AI Techniques in IC Image and Netlist Analysis for Hardware Assurance

Bah-Hwee Gwee
ebhgwee@ntu.edu.sg

Nanyang Technological University, Singapore
27 Mar 2024
Outline

- Introduction
- IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification
- Conclusions
Outline

Introduction

IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow

IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification

Conclusions
Introduction – IC Circuit Extraction

The objectives of IC Circuit Extraction:
- Intellectual Property (IP) infringement investigation;
- Detection of malicious hardware, e.g. hardware Trojans;
- Hardware failure analysis
Introduction – IC Circuit Extraction

Packaged IC → Package Removal → Delayering

IC Netlist Analysis → IC Image Analysis → Imaging
Introduction – Delayered IC Images

Major Challenges:

- Millions of images per layer
- Image variations across layers and regions
- Anomalies from sample preparation & imaging
- Small feature size / narrow gap between features
Outline

- **Introduction**
- **IC Image Analysis**
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- **IC Netlist Analysis**
  - Netlist Partition
  - Netlist Identification
- **Conclusions**
IC Image Analysis – Tasks

- Tasks for IC image analysis
  - Image stitching (and results evaluation)
  - Feature extraction (segmentation & object detection)
  - Image stacking (and netlist generation)

Sample delayered IC images taken by SEM (Scanning Electron Microscopy) from different chips, different layers, and with different imaging settings.
IC Image Analysis – Goals

- Techniques that are **less human-dependent**
  - The human involvement should be limited.
- Techniques that are **not data-hungry**
  - Less data labelling is desired.
- Techniques that are **adaptable**
  - Continuous data labelling for different chips, layers and imaging settings, is not acceptable.
Advantages of Deep Learning (DL)-based Methods

- Fast due to inherent leverage on parallel processing hardware (e.g. CNNs on GPUs)
- Learning extraction rules from large amount of data ensures robust extraction against image noise
- End-to-end training and inference allows realization of complex tasks
Data Analysis and Preparation in DL-Based IC Image Analysis
Outline

- Introduction

- IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow

- IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification

- Conclusions
Anomalies in Image Data

Dataset (one layer) → Inspection & Selection → Samples → Manual / Semi-auto → Label

- Anomaly Detection
- Augmentation
- Image Clustering
- Image Synthesis

Standard Cell Detection → Interconnect Segmentation → DL Model

Domain Adaptation → Future Dataset → Evaluation

Samples

Images:
- Normal patterns
- Anomalies

Annotations:
- Bounding box
- Pixel-wise mask
Self-supervised Anomaly Detection with GAN (Generative Adversarial Networks)

A low-dimensional representation
Self-supervised Anomaly Detection with GAN

Training Stage:
- Alternating training between Encoder/Decoder (Generator) and Discriminator
- Generator is optimized with weighted sum of 4 loss terms
- Discriminator is optimized with adversarial loss

Testing Stage:
- Reconstruction loss, z-loss, and feature loss are computed for patches of input images
- 3 loss values are normalized as anomaly scores to determine anomalous images.
Self-supervised Anomaly Detection: Score Ranking

Low anomaly score

Medium anomaly score

High anomaly score
Self-supervised Anomaly Detection: Score Ranking

- Low anomaly score
- Medium anomaly score
- High anomaly score
Self-supervised Anomaly Detection: Accuracy

Anomalous regions are highlighted by high loss values (anomaly score).

Better performance than reported methods, without supervision/data labeling.

<table>
<thead>
<tr>
<th>Method</th>
<th>AUC</th>
<th>F1</th>
<th>TPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet f-anoGAN [17]</td>
<td>0.5623</td>
<td>0.3013</td>
<td>0.1896</td>
<td>0.0063</td>
</tr>
<tr>
<td>ConvNet f-anoGAN</td>
<td>0.5638</td>
<td>0.3165</td>
<td>0.1896</td>
<td>0.0008</td>
</tr>
<tr>
<td>GANomaly [18]</td>
<td>0.9334</td>
<td>0.7464</td>
<td>0.6724</td>
<td>0.0118</td>
</tr>
<tr>
<td>Ours (IAD only)</td>
<td>0.9728</td>
<td>0.8348</td>
<td>0.7845</td>
<td>0.0086</td>
</tr>
</tbody>
</table>

