Trojan Localization using Symbolic Algebra

Farimah Farahmandi, Yuanwen Haung and Prabhat Mishra

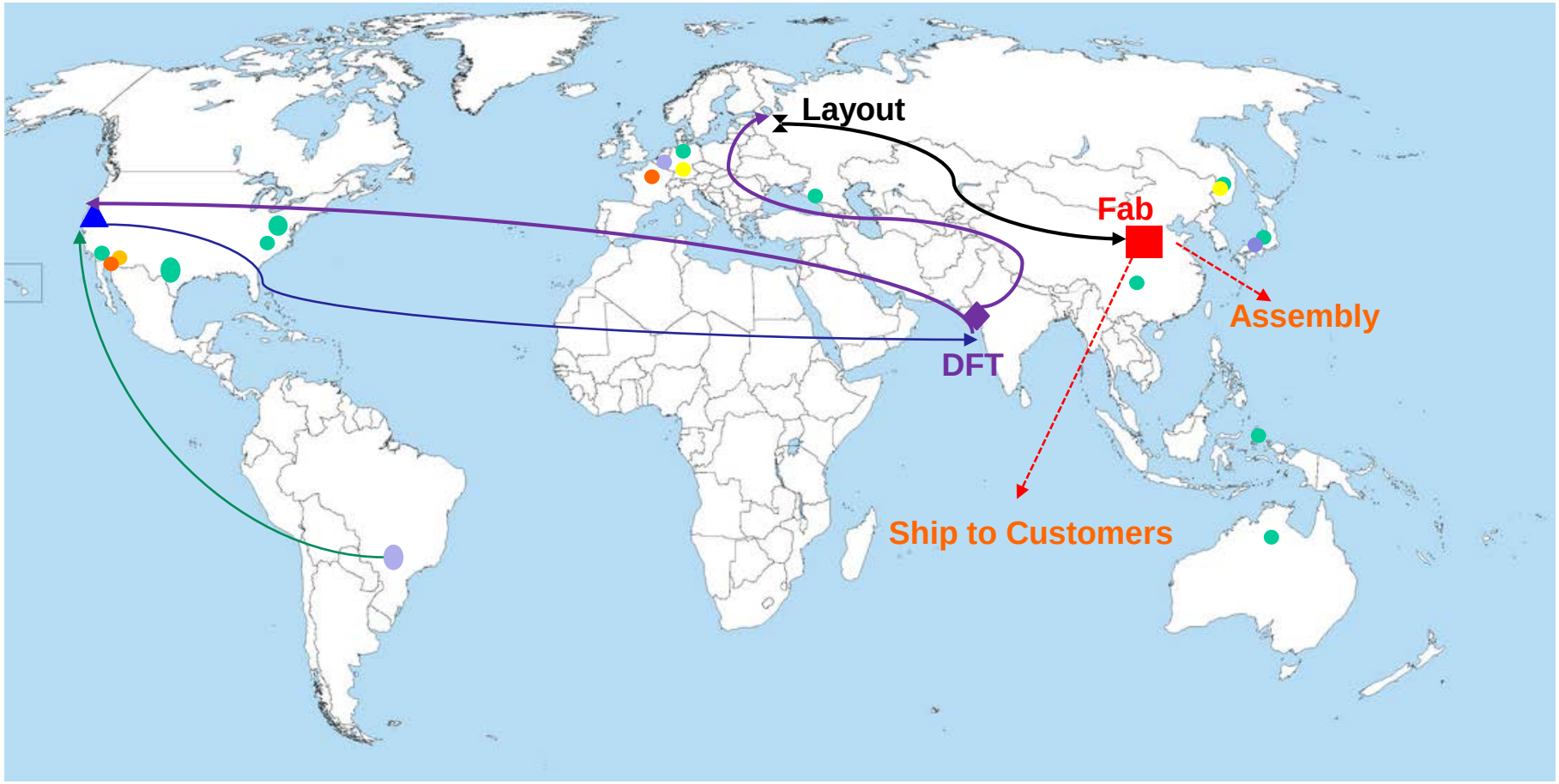
Computer and Information Science and Engineering

