DiffCCD: Differentiable Concurrent Clock and Data Optimization

Yuhao Ji[1,3] Yuntao Lu[1] Zuodong Zhang[3] Zizheng Guo[2,3] Yibo Lin[2,3,4] Bei Yu[1]

[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR
[2]School of Integrated Circuits, Peking University, Beijing, China
[3]Institute of Electronic Design Automation, Peking University, Wuxi, China
[4]Advanced Innovation Center for Integrated Circuits, Beijing, China

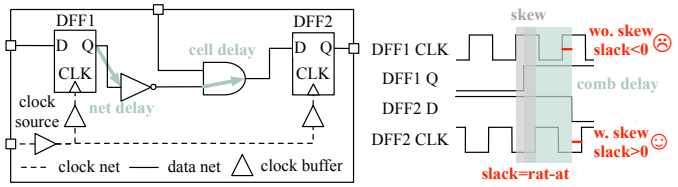
Abstract*—Timing optimization following clock tree synthesis (post-CTS) is a crucial step in very large scale integration (VLSI) physical design for achieving timing closure. During this stage, clock skew significantly impacts circuit timing performance, making useful skew optimization essential for enhancing design quality. However, traditional skew optimization methods face challenges due to their insufficient consideration of physical implementation constraints. To overcome these limitations, we propose a GPU-accelerated differentiable concurrent clock and data (CCD) optimization framework, which simultaneously optimizes clock skew and logic delays to enhance overall timing performance with the consideration of physical constraints. We implement the CCD optimization method as a step involving buffer sizing in the clock network and refining placement results. The key innovation of our approach lies in formulating a smooth and differentiable process for CCD optimization with a calibration mechanism to ensure accurate gradient computations. Additionally, we employ an alternating direction method of multipliers (ADMM)-based strategy to decompose the entire optimization problem into several manageable subproblems, effectively balancing timing optimization with physical implementation constraints. Experimental results on open-source industrial benchmarks demonstrate that our CCD optimization framework achieves superior timing closure compared to a baseline approach within an open-source physical design tool. Our method yields an average improvement of 22.4% in worst negative slack (WNS) and 45.0% in total negative slack (TNS), along with a 9.434× runtime speedup. To our knowledge, this is the first work to incorporate clock skew effects into gradient-based timing optimization.

I. Introduction

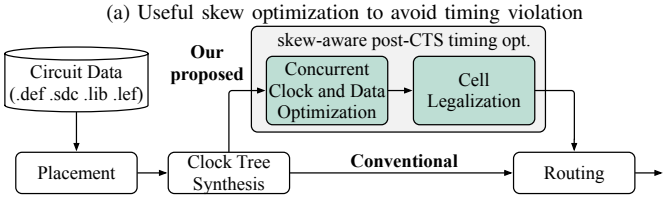
As circuit complexity in VLSI design continues to increase, achieving timing closure remains a critical challenge due to sophisticated timing models and tight coupling with other design objectives. This challenge is exacerbated in the post clock tree synthesis (CTS) stage, where timing analysis must account for clock net delays. The variation in clock signal arrival times at different locations, known as clock skew, significantly affects the overall circuit timing performance.

As illustrated in Fig. 1(a), by properly controlling clock skew, designers can improve both system robustness and performance [1]. However, traditional CTS methodologies primarily aim to minimize skew by constructing balanced clock networks [2]–[4]. While this simplifies implementation, it fundamentally prevents using clock skew as a valuable timing optimization resource, which becomes particularly problematic for high-speed designs. Similarly, research in other physical design flows, such as placement, typically operates under the assumption of an ideal clock net, i.e. zero skew [5], [6]. This assumption leads to either overly optimistic timing estimations that result in closure failures, or pessimistic estimations that cause over-design, wasting power and area while extending optimization cycles.

Recognizing these limitations, numerous efforts have emerged to leverage useful skew to improve timing performance. Specifically, carefully adjusted clock skew is adopted to resolve timing violations. Existing works [1], [7]–[11] have presented effective algorithms for



(a) Useful skew optimization to avoid timing violation



(b) Physical design workflow w./wo. CCD optimization

Fig. 1 One example of useful skew optimization and the proposed physical design workflow. The netlist diagram distinguishes between **clock net** (dashed line) and **data net** (solid line), showing how proper clock skew can compensate for delays to prevent timing violations.

deriving useful skew solutions. However, traditional useful skew optimizations face a chicken-and-egg dilemma due to the interdependence and lack of integration between skew optimizations and other physical design flows [8]. Although researchers have explored approaches such as iterative back-annotation [7] or predictive optimization flow [8], they implement clock skew scheduling as an isolated step. This isolation limits the global view and makes it difficult to capture the intricate interplay between clock net optimization and other optimization processes. Consequently, most works neglect physical implementation constraints for their proposed useful skew solutions, leading to poor optimization performance in practice. For instance, it is shown that there should be a compromise in area efficiency and power consumption when skew significantly differs among nearby flip-flops [12]. While [13] considers useful skew scheduling and implements the solution during CTS, the fixed placement limits the solution space for further timing improvement.

Furthermore, we argue that useful skew optimization presents a highly non-smooth solution space where greedy methods easily converge to local optima. Unlike conventional timing optimizations that primarily focus on delay reduction in datapaths [14], [15], useful skew optimization requires bidirectional delay adjustments where certain flip-flops must receive accelerated clock signals while others require deliberately delayed clock delivery. This fundamental difference necessitates a comprehensive global optimization framework capable of handling the complex and competing requirements across the entire design.

To overcome these limitations, we propose DiffCCD, a GPU-

accelerated differentiable concurrent clock and data (CCD) optimization approach, which simultaneously optimizes clock skew and logic delay to enhance timing performance. Since clock net delay is primarily attributed to buffer and wire delays, our method performs clock buffer sizing and refines placement results in the post-CTS stage. Specifically, our approach leverages a gradient-based optimization framework, analogizing timing optimization to quantization-aware training (QAT) in deep learning, with an integrated calibration mechanism to ensure gradient correctness. In contrast to traditional step-by-step optimizations, our solution explores a broader design space while considering physical implementation constraints, resulting in a fine balance between clock net and data net requirements. Also, our proposed CCD optimization can be integrated seamlessly with existing EDA flows, the overall workflow of which is illustrated in Fig. 1(b). Comprehensive experiments demonstrate superior timing closure results compared to conventional methods. Our key contributions are summarized as follows:

- We build a differentiable CCD optimization framework that enables effective and efficient useful skew optimization in the post-CTS stage.
- We propose a logits-based parameterization to model buffer sizing decisions, allowing for smooth transitions between discrete choices. Also, we innovatively introduce straight-through estimator (STE) approaches with a calibration mechanism that effectively bridge the gap between accurate timing analysis and gradient generation, ensuring stable convergence during optimization.
- We introduce an alternating direction method of multipliers (ADMM) optimization approach that effectively handles the gradient conflicts between timing, wirelength, and density objectives.
- We implement the entire framework with GPU acceleration, demonstrating superior timing performance and runtime improvement compared to the baseline.

