

TAP-MAC: Low-Power Optimization of Multipliers and Multiply-Accumulators via Topology-Aware Power Modeling

Dongsheng Zuo, Yang Luo, Jiadong Zhu, Yuzhe Ma*
The Hong Kong University of Science and Technology (Guangzhou)

Abstract

Multipliers and multiply-accumulators (MACs) are fundamental arithmetic building blocks in modern digital systems, yet their optimization has largely focused on area and timing rather than power. This gap has become particularly acute in the large-model era, where power dominates the cost of AI computation. Unlike timing and area, power is inherently complex to model: the intricate internal topology induces substantial glitch and reconvergent fanout effects and yields an enormous combinatorial design space, making both modeling and optimization challenging. To address these challenges, we propose TAP-MAC, a topology-aware low-power optimization framework for multipliers and MACs. TAP-MAC builds a high-fidelity, differentiable power surrogate model capturing both functional switching and glitch power, and couples differentiable relaxation for design-space contraction with MCTS-based discrete refinement via a GPU-accelerated evaluator. Experiments show up to 15% power reduction over a commercial IP and consistent gains over strong baselines, with further reductions in ISP and systolic-array deployments under real workloads.

ACM Reference Format:

Dongsheng Zuo, Yang Luo, Jiadong Zhu, Yuzhe Ma*. 2026. TAP-MAC: Low-Power Optimization of Multipliers and Multiply-Accumulators via Topology-Aware Power Modeling. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26)*, November 08–12, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3831252.3834049>

1 Introduction

Multipliers and multiply-accumulators (MACs) are fundamental arithmetic building blocks of modern digital systems, spanning digital signal processing (DSP) pipelines and AI accelerators, and their energy efficiency increasingly constrains system-level performance. This pressure is particularly acute in the large-model era, where power dominates the cost of AI computation. Automated low-power optimization of multipliers and MACs is thus a central concern for modern compute hardware.

A typical integer multiplier comprises three components: a partial-product generator (PPG), a compressor tree (CT), and a carry-propagate adder (CPA). As shown in Figure 1(b), the CT accounts for 77% of total multiplier power and typically dominates both area and critical-path delay. The CT assignment specifies how compressors are allocated across stages and columns to reduce the partial products. It has been

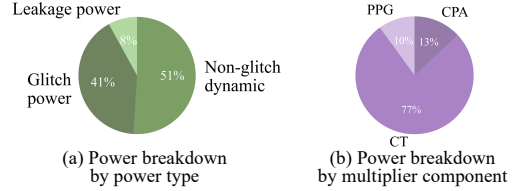


Figure 1: Power concentration in a 16-bit multiplier.

extensively studied through a range of hand-crafted, heuristic, ILP-based, and learning-based methods [1–5]. Yet the internal rewiring of the CT under a fixed assignment remains a vast and largely underexplored design space, with substantial impact on both timing and power.

Recent work extends optimization to the internal interconnect rewiring of the CT: UFO-MAC [6] uses ILP and DOMAC [7] uses differentiable optimization, but both focus on timing. Differentiable optimization has more broadly emerged as an effective EDA paradigm for large coupled design spaces, leveraging first-order information to jointly steer many design variables, as exemplified by analytical placement [8]. By contrast, existing multiplier optimization has focused largely on timing and area, with only a few prior works targeting power. These works [9, 10] rely on coarse power models and simple search strategies such as pruning-and-enumeration or greedy methods. High-fidelity power modeling and effective power optimization for multipliers therefore remain open challenges.

Unlike delay and area, which admit linear analytical models, multiplier power is substantially harder to model. Different rewirings reshape signal pairing, arrival-time alignment, and local loading, inducing distinct patterns of switching and glitch activity throughout the CT. In particular, as shown in Figure 1(a), glitch power alone accounts for 41% of total multiplier power. The effect is not confined to the CT itself: although the CPA is structurally fixed, the altered switching and glitch activity at the CT outputs propagate into the CPA and modulate its power as well. Generic vectorless power analysis [11] cannot adequately capture such glitch behavior or reconvergent fanout across the CT, leaving high-fidelity yet optimization-friendly power modeling an open challenge. The CT rewiring space itself grows combinatorially to $\sim 10^{27}$ for a 16-bit multiplier, while evaluating each candidate requires minute-scale gate-level simulation and timing-based power analysis. Together with the intrinsically discrete nature of rewiring decisions, this combinatorial scale and evaluation cost pose an equally open optimization challenge.

Motivated by these observations, we propose TAP-MAC, a topology-aware low-power optimization framework for multipliers and MACs. A key enabler is a differentiable multiplier power surrogate model: TAP-MAC first constructs one that captures both functional switching and glitch power as the internal topology changes. Rather than

*Corresponding author.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
ICCAD '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
<https://doi.org/10.1145/3831252.3834049>

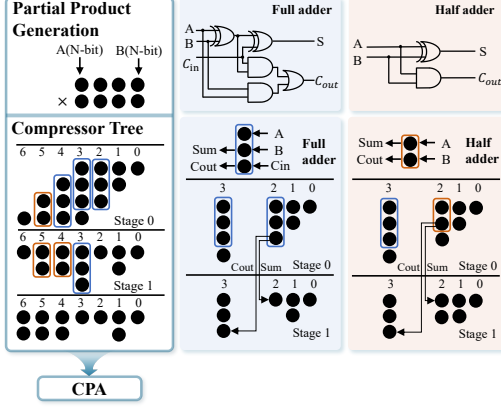


Figure 2: Multiplier datapath and stage-column semantics of FA/HA cells.

directly replacing discrete search, this surrogate model drives a differentiable relaxation that contracts the combinatorial design space to a small set of promising local rewiring choices. TAP-MAC then performs MCTS-based discrete refinement over the pruned space, with a GPU-accelerated evaluator enabling large-scale parallel power evaluation of candidate designs. Because MAC units share the same CT-based reduction structure, the framework extends naturally from multipliers to MACs.

In summary, the contributions of this work are as follows:

- We propose TAP-MAC, a topology-aware low-power optimization framework for multipliers and MACs.
- We develop a differentiable multiplier power surrogate model that captures both functional switching and glitch power.
- We present a two-phase optimization flow that combines differentiable relaxation for search-space contraction with MCTS-based discrete refinement and GPU-accelerated evaluation.
- Experimental results demonstrate up to 15% power reduction over a commercial IP and consistent gains over strong baselines, with further reductions when deployed in an image signal processor and a systolic array under real workloads.