University of Florida, USA

ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE

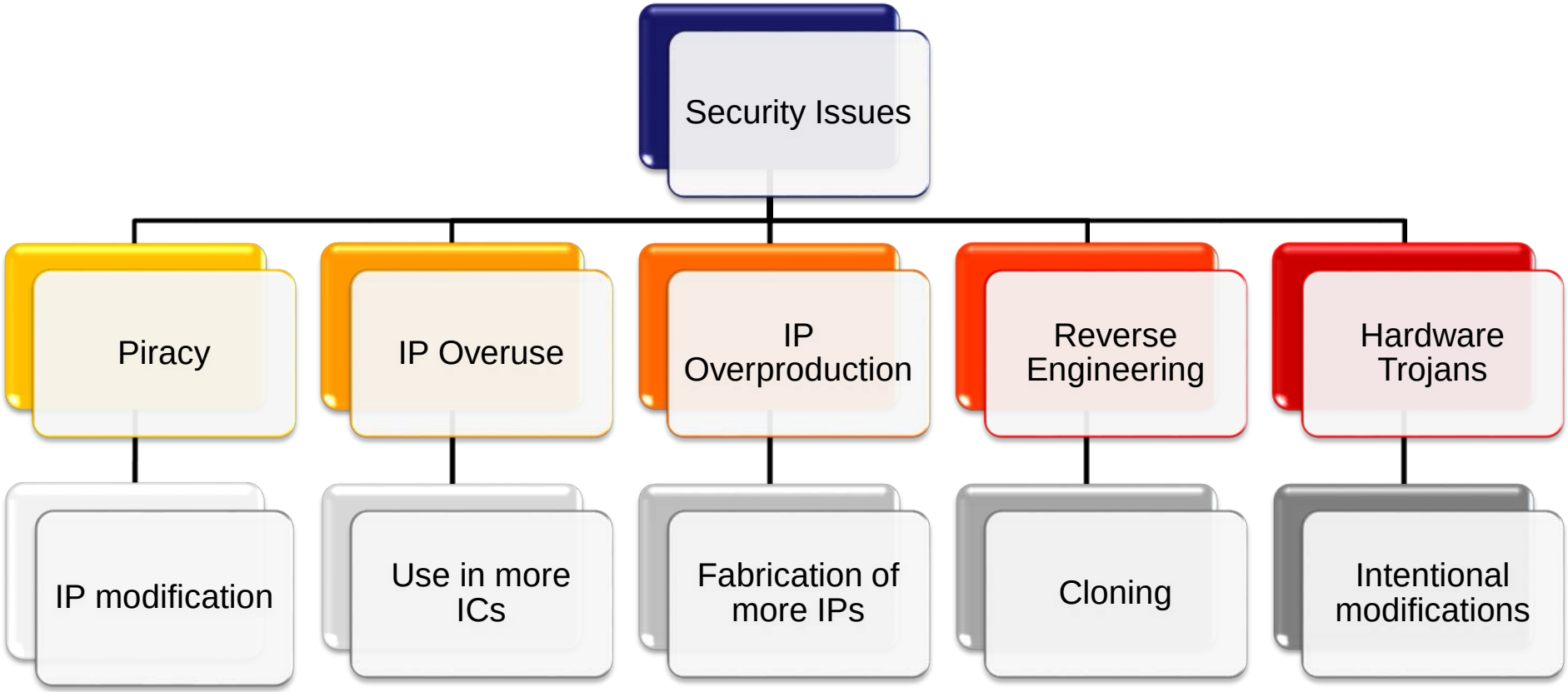
UF UNIVERSITY of FLORIDA | EMBEDDED SYSTEMS LAB

Semiconductor Supply Chain

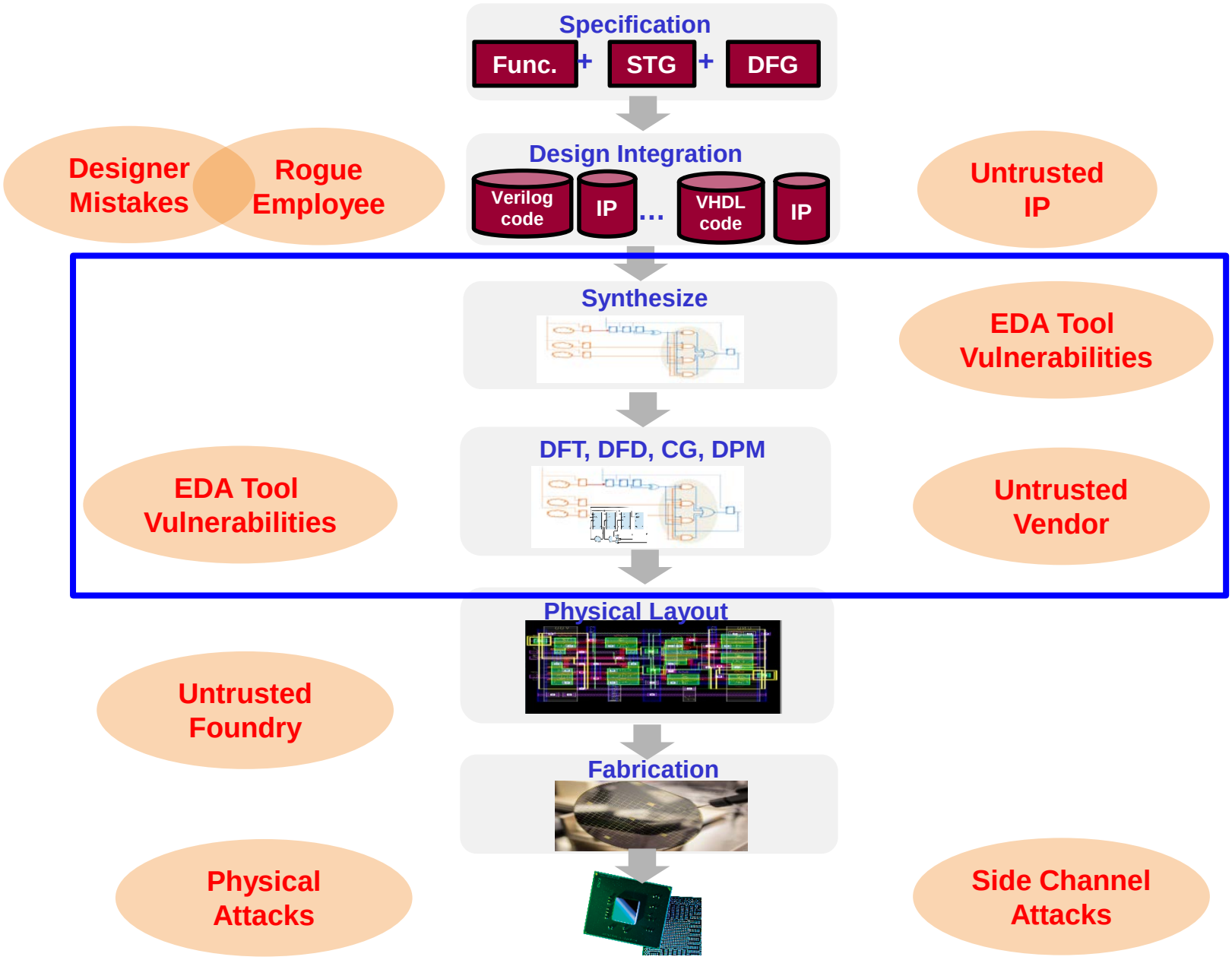


Long and globally distributed supply chain of hardware IPs makes SoC design increasingly vulnerable to diverse trust/integrity issues.