II. Preliminaries

A. Static Timing Analysis

Static timing analysis (STA) is essential for validating a design's timing compliance with specified performance requirements, given input clock definitions and external constraints. The STA flow consists of three fundamental stages, as shown in Fig. 2. First, STA constructs routing trees based on cell positions to extract parasitic parameters. The Elmore model [16] is predominantly employed to compute key electrical parameters, including net delay, impulse response, and downstream capacitive load. The model's results depend on both routing tree topology and pin capacitances that vary with cell sizes. Second, STA performs level-by-level slew propagation through the circuit graph to derive cell delay. While the Elmore model provides analytical equations for interconnect analysis, cell timing characterization relies on nonlinear delay model (NLDM). NLDM employs a comprehensive set of look-up tables (LUTs) that characterize cell delay and output slew as functions of input slew and output load capacitance. Each cell type corresponds to a unique set of LUTs, providing accurate timing information across different operating conditions. Finally, STA generates a timing graph by transforming the circuit netlist into a directed acyclic graph (DAG). In this representation, nodes correspond to cell pins while edges represent timing relationships through both cell and net arcs. The analysis then propagates arrival times through this DAG and incorporates timing constraints to compute critical timing metrics, specifically the total negative slack (TNS) and worst negative slack (WNS), which quantify the timing violations.

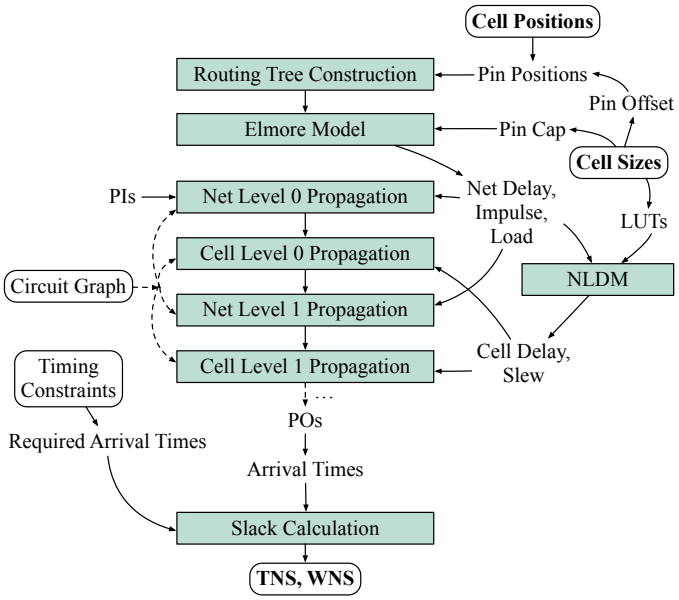


Fig. 2 Overview of STA flow, showing the computational process from cell positions and cell sizes to timing metrics, i.e. TNS and WNS.

B. Gradient-based Optimizations

Gradient-based optimizations have been widely adopted in physical design stages, such as placement [5], [15], [17] and routing [18]. This methodology computes gradients of the objective function with respect to design variables, such as cell positions, through backward propagation, and iteratively updates them using gradient descent.

To enable timing-driven optimization, existing works explicitly incorporate timing terms, typically TNS and WNS, into the objective function, thus requiring the entire timing analysis process to be differentiable. As discussed in Section II-A, accurate timing analysis requires constructing a routing tree for each net for accurate extraction of RC parasitics. Rectilinear Steiner Minimal Tree (RSMT) has been demonstrated to be a reliable approximation for routing solutions, closely representing the real implementation. However, the construction of RSMT is inherently non-continuous [19], i.e., small perturbations in pin positions can trigger abrupt topological transitions in the RSMT structure, leading to discrete jumps in RC parasitics and subsequent timing calculations. This non-differentiable behavior poses significant challenges when integrated into gradient-based optimization frameworks, where smooth and continuous objective functions are essential for convergence. [5] proposed directly applying gradients of steiner points to corresponding branches while periodically reconstructing the RSMT to maintain gradient accuracy. However, this approximation merely enforces differentiability without addressing the fundamental discontinuity in RC tree construction.

To establish a truly differentiable framework for routing tree construction, we draw inspiration from the straight-through estimator (STE) approach [20], a technique widely used in quantized neural network training in deep learning. STE enables optimization through non-differentiable operations by maintaining precise computations in forward propagation while approximating their gradients in backward propagation. Specifically, our methodology preserves accurate RSMT-based delay calculations during forward timing analysis while implementing a continuous and smooth degenerated RC model with calibrated gradients for backward propagation. The details are elaborated in Section IV-C.

TABLE I Summary of notations.

Notation	Description
TM	Timing objective function
$\mathcal{N}$	Set of all circuit instances
$\mathcal{N}_\mathrm{c}$	Set of clock buffer instances (subset of $\mathcal{N}$)
$\mathcal{S}$	Set of available clock buffer sizes
$\mathcal{P}$	Coordinates of all instances
$\mathcal{L}_\mathrm{c}$	Logits for clock buffer sizes
$\boldsymbol{s}_\mathrm{c}$	Vector of chosen clock buffer sizes
l_b^s	Logit value for buffer b selecting size s
θ_b^s	Probability of buffer b selecting size s
s_b	Chosen size for clock buffer b
τ	Temperature parameter for Gumbel-Softmax
$Load_{R/S}(u)$	Load capacitance at node u (RSMT/SPT based)
$Delay_{R/S}(u)$	Elmore delay at node u (RSMT/SPT based)
$Impulse_{R/S}(u)$	Impulse value at node u (RSMT/SPT based)
AT_{CLK}	Arrival time at the clock port (CLK)
AT_{DATA}	Arrival time at the data port (DATA)
T_period	Target clock period
T_clk2q	Flip-flop clock-to-output delay

III. Overview

Unlike previous timing-driven frameworks [5], [15] that assume zero clock skew, our approach incorporates clock skew effects to achieve more accurate timing analysis and unlock a broader optimization solution space. The key idea is to simultaneously optimize clock skew and logic delays through gradient-based optimization. Specifically, our framework optimizes both clock buffer sizes and cell positions to improve timing performance. We keep the sizes of other clock elements, e.g. integrated clock gating (ICG) cells, fixed, since their impact on clock net delay is relatively small. Also, to maintain routability and prevent excessive cell overlap, the objective function incorporates wirelength and density. The complete formulation of CCD optimization is as follows:

$$\min_{\mathcal{P},\boldsymbol{s}_\mathrm{c}} \sum_{\text{endpoint } ep} \mathrm{TM}_{ep}(\mathcal{P},\boldsymbol{s}_\mathrm{c}) + \sum_{\text{net } e} \mathrm{WL}_e(\mathcal{P},\boldsymbol{s}_\mathrm{c}) + \lambda \mathrm{Density}(\mathcal{P},\boldsymbol{s}_\mathrm{c}). \tag{1}$$

The timing objective TM can be further formulated as:

$$\mathrm{TM}_{ep} = -t_1 \mathrm{WNS}_{ep} - t_2 \mathrm{TNS}_{ep}, \tag{2}$$

where $\mathcal{P}$ represents coordinates of instances, $\boldsymbol{s}_\mathrm{c}$ denotes clock buffer sizes, and λ, t_1, t_2 are weighting factors for different objectives. Considering that hold time violations can be relatively easily resolved by inserting delay cells like buffers, this work primarily focuses on addressing setup violations. The proposed CCD optimization is followed by DREAMPlace's [17] greedy legalization to resolve potential cell overlaps. The whole process should be regarded as skew-aware post-CTS timing optimization.

