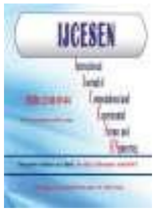
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 9088-9098
http://www.ijcesen.com

Research Article

ISSN: 2149-9144



Advanced Timing Closure Techniques in Full Chip Integration of Adaptive SoCs

Ujjwal Singh*

Cornell, Ithaca, NY
* **Corresponding Author Email:** us62@cornell.edu- **ORCID:** 0009-0009-7877-2530

Article Info:

DOI: 10.22399/ijcesen.4369
Received : 01 February 2025
Accepted : 29 March 2025

Keywords

Timing Closure;

Adaptive System-on-Chip (SoC); Static Timing Analysis (STA); Machine Learning in EDA; Multi-Mode Multi-Corner (MMMC) Analysis

Abstract:

Meeting strict timing specifications in full-chip integration is becoming increasingly challenging with the growing adoption of adaptive system-on-chip (SoC) architectures. SoCs that incorporate programmable logic, Dynamic Voltage and Frequency Scaling (DVFS), and heterogeneous compute instances will require more advanced analysis methods as they are operating within the picosecond range. Such SoCs featuring programmable logic as well as Dynamic Voltage and Frequency Scaling (DVFS), and heterogeneous compute instances will need more powerful methods to analyze than the picoseconds range. This paper explores the extended timing closure techniques explicitly applied to full-chip implementations of adaptive SoCs, including Multi-Mode Multi-Corner (MMMC) analysis, hierarchical abstraction, and machine learning-aided path optimization. The issues of the Incremental Design Verification (IDV), Clock Domain Crossings (CDCs), and Advanced Formal Signoff (AFS) are given special concerns. Real-time design feedback is integrated with the capability of AI-based timing anomaly detection. It also highlights the application of physical-aware timing ECOs (Engineering Change Orders) and accumulated P&R flows as an example of improved closure efficacy. By using comprehensive case studies and empirical measurements, it shows that the provisional tools and techniques facilitate significant improvement of timing convergence, accuracy, performance predictability, and preparation of post-silicon validation. The findings are indicative of scalable and adaptive timing techniques that are increasingly gaining relevance to future SoC design, where timing design closure will have to assimilate both the static and dynamic system responses.

1. Introduction

With the continued miniaturization of semiconductor processes into sub-40nm process nodes, SoC design integration and performance are hit with a major problem: How can timing closure be achieved across an increasingly complex, variable, and dynamically configurable SoC architecture? Adaptive SoCs possess a large amount of programmable logic blocks, embedded processors, and configurable accelerators, and have the ability to support Dynamic Voltage and Frequency Scaling (DVFS), which is temporally unpredictable. This complexity makes it very necessary to have another type of timing closure techniques that not only rely on the usual static analysis, but also cope with the very nature of these platforms' dynamics [1, 2].
Functional behavior and performance goals, power budgets, and manufacturability targets are difficult to meet unless robust timing closure is achieved in modern SoCs. The paper addresses and reports on a collection of high-end timing closure methods optimized to perform well in the case of adaptive SoC architecture. These methods are the Multi-Mode Multi-Corner (MMMC) analysis, hierarchical timing schemes, AI-enhanced path identification and associated optimization, and clock domain crossing management. The exploration extends to physically sensitive Engineering Change Orders (ECOs) and sophisticated signoff capability, which incorporates a feedback loop and anomaly detection [3, 4]. The major aim of this paper is to narrow the gap that exists between theoretical concepts of timing closure and its applicability to adaptive SoC designs. It attempts to solve the timing variability presented by heterogeneous workloads, and by runtime reconfiguration and voltage-frequency scaling. Consequently, the scope of the work extends to foundations and timing analysis,

contemporary closure techniques, and the application of AI in the design of physical design flows, concluding with case studies and future directions in the field.

2. Overview of Adaptive SoC Architectures

Understanding the nuances of adaptive SoC architectures is essential to appreciating the challenges in timing closure. These architectures represent an evolution from fixed-function SoCs to dynamic, reconfigurable platforms that integrate general-purpose cores, domain-specific accelerators, FPGAs, and AI engines. The growing demand for application-specific flexibility in domains such as AI inference, edge computing, and autonomous systems has accelerated the adoption of adaptive designs [5, 6].

