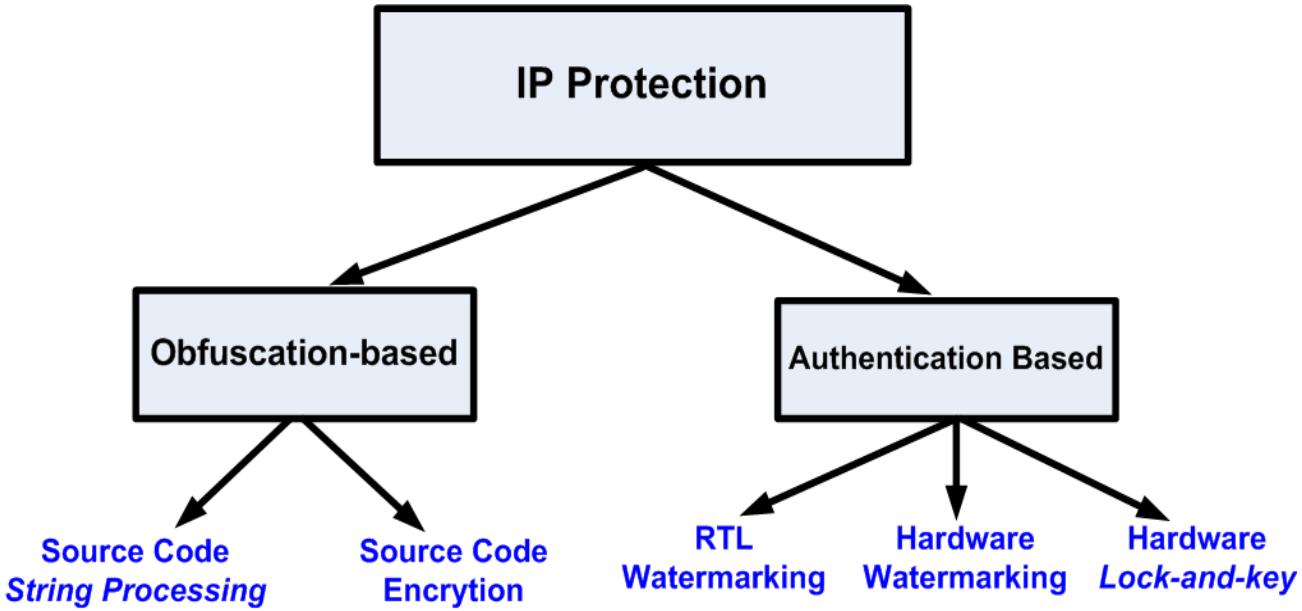


Outline

- Introduction
- Design for Security
  - Logic locking, obfuscation, watermarking, PUF, …
- IP Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Conclusion

HW Obfuscation for IP Protection

- Global Hardware Piracy estimated at $1B/day*
- Causes loss of market share, revenue and reputation
- Affects all parties (IP vendors, IC design houses and System Designers)



Watermark Example

```
case (case_select)
      3'd0 : out = 4'd1;
      3'd3 : out = 4'd4;
      3'd5:  out = 4'd6;
      3'd7:  out = 4'd8;
      default : out =
      4'b0;
endcase
```

Obfuscation-based IP Protection

- **Cryptography-based:**
  - HDL source-code is encrypted [Cadence ’05, Xilinx]
  - Licensed customer with correct key can de-crypt and use
  - Require proprietary design platform support
  - Unacceptable to many SoC design-houses
- **String-processing based [Semantic]**
  - Removes comments
  - Re-names internal wires and registers
  - Affects readability and comprehensibility
- **Code transformation based [Brzozowski & Yarmolik]**
  - Loop unrolling
  - Parallel block => sequential block
  - Flattening register banks

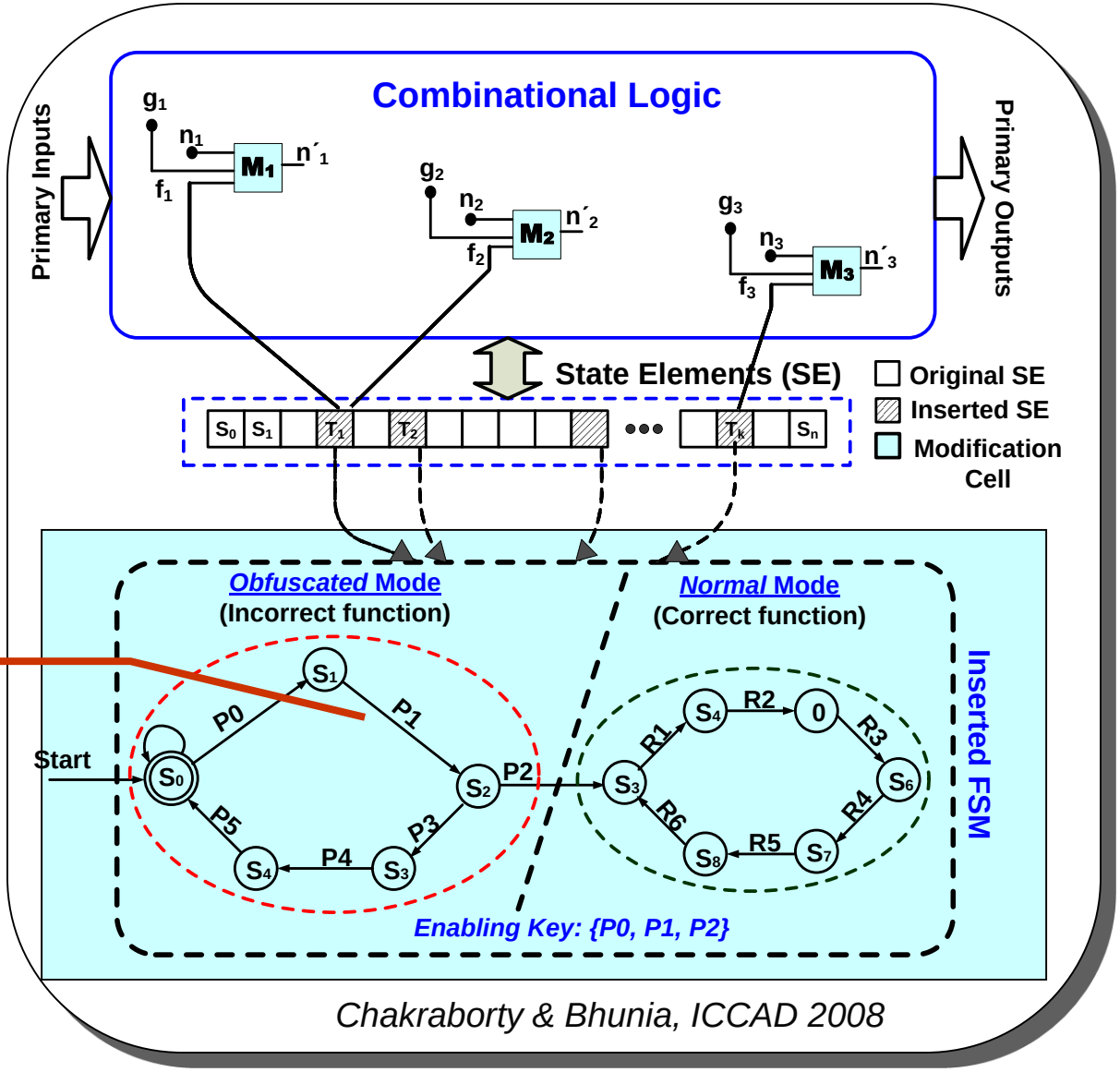
Security Through Key-based Obfuscation

Basic Idea:

- Obfuscate the design functionally and structurally
- Achieved by modifying the state transition function
- Normal behavior is *enabled* only upon application of a key!

Prevents illegal usage of IPs!

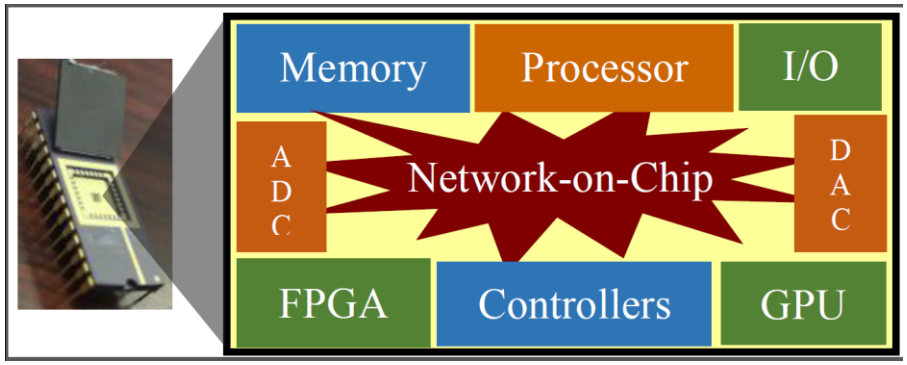
Chakraborty & Bhunia, TCAD 2009

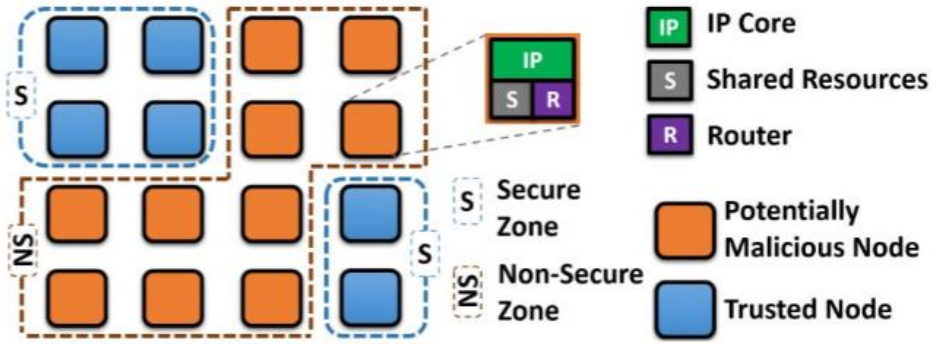


Chakraborty & Bhunia, ICCAD 2008

Design for Security

- IP Specific (Network-on-Chip) Protection
  - Anonymous Routing
  - Trust-aware Routing
  - Authenticated Encryption
  - Detection and Localization of DoS



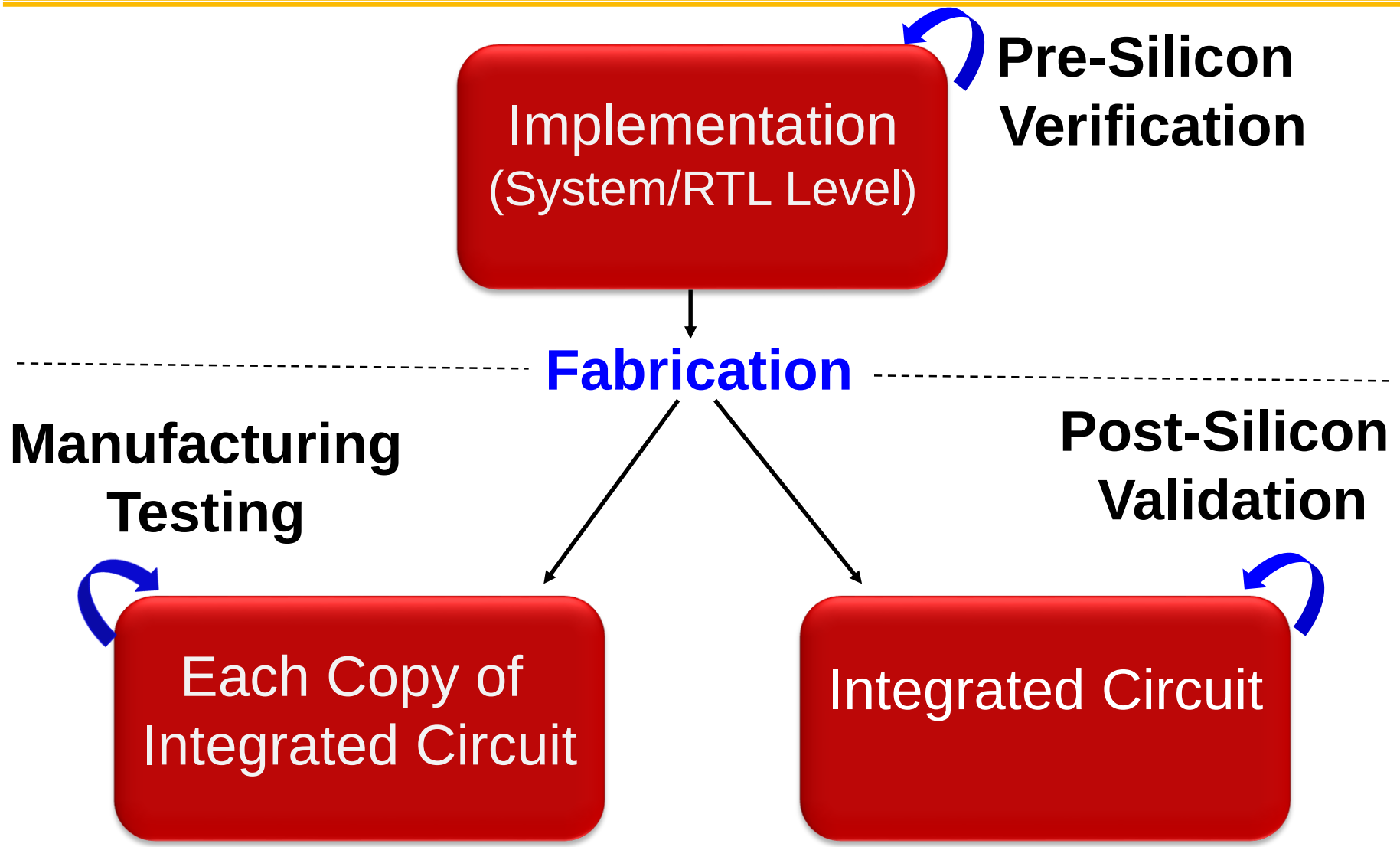


S. Charles, Y. Lyu, P. Mishra, Real-time Detection and Localization of DoS Attacks in NoC based SoCs, Design Automation and Test in Europe (DATE), Florence, Italy, 2019.

Outline

- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Conclusion

Validation of System-on-Chip (SoC) Designs



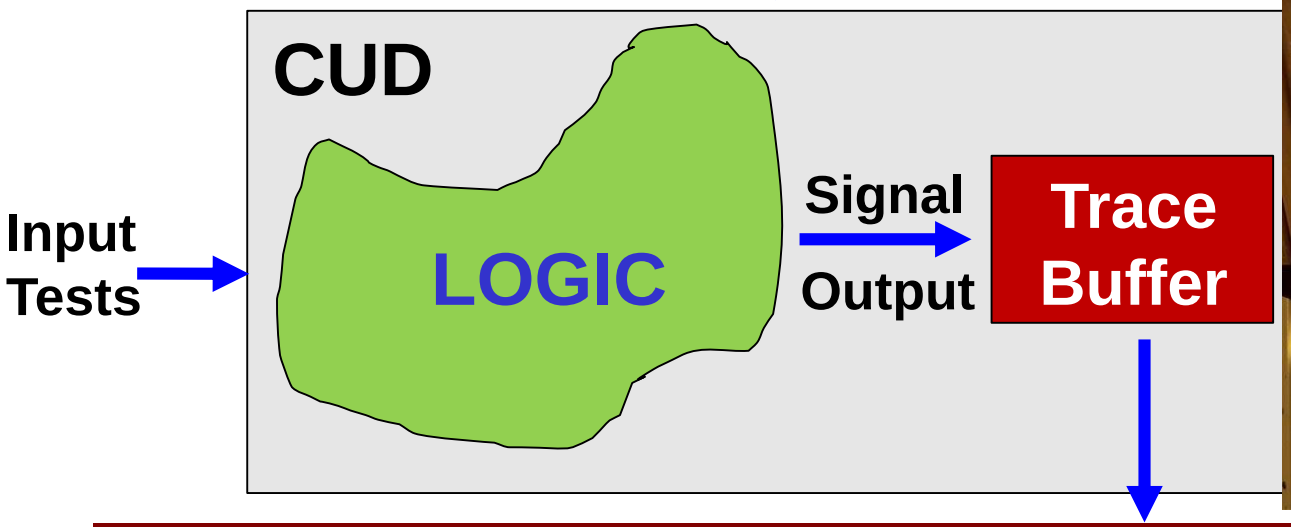
M. Chen, X. Qin, H. Koo and P. Mishra, System-Level Validation: High-Level Modeling and Directed Test Generation Techniques, ISBN: 978-1-4614-1358-5, Springer, 2012.

Post-Silicon Validation

Signal Selection

Implementation (RTL/Gate Level)

Manufacturing

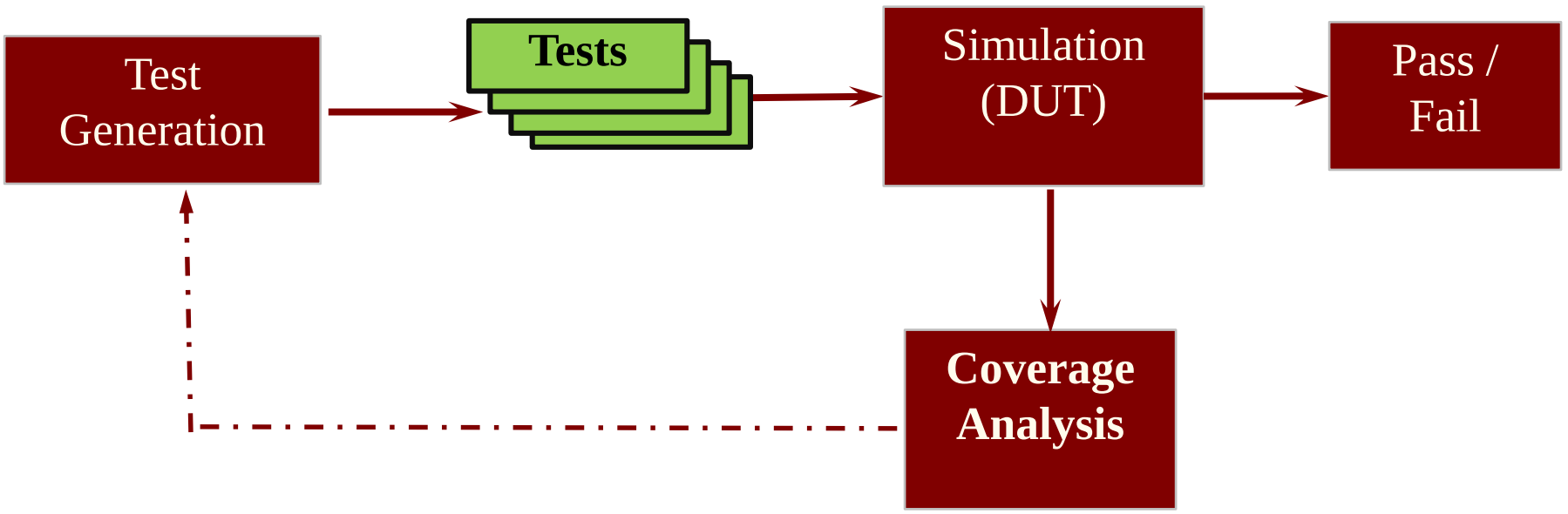


Prabhat Mishra and Farimah Farahmandi (Editors), Post-Silicon Validation and Debug, ISBN: 978-3-319-98115-4, Springer, 2018.

Outline

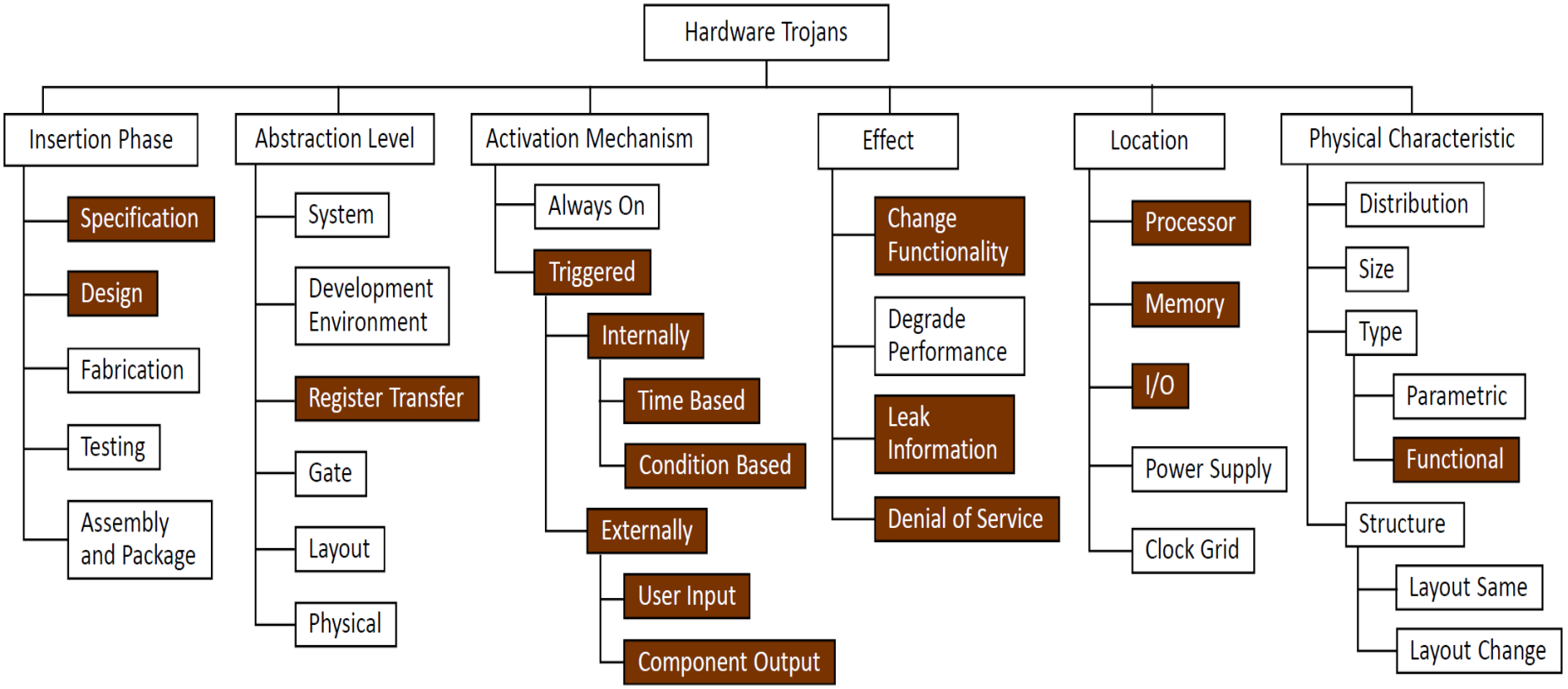
- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Conclusion

Simulation-based Validation



- Simulation-based validation is widely used
  - Uses billions to trillions of random tests
  - Still no guarantee of covering important scenarios

Threat Model



Trojan taxonomy from *www.trust-hub.org*

Trojan detectable by our approach is highlighted

A. Ahmed, F. Farahmandi, Y. Iskander and P. Mishra, Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution, ITC, 2018.