In order to optimize the objective function in Equation (1) using GPU-accelerated gradient descent, we employ differentiable models for timing, wirelength, and density. The differentiable wirelength and density models are adapted from prior work [6]. Gradients for wirelength and density are computed solely with respect to cell positions, as clock buffer sizing has a negligible effect on these objectives, more details of which are discussed in Section IV-A. Therefore, the next section focuses on detailing our novel differentiable timing model with the consideration of clock buffer sizing.

IV. Differentiable Timing Model

This section presents our well-designed differentiable timing model, detailing the forward propagation and gradient flow in backward propagation. Section IV-A introduces the key parameters and definitions, focusing on the initialization of clock buffer sizes and their logit-based parameterization. Section IV-B explains our skew-aware timing objective formulation, which incorporates RC tree construction and delay modeling to explicitly account for clock skew. Section IV-C elaborates the gradient generation process using the idea of straight-through estimator (STE), covering both an RC tree calibration mechanism and a novel method for approximating clock buffer sizing gradients. This approach ensures robust and stable convergence.

A. Parameters Definition

The optimization process begins with parameter initialization. After CTS, we obtain the cell coordinates $\mathcal{P} = \{(x_g, y_g) \mid g \in \mathcal{N}\}$ and the vector of chosen clock buffer sizes $\boldsymbol{s}_\mathrm{c} = \{s_b \mid s_b \in \mathcal{S}, b \in \mathcal{N}_c \subsetneq \mathcal{N}\}$. Here, $\mathcal{N}$ is the complete set of circuit instances, $\mathcal{N}_c$ is the set of clock buffer instances, and $\mathcal{S}$ is the set of available clock buffer sizes. Evidently, the solution space of clock buffer sizing grows exponentially with the cardinality of $\mathcal{S}$.

We propose a logit-based continuous parameterization to enable gradient-based optimization for the discrete sizing problem. Specifically, we model the clock buffer sizes using logits $\mathcal{L}_\mathrm{c} = \{\boldsymbol{l}_b \mid \boldsymbol{l}_b \in \mathbb{R}^{|\mathcal{S}|}, b \in \mathcal{N}_c\}$, initialized as follows:

$$l_b^s \overset{\text{init}}{=} \begin{cases} c, & \text{if } s = s_{b0} \\ 1, & \text{otherwise} \end{cases}, \quad s \in \mathcal{S}, b \in \mathcal{N}_c, \tag{3}$$

where s_{b0} is the initial buffer size of clock buffer b derived from CTS and c is a constant which is greater than 1, indicating a higher initial preference for the buffer size s_{b0}.

The adoption of logit-based parameterization is motivated by two fundamental properties. First, unlike probability distributions restricted to $[0, 1]$ interval, logits operate in the unconstrained real number space $\mathbb{R}$. This property eliminates the need for complex constraint handling mechanisms and facilitates gradient-based optimization. Second, compared to directly interpolating discrete buffer sizes as done in prior works [15], logits parameterization provides a more principled way to model size preferences. Instead of working directly in a continuous size space, this parameterization allows the optimizer to express relative preferences between discrete choices through smooth logit values, enabling more natural upsizing or downsizing decisions during optimization.

Gumbel-Softmax [21] is then adopted to transform logits $\mathcal{L}_\mathrm{c}$ into probabilities $\boldsymbol{\theta}$, followed by argmax to determine the actual sizes $\boldsymbol{s}_\mathrm{c}$. For each clock buffer instance $b \in \mathcal{N}_c$, the formulation is shown as follows:

$$\theta_b^s = \frac{\exp((l_b^s + \mathrm{gumbel}(u^s))/\tau)}{\sum_{i \in \mathcal{S}} \exp((l_b^i + \mathrm{gumbel}(u^i))/\tau)}, \quad s \in \mathcal{S}, \tag{4}$$

$$s_b = \underset{s \in \mathcal{S}}{\mathrm{argmax}}(\theta_b^s), \quad b \in \mathcal{N}_c, \tag{5}$$

where the $\{u^i | i \in \mathcal{S}\}$ are i.i.d samples drawn from a uniform distribution over $[0, 1]$ and $\mathrm{gumbel}(u^i) = -\log(-\log(u^i))$ is the corresponding Gumbel noise. τ is the temperature parameter for Gumbel-Softmax which controls the sharpness of the probability distribution. The introduction of Gumbel-Softmax instead of standard Softmax is crucial for useful skew optimization. Our empirical study reveals that optimal buffer configurations often differ substantially from the initial solutions provided by CTS, suggesting the necessity of extensive solution space exploration in the early stage. In high-dimensional discrete optimization, the randomization effect of Gumbel noise effectively prevents premature convergence to suboptimal solutions by encouraging exploration of different buffer configurations. Moreover, the temperature parameter τ is used to control

the trade-off between exploration and exploitation. By gradually decreasing τ during optimization, we initially sample from a smooth distribution over buffer configurations to encourage broad exploration, and progressively sharpen the distribution to favor high-probability configurations such that the optimization converges. In each gradient descent iteration, the temperature parameter τ updates as follows:

$$\tau := \tau \cdot (1 - \epsilon), \tag{6}$$

where $\epsilon \in [0, 1]$ is a decay hyperparameter.

Based on the definitions above, the optimization problem can be reformulated as follows:

$$\min_{\mathcal{P}, \mathcal{L}_c / \boldsymbol{s}_c} \sum_{\text{endpoint } ep} \text{TM}_{ep}(\mathcal{P}, \mathcal{L}_c) + \sum_{\text{net } e} \text{WL}_e(\mathcal{P}, \boldsymbol{s}_c) + \lambda \text{Density}(\mathcal{P}, \boldsymbol{s}_c). \tag{7}$$

Note that clock buffer sizing exhibits negligible impact on density and wirelength. Thus, to reduce computational overhead, we omit the calculation of sizing gradients for these objectives. Instead, parameters such as buffer areas and pin offsets are updated directly according to $\boldsymbol{s}_c$ derived from Equation (5) within the density and wirelength calculations. This strategy maintains accurate objective evaluation while avoiding the complex differentiation of density and wirelength with respect to clock buffer sizes.

B. Skew-aware Timing Objective

Our timing analysis framework explicitly models clock net delays and data path propagation, enabling comprehensive timing optimization. The framework begins with accurate parasitic modeling through routing tree construction. We employ FLUTE [22] to generate Rectilinear Steiner Minimal Tree (RSMT) during forward propagation, which provides a realistic approximation of the eventual routing topology. The structure of RSMT is illustrated in Fig. 3. The parasitic effects are then characterized using the Elmore model, capturing the first-order moment of impulse response in RC networks:

$$\begin{aligned}
Load_R(u) &= cap_u + \sum_{v \in children(u)} Load_R(v), \\
Delay_R(u) &= Delay_R(fa(u)) + res_{fa(u)\to u} \cdot Load_R(u), \\
LDelay_R(u) &= cap_u \cdot Delay_R(u) + \sum_{v \in children(u)} LDelay_R(v), \\
Beta_R(u) &= Beta_R(fa(u)) + res_{fa(u)\to u} \cdot LDelay_R(u), \\
Impulse^2_R(u) &= 2 \cdot Beta_R(u) - Delay^2_R(u),
\end{aligned} \tag{8}$$

where R denotes RSMT, *cap* represents pin capacitance (from liberty files, *.lib*), and *res* represents wire resistance (from extracted parasitic parameters). *LDelay* and *Beta* serve as intermediate computational values. $fa(u)$ is the parent node of node u. This computation is typically implemented through four dynamic programming passes on the tree structure, alternating between bottom-up and top-down traversals [5]. Signal transition times (slews) are then propagated through the circuit topology according to the signal flow, as shown in Fig. 4(a). For an edge from node $u \to v$, the formulation is as follows:

$$Slew^2(v) = Slew^2(u) + Impulse^2(v). \tag{9}$$

Since $Impulse^2$ is directly used in slew propagation, we don't need to calculate the value of *Impulse* in practice.