Unlike conventional SoCs, where timing constraints are pre-determined and mostly fixed, adaptive SoCs are dynamically controlled. DVFS schemes adjust operating voltages and clock frequencies are dynamically changed in an attempt to achieve the best power-performance trade-offs at rRuntime. In addition to that, adaptive systems also tend to have multiple clock domains, programmable logic fabrics, and even chiplet-based designs that all add to timing unpredictability. Such a high level of configurability and parallelism results in an exponential growth of timing scenarios. Therefore, numerous mode-corner combinations, clock domain crossings, and logic reconfigurations must be considered during timing analysis by a designer. Adaptive SoCs also complicate traditional verification flows because they have more path permutations in terms of timing and greater physical design interconnects [7]. The environment of adaptive SoCs consequently creates new requirements to timing closure approaches-requirements that extend over worst-case static analysis and require dynamic, hierarchical, and machine-intelligent methodologies to characterize the true behavior of the computing system.

3. Timing Closure Fundamentals

To fully understand the developments proposed in this paper, it is essential to first establish the foundational principles upon which timing closure is based. To make sense of the developments suggested in the present paper, it is important to ground it in the underlying principles on which timing closure is based. On a fundamental level, timing closure ensures that all signal paths within a work design meet hold-time and setup-time demands all through the surrounding conditions of operationing. Static Timing Analysis STA is the older methodology of analyzing the timing paths, assessing the delay of combinational logic in between flip-flops without requiring simulation [8, 9].

Since adaptive SoCs require more than one operating point, MMMC analysis has become a requirement. MMMC does not assume a single operating voltage or temperature or a single process corner, and shows a more realistic perspective of the timing behaviour. It produces timing reports in various modes (functional, scan, test) and corners (slow, fast, typical), and thus can help a designer to understand violations that may only arise in a specific situation. Sign-off criteria in modern flows include slack analysis, timing uncertainty, clock skew management, and derating techniques that model on-chip variability. Slack-based metrics such as WNS are essential for assessing design maturity; however, their interpretation should account for hierarchical effects, multi-mode and multi-corner conditions, as well as dynamic variations in modern adaptive systems. Metrics such as Worst Negative Slack (WNS), Total Negative Slack (TNS), and the number of timing violations per mode-corner combination provide key indicators for design readiness. However, these metrics must be interpreted with awareness of design hierarchy and dynamic conditions prevalent in adaptive architectures [10, 11]. These fundamentals serve as the building blocks for more advanced strategies. As the subsequent section explores, hierarchical and incremental closure methodologies are built upon the limitations of flat STA in large-scale adaptive SoCs.

4. Advanced Timing Closure Techniques

More conventional timing closure methods (such as flat STA throughout a whole SoC) are now highly inefficient and, in some cases, impossible to perform on adaptive SoCs because of their sheer size and configurability. Hierarchical timing closure methodologies are one required answer. Hierarchical methods simplify the complexity of the analysis by modeling timing properties of IP blocks in abstract and inter-module interface constraints and propagation/timing budgeting [12, 13, 14]. Synchronization allows an effective path convergence through slack target allocation to modules or partitions in a synchronization budget allocation. These goals are successively improved on the basis of timing reports, allowing geographically distributed design groups to work in parallel. The interdependency between data-paths and physical proximity needs to be considered as partitioning strategies to reduce the inter-partition violations and timing loops [14].

Incremental timing closure enables the top-to-bottom approaches, supplementing the hierarchical approaches, which means that localized changes are not mandatory, but a reanalysis is conducted only over the entire chip. These methodologies are especially beneficial to streamline later Engineering Change Orders (ECOs), where little engineering changes have to be confirmed in a short time. Parameters such as incremental static timing analysis (iSTA), dynamic path pruning, and localized P&R tuning can be used to speedily work through design iterations and reduce time-to-signoff [15]. Together, these advanced techniques form a critical arsenal for navigating the complex timing closure landscape of adaptive SoCs. They set the stage for even more intelligent optimization strategies, as explored in the next section focused on machine learning and AI integration.