Joint Task on Anomaly Detection & Inpainting

Concurrent anomaly detection and inpainting

By adding pairs of corrupted and corresponding clean images into training

<table>
<thead>
<tr>
<th>Model</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDBP [26]</td>
<td>31.3390</td>
<td>0.9575</td>
</tr>
<tr>
<td>IRCNN [6]</td>
<td>31.2245</td>
<td>0.9586</td>
</tr>
<tr>
<td>TSLRA [5]</td>
<td>30.4554</td>
<td>0.9563</td>
</tr>
<tr>
<td>Ours (IAD + Inpainting)</td>
<td><strong>34.4798</strong></td>
<td><strong>0.9627</strong></td>
</tr>
</tbody>
</table>

Good performance on image inpainting

<table>
<thead>
<tr>
<th>Method</th>
<th>AUC</th>
<th>F1</th>
<th>TPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet f-anoGAN [17]</td>
<td>0.5623</td>
<td>0.3013</td>
<td>0.1896</td>
<td>0.0063</td>
</tr>
<tr>
<td>ConvNet f-anoGAN</td>
<td>0.5638</td>
<td>0.3165</td>
<td>0.1896</td>
<td>0.0008</td>
</tr>
<tr>
<td>GANomaly [18]</td>
<td>0.9334</td>
<td>0.7464</td>
<td>0.6724</td>
<td>0.0118</td>
</tr>
<tr>
<td>Ours (IAD only)</td>
<td>0.9728</td>
<td>0.8348</td>
<td>0.7845</td>
<td>0.0086</td>
</tr>
<tr>
<td>Ours(IAD+Inpainting)</td>
<td><strong>0.9927</strong></td>
<td><strong>0.9123</strong></td>
<td>0.8966</td>
<td>0.0063</td>
</tr>
</tbody>
</table>

Further improve image anomaly detection

Outline

- **Introduction**

- **IC Image Analysis**
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow

- **IC Netlist Analysis**
  - Netlist Partition
  - Netlist Identification

- **Conclusions**
**Image Stitching Stage:**
- Stitch SEM images using phase correlation
- Use reported DL object detection model to check stitching results and detect misalignment
- Use a fully automated method to prepare synthetic training data for detection

**Feature Extraction Stage:**
- Use reported DL object detection model to detect standard cells
- Use reported DL semantic segmentation model to segment contacts, vias and metal lines
- Use a fully automated method to prepare synthetic training data for detection and a semi-automated method to prepare training data for segmentation

**Image Stacking Stage:**
- Use custom DL regression model to estimate stacking movement
- Use a fully automated method to prepare synthetic training data for regression
Experiment Results: Misalignment Detection

Sample DL-based Misalignment Detection Result

DL-based Misalignment Detection

- Stitching misalignments > ~2 pixels were correctly identified by our DL object detection model with high accuracy
- Fast processing speed: ~0.5 seconds to process an image of size 1,600×1,600 pixels (on GPU)
Experiment Results: Standard Cell Detection

- Multiple instances of standard cells were correctly identified by our DL object detection model with high accuracy.
- Fast processing speed: ~3 seconds to process an image of size 10,000×10,000 pixels (on GPU).

Sample DL-based Standard Cell Detection Result
Experiment Results: Via/Contact Segmentation

- Vias and contacts were correctly segmented by our DL semantic segmentation model; models achieved high pixel accuracy >97%
- Fast processing speed: ~0.4 seconds to process an image of size 1,024×1,024 pixels (on GPU)
Experiment Results: Metal Line Segmentation

Sample DL-based Metal Line Segmentation Result

DL-based Metal Line Segmentation

- Metal lines were correctly segmented by our DL semantic segmentation model; models achieved high pixel accuracy >97%
- Fast processing speed: ~0.4 seconds to process an image of size 1,024×1,024 pixels (on GPU)
Experiment Results: Image Stacking

DL-based Image Stacking

- Stacking movements were correctly estimated by our DL regression model and vias from the lower layer were correctly aligned to metal lines from the upper layer; can move up to 50 pixels in both directions
- Fast processing speed: ~0.8 seconds to process an image of size 980×980 pixels (on GPU)
Experiment Results: Robust Extraction against Image Noises

- Metal line segmentation - Charging Noise was correctly rejected by our DL model
Experiment Results: Robust Extraction against Image Noises

- Dust noise was correctly rejected by our DL model.

