



Machine Learning and Structural Characteristics for Reverse Engineering

Johanna Baehr, Alessandro Bernardini, Georg Sigl and Ulf Schlichtmann
Technical University of Munich
Department of Electrical and Computer Engineering
Institute for Security in Information Technology
Tokyo, 22nd January 2019



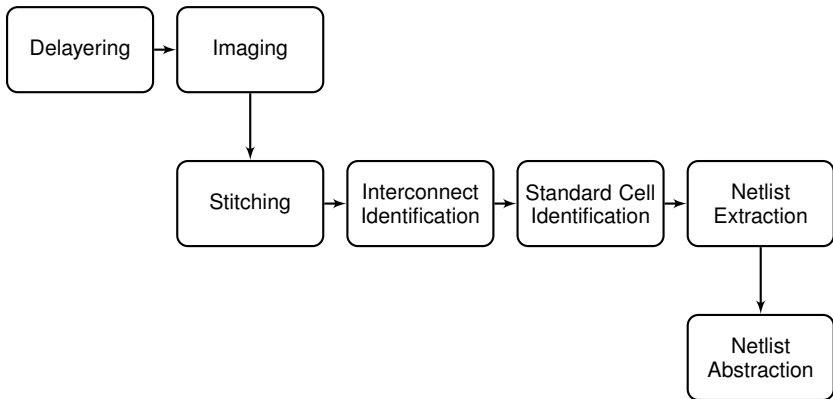
TUM Uhrenturm



The Problem

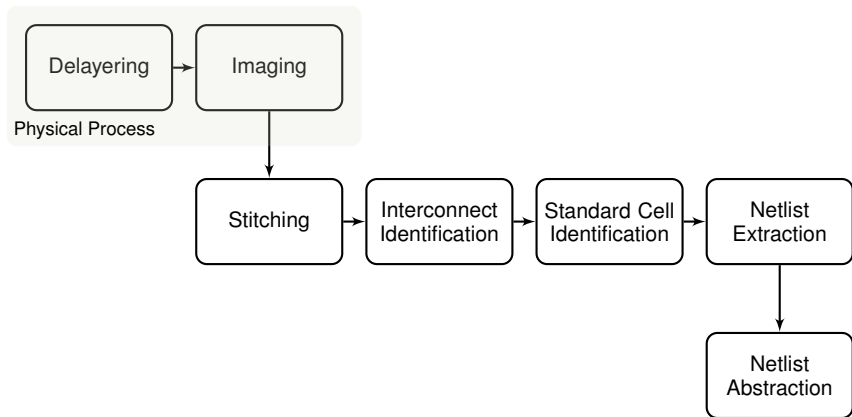


Circuit Extraction



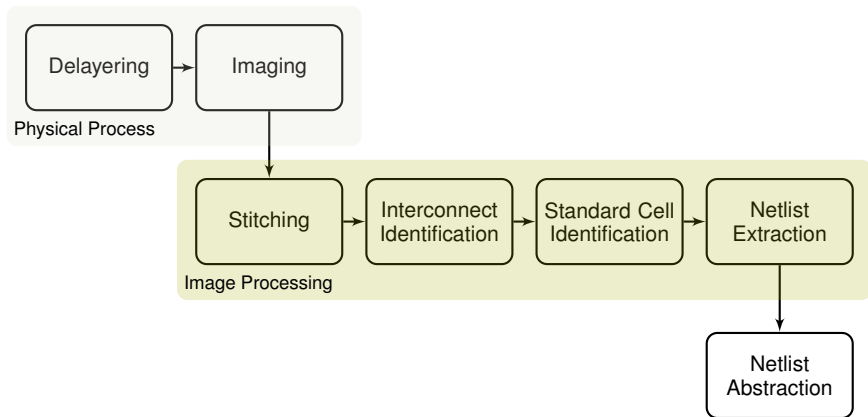


Circuit Extraction



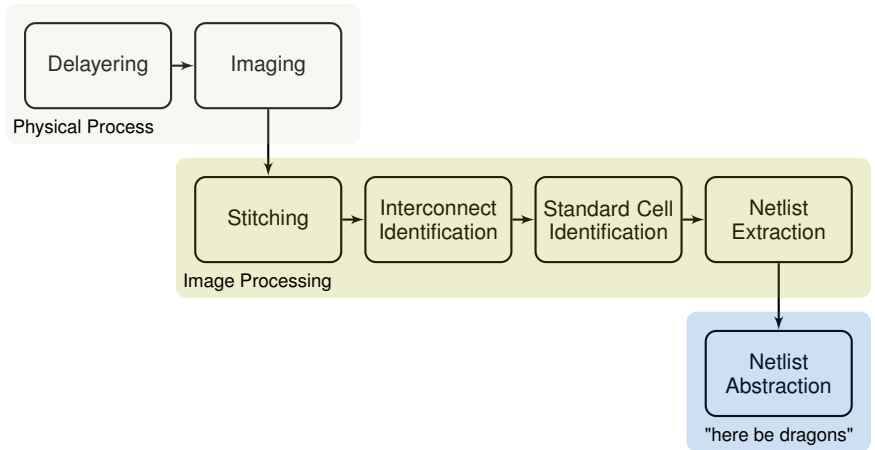


Circuit Extraction





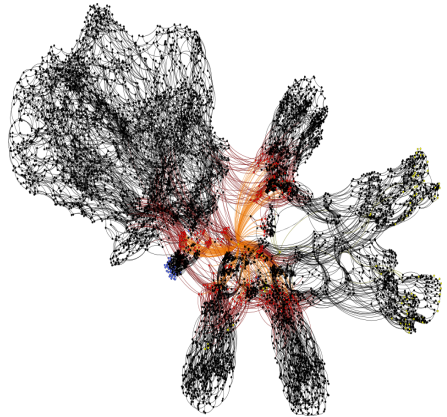
Circuit Extraction





Netlist abstraction

Goal: From gate-level netlist to human understandable, high level description of functionality.

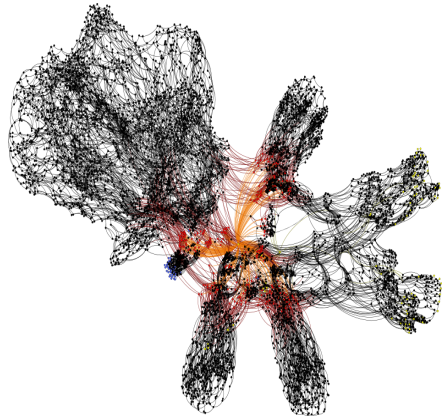




Netlist abstraction

Goal: From gate-level netlist to human understandable, high level description of functionality.

Assumption: Functionality and hierarchy through comparison with something known.



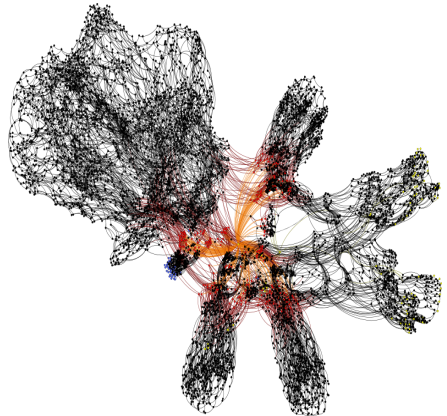


Netlist abstraction

Goal: From gate-level netlist to human understandable, high level description of functionality.

Assumption: Functionality and hierarchy through comparison with something known.

If we cannot understand the whole circuit, **divide** and **conquer**:



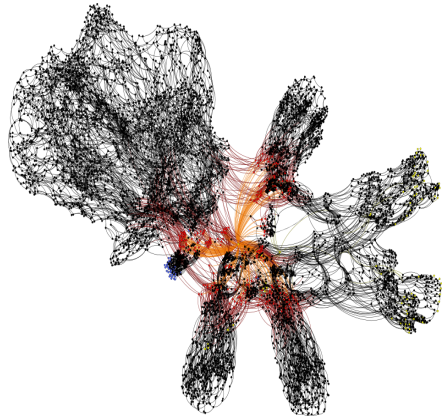
Netlist abstraction

Goal: From gate-level netlist to human understandable, high level description of functionality.

Assumption: Functionality and hierarchy through comparison with something known.

If we cannot understand the whole circuit, **divide** and **conquer**:

1. **Find potential submodules / building blocks**
2. **Identify the functionality of potential building blocks**

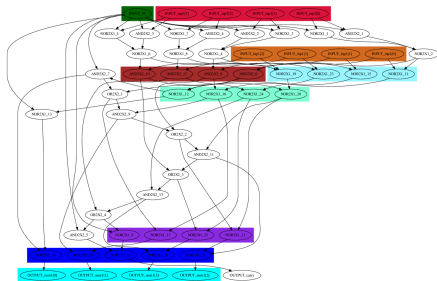




State of the Art

Building Block identification

Functional Data Path Analysis [1]:
Operations are carried out in parallel.





State of the Art

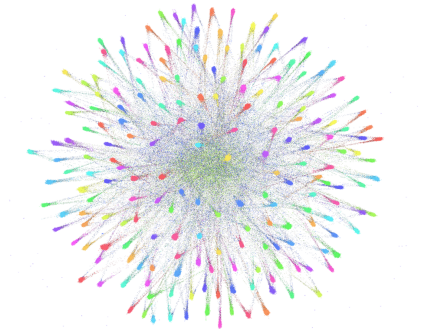
Building Block identification

Functional Data Path Analysis [1]:

Operations are carried out in parallel.

Cluster Analysis [2, 3, 4]:

Submodules are more interconnected.