While fundamental timing metrics form the basis of timing validation, large-scale adaptive SoCs necessitate more granular and modular strategies to manage closure complexity. This requires leveraging both structured abstraction and localized optimization. To further highlight the distinctions between traditional and modern closure approaches, the following table summarizes their comparative characteristics.

The increasing reliance on modular and intelligent flows, as presented in Table 1, confirms the inadequacy of legacy timing approaches for next-generation SoCs. These methods lay the groundwork for more intelligent and predictive closure techniques, especially those driven by artificial intelligence and machine learning, which will be addressed in the next section.

5. Machine Learning and AI in Timing Closure

With the advent of AI-driven approaches to Electronic Design Automation (EDA), timing closure on adaptive SoCs has become a fundamentally redefined concept, as demonstrated in the figure below (Figure 1). Traditional methods, which do not require exhaustive verification as they are guided by deterministic rules, are inadequate in the face of the inordinate timing variation inflicted by multi-mode multi-corner (MMMC) environments, DVFS, as well as programmable logic fabrics. Data-driven optimization with predictive machine learning can be applied across all these variables, which greatly enhances the efficiency and resilience of closure [16, 17].

Predictive modeling to identify critical paths is one of the important AI uses in timing closure. Rather than repeating the analysis of each timing path to search exhaustively, models based on historical and design-specific attributes can be trained to predict which paths are likely to be/not be violated by different conditions. Such models take advantage of the path length, the distributions of cell types, fanout, and spatial density of the placement to achieve high accuracy in predicting timing hotspots. This narrows the analysis and puts designers in a position to focus resources on real areas of concern [18].

AI-based ECO (Engineering Change Order) optimization works off this predictive model by proposing minimal logic and placement changes to correct violations. As an example, reinforcement learning can be utilized to run iteratively, to determine the best possible gate-resizing or insertion of buffers to reduce slack on critical paths. These techniques even compare more favorably to manual ECO closure in that by examining each past design iteration and extrapolating changes which offer the maximum performance benefits at the smallest area cost or power cost, one can accurately determine which changes are worthwhile to apply to a design in a purely manual (which is to say, an expert-driven) ECO process [19]. The other useful application is timing anomaly detection. Non-obvious timing behaviors are a common problem with complex SoCs, as hierarchical interactions and cross-domain effects or physical layout constraints can cause this to be a non-obvious behavior. Uncertainty-aware machine-learning models for timing prediction can quantify confidence and automatically flag low-confidence paths for targeted re-analysis, reducing exhaustive reviews on large designs. In practice, the model predicts slack across corners and emits a confidence score (for example, via Bayesian regression, ensembles, or dropout-based uncertainty); paths whose predicted error bars straddle the violation threshold are queued for selective STA reruns. This triage focuses compute and engineer attention on the few path families most likely to be misclassified, while allowing high-confidence, noncritical regions to pass without repeated analysis. The loop then learns from the new STA results updating the model, shrinking uncertainty on similar structures, and steadily decreasing the volume of full rechecks over iterations. Integrated into signoff dashboards, this approach yields faster convergence, fewer false alarms, and clearer evidence for ECO decisions [20]. AI integration thus transforms timing closure into an adaptive, intelligent process aligned with the very nature of adaptive SoCs. These technologies augment traditional signoff flows with predictive analytics, resulting in shorter design cycles, better quality of results, and more deterministic convergence. As timing analysis grows increasingly dependent on clocking behavior and synchronization strategies, the next section addresses how modern SoCs tackle challenges related to clock domain crossings, voltage scaling, and power gating mechanisms.

6. Clocking and Synchronization Techniques

Effective clocking strategies are essential to ensure data consistency, timing reliability, and power efficiency in adaptive SoCs, as shown in Figure 2. These systems typically operate across multiple asynchronous or mesochronous clock domains, each potentially governed by its own voltage-frequency relationship due to DVFS. Consequently, Clock Domain Crossing (CDC) analysis becomes a cornerstone in timing closure validation [21, 22].

