Robust Power Estimation and Simultaneous Switching Noise Prediction Methods Using Machine Learning

March 20th, 2019
Robust Simultaneous Switching Noise Prediction for Test using Deep Neural Network

Seyed Nima Mozaffari, Bonita Bhaskaran, Kaushik Narayanun
Ayub Abdollahian, Vinod Pagalone, Shantanu Sarangi

RTL-Level Power Estimation Using Machine Learning

Mark Ren, Yan Zhang, Ben Keller, Brucek Khailany
Yuan Zhou, Zhiru Zhang
Robust Simultaneous Switching Noise Prediction for Test using Deep Neural Network

Seyed Nima Mozaffari, Bonita Bhaskaran, Kaushik Narayanun
Ayub Abdollahian, Vinod Pagalone, Shantanu Sarangi
DFT - A BIRD’S EYE VIEW

- At-Speed Tests - verify performance
- Stuck-at Tests - detect logical faults
- Parametric Tests - verify AC/DC parameters
- Leakage Tests - catch defects that cause high leakage
SCAN TEST - SHIFT

Combinational Logic

Primary Inputs (SI)
Scan Enable (SE) = 1
Slow capture clk
Test Clk

Data In (D) -> Data Out (Q)
Scan In (SI) -> Scan Out (SO)

Clocks:
- Test Clk
- Slow capture clk

Timing:
- Shift
- Shift
- Dead Cycles (optional)
- PLL Clock Launch
- PLL Clock Capture
- Shift

Data:
- D
- Q
- SI
- SO

Primary Inputs
Primary Outputs

NVIDIA CONFIDENTIAL. DO NOT DISTRIBUTE.
SCAN TEST - CAPTURE

Combinational Logic

Primary Inputs

Scan In (SI) → Data SI

Scan Enable (SE) = 0

Slow capture clk

Test Clk

D Q

0
1

 clk

 clk

 clk

Scan Out (SO)

Primary Outputs

D Q

Primary Inputs

D Q

Primary Outputs

D Q

D Q

D Q
TEST WASTE FROM POWER NOISE

- Power balls overheated; Scan Freq target was lowered
  - Slower frequency $\rightarrow$ Test Cost
- Higher Vmin issue
  - Vmin thresholds had to be raised; impacts DPPM.
- During MBIST, overheating was observed
  - Serialized tests; increase in Test Time & Test Cost
- Vmin issues observed and being debugged
CAPTURE NOISE
Low Power Capture Controller
TEST NOISE ESTIMATION

The traditional way

Pre-Silicon Estimation

Post-Silicon Validation

Issues

- Can simulate only a handful of vectors
- Not easy to pick top IR-Drop inducing test patterns always
- Machine Time to simulate 3000 patterns is 6-7 years!
- Measurement is feasible for 3-5K patterns

Power noise during test <= functional budget directly impacts test quality!
Strategy – we pick conservative LPC settings!

- Higher Test Time $\rightarrow$ Higher Test Cost
- For example - Test Time savings of 40% could have been achieved.
Why is Deep Learning a good fit?

- Labeled data is available
- Precision is not the focus
- Need a prediction scheme that encompasses the entire production set
PROPOSED APPROACH

• Design Flow
• Feature Engineering
• Deep Learning Models
• Classification and Regression
PROPOSED APPROACH

• Design Flow
• Feature Engineering
• Deep Learning Models
• Classification and Regression
DESIGN FLOW

Goal:
- Supervised learning model to reduce the time and effort spent
- Most effective set of input features

Dataset:
- Input features $\rightarrow$ parameters that impact the $V_{\text{droop}}$
- Labels $\rightarrow$ $V_{\text{droop}}$ values from silicon measurements
- Train phase $\rightarrow$ train:80% & dev:10%
- Inference phase $\rightarrow$ test:10%