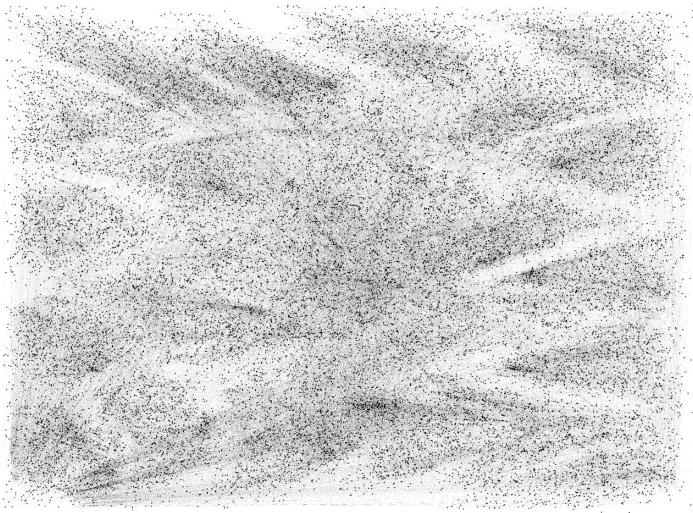


State of the Art



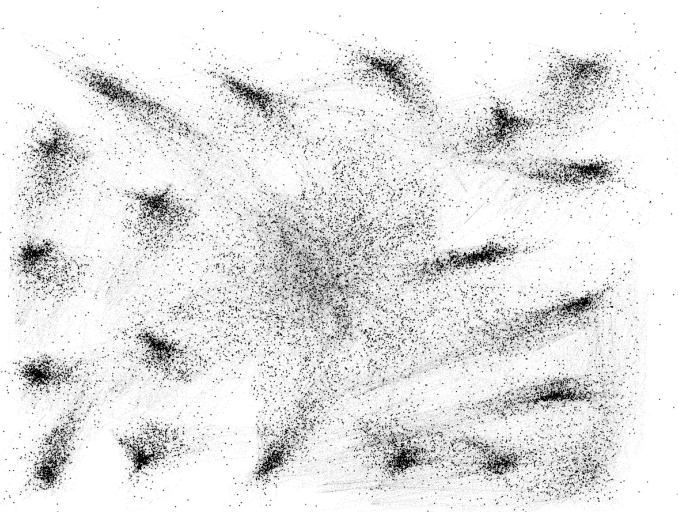


State of the Art



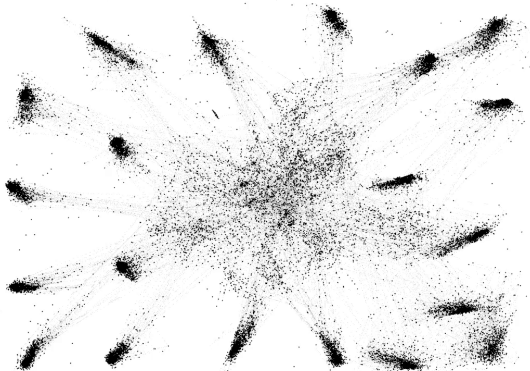


State of the Art



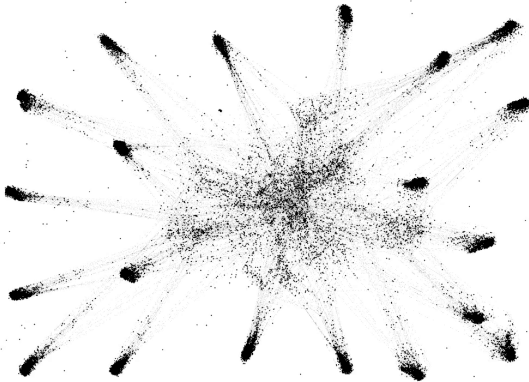


State of the Art



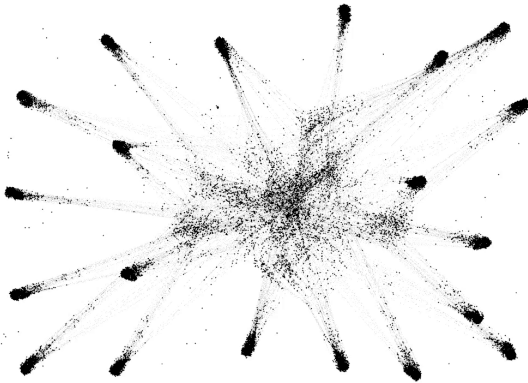


State of the Art



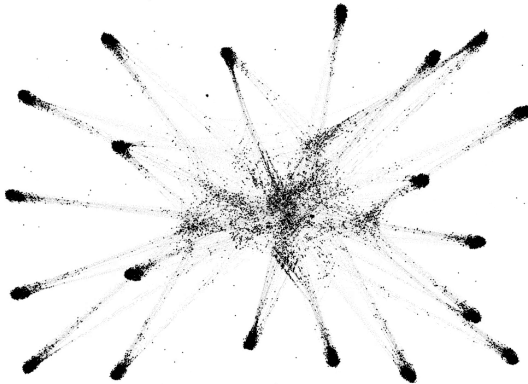


State of the Art



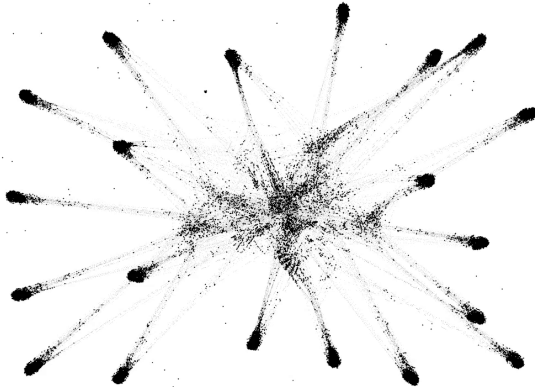


State of the Art





State of the Art





State of the Art





State of the Art

Building Block identification

Functional Data Path Analysis [1]:

Operations are carried out in parallel.

Cluster Analysis [2, 3, 4]:

Submodules are more interconnected.

State of the Art

Building Block identification

Functional Data Path Analysis [1]:

Operations are carried out in parallel.

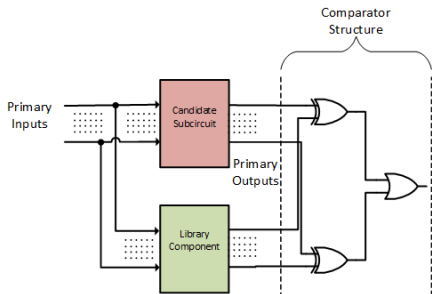
Cluster Analysis [2, 3, 4]:

Submodules are more interconnected.

Functionality identification

Formal Methods [1, 5]:

Equivalence with functionally equivalent IP.



State of the Art

Building Block identification

Functional Data Path Analysis [1]:

Operations are carried out in parallel.

Cluster Analysis [2, 3, 4]:

Submodules are more interconnected.

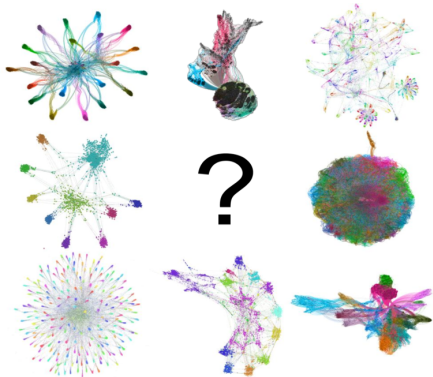
Functionality identification

Formal Methods [1, 5]:

Equivalence with functionally equivalent IP.

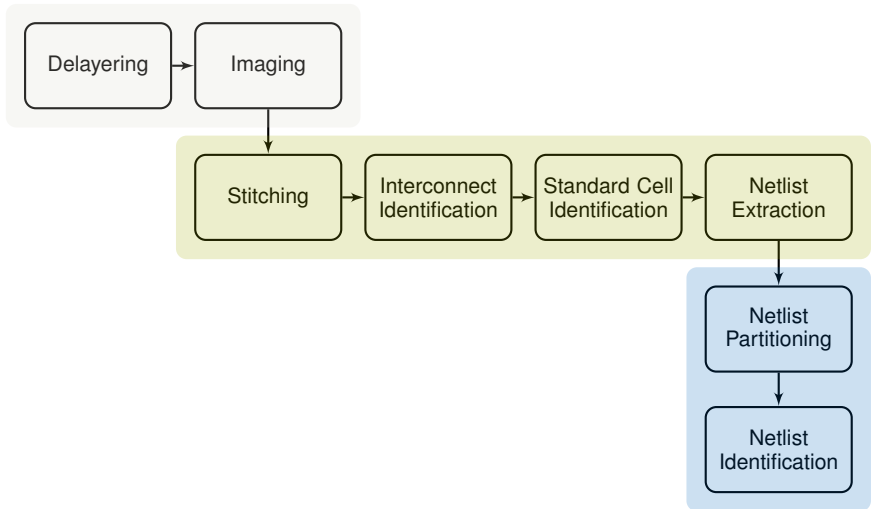
Fuzzy Methods [3, 6]:

Structural Characteristics of similar IP.