Trust Metrics and Benchmarks

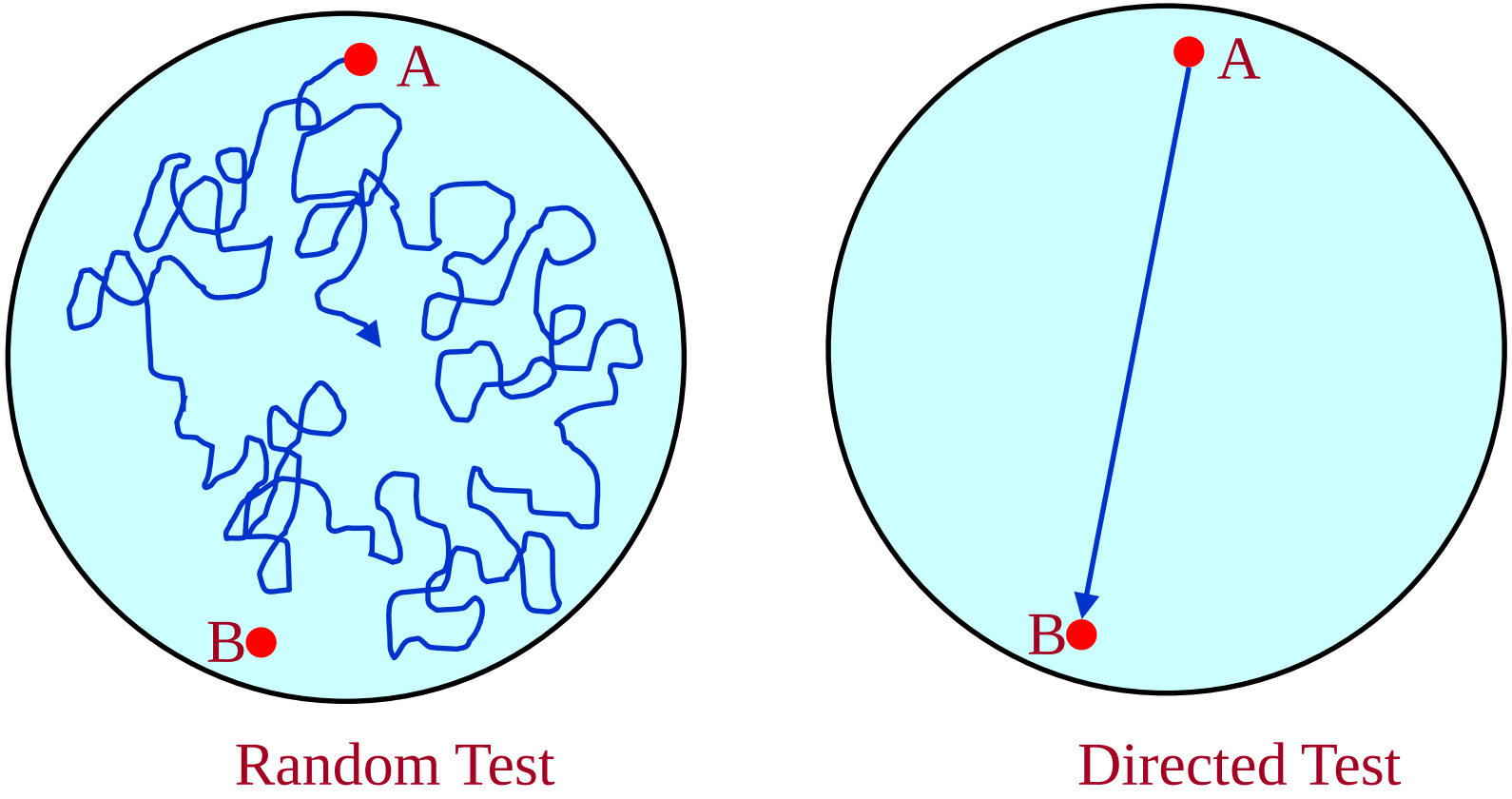
- Functional Validation
  - Code coverage (statement / branch / path)
  - FSM coverage (states and transitions)
  - Property coverage (functional scenarios)
- Parametric Validation
  - Power / thermal violations
  - Real-time violations
  - Rare-node / rare-scenario activations

Jonathan Cruz, Prabhat Mishra and Swarup Bhunia, The Metric Matters: How to Measure Trust, Design Automation Conference (DAC), 2019.

- Static and Dynamic Benchmarks

J. Cruz, Y. Huang, P. Mishra, S. Bhunia, An Automated Configurable Trojan Insertion Framework for Dynamic Trust Benchmarks, Design Automation & Test in Europe 2018.

Directed Test Generation



Random Test

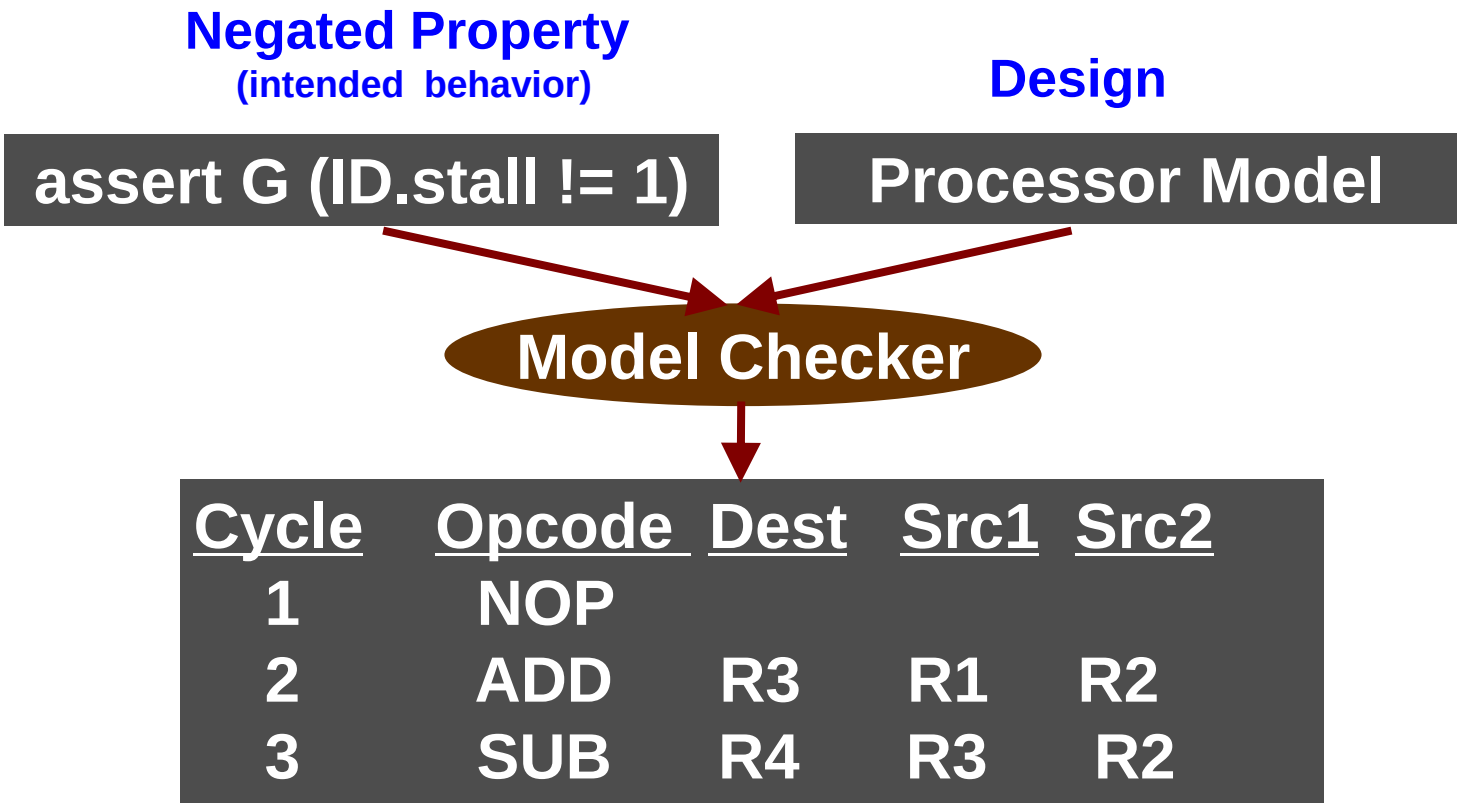
Directed Test

- Significantly less number of directed tests can achieve same coverage goal than random tests
- Need for automated generation of directed tests

Y. Lyu, X. Qin, M. Chen and P. Mishra, Directed Test Generation for Validation of Cache Coherence Protocols, IEEE Transactions on CAD (TCAD), February 2018.

Test Generation using Model Checking

Example: Generate a directed test to stall a decode unit (ID)

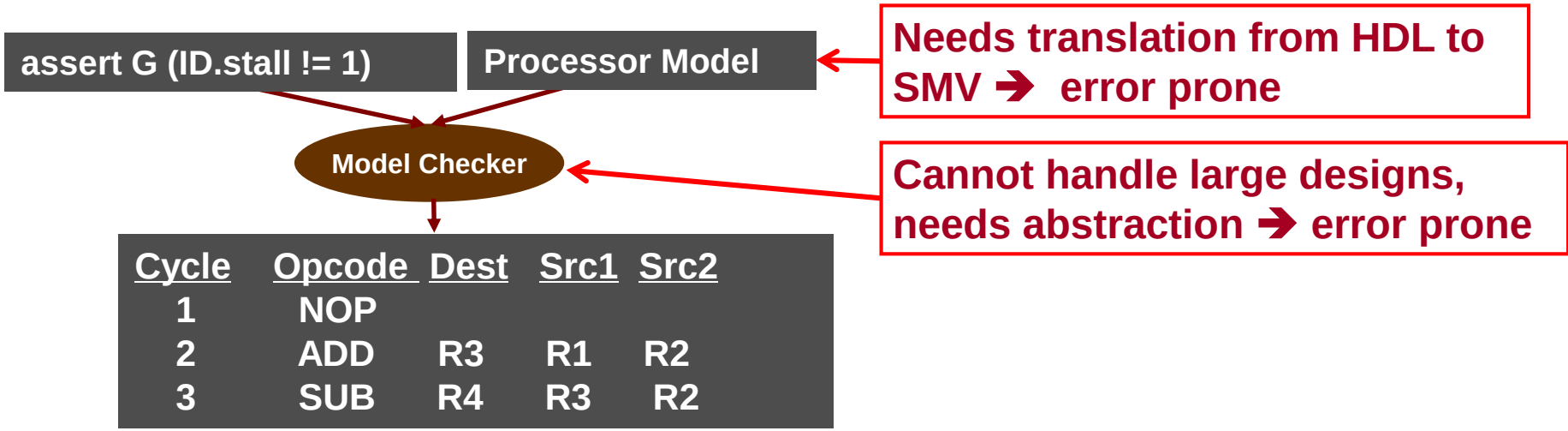


Solution: Exploit learning to reduce test generation complexity

Problem: Test generation is time consuming and may not be possible when complex design and properties are involved

Scalable Directed Test Generation

- Test generation based on model checking

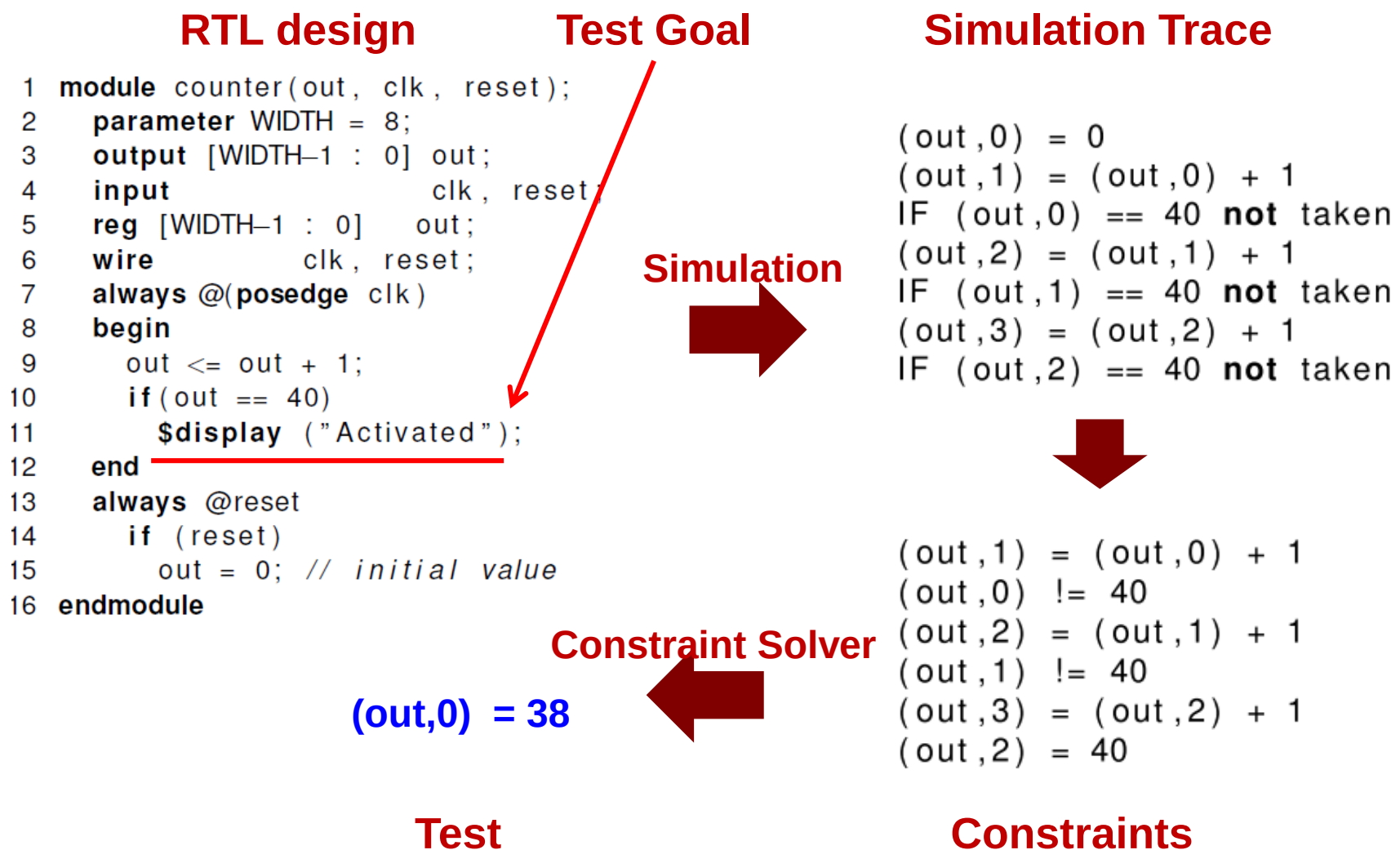


Desirable to verify the HDL directly!

- Concolic Testing – Interleaved concrete and symbolic execution [Sen, CAV 2006]

Yangdi Lyu, Alif Ahmed and Prabhat Mishra, Automated Activation of Multiple Targets in RTL Models using Concolic Testing, Design Automation and Test in Europe (DATE), Florence, Italy, March 25 - 29, 2019.

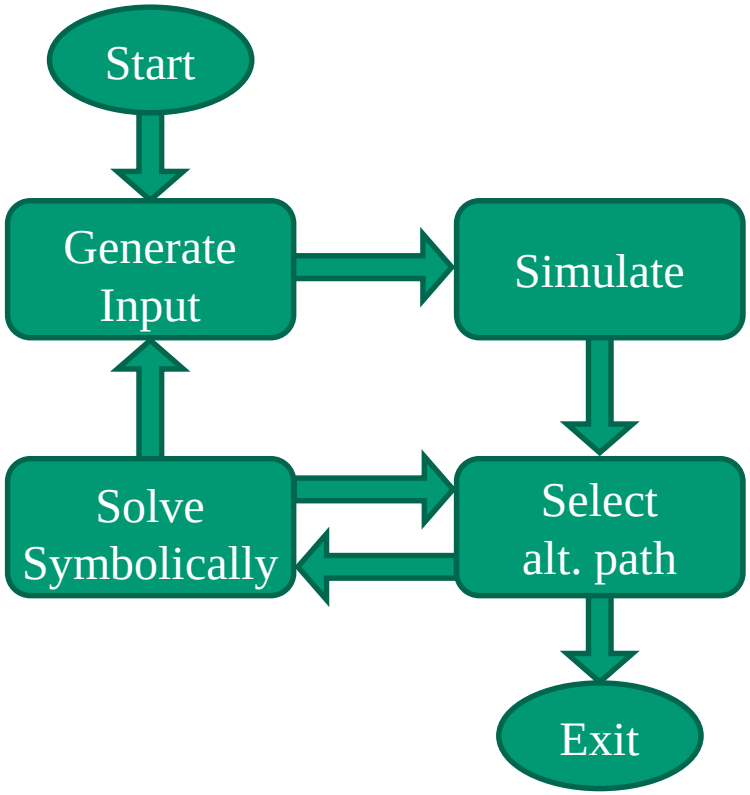
Scalable Directed Test Generation

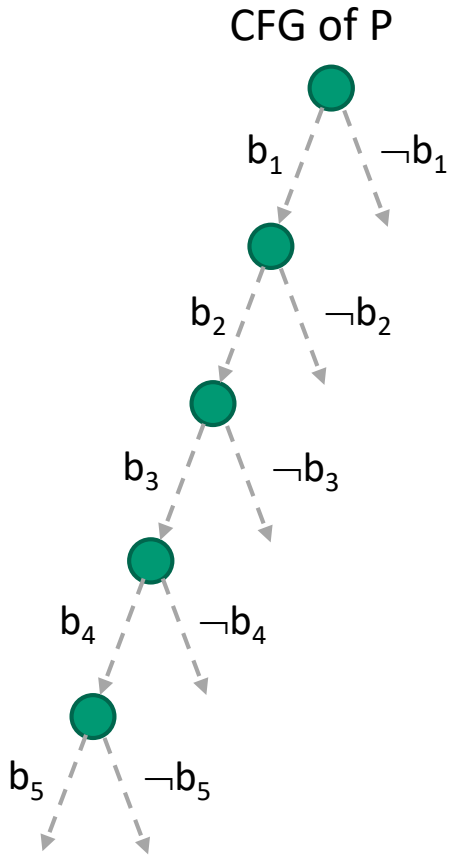


A. Ahmed, F. Farahmandi and P. Mishra, Directed Test Generation using Concolic Testing of RTL Models, Design Automation and Test in Europe (DATE), 2018.