Addresses the following:
- Takes into account all the corner cases for PVT$f$ variations
- Helps predict achievable $V_{\text{min}}$
- Cuts down post-silicon measurements – typically 6-8 weeks of engineering effort
HARDWARE SET-UP AND SCOPESHOT

- Yellow – PSN
- Green – Scan Enable
- Purple – CLK
- Pink – Trigger
MATLAB POST PROCESSING

• To be able to accurately tabulate the VDD_Sense droop vs. respective clock domain frequency, a Matlab script is used.
  • Inputs to this script are the stored “.bin” files from the scope
  • Outputs from Matlab script are:
# SNAPSHOT OF DATASET

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Global Switch Factor %</th>
<th>Process</th>
<th>Voltage</th>
<th>Temp</th>
<th>Freq (MHz)</th>
<th>IP Name</th>
<th>Product</th>
<th>LPC</th>
<th>Droop (mV)</th>
<th>Granular Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>60.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>45.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>65.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>36.10%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>36.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>33.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>50.00%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2998</td>
<td>29.87%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>2999</td>
<td>47.84%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>58.92%</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>
DEPLOYMENT

Goal

- Optimize low power DFT architecture
- Generate reliable test patterns

PSN analysis is repeated

- at various milestones of the chip design cycle and finalized close to tape-out.
- until there are no violations for any of the test patterns.
PROPOSED APPROACH

- Design Flow
- Feature Engineering
- Deep Learning Models
- Classification and Regression
FEATURE ENGINEERING

IP-level (Global)
- GSF
- PVT
- PLL frequency $f$
- LP_Value
- Type

SoC sub-block-level (Local)
- LSF
- Instance_Count
- Sense_Distance
- Area
EXAMPLE: FEATURE EXTRACTION

- on-chip measurement point location
- sense point neighborhood-level graph
- global and local feature vectors

Sub-Block-Level layout of an SoC

Global Vector:

<table>
<thead>
<tr>
<th>GSF (%)</th>
<th>μ</th>
<th>V (V)</th>
<th>T (°C)</th>
<th>f (GHz)</th>
<th>Type</th>
<th>LSI_Value (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>2</td>
<td>1.0</td>
<td>20</td>
<td>1.0</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>

Local Vector:

<table>
<thead>
<tr>
<th>LSF (×1000)</th>
<th>Instance_Count (×1000)</th>
<th>Sense_Distance (mm)</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>3.0</td>
</tr>
<tr>
<td>6</td>
<td>7.0</td>
<td>6.8</td>
<td>7.5</td>
</tr>
<tr>
<td>7.5</td>
<td>9.3</td>
<td>9.0</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>14</td>
<td>2.5</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>6.5</td>
<td>6.5</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
PROPOSED APPROACH

• Design Flow
• Feature Engineering
• Deep Learning Models
• Classification and Regression
DEEP LEARNING MODELS

Fully Connected (FC) model
- basic type of neural network and is used in most of the models.
- Flattened FC model
- Hybrid FC model

Natural Language Processing-based (NLP) model
- NLP is traditionally used to analyze human language data.
- we apply the concept of the averaging layer to our IR drop prediction problem.
- Model is independent of the number of sub-blocks in a chip.
FLATTENED FC MODEL

All the input features are applied simultaneously to the first layer.
HYBRID FC MODEL

Input features are divided into different groups, each applied to a different layer.

\[ X_1 = \{ \text{GSF}, \text{PVT, f}, \text{LPC}_\text{Value}, \text{Type} \} \]
\[ X_2 = \{ \text{LSF, Instance_Count} \} \]
\[ X_3 = \{ \text{Sense_Distance, Area} \} \]
NLP MODEL

- Local features of each sub-block form an individual bag of numbers.
- Filtered Average (FA): 1) filters out non-toggled sub-blocks, 2) calculates the average.
PROPOSED APPROACH

• Design Flow
• Feature Engineering
• Deep Learning Models
• Classification and Regression
Classification and Regression