Supply Chain Security Challenges



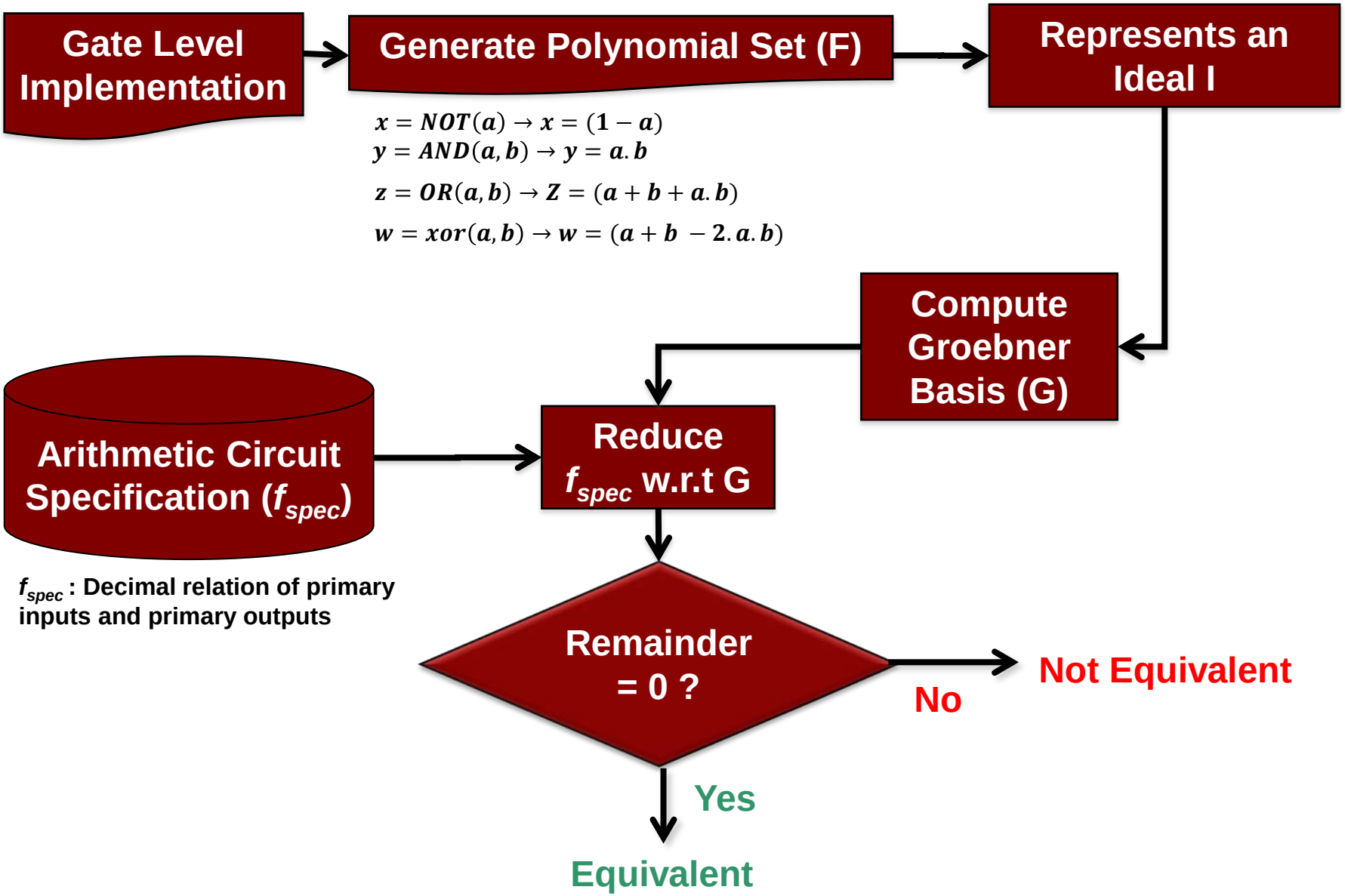
Potential Threats in SoC Design



Related Work

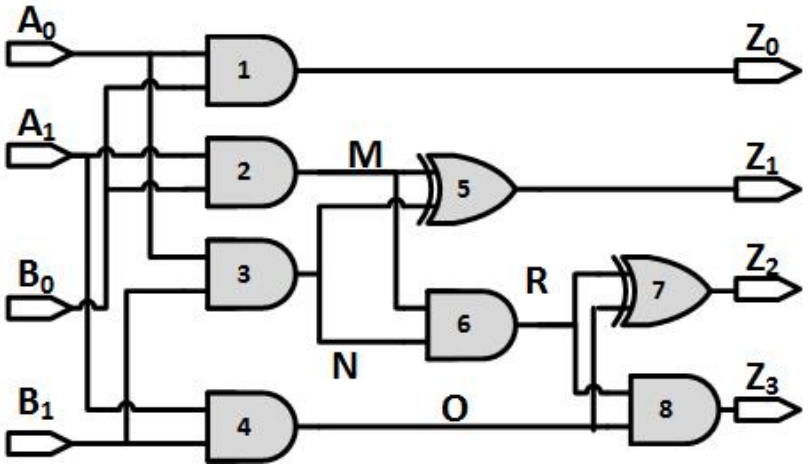
- ATPG based functional Testing
  - MERO, Bhunia et al., CHES 2009
- Boolean functional analysis
  - FANCI, Waksman et al., CCS 2013
- Trojan Template Detection
  - Oya et al., DATE, 2015
- SAT Solvers
  - Banga et al., HOST 2010
- Model Checkers
  - Rajendran et al., DAC 2015
  - Guo et al., HOST 2016

Background: Verification of Arithmetic Circuits