Concolic Testing

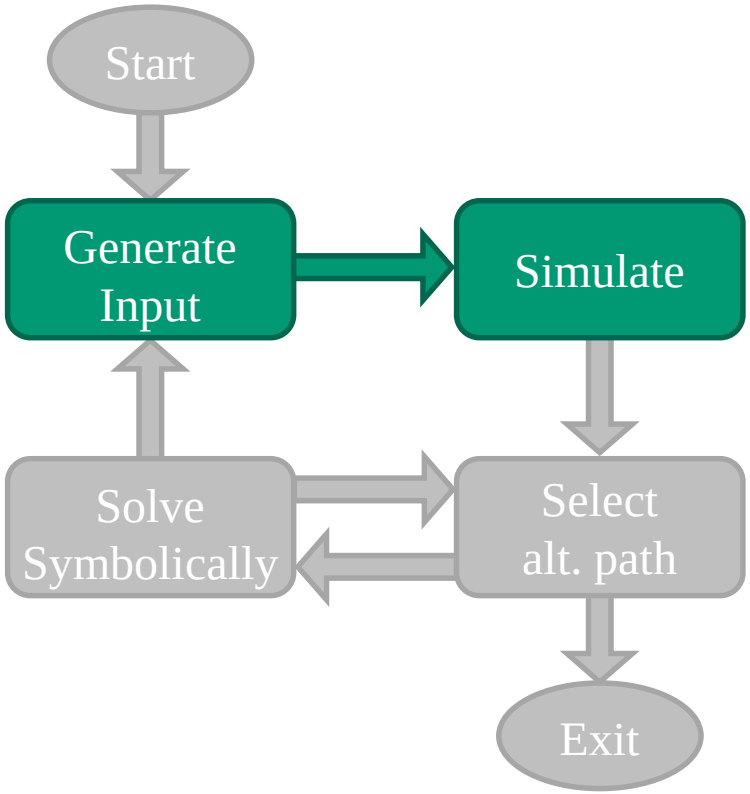
Combines concrete and symbolic execution

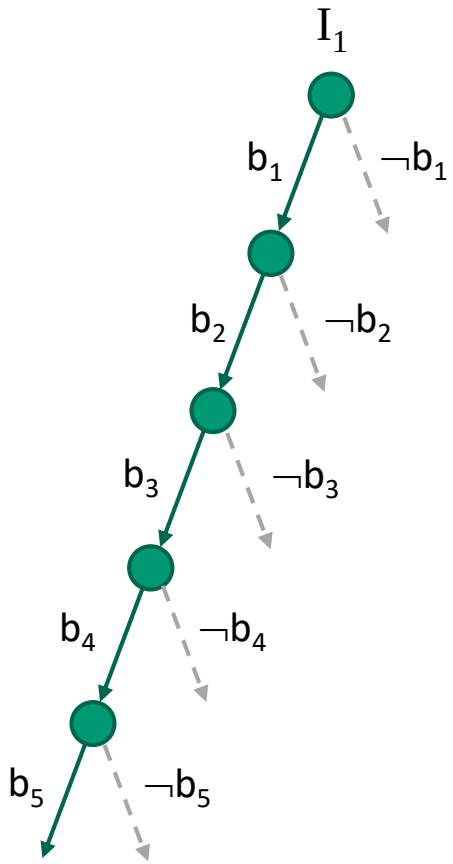




Concolic Testing

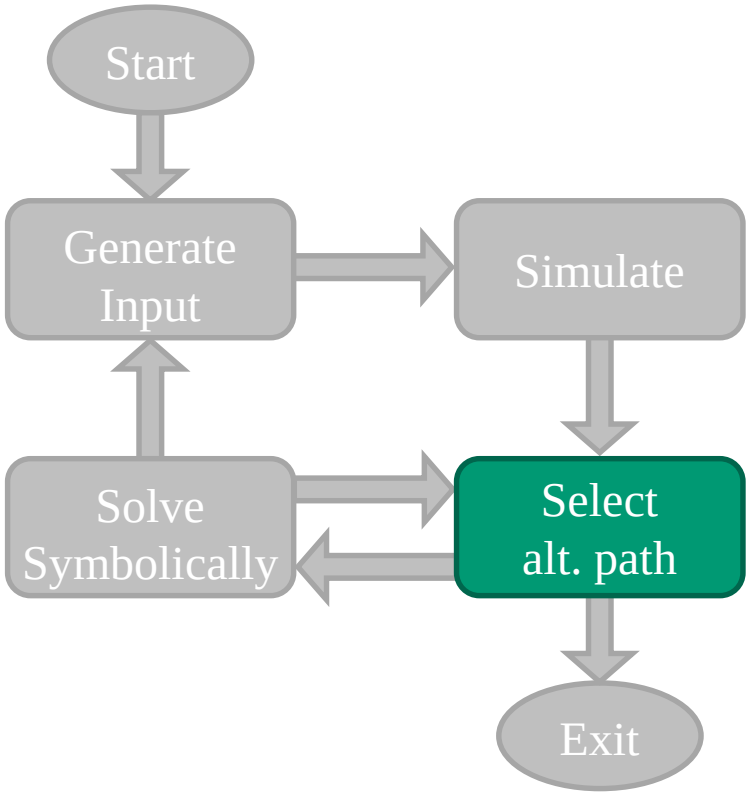
Combines concrete and symbolic execution

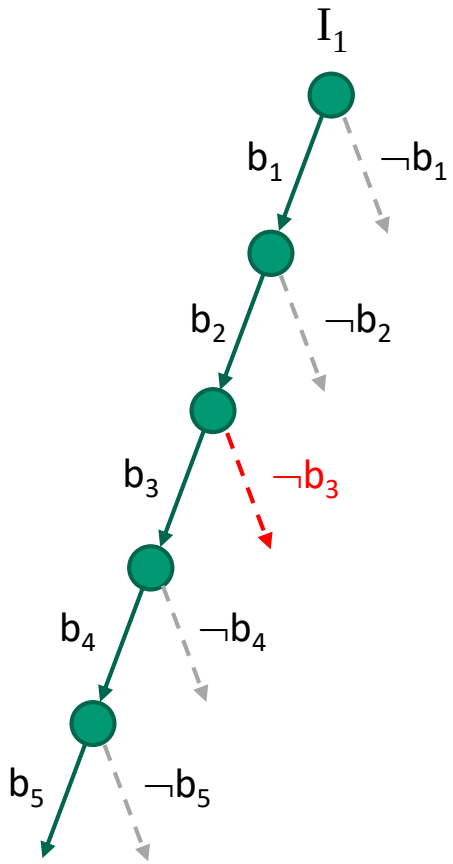




Concolic Testing

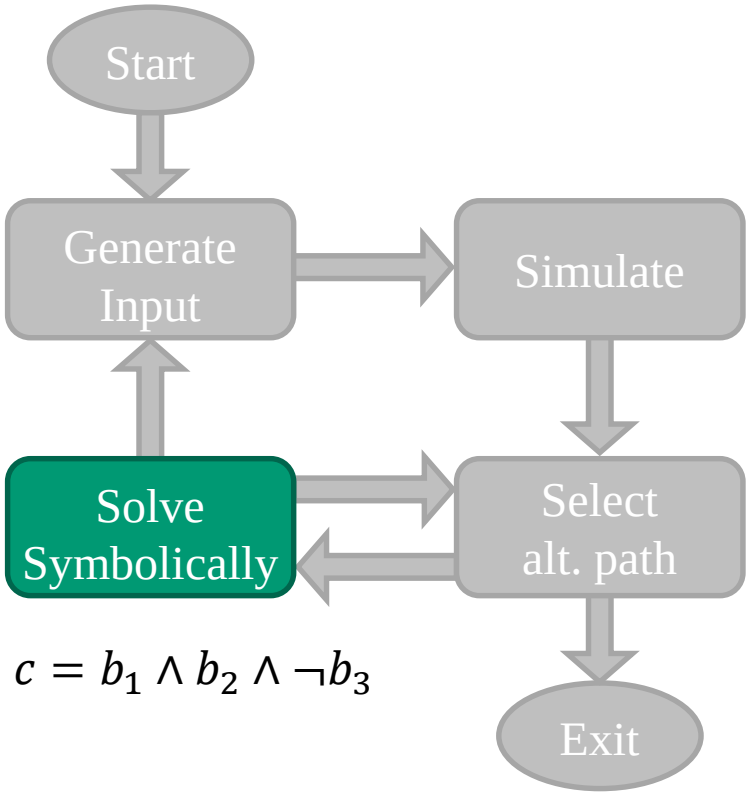
Combines concrete and symbolic execution

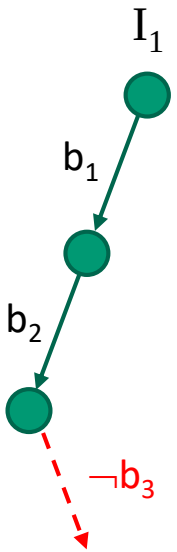




Concolic Testing

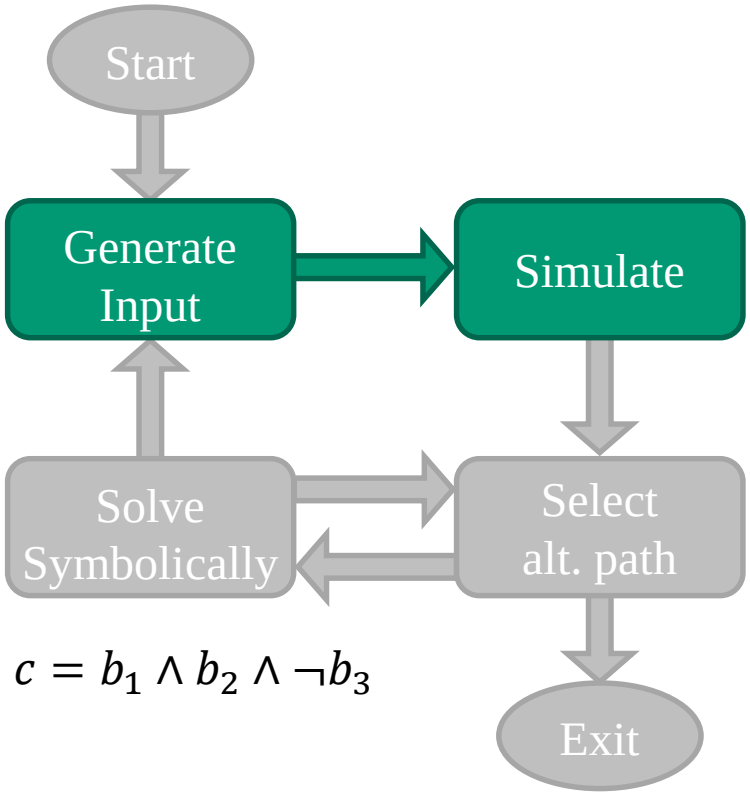
Combines concrete and symbolic execution

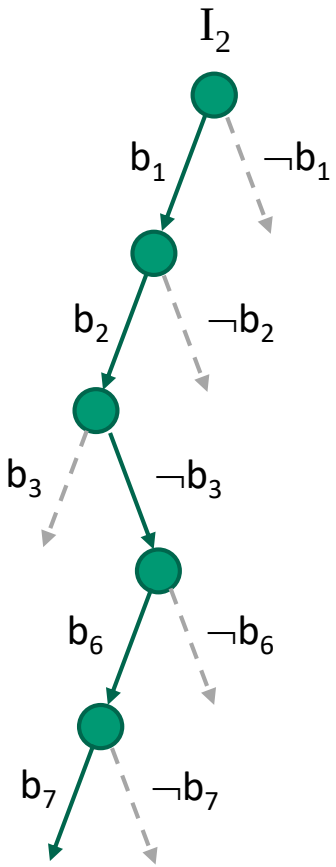




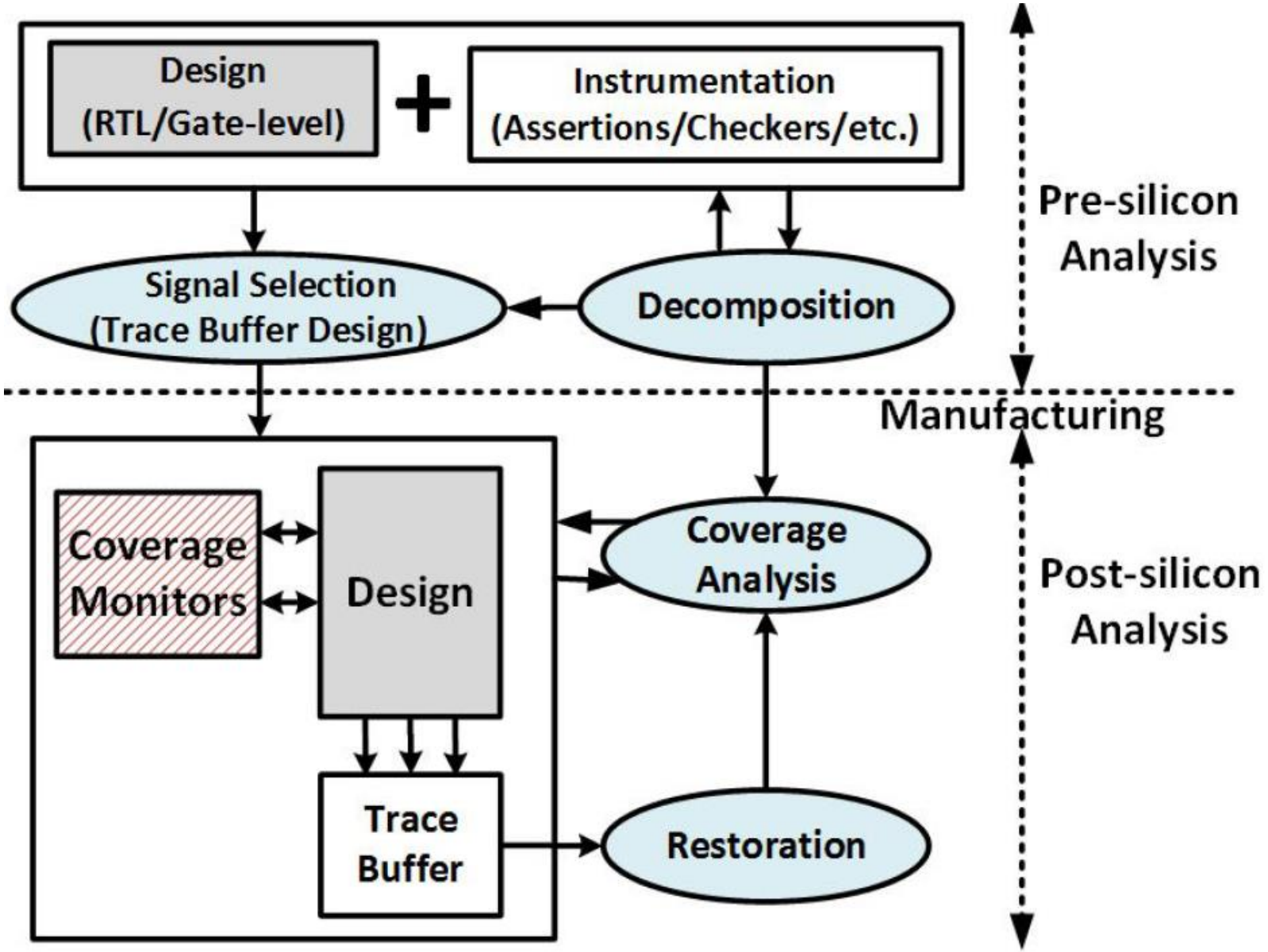
Concolic Testing

Combines concrete and symbolic execution



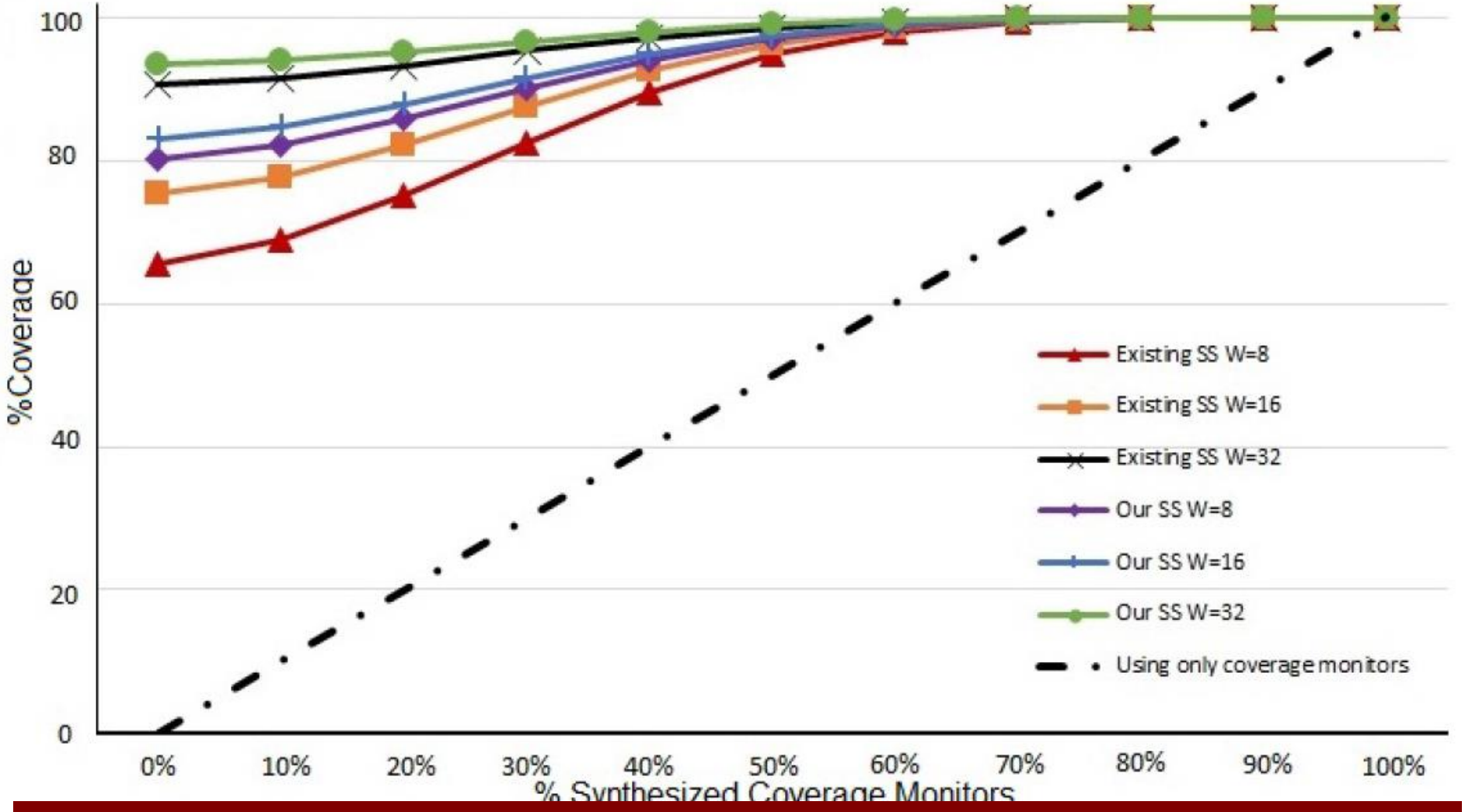


Assertion-based Validation



Coverage analysis for s9234

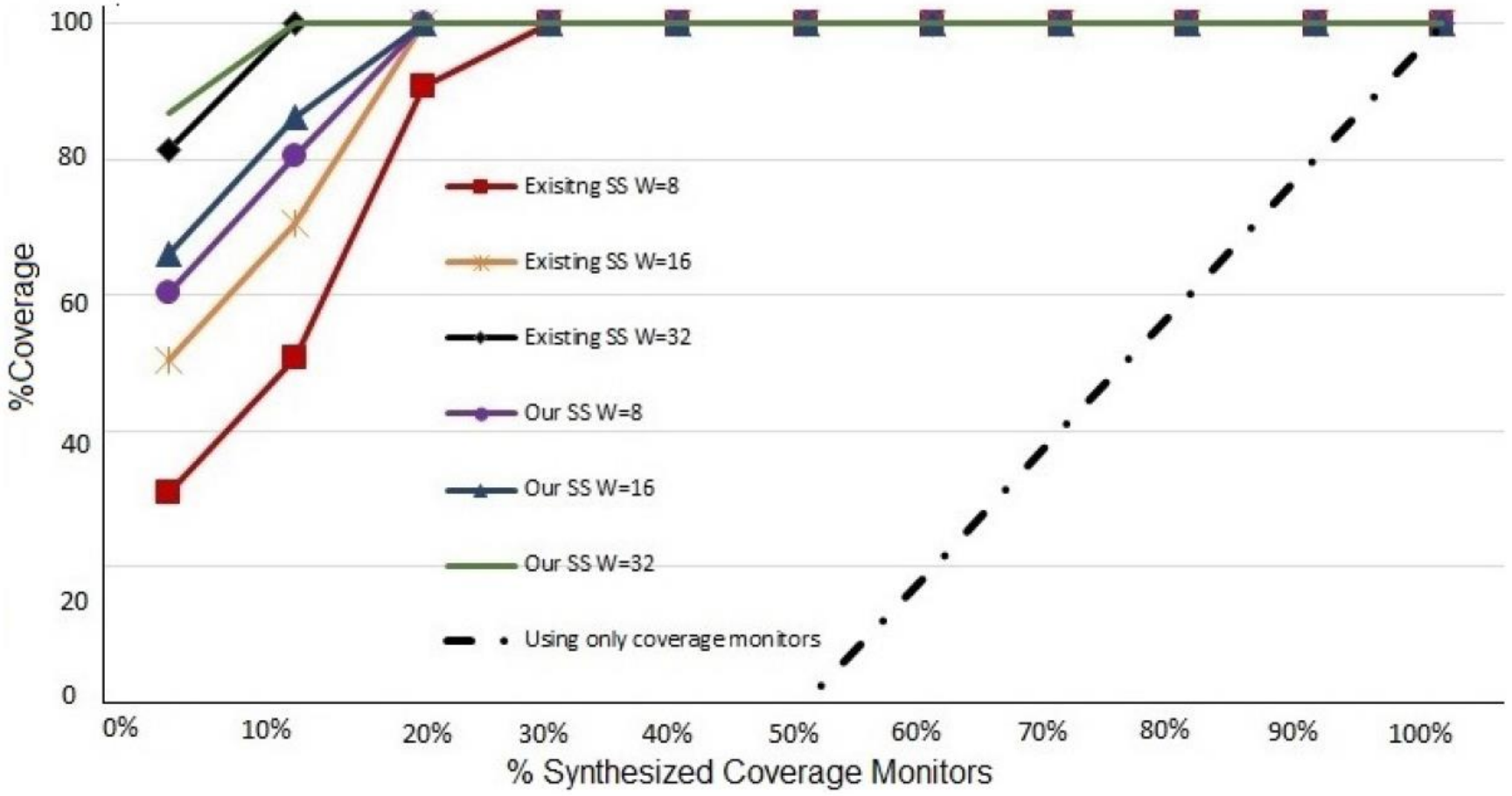
Coverage monitors are selected randomly



F. Farahmandi, R. Morad, A. Ziv, Z. Nevo and P. Mishra, Cost-Effective Analysis of Post-Silicon Functional Coverage Events, Design Automation and Test in Europe (DATE), 392-397, 2017.

Coverage analysis for s9234

Coverage monitors are selected from hard-to-detect events



F. Farahmandi, R. Morad, A. Ziv, Z. Nevo and P. Mishra, Cost-Effective Analysis of Post-Silicon Functional Coverage Events, Design Automation and Test in Europe (DATE), 392-397, 2017.

