A Practical Split Manufacturing Framework for Trojan Prevention via Simultaneous Wire Lifting and Cell Insertion

Meng Li[1], Bei Yu[2], Yibo Lin[1], Xiaoqing Xu[1], Wuxi Li[1], David Z. Pan[1]

[1]Electrical & Computer Engineering,
University of Texas at Austin

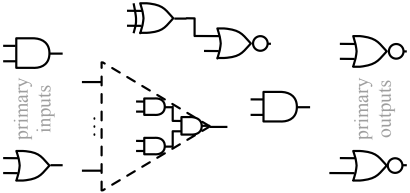
[2]Computer Science & Engineering,
The Chinese University of Hong Kong

TEXAS
The University of Texas at Austin

ASPDAC 2018 - Jan 22, 2017 - Jeju Island, Korea

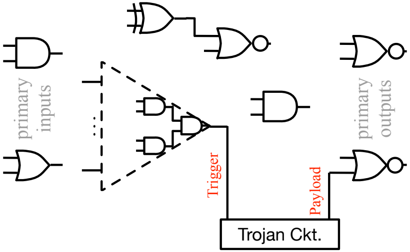
Motivation: Hardware Trojan

- Trojans inserted by untrusted foundries threaten system security
  - Malicious modifications to the original design
  - Ultra lightweight but can completely ruin the system security mechanisms
- Inserted stealthily to prevent post-silicon testing
  - Require strict conditions to trigger the Trojans



Motivation: Hardware Trojan

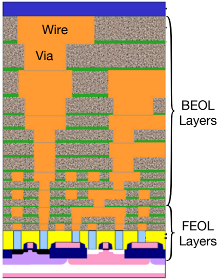
- Trojans inserted by untrusted foundries threaten system security
  - Malicious modifications to the original design
  - Ultra lightweight but can completely ruin the system security mechanisms
- Inserted stealthily to prevent post-silicon testing
  - Require strict conditions to trigger the Trojans

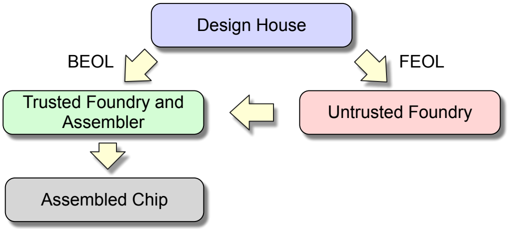


Cells with rare circuit events are more vulnerable to Trojan insertion

What is Split Manufacturing?

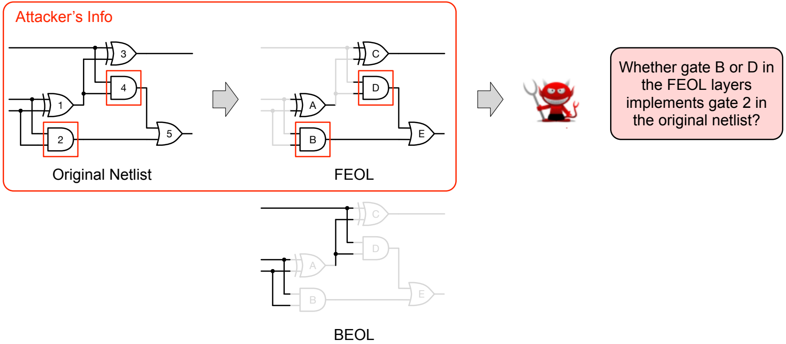
- Target at preventing Trojan insertion by untrusted foundries
  - Front-end-of-line (FEOL): cells and wires in lower metal layers, untrusted foundries
  - Back-end-of-line (BEOL): wires in higher metal layers, trusted foundries
- Wire connections in BEOL layers are hidden from the attackers
  - Incur overhead for the wires in the BEOL layers





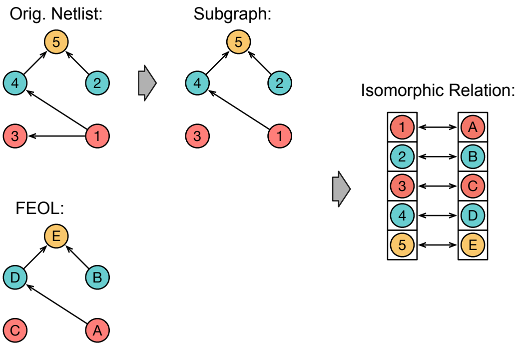
Why Split Manufacturing Deters Trojan Insertion?

