Machine Learning-Enabled High-Frequency Low-Power Digital Design Implementation At Advanced Process Nodes

ISPD 2021

Siddhartha Nath and Vishal Khandelwal
Digital Design Group
Synopsys Inc.
Outline

• Background and Motivation
• Previous Work
• Problem Formulation
• Methodology
• Experimental Results
• Summary and Future Work
Path: ff1/CK \rightarrow ff1/Q \rightarrow u1/A \rightarrow u1/Z \rightarrow u2/A \rightarrow u2/ZN \rightarrow u3/B \rightarrow u3/ZN \rightarrow ff2/D

Merging of worst transitions at merge nodes \rightarrow \text{pessimism}
PBA STA

Path: ff1/CK → ff1/Q → u1/A → u1/Z → u2/A → u2/ZN → u3/B → u3/ZN → ff2/D

Actual transition propagation through nodes
PBA in Digital Implementation: Opportunities

• Achieve signoff-quality timing and power for best-on-class power performance and area (PPA)
• Large gap between GBA and PBA at advanced nodes
  – 78ps delta is ~4-5 stages of logic at 7nm
• Rich optimization moves as compared to ECO signoff
  – Advance logic restructuring
  – Layer promotion/NDR with re-route and rebuffering
  – Flexible clock and data optimization
  – Legalization- and eco-routing-aware
• Prevents over-design (area / power loss)
PBA in Digital Implementation: Challenges

• “True” PBA analysis requires exponential path tracing

• Massive increase in memory and runtime
  – Overhead can be ~50% of GBA runtime

• “True” PBA exhaustive makes TAT infeasible on large-scale designs with 1-10M instances
  – PBA path is an alternative (retimes the worst GBA path using PBA analysis)
  – Practical considerations limit design companies to analyze top-K paths (e.g., K = 10000)

• Incremental STA using PBA in implementation flows is tricky
  – Fine balance of runtime vs. correct analyses
Previous Works

• PBA runtime improvement heuristics
  – Bai et al. propose PBA path, top-K path analyses
  – Gupta et al. propose IPBA, an infinite-depth parallel traversal of timing graph using BFS and speedup stage-wise computation
  – Shyamsukha et al. perform selective PBA analyses based on statistical and non-statistical path characteristics
  – Wrixon et al. propose a thresholding mechanism while merging timing values

• ML-based heuristics
  – Huang and Wong create a task graph and use MapReduce to massive scale PBA analyses
  – Kahng et al. predict PBA timing using GBA features using a bigram approach to capture path timing features
    – Closest to our work
    – Large runtime for deployment in implementation flows
Our Key Contributions

• Our is the **first** application of ML paradigm to **optimize** post-routed designs using predicted PBA slack using a **ML-augmented** GBA flow

• We propose **on-the-fly training** methodology and **insights** on choice of features and model algorithm for our prediction task

• We apply **engineering strategies** on model guidance for delay, area and power recovery to reduce GBA flow over-design

• We **integrate** our methodology in a **commercial EDA tool** and present non-trivial PPA gains on **real industrial designs** across 5nm – 16nm

• Our experimental results have a moderate runtime overhead ~3% (vs. GBA) for up to 11.7% improvement in leakage power and 1.16% in total power
  – Baseline GBA runs state-of-the-art power flows on industrial designs where every 1% total power matters
Problem Formulation

- Design ML-based predictor of PBA slack given GBA features
- Predict MCMM PBA slack across all critical paths
- Separate models for delay and recovery
- Integrate models in a commercial post-route optimization flow
- Adjust GBA timing on critical paths based on ML model predictions
On-The-Fly Training

• Key innovation in our work

• Eliminate need for two-pass training and inference flow requirements for supervised modeling
  – Causes deployment challenges at customers due to data management

• Develop a one-pass flow
  – Model trained when large changes to netlist occurs that shifts PBA labels
  – Model inferred / refreshed at strategic points in the optimization flow

• Requires a light-weight and accurate training methodology
  – Low overhead of feature extraction
  – Fast and parallel algorithms for training
  – Runtime overhead of above <5% of a baseline GBA flow
  – Cost function that accurately models design PPA
# Our Features