Example (Correct Implementation)

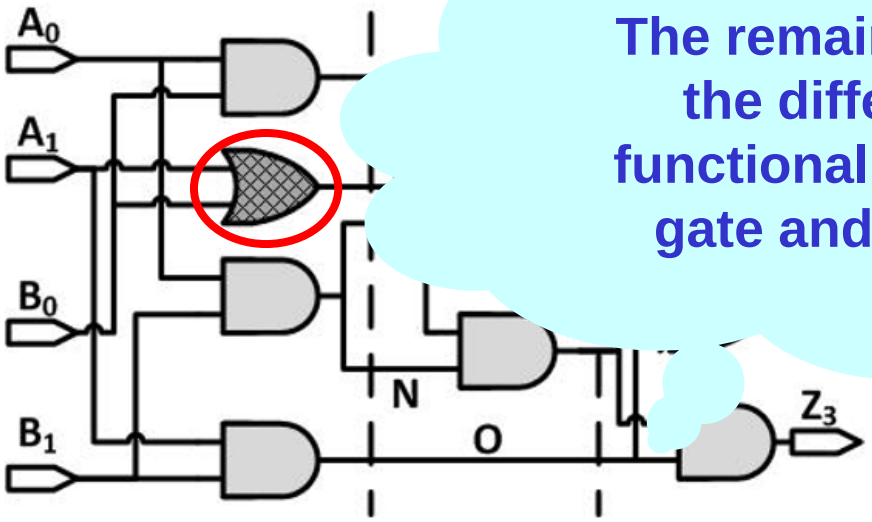
- Consider a 2-bit Multiplier specification
  - $f_{spec} := Z - (A.B)$
    - $\mathbf{Z} = 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0$
    - $\mathbf{A} = 2.A_1 + A_0$, $\mathbf{B} = 2.B_1 + B_0$



- Model gates as polynomials
- Order:
  - $\{Z_2, Z_3\} > \{Z_1, R\} > \{Z_0, M, N, O\} > \{A_1, A_0, B_1, B_0\}$
- 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, M, N, O
  - Step 3: (remainder): 0

Example (Buggy Implementation)