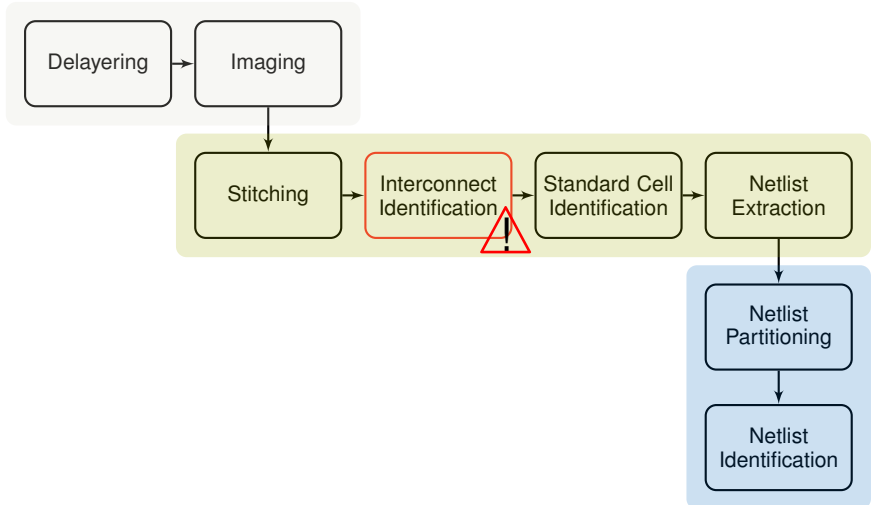


Errors in the Reverse Engineering Flow



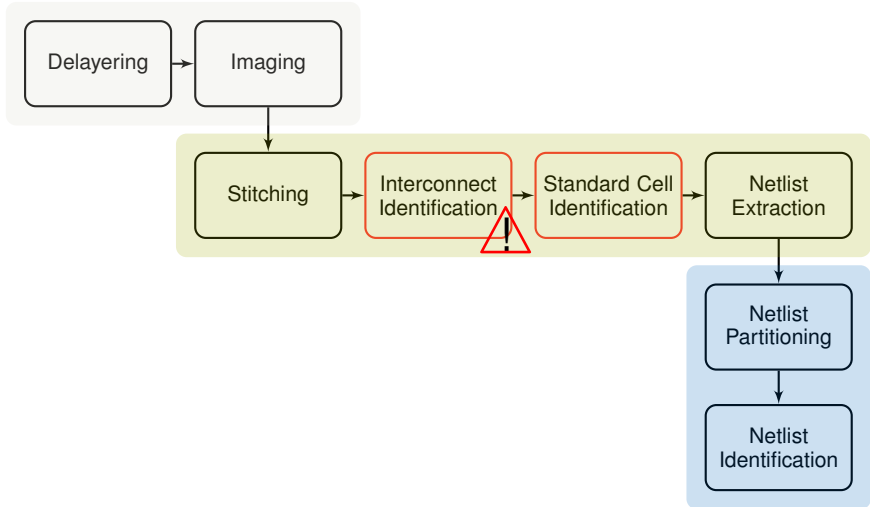


Errors in the Reverse Engineering Flow



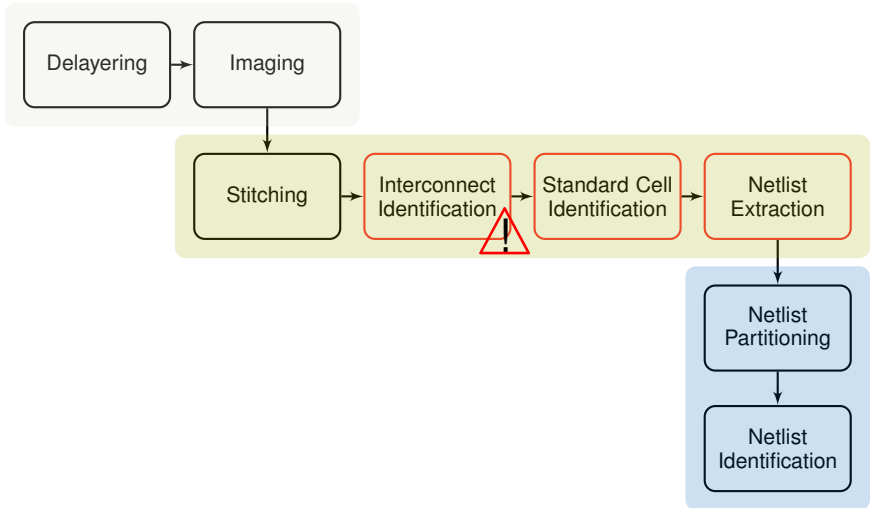


Errors in the Reverse Engineering Flow



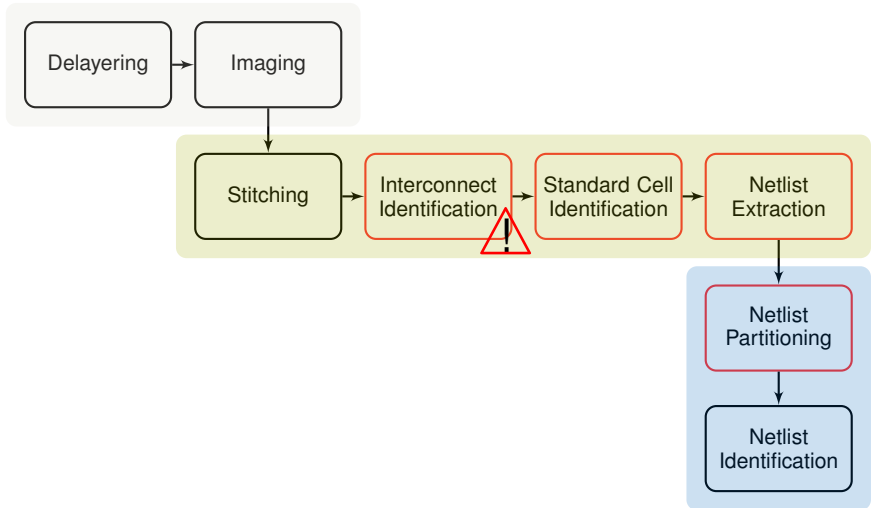


Errors in the Reverse Engineering Flow



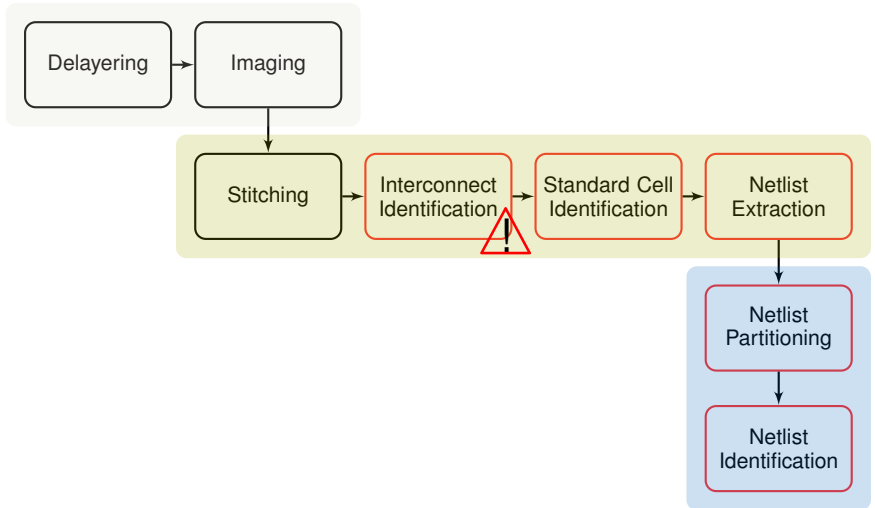


Errors in the Reverse Engineering Flow



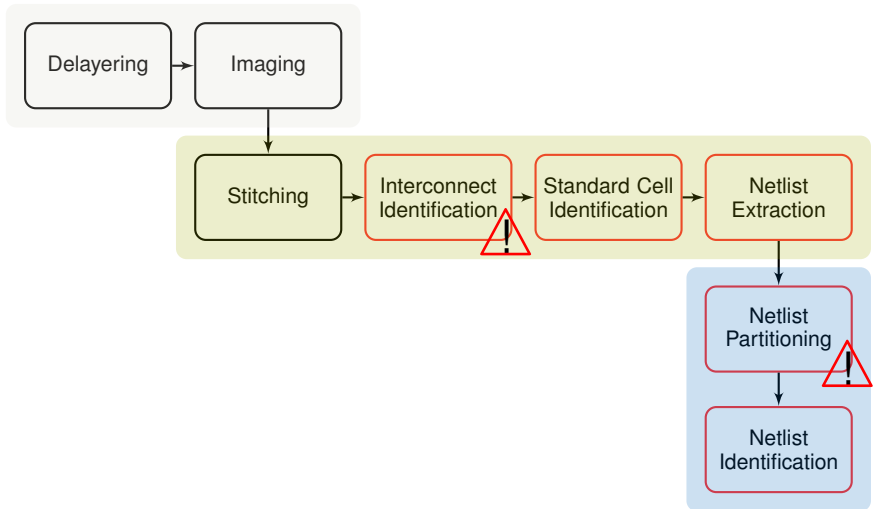


Errors in the Reverse Engineering Flow

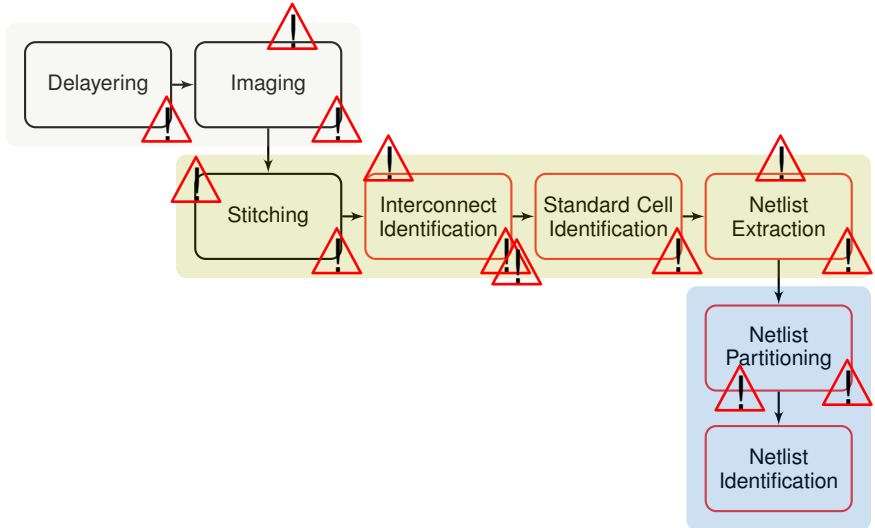




Errors in the Reverse Engineering Flow



Errors in the Reverse Engineering Flow





Functionality identification with errors

Formal Methods



Functionality identification with errors

Formal Methods

- Comparison between netlist and functionally equivalent component.



Functionality identification with errors

Formal Methods

- Comparison between netlist and functionally equivalent component.
- Equivalence Checking: Satisfiability Solving or extensions thereof.



Functionality identification with errors

Formal Methods

- Comparison between netlist and functionally equivalent component.
- Equivalence Checking: Satisfiability Solving or extensions thereof.
- **A single error will mean non-equivalence.**



Functionality identification with errors

Formal Methods

- Comparison between netlist and functionally equivalent component.
- Equivalence Checking: Satisfiability Solving or extensions thereof.
- **A single error will mean non-equivalence.**

Proposal: Fuzzy Methods

1. Fuzzy Structural Matching: Structural feature based K-Nearest-Neighbor Classification.
2. Fuzzy Graph Isomorphism: Robust structural and functional graph isomorphism based on node hashes.



The Solution



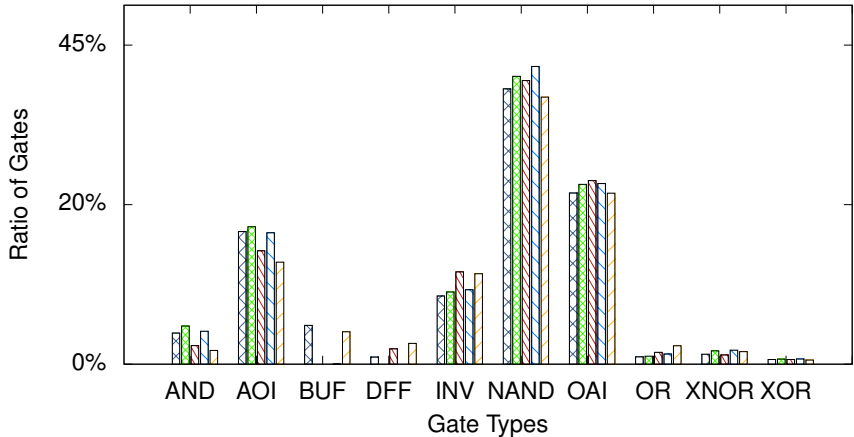
Method 1: Structural Features

- **Ratio of Gate Functionalities**

Graphs have a distinctive distribution of gates.