- Classification models predict a discrete value (or a bin).
- Regression models predict the absolute value.
- Optimization:
  - Cost Function:
    \[ J = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i) + \phi(w) \]
  - Loss Function: \( L(y_i, \hat{y}_i) \)
    - Classification: \(-y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\)
    - Regression: \(\sqrt{\frac{1}{k} \sum_{i=1}^{k} (y_i - \hat{y}_i)^2}\)

Additional techniques:
- Input Normalization
- Adam optimizer
- Learning rate decay
- L2 regularization
RESULTS

Benchmark Information - 16nm GPU chips: Volta-IP1 and Xavier-IP2

➢ Local features are wrapped with zero-padding (only for FC)
➢ Approximately 90% of the samples for training and validation
➢ Approximately 10% of the samples for inference.

Models were developed in Python using TensorFlow and NumPy libraries. Models were run on a cloud-based system with 2 CPUs, 2 GPUs and 32GB memory.

<table>
<thead>
<tr>
<th>GPU</th>
<th>No. of Features</th>
<th>No. of Train Samples</th>
<th>No. Inference Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volta-IP1</td>
<td>323</td>
<td>16500</td>
<td>1500</td>
</tr>
<tr>
<td>Xavier-IP2</td>
<td>239</td>
<td>2500</td>
<td>500</td>
</tr>
</tbody>
</table>
# RESULTS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model-Architecture</th>
<th>Train Accuracy (%)</th>
<th>Inference Accuracy (%)</th>
<th>Train Time (minutes)</th>
<th>MAE (mV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volta-IP1 + Xavier-IP2</td>
<td>Classification-Flushed FC</td>
<td>94.5</td>
<td>94.5</td>
<td>10</td>
<td>7.30</td>
</tr>
<tr>
<td></td>
<td>Classification-Hybrid FC</td>
<td>96.0</td>
<td>96.0</td>
<td>3</td>
<td>6.90</td>
</tr>
<tr>
<td></td>
<td>Classification-NLP</td>
<td>92.6</td>
<td>92.6</td>
<td>80</td>
<td>7.46</td>
</tr>
<tr>
<td></td>
<td>Regression-Flushed FC</td>
<td>98.0</td>
<td>93.0</td>
<td>9</td>
<td>7.79</td>
</tr>
<tr>
<td></td>
<td>Regression-Hybrid FC</td>
<td>98.0</td>
<td>96.0</td>
<td>3</td>
<td>7.25</td>
</tr>
<tr>
<td></td>
<td>Regression-NLP</td>
<td>95.0</td>
<td>95.0</td>
<td>90</td>
<td>7.28</td>
</tr>
</tbody>
</table>

**Average run-time or prediction time**

➢ For a 500-pattern set

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Silicon Simulation</td>
<td>416 days</td>
</tr>
<tr>
<td>Post-Silicon Validation</td>
<td>84 mins</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.33 secs</td>
</tr>
</tbody>
</table>
RESULTS

Correlation between the predicted and the silicon-measured $V_{\text{droop}}$

Classification

Regression
FUTURE WORK

- Train and apply DL for in-field test vectors noise estimation
- Shift Noise prediction
- Additional physical parameters
- Other architectures
RTL-Level Power Estimation Using Machine Learning

Mark Ren, Yan Zhang, Ben Keller, Bruce Khailany

Yuan Zhou, Zhiru Zhang

NVIDIA®

Cornell University®
MOTIVATION

- Power modeling is either slow or inaccurate.
- Get power with accurate power estimation using simulation traces at early design stages?