Outline

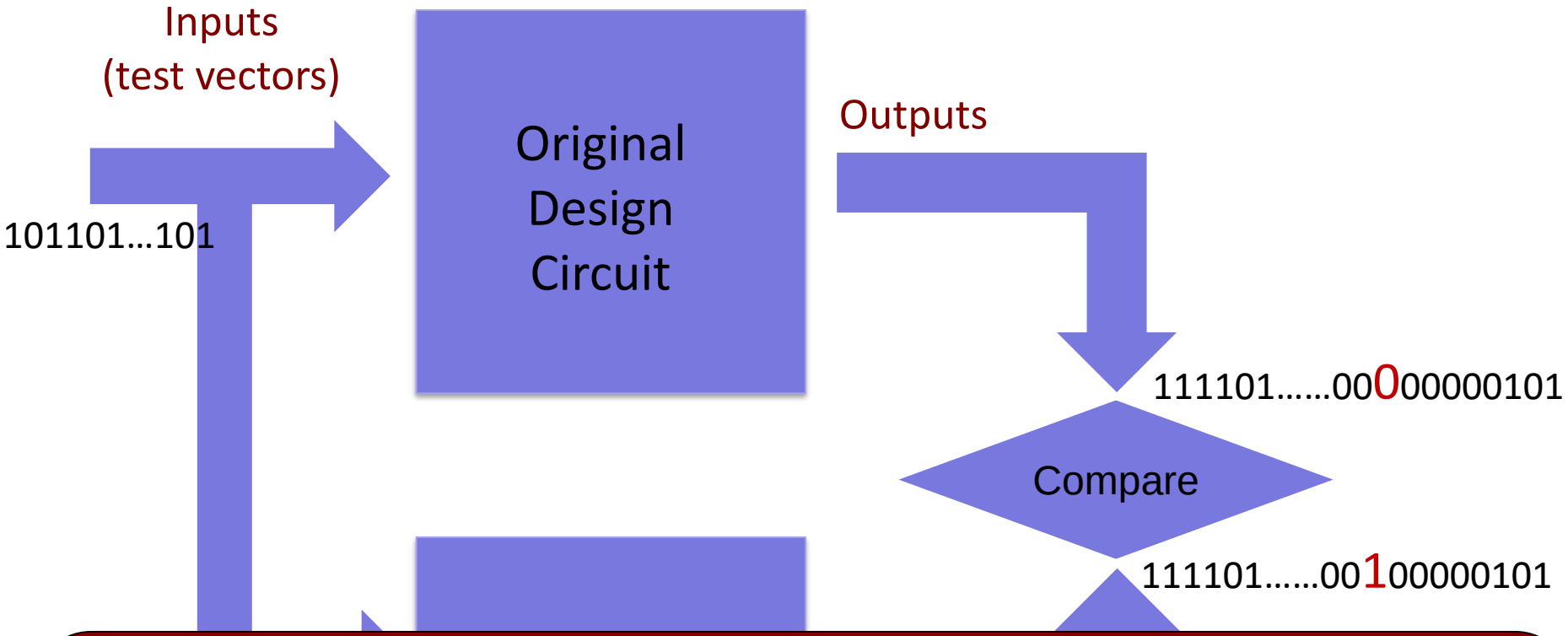
- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Conclusion

HW Trojan Detection

	Logic Testing	Side-Channel Analysis
Pros	• Robust under process noise • Effective for ultra-small Trojans	• Effective for large Trojans • Easy to generate test vectors
Cons	• Difficult to generate test vectors • Large Troj. detection challenging	• Vulnerable to process noise • Ultra-small Troj. Det. challenging

Y. Huang, S. Bhunia and P. Mishra, MERS: Statistical Test Generation for Side-Channel Analysis based Trojan Detection, ACM Conference on Computer and Communications Security (CCS), pages 130-141, 2016.

Logic Testing for Trojan Detection



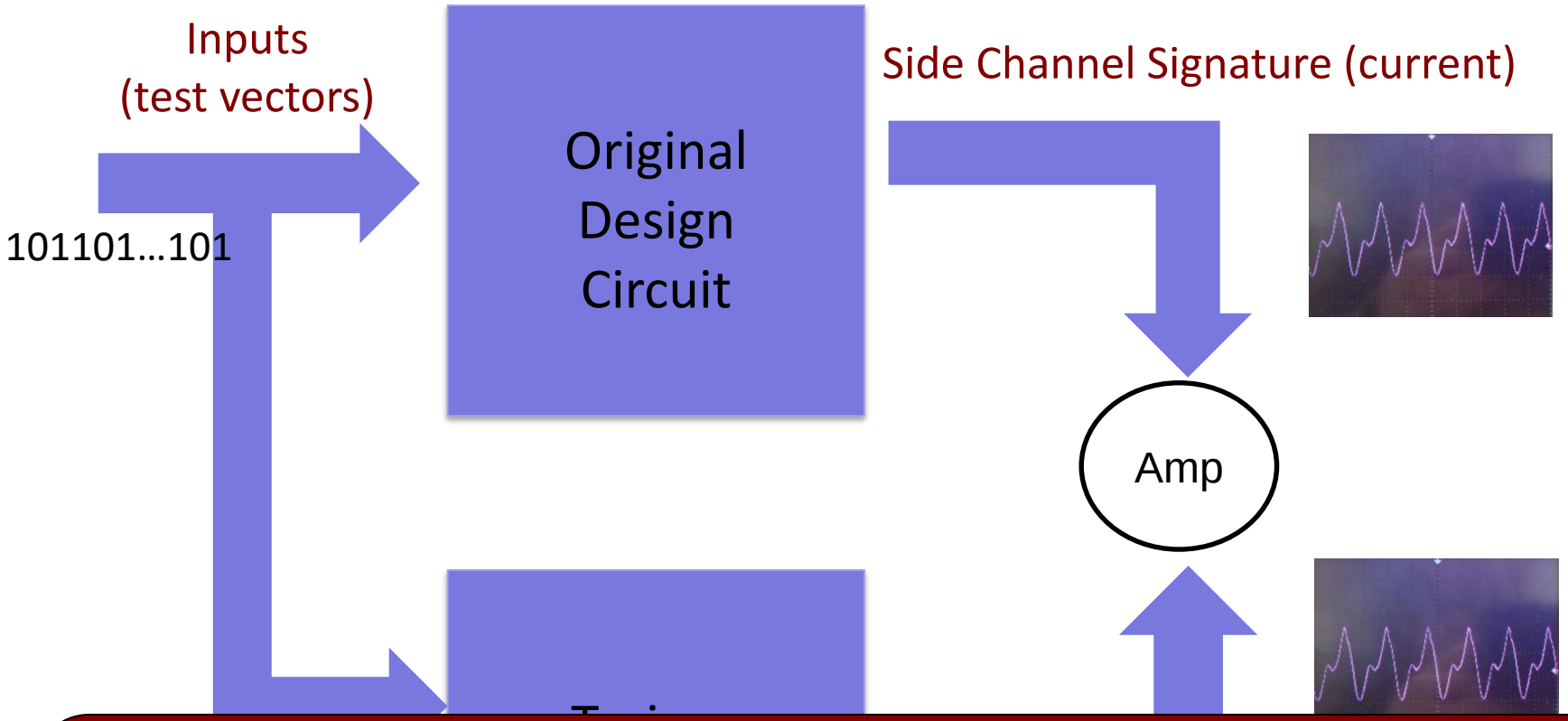
Not effective:

(1) Test space (no way to cover all inputs and all circuit states)

(2) Trojan space (unknown locations, unknown triggers)

(3) Trojan is stealthy (rare triggering)

Side Channel Analysis (SCA) for Trojan Detection

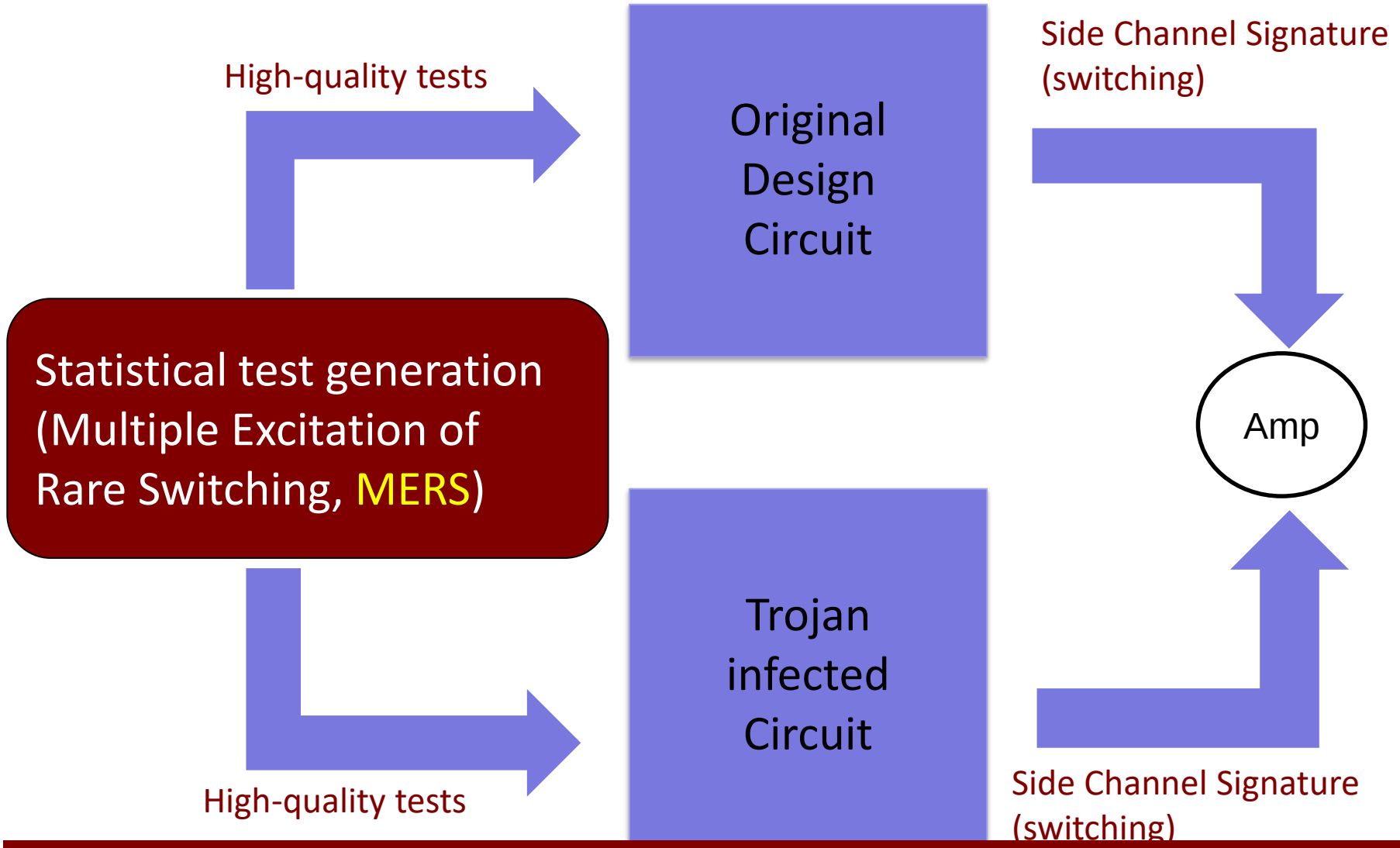


Not effective:

(1) Trojan is small and dormant (different of signature is small)

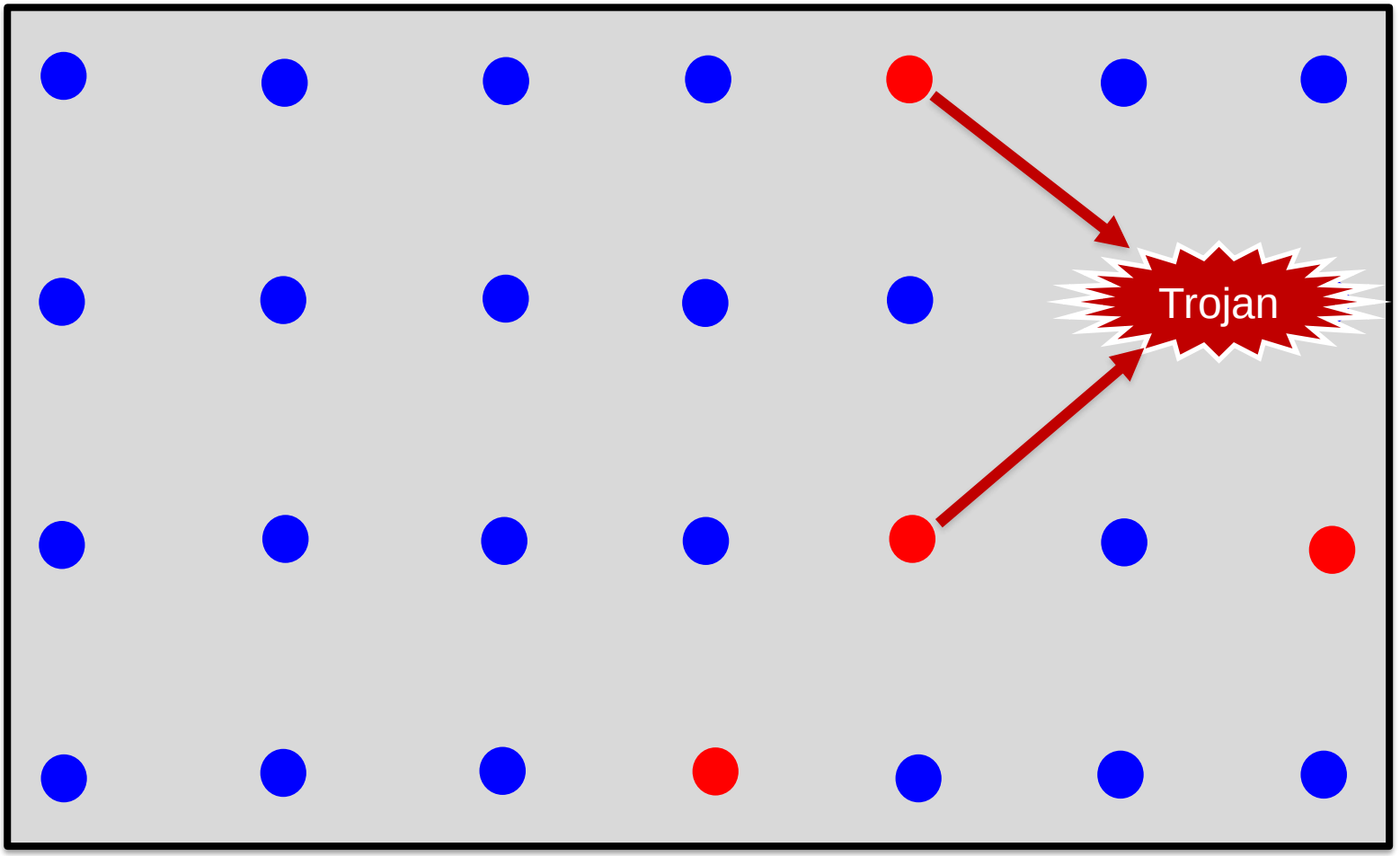
(2) Sensitivity (process noise and background switching)

Our Approach: Logic Testing + SCA



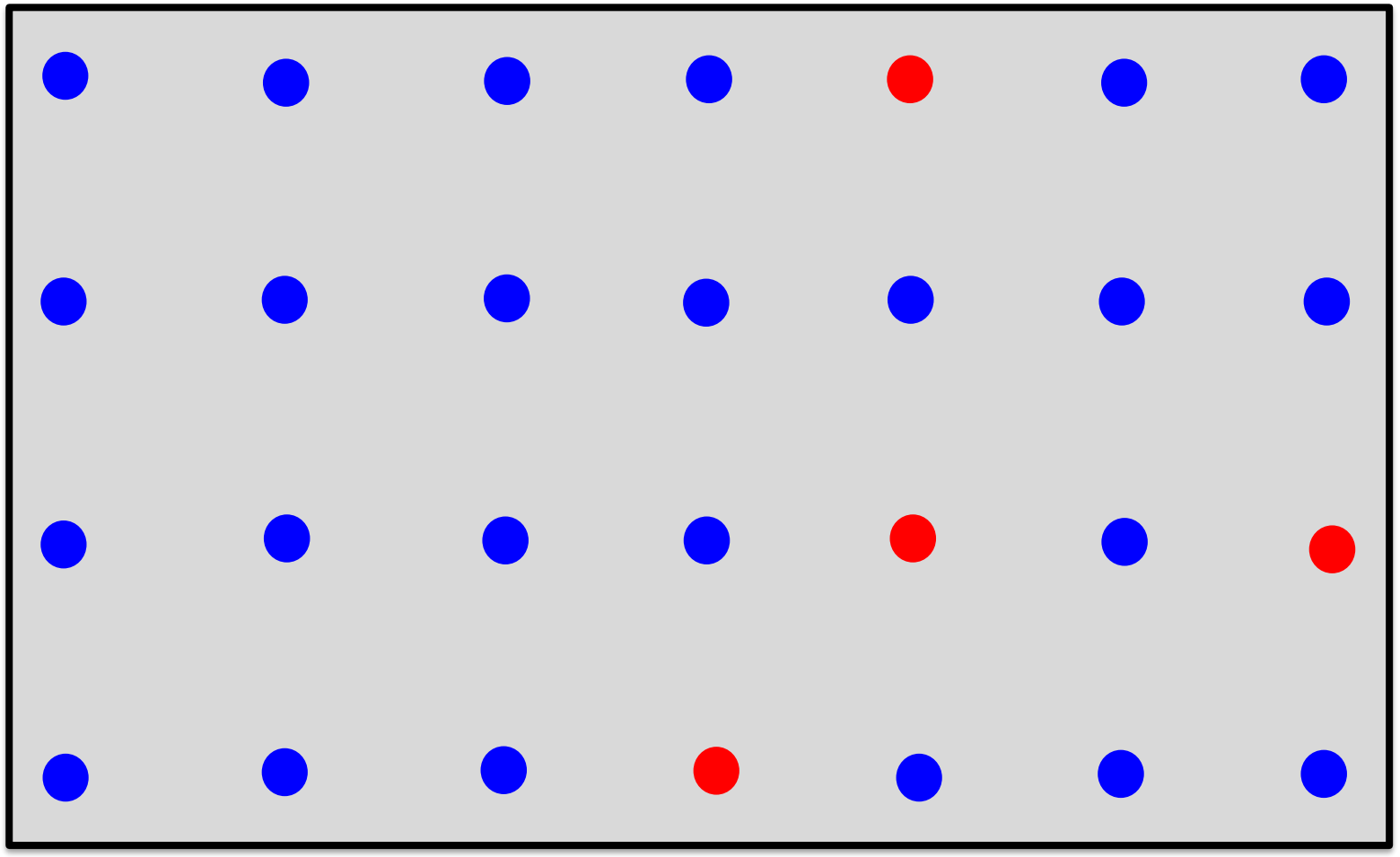
Y. Huang, S. Bhunia, P. Mishra, Scalable Test Generation for Trojan Detection using Side Channel Analysis, IEEE Trans. on Information Forensics & Security (TIFS), 2018.

Multiple Excitation of Rare Switching (MERS)



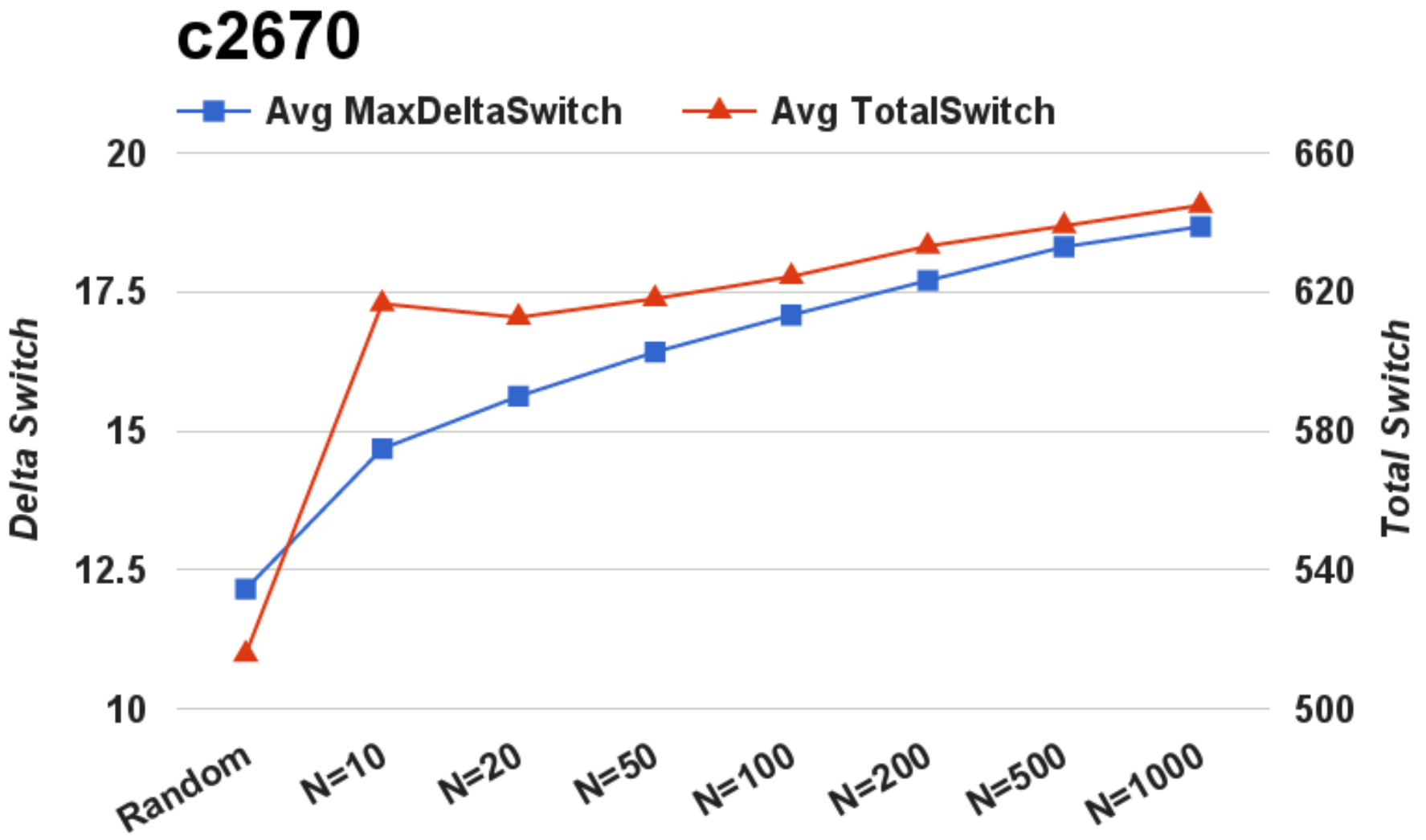
MERS: Test Reordering

Before reordering:



Non-rare node Rare node

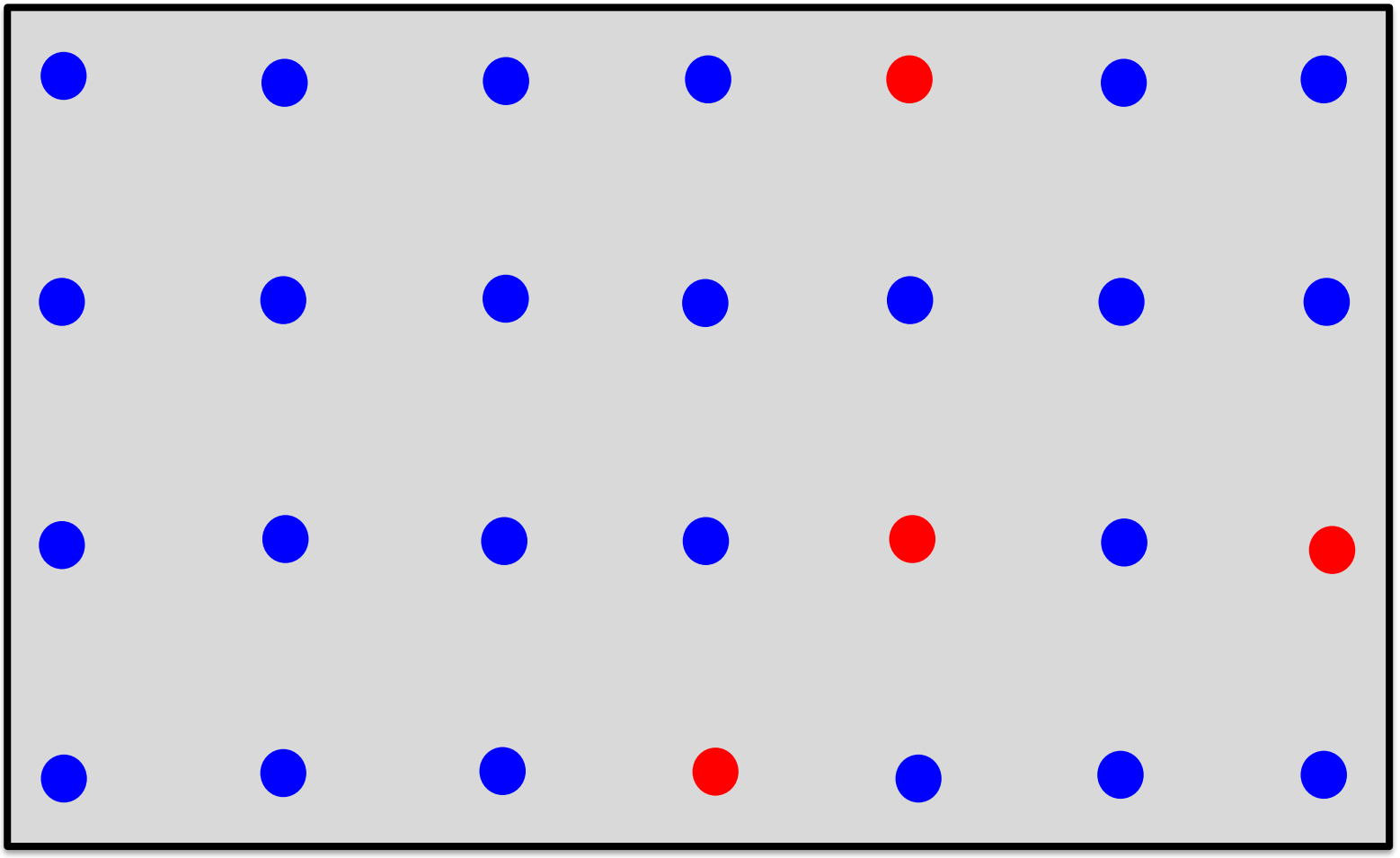
Effect of Increased Total Switching



Total Switch also increases with N =====> Tests reordering

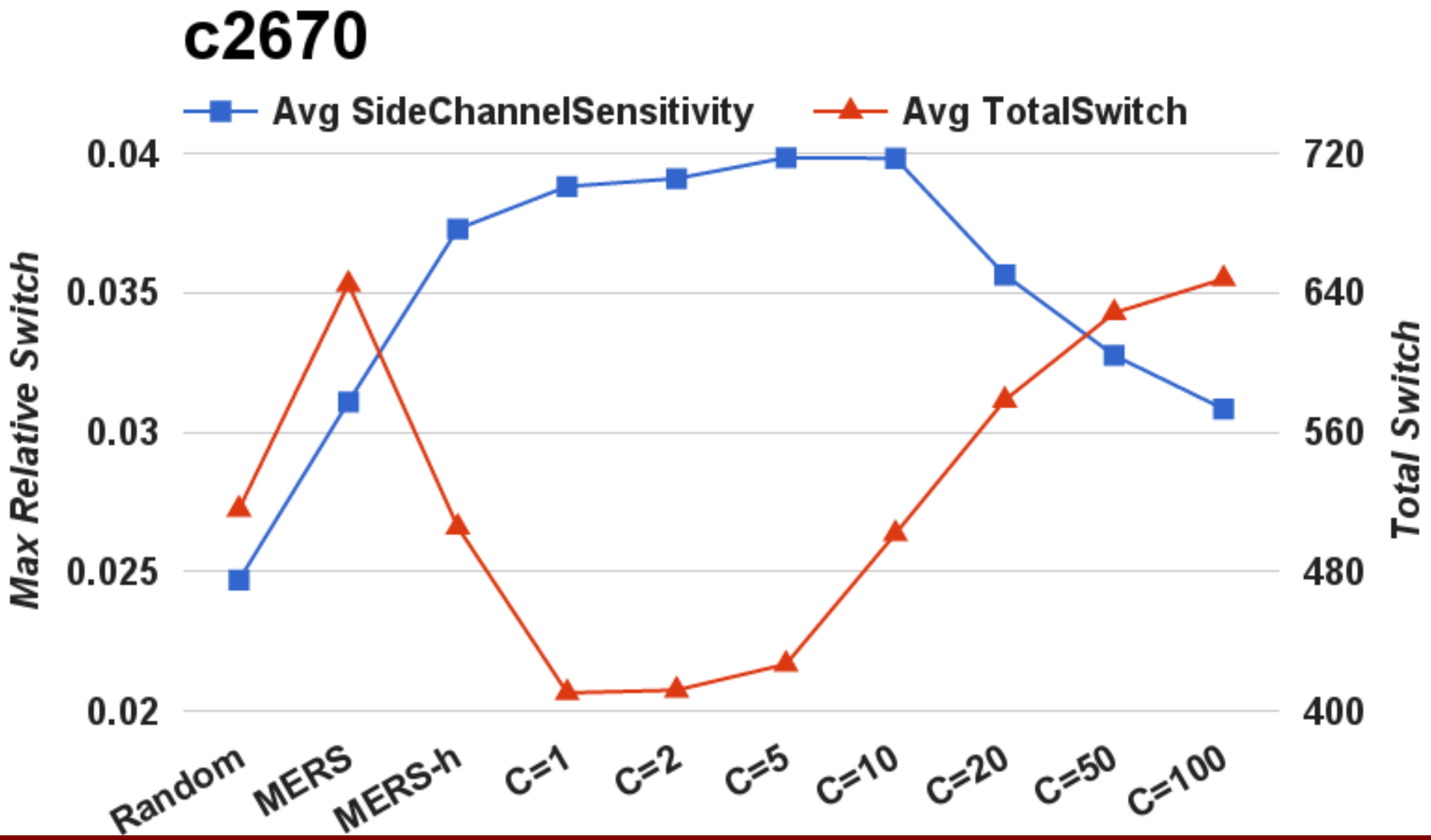
MERS: Test Reordering

After reordering:



Non-rare node Rare node

Effect of Weight Ratio (C)



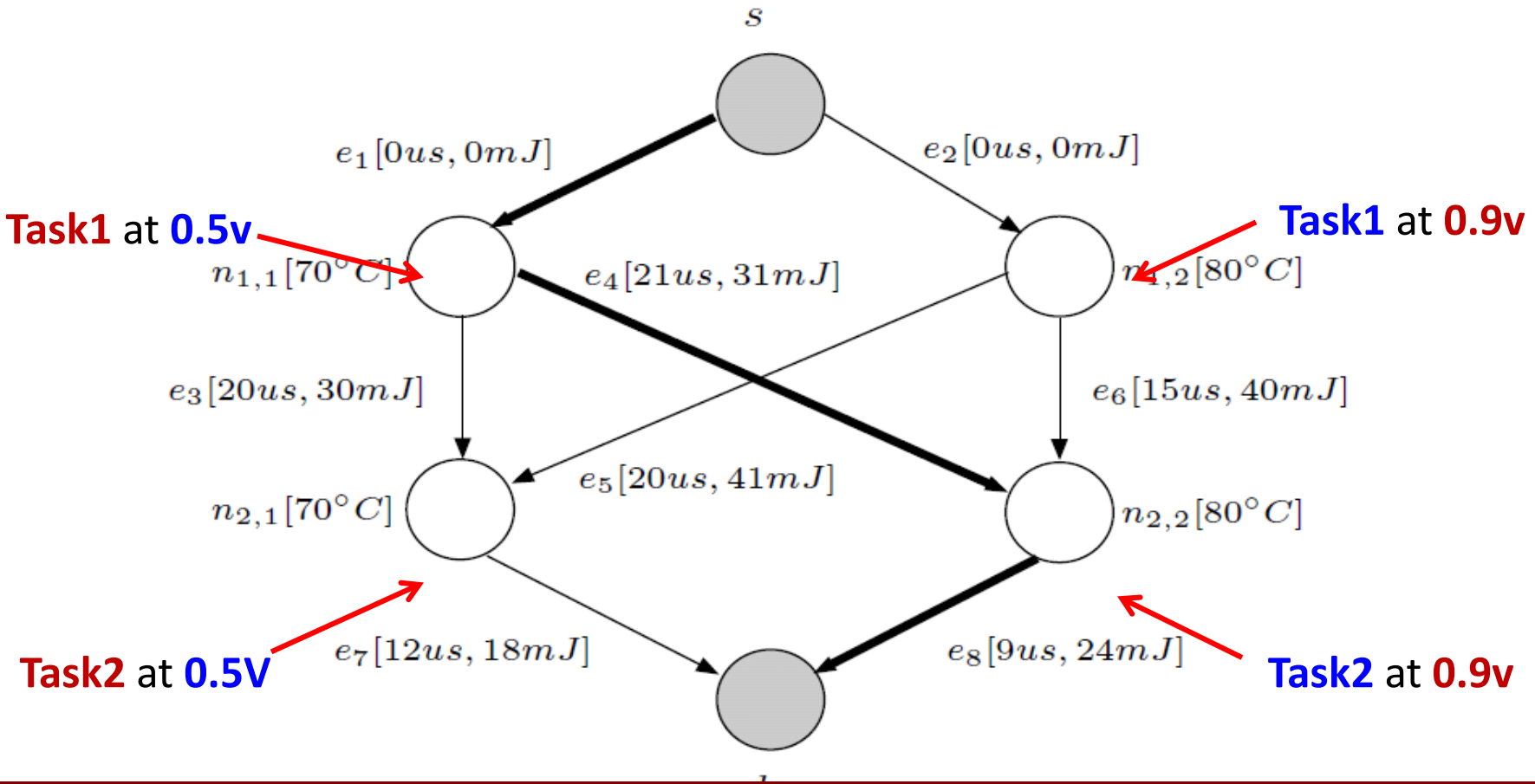
Y. Huang, S. Bhunia P. Mishra, MERS: Statistical Test Generation for Side-Channel Analysis based Trojan Detection, ACM Conf. on Computer and Communications Security (CCS), 2016.

Outline

- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
    - Property Checking of Unwanted Scenarios
    - Equivalence Checking to Identify Threats
    - Theorem Proving of Design Alternations
- Conclusion

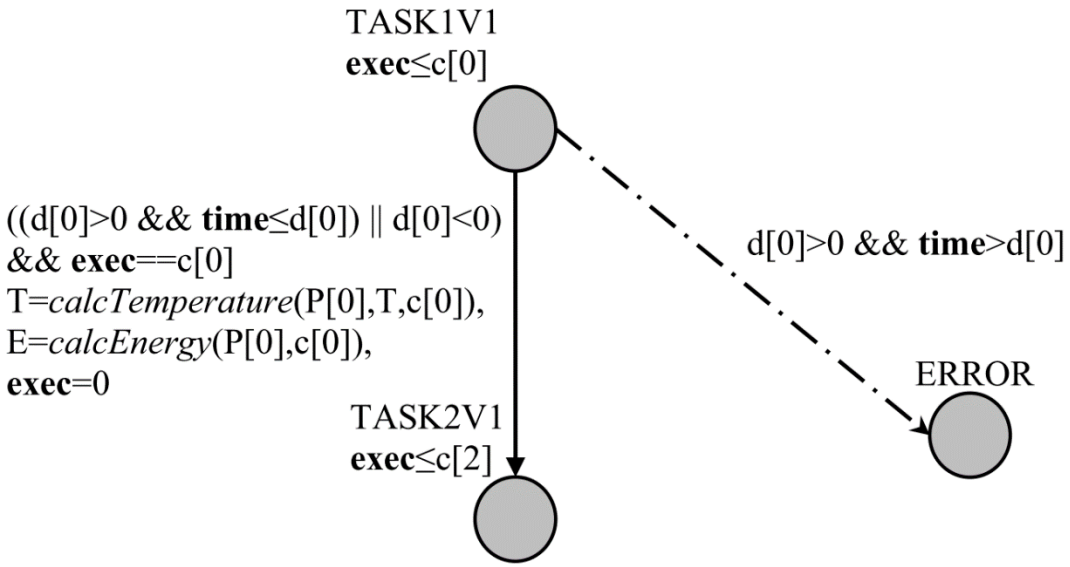
Checking Non-functional Properties

- Find a path that satisfies a specific property



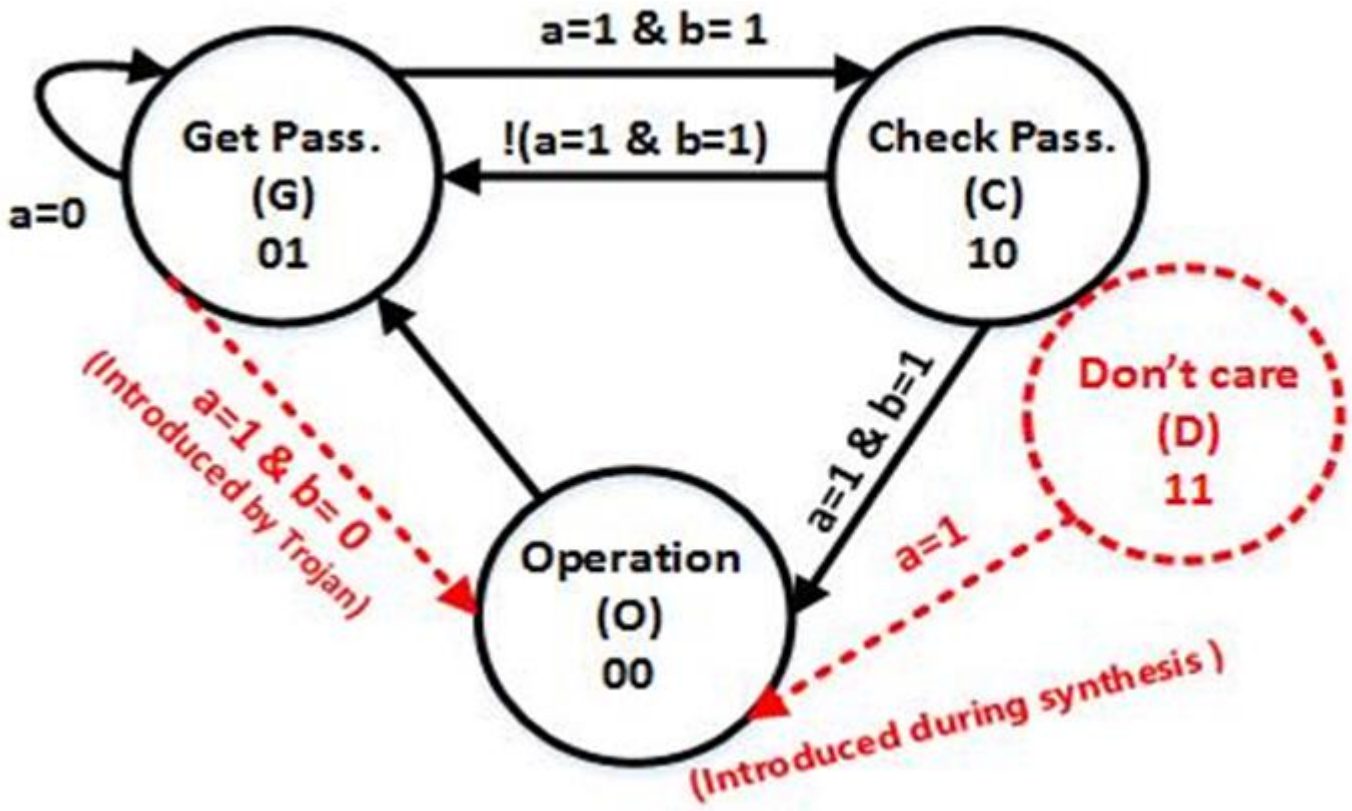
X. Qin, W. Wang and P. Mishra, Temperature- and Energy-Constrained Scheduling in Real-Time Multitasking Systems, IEEE Transactions on CAD (TCAD), 2012.

Model Checking using UPPAL



- TCEC requirement can be written in CTL as:
  - EG ((*T* < T_{max} ∧ *E* < E_{budget}) U *A.end*)
- If the model checker does not support "until":
  - EF (*isTSafe* ∧ *isESafe* ∧ *A.end*)
- In UPPAAL's property description language:

FSM Anomaly Detection



A. Nahiyan et al., Security-aware FSM Design Flow for Identifying and Mitigating Vulnerabilities to Fault Attacks, IEEE Transactions on CAD (TCAD), 2018.

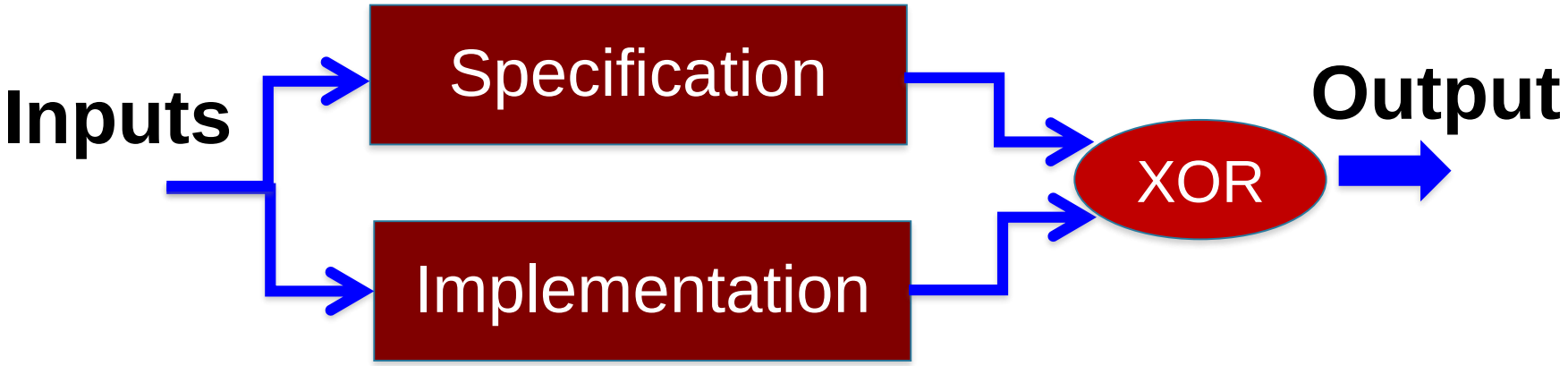
F. Farahmandi and P. Mishra, FSM Anomaly Detection using Formal Analysis, IEEE International Conference on Computer Design (ICCD), 2017.