Following slew propagation, we proceed to construct a timing graph that models the complete circuit timing behavior. The graph incorporates combinational arcs representing signal gate or net delays and clock arcs capturing clock net delays. As shown in Fig. 4(b), we also model sequential timing behavior by establishing *CLK*-to-*OUTPUT* timing arcs with delay T_{clk2q} from each flip-flop's *CLK* port to its *OUTPUT* port. Signal arrival times are then systematically propagated through this graph following topological ordering. To maintain differentiability throughout the timing analysis, we employ the Log-Sum-Exp (LSE) function as a smooth approximation for the inherently discontinuous $\max$ operations in arrival time propagation. For timing verification at each endpoint, typically a flip-flop *DATA* port, we evaluate setup constraints by comparing the arrival time at the endpoint's input data pin with the arrival time at its related clock pin: In this way, gradients can be naturally propagated to the clock net during back propagation, enabling adjustment of clock skew. where AT_{DATA} is the data net arrival time at the flip-flop's *DATA* port and $AT_{related\text{-}CLK}$ is the related clock signal arrival time at the flip-flop's *CLK* port. T_{period} is the target clock period specified in *.sdc* timing constraints. The setup constraint $Slack \geq 0$ ensures that data arrives prior to the required time at the receiving flip-flop. Through the *CLK*-to-*OUTPUT* timing arcs and timing propagation, both AT_{DATA} and $AT_{related\text{-}CLK}$ have gradient paths back to the clock net. This enables simultaneous optimization of logic path delays and clock skew during backpropagation, allowing our framework to identify and exploit useful clock skew for timing closure.

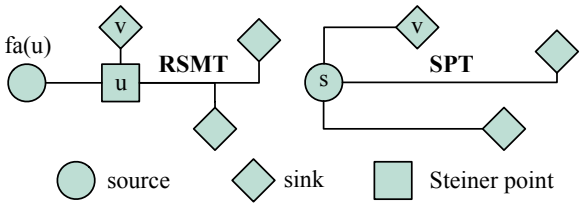


Fig. 3 Illustration of RSMT and SPT structures.

C. Gradients Generation with STE

Differentiable RC Tree with Calibration. As discussed in Section II-B, the inherent discontinuity of RSMT construction poses a critical challenge for gradient-based optimization approaches. To overcome this limitation, we introduce a novel adaptation of the STE methodology, using RSMT for accurate forward computation while enabling smooth gradient propagation via SPT.

The key advantage of SPT lies in its continuous and differentiable construction process since the source pins and sink pins are directly connected without any Steiner point, which is illustrated in Fig. 3. This enables the backward propagation of gradients from the timing objective through the Elmore model outputs, i.e., net delay, impulse, and downstream capacitance load, ultimately to both clock buffer sizes and cell positions, as shown in Fig. 4(c). We can formulate the SPT-based Elmore model through the following set of equations:

$$\begin{aligned}
Load_S(s) &= \sum_{v \in sinks} Load_S(v), \\
Delay_S(v) &= res_{s\to v} \cdot Load_S(v), \\
LDelay_S(v) &= cap_v \cdot Delay_S(v), \\
Beta_S(v) &= res_{s\to v} \cdot LDelay_S(v), \\
Impulse^2_S(v) &= 2 \cdot Beta_S(v) - Delay^2_S(v),
\end{aligned} \tag{10}$$

where S denotes SPT, s is the driving source pin and *sinks* is the set of sink pins in the net.

While SPT enables smooth gradient propagation, its structural simplicity relative to RSMT can cause discrepancies in gradient directions. To bridge this gap, we propose a calibration mechanism that introduces scaling coefficients between SPT and RSMT computations. Although the relationship between SPT and RSMT Elmore

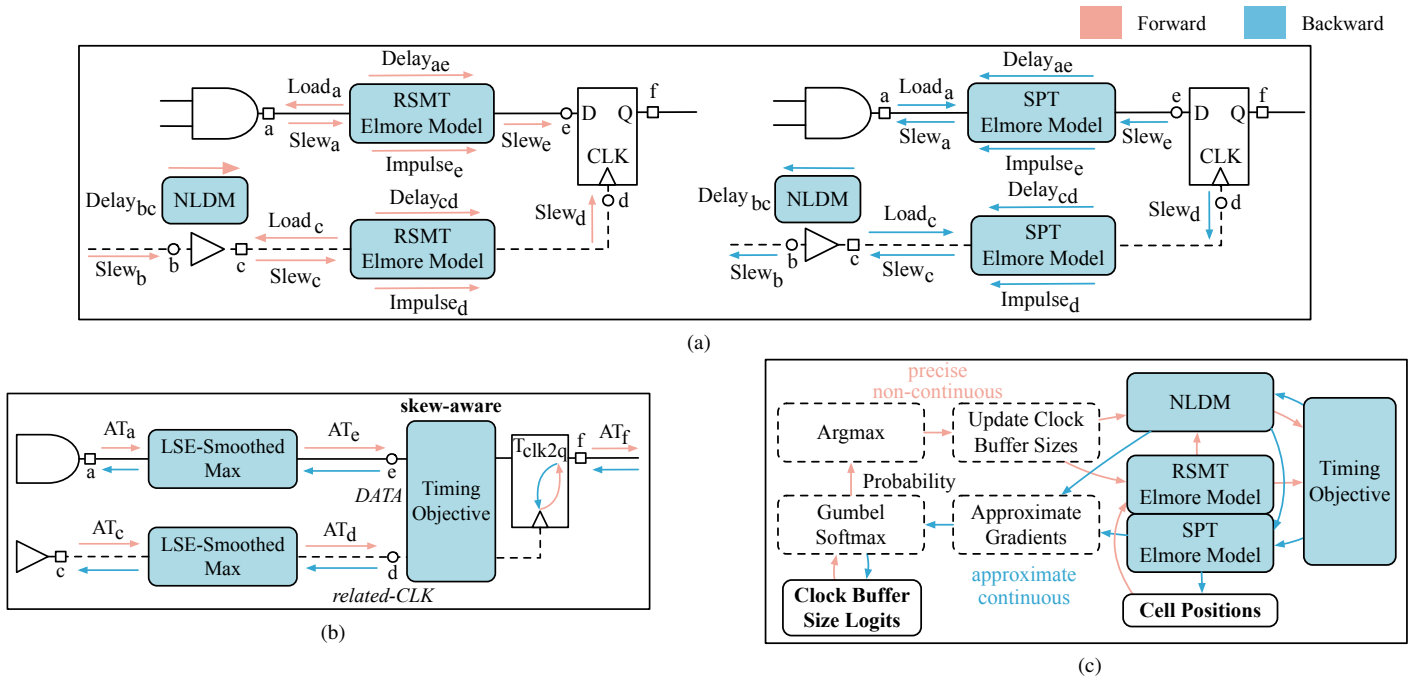


Fig. 4 (a) Forward/Backward propagation of Elmore model computation and slew propagation. (b) Forward/Backward propagation of arrival time propagation with clock skew. (c) The overall gradient flow of CCD timing model with respect to clock buffer size logits and cell positions. a, b, .. are input or output pins of instances.

model outputs may not be strictly proportional, our empirical studies suggest that using multiplicative coefficients provides an adequate approximation for correcting the gradient direction. Specifically, these coefficients, namely α, β and γ, are dynamically updated at each iteration following:

$$\begin{aligned} Load_R(s) &\simeq \alpha Load_S(s), \\ Delay_R(v) &\simeq \beta Delay_S(v), \\ Impulse_R^2(v) &\simeq \gamma Impulse_S^2(v). \end{aligned} \tag{11}$$

The overhead introduced by this calibration primarily consists of calculating the SPT-based Elmore model values using Equation (10) and performing the divisions to obtain the coefficients α, β, γ. Notably, the SPT computations do not require explicit tree construction and can be performed efficiently via direct calculations between the source and each sink pin, resulting in minimal overall runtime overhead.