2 Preliminaries

2.1 Multiplier Datapath

As illustrated in Figure 2, a typical multiplier datapath consists of a partial-product generator (PPG), a compressor tree (CT), and a carry-propagate adder (CPA). Given two N -bit operands, the PPG produces a shifted partial-product matrix, which the CT reduces to two rows that the CPA sums into the final product.

The CT is built from full adders (FAs) and half adders (HAs). An FA at stage i , column j takes three input bits and produces a sum (XOR3) at $(i+1, j)$ and a carry (MAJ3) at $(i+1, j+1)$; an HA is analogous with two inputs. This stage-column organization defines the reduction structure and signal flow inside the compressor tree.

For a MAC unit, the accumulated addend can be incorporated into the partial-product matrix and reduced by the same CT-style structure. Therefore, the optimization methodology developed for multipliers extends naturally to MACs.

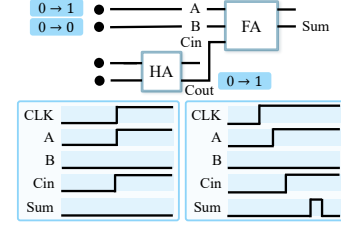


Figure 3: Delay-induced glitch in a compressor tree: misaligned input arrivals create extra transient transitions even when the final logic value is unchanged.

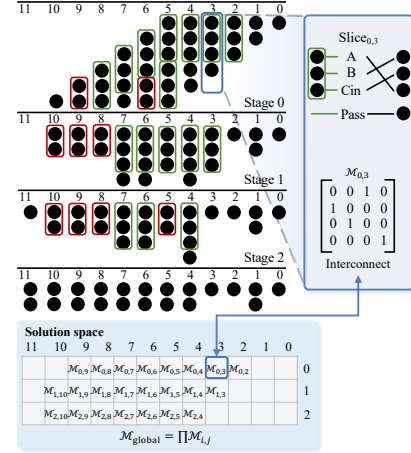


Figure 4: CT rewiring space: each active stage-column slice is a local permutation matrix; the global space is their Cartesian product.

2.2 Dynamic Power Decomposition

Dynamic power in CMOS multiplier datapaths arises from capacitive switching and can be written as

$$P_{\text{dyn}} = \frac{1}{2} V_{DD}^2 f_{\text{clk}} \sum_i C_i \alpha(x_i), \quad (1)$$

where V_{DD} is the supply voltage, f_{clk} is the clock frequency, C_i is the effective capacitance at node x_i , and $\alpha(x_i)$ is the average number of logic transitions per clock cycle at that node.

For each signal x_i , we decompose its transition activity into a functional component and a glitch component,

$$\alpha(x_i) = \alpha_{\text{func}}(x_i) + \alpha_{\text{glitch}}(x_i), \quad (2)$$

yielding a power decomposition

$$P_{\text{dyn}} = P_{\text{func}} + P_{\text{glitch}}. \quad (3)$$

Here, $\alpha_{\text{func}}(x_i)$ captures the logic-dependent transitions determined by the input statistics, whereas $\alpha_{\text{glitch}}(x_i)$ captures extra delay-dependent transitions caused by arrival-time imbalance along different timing paths. In multipliers and MACs, these glitches may be generated locally at a gate and then propagated along the CT reduction paths and the CPA carry chain.

Why rewiring affects power. The multiplier's rich internal reconvergent structure makes both P_{func} and P_{glitch} sensitive to rewiring

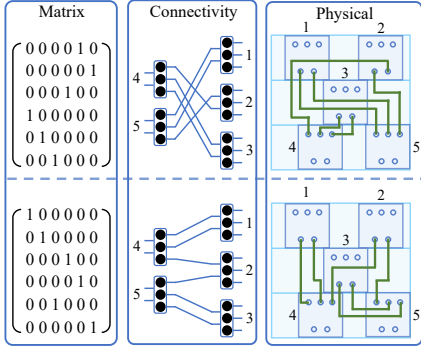


Figure 5: Equivalent views of CT rewiring: matrix representation, logical source-to-sink connectivity, and ECO-style post-placement rewiring.

even under a fixed compressor assignment, since different source-to-sink choices alter switching patterns, wire loads, and arrival-time alignment, which in turn modulates glitch generation and propagation.

2.3 Compressor-Tree Rewiring Space

Given a CT with a predetermined stage-column organization and compressor assignment, we optimize only its internal rewiring. We assume an ECO-style post-placement setting: given an initial placement of compressors, only the interconnect is rewired. Let $\mathcal{S}$ denote the set of active stage-column locations. As illustrated in Figure 4, each $(i, j) \in \mathcal{S}$ defines a slice, where incoming source signals are assigned to sink positions, including compressor input pins and pass-through destinations. Figure 5 shows the equivalent matrix, logical source-to-sink, and ECO-style post-placement rewiring views of one such local assignment.

For a slice (i, j) with $s_{i,j}$ source signals and $s_{i,j}$ sink positions, a legal local rewiring is represented by a permutation matrix $M_{i,j} \in \{0, 1\}^{s_{i,j} \times s_{i,j}}$, where each row selects the sink of one source and each column is occupied by exactly one source, so $|M_{i,j}| = s_{i,j}!$. A complete CT rewiring is the collection $M = \{M_{i,j}\}_{(i,j) \in \mathcal{S}}$, and the global design space is

$$\mathcal{M}_{\text{global}} = \prod_{(i,j) \in \mathcal{S}} \mathcal{M}_{i,j}. \quad (4)$$

For the 16-bit case considered in this work, $|\mathcal{M}_{\text{global}}|$ is on the order of 10^{278} , making direct enumeration infeasible.