CDC issues arise when data transfers between domains with non-coherent clocks. If not properly managed, this can lead to metastability, setup or hold violations, and data corruption. Standard practices include synchronizer insertion, handshake protocols, and FIFO buffering. However, adaptive SoCs introduce runtime variability in clock domains, necessitating dynamic CDC verification methods [23]. Advanced static and formal CDC tools are employed to exhaustively analyze all possible domain interactions under varied timing constraints. Dynamic Voltage and Frequency Scaling adds a further layer of complexity. While DVFS enables runtime optimization of power and performance, it also causes fluctuations in path delays and timing margins. Consequently, timing analysis must address DVFS conditions not only at the synthesis stage but also throughout runtime closure iterations. To preserve timing integrity under varying voltage and frequency levels, techniques such as per-network DVFS domains, FIFO-based resynchronizers at frequency boundaries, and PLL-driven actuators are employed. Timing analysis must therefore consider DVFS corners not only during synthesis but also during dynamic closure iterations. Some of the major methods to meet these problems include voltage-aware timing signoff and adaptive clock trees with scaling buffers [24]. A similar power reduction focus is also applied in adaptive SoCs with the use of clock gating to reduce dynamic power. Poorly designed gating logic or faulty enable conditions may cause glitches or timing failure to close. Clock-gating logic must be properly examined by confirmation tools in all the active modes and power domains [25]. Increasingly, these checks are being enhanced by the ability of AI models to spot aberrant gating patterns or to propose gating opportunities on switching activity profiles. Physical design correlations in adaptive SoCs, therefore, require a multi-dimensional process spanning structural analysis, dynamic corner evaluation, and formal verification, which reiterates the requirement of a tightly integrated closure process as will be discussed later in the section on physical design correlations.

7. Physical Design and Timing Correlation

Timing closure cannot be decoupled from physical design. In fact, many late-stage violations emerge not from logical constraints, but from spatial layout-induced effects such as wire delay, congestion, and placement irregularities. Adaptive SoCs further complicate this interdependency through reconfigurable fabrics and dense heterogeneous integration, making physical-aware timing optimization indispensable [26, 27].

Dynamic placement-aware timing optimization is one of the commonly used steps in physical synthesis. Timing-driven placement iteratively searches to flatten critical paths and place congruent cells near each other, to decrease slack violations. In the case of adaptive designs, this optimization is further optimized using programmable resource awareness, where logic, which can map to configurable regions, maintains timing integrity under different configurations. Fine-grained and time-sensitive routing algorithms will supplement it by prioritizing time-sensitive nets and crosstalk-induced delay variation [28]. A long-time issue in the physical design space is the trade-off between congestion and timing closure. It is possible that dense areas can be advantageous to timing because of shorter net lengths, but at the threat of routing congestion, thermal hotspots, and signal integrity degradation. On the other hand, applying spreading logic has the effect of reducing routability, although it might increase delay. Adaptive closure flows to SoC have to trade-offs between these factors, and many use machine-learned congestion prediction models to inform preliminary floor-planning decisions and set timing budgets [29].

ECOs are highly significant in closure periods in physical-aware deployment. The physical-aware changes take into account real placement, routing, and parasitic elements to make realistic suggestions on possible changes. Localized buffering, gate reordering, or re-routing means support by tools that rely on extracted RC tracking dissimilarities. To ensure that the new violations are not introduced or affect the adjacent logic, such ECOs are validated step-by-step [30]. The physical-design-informed timing strategies significantly enhance the closure process, making it more robust and layout-accurate. These techniques must be embedded within integrated tool flows that can seamlessly propagate constraints and feedback across the design hierarchy, which will be explored next. Continuing from the previous discussion, the article highlights how tool flows and methodology integration form the connective framework that enables all previously discussed timing closure strategies, logical, physical, and AI-driven, to operate cohesively in adaptive SoC environments.