- Assume attackers have the original netlist and a full control of FEOL
  - Determine logic signals used to trigger the Trojan based on the original netlist
  - Determine the target locations to insert Trojans in the FEOL layers
- Critical nodes can still be protected under such a strong attack model



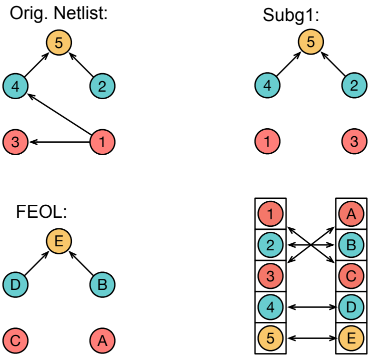
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Regard FEOL layers and the original netlist as graphs
  - The FEOL graph must be a subgraph of the original netlist
- An attacker can identify the physical implementation by subgraph isomorphism relation



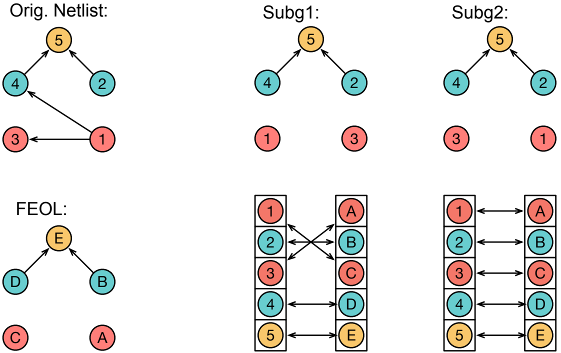
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Different isomorphism relations lead to multiple possible physical implementations
- Previous security criterion: k-security
  - For one cell in the original netlist, require k different possible implementations
  - For the netlist, require each cell to be at least k secure



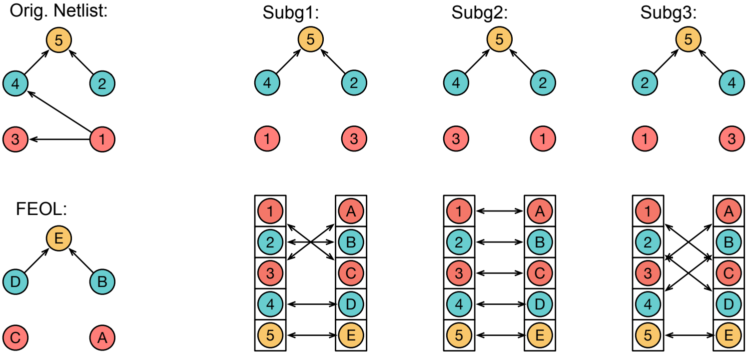
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Different isomorphism relations lead to multiple possible physical implementations
- Previous security criterion: k-security
  - For one cell in the original netlist, require k different possible implementations
  - For the netlist, require each cell to be at least k secure



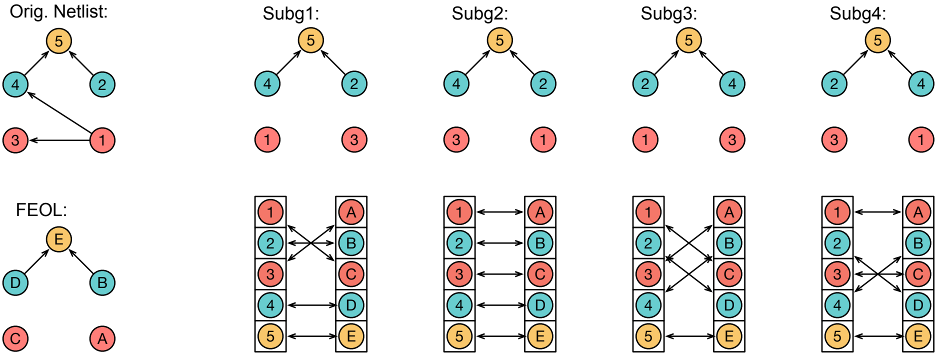
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Different isomorphism relations lead to multiple possible physical implementations
- Previous security criterion: k-security
    - For one cell in the original netlist, require k different possible implementations
    - For the netlist, require each cell to be at least k secure