Outline

- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
    - Property Checking of Unwanted Scenarios
    - Equivalence Checking to Identify Threats
    - Theorem Proving of Design Alternations
- Conclusion

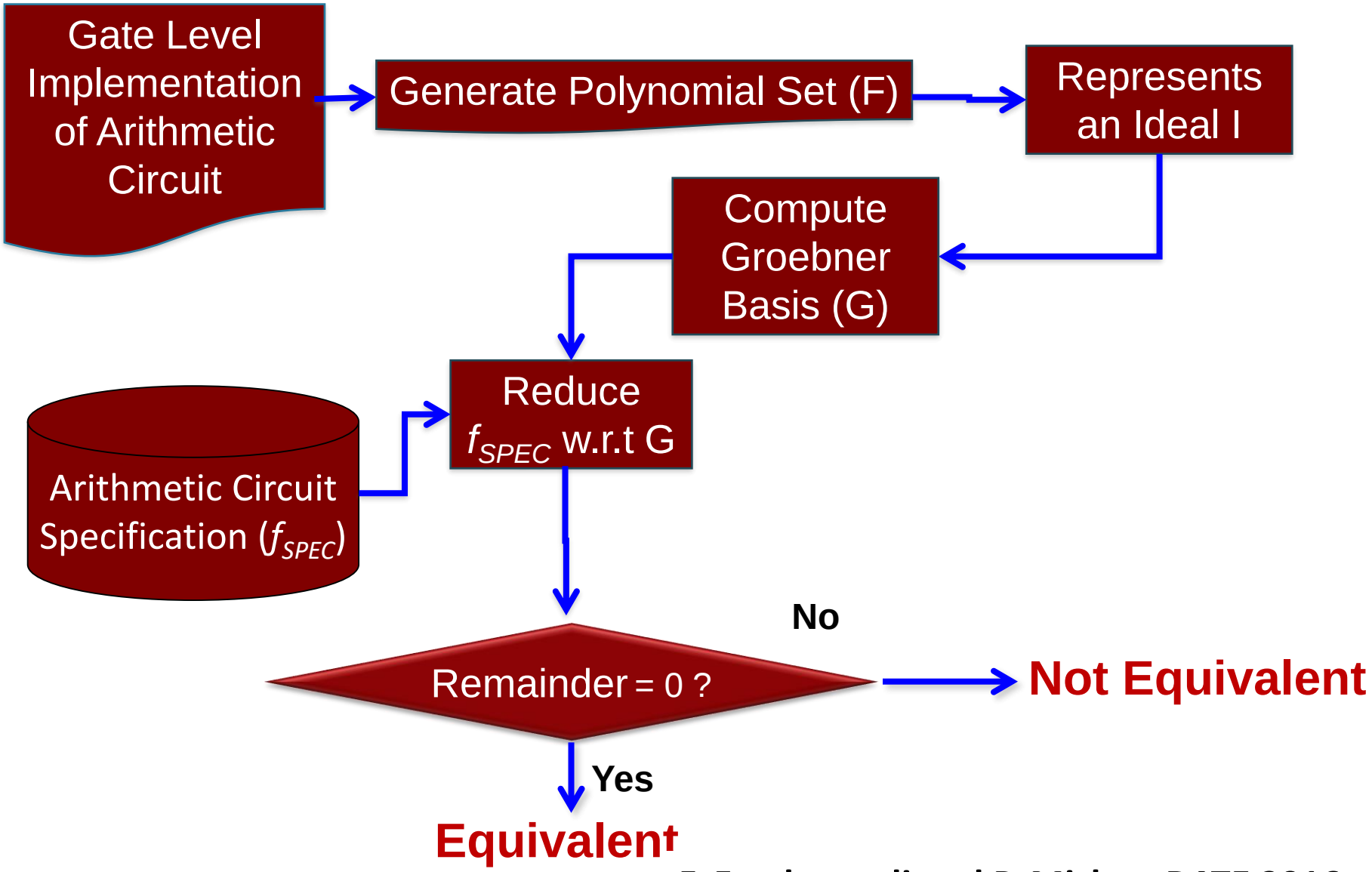
Equivalence Checking

- Traditional Equivalence Checkers
- Equivalence Checking using SAT Solvers



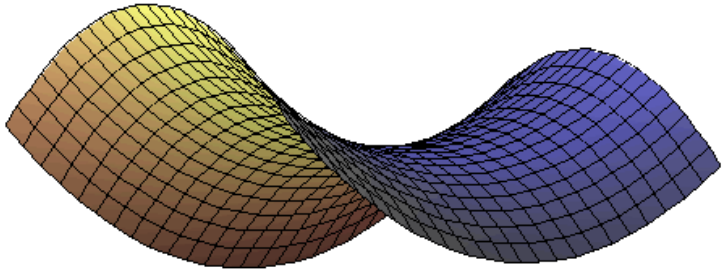
- Does not work for industrial designs unless the design structure (FSM) is similar

Groebner Basis and Polynomials



F. Farahmandi and P. Mishra, DATE 2016.

Groebner Basis



- Let **K** be a computable field
  - **K**$[x_1, x_2, \ldots, x_n]$ be the polynomial ring in n variables
- Polynomial f ∈ $K[x_1, x_2, \ldots, x_n]$ is written as

$$f = c_1 M_1 + c_2 M_2 + \cdots + c_d M_d$$

- Ideal **I** is represented by

$$< f_1, f_2, \ldots, f_s > = \{ \sum_{i=1}^{s} h_i f_i : h_1, h_2, \ldots, h_s \in K[x_1, x_2, \ldots, x_n] \}$$

- F = $\{f_1, f_2, \ldots, f_s\}$ is called generator or basis of ideal **I**
- Every arbitrary ideal other than {0} has a basis with specific properties which is called **Groebner basis**

Ideal Membership Algorithm

- *The set **G** Groebner basis of ideal **I** if and only if*
  - *For all polynomial $f \in I$ the remainder of reducing f w.r.t polynomials of G is zero*
  - *Reduction is a sequential division of f on set G with respect to a specific order*

- Groebner Basis has to be computed

Integer Polynomial of Logical Gates

- *Every Boolean variable a can be consider as*
  - $\mathbf{a} \in \{0,1\} \subset \boldsymbol{Z}$
  - $\boldsymbol{a^2 = a}$
- *Every logical gate can be modeled with an integer polynomial*

$$z = NOT(a) \Rightarrow \quad f = z - (1 - a) = \mathbf{0}$$

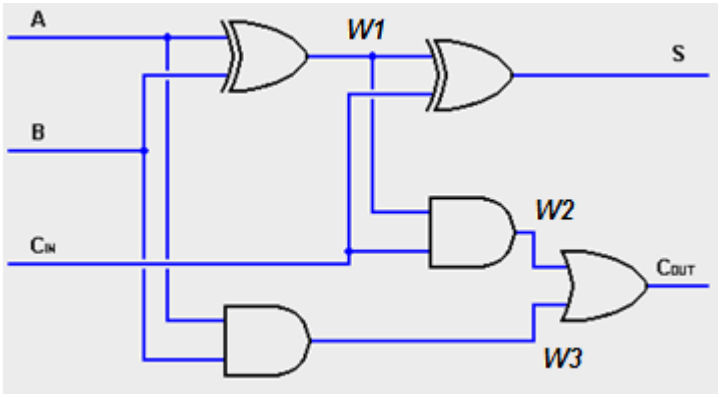
$$z = AND(a, b) \Rightarrow f = z - a.b = \mathbf{0}$$

$$z = OR(a, b) \Rightarrow f = z - (a + b - a.b) = \mathbf{0}\ z = XOR(a, b) \Rightarrow f = z - (a + b - 2.a.b) = \mathbf{0}$$

F. Farahmandi et al., Effective Combination of Algebraic Techniques and Decision Diagrams to Formally Verify Large Arithmetic Circuits, ISVLSI 2014.

Illustrative Example

$$f_{spec} := 2 * Cout + S - (A + B + Cin) = 0$$



Cout > S > { W3> W2} > W1> {A> B> Cin}

Circuit's Polynomials:

F = { W1 – (A+ B – 2*A*B) = 0,
W2 – (W1 * Cin) = 0,
W3 – (A*B) = 0,
S – (W1 +Cin – 2*W1*Cin) = 0,
Cout – (W2 + W3 - W2*W3) = 0 }

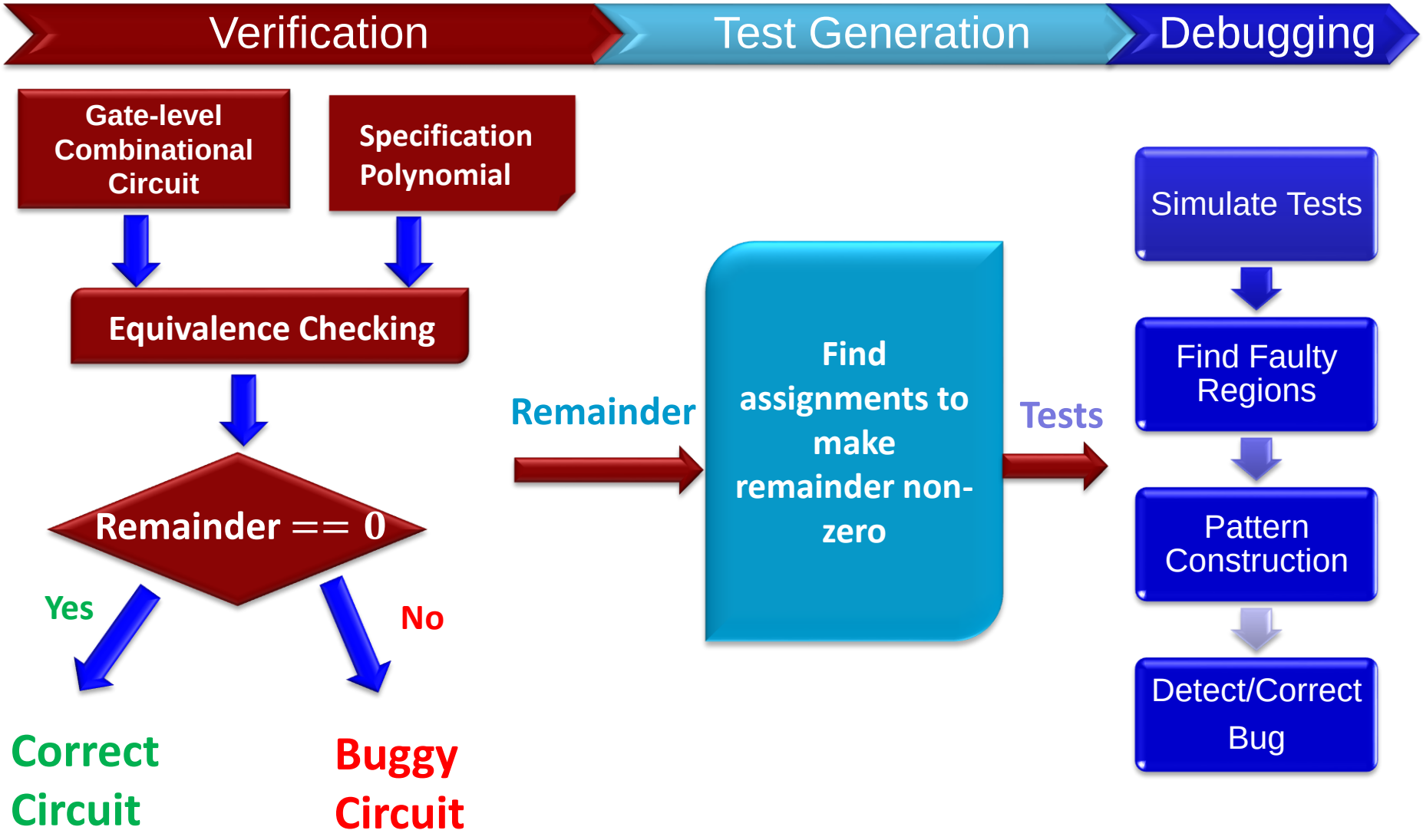
All circuit polynomials have relatively prime leading terms → F = G

Illustrative Example: Reduction Step

- Sequential Division with following order:
  - Cout ➔ S ➔ W3 ➔ W2 ➔ W1 ➔ Cin ➔ A ➔ B
- The dividend is
  - $f_{spec} := 2 * Cout + S - (A + B + Cin) = 0$
- Steps:
  - 1: cancle Cout with 2*(Cout – (W2 + W3 - W2*W3))
    - Remainder = S -2*W2*W3+ 2*W3+ 2*W2 - A –B - Cin
  - 2: cancle S with 1*(S – (W1 +Cin – 2*W1*Cin))
    - Remainder = -2*W2*W3+2*W3+ 2*W2+ – 2*W1*Cin+ W1 - A –B
  - 3: cancle W3 with (2-2W2) *(W3 – (A*B))
    - Remainder = 2*W2+2*W2*A*B – 2*W1*Cin + W1+2*A*B- A –B
  - 4: cancle W2 with (2 + 2*A*B)*(W2 – (W1 * Cin))
    - Remainder = 2*A*B*C*W1+W1+ 2*A*B -A - B
  - 5: cancle W1 with (2*A*B*C+1) * (W1 – (A+ B – 2*A*B))
    - Remainder =0

The design has correctly implemented the specification

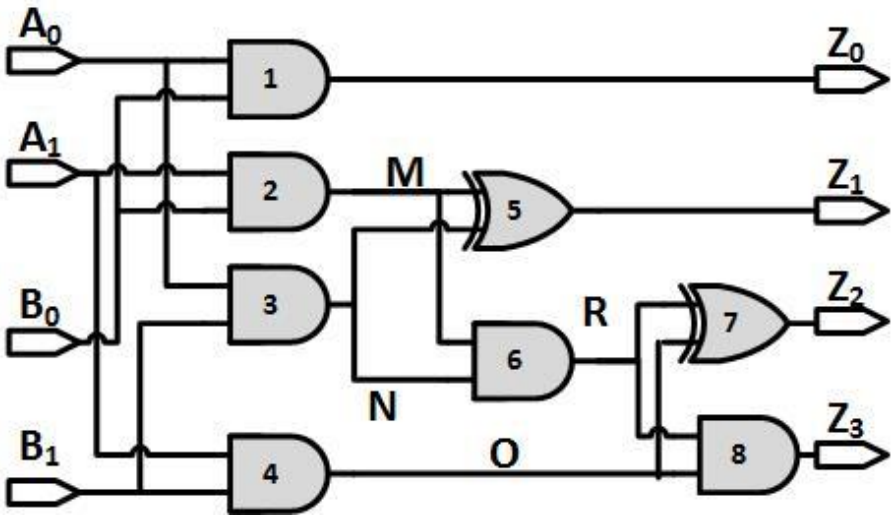
Automated Detection and Correction



F. Farahmandi and P. Mishra, Automated Test Generation for Debugging Multiple Bugs in Arithmetic Circuits, IEEE Transactions on Computers (TC), 68(2), 182-197, 2019.

Example (Correct Implementation)

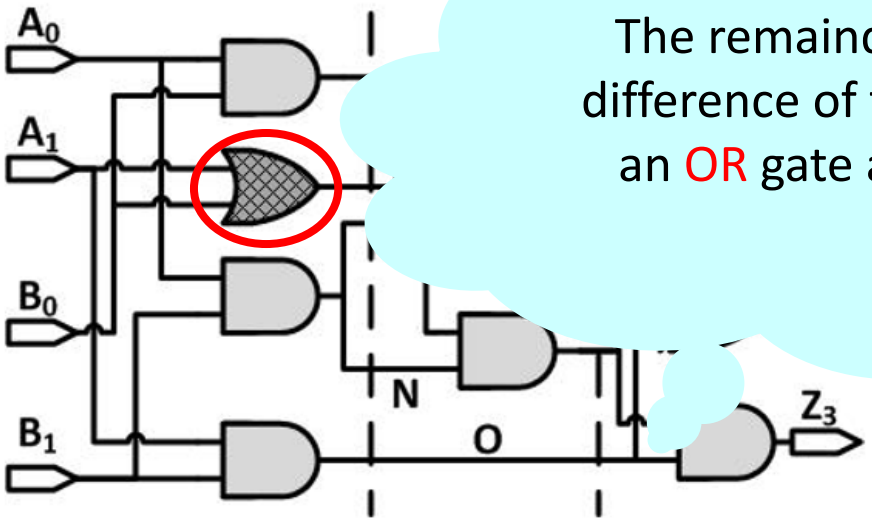
- Consider a 2-bit Multiplier
  - $f_{spec} := Z - (A.B)$
  - Order: $\{Z_2, Z_3\} > \{Z_1, R\} >$
  - $\{Z_0, \mathrm{M}, \mathrm{N}, \mathrm{O}\} > \{A_1, A_0, B_1, B_0\}$



- Verification Steps:
  - $f_{spec}: 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0 - 4.A_1.B_1 - 2.A_1.B_0 - 2A_0B_1 - A_0B_0$
- Cancel Z_2 and Z_3
  - Step1: $4.R + 4.O + 2.Z_1 + Z_0 - 4.A_1.B_1 - 2.A_1.B_0 - 2A_0B_1 - A_0B_0$
- Cancel R and Z_1
  - Step 2: $4.O + 2.M + 2.N + Z_0 - 4.A_1.B_1 - 2.A_1.B_0 - 2A_0B_1 - A_0B_0$
- Cancel $Z_0, \mathrm{M}, \mathrm{N}, \mathrm{O}$
  - Step 3: (remainder): 0

Example (Buggy Implementation)

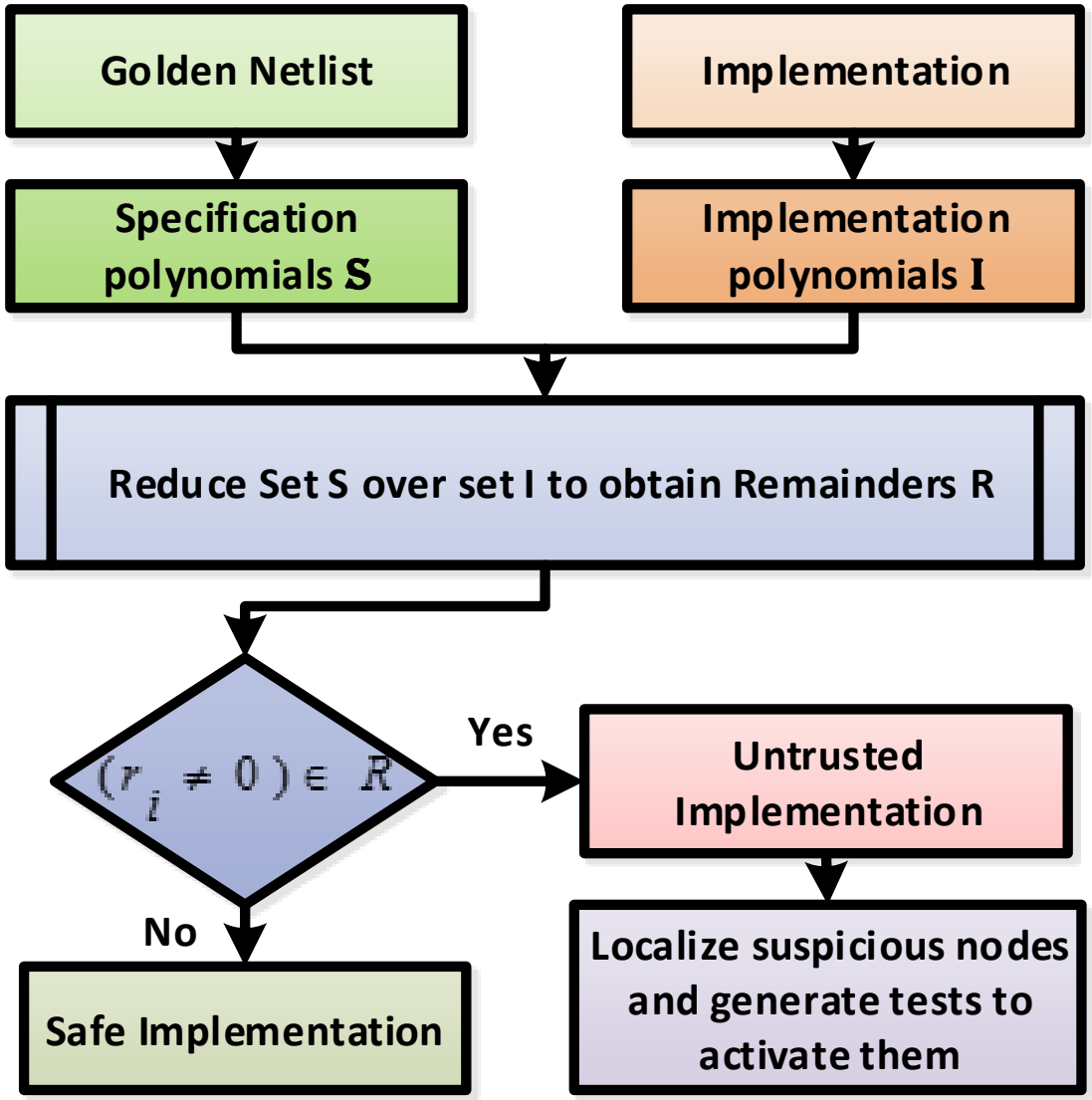
- Consider a buggy 2-bit Multiplier
  - $f_{spec} := Z - (A.B)$
  - $f_{spec} := 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0 - ((2.A_1 + A_0).(2.B_1 + B_0))$



The remainder shows the difference of functionality of an OR gate and AND gate

$$f_{spec_0} : 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_1} : 4.R + 4.O + 2.z_1 + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_2} : 4.O + 2.M + 2.N + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_3}(remiander) : 2.A_1 + 2.B_0 - 4.A_1.B_0$$

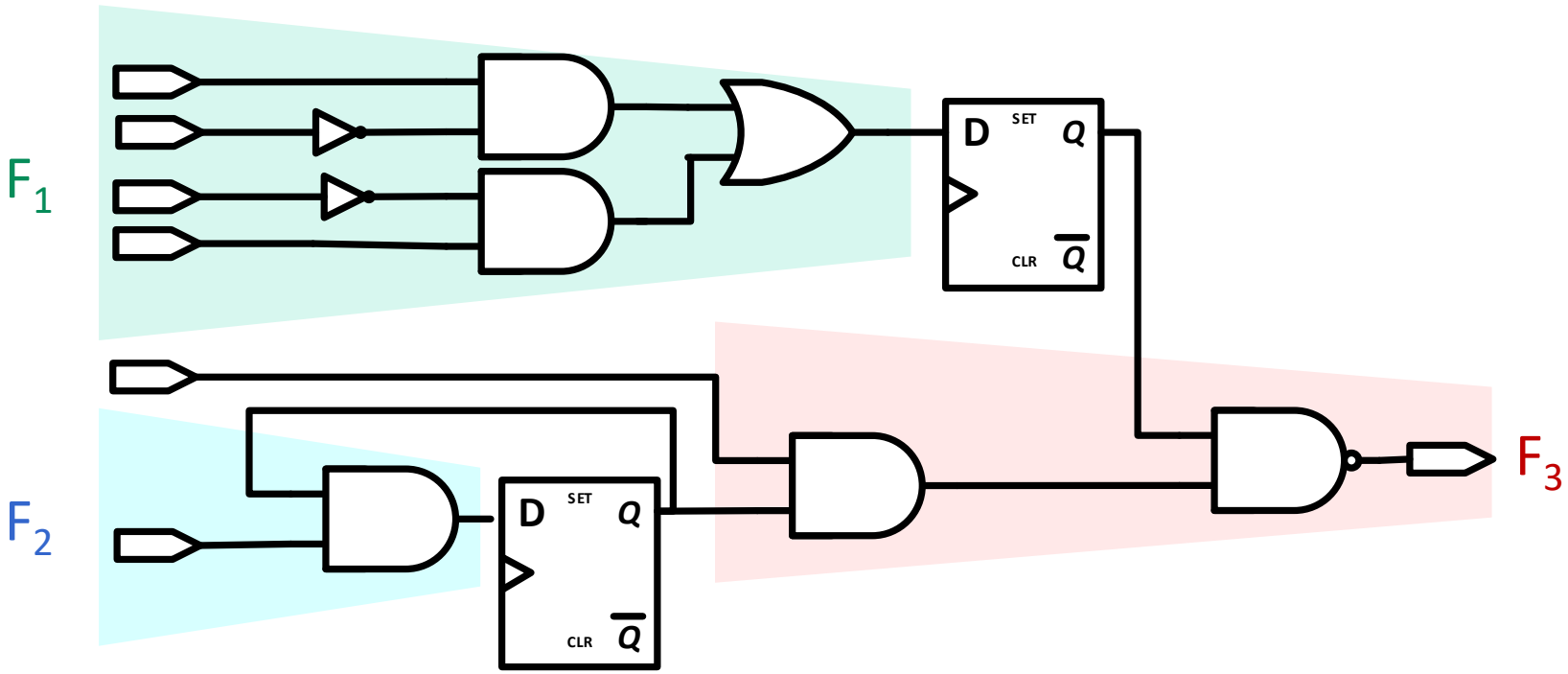
Trojan Detection using Polynomials



F. Farahmandi, Y. Huang and P. Mishra, Trojan Localization using Symbolic Algebra, Asia and South Pacific Design Automation Conference (ASPDAC), 2017.