Via segmentation result - Dust Noise was correctly rejected by our DL model.
Experiment Results: Comparison of DL-based Method with Classical Image Processing Techniques

Comparison Results of Via Annotation Errors Using DL Model and Image Processing Techniques (CHT = Circular Hough Transform)

<table>
<thead>
<tr>
<th>Method</th>
<th>using DL model</th>
<th>using image processing technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CHT Sensitivity=0.85</td>
</tr>
<tr>
<td><strong>Errors/image</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>0.47</td>
<td>6.47</td>
</tr>
<tr>
<td><strong>FN</strong></td>
<td>0.02</td>
<td>11.36</td>
</tr>
</tbody>
</table>

- Achieved much lower False Positive (FP) and False Negative (FN) Errors

Experiment Results: Comparison of DL-based Method with Classical Image Processing Techniques

Comparison Results of Metal Line Annotation Errors Using DL Model and Image Processing Techniques

<table>
<thead>
<tr>
<th>Errors/image</th>
<th>Method using DL model</th>
<th>using image processing technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Median filtering</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neighbourhood size=12</td>
</tr>
<tr>
<td>Short-circuit errors</td>
<td>0.83</td>
<td>4.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neighbourhood size=15</td>
</tr>
<tr>
<td>Open-circuit errors</td>
<td>0.26</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Achieved much lower **Short-Circuit** and **Open-Circuit Errors**

Outline

Introduction

IC Image Analysis
- Self-Supervised Anomaly Detection
- DL-Based Image Analysis Flow

IC Netlist Analysis
- Netlist Partition
- Netlist Identification

Conclusions
IC Netlist Analysis – Tasks

- Modern SoC netlists consist of many functional blocks and sub-circuits:
  - Difficult to analyse as a whole.
  - Not all functional blocks or sub-circuits are of interest.

- A ‘divide-and-conquer’ approach is usually adopted, which consists of:
  - Netlist Partition: to partition a large circuit netlist into smaller sub-circuits.
  - Netlist Identification: to identify the functionality of a sub-circuit.
Outline

- **Introduction**
- **IC Image Analysis**
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- **IC Netlist Analysis**
  - Netlist Partition
  - Netlist Identification
- **Conclusions**
Netlist Partition: The Problem

- To solve the ‘Normalized-cut’ (N-cut) graph partition/clustering problem:
  - Observation: sub-circuits have more connections within than in-between.
  - To ‘cut’ as little connections as possible yet to have meaningful size for each partition.

\[
n\text{-}cut = \frac{1}{k} \sum_{i=1}^{k} \frac{\text{link}(V_i, V_i \backslash V)}{\text{link}(V_i, V)}
\]

- Existing methods and issues:
  - N-cut problem is NP-hard and its solution is usually approximated.
  - Existing methods either do not optimize for n-cut directly such as spectral clustering or may stuck at local minima such as methods based on iterative search algorithms.
  - Further, existing methods only leverage on connectivity but not node features.
Graph Neural Network (GNN) for Netlist Partition

• **Advantages of GNN:**
  – GNN leverages on both connectivity and node features.
  – Can optimize for an objective function (e.g. N-cut) directly as a loss function (unsupervised setting).

• **Challenges of GNN:**
  – GNN is inherently local and deep architecture is difficult.
  – Need to find a meaningful node feature for the intended task.

• **We propose a novel GNN for netlist partition named ‘GraphClusNet’:**
  – A novel hierarchical architecture which finds clusters from local to global.
  – An n-cut-based loss function to optimize for the objective function directly.
  – A location-based node feature which suits the partition task and avoids local minima.
Proposed Architecture

• Multi-stage hierarchical architecture:
  – Intuition: sub-circuits group **hierarchically** into larger circuits.
  – Optimize for ‘n-cut’ loss at each stage.
  – Final stage can perform either **bipartition** or **multiway** partition.
Proposed Loss Function

• ‘N-cut’ based loss function:
  – The numerator computes the intra-cluster connections of each cluster.
  – The denominator computes the total connections of each cluster.
  – Effectively searches for clusters that have more connections within and less connections in-between.

\[
\mathcal{L}_{\text{ncut}} = 1 - \frac{\text{Diag}(S^T A S)}{\text{Diag}(S^T D S)}
\]

where \(A\) is the adjacency matrix, \(D\) is the degree matrix, and \(S\) is the cluster assignment matrix.

• Allows direct optimization of the N-cut objective function.
Proposed Node Feature

- **Location-based node feature:**

Intuition: logic gates from the same sub-circuit tend to locate close to each other on the floorplan.

- Divide floorplan into squares of different sizes.

- Assign node feature to nodes based on their location number at each square size.