Although the decision variables lie in the CT, rewiring also affects the CT outputs and hence the CPA power. We therefore formulate the optimization objective at the full-multiplier level as

$$\min_{M \in \mathcal{M}_{\text{global}}} P_{\text{mul}}(M) \quad \text{s.t.} \quad D_{\text{mul}}(M) \leq T_{\text{clk}}. \quad (5)$$

3 Topology-Aware Differentiable Power Surrogate Modeling

We construct a differentiable, topology-aware power surrogate model built on the activity decomposition of Equation (2). Throughout this section, each signal x is characterized by its signal probability p_x and latest arrival time AT_x under the current rewiring M , both propagated

topologically through the CT. Following the timing style of UFO-MAC [6] and DOMAC [7], AT_x combines the source arrival, the post-placement wire delay estimated from the Manhattan source-to-sink distance, and the cell arc delay, with hard max/min replaced by smooth counterparts so the propagation remains differentiable with respect to M . Aggregating the arrival times at the CPA outputs yields a differentiable full-multiplier delay estimate $\widehat{D}_{\text{mul}}(M)$, which is used both by the glitch skew in Section 3.3 and as the timing constraint in Section 4. We first propagate marginal probabilities, then model the functional and glitch components of transition activity, and finally incorporate reconvergent-fanout effects and a lightweight calibration layer.

3.1 Signal Probability Propagation

We first propagate signal probabilities over the given compressor-tree structure. For any Boolean signal x , define its signal probability as

$$p_x \triangleq \Pr\{x = 1\}. \quad (6)$$

Under a given rewiring, each compressor input pin inherits the signal probability of its fanin signal. Since the FA is the dominant compressor in the CT, we use it as the running example. For an FA cell with input-pin signal probabilities p_A , p_B , and $p_{C_{\text{in}}}$, the signal probabilities of its SUM and CARRY outputs are determined by the Boolean functions of XOR3 and MAJ3, respectively:

$$p_S = p_A + p_B + p_{C_{\text{in}}} - 2(p_A p_B + p_A p_{C_{\text{in}}} + p_B p_{C_{\text{in}}}) + 4p_A p_B p_{C_{\text{in}}}, \quad (7)$$

$$p_{C_{\text{out}}} = p_A p_B + p_A p_{C_{\text{in}}} + p_B p_{C_{\text{in}}} - 2p_A p_B p_{C_{\text{in}}}. \quad (8)$$

The HA case can be derived in the same way. By traversing the CT in topological order and recursively applying the corresponding Boolean update at each compressor, we obtain the signal probabilities of all internal nodes under the current rewiring.

3.2 Functional Activity Propagation

We next model the functional component in Equation (2). Assuming temporally independent cycles, the functional activity of signal x is computed from its signal probability as

$$\alpha_{\text{func}}(x) = 2p_x(1 - p_x). \quad (9)$$

For the FA outputs, combining Equation (9) with Equations (7) and (8) gives

$$\alpha_{S,\text{func}} = 2p_S(1 - p_S), \quad (10)$$

$$\alpha_{C_{\text{out}},\text{func}} = 2p_{C_{\text{out}}}(1 - p_{C_{\text{out}}}). \quad (11)$$

Thus, rewiring changes the functional activity by changing the signal probabilities at the FA input pins, and hence the output probabilities of XOR3 and MAJ3. Propagating Equations (10) and (11) over the CT topology yields the functional activity of all internal signals under the current rewiring.

3.3 Glitch Activity Branch

We next model the glitch component in Equation (2). For a compressor output signal x , we decompose its glitch activity as

$$\alpha_{\text{glitch}}(x) = \alpha_{\text{gen}}(x) + \alpha_{\text{prop}}(x), \quad (12)$$

where $\alpha_{\text{gen}}(x)$ denotes the locally generated glitch caused by arrival-time imbalance among the input pins of the current compressor, and $\alpha_{\text{prop}}(x)$ denotes the glitch propagated from upstream compressors. Extra transitions that occur before an input signal settles to its latest arrival time are not reinterpreted at the current compressor and are instead absorbed into the propagated-glitch activity carried by that input signal.

Locally generated glitch. Let AT_A , AT_B , and $AT_{C_{\text{in}}}$ denote the latest arrival times at the three input pins of the current FA. We define the input-arrival skew as

$$\text{skew} = \max(AT_A, AT_B, AT_{C_{\text{in}}}) - \min(AT_A, AT_B, AT_{C_{\text{in}}}), \quad (13)$$

A locally generated glitch can survive only when the input-arrival skew exceeds the timing-arc threshold of the corresponding output. We therefore define

$$H(\text{skew}, T_{\text{arc}}) = \mathbb{I}(\text{skew} > T_{\text{arc}}), \quad (14)$$

where $\mathbb{I}(\cdot)$ is the indicator function. In the differentiable model, the hard max/min and the hard indicator in Equation (14) are replaced by smooth counterparts, specifically an LSE-based soft max/min and a sigmoid gate.

A surviving glitch event contributes one rising and one falling transition, and therefore corresponds to two extra transitions in α_{glitch} . The logical generation term further depends on the gate function. For XOR3, pairwise input-switching events are fully visible at the output. For MAJ3, pairwise events are only partially visible because of logic masking. This yields coefficients (2, 4) for XOR3 and $(\frac{1}{2}, 1)$ for MAJ3, obtained by enumerating the visibility of pairwise and triple input-switching events under the respective truth tables.

For the SUM output implementing XOR3, the locally generated glitch activity is

$$\alpha_{\text{gen}}(S) = \left[2(\alpha_{A,\text{func}}\alpha_{B,\text{func}} + \alpha_{A,\text{func}}\alpha_{C_{\text{in}},\text{func}} + \alpha_{B,\text{func}}\alpha_{C_{\text{in}},\text{func}}) - 4\alpha_{A,\text{func}}\alpha_{B,\text{func}}\alpha_{C_{\text{in}},\text{func}} \right] H(\text{skew}, T_{\text{arc}}^S), \quad (15)$$

where the first term denotes pairwise glitch generation and the last term removes the overlap when all inputs switch in the same cycle.