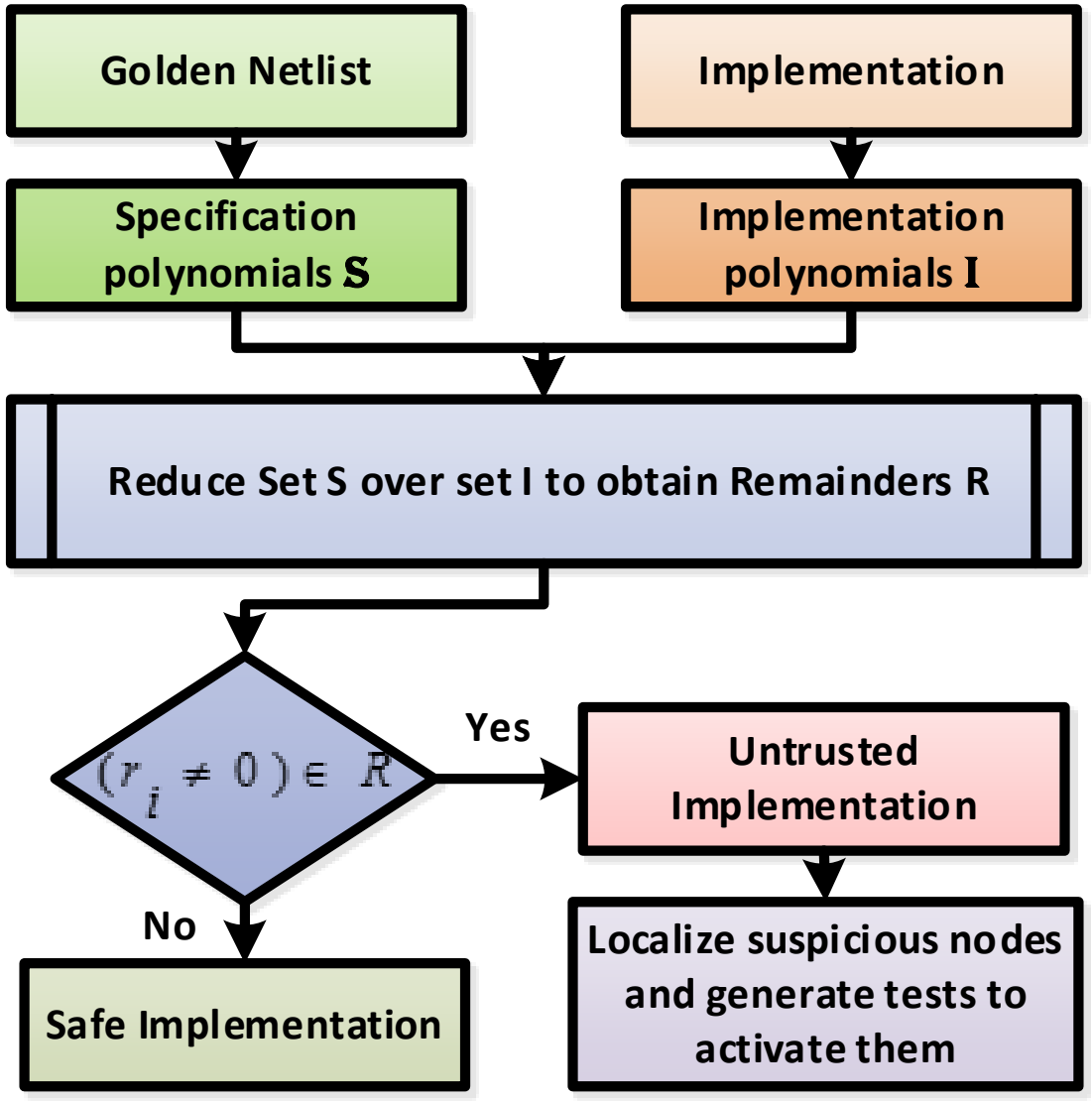
- Consider a buggy 2-bit Multiplier
  - $f_{spec} := Z - (A.B)$
  - $f_{spec} := 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0 - ((2.A_1 + A_0).(2.B_1 + B_0))$