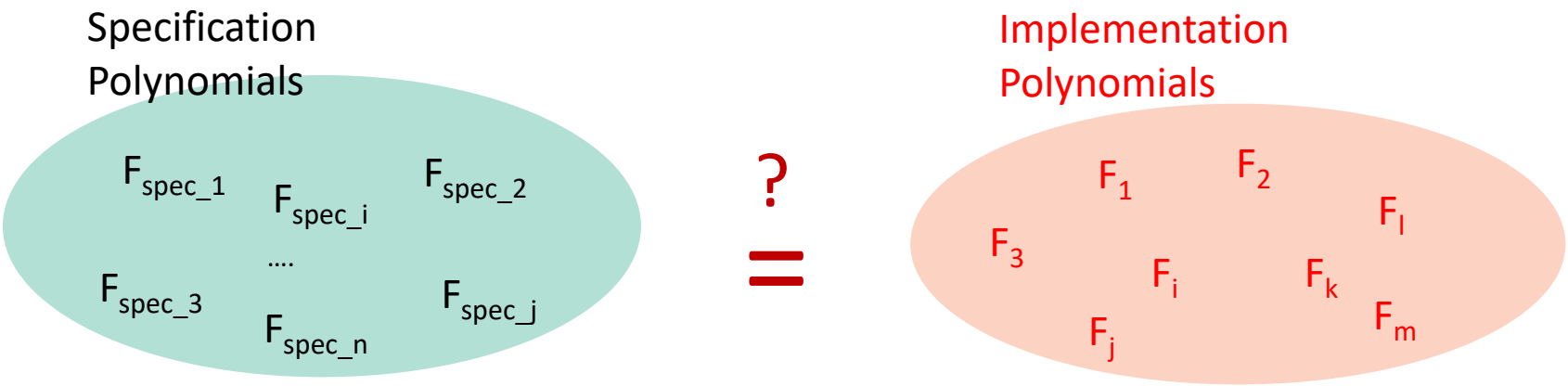
Model Specification and Implementation to Polynomials

- Partition Specification and Implementation Netlists to combinational regions
  - Model each region as a one Polynomial



Equivalence Checking

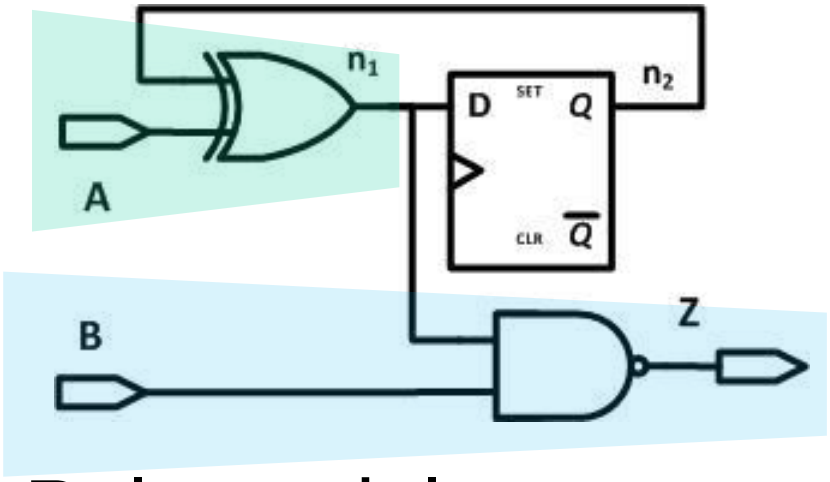
- Reduce each F_{spec_i} over corresponding implementation polynomials



- $F_{spec\_i} = C_s*F_s + C_{s+1}*F_{s+1} + \ldots + C_k*F_k + r_i$
  - If r_i is zero, implementation polynomials safely implement the function F_{spec_i}
  - Corresponding gates of implementations are safe
  - If r_i is non-zero, Malfunctions exist
  - There are som untrustworthy gates

Example: Extracting Specification Polynomials

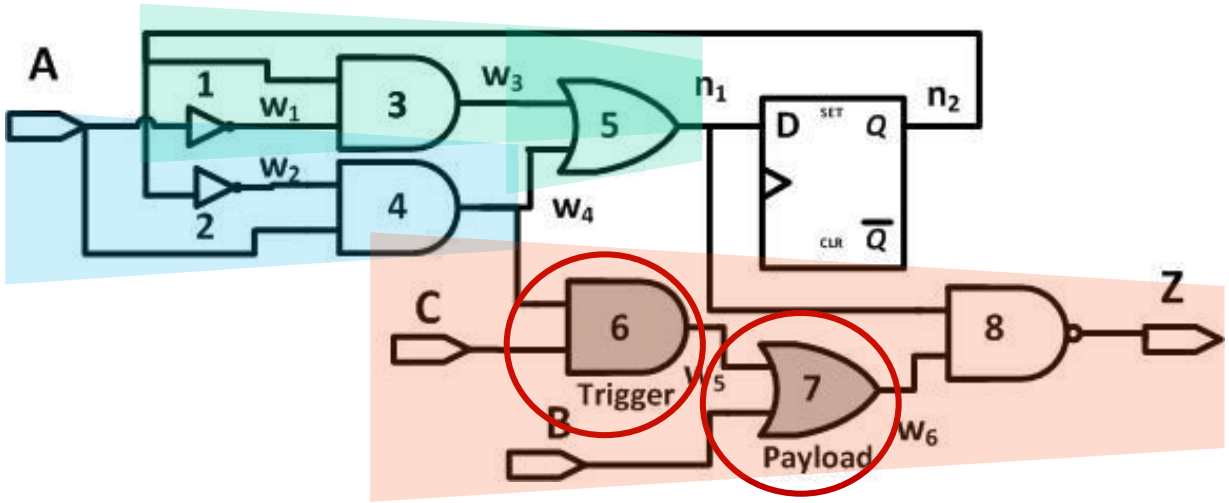
- Part of specification netlist



- Specification Polynomials:
  - F_{spec1}: $n_1 - (A+n_2-2*A*n_2)=0$
  - F_{spec2}: $Z - (n_1*B)=0$

Example: Extracting Implementation Polynomials

- Correspoonding part of implementation Netlist
  - Trojan is inserted

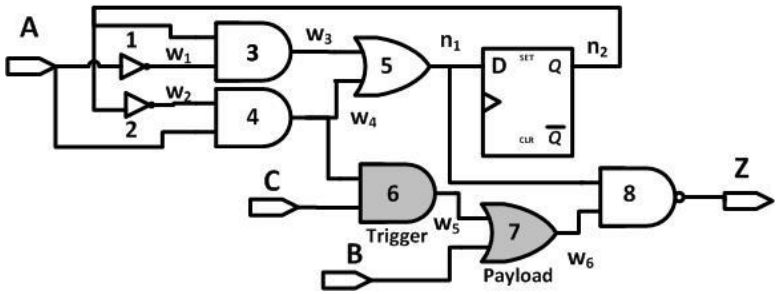


- Implementation polynomials
  - F_{spec1}: $n_1 - (n_2*w_4*A - n_2*w_4 + w_4 - n_2*A)=0$
  - F_{spec2}: $w_4 - (A - n_2*A)=0$
  - F_{spec3}: $Z - (n_1*w_4*C*B- + n_1*w_4*C- n_1*B+1)=0$

Example: Equivalence Checking

F_{spec1}: $n_1 - (A+n_2-2*A*n_2)=0$

F_{spec2}: $Z - (n_1*B)=0$



Specification

Implementation

$$f_{spec_1} : n_1 + 2.A.n_2 - n_2 - A$$
$$step_{11} : -1.w_3.w_4 + w_3 + w_4 + 2.n_2.A - n_2 - A$$
$$step_{12} : -1.w_2.w_1.n_2.A + n_2.w_1 + A.w_2 + 2.n_2.A - n_2 - A$$
$$step_{13}(r_1) : 0$$

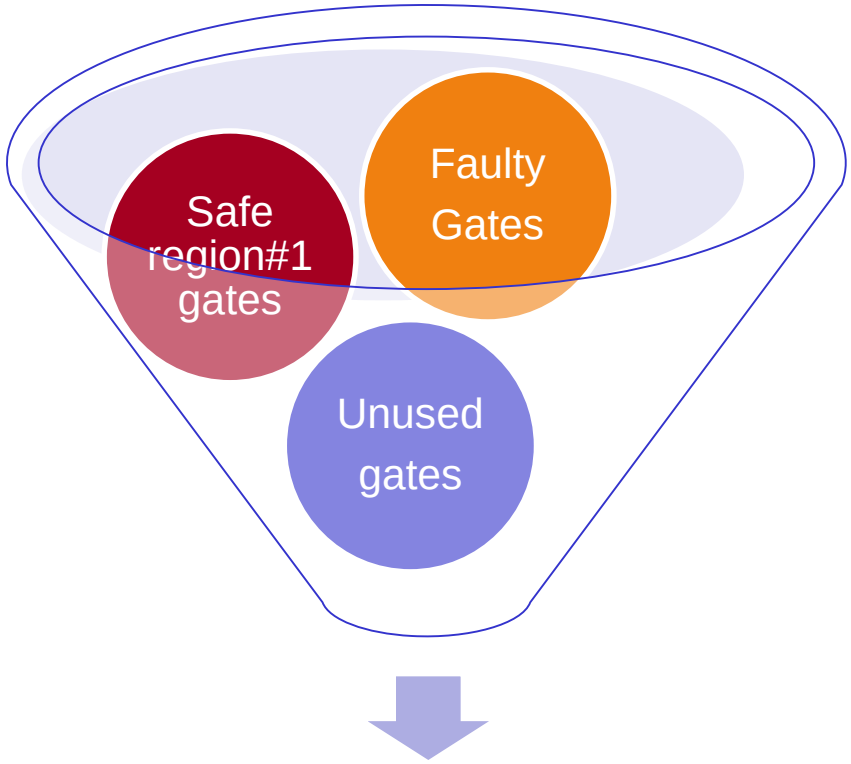
➔ Gates {1,2,3,4,5} which construct the F_{spec1} are safe

$$f_{spec_2} : Z + n_1.B - 1$$
$$step_{21} : -1.w_6.n_1 + n_1.B$$
$$step_{22} : -1.n_1.w_5 + B.n_1.w_5$$
$$step_{23} : -1.n_1.C.w_4 + B.n_1.C.w_4$$
$$step_{24} : -1.n_1.C.A.w_2 + B.n_1.C.A.w_2$$
$$step_{25}(r_2) : -1.n_1.A.C + n_1.n_2.A.C + A.B.C.n_1 - A.B.C.n_1.n_2$$

➔ Gates {2,4,6,7,8} which construct the F_{spec2} are suspicious

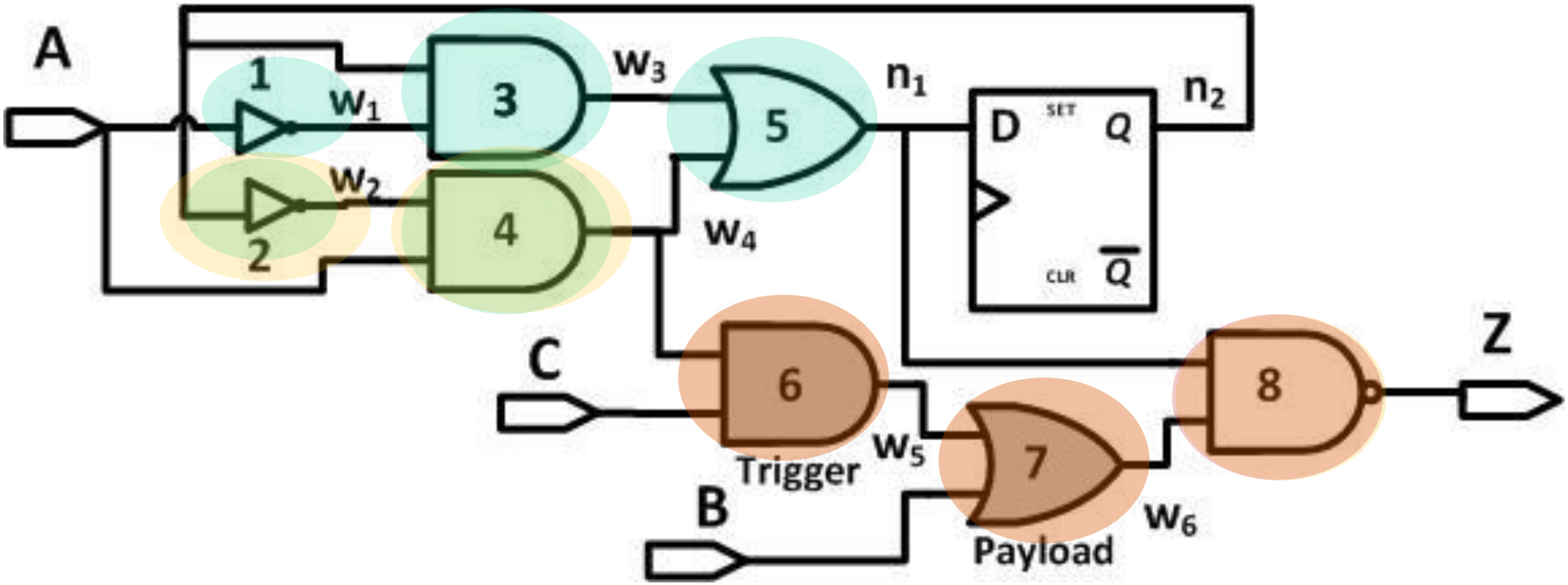
Trojan Localization

- Safe Gates G_S:
  - Which are contributing in generating zero remainders
- Faulty Gates G_F:
  - Which are contributing in generationg non-zero remainders
- Unused Gates G_U:
  - Extra gates that does not map to any of specification functionalities
- Potential Trojan Gates
  - $GT = G_F - G_S \cup G_U$



Example: Trojan Localization

- Safe Gates: {1,2,3,4,5}
- Faulty Gates: {2,4,6,7,8}
- Potential Trojan Gates: {6,7,8}



Results: Trojan Localization

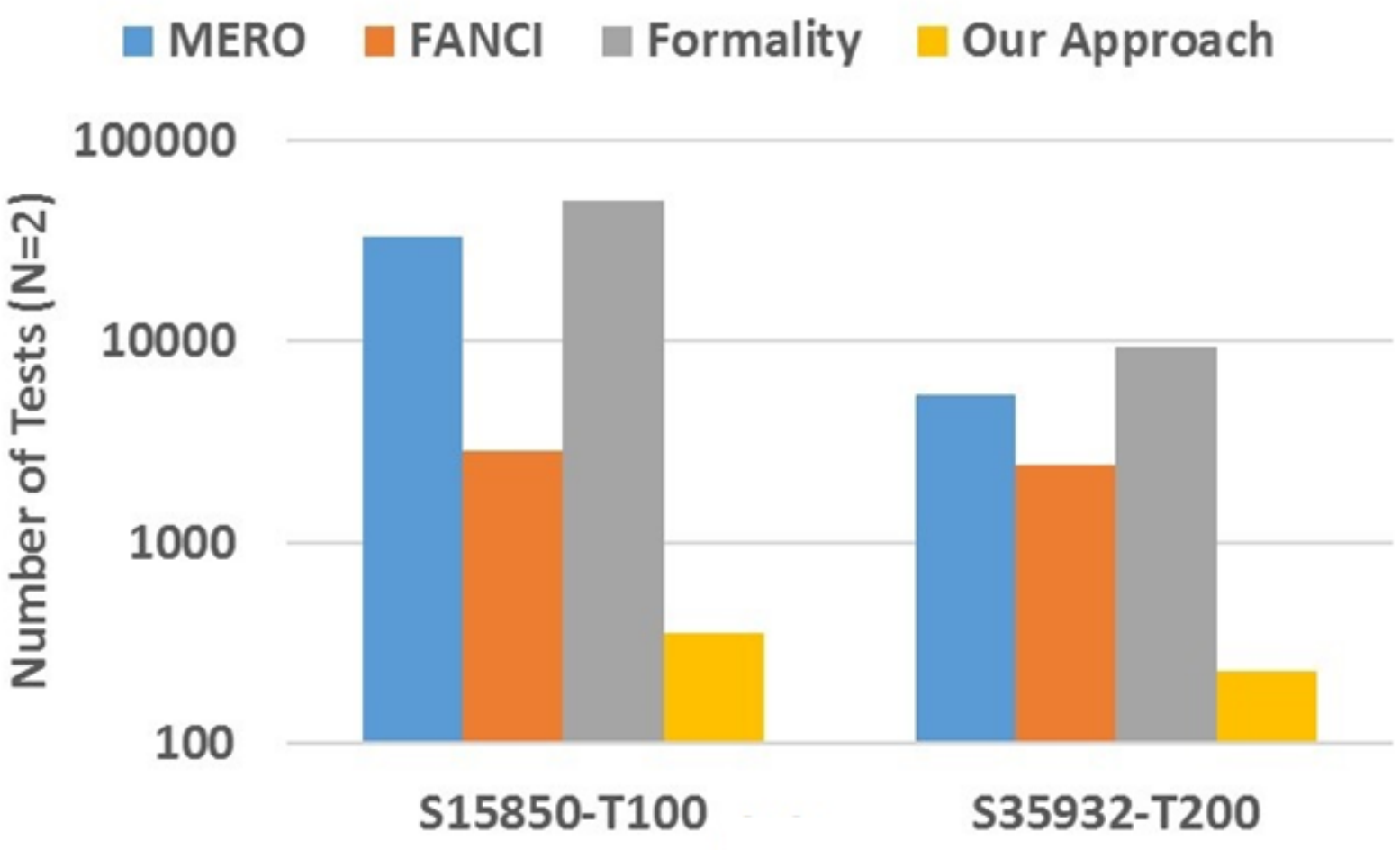
Benchmark			#Suspicious Gates			False Positives	False positive Improvement	
Type	Gates	#Trojan GAtes	FANCI	Formality	Ours	Our	FANCI	Formality
RS232-T1000	311	13	37	214	13	0	*	*
RS232-T1100	310	12	36	213	14	2	31x	100.5x
S15850-T100	2456	27	76	710	27	0	*	*
S38417-T100	5819	11	69	**	13	2	29x	**
S38417-T200	5823	15	73	2653	26	11	5.27x	240x
S35932-T200	5445	16	70	138	22	6	9x	20.3x
S38584-T200	7580	11	85	47	9	11	37.5x	23.5x
Vga-lcd-T100	70162	5	706	**	22	17	41x	**

"*" indicates our approach does not produce any false positive gates (infinite improvement)

"**" shows the cases that Formality could not detect the Trojans.

[FANCI] A. Waksman et al., CCS, 2013.

Trojan Activation



A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution, International Test Conference (ITC), 2018.

Outline

- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
    - Property Checking of Unwanted Scenarios
    - Equivalence Checking to Identify Threats
    - Theorem Proving of Design Alternations
- Conclusion

Results: Trojan Localization

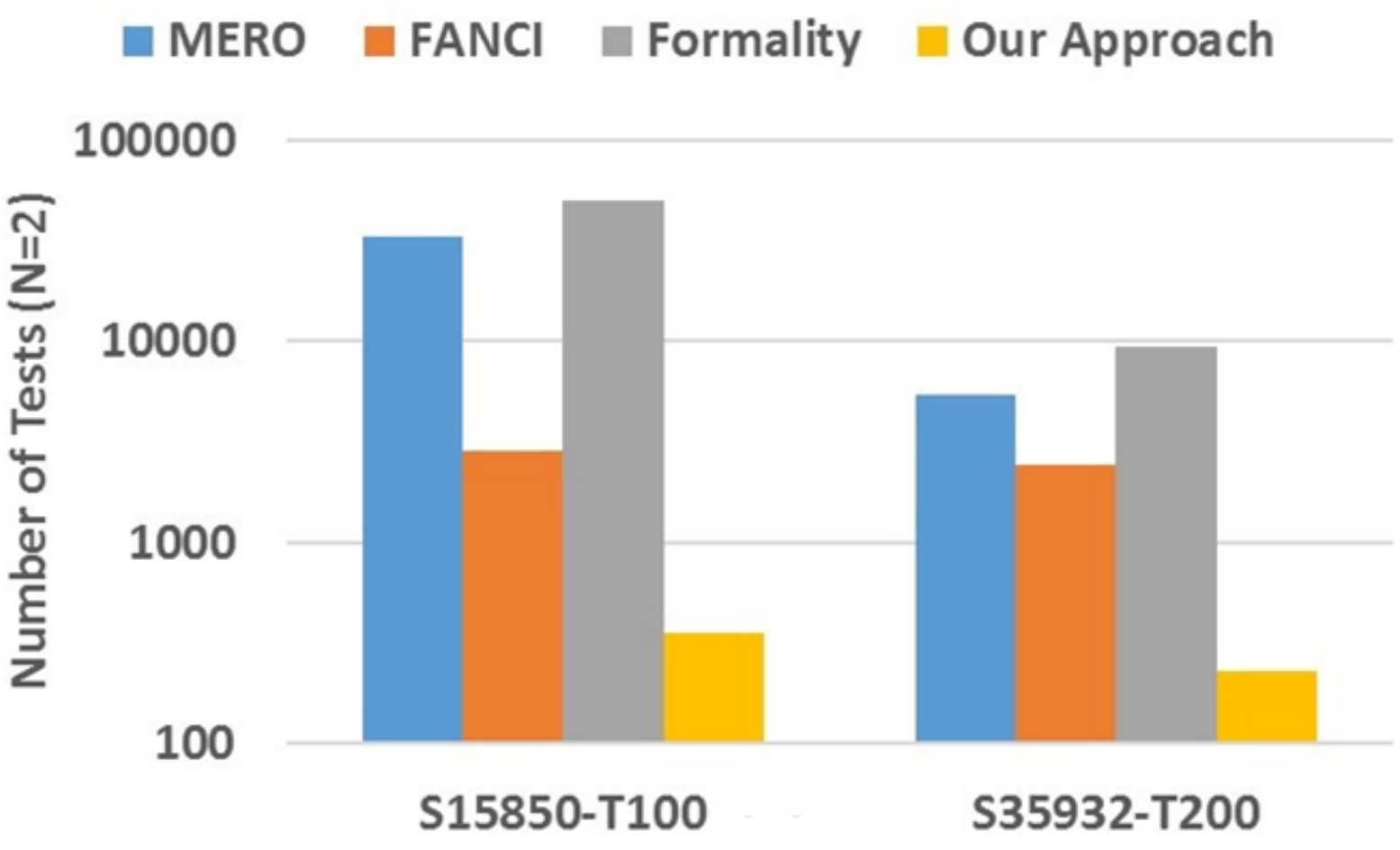
Benchmark			#Suspicious Gates			False Positives	False positive Improvement	
Type	Gates	#Trojan GAtes	FANCI	Formality	Ours	Our	FANCI	Formality
RS232-T1000	311	13	37	214	13	0	*	*
RS232-T1100	310	12	36	213	14	2	31x	100.5x
S15850-T100	2456	27	76	710	27	0	*	*
S38417-T100	5819	11	69	**	13	2	29x	**
S38417-T200	5823	15	73	2653	26	11	5.27x	240x
S35932-T200	5445	16	70	138	22	6	9x	20.3x
S38584-T200	7580	11	85	47	9	11	37.5x	23.5x
Vga-lcd-T100	70162	5	706	**	22	17	41x	**

"*" indicates our approach does not produce any false positive gates (infinite improvement)

"**" shows the cases that Formality could not detect the Trojans.