<table>
<thead>
<tr>
<th>Category</th>
<th>#</th>
<th>Features Examples</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Context</td>
<td>4</td>
<td>SP(x, y); EP(x, y); WL(X); WL (Y)</td>
<td>Discriminates path layouts</td>
</tr>
<tr>
<td>Logical Context</td>
<td>4</td>
<td>#stages in critical path; max fanout; avg fanout; avg drive resistance</td>
<td>Indicates extent of GBA-PBA divergence</td>
</tr>
<tr>
<td>Timing Context</td>
<td>9</td>
<td>SP cap, slew; EP cap, slew; cell and net delays; ratio of net to total delay; avg slew, cap</td>
<td>Discriminates between path timing characteristics</td>
</tr>
<tr>
<td>Physical Constraints</td>
<td>2</td>
<td>#dont_touch,; #size_only</td>
<td>Indicates extent of GBA-PBA divergence</td>
</tr>
<tr>
<td>Timing Constraints</td>
<td>4</td>
<td>#maxtran violators on critical paths; EP arrival, required; MCMM scenario</td>
<td>Discriminates types of paths and indicates the extent of GBA-PBA divergence</td>
</tr>
</tbody>
</table>
Feature Importances

- EP arr
- EP req
- SP slew
- SP cap

Variation in ensemble

Fitting residuals
Machine Learning Algorithm

- Gradient-boosted Random Forest regressor
  - Fast to train and accurate
  - Easier to debug outliers as compared to DNNs or other complex models
  - Intuitive hyperparameters and easier to tune as compared to those of DNNs
- Algorithm bootstrapped using 50-70% of training dataset
- At each node, randomly sample 50-70% of features
- Select feature with the largest information gain
- Prune trees that violate max depth or min #child nodes criteria
- Objective is to minimize RMSE between GBA and PBA slack → accurate indicator of design TNS
ML-Guided Post-Route Optimization

**Baseline GBA Flow**

1. Routed Netlist
2. Logical DRC, SI, Setup and Hold Fixing using Datapath and Clock Optimization Techniques
3. Area and Power Recovery using Datapath and Clock Optimization Techniques
4. Legalize and ECO route

**ML-Augmented GBA Flow**

1. Routed Netlist
2. Logical DRC, SI, Setup and Hold Fixing using Datapath and Clock Optimization Techniques
3. Train
4. Infer
5. PBA Delay ML Model
6. Train
7. Infer
8. PBA Recovery ML Model
9. Area and Power Recovery using Datapath and Clock Optimization Techniques
10. Legalize and ECO route
Experimental Setup

- Modeling implemented in C++ and Python; 80%-20% train-test split
- Model trained using PBA path for fast turnaround of experiments
- Integrate in post-route optimization flow of a commercial EDA tool
- Experiments run on 21 industrial designs and run on Intel Xeon 16-core 2.6GHz machines

<table>
<thead>
<tr>
<th>Node</th>
<th>Designs</th>
<th>#</th>
<th>#Instances Range</th>
<th>Power Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>16nm</td>
<td>D16A – D16D</td>
<td>4</td>
<td>161K – 2.4M</td>
<td>Total power</td>
</tr>
<tr>
<td>12nm</td>
<td>D12A – D12C</td>
<td>3</td>
<td>130K – 855K</td>
<td>Total power</td>
</tr>
<tr>
<td>7nm</td>
<td>D7A – D7E, D7F – D7I</td>
<td>5</td>
<td>331K – 1.5M</td>
<td>Total power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>272K – 492K</td>
<td>Leakage power</td>
</tr>
<tr>
<td>5nm</td>
<td>D5A – D5D, D5E</td>
<td>4</td>
<td>94K – 359K</td>
<td>Total power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>67K</td>
<td>Leakage power</td>
</tr>
</tbody>
</table>
ML Model Accuracy: Delay

RMSE Ratio

~30X

D16B and D5C have small GBA – PBA gap

RMSE (GBA – true PBA)

Ratio = ---------------------------------------------

RMSE (Pred. PBA – true PBA)
ML Model Accuracy: Recovery

RMSE Ratio = \frac{\text{RMSE (GBA – true PBA)}}{\text{RMSE (Pred. PBA – true PBA)}}

D5C have small GBA – PBA gap
Impact on Post-Route Optimization

• PBA reduces delay pessimism
  – Fewer #instances selected for optimization during delay fixing as compared to GBA
  – Larger #instances selected for optimization during area, power recovery as compared to GBA