8. Tool Flows and Methodology Integration

An effective timing closure methodology, when used in support of adaptive SoCs, will be as good as the tools and methodology on which it is based. The system-level nature of adaptive architectures requires flows capable of supporting hierarchical timing models, design iterations, and synthesis, placement, and signoff integration. The timing closure process is no longer linear, but an iterative, information-rich verification loop among functional modeling, physical design, and verification tools [1, 2, 31]. The new industry signoff tooling is capable of Multi-Mode Multi-Corner (MMMC) STA environments that can be expanded to include thousands of scenarios. The tools are mandatory in adaptive SoCs where the logic blocks may be configured on the fly, and a timing analysis must be performed under different corner cases. Key capabilities are support of incremental analysis, hierarchical modeling, voltage-aware delay modeling, and in-design fix suggestion using physical data [3, 32, 33].

In addition to signoff, automated flow integration becomes more and more a necessity. Flows are automated and used to coordinate timing-motivated synthesis, timing-driven clock tree synthesis, physical placement, routing, and hardware ECO application in closely synchronized steps. Timing reports feedback is employed to automatically make design parameter adjustments, local optimization runs, or machine-learning-based fixes. These looped approaches eliminate the manual iteration in it and enable parallel design activities, which can lead to a shorter closure. Hybrid timing closure flows further optimize results by blending traditional STA with formal verification and machine learning analytics. For example, formal methods may confirm logical correctness across all configurations, while STA tools verify temporal validity under worst-case process-voltage-temperature (PVT) conditions. In parallel, AI engines highlight suspicious timing paths or recommend constraint refinements. This hybrid approach enhances timing coverage and bridges gaps between rule-based and data-driven validation [5]. Cross-tool integration also ensures consistent modeling across abstraction levels. Timing constraints defined in RTL must propagate accurately to gate-level netlists and physical designs. Tools support this through unified timing constraint languages, static rule checkers, and signoff correlation engines that trace constraint violations across stages. For adaptive SoCs with multiple design teams and third-party IPs, such integration is critical to maintain coherence and timing predictability [34, 35]. Having explored the architecture, strategies, and tools, it is now essential to illustrate how these methodologies function in practice. The next section presents case studies and experimental outcomes that validate the advanced timing closure techniques discussed.