For the CARRY output implementing MAJ3, the locally generated glitch activity is

$$\alpha_{\text{gen}}(C_{\text{out}}) = \left[\frac{1}{2}(\alpha_{A,\text{func}}\alpha_{B,\text{func}} + \alpha_{A,\text{func}}\alpha_{C_{\text{in}},\text{func}} + \alpha_{B,\text{func}}\alpha_{C_{\text{in}},\text{func}}) - \alpha_{A,\text{func}}\alpha_{B,\text{func}}\alpha_{C_{\text{in}},\text{func}} \right] H(\text{skew}, T_{\text{arc}}^{C_{\text{out}}}), \quad (16)$$

Propagated glitch. In addition to local generation, the current FA output may also propagate glitch generated by the preceding stage. Here $\alpha_{\text{glitch}}(A)$, $\alpha_{\text{glitch}}(B)$, and $\alpha_{\text{glitch}}(C_{\text{in}})$ denote the glitch activities carried by the three input signals entering the current FA. For the SUM output implementing XOR3, every input glitch is fully visible at the output. Therefore,

$$\alpha_{\text{prop}}(S) = \alpha_{\text{glitch}}(A) + \alpha_{\text{glitch}}(B) + \alpha_{\text{glitch}}(C_{\text{in}}). \quad (17)$$

For the CARRY output implementing MAJ3, the propagated-glitch contribution is weighted by the corresponding input sensitivities:

$$\Pi_A^{C_{\text{out}}} = p_B + p_{C_{\text{in}}} - 2p_B p_{C_{\text{in}}}, \quad (18)$$

$$\Pi_B^{C_{\text{out}}} = p_A + p_{C_{\text{in}}} - 2p_A p_{C_{\text{in}}}, \quad (19)$$

$$\Pi_{C_{\text{in}}}^{C_{\text{out}}} = p_A + p_B - 2p_A p_B. \quad (20)$$

Then

$$\alpha_{\text{prop}}(C_{\text{out}}) = \alpha_{\text{glitch}}(A)\Pi_A^{C_{\text{out}}} + \alpha_{\text{glitch}}(B)\Pi_B^{C_{\text{out}}} + \alpha_{\text{glitch}}(C_{\text{in}})\Pi_{C_{\text{in}}}^{C_{\text{out}}}. \quad (21)$$

Combining Equations (12), (15) to (17) and (21), the overall transition activities of the FA outputs are

$$\alpha(S) = \alpha_{S,\text{func}} + \alpha_{\text{glitch}}(S), \quad (22)$$

$$\alpha(C_{\text{out}}) = \alpha_{C_{\text{out}},\text{func}} + \alpha_{\text{glitch}}(C_{\text{out}}). \quad (23)$$

Propagating Equations (22) and (23) over the CT topology yields the rewiring-dependent transition activity of all internal signals.

3.4 Reconvergence-Aware Correction

The expressions in Sections 3.1 to 3.3 use marginal probabilities and therefore implicitly approximate the inputs of each compressor as independent. This becomes inaccurate when two internal signals depend on overlapping PP ancestors. In a CT, however, such dependence is highly structured: the rewired tree remains locally connected and does not create new arbitrary fanout mid-tree, so the dominant correlation is inherited from shared PP ancestry. Then we apply a compact first-order correction based on source sensitivities.

For each stage-0 PP source q_k , we define an ancestry vector $\mathbf{v}_{q_k} = \mathbf{e}_k$. For an internal signal $y = g(x_1, \dots, x_m)$, we propagate its ancestry by

$$\mathbf{v}_y = \sum_{t=1}^m \frac{\partial p_y}{\partial p_{x_t}} \mathbf{v}_{x_t}, \quad (24)$$

where the derivatives come directly from the analytic probability expressions in Section 3.1. Each entry $v_y^{(k)}$ thus measures the first-order sensitivity of p_y to the PP source q_k , rather than a binary ancestor indicator. Using the covariance matrix Σ_0 of the stage-0 PP sources under the workload, we approximate the corrected joint terms as

$$\text{Cov}(x, y) \approx \mathbf{v}_x^\top \Sigma_0 \mathbf{v}_y, \quad (25)$$

$$J_{xy} = p_x p_y + \text{Cov}(x, y),$$

$$J_{xyz} \approx p_x J_{yz} + p_y J_{xz} + p_z J_{xy} - 2p_x p_y p_z.$$

Here, J_{xy} and J_{xyz} replace the independence products $p_x p_y$ and $p_x p_y p_z$ in the compressor formulas. We retain pairwise shared-input correlations through Σ_0 and use a pairwise-consistent closure for the triple joint term, neglecting higher-order cumulants.

For example, the corrected FA SUM probability is written as

$$p_S^{\text{rcv}} = p_A + p_B + p_{C_{\text{in}}} - 2(J_{AB} + J_{AC_{\text{in}}} + J_{BC_{\text{in}}}) + 4J_{ABC_{\text{in}}}, \quad (26)$$

and $p_{C_{\text{out}}}^{\text{rcv}}$ is obtained analogously from Equation (8). The MAJ3 sensitivity terms in Equations (18) to (20) are corrected in the same way by replacing the corresponding independence products with the joint terms in Equation (25). The corrected probabilities and sensitivities are then reused in Equations (9), (15) to (17) and (21). Note that this correction targets only the logic-probability and sensitization expressions; residual higher-order co-switching and pulse-shape effects are

left to the calibration layer in Section 3.5. Since Equations (24) to (26) involve only analytic probability derivatives, bilinear forms, and low-order algebraic operations, the correction remains differentiable with respect to the rewiring variables.

3.5 Final Power Surrogate

After propagation and correction, each compressor output has rewiring-dependent functional, generated-glitch, and propagated-glitch activities. We convert them into local power terms. For a fixed compressor output node r and component $\star \in \{\text{func}, \text{gen}, \text{prop}\}$, we write

$$P_{r,\star}^{\text{ana}}(M) = f_{\text{clk}} E_r(M) \alpha_{\star}(r; M), \quad (27)$$

where $E_r(M) = \frac{1}{2} V_{DD}^2 C_r(M)$ is the effective energy per transition at output node r , with $C_r(M)$ collecting the cell-internal switching capacitance and the local load/wire contribution after placement, consistent with Equation (1).

A small systematic residual remains because timing filtering is smoothed and reconvergence is corrected only to first order. We therefore apply a lightweight position-wise calibration directly to the three analytic power components in Equation (27):

$$\hat{P}_{\text{CT}}(M) = \sum_{r \in \mathcal{R}_{\text{CT}}} C_r \left(P_{r,\text{func}}^{\text{ana}}(M), P_{r,\text{gen}}^{\text{ana}}(M), P_{r,\text{prop}}^{\text{ana}}(M) \right), \quad (28)$$

where $\mathcal{R}_{\text{CT}}$ denotes the set of fixed compressor output nodes. Each C_r is a 4-parameter non-negative affine map $\beta_{r,0} + \beta_{r,\text{func}} z_{\text{func}} + \beta_{r,\text{gen}} z_{\text{gen}} + \beta_{r,\text{prop}} z_{\text{prop}}$ with $\beta_{r,\cdot} \geq 0$, fit by Tikhonov-regularized non-negative least squares [12] on a held-out calibration set. The non-negativity constraint enforces monotone contribution of each physical power component. Because the calibration only applies a low-dimensional residual correction to the physically defined local power components above, introducing no free-form topology features or learned latent representations, the model remains compact and interpretable. Since $E_r(M)$, $\alpha_{\star}(r; M)$, and $C_r(\cdot)$ are differentiable, $\hat{P}_{\text{CT}}(M)$ is differentiable with respect to the rewiring variables.