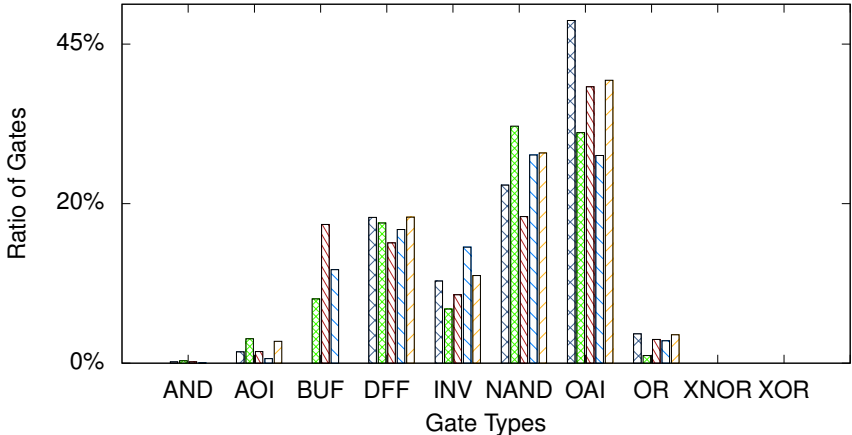
Method 1: Structural Features

Node Functionality Distribution for Multipliers



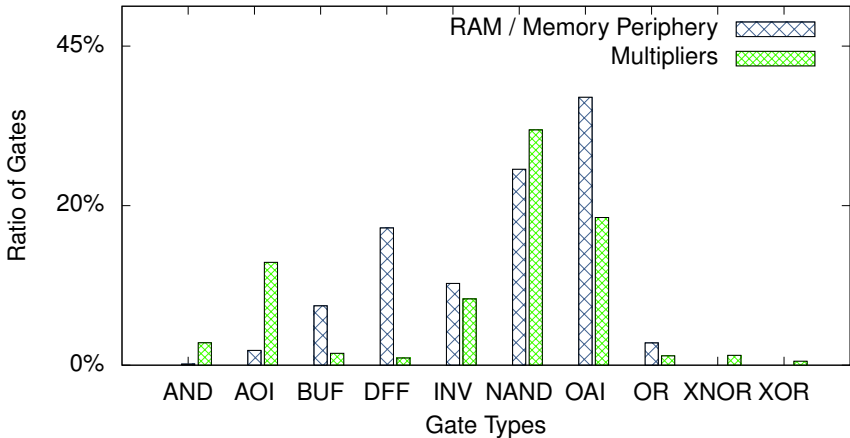
Method 1: Structural Features

Node Functionality Distribution for RAM / Memory Periphery



Method 1: Structural Features

Averaged Ratio of Gates





Method 1: Structural Features

- **Ratio of Gate Functionalities**

Graphs have a distinctive distribution of gates.

- **Centrality Measures**

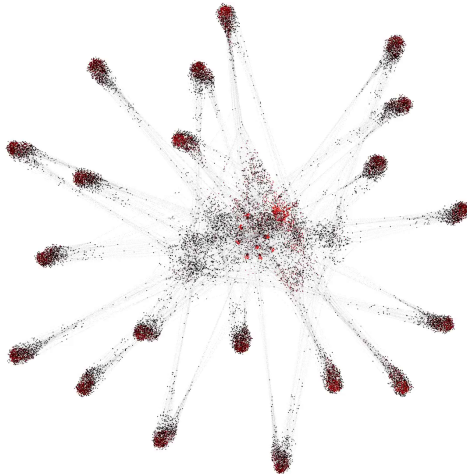
How do gates affect transfer of data?

Which and how many gates strongly affect transfer of data?

- ▶ Betweenness Centrality
- ▶ Closeness Centrality
- ▶ Eigenvector Centrality
- ▶ Pagerank
- ▶ Degree Centrality
- ▶ ...

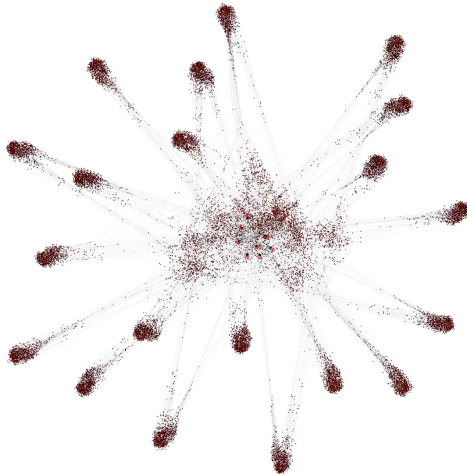


Method 1: Structural Features



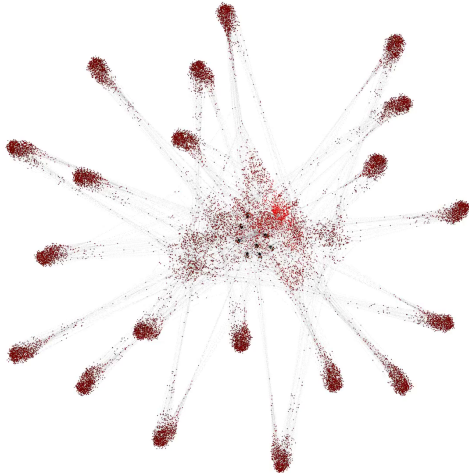


Method 1: Structural Features



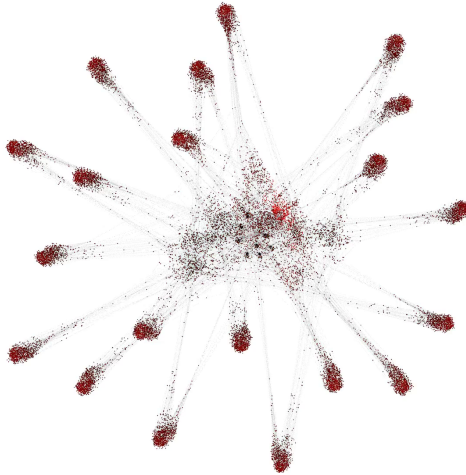


Method 1: Structural Features





Method 1: Structural Features





Method 1: Structural Features

- **Ratio of Gate Functionalities**

Graphs have a distinctive distribution of gates.

- **Centrality Measures**

How do gates affect transfer of data?

Which and how many gates strongly affect transfer of data?

- ▶ Betweenness Centrality
- ▶ Closeness Centrality
- ▶ Eigenvector Centrality
- ▶ Pagerank
- ▶ Degree Centrality
- ▶ ...



Method 1: Structural Features

- **Ratio of Gate Functionalities**

Graphs have a distinctive distribution of gates.

- **Centrality Measures**

How do gates affect transfer of data?

Which and how many gates strongly affect transfer of data?

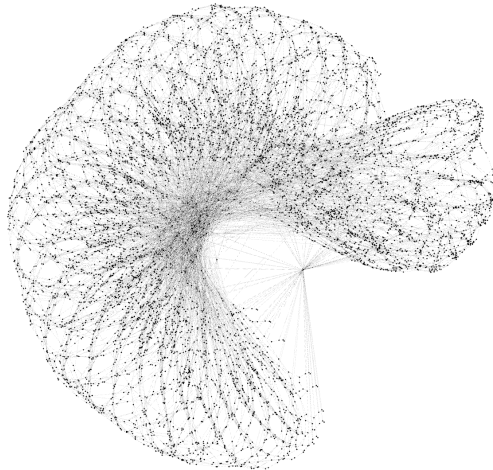
- ▶ Betweenness Centrality
- ▶ Closeness Centrality
- ▶ Eigenvector Centrality
- ▶ Pagerank
- ▶ Degree Centrality
- ▶ ...

- **Distribution of Degree**

Uniform number of connections vs. control logic.



Method 1: Structural Features





Method 1: Structural Features

- **Ratio of Gate Functionalities**

Graphs have a distinctive distribution of gates.

- **Centrality Measures**

How do gates affect transfer of data?

Which and how many gates strongly affect transfer of data?

- ▶ Betweenness Centrality
- ▶ Closeness Centrality
- ▶ Eigenvector Centrality
- ▶ Pagerank
- ▶ Degree Centrality
- ▶ ...

- **Distribution of Degree**

Uniform number of connections vs. control logic.

- **Density of graph**

Interconnectedness of all gates in the graph.