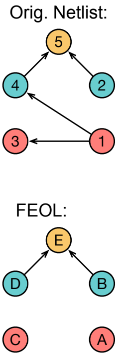
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Different isomorphism relations lead to multiple possible physical implementations
- Previous security criterion: *k*-security
  - For one cell in the original netlist, require k different possible implementations
  - For the netlist, require each cell to be at least k secure



Previous Split Manufacturing Framework [Imeson+, Usenix'13]

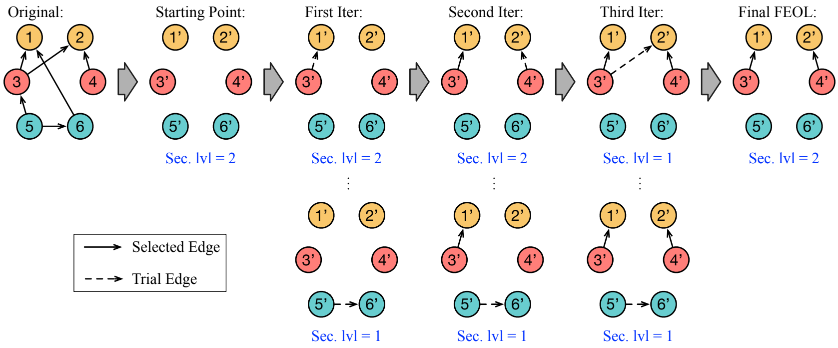
- Different isomorphism relations lead to multiple possible physical implementations
- Previous security criterion: *k*-security
  - For one cell in the original netlist, require k different possible implementations
  - For the netlist, require each cell to be at least k secure



Nodes 1, 2, 3, 4 are 2-secure.
Node 5 is 1-secure.
The netlist is 1-secure.

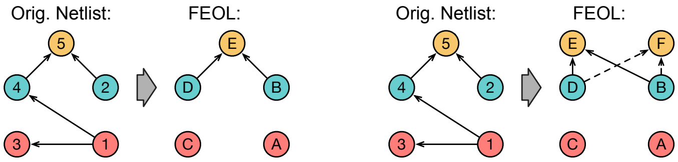
Previous Split Manufacturing Framework [Imeson+, Usenix'13]

- Greedy split manufacturing flow [Imeson+, Usenix'13]
  - Start by lifting all wires to BEOL layers and add them back iteratively
  - Greedily select wires with the maximized netlist security
- Poor scalability due to repetitive subgraph isomorphism checking



Overview of Our Proposed Solution

- Besides scalability, [Imeson+, Usenix'13] cannot always achieve required security levels
- New solution: allowing the dummy node/wire insertion together with wire lifting
  - Only allow inserting wires pointing to dummy nodes



- However, still need to resolve two new issues
  - How to define the security criterion since FEOL is not a subgraph of the original netlist
  - How to enhance the scalability and allow concurrent node/wire insertion

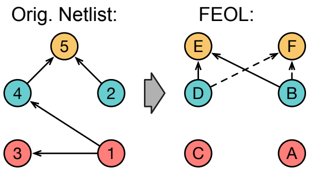
Generalized Security Criterion

- Invariant relations between the FEOL layers and the original netlist

Relation One

Each node in the original netlist has exactly one actual implementation in FEOL

- For example, one of nodes B and D in FEOL must implement node 2



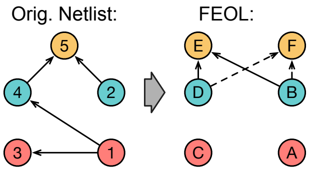
Generalized Security Criterion

- Invariant relations between the FEOL layers and the original netlist

Relation Two

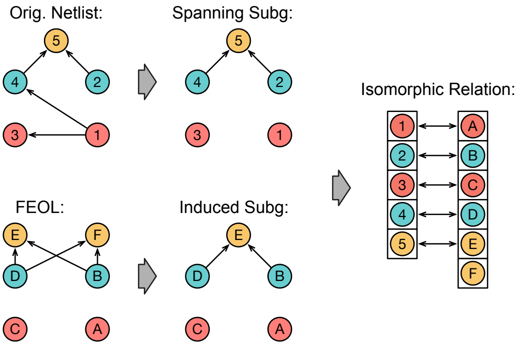
If a node in FEOL is the actual physical implementation of a certain node in the original netlist, none of edges pointing to the node can be dummy