The CPA netlist remains fixed and contributes a smaller fraction of multiplier power than the CT, but its input statistics are still determined by the CT outputs. We therefore propagate probabilities and activities over the fixed CPA netlist using the same formulation and obtain $\hat{P}_{\text{CPA}}(M)$. The full multiplier surrogate model is then

$$\hat{P}_{\text{mul}}(M) = \hat{P}_{\text{CT}}(M) + \hat{P}_{\text{CPA}}(M), \quad (29)$$

omitting the partial-product generator, which is design-independent under rewiring and contributes only a constant offset. This surrogate model, together with the delay estimate $\hat{D}_{\text{mul}}(M)$ introduced above, is used in Section 4.

4 Two-Phase Optimization

4.1 Overview

With the differentiable power surrogate model of Section 3 in place, the remaining question is how to optimize over the combinatorial rewiring space. Directly searching the discrete space $\mathcal{M}_{\text{global}} = \prod_{(i,j) \in \mathcal{S}} \mathcal{M}_{i,j}$ is infeasible. Its size grows combinatorially as the Cartesian product of per-slice permutation sets, while evaluating each candidate requires gate-level simulation and timing-based power analysis, which becomes prohibitive at search scale.

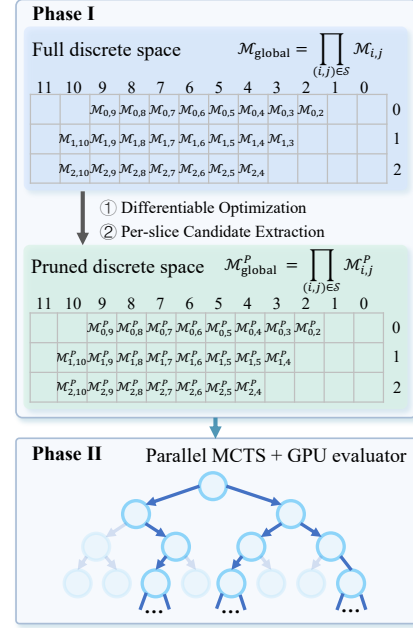


Figure 6: Two-phase rewiring optimization pipeline. Phase I contracts the full discrete rewiring space into a pruned discrete space through differentiable optimization and per-slice candidate hardening. Phase II searches the pruned space with parallel MCTS and batched GPU evaluation.

We therefore adopt a two-phase framework, illustrated in Figure 6. In Phase I, each slice-wise rewiring matrix is relaxed to a continuous doubly-stochastic matrix and optimized with the differentiable power surrogate model of Section 3 under a joint power–timing objective. Its role is not to output the final discrete solution, but to concentrate probability mass on a small set of promising local bijections and contract the search space. In Phase II, we search the pruned space via Monte Carlo tree search (MCTS), with leaf rewards computed by a GPU evaluator. Differentiable optimization is thus used to make discrete search tractable, not to replace it.

4.2 Phase I: Differentiable Rewiring Optimization

Continuous relaxation. For each slice (i, j) , we relax the discrete rewiring matrix $M_{i,j} \in \{0, 1\}^{s_{i,j} \times s_{i,j}}$ via the Sinkhorn operator on a real-valued logit matrix $\tilde{M}_{i,j} \in \mathbb{R}^{s_{i,j} \times s_{i,j}}$:

$$M_{i,j} = \text{Sinkhorn}(\tilde{M}_{i,j}/\tau), \quad (30)$$

which yields a doubly-stochastic $M_{i,j}$ that anneals toward a permutation as $\tau \rightarrow 0$. Unlike softmax-based relaxations that require auxiliary bijection and discretization losses [7], Sinkhorn preserves doubly-stochasticity by construction.

Joint power–timing objective. Using the differentiable power surrogate model $\hat{P}_{\text{mul}}(M)$ and delay estimate $\hat{D}_{\text{mul}}(M)$ of Section 3, we minimize

$$\mathcal{L}(M) = \hat{P}_{\text{mul}}(M) + \lambda \psi(\hat{D}_{\text{mul}}(M) - T_{\text{clk}}), \quad (31)$$

Algorithm 1 Phase I: Differentiable Optimization and Hardening

Require: Multiplier structure, workload statistics, differentiable surrogate models $\widehat{P}_{\text{mul}}$ and $\widehat{D}_{\text{mul}}$, clock period T_{clk} , candidate budget K

Ensure: Pruned search space $\{\mathcal{M}_{i,j}^P\}_{(i,j) \in \mathcal{S}}$

1: Initialize logits $\{\tilde{M}_{i,j}\}_{(i,j) \in \mathcal{S}}$

Progressive Differentiable Optimization

2: **for** stage $s = 0, 1, \dots, n-1$ **do**

3: Freeze all variables except $\{\tilde{M}_{s,j}\}_j$

4: Optimize $\{\tilde{M}_{s,j}\}_j$ for several gradient steps by minimizing

$$\mathcal{L}(M) = \widehat{P}_{\text{mul}}(M) + \lambda \psi(\widehat{D}_{\text{mul}}(M) - T_{\text{clk}})$$

where $M_{i,j} = \text{Sinkhorn}(\tilde{M}_{i,j}/\tau)$

5: **end for**

6: Unfreeze all $\{\tilde{M}_{i,j}\}$ and jointly refine

7: Obtain refined relaxed matrices $\{M_{i,j}^*\}_{(i,j) \in \mathcal{S}}$

Per-Slice Hardening

8: **for** each active slice $(i, j) \in \mathcal{S}$ **do**

9: Extract K high-scoring feasible permutations from $M_{i,j}^*$

10: Form $\mathcal{M}_{i,j}^P = \{\pi_{i,j}^{(1)}, \dots, \pi_{i,j}^{(K)}\}$

11: **end for**

12: **return** $\{\mathcal{M}_{i,j}^P\}_{(i,j) \in \mathcal{S}}$

where $\psi(\cdot)$ is a differentiable penalty that is inactive when the timing constraint is satisfied and λ controls the power–timing trade-off. This formulation keeps the optimization aligned with the original constrained problem while remaining fully differentiable.