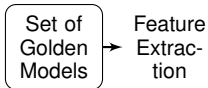


Method 1: Training & Testing

Set of
Golden
Models

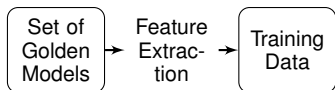


Method 1: Training & Testing



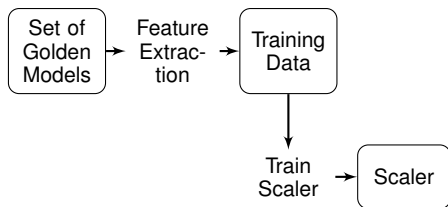


Method 1: Training & Testing



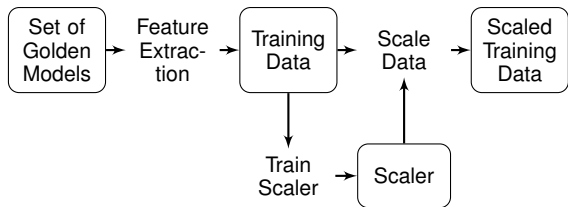


Method 1: Training & Testing



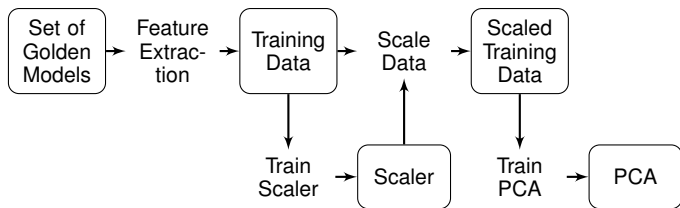


Method 1: Training & Testing



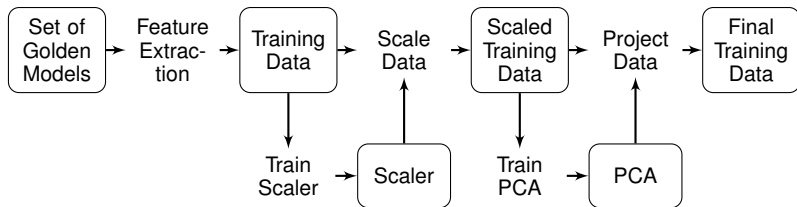


Method 1: Training & Testing



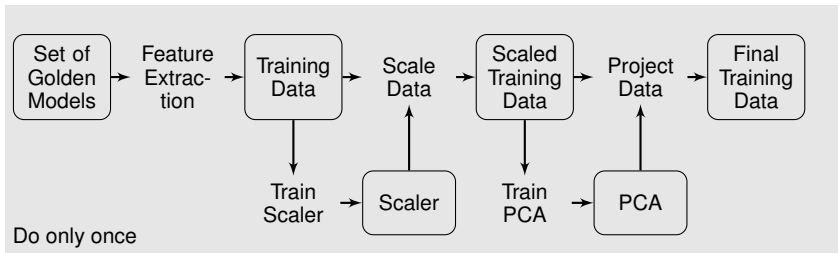


Method 1: Training & Testing



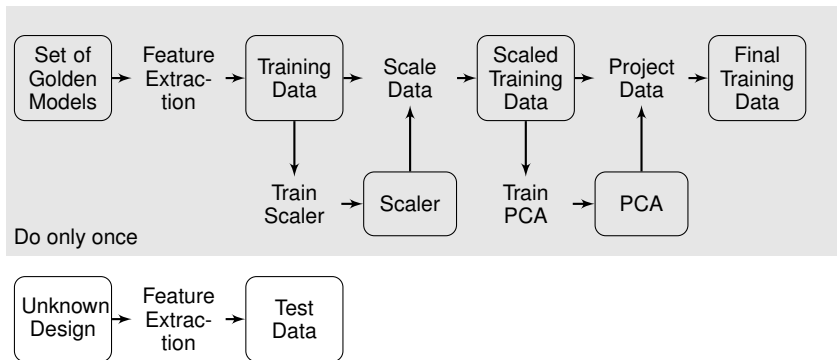


Method 1: Training & Testing

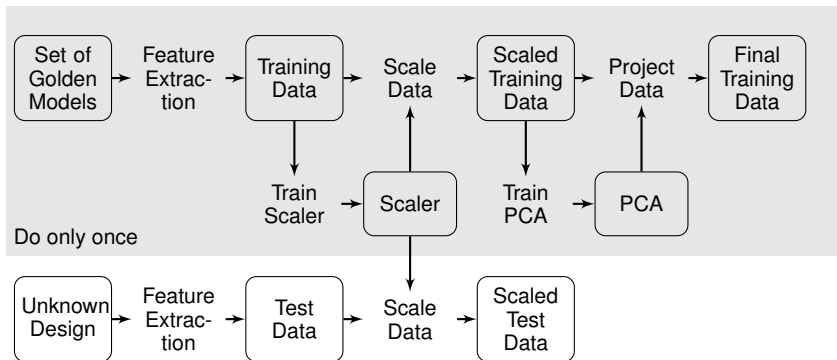




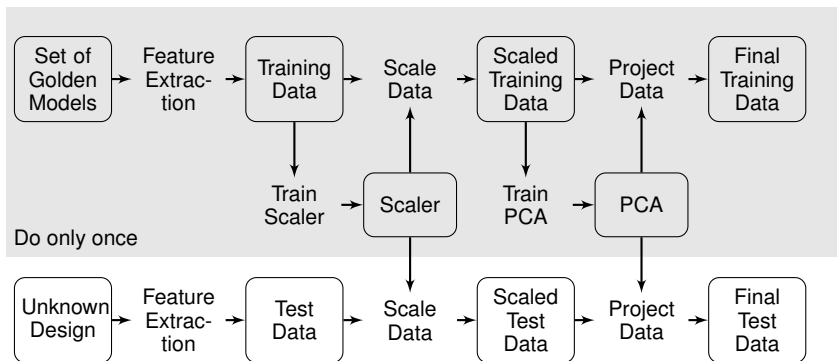
Method 1: Training & Testing



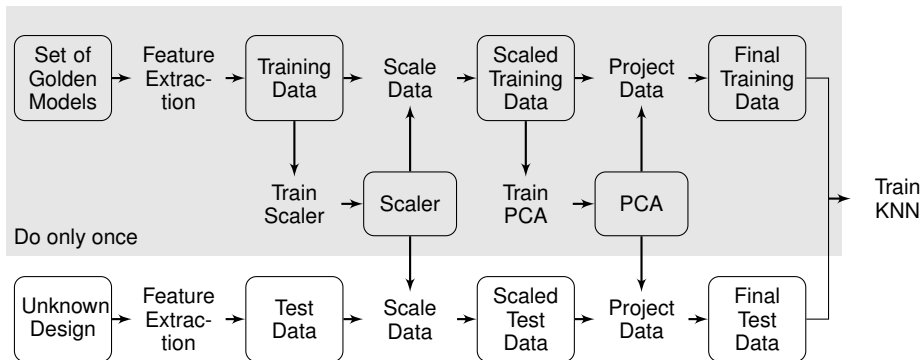
Method 1: Training & Testing



Method 1: Training & Testing



Method 1: Training & Testing





Method 2: Structural Hashing

Similarity Measure: Hash



Method 2: Structural Hashing

Similarity Measure: Hash

- Each gate functionality is assigned a value.
- For each gate, the values of the predecessors are combined.
- Each subsequent level is less important.
- $\mathcal{H}(gate)$ **depends on structure and functionality of predecessors.**



Method 2: Structural Hashing

Similarity Measure: Hash

- Each gate functionality is assigned a value.
- For each gate, the values of the predecessors are combined.
- Each subsequent level is less important.
- $\mathcal{H}(gate)$ **depends on structure and functionality of predecessors.**

Heuristic: Is Graph Isomorphism likely?



Method 2: Structural Hashing

Similarity Measure: Hash

- Each gate functionality is assigned a value.
- For each gate, the values of the predecessors are combined.
- Each subsequent level is less important.
- $\mathcal{H}(\text{gate})$ **depends on structure and functionality of predecessors.**

Heuristic: Is Graph Isomorphism likely?

- If $\mathcal{H}(\text{gate}_{\text{GoldenModel}}) = \mathcal{H}(\text{gate}_{\text{Design}})$, check surrounding gates.
- Repeat for different gate functionality values.
- Likelihood of graph isomorphism depends on number of matched gates.



Combination of Methods

- No precomputed features set required.
- Golden model library can be updated for each run.
- Recalculation of Hashes for test and golden model library during run time.



Combination of Methods

- No precomputed features set required.
- Golden model library can be updated for each run.
- Recalculation of Hashes for test and golden model library during run time.

- Parameter choice is not trivial.
- False positives, but few false negatives.



Combination of Methods

- No precomputed features set required.
- Golden model library can be updated for each run.
- Recalculation of Hashes for test and golden model library during run time.

- Parameter choice is not trivial.
- False positives, but few false negatives.

- **Refinement Method**

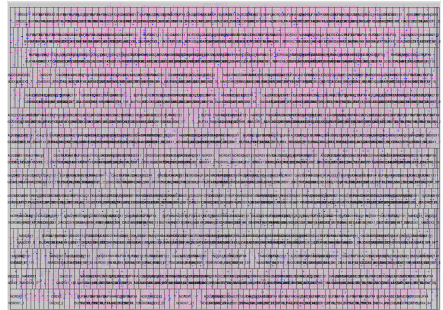


The Data Set



Simulating Errors

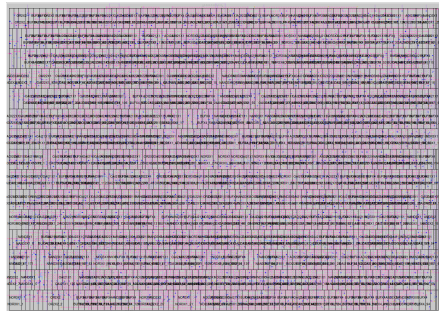
Netlist Extraction Errors





Simulating Errors

Netlist Extraction Errors Connection Error (random)

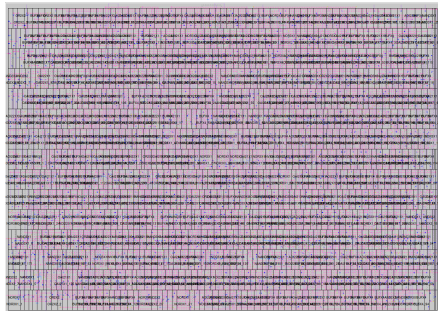




Simulating Errors

Netlist Extraction Errors

- Connection Error (random)**
- Connection Error (layout)**





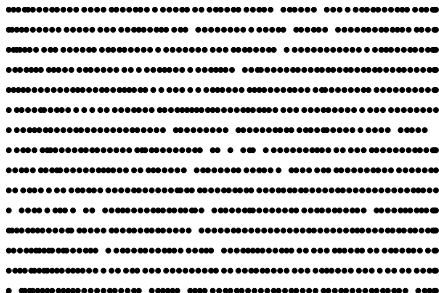
Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal





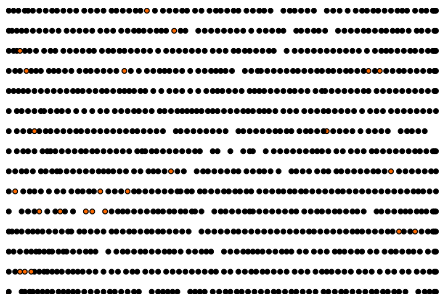
Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal





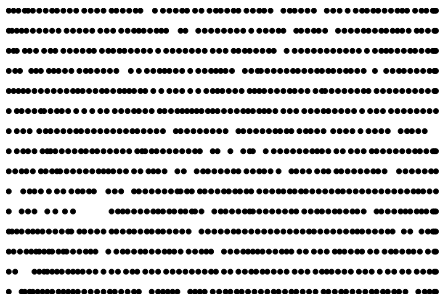
Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal





Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal

Partitioning Errors

$$\text{Sensitivity} = \frac{\# \text{ correct gates in partition}}{\# \text{ gates in ideal partition}}$$

$$\text{Overhead} = \frac{\# \text{ incorrect gates in partition}}{\# \text{ gates in ideal partition}}$$



Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal

Partitioning Errors

Simulated Errors:

- Iterative removal of gates at border
- Reduces sensitivity, overhead unchanged

$$\text{Sensitivity} = \frac{\# \text{ correct gates in partition}}{\# \text{ gates in ideal partition}}$$

$$\text{Overhead} = \frac{\# \text{ incorrect gates in partition}}{\# \text{ gates in ideal partition}}$$



Simulating Errors

Netlist Extraction Errors

Connection Error (random)

Connection Error (layout)

Speckled: Gate Removal

Partitioning Errors

Simulated Errors:

- Iterative removal of gates at border
- Reduces sensitivity, overhead unchanged

Realistic Errors:

- Cluster based partitions
- Reduces sensitivity, increases overhead

$$\text{Sensitivity} = \frac{\# \text{ correct gates in partition}}{\# \text{ gates in ideal partition}}$$

$$\text{Overhead} = \frac{\# \text{ incorrect gates in partition}}{\# \text{ gates in ideal partition}}$$