- Recall inserting dummy wires pointing to the actual physical implementation is not allowed
- For example, if F is the implementation of 5, then (D, F) and (B, F) are not dummy



Generalized Security Criterion

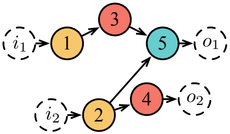
- Now, define new security criterion to accommodate node/wire insertion
- To identify the possible implementation, build Subgraph Isomorphism Relation between
  - Spanning subgraph of the original netlist and induced subgraph of FEOL
- k-security can be defined based on the subgraph isomorphism relation



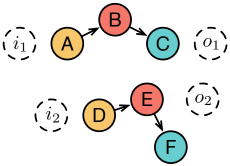
Sufficient Condition for Security Criterion

- New security criterion does not help with scalability
  - Graph isomorphism checking is still required to determine security
- Sufficient condition based on k-isomorphism [Cheng+, SIGMOD'10]:
  - A graph composed of k disjoint isomorphic subgraphs is k-isomorphic
  - A k-isomorphic FEOL graph guarantees k security
- Avoid isomorphism checking by achieving the sufficient condition

Orig. Netlist:



FEOL Layers:



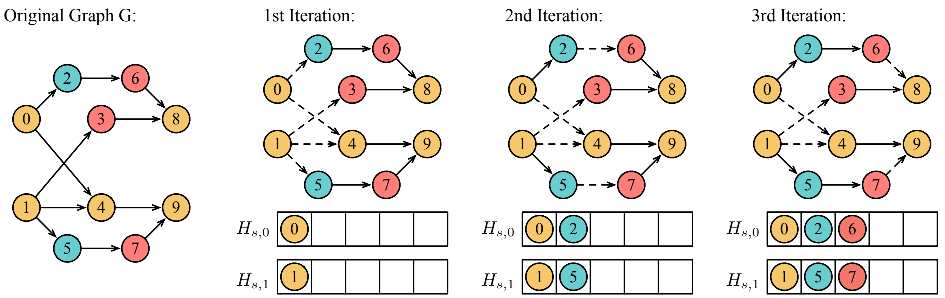
If A is the candidate node of 1, then D must be the candidate node of 1 as well.

MILP based FEOL Generation

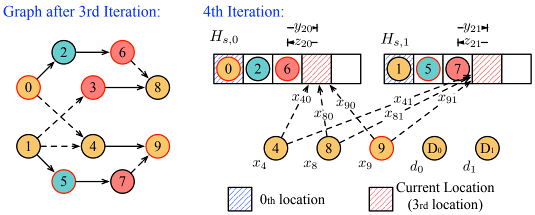
Problem Formulation

Generate FEOL that satisfies the sufficient condition for the required security level, i.e. *k*-isomorphism, and minimizes the introduced overhead.

- Insert nodes into the subgraphs iteratively and guarantee isomorphism simultaneously



MILP based FEOL Generation



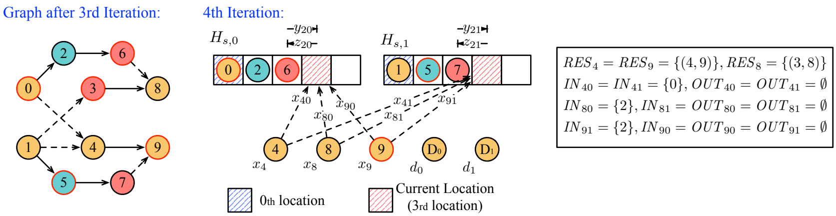
$RES_4 = RES_9 = \{(4,9)\}, RES_8 = \{(3,8)\}$
$IN_{40} = IN_{41} = \{0\}, OUT_{40} = OUT_{41} = \emptyset$
$IN_{80} = \{2\}, IN_{81} = OUT_{80} = OUT_{81} = \emptyset$
$IN_{91} = \{2\}, IN_{90} = OUT_{90} = OUT_{91} = \emptyset$