**Behavioral Level**

- C++
- SystemC

Very fast: > 10k cycles/s
(Source: [Ahuja ISQED’09] [Shao ISCA’14])

Only average power
Not that accurate

**RTL Level**

- RTL

Slower: 1k-10k cycles/s
(Source: [Yang ASP-DAC’15][PowerArtist])

Not-so-great accuracy
Some still only model average power

**Gate Level**

- Gate-level Netlist

Slowest: 10-100 cycles/s
(Source: [VCS, Primetime PTPX])

Cycle-level power trace
Very accurate

---


---

34
OPPORTUNITY: ML FOR EDA

- Emerging field using Machine Learning for Electronic Design Automation (EDA) tasks
- Utilize GPU proficiency in ML tasks + find a way to map EDA applications to fit ML
- Use machine learning / deep learning techniques to accurately estimate power at higher design abstraction level (RTL)
  - Shorter turn-around time, faster power validation, covers a diverse range of different workloads
PROPOSED SOLUTION: ML-BASED POWER ESTIMATION WORKFLOW

Gather Training Data

Simulation
  Simulation Results
  Power Analysis
  Power Results

Once

Feature Engineering
Model Training

Simulation Results

Power Results

Feature Construction

ML Model Training

Trained Power Model

New Test Cases

Simulation
  New Simulation Results
  Feature Construction
  Trained Power Model
  ML Model Inference
  New Power Results

Model Application

“Free”
POWER ESTIMATION: CIRCUIT PERSPECTIVE

- Our models are essentially learning the switching capacitance associated with certain register switching activities
- Figuring out which caps switch and by how much is inhumanely complex and non-linear
- → Perfect for machine learning!
- Example:

\[
P = CV^2f
\]
MODEL SELECTION

- **Traditional ML:** linear model, XGBoost
  - With principal component analysis (PCA) applied for overfitting avoidance
  - Pros: smaller model, faster training
  - Cons: Hard to capture non-linearities

- **DL:** convolutional neural net (CNN), multi-layer perceptron (MLP)
  - Pros: good for all sorts of non-linear models, good scalability
  - Cons: large model, longer training times, scalable but at a large startup cost (lots of parameters/nodes)

\[
P = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \cdots + a_n x_n
\]

\[
\begin{pmatrix}
p_1 \\
p_2 \\
\vdots \\
p_m
\end{pmatrix} =
\begin{pmatrix}
a_1 \\
a_2 \\
\vdots \\
a_n
\end{pmatrix} \cdot
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_m
\end{pmatrix}
\]

Linear regression model

**FEATURE CONSTRUCTION**

- What information to use?
  - Register 0/1 state as inputs into model
- How to encode? CNNs work best when features have spatial relationship for their inputs
  - Default (naïve) encoding: random placement of register traces in CNN input
  - Graph-partition based: treat register relations as a graph, then partition to determine input placement
  - Node-embedding based: Use `node2vec` to convert graph nodes into embeddings (Source: [Grover SIGKDD’16])

Default encoding

Graph-partition

Node-embedding

---

Normalized Root Mean Square Error (NRMSE)

\[
\text{NRMSE} = \frac{\text{RMSE}}{\bar{y}}
\]

Cycle-by-cycle basis

Directly look at the power traces to see how good it fits

- Good for catching outliers
- Cycle-by-cycle basis
EXPERIMENT SETUP

- ML training and inference infrastructure:
  - NVIDIA 1080Ti GPU
  - Software packages: network,metis, node2vec, Python 3.5, Keras 2.1.6, scikit-learn, xgboost 0.72.1

- Ground truth and comparison baseline gate level power analysis infrastructure
  - Intel Xeon CPU server, 64GB RAM
RESULTS

- Good accuracy
  - <5% average power estimation for all test cases
  - CNNs outperform linear models for bigger designs
  - Accuracy outperforms commercial tool

- 50X speedup against gate simulation + power analysis

- Cycle-by-cycle traces show better accuracy for CNNs compared to linear models

CONCLUSIONS

- We can get both good accuracy and high speedup with ML-based power estimation
- Achieves ~50X speedup over baseline with <5% error
- A good example of using ML for EDA purposes
- GPUs greatly benefit training/inference time in ML for EDA