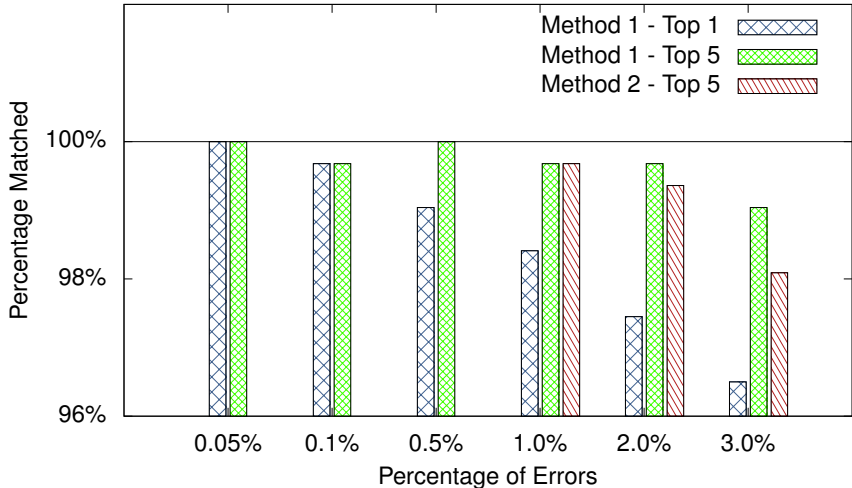
Data Set

	#Designs	Gate Count	Error Percentage
Golden Model	628	800 - 50,000	
Connection Error (random)			
Connection Error (layout)	1,884	800 - 50,000	0.05% - 3.00%
Speckled Error			
Simulated Partitions	1,884	800 - 50,000	0.05% - 3.00%
Realistic Partitions	1,892	800 - 25,000	$\geq 80\%$ Sensitivity $\leq 100\%$ Overhead

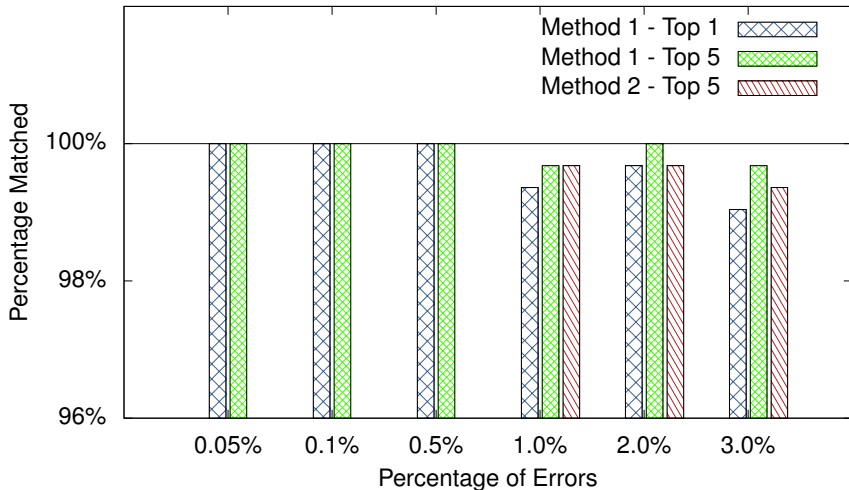


The Results

Results Random Connection Error

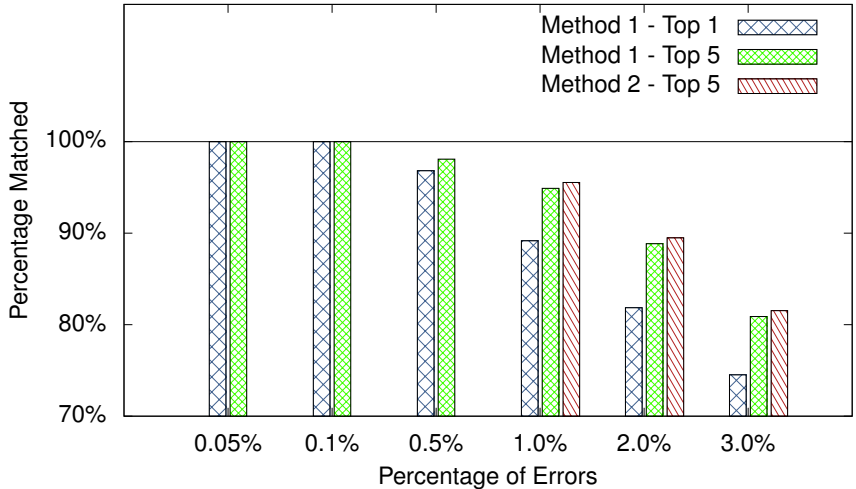


Results Layout based Connection Error

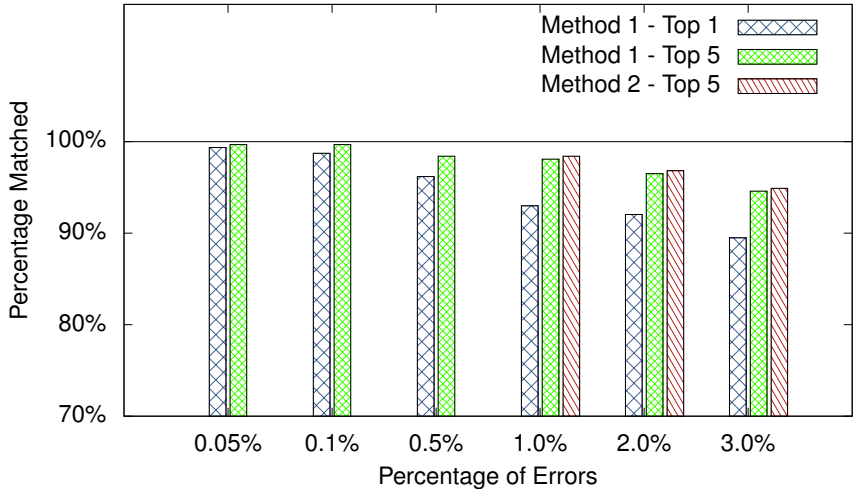




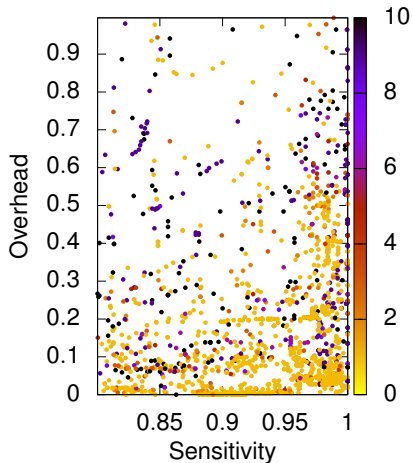
Results Speckled



Results Simulated Partitions



Results Realistic Partitions



$$\text{Sensitivity} = \frac{\# \text{ correct gates in partition}}{\# \text{ gates in ideal partition}}$$

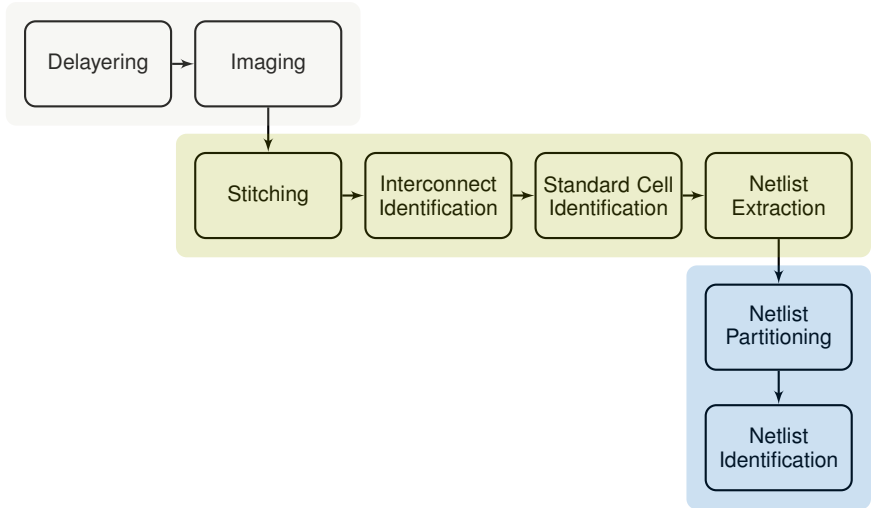
$$\text{Overhead} = \frac{\# \text{ incorrect gates in partition}}{\# \text{ gates in ideal partition}}$$



The Conclusion

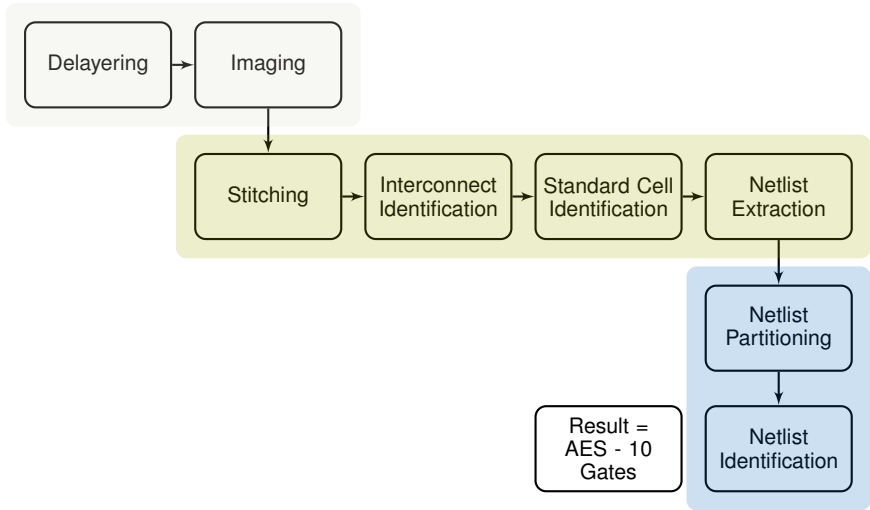


What happens next?



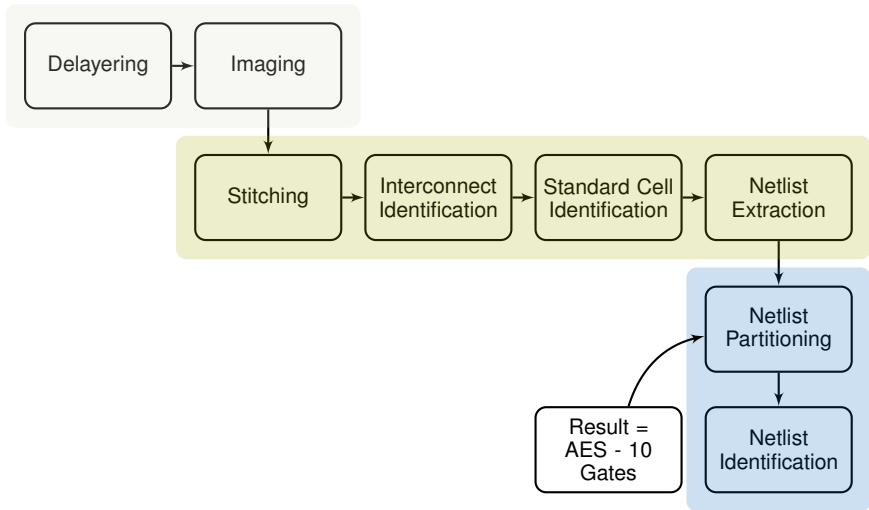


What happens next?