9. Case Studies on Timing Closure Techniques in Adaptive SoC Integration

Achieving robust timing closure in the context of full-chip integration for adaptive SoCs necessitates a holistic approach that spans architectural planning, floorplanning, physical synthesis, and high-level synthesis (HLS) optimization. Given the inherent complexity introduced by multiple timing domains, reconfigurable fabrics, and data-dependent control logic, traditional flat design flows fall short in delivering predictable performance metrics. Recent research has increasingly focused on innovative methodologies that emphasize timing-aware partitioning, high-level floorplan directives, and network-on-chip (NoC) integration strategies to address the intricacies of modern SoC architectures. Guo et al. [36] introduce RapidStream 2.0, a timing-driven, split-compilation framework engineered to expedite FPGA implementation for large-scale latency-insensitive accelerator designs while enhancing timing reliability. The methodology capitalizes on the inherent task-pipeline architecture of TAPA dataflow designs, which comprise computational kernels interconnected through FIFO channels. At each FIFO boundary, the authors insert pipeline registers positioned within constrained areas designated as anchor regions. These anchor regions function as timing-isolation boundaries that partition lengthy cross-module paths, enabling each computational island to achieve local timing closure independently of the broader system.The workflow initiates by constructing a skeleton design that retains solely the inter-island connectivity: anchor registers, net stubs, and virtual partition pins that represent the routing topology of each inter-island connection. This skeleton undergoes global routing to establish deterministic delays across anchor boundaries. Subsequently, each island is individually floorplanned, placed, and routed in parallel through distributed compilation. RapidStream 2.0 incorporates several refinements relative to its predecessor, including enhanced ILP-based island partitioning, anchor-aware parallel placement to align island and boundary register locations, and clock trunk planning to preserve uniform skew throughout final integration. Following island-level routing completion, all islands are integrated into the top-level design via a DFX-compatible assembly workflow that enforces regional isolation and maintains the pre-routed inter-island connectivity. Experimental evaluation using accelerator workloads deployed on a Xilinx Alveo U280 demonstrates 5-7× overall compilation speedup and up to 1.3× frequency enhancement compared to a conventional Vivado workflow, with additional performance gains relative to RapidStream 1.0 and floorplan-guided baseline approaches. The research confirms that latency-insensitive partitioning integrated with anchor-guided routing delivers a scalable and predictable methodology for timing closure in contemporary adaptive FPGA-based SoC architectures. The INSTA study by Lu et al. [37] introduces a tool-accurate, differentiable, statistical static timing analysis (STA) engine specifically designed for advanced industrial physical design flows at technology nodes as aggressive as 3 nm. In contrast to prior GPU-accelerated STA methodologies that reconstruct delay models from foundational principles and consequently exhibit poor correlation with signoff tools, INSTA explicitly separates delay calculation from timing propagation. During a one-time initialization phase, its "clones" per-arc attributes from a reference commercial STA engine, encompassing rise/fall delays, unateness properties, statistical parameters (mean and sigma for POCV), and timing constraints including multicycle paths and false paths, together with startpoint/endpoint clock information and required arrival times. Subsequently, the timing graph is levelized, and all timing propagation is executed on the GPU using custom CUDA kernels, modeling arrival times as Gaussian random variables while maintaining Top-K candidate arrivals per endpoint with distinct startpoints to enable accurate common path pessimism removal (CPPR) within a statistical framework. The core algorithmic approach comprises a forward kernel for OCV-aware statistical propagation and a backward kernel for gradient backpropagation. During the forward pass, each pin at a specified timing level is assigned to a CUDA thread, which aggregates parent arrivals and arc delays while accounting for rise/fall transitions, unateness, and common path sharing. The traditional max operator across candidate arrivals is substituted with a smooth log-sum-exp formulation, rendering endpoint slack expressions differentiable with respect to leaf variables such as gate sizes and cell coordinates. This enables derivation of "timing gradients", partial derivatives of global metrics including worst negative slack (WNS) and total negative slack (TNS) with respect to physical or logical parameters, which can subsequently drive gradient-based optimization routines. INSTA operates in two primary modes: an evaluation mode, wherein only delays on modified arcs are re-annotated and the complete timing graph is re-propagated on GPU for rapid incremental STA; and an optimization mode, were gradients guide transformations such as cell sizing or placement adjustments. Experimental validation demonstrates INSTA's performance on multiple production high-performance designs fabricated in a foundry 3 nm process with OCV enabled. For the largest evaluated design (approximately 15 million pins), INSTA achieves full-graph timing propagation in under 0.1 seconds with endpoint slack correlation ranging from 0.999 to 0.9999 relative to an industry-leading signoff engine, despite relying on cloned rather than natively implemented delay models. When integrated as a rapid timing evaluator within a commercial gate-sizing workflow, it delivers approximately 25× speedup over the reference tool's incremental timing analysis with negligible quality loss. Building upon this capability, the authors introduce INSTA-Size, a gradient-based gate sizing optimizer that leverages timing gradients to prioritize only the most influential cells: compared to the commercial signoff tool, it achieves up to 15 percent TNS improvement while resizing 68 percent fewer cells. Similarly, INSTA-Place employs timing gradients to enhance a global placement engine, surpassing a state-of-the-art net-weighting-based timing-driven placer on ICCAD'15 benchmarks by up to 59.4 percent in TNS and 16.2 percent in half-perimeter wirelength. Collectively, these results demonstrate that a tool-accurate, GPU-accelerated, differentiable STA engine can simultaneously achieve near-signoff precision and order-of-magnitude runtime improvements, thereby enabling truly global, gradient-driven timing closure strategies appropriate for advanced SoC integration environments. Chen et al. [38] introduce a virtual-path-based timing optimization framework designed to enhance timing performance during the global placement phase of VLSI physical design. Conventional timing-driven placement techniques rely predominantly on net-weighting, where critical nets receive elevated weights to influence placer behavior. However, these approaches frequently overlook the complex interdependencies inherent in complete timing paths, leading to suboptimal optimization outcomes or unintended degradation of non-critical paths. The proposed framework remedies this shortcoming by modeling timing-critical arcs (ARCs) alongside their spatial relationships using virtual nets, thereby enabling more holistic timing-driven placement optimization. The methodology initiates with the extraction of critical timing paths from static timing analysis (STA), which are subsequently decomposed into individual ARCs. Each ARC receives a criticality score calculated from several attributes, including path slack, physical ARC length, occurrence frequency, and Manhattan distance. These metrics ensure that optimization efforts concentrate on circuit regions exhibiting both high timing sensitivity and significant spatial influence. The algorithm further extends selected ARCs via breadth-limited expansion to encompass adjacent non-critical paths, thereby mitigating potential timing regressions in neighboring logic structures. For every chosen ARC, a two-pin virtual net is constructed, with its associated weight computed using a closed-form function that accounts for ARC depth and timing sensitivity. These virtual nets serve as guidance mechanisms during analytical placement, promoting cell proximity to minimize delay while preserving reasonable wirelength. To enhance solver stability and computational efficiency, the authors incorporate a Jacobi diagonal preconditioner that approximates the Hessian matrix, thereby accelerating convergence within the analytical placement engine. The proposed optimization workflow was integrated into an analytical placer and validated using ICCAD-2015 and proprietary industrial benchmarks. Placement results were legalized through Jezz and subsequently analyzed using Cadence Innovus 2020.13. Relative to a dynamic net-weighting baseline, the framework delivered an 11.2 percent improvement in worst negative slack (WNS) and a 15.9 percent reduction in total negative slack (TNS), while maintaining competitive runtime and wirelength metrics. These outcomes underscore that path-aware optimization introduced at early design stages can substantially enhance timing convergence and predictability, providing a practical methodology for addressing sophisticated timing closure challenges in contemporary full-chip integration flows. Lecler and Baillieu [39] describe an application-driven methodology for architecting and optimizing a network-on-chip (NoC) interconnect within a complex, DRAM-centric SoC deployed in a handheld gaming device. The case study employs Arteris NoC technology and advocates a layered, top-down design flow that commences with a comprehensive functional specification defining initiator/target sockets, memory maps, and protocol translations (AXI, OCP, AHB, etc.), subsequently augmented by a performance specification articulated through executable traffic scenarios. These scenarios, authored in a concise scripting language and translated to SystemC TLM-2.0, represent realistic system behaviors and quality-of-service (QoS) requirements across heterogeneous subsystems including multi-core CPUs, display controllers, imaging blocks, modems, and background I/O peripherals. Early-stage cycle-accurate "Architect View" (AV) simulations are then employed to assess whether the interconnect architecture satisfies bandwidth, latency, and efficiency targets under representative operational workloads.