- Effectively provides a node feature where nodes close to each other have more similar entries.
Partition Results: Bipartition on SoC Netlists

- Performed bipartition on real FPGA SoC circuit netlists:
  - To extract major functional block from a netlist.
  - Our proposed GraphClusNet achieved highest NMI and usually lowest n-cut among competing methods.
  - It avoided local minima and can obtain more meaningful partitions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8051 SoC</td>
<td>NMI</td>
<td>1</td>
<td>0.579±0.297</td>
<td>0.891±0.023</td>
<td>0.752±0.248</td>
<td>0.823±0.018</td>
<td>0.867±0.232</td>
<td>0.967±0.030</td>
<td>0.965±0.032</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>1.060</td>
<td>2.664±1.574</td>
<td>1.289±0.082</td>
<td>1.289±0.079</td>
<td>3.227±0.441</td>
<td>1.037±0.041</td>
<td>1.009±0.065</td>
<td>1.026±0.084</td>
</tr>
<tr>
<td>ARM CORTEX SoC</td>
<td>NMI</td>
<td>1</td>
<td>0.982±0.002</td>
<td>0.946±0.004</td>
<td>0.986±0.003</td>
<td>0.858±0.0017</td>
<td>0.963±0.038</td>
<td>0.987±0.006</td>
<td>0.990±0.000</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>1.376</td>
<td>1.397±0.041</td>
<td>2.511±1.771</td>
<td>1.364±0.000</td>
<td>3.170±0.422</td>
<td>1.511±0.211</td>
<td>1.362±0.000</td>
<td>1.356±0.000</td>
</tr>
<tr>
<td>RISC-V-I SoC</td>
<td>NMI</td>
<td>1</td>
<td>0.858±0.101</td>
<td>0.838±0.018</td>
<td>0.805±0.055</td>
<td>0.581±0.055</td>
<td>0.886±0.070</td>
<td>0.928±0.009</td>
<td>0.921±0.008</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>2.940</td>
<td>3.557±0.962</td>
<td>5.851±3.312</td>
<td>3.145±0.251</td>
<td>9.132±1.032</td>
<td>2.867±0.114</td>
<td>2.794±0.046</td>
<td>2.787±0.025</td>
</tr>
<tr>
<td>RISC-V-IMSU SoC</td>
<td>NMI</td>
<td>1</td>
<td>0.850±0.016</td>
<td>0.869±0.083</td>
<td>0.798±0.076</td>
<td>0.210±0.067</td>
<td>0.847±0.034</td>
<td>0.857±0.064</td>
<td>0.896±0.075</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>2.775</td>
<td>3.775±0.288</td>
<td>11.55±14.76</td>
<td>3.010±0.090</td>
<td>27.34±13.16</td>
<td>3.607±0.545</td>
<td>3.629±0.284</td>
<td>2.883±0.090</td>
</tr>
<tr>
<td>RISC-V-IMZCSR SoC</td>
<td>NMI</td>
<td>1</td>
<td>0.865±0.055</td>
<td>0.886±0.005</td>
<td>0.856±0.078</td>
<td>0.349±0.122</td>
<td>0.930±0.055</td>
<td>0.986±0.005</td>
<td>0.988±0.005</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>2.254</td>
<td>2.871±0.539</td>
<td>5.149±7.039</td>
<td>2.603±0.246</td>
<td>17.11±6.762</td>
<td>2.539±1.089</td>
<td>2.268±0.046</td>
<td>2.257±0.043</td>
</tr>
<tr>
<td>openFPU</td>
<td>NMI</td>
<td>1</td>
<td>0.792±0.005</td>
<td>0.776±0.089</td>
<td>0.812±0.136</td>
<td>0.318±0.110</td>
<td>0.782±0.162</td>
<td>0.865±0.128</td>
<td>0.874±0.123</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>4.929</td>
<td>5.675±0.800</td>
<td>6.180±0.661</td>
<td>5.802±0.963</td>
<td>67.12±33.09</td>
<td>5.117±0.241</td>
<td>5.305±0.879</td>
<td>5.280±0.870</td>
</tr>
<tr>
<td>aoOCS$^3$</td>
<td>NMI</td>
<td>1</td>
<td>0.542±0.066</td>
<td>0.542±0.032</td>
<td>0.777±0.096</td>
<td>0.419±0.003</td>
<td>0.638±0.082</td>
<td>0.906±0.083</td>
<td>0.906±0.083</td>
</tr>
<tr>
<td></td>
<td>n-cut</td>
<td>1.605</td>
<td>38.95±4.239</td>
<td>24.33±1.813</td>
<td>1.730±0.876</td>
<td>107.5±0.666</td>
<td>2.788±0.479</td>
<td>1.756±0.785</td>
<td>1.739±0.771</td>
</tr>
</tbody>
</table>
Partition Results: Multiway Partition