<table>
<thead>
<tr>
<th>Design</th>
<th>Attempted</th>
<th>Accepted</th>
<th>Runtime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GBA</td>
<td>ML</td>
<td>GBA</td>
</tr>
<tr>
<td>Delay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D16D</td>
<td>53K</td>
<td>47K</td>
<td>27K</td>
</tr>
<tr>
<td>D12A</td>
<td>294K</td>
<td>247K</td>
<td>181K</td>
</tr>
<tr>
<td>D7B</td>
<td>180K</td>
<td>168K</td>
<td>8.1K</td>
</tr>
<tr>
<td>D7F</td>
<td>58K</td>
<td>49K</td>
<td>42K</td>
</tr>
<tr>
<td>D5C</td>
<td>12K</td>
<td>8.3K</td>
<td>6.1K</td>
</tr>
<tr>
<td>Recovery</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D16D</td>
<td>1.75M</td>
<td>1.83M</td>
<td>299K</td>
</tr>
<tr>
<td>D16D</td>
<td>1.65M</td>
<td>1.66M</td>
<td>194K</td>
</tr>
<tr>
<td>D12A</td>
<td>49</td>
<td>1.93M</td>
<td>10</td>
</tr>
<tr>
<td>D7B</td>
<td>3.74M</td>
<td>3.74M</td>
<td>364K</td>
</tr>
<tr>
<td>D7F</td>
<td>694K</td>
<td>793K</td>
<td>206K</td>
</tr>
<tr>
<td>D5C</td>
<td>125K</td>
<td>129K</td>
<td>40K</td>
</tr>
</tbody>
</table>

Small GBA-PBA gap ➔ insignificant impact on D5C

Less runtime in delay optimization

More runtime in area, power optimization
### Post-Route Optimization PPA

| Design | GBA | | | ML | 
|--------|-----|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
|        | TNS (ns) | Area (µm²) | Power (mW) | TNS (ns) | Area (µm²) | Power (mW) | TNS (ns) | Area (µm²) | Power (mW) |
| D16C   | -14.3 | 813K | 3567 | -19.6 | 812K | 3526 | (0.23%) | (-0.02%) | (-1.16%) |
| D16D   | -2.64 | 701K | 762 | -13.1 | 700K | 3626 | (0.16%) | (-0.04%) | (-0.87%) |
| D12Aγ  | -99.4 | 286K | 30.4 | -81.3 | 283K | 28.2 | (-0.41%) | (-0.92%) | (-7.19%) |
| D7B    | -4.16 | 291K | 1145 | -6.83 | 291K | 1142 | (0.20%) | (-0.07%) | (-0.23%) |
| D7Fγ   | -0.25 | 74K | 9.97 | -0.55 | 72K | 8.80 | (0.13%) | (-0.86%) | (-11.7%) |
| D5C    | -1.22 | 11K | 201 | -1.14 | 10K | 202 | (-0.08%) | (-0.39%) | (0.25%) |

- TNS is normalized to path delay for unbiased comparisons
- ML-augmented GBA flow improves area and power as expected
- Sources of TNS degradation
  - Inaccuracies due to model reuse
  - Model limitation as PBA path is used for training

Y → Leakage-only design
## Statistics of Overall PPA, Runtime

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Area</th>
<th>Leakage</th>
<th>Dynamic</th>
<th>DRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (μ)</td>
<td>-0.12%</td>
<td>-2.30%</td>
<td>-0.14%</td>
<td>-4.76%</td>
</tr>
<tr>
<td>Avg μ/σ</td>
<td>-2.94</td>
<td>-3.17</td>
<td>-2.95</td>
<td>-0.37</td>
</tr>
<tr>
<td>%Win</td>
<td>81%</td>
<td>95%</td>
<td>79%</td>
<td>10%</td>
</tr>
<tr>
<td>%Neutral</td>
<td>0%</td>
<td>0%</td>
<td>21%</td>
<td>80%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistics</th>
<th>WNS</th>
<th>TNS</th>
<th>THV</th>
<th>#MaxTran Violations</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (μ)</td>
<td>+0.02%</td>
<td>+0.07%</td>
<td>+6.27%</td>
<td>-3.12%</td>
<td>+3.37%</td>
</tr>
<tr>
<td>Avg μ/σ</td>
<td>+0.11</td>
<td>+1.88</td>
<td>+0.74</td>
<td>-0.88</td>
<td>+1.78</td>
</tr>
<tr>
<td>%Win</td>
<td>48%</td>
<td>81%</td>
<td>43%</td>
<td>76%</td>
<td>43%</td>
</tr>
<tr>
<td>%Neutral</td>
<td>38%</td>
<td>0%</td>
<td>10%</td>
<td>10%</td>
<td>19%</td>
</tr>
</tbody>
</table>
Summary and Future Work

• PPA optimization at advanced process nodes is challenging in traditional GBA-based implementation flows
• We demonstrate a fast and accurate ML-enabled methodology to learn PBA timing and apply in a ML-augmented post-route optimization flow
• Our detailed insights on feature engineering, model selection hopefully helps other ML projects in digital implementation
• We achieve up to 11.7% leakage and 1.16% total power reduction across multiple industrial designs from top semiconductor companies spanning from 16nm down to 5nm
• Our ongoing works
  – Improve timing outliers
  – Develop ML-augmented pre-route GBA flows
Thank You