- Objective function: minimize overhead for the current iteration

$$\min_{x,d} \ \alpha \sum_i |RES_i| x_i - \beta k \sum_l (y_l + z_l) + \gamma A \sum_j d_j$$

  - Area of dummy cells to insert
  - Number of wires to lift to BEOL
  - Number of wires to add back to FEOL

MILP based FEOL Generation



- Constraints: node selection, subgraph selection, and edge insertion

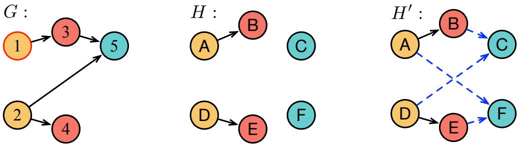
$$\sum_i x_i w_i = 1, \sum_{j=0}^{k-1} x_{ij} = x_i, \qquad \forall i; \qquad \sum_i x_{ij} + d_j = 1, \qquad \forall j \in \{0, \dots, k-1\};$$

$$y_l \leq \sum_i x_{ij} \cdot 1_{l \in IN_{ij}} + d_j, \qquad z_l \leq \sum_i x_{ij} \cdot 1_{l \in OUT_{ij}}, \qquad \forall j, l.$$

- y_l and z_l can be relaxed to continuous variables without impacting the solution optimality

k-Secure Layout Refinement

- Guarantee k-security in the placement stage
- Previous method: ignore interconnections in BEOL layers
  - Suffer from large overhead since cells are floating in FEOL layers
- Our method: insert virtual nets in the placement stage



Experimental Setup

- Benchmarks: ISCAS 85 and OpenSPARC T1
- Program implemented in C++
- MILP solver: GUROBI
- To protect a subset of circuit nodes, we select the nodes considering Trojan insertion strategies used in TrustHub

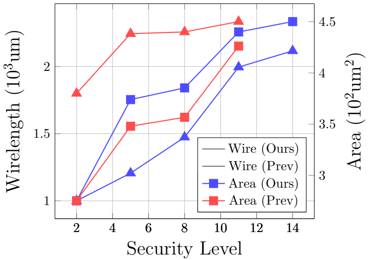
Experimental Results: Runtime Comparison

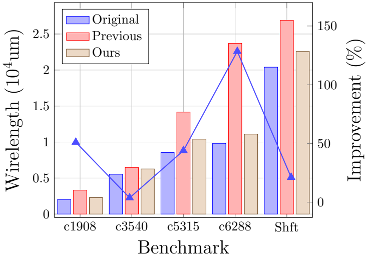
- Comparison with [Imeson+, Usenix'13] on FEOL generation:
  - Achieve 10-security and protect 5% nodes
  - $\alpha = 0.5, \beta = 2.0$, and $\gamma = 0.8$

Bench	# Protect	# Nodes	Prev (s)	Ours (s)
c432	23	214	140.8	0.5
c880	19	355	979.6	3.2
c1908	24	519	>100000	8.1
c3540	49	1012	>100000	37.0
c5315	73	1864	>100000	135.0
c6288	90	2568	>100000	297.9
Shifter	84	2579	>100000	273.9
Norm			293.9	1.0

Experimental Results: Overhead Comparison

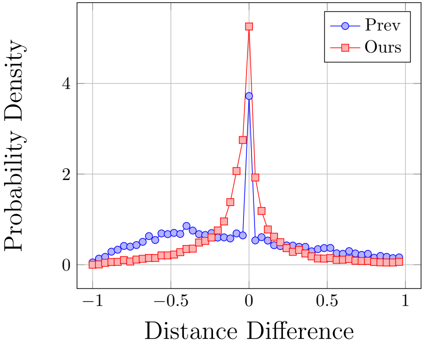
- Comparison with [Imeson+, Usenix'13] on routed wirelength
  - For the FEOL generation strategy, on average 59.1% wirelength overhead reduction with less than 4% area overhead increase
  - For the placement refinement, on average 49.6% wirelength overhead reduction





Experimental Results: Proximity Checking

- Comparison with [Imeson+, Usenix'13]
  - Distance between protected nodes and their candidates
  - For all benchmarks, none of correct connections can be recovered



Thank you for your attention!

Backup: Overhead Dependency

- Overhead dependency on
  - Security level
  - Number of protected nodes
  - MILP coefficient γ