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

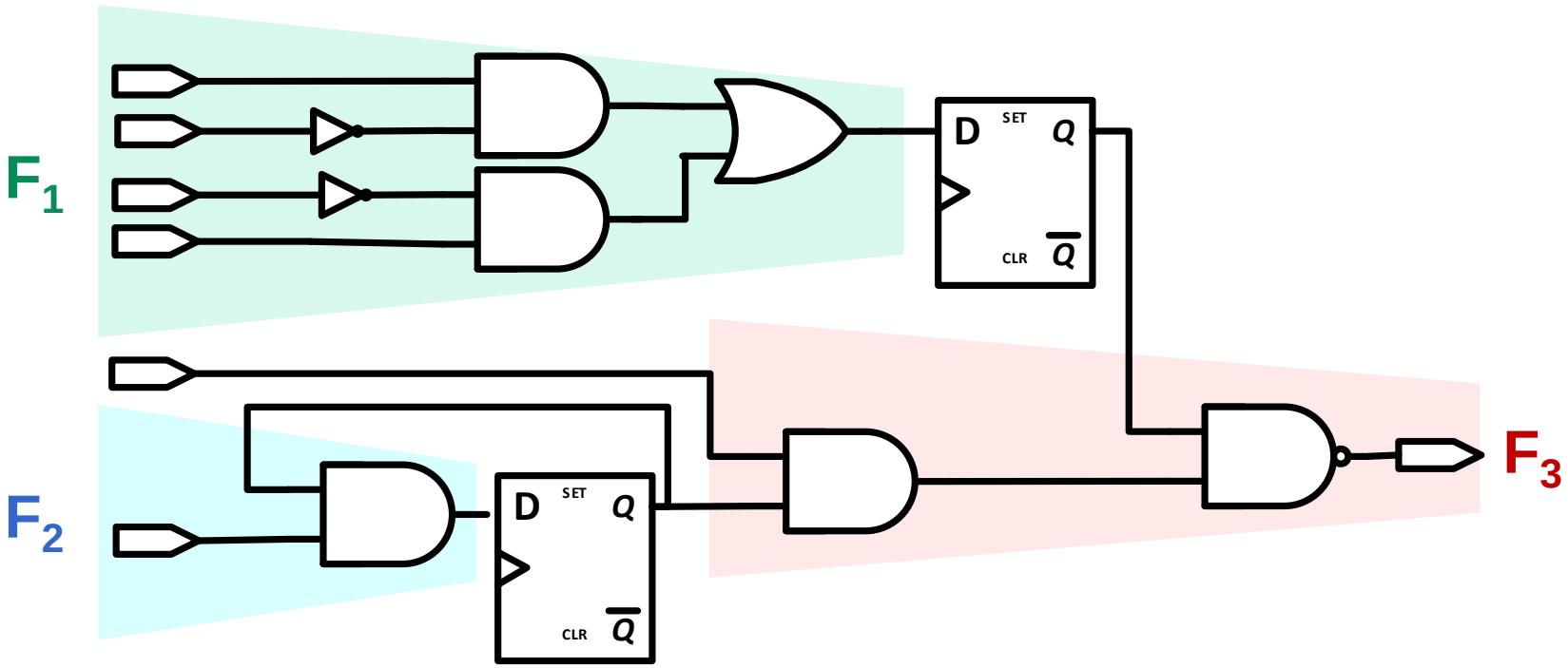
$$f_{spec_0} : 8.Z_3 + 4.Z_2 + 2.Z_1 + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_1} : 4.R + 4.O + 2.z_1 + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_2} : 4.O + 2.M + 2.N + Z_0 - 4.A_1.B_1 - 2.A_1B_0 - 2.A_0.B_1 - A_0.B_0$$
$$f_{spec_3}(remiander) : 2.A_1 + 2.B_0 -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

Our Proposed Flow



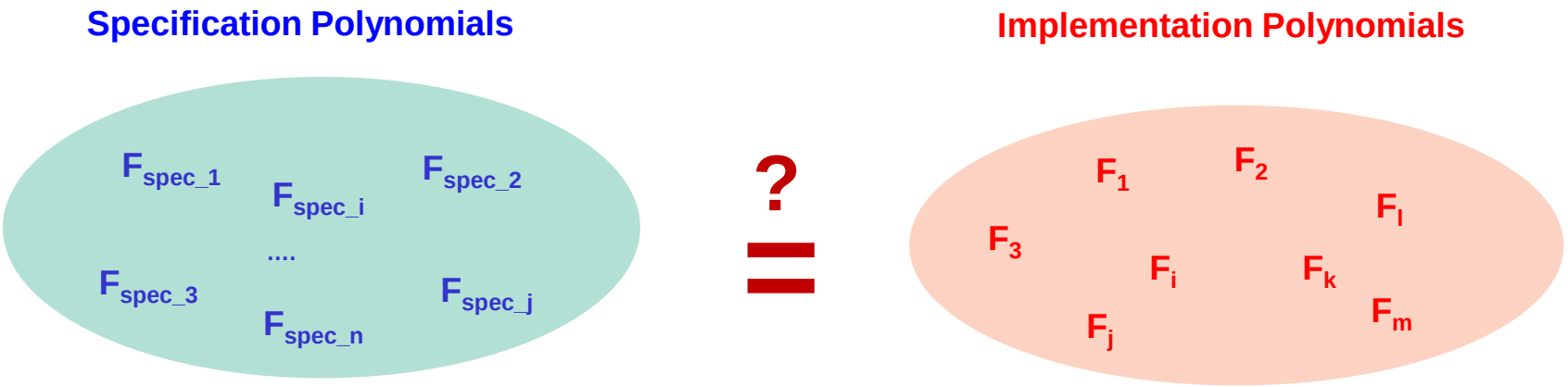
Model Specification and Implementation to Polynomials

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



Equivalence Checking

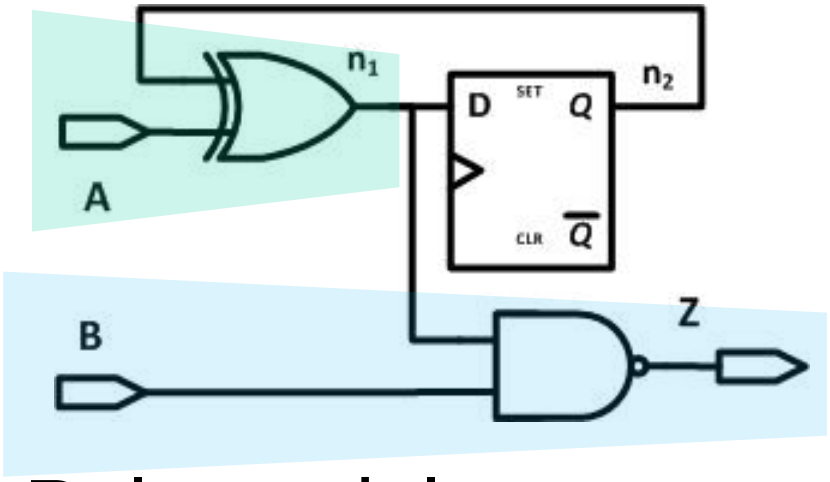
- Reduce each F_{spec_i} over corresponding implementation polynomials