- Performed multiway partition on 8051 microcontroller core netlist:
  - To extract multiple functional blocks from a netlist.
  - Our proposed GraphClusNet achieved highest NMI and F1-score among competing methods.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>456</td>
<td>F1-score</td>
<td>0.8896±0.0387</td>
<td>0.9251±0.0294</td>
<td>0.9270±0.0074</td>
<td>0.6661±0.0109</td>
</tr>
<tr>
<td>SFR</td>
<td>1027</td>
<td>F1-score</td>
<td>0.8968±0.0263</td>
<td>0.7658±0.1074</td>
<td>0.8869±0.1013</td>
<td>0.7216±0.0110</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>494</td>
<td>F1-score</td>
<td>0.5805±0.0986</td>
<td>0.5489±0.1242</td>
<td>0.7125±0.1324</td>
<td>0.5767±0.0188</td>
</tr>
<tr>
<td>Decoder</td>
<td>252</td>
<td>F1-score</td>
<td>0.6738±0.1434</td>
<td>0.6426±0.0810</td>
<td>0.8221±0.1580</td>
<td>0.5928±0.0070</td>
</tr>
<tr>
<td>8051 Core</td>
<td>2229</td>
<td>NMI</td>
<td>0.5966±0.0574</td>
<td>0.5621±0.0329</td>
<td>0.6574±0.0683</td>
<td>0.4742±0.0070</td>
</tr>
</tbody>
</table>

Visualization of Partition Results

- We visualized node embeddings after each stage of GNN:
  - Local clusters were merged into higher level clusters.
  - Cluster purity also improved at higher levels.

![t-SNE Visualization of Node Embeddings after Each Stage of GNN (8051 SoC)](image)

Outline

- Introduction
- IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification
- Conclusions
Netlist Identification: The Problem

- To identify the functionality of a flattened netlist:
  - Used to be done manually with expert knowledge.
  - Observation: different circuit graphs have distinctive structures and gate compositions.
  - Netlist identification problem may thus be formulated as a graph classification problem using machine-learning methods.
GNN for Netlist Identification

- Train a GNN to classify unknown netlists into known classes:
  - Input is a circuit graph with **gate type** as node feature and output is a class label indicating the type of circuit.
  - Our GNN consists of two layers of Graph Convolutional Network (GCN).
Case Study: Adder Circuit Classification

- Classify four types of adder circuits:
  - Four adder structures: Ripple Carry Adder (RCA), Carry Look-Ahead Adder (CLA), Carry Select Adder (CSLA), and Carry Skip Adder (CSKA).

12-bit Ripple Carry Adder (RCA)

12-bit Carry Look-Ahead Adder (CLA)

FA = Full Adder
G = Generate
P = Propagate
Case Study: Adder Circuit Classification

12-bit Carry Select Adder (CSLA)

12-bit Carry Skip Adder (CSKA)

FA = Full Adder
P = Propagate
AND = AND Gate
MUX = Multiplexer
Data Preparation

- Synthesized circuit netlists of varying bit-widths for training and testing data:
  - Synthesized 4 types of adder circuits from 5-bit to 64-bit resulting a total of 240 netlists.
  - Used 40 netlists for training GNN and remaining 200 netlists for testing.
  - Used one-hot encoded gate type as node features.

- Different node colours represent different gate types
Graph Visualization of Adder Circuit Netlists

- 8-bit RCA
- 12-bit RCA
- 16-bit RCA
- 8-bit CLA
- 12-bit CLA
- 16-bit CLA
- 8-bit CSLA
- 12-bit CSLA
- 16-bit CSLA
- 8-bit CSKA
- 12-bit CSKA
- 16-bit CSKA
Netlist Classification Results

- Our GNN achieved high classification accuracy on unseen test data:
  - GNN achieved classification accuracy of 99% on unseen test data.
  - Graph embeddings of different class netlists grow separated after each layer of GNN demonstrating its discriminating power.

Conclusions

- Delayered IC image analysis is one of the most reliable approach to chip integrity and functionality assurance.
- With the problem scale and limited data, we aim to develop less data-hungry and adaptable deep learning algorithms for automatic IC image and netlist analysis.
- A self-supervised GAN-based network has been presented for concurrent IC image anomaly detection and inpainting.
- A deep learning-based framework for IC image analysis has been presented. Deep learning models can be effectively applied to retrieve the standard cells and interconnects in IC images.
- GNN has demonstrated some unique advantages over conventional machine-learning methods for netlist analysis including its ability to process graph connectivity together with node features.
- Novel GNN architectures for netlist partition and netlist identification have been presented.
Thank You!

Questions?