The corresponding gradient computations with these calibration coefficients are then derived as follows:

$$\begin{aligned} \nabla Load_R(s) &\simeq \alpha \sum_{v \in sinks} \nabla Load_S(v), \\ \nabla Delay_R(v) &\simeq \beta(\nabla res_{s\to v} \cdot Load_S(v) + res_{s\to v} \cdot \nabla Load_S(v)), \\ \nabla Impulse_R^2(v) &\simeq 2\gamma \cdot (\nabla Beta_S(v) - Delay_S(v) \cdot \nabla Delay_S(v)), \\ \nabla LDelay_S(v) &= \nabla cap(v) \cdot Delay_S(v) + cap(v) \cdot \nabla Delay_S(v), \\ \nabla Beta_S(v) &= \nabla res_{s\to v} \cdot LDelay_S(v) + res_{s\to v} \cdot \nabla LDelay_S(v). \end{aligned} \tag{12}$$

Our approach differs from existing calibration strategies in several key aspects, particularly compared to the additive bias method introduced in [23]. The fundamental distinction lies in how these methods handle gradient calibration during optimization. The previous additive bias approach employs an offset term in the form of $Delay_R = Delay_S + \delta$, which can only affect the gradient relationship between timing objectives and Elmore model outputs, as evidenced by $\nabla_{Delay_R}\text{TM} \neq \nabla_{Delay_S}\text{TM}$ but $\nabla Delay_R = \nabla Delay_S$. This limited modification constrains the flexibility to adjust the underlying gradients $\nabla Delay_S$, restricting its effectiveness for our gradient calibration approach to consider structural differences between RSMT and SPT. Instead, our multiplicative coefficient method introduces a more comprehensive gradient adjustment, indicating that $\nabla_{Delay_R}\text{TM} \neq \nabla_{Delay_S}\text{TM}$ and $\nabla Delay_R \neq \nabla Delay_S$, which affects multiple steps in the backward propagation chain. By maintaining accurate timing analysis using RSMT in the forward pass and performing calibration at each iteration, our method ensures the accuracy of the objective evaluation and guides the optimization direction.

Approximate Clock Buffer Sizing Gradients. As illustrated in Fig. 2, the cell sizes impact both Elmore model and NLDM calculations for timing analysis. After determining each clock buffer size following Equation (5), we can update the corresponding parameters for both RSMT-based Elmore model and NLDM in forward propagation. Specifically, clock buffer sizes determine the input pin capacitance, which affects the outputs of the Elmore model. Note that we disregard the effects of clock buffer sizes on pin offsets during gradient computation following prior work [15], as these effects minimally impact the timing. Pin offsets are subsequently updated for timing calculation after each clock buffer size modification to maintain computational accuracy. For NLDM, different clock buffer sizes correspond to different LUTs, which are used to compute cell delay and output slew. The mathematical formulations are as follows:

$$\begin{aligned} cap_i &= cap_i^{s_b}, \\ Delay_{i\to o} &= LUT_{s_b, i\to o} \end{aligned}, \quad arc(i,o) \in \text{buffer } b, \tag{13}$$

where s_b is the clock buffer size, and i and o denote the input and output pins of buffer b respectively. The delay value $LUT_{s_b,i\to o}$ is formally expressed as $LUT_{s_b}(Slew_i, Load_o)$, representing the cell delay retrieved from NLDM lookup tables of s_b indexed by input slew

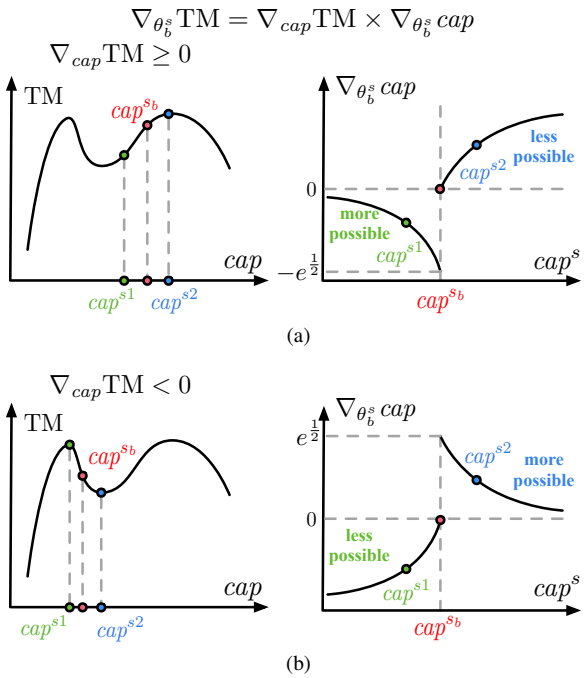


Fig. 5 The function curves of approximate clock buffer sizing gradient calculation. Here the variable *cap* is taken for illustration, where (a) $\nabla_{cap}\mathrm{TM} \geq 0$ and (b) $\nabla_{cap}\mathrm{TM} < 0$.

$Slew_i$ and output capacitive load $Load_o$. To simplify the discussion, we demonstrate our approach using pin capacitance (*cap*) and cell delay (*Delay*) as representative variables in this subsection. The handling of output transition time (*Slew*) follows a similar process. We then use *LUT* to refer to the delay lookup table throughout this paper for notational simplicity.

A fundamental challenge in this optimization is that the Elmore model and NLDM outputs are not differentiable with respect to clock buffer sizes. This non-differentiability stems from the discrete mapping relationship between cell sizes and their corresponding parameters like pin capacitance and lookup table selection. *Therefore, we must introduce an approach to approximate the gradients, thus enabling gradient-based optimization*. Here is the discussion:

(An Unsuccessful Exploration). Given that we model the probability distribution of different clock buffer sizes, a natural approach would be to approximate this process using mathematical expectation, as shown in the following equations:

$$\begin{aligned} cap \simeq \sum_{s\in\mathbb{S}} \theta_b^s cap^s, \quad Delay \simeq \sum_{s\in\mathbb{S}} \theta_b^s LUT_s, \\ \nabla_{\theta_b^s} cap \simeq cap^s, \quad \nabla_{\theta_b^s} Delay \simeq LUT_s. \end{aligned} \tag{14}$$

For brevity, we omit the input and output pin indices i and o here. While the expectation-based approach appears mathematically sound, it suffers from a practical limitation. During optimization, the gradient descent algorithm inherently favors directions that maximize objective improvement, leading to unstable buffer size selection. For example, when the gradient suggests increasing input pin capacitance, the optimization immediately moves toward buffer configurations with maximum capacitance values, since $\nabla_{\theta_b^{s2}} cap > \nabla_{\theta_b^{s1}} cap$ iff $cap^{s2} > cap^{s1}$. These sudden transitions between buffer sizes severely impair the optimization's convergence behavior.