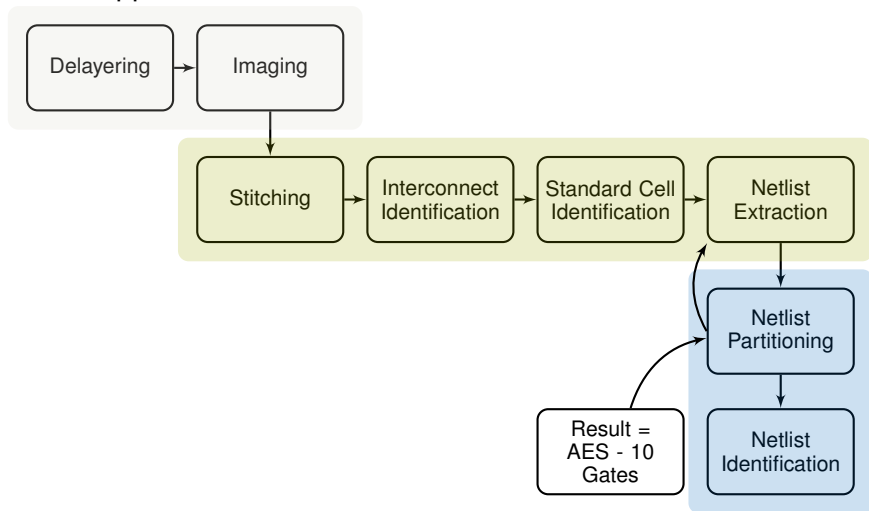


What happens next?



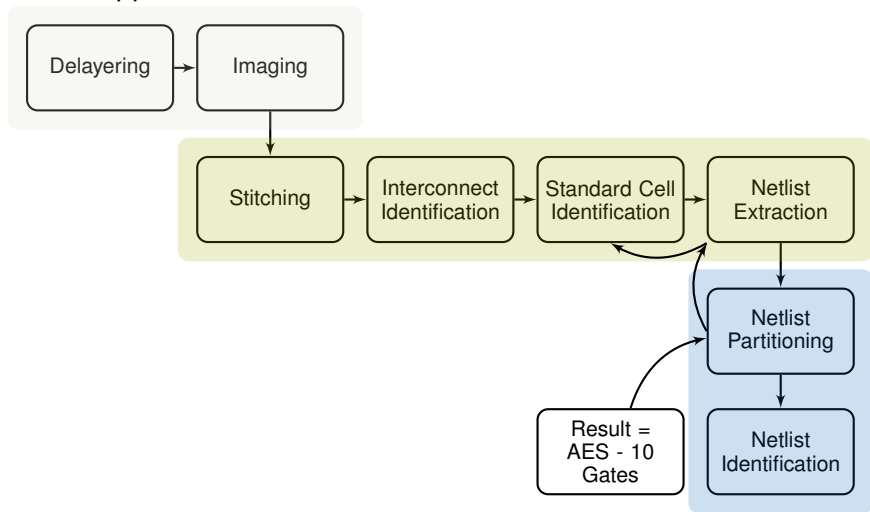


What happens next?



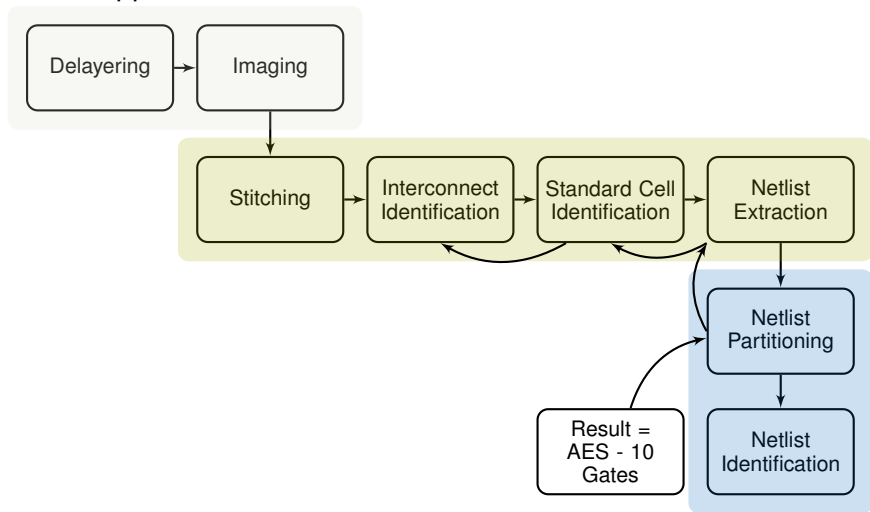


What happens next?



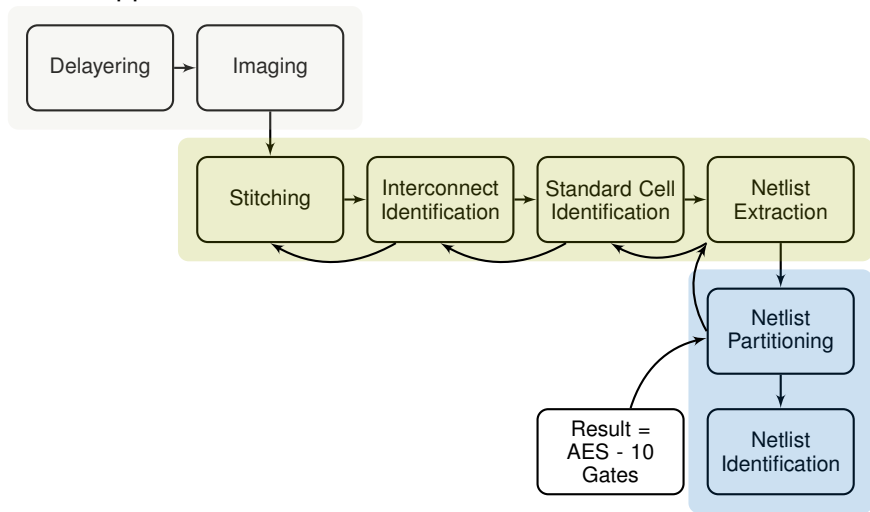


What happens next?



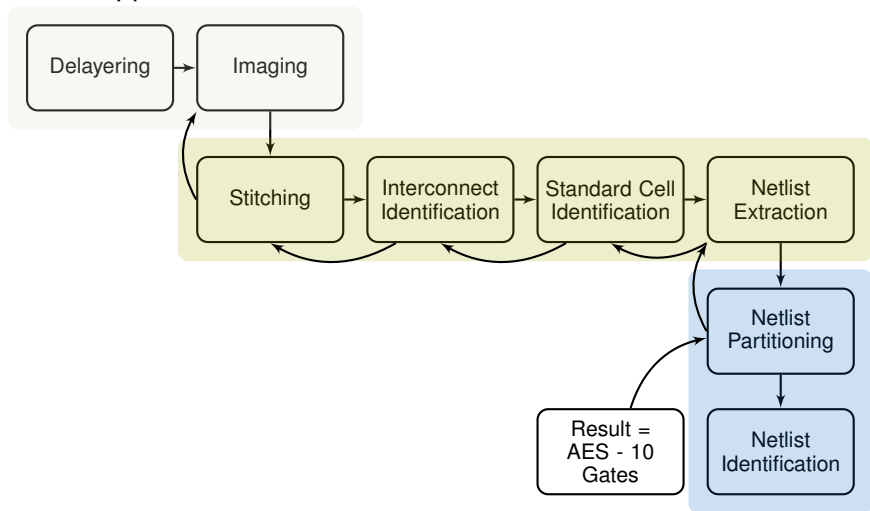


What happens next?



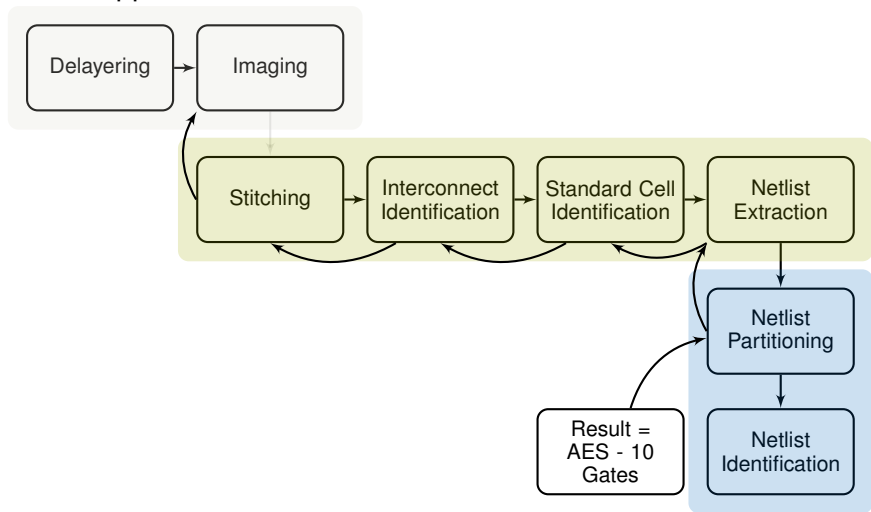


What happens next?



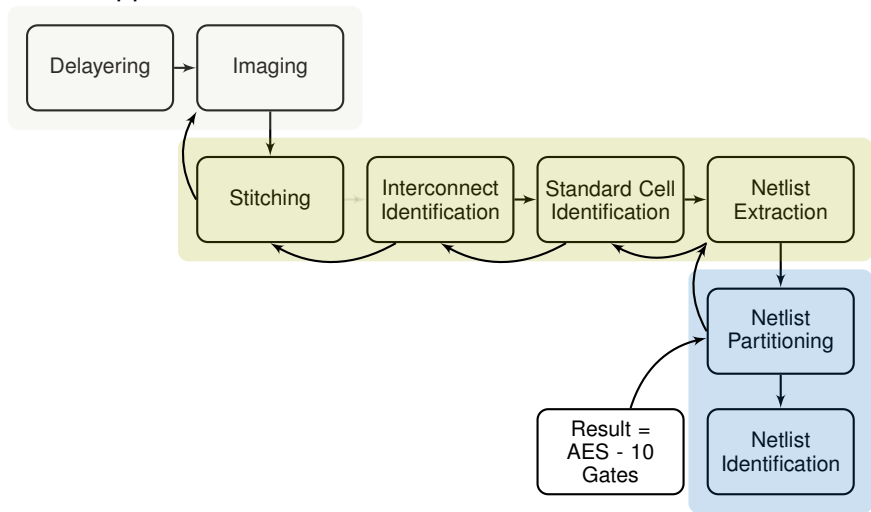


What happens next?