- $F_{spec\_i} = C_s*F_s + C_{s+1}*F_{s+1} + \ldots + C_k*F_k + r_i$
  - 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 some untrustworthy gates

Example: Extracting Specification Polynomials

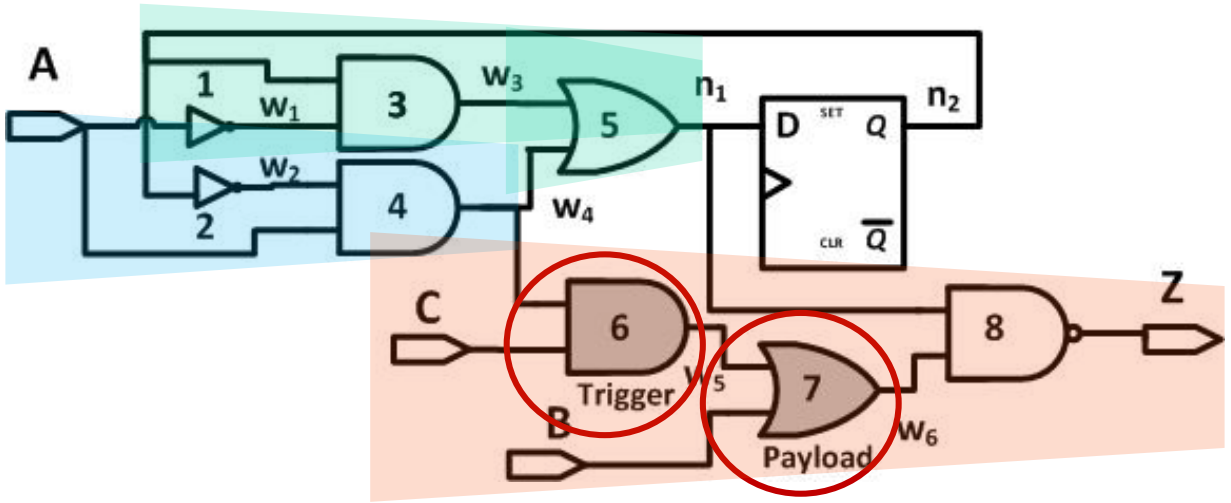
- Part of specification netlist



- 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

- Corresponding part of implementation Netlist
  - Trojan is inserted

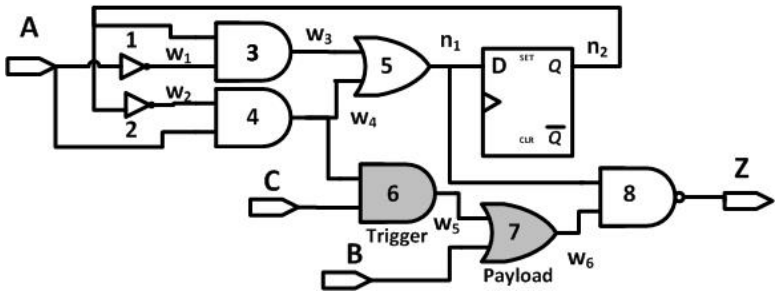


- Implementation polynomials
  - F_1: $n_1 - (n_2*w_4*A - n_2*w_4 + w_4 - n_2*A)=0$
  - F_2: $w_4 - (A - n_2*A)=0$
  - F_3: $Z - (n_1*w_4*C*B- + n_1*w_4*C- n_1*B+1)=0$

Example: Equivalence Checking

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

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



Specification **Implementation**

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

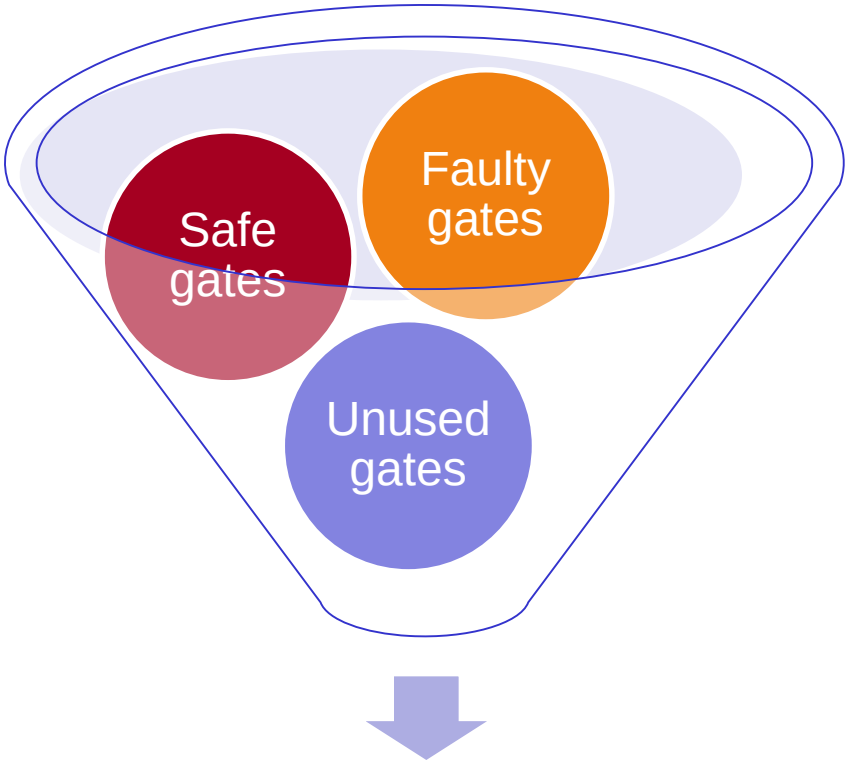
➔ Gates {1,2,3,4,5} which construct the F_{spec1} are safe