Progressive stage-wise optimization. Because the CT is strictly feed-forward, the signal statistics and arrival times at stage s are determined by the upstream stages. We therefore optimize progressively from stage 0 to stage $n-1$, updating only the variables of the current stage while the rest remain fixed. Each stage is optimized against the actual upstream distributions produced by the current solution, rather than against transient averages from a simultaneous joint update. Later stages are temporarily held fixed and revisited in subsequent stages and in the final joint refinement. After the sweep, we unfreeze all variables for a short refinement pass to recover residual cross-stage coupling.

The refined relaxed solution is then hardened per-slice into a candidate set of K high-scoring permutations as described in Section 4.3. Algorithm 1 summarizes the full Phase I procedure.

4.3 Per-Slice Candidate Extraction and Space Pruning

Figure 7 zooms into one slice. Rather than commit that slice to a single permutation after Phase I, which would prematurely eliminate combinations that remain globally competitive under discrete coupling, we retain a per-slice candidate set

$$\mathcal{M}_{i,j}^P = \{\pi_{i,j}^{(1)}, \dots, \pi_{i,j}^{(K)}\} \subset \mathcal{M}_{i,j} \quad (32)$$

of K high-scoring feasible permutations extracted from $M_{i,j}^*$, yielding a pruned global space $\mathcal{M}_{\text{global}}^P = \prod_{(i,j) \in \mathcal{S}} \mathcal{M}_{i,j}^P$ with per-slice branching factor reduced from $s_{i,j}!$ to K .

Although extraction is performed independently per slice, the scores in $M_{i,j}^*$ are produced by the joint Phase I optimization and

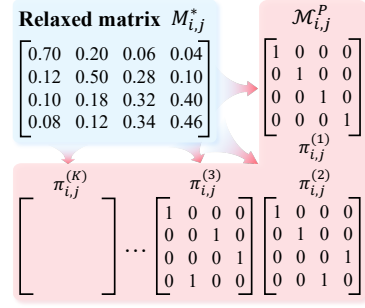


Figure 7: Per-slice candidate extraction. For one slice (i, j) , the relaxed matrix $M_{i,j}^*$ is hardened into a small candidate set $\mathcal{M}_{i,j}^P = \{\pi_{i,j}^{(1)}, \dots, \pi_{i,j}^{(K)}\}$ for Phase II search.

thus carry the cross-slice context. Residual inter-slice coupling is resolved explicitly in Phase II through parallel MCTS.

4.4 GPU-Accelerated Power Evaluator

Even after Phase I pruning, the discrete space $\mathcal{M}_{\text{global}}^P$ remains combinatorially large. Phase II therefore requires fast, large-scale parallel evaluation, which the conventional EDA flow cannot provide: per-design gate-level simulation with timing-based power analysis still runs on the order of minutes and is further bounded by license quotas and CPU-side parallelism.

We therefore develop a GPU-accelerated evaluator tailored to multiplier and MAC rewiring. Unlike prior GPU simulators for general circuits [13, 14], which accelerate a single netlist by parallelizing across its gates and cycles, our setting offers a different opportunity: all candidate designs share the identical compressor assignment and differ only in their internal CT wiring. We exploit this by compiling once into a shared template (CT structure, leveled execution schedule, and delay/power lookup tables), and representing each candidate as a compact wiring tensor of integer indices. Changing a candidate amounts to updating these indices, with no graph reconstruction.

This formulation exposes three orthogonal forms of parallelism: (i) *design-parallel*, since N candidates follow the same simulation schedule; (ii) *cycle-parallel*, since the CT is combinational and different input cycles are independent; (iii) *gate-parallel*, since compressors at the same stage are mutually independent under leveled evaluation.

The same compile-once mechanism extends directly to the CPA, yielding full-multiplier dynamic power evaluation; the partial-product generator contributes a design-independent constant and is absorbed outside the search reward. For a 16-bit multiplier, the evaluator processes one design in 2.5 ms on a single GPU, achieves over 99% gate-level transition accuracy relative to commercial SDF-annotated simulation, and supports batched evaluation of thousands of candidates. Against the several-minute cost of the full EDA flow per design, this throughput is what makes large-scale parallel MCTS exploration of the pruned space feasible.

4.5 Phase II: Parallel MCTS in the Pruned Space

With the pruned space of Section 4.3 and the batched evaluator of Section 4.4 in place, the remaining task is to allocate the evaluation budget over candidate combinations. We therefore perform a second-stage discrete search with parallel MCTS, whose role is to refine the

Table 1: Power model fidelity comparison.

Method	Spearman ρ
OpenROAD [15]	-0.239
Commercial IP	0.456
GLSVLSI'07 [9]	0.406
TVLSI'20 [10]	0.431
MLP	0.627
GCN	0.726
TAP-MAC (Ours)	0.916

pruned space and recover the residual inter-slice coupling left by per-slice hardening.

Let $(i_1, j_1), (i_2, j_2), \dots, (i_{|S|}, j_{|S|})$ denote the active slices ordered first by stage and then by column. A node at depth t represents a partial assignment to the first t slices in this order. Expanding a node selects one candidate permutation from $\mathcal{M}_{i_{t+1}, j_{t+1}}^P$ for the next slice, and a complete root-to-leaf path therefore defines one discrete rewiring in $\mathcal{M}_{\text{global}}^P$. This ordering follows the stage-column dependency order of the CT while keeping the tree definition consistent with the per-slice hardening in Section 4.3.

Newly expanded leaves are evaluated by the GPU evaluator of Section 4.4, which reports full-multiplier dynamic power under delay-aware simulation. Timing feasibility is incorporated directly in the search reward, and multiple tree traversals are executed in parallel so that their leaf evaluations can be grouped into GPU batches before backup. MCTS is well suited to this setting because the quality of a partial assignment cannot be determined reliably from local Phase-I slice scores alone; it becomes clear only after downstream slices are instantiated and evaluated jointly. Phase II therefore refines the Phase-I-pruned space under batched GPU evaluation and recovers the remaining discrete coupling.