[FANCI] A. Waksman et al., CCS, 2013.

Trojan Activation



A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution, International Test Conference (ITC), 2018.

Outline

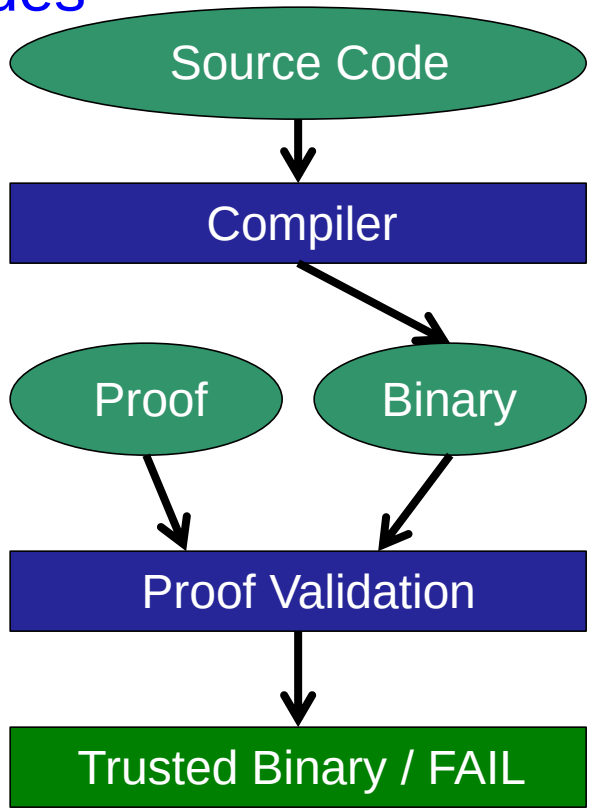
- Introduction
- Design for Security
- Simulation-based Validation
  - Test Generation for Trust Validation
  - Side Channel Analysis
- Formal Verification Approaches
  - Property Checking of Unwanted Scenarios
  - Equivalence Checking to Identify Threats
  - Theorem Proving of Design Alternations
- Conclusion

Theorem Proving

- **Theorem Proving:** Prove/disprove properties of systems expressed as logical statements
- **Types:** Automated Theorem Provers (SMT, SAT solvers) and Interactive Theorem Provers (Coq, NuPRL)
- **Advantage:** Verification of large hardware designs
- **Limitation:** Proof construction in interactive theorem provers could be tedious
- **Application:** Use of Coq in Proof-Carrying Hardware framework for verifying soft-IP cores

Proof-Carrying Code (PCC)

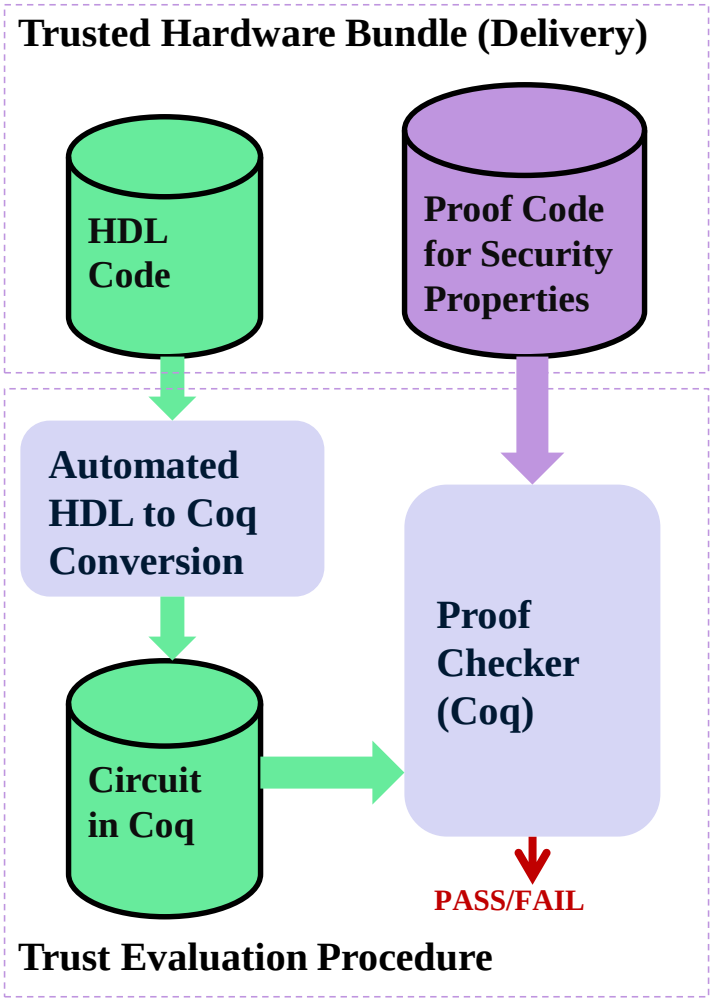
- Use formal proof to establish software trustworthiness
  - Developed by G. Necula and P. Lee in ’96
  - Central idea: supplier of software provides formal proof ensuring software’s safety
- Implementation Procedure
  - Compile Source
  - Write proof of specification for the binary code
  - Validate Proof
  - Execute



G. C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106-119, 1997.

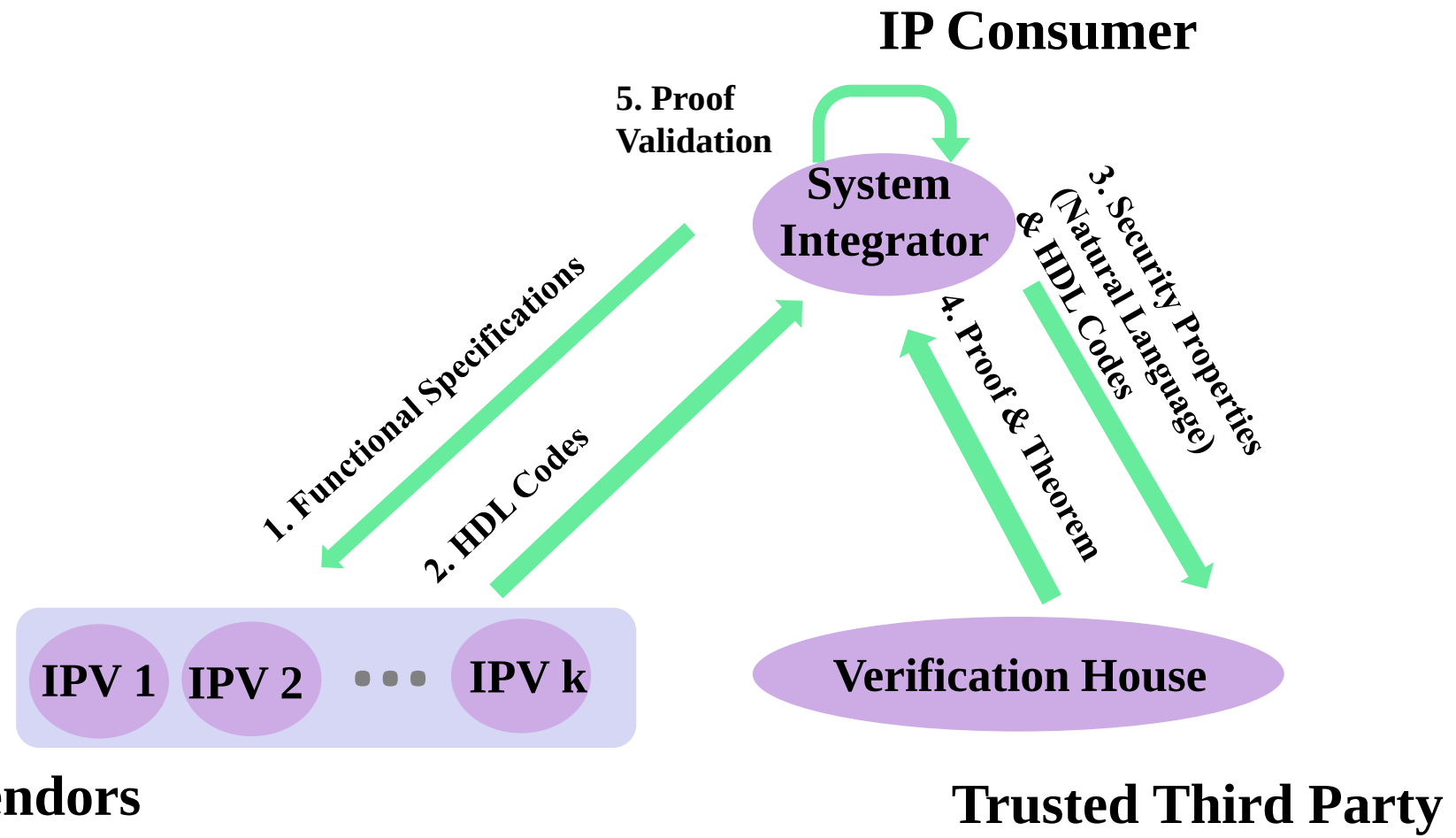
Proof-Carrying Hardware IP Cores

- Trusted IP Acquisition (consumers)
  - User receives IP code AND a formal proof regarding the code's trustworthiness
  - Existence of Proofs certify verification of HDL code against security properties
  - Proofs are validated automatically and efficiently by the proof checker in Coq
  - Unlike functional specifications, security properties concern both functionality and information sensitivity



X. Guo, R. Dutta, P. Mishra and Y. Jin, Scalable SoC Trust Verification using Integrated Theorem Proving and Model Checking, HOST, pages 124-129, 2016.

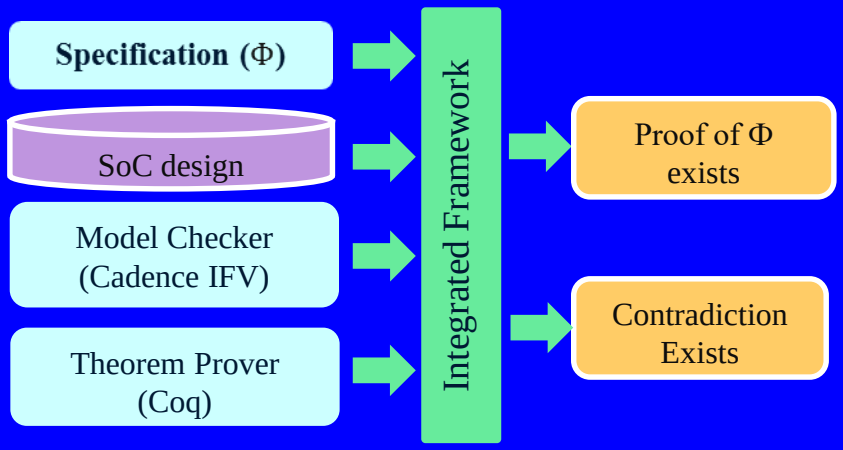
Working Procedure – Main Parties



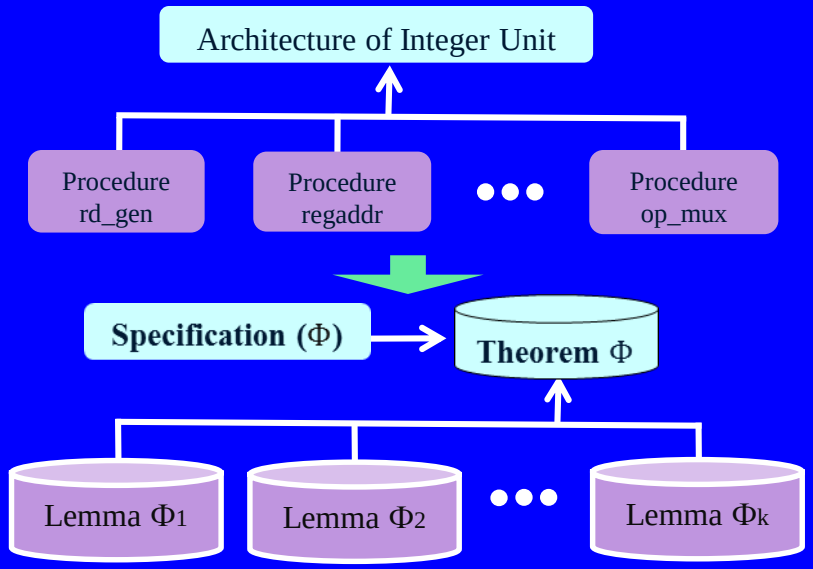
X. Guo, R. Dutta, P. Mishra and Y. Jin, Automatic Code Converter Enhanced PCH Framework for SoC Trust Verification, IEEE Transactions on VLSI, December 2017.

Scalable SoC Trust Verification using Integrated Theorem Proving and Model Checking

- Formal Methods Integration
  - Theorem Prover (TP) - Coq
  - Model Checker (MC) – Cadence IFV
  - First attempt to verify security properties on large-scale hardware by integrating TP and MC



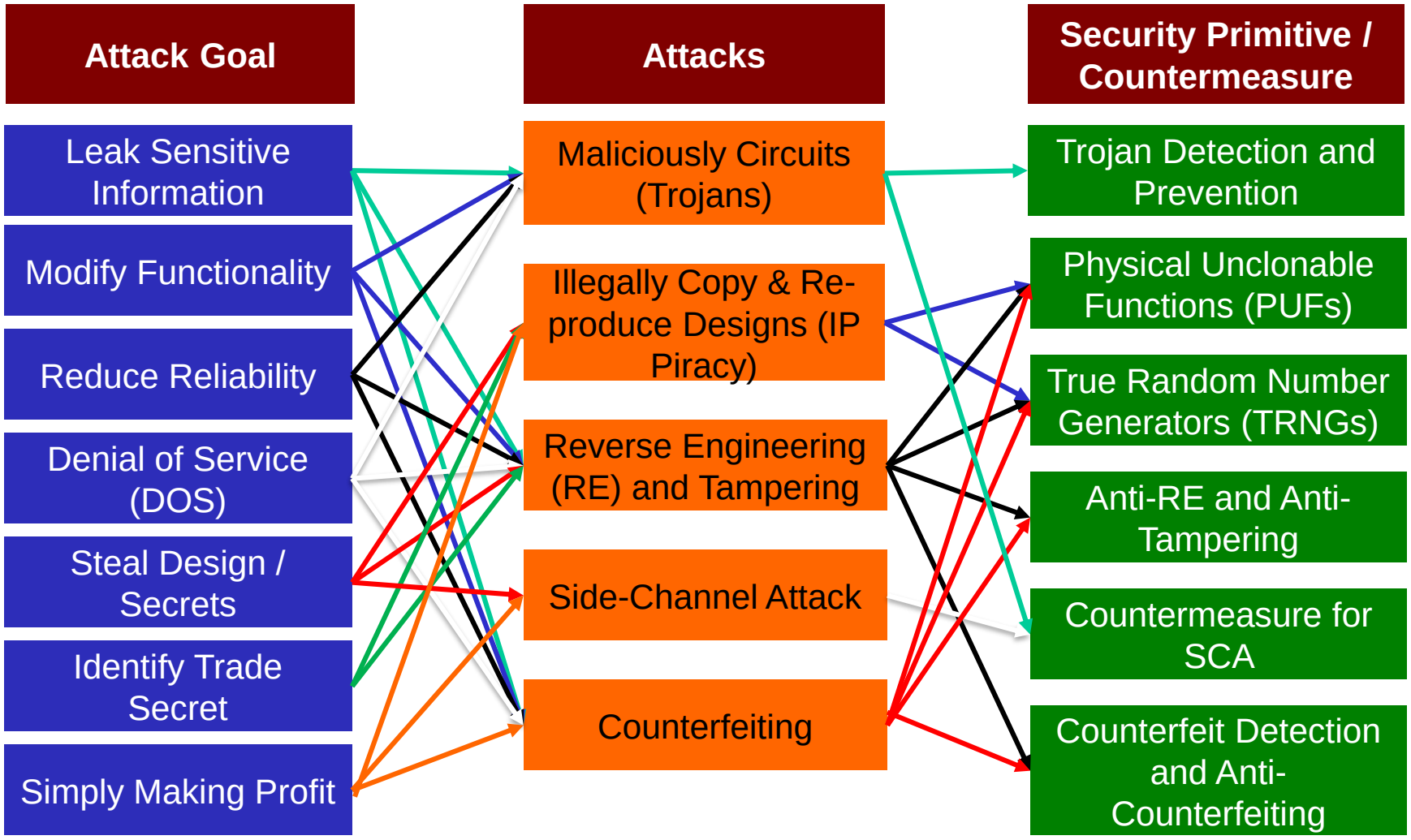
- Distributed Proof Construction
  - Decomposition of hardware design & security specification theorem
  - Sub-modules against lemmas of security properties
  - Prove security specification by combining results of lemmas of security properties



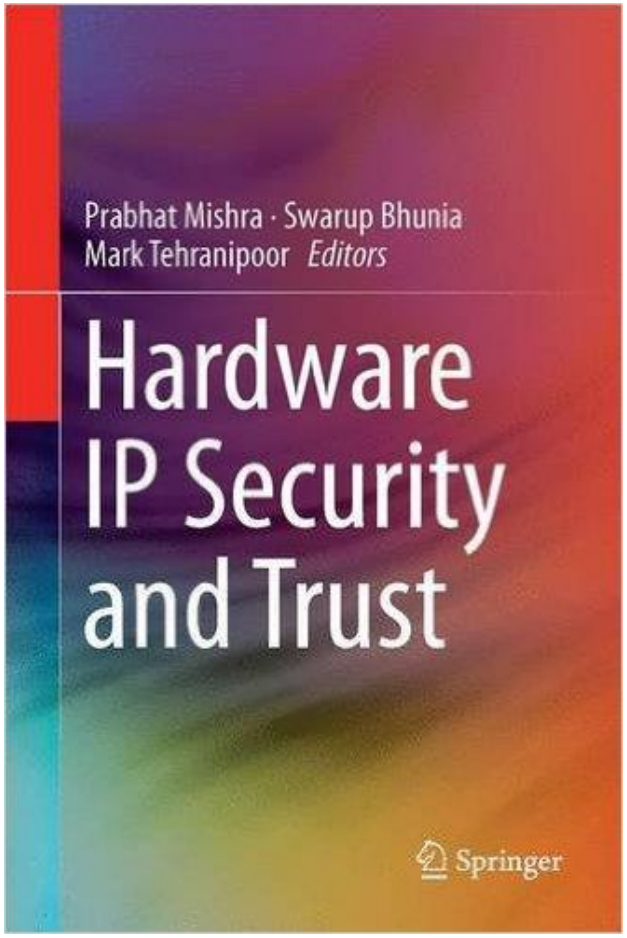
Outline

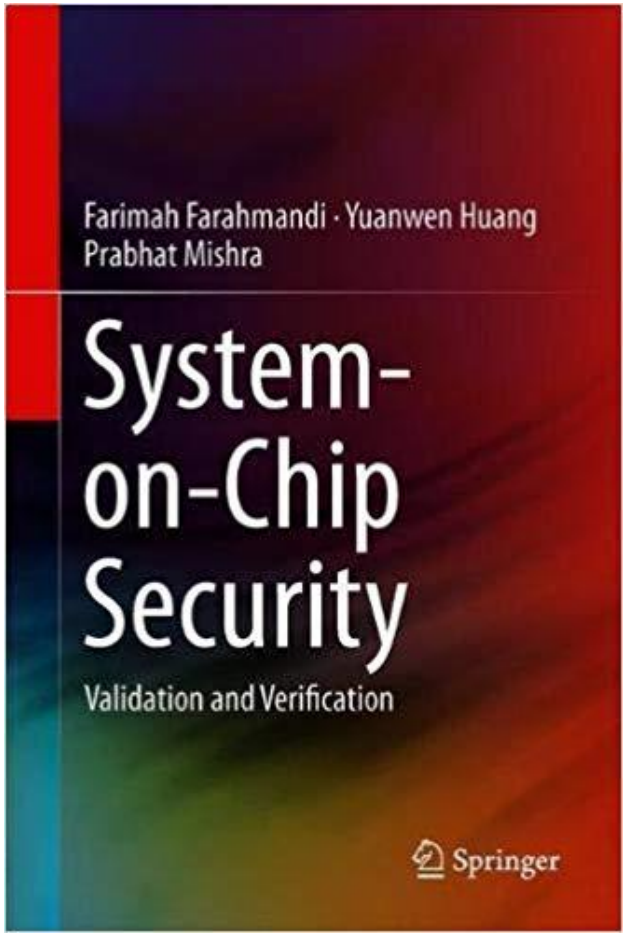
- Introduction
- Design for Security
- Security and Trust Validation
  - Simulation-based Validation
  - Side Channel Analysis
  - Formal Verification
- Analog/Mixed-Signal Validation
- Conclusion

Attacks and Countermeasures



Thank you!





prabhat@ufl.edu
http://www.cise.ufl.edu/~prabhat