To avoid this problem, we then propose a novel gradient approximation approach that enables smooth and controlled optimization. We manually define two fundamental functions $\Psi_y(x)$ and $-\Psi_y(2y-x)$ and the gradient computations are formulated in Equation (15).

$$\begin{aligned} \Psi_y(x) &= \begin{cases} e^{\frac{2\cdot y-x}{2\cdot y}-\frac{(x-y)^2}{\sigma\cdot y}}, & x \geq y, \\ e^{\frac{x}{2\cdot y}-\frac{(x-y)^2}{\sigma\cdot y}} - e^{\frac{1}{2}}, & \text{otherwise}, \end{cases} \\ \nabla_{\theta_b^s} cap &\simeq \begin{cases} \Psi_{cap^{s_b}}(cap^s), & \nabla_{cap}\mathrm{TM} \geq 0, \\ -\Psi_{cap^{s_b}}(2cap^{s_b} - cap^s), & \text{otherwise}, \end{cases} \\ \nabla_{\theta_b^s} Delay &\simeq \begin{cases} \Psi_{LUT_{s_b}}(LUT_s), & \nabla_{Delay}\mathrm{TM} \geq 0, \\ -\Psi_{LUT_{s_b}}(2LUT_{s_b} - LUT_s), & \text{otherwise}, \end{cases} \end{aligned} \tag{15}$$

where σ is a hyperparameter which controls the exponential decay.

As shown in Fig. 5(a), when $\nabla_{cap}\mathrm{TM} \geq 0$, indicating that reduced capacitance improves timing, the approximation $\Psi_{cap^{s_b}}(cap^s)$ yields larger gradient magnitudes for size options s where cap^s is slightly smaller than the current cap^{s_b}, encouraging incremental downsizing, while assigning smaller or negative gradients to options with significantly smaller or larger capacitances. This mechanism guides the optimization toward smaller capacitance values, thereby improving timing performance. As our formulation introduces exponential terms, the gradient magnitude reaches its maximum when the target capacitance is slightly smaller than the current value, and gradually approaches zero as the target capacitance becomes significantly smaller. This characteristic ensures that buffer sizes change incrementally rather than abruptly, preventing sudden large transitions in the optimization process. A similar analysis applies when $\nabla_{cap}\mathrm{TM} < 0$, indicating increased capacitance is needed, where the function $-\Psi_{cap^{s_b}}(2cap^{s_b} - cap^s)$ guides the optimization towards larger capacitance values in a controlled manner. While this formulation is empirically motivated rather than derived from physical principles, comprehensive experiments demonstrate its effectiveness in achieving stable convergence and high-quality timing results while maintaining controlled size transitions throughout the optimization process.

V. ADMM-Based CCD Optimization

As shown in Section III, CCD optimization is a multi-objective optimization task, which is inherently complex, particularly when the objectives are conflicting. In our case, the three primary objectives, including timing, wirelength and density, often generate opposing gradients, potentially impeding convergence. We then leverage the alternating direction method of multipliers (ADMM) to alleviate this problem, which decomposes the original problem into several manageable subproblems. Let $\mathcal{Q} = \mathcal{P}$, the augmented Lagrangian function of Equation (7) is formulated as:

$$\begin{aligned} \mathcal{L}(\mathcal{P}, \mathcal{Q}, \mathcal{L}_c, \mathcal{U}) = \sum \mathrm{TM}_{ep}(\mathcal{P}, \mathcal{L}_c) + \sum \mathrm{WL}_e(\mathcal{Q}, \boldsymbol{s}_c) + \\ \lambda \mathrm{Density}(\mathcal{Q}, \boldsymbol{s}_c) + \langle \mathcal{U}, \mathcal{P} - \mathcal{Q} \rangle + \frac{\rho}{2} \|\mathcal{P} - \mathcal{Q}\|_2^2, \end{aligned} \tag{16}$$

where ep and e denote each endpoint and net, respectively, following Equation (1). These indices are omitted from the summation symbols for brevity. $\mathcal{U}$ is the dual variable for the equality constraint $\mathcal{Q} = \mathcal{P}$, ρ is a penalty factor. In simple terms, ADMM updates are performed by alternatively minimizing the augmented Lagrangian function $\mathcal{L}(\mathcal{P}, \mathcal{Q}, \mathcal{L}_c, \mathcal{U})$ over the primal variables $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{L}_c$ and the dual variable $\mathcal{U}$ at each iteration, as detailed in Algorithm 1. Note that the updates of $\mathcal{P}$ and $\mathcal{Q}$ are based on the timing-driven gradient descent method proposed in [5].

The ADMM-based decomposition enables effective optimization by separating the timing, wirelength, and density objectives into subproblems. Each subproblem can be solved independently while

Algorithm 1 ADMM-based CCD Optimization

Input: Initial clock buffer size logits $\mathcal{L}_c^0$ and positions $\mathcal{P}^0$;
Output: Optimized clock buffer sizes and instance positions;
$k \leftarrow 0$
while not converged **do**
  Update instance positions $\mathcal{P}^{k+1}$; ▷ Equation (17)
  Update clock buffer size logits $\mathcal{L}_c^{k+1}$; ▷ Equation (18)
  Update $\boldsymbol{s}_c^{k+1}$ based on $\mathcal{L}_c^{k+1}$; ▷ Equations (4) and (5)
  Update proxy instance positions $\mathcal{Q}^{k+1}$; ▷ Equation (19)
  Update dual variables $\mathcal{U}^{k+1}$; ▷ Equation (20)
  $k \leftarrow k+1$;
end while
return Final $\boldsymbol{s}_c$ and $\mathcal{P}$;

$$\mathcal{P}^{k+1} = \underset{\mathcal{P}}{\operatorname{argmin}} \sum \mathrm{TM}_{ep}(\mathcal{P}, \mathcal{L}_c^k) + \langle \mathcal{U}^k, \mathcal{P} - \mathcal{Q}^k \rangle + \frac{\rho}{2} \left\| \mathcal{P} - \mathcal{Q}^k \right\|_2^2, \tag{17}$$

$$\mathcal{L}_c^{k+1} = \underset{\mathcal{L}_c}{\operatorname{argmin}} \sum \mathrm{TM}_{ep}(\mathcal{P}^{k+1}, \mathcal{L}_c), \tag{18}$$

$$\mathcal{Q}^{k+1} = \underset{\mathcal{Q}}{\operatorname{argmin}} \sum \mathrm{WL}_e(\mathcal{Q}, \boldsymbol{s}_c^{k+1}) + \lambda \mathrm{Density}(\mathcal{Q}, \boldsymbol{s}_c^{k+1}) + \langle \mathcal{U}^k, \mathcal{P}^{k+1} - \mathcal{Q} \rangle + \frac{\rho}{2} \left\| \mathcal{P}^{k+1} - \mathcal{Q} \right\|_2^2, \tag{19}$$

$$\mathcal{U}^{k+1} = \mathcal{U}^k + \rho(\mathcal{P}^{k+1} - \mathcal{Q}^{k+1}). \tag{20}$$

maintaining coordination through the dual variable $\mathcal{U}$ and penalty factor ρ, thus facilitating the search for solutions that balance these competing objectives. As discussed in Section IV-A, since clock buffer sizes only contribute as parameters to wirelength and density computations, we exclude these terms from the optimization objective in Equation (18). Also, building upon the observation that post-placement cell positions typically provide sufficiently good solutions, the optimization constrains cell positions $\mathcal{P}$ and $\mathcal{Q}$ to local adjustments around their initial values, preserving solution quality while enabling necessary refinements.

$$||\mathcal{P}^k - \mathcal{P}^0||_2 \leq \delta, \quad ||\mathcal{Q}^k - \mathcal{Q}^0||_2 \leq \delta, \tag{21}$$

where δ is a hyperparameter that controls the maximum allowed change in cell positions.