5 Experimental Results

5.1 Setup

The proposed TAP-MAC framework is implemented on a Linux server equipped with an Intel Xeon Platinum 8468 CPU @ 2.1 GHz, 2 TB of memory, and eight NVIDIA H100 GPUs. We evaluate TAP-MAC on both multipliers and MAC units at 8-bit and 16-bit configurations. Compressor trees are constructed from FA and HA cells in the NanGate 45 nm Open Cell Library [16], using the same CT structures as UFO-MAC [6], and all designs share an identical CPA structure. Within each configuration, all candidate designs have an identical structure except for the internal CT wiring. All designs share a common initial cell placement generated by Cadence Innovus [17]. For ECO-style rewiring, we perform incremental routing using the ecoDesign flow in Innovus. We then run Synopsys VCS [18] to simulate representative workloads and generate VCD activity traces, which are analyzed by Synopsys PrimeTime PX [19] in time-based power analysis mode to obtain workload-specific dynamic power and critical-path delay.

TAP-MAC is implemented in PyTorch [20]. Phase I runs 1500 Adam steps per stage (learning rate 0.05, Sinkhorn temperature annealed from $\tau = 2.0$ to 0.05) and retains $K = 5$ candidate permutations per slice. Phase II performs MCTS over 10^5 leaf evaluations via the GPU evaluator. The clock constraint T_{clk} matches the critical-path delay of DOMAC [7], and the timing penalty weight is set to $\lambda = 1$.

5.2 Fidelity of the Power Model

We evaluate the fidelity of our power model on a 16-bit multiplier under a non-uniform workload, generating 20,000 random CT wiring variants and running the full EDA flow to obtain ground-truth power. Following [9], the non-uniform workload samples operand values from a log-normal distribution to capture realistic skew in input bit-level statistics. We compare TAP-MAC against six baselines in three groups: (i) two static vectorless power analysis EDA tools, OpenROAD [15], whose integrated OpenSTA [21] engine implements Najm’s transition-density propagation [11], and a leading commercial tool, both supplied with the workload’s per-bit signal probabilities and toggle rates; (ii) two prior analytic models with coarse or simplified modeling, GLSVLSI’07 [9] and TVLSI’20 [10]; and (iii) two learned regressors trained on the same labeled designs under 5-fold stratified cross-validation, a Multi-Layer Perceptron (MLP) on flat per-design features and a Graph Convolutional Network (GCN) on the explicit circuit graph.

We report the Spearman rank correlation ρ between each model’s predicted power and the EDA ground truth. Given N designs with predicted ranks r_i^{pred} and true ranks r_i^{true} , Spearman’s ρ is computed as

$$\rho = 1 - \frac{6 \sum_{i=1}^N (r_i^{\text{pred}} - r_i^{\text{true}})^2}{N(N^2 - 1)}, \quad (33)$$

and ranges over $[-1, 1]$: $\rho = 1$ indicates a perfect monotonic ranking.

Table 1 reports the fidelity of each model. By explicitly modeling glitch power and the spatial correlations induced by reconvergent fanout, TAP-MAC attains substantially higher fidelity than all baselines. The learned MLP and GCN regressors also fall short: the power-activity structure inside a multiplier’s compression tree is complex, governed by many interacting effects, and the narrow spread of power values leaves too little discriminative signal for purely data-driven fits.

5.3 Comparison on Multipliers and MACs

We evaluate dynamic power and critical-path delay of multipliers and MAC units under uniform and non-uniform workloads. Since ECO-style post-placement rewiring preserves the initial cell placement and total cell area, power and delay are the two metrics that vary across methods. Besides a commercial IP, we compare against the following rewiring-based baselines:

- **UFO-MAC** [6]: ILP-based rewiring targeting timing.
- **DOMAC** [7]: differentiable rewiring targeting timing.
- **GLSVLSI’07** [9]: heuristic pruning-and-enumeration for power.
- **TVLSI’20** [10]: greedy rewiring for power.
- **RL**: RL-based multiplier optimization [5], originally aimed at compressor-tree structural search; we adapt it to the rewiring problem.

The comparison results are shown in Tables 2 and 3. In every configuration, TAP-MAC achieves the lowest power across all baselines. Over the commercial IP, the power reduction reaches up to 8% on the uniform workload and up to 15% on the non-uniform workload, indicating that rewiring yields greater benefit when the workload is more imbalanced. The RL baseline fails to converge in such a combinatorial rewiring space. Prior analytic methods [9, 10] rely on

Table 2: Multiplier (MUL) comparison across bitwidths and workloads.

Method	Uniform workload				Non-uniform workload			
	8-bit		16-bit		8-bit		16-bit	
	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)
Commercial IP	1.498 (1.065×)	1.413	4.288 (1.047×)	1.946	0.759 (1.182×)	1.413	2.135 (1.182×)	1.946
RL	1.516 (1.078×)	1.434	4.367 (1.066×)	1.949	0.724 (1.128×)	1.423	2.019 (1.118×)	1.957
UFO-MAC [6]	1.529 (1.087×)	1.361	4.519 (1.103×)	1.913	0.764 (1.190×)	1.361	2.167 (1.200×)	1.913
DOMAC [7]	1.546 (1.100×)	1.367	4.492 (1.096×)	1.913	0.752 (1.171×)	1.367	2.154 (1.193×)	1.913
GLSVLSI'07 [9]	1.512 (1.075×)	1.429	4.261 (1.040×)	1.951	0.709 (1.104×)	1.437	1.948 (1.079×)	1.949
TVLSI'20 [10]	1.519 (1.080×)	1.427	4.307 (1.051×)	1.956	0.728 (1.134×)	1.419	1.971 (1.091×)	1.951
TAP-MAC (Ours)	1.406 (1.000×)	1.361	4.097 (1.000×)	1.910	0.642 (1.000×)	1.358	1.806 (1.000×)	1.905

Table 3: Multiply-accumulate (MAC) unit comparison across bitwidths and workloads.