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

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

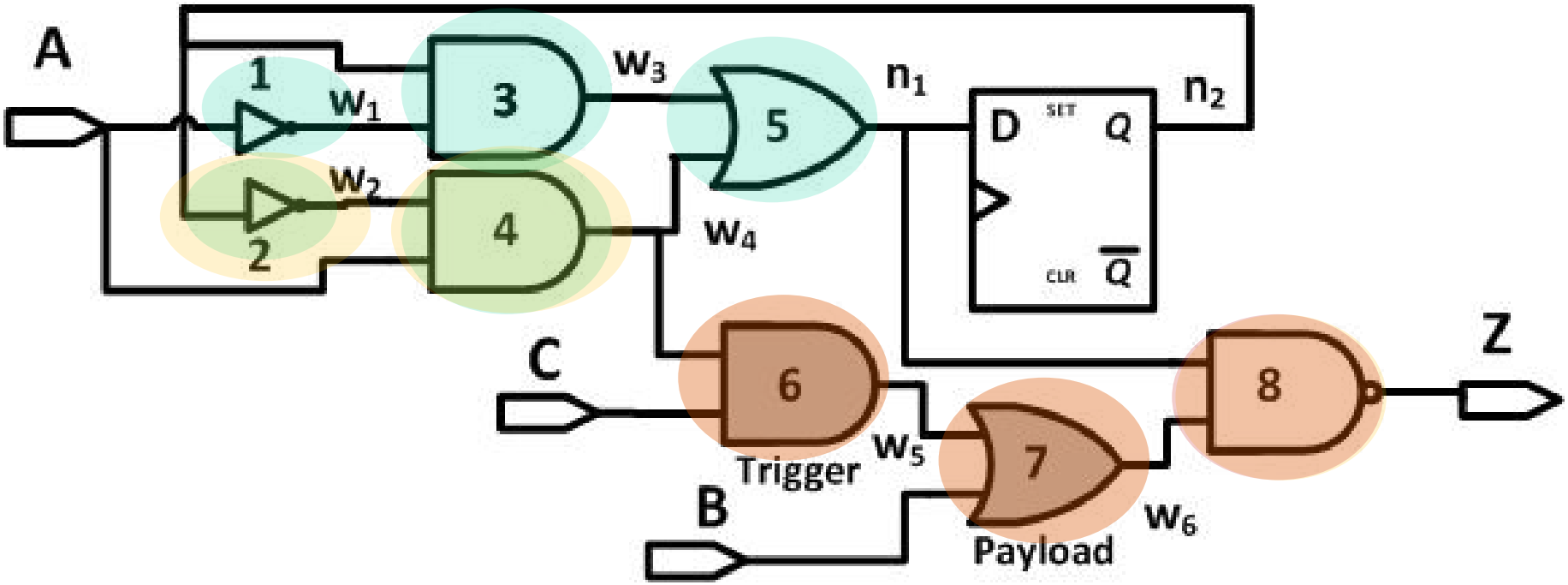
Trojan Localization

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



Example: Trojan Localization

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



Trojan Activation

- Safe gates that are contributing only in faulty regions can be considered as potential Trojan Gates

- ATPG can be used to activate each of the gate in the potential Trojan gates to observe the effect in observable points
  - Activate n nodes at a time (n = {1,2,4,…})
  - To identify Trojan gates

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	12x	100.5x
S15850-T100	2456	27	76	710	27	0	*	*
S38417-T200	5823	15	73	2653	26	11	5.27x	239.8x
S35932-T200	5445	16	70	138	22	6	9x	20.3x
S38584-T200	7580	9	85	47	11	2	38x	19x
Vga-lcd-T100	70162	5	706	**	22	17	41.2x	**

"*" 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., "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in CCS, 2013.

Trojan Activation

- Number of tests needed to activate Trojans

Conclusion

- We proposed an automated approach to localize functional Trojans
  - We identified whether an IP contains malfunctions after performing non-functional changes
  - We presented an algorithm to localize the suspicious area
    - less than 0.03% of the original design in most cases
- Our approach does not require any unrolling or simulation of the design
- A greedy test generation method can be used to activate the Trojan

Thank you !