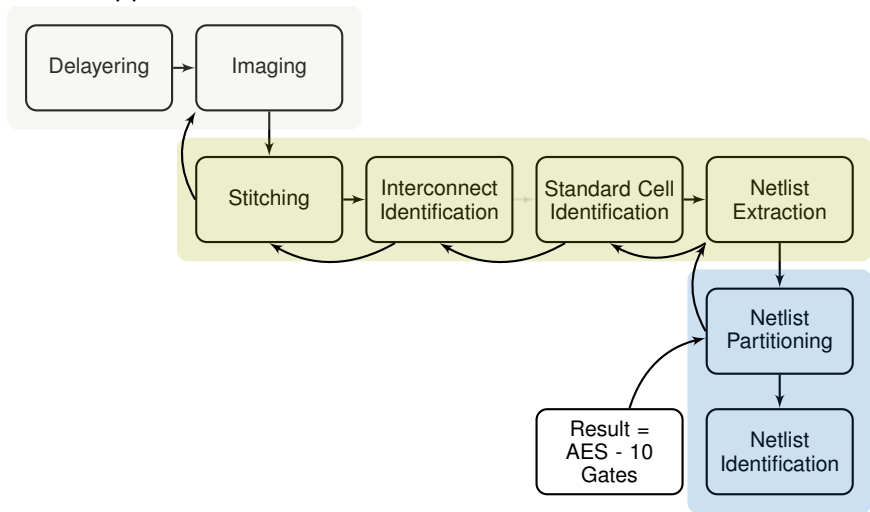


What happens next?



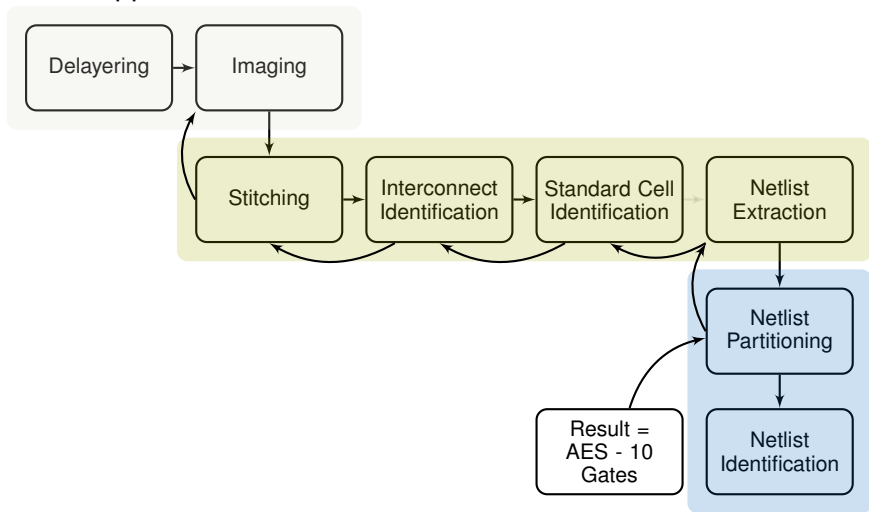


What happens next?



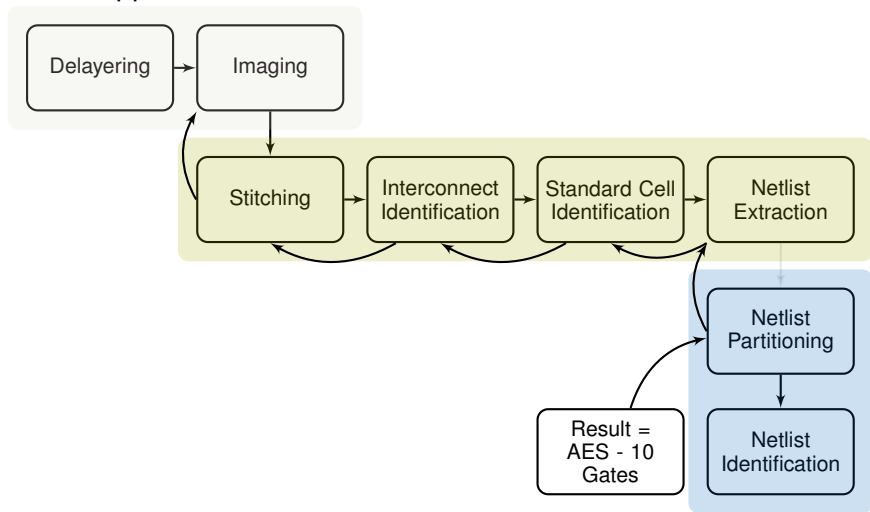


What happens next?



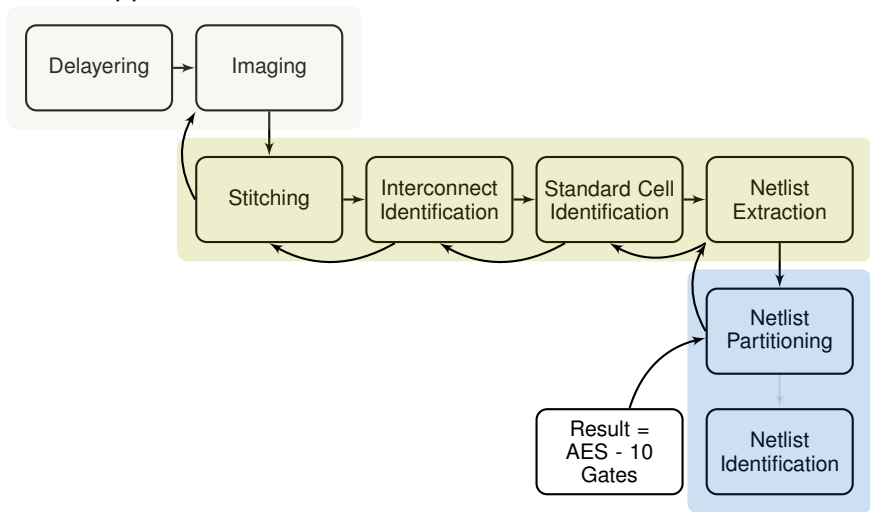


What happens next?



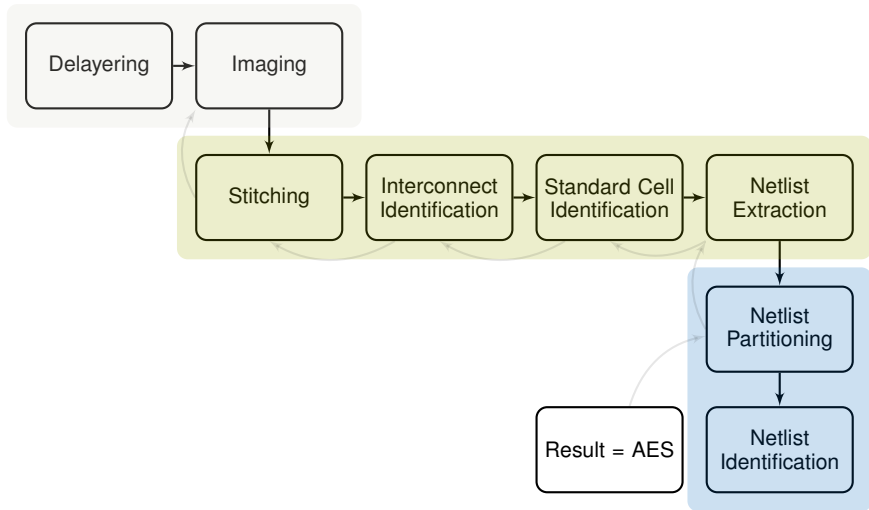


What happens next?





What happens next?





Take-aways and Discussion

- **Errors do occur**, both during netlist extraction and partitioning.
- Formal Verification based **equivalence checking is no longer viable**.
- Fuzzy methods are a first important step for a realistic RE work flow.



Take-aways and Discussion

- **Errors do occur**, both during netlist extraction and partitioning.
- Formal Verification based **equivalence checking is no longer viable**.
- Fuzzy methods are a first important step for a realistic RE work flow.

- **Structural Features can be used** to identify golden model designs for an erroneous netlist.
- Machine Learning helps to deal with large number of structural features.
- **Back-tracing is possible**, and advisable to identify errors.



Take-aways and Discussion




- **Errors do occur**, both during netlist extraction and partitioning.
- Formal Verification based **equivalence checking is no longer viable**.
- Fuzzy methods are a first important step for a realistic RE work flow.

- **Structural Features can be used** to identify golden model designs for an erroneous netlist.
- Machine Learning helps to deal with large number of structural features.
- **Back-tracing is possible**, and advisable to identify errors.

- More $\left\{ \begin{array}{l} \text{Designs} \\ \text{Features} \\ \text{Machine Learning Algorithms} \\ \text{Cell Libraries} \end{array} \right\}$ are required!






References

-  P. Subramanyan, N. Tsiskaridze, Wenchao Li, A. Gascon, Wei Yang Tan, A. Tiwari, N. Shankar, S.A. Seshia, and S. Malik.
Reverse engineering digital circuits using structural and functional analyses.
Emerging Topics in Computing, IEEE Transactions on, 2(1):63–80, March 2014.
-  J. Couch, E. Reilly, M. Schuyler, and B. Barrett.
Functional block identification in circuit design recovery.
In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 75–78, May 2016.
-  M. Werner, B. Lippmann, J. Baehr, and H. Gräß.
Reverse engineering of cryptographic cores by structural interpretation through graph analysis.
In *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pages 13–18, July 2018.



References

-  Travis Meade, Kaveh Shamsi, Thao Le, Jia Di, Shaojie Zhang, and Yier Jin. The Old Frontier of Reverse Engineering: Netlist Partitioning. *Journal of Hardware and Systems Security*, 2(3):201–213, September 2018.
-  A. Gascon, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanovic, and S. Malik. Template-based circuit understanding. In *Formal Methods in Computer-Aided Design (FMCAD)*, 2014, pages 83–90, Oct 2014.
-  Burcin Cakir and Sharad Malik. Reverse engineering digital ics through geometric embedding of circuit graphs. *ACM Trans. Des. Autom. Electron. Syst.*, 23(4):50:1–50:19, July 2018.