Method	Uniform workload				Non-uniform workload			
	8-bit		16-bit		8-bit		16-bit	
	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)	Power (mW)	Delay (ns)
Commercial IP	2.101 (1.068×)	1.479	4.876 (1.087×)	2.069	1.440 (1.163×)	1.479	2.892 (1.131×)	2.069
RL	2.205 (1.120×)	1.496	4.902 (1.093×)	2.162	1.429 (1.154×)	1.513	2.779 (1.087×)	2.110
UFO-MAC [6]	2.289 (1.163×)	1.426	5.146 (1.147×)	2.009	1.513 (1.222×)	1.426	2.953 (1.155×)	2.009
DOMAC [7]	2.243 (1.140×)	1.428	5.097 (1.136×)	1.998	1.492 (1.205×)	1.428	2.917 (1.141×)	1.998
GLSVLSI'07 [9]	2.161 (1.098×)	1.504	4.895 (1.091×)	2.113	1.361 (1.099×)	1.502	2.684 (1.050×)	2.078
TVLSI'20 [10]	2.194 (1.115×)	1.495	4.962 (1.106×)	2.206	1.396 (1.128×)	1.496	2.754 (1.077×)	2.088
TAP-MAC (Ours)	1.968 (1.000×)	1.426	4.485 (1.000×)	1.994	1.238 (1.000×)	1.426	2.557 (1.000×)	1.996

Table 4: Ablation of the two-phase optimization (16-bit MUL, non-uniform workload).

Variant	Power (mW)	Δ (%)
Phase I only	1.864	+3.21
Phase II only	2.077	+15.01
Phase I + Phase II-surrogate	1.834	+1.55
TAP-MAC (Full)	1.806	–

coarser power models and simpler search strategies, whereas TAP-MAC combines a finer-grained analytic model with the two-phase optimization framework to achieve the best overall results.

5.4 Ablation Studies

We ablate the two-phase optimization framework on the 16-bit multiplier under the non-uniform workload; results are reported in Table 4. We consider three ablated variants: *Phase I only* drops the Phase II MCTS refinement and takes the discretized Phase I solution as the final output. *Phase II only* skips Phase I and runs MCTS directly over the full solution space without pruning. *Phase I + Phase II-surrogate* replaces the Phase II GPU-simulator reward with the differentiable surrogate model of Section 3.

Removing either phase degrades the final power. Phase II only incurs the largest loss (+15.01%): without Phase I pruning, MCTS cannot converge within the combinatorial rewiring space. Phase I only loses 3.21%, reflecting the discrete refinement that MCTS adds on top of Phase I. Replacing the Phase II reward with the analytic surrogate model worsens the result by +1.55%, indicating that the higher-fidelity GPU evaluator contributes to Phase II quality.

5.5 Integration in DSP and DNN Accelerator Blocks

To evaluate TAP-MAC under real workloads, we integrate the optimized multipliers and MACs into two representative functional

Table 5: Dynamic power of representative functional blocks instantiated with the commercial IP versus TAP-MAC.

Block	Freq.	Commercial	TAP-MAC	Δ (%)
CCM	250MHz	3.875mW	3.367mW	–13.1
Systolic	500MHz	257.4mW	234.4mW	–8.9

blocks and compare against the commercial IP. For the DSP case, we replace the 16-bit multipliers inside a 3×3 color correction matrix (CCM) from an image signal processor (ISP) pipeline under a low-light surveillance pixel workload. For the DNN case, we replace the 8-bit MACs inside a 16×16 weight-stationary systolic array under a real workload from a quantized layer of GPT-2 [22]. Table 5 reports the dynamic power of each block and the relative reduction over the commercial IP. TAP-MAC reduces the CCM power by 13.1% at 250 MHz and the systolic-array power by 8.9% at 500 MHz. These results confirm that TAP-MAC delivers consistent power savings when deployed in representative DSP and DNN functional blocks.

6 Conclusion

We presented TAP-MAC, a topology-aware low-power optimization framework for multipliers and MACs. TAP-MAC combines a differentiable power surrogate model with a two-phase optimizer. Experiments show substantially higher power prediction fidelity than prior baselines. Future work will extend this methodology to other datapath components.

Acknowledgment

This work is supported in part by the Department of Education of Guangdong Province (No. 2024KTSCX037).

References

- [1] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, 1964.
- [2] L. Dadda, "Some schemes for fast serial input multipliers," in *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, 1983.
- [3] V. Oklobdzija, D. Villeger, and S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Transactions on Computers*, 1996.
- [4] W. Xiao, W. Qian, and W. Liu, "Gomil: Global optimization of multiplier by integer linear programming," in *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021.
- [5] D. Zuo, Y. Ouyang, and Y. Ma, "RI-mul: Multiplier design optimization with deep reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.
- [6] D. Zuo, J. Zhu, C. Li, and Y. Ma, "Ufo-mac: A unified framework for optimization of high-performance multipliers and multiply-accumulators," in *Proc. ICCAD*, 2024.
- [7] C. Xue, Y. Ren, J. Zhou, K. Li, C. Zhang, Y. Lin, L. Zhang, Q. Xu, and G. Sun, "Domac: Differentiable optimization for high-speed multipliers and multiply-accumulators," in *2025 International Symposium of Electronics Design Automation (ISED)*, 2025.
- [8] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: <https://doi.org/10.1145/3316781.3317803>
- [9] S. T. Oskuii, P. G. Kjeldsberg, and O. Gustafsson, "Transition-activity aware design of reduction-stages for parallel multipliers," in *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '07, 2007.
- [10] A. C. Ranasinghe and S. H. Gerez, "Glitch-optimized circuit blocks for low-power high-performance Booth multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2020.
- [11] F. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1993.
- [12] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, 1970.
- [13] Y. Zhang, H. Ren, A. Sridharan, and B. Khailany, "GATSPI: GPU accelerated gate-level simulation for power improvement," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: <https://doi.org/10.1145/3489517.3530601>
- [14] Y. Zhang, H. Ren, and B. Khailany, "GL0AM: GPU logic simulation using 0-delay and re-simulation acceleration method," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: <https://doi.org/10.1145/3676536.3676675>
- [15] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. DAC*, 2019.
- [16] Nangate Inc., "Open Cell Library v2008_10 SP1," <http://www.nangate.com/openlibrary/>, 2008.
- [17] Cadence Design Systems, Inc., "Innovus Implementation System," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html.
- [18] Synopsys, Inc., "VCS Functional Verification Solution," <https://www.synopsys.com/verification/simulation/vcs.html>.
- [19] —, "PrimeTime: Signoff Timing and Power Analysis," <https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html>.
- [20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimselshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 8024–8035.
- [21] Parallax Software Inc., "OpenSTA," <https://github.com/The-OpenROAD-Project/OpenSTA>.
- [22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.