The gradient descent approach employed in our problem incurs significant computational costs, typically needing several hundred iterations for convergence. To mitigate this, the DiffCCD framework is implemented entirely on GPUs, capitalizing on their parallel processing power for acceleration. For enhanced performance, core computations, including the forward propagation for objective function evaluation and backward propagation for gradient generation, are executed using dedicated CUDA kernels.

VI. Experimental Results

A. Experimental Setup

Our proposed DiffCCD framework is implemented in C++ with CUDA acceleration, and the experimental evaluation is performed on a Linux server equipped with a 2.60GHz Intel Xeon CPU and an NVIDIA RTX A800 GPU. Algorithmic performance is evaluated using industrial designs from CircuitNet [24], synthesized using commercial 28nm or 14nm process design kits (PDKs). These benchmarks were selected to represent a diverse range of design complexities. Detailed benchmark statistics are provided in TABLE II. We set the hyperparameters t_1 and t_2 to 0.1 and 0.001, respectively. To facilitate early-stage exploration, the temperature parameter τ is initialized at 3000 and iteratively decreased by a factor of $\epsilon = 0.02$. σ is set to 2000. The initial logit value c is set to 2.0. The Lagrangian penalty factor ρ is set to approximately 1.0.

TABLE II Benchmark statistics.

Benchmark	Tech.	Period	#Cells	#Nets	#Pins	#Macros
VORTEX	14nm	2ns	112478	121142	430699	43
LARGEBOOM	14nm	2ns	787298	805963	3027324	636
OPENC910	14nm	2ns	798166	811653	3149667	32
PULPINO	28nm	2ns	20374	21248	75079	3
RISCY-a	28nm	2ns	56580	57168	220791	3
RISCY-FPU-a	28nm	2ns	78311	79253	300634	3
zero-riscy-a	28nm	2ns	46206	45972	176171	3

To validate the proposed algorithm's effectiveness, we established a test flow based on the widely used open-source EDA flow, OpenROAD [25]. The flow commences with timing-driven global placement to achieve initial timing optimization. Subsequently, OpenROAD's *repair_design* command is employed for pre-CTS timing optimization. This step utilizes strategic buffer insertion and gate sizing to mitigate maximum slew, capacitance, and fanout violations, while also optimizing RC delay on long interconnects to normalize signal transition times. Following the reduction of timing violations, OpenROAD's *detailed_placement* command is executed for detailed placement and instance locations legalization. The *clock_tree_synthesis* (CTS) command is then applied to construct the clock networks. During CTS, we carefully tune the parameters to achieve balanced source-to-sink delays, thereby minimizing timing degradation between the pre-CTS and post-CTS stages. This balanced clock network ensures that subsequent timing improvements can be primarily attributed to useful-skew optimization.

In the post-CTS stage, OpenROAD's *repair_timing* command, which optimizes timing while accounting for clock skew, serves as our baseline. We then conduct comparative experiments between the baseline, our proposed DiffCCD optimization framework, and a combined flow that integrates both approaches. For CCD implementation, clock buffer candidates covering multiple drive strength options were carefully selected from the respective technology libraries: 25 buffers from the 14nm library and 10 from the 28nm library. To ensure a fair comparison, timing metrics WNS and TNS are evaluated using the open-source timer OpenSTA [26]. Additionally, we measure the computational runtime to assess the efficiency of each approach.

B. Post-CTS Timing Performance Improvements

TABLE III demonstrates that our DiffCCD framework yields significant improvements in both timing quality and computational efficiency. Compared to the initial post-CTS designs, DiffCCD achieves average reductions of 50.1% (1-0.499) in WNS and 70.2% (1-0.298) in TNS across the benchmarks. Compared to the baseline *repair_timing*, our approach yields average improvements of 22.4% (1-0.499/0.643) in WNS and 45.0% (1-0.298/0.542) in TNS.

For the PULPINO design, the baseline method exhibits an anomalously long runtime 10977s, suggesting potential instability or inefficiency in some situations. In contrast, DiffCCD demonstrates superior efficiency and maintains predictable runtime scaling relative to design complexity. This high efficiency stems from our CUDA-accelerated implementation, enabling efficient parallel processing and resulting in an average 9.434 (1/0.106)× runtime reduction compared to the baseline. This consistent performance advantage underscores the value of our approach for modern, large-scale circuit designs where optimization runtime is often a critical constraint.

TABLE III WNS, TNS and runtime comparisons. The average ratio represents the mean absolute ratio across all benchmarks, with lower values indicating superior performance for all metrics.

Benchmark	OpenROAD CTS [25]		OpenROAD CTS + *repair_timing* [25]			OpenROAD CTS [25] + Our DiffCCD		
	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	Runtime (s)	WNS (ns)	TNS (ns)	Runtime (s)
VORTEX	**-1.034**	-3324.667	-1.529	-3354.724	1221	-1.099	**-1709.71**	**261**
LARGEBOOM	-0.617	-881.285	-0.764	-1354.435	4870	**-0.581**	**-664.672**	**805**
OPENC910	-0.917	-331.766	**-0.498**	-268.719	4802	-0.504	**-152.962**	**1101**
PULPINO	-0.641	-22.708	-0.049	-0.945	10977	**0**	**0**	**133**
RISCY-a	-0.559	-554.164	-0.128	**-4.59**	2187	**-0.097**	-6.424	**264**
RISCY-FPU-a	-1.097	-2515.743	-0.493	-853.758	5443	**-0.475**	**-830.637**	**333**
zero-riscy-a	-0.339	-46.185	-0.165	-2.153	**206**	**-0.114**	**-0.567**	239
Average Ratio	1.000	1.000	0.643	0.542	1.000	**0.499**	**0.298**	**0.106**

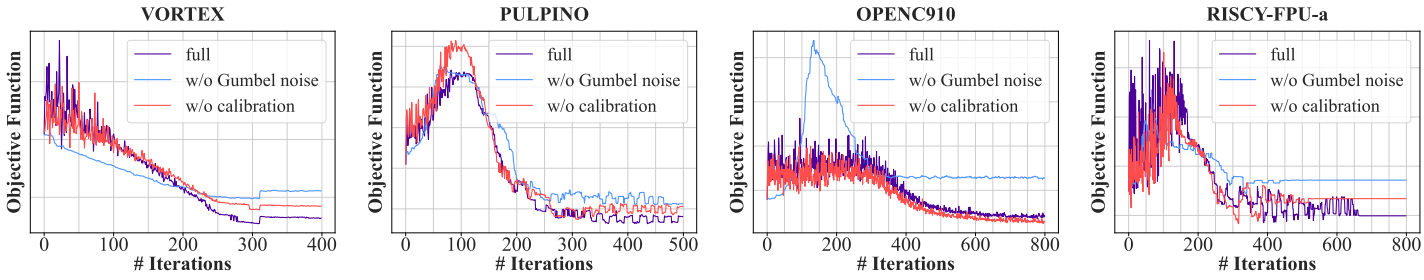


Fig. 6 Loss curves of Equation (18) optimization under three configurations, where full method is our complete CCD optimization framework with both Gumbel noise and calibration mechanism.

TABLE IV Comparison between our DiffCCD and combined flow, i.e. OpenROAD *repair_timing* [25] followed by DiffCCD.

Benchmark	Our DiffCCD		Combined Flow	
	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)
VORTEX	**-1.099**	-1709.71	-1.400	**-1643.371**
LARGEBOOM	**-0.581**	**-664.672**	-0.601	-723.397
OPENC910	-0.504	**-152.962**	**-0.502**	-205.938
PULPINO	**0**	**0**	-0.006	-0.006
RISCY-a	-0.097	-6.424	**0**	**0**
RISCY-FPU-a	**-0.475**	**-830.637**	**-0.17**	**-102.216**
zero-riscy-a	**-0.114**	**-0.567**	**-0.064**	**-0.219**
Average	-0.410	-480.710	**-0.391**	**-382.164**

It is worth noting that for certain benchmarks like VORTEX, while TNS shows significant improvement, WNS may degrade slightly. We believe this could potentially be attributed to slight discrepancies between the differentiable timer used during optimization, which indicates consistent improvements in WNS, and the final evaluation timer, OpenSTA, or to the inherent trade-off in the loss function aiming to balance both WNS and TNS improvements.

Furthermore, TABLE IV shows that for several benchmarks, e.g., RISCY-a, RISCY-FPU-a, an integrated flow combining *repair_timing* and DiffCCD achieves superior timing metrics compared to either method applied individually. These complementary improvements likely stem from the distinct optimization targets of each method. While *repair_timing* primarily focuses on data path optimization via gate sizing, our DiffCCD framework specifically targets clock network optimization through strategic clock buffer sizing. This suggests that integrating DiffCCD into existing timing optimization flows may yield comprehensive timing improvements.

C. Ablation Study

To validate the effectiveness of the components within our proposed approach, we conducted ablation studies on four representative benchmarks. Fig. 6 illustrates the loss curves corresponding to the optimization of Equation (18) under three configurations: (1) without Gumbel noise, (2) without the calibration mechanism, and (3) our complete method incorporating both components. The results demonstrate that introducing Gumbel noise significantly influences the early-stage optimization dynamics. Although these perturbations induce objective function fluctuations in the early stage, they consistently lead to superior final timing results across all test cases, thereby validating the importance of exploration during the early optimization stages. Furthermore, despite this initial volatility, all configurations exhibit stable convergence, highlighting the overall robustness of our algorithm. The impact of the calibration mechanism, while beneficial, appears less pronounced than that of Gumbel noise. This might be attributed to the inherent structural similarity between the SPT and RSMT models. Empirically, combining both Gumbel noise and the calibration mechanism yields the best performance in most test cases.

VII. Conclusion

This paper presents a differentiable concurrent clock and data optimization framework to achieve useful skew optimization in post-CTS timing closure. Specifically, we propose a well-designed gradient-based approach that optimizes clock buffer sizes while simultaneously refining placement solutions to enhance timing performance. Experimental results on open-source industrial benchmarks demonstrate that our CCD optimization framework achieves superior timing closure compared to a baseline post-CTS timing optimization approach, achieving an average reduction of 22.4% in WNS and 45.0% in TNS, along with a significant runtime speedup of 9.434×. These results validate the effectiveness of our framework as a practical solution for modern timing closure challenges. We believe this work provides a new perspective on applying differentiable optimization to discrete EDA problems, paving the way for future research in this direction.

Acknowledgements

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14211824).

REFERENCES

[1] V. Nawale and T. W. Chen, "Optimal useful clock skew scheduling in the presence of variations using robust ILP formulations," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2006, p. 27–32.

[2] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, and A. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 39, no. 11, pp. 799–814, 1992.

[3] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and Steiner routing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 3, pp. 341–388, 1998.

[4] K. Han, A. B. Kahng, and J. Li, "Optimal generalized h-tree topology and buffering for high-performance and low-power clock distribution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 2, pp. 478–491, 2018.

[5] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *DAC*. ACM, 2022, pp. 1315–1320.

[6] J. Lu, P. Chen, C. Chang, L. Sha, D. J. Huang, C. Teng, and C. Cheng, "ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 2, pp. 17:1–17:34, 2015.

[7] K. Wang, L. Duan, and X. Cheng, "ExtensiveSlackBalance: an approach to make front-end tools aware of clock skew scheduling," in *ACM/IEEE Design Automation Conference (DAC)*, 2006, pp. 951–954.

[8] T. Chan, A. B. Kahng, and J. Li, "NOLO: A no-loop, predictive useful skew methodology for improved timing in IC implementation," in *ISQED*, 2014, pp. 504–509.

[9] L. Chao and E. H. Sha, "Retiming and clock skew for synchronous systems," in *ISCAS*, 1994, pp. 283–286.

[10] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *ISCAS*, 1994, pp. 407–410.

[11] I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," in *ICCAD*, 1999, pp. 239–243.

[12] M. Saitoh, M. Azuma, and A. Takahashi, "Clustering based fast clock scheduling for light clock-tree," in *DATE*, 2001, pp. 240–245.

[13] J. G. Xi and W. W. Dai, "Useful-skew clock routing with gate sizing for low power design," in *DAC*. ACM Press, 1996, pp. 383–388.

[14] L. P. Van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1990, pp. 865–868.

[15] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of Global Placement and Gate Sizing with Differentiable Optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[16] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[17] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 4, pp. 748–761, 2021.

[18] W. Li, R. Liang, A. Agnesina, H. Yang, C.-T. Ho, A. Rajaram, and H. Ren, "DGR: Differentiable Global Router," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[19] B. Fu, L. Liu, M. D. Wong, and E. F. Young, "Hybrid Modeling and Weighting for Timing-driven Placement with Efficient Calibration," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[20] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *CoRR*, vol. abs/1308.3432, 2013.

[21] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[22] C. C. N. Chu and Y. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 1, pp. 70–83, 2008.

[23] W. Li, Y. Kukimoto, G. Servel, I. Bustany, and M. E. Dehkordi, "Calibration-Based Differentiable Timing Optimization in Non-linear Global Placement," in *ACM International Symposium on Physical Design (ISPD)*, 2024, pp. 31–39.

[24] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "CircuitNet: An Open-Source Dataset for Machine Learning in VLSI CAD Applications With Improved Domain-Specific Evaluation Metric and Learning Strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 12, pp. 5034–5047, 2023.

[25] The-OpenROAD-Project, "Openroad," GitHub repository, 2024, available online: https://github.com/The-OpenROAD-Project/OpenROAD.

[26] T. O. Project, "Parallax static timing analyzer," 2025. [Online]. Available: https://github.com/The-OpenROAD-Project/OpenSTA