Architectural exploration advances through multiple refinement stages. Initially, a path-based topology is established using abstract links that aggregate and distribute traffic classes, incorporating explicit controls for serialization, clock domain crossings, and DRAM scheduler port allocation. Subsequently, buffering and "rate adaptation" mechanisms are introduced to accommodate mismatched peak throughputs and manage backpressure, particularly along DRAM and display pathways. A hierarchical QoS framework is then implemented, utilizing urgency and hurry tags alongside bandwidth regulators to prioritize hard real-time transactions (display, modem) over soft real-time (imaging) and best-effort traffic. Following this, cost optimization reduces context depths and resource allocations based on measured utilization metrics, while pipeline stages are strategically inserted along routing paths to facilitate physical timing closure across multiple clock and power domains without incurring excessive area or latency overhead. Ultimately, an automated "Structure" synthesis phase transforms the architecture into a concrete RTL netlist comprising network interface units, transport blocks, and clocking infrastructure, followed by a "Verification View" (VV) that re-executes identical scenarios on a bit- and cycle-accurate model. The strong correlation between AV and VV results validates that early architectural exploration can reliably inform design decisions regarding QoS allocation, buffering strategies, and pipelining, while ensuring adherence to timing, bandwidth, and integration requirements within a full-chip SoC environment. Collectively, these case studies underscore a pivotal trend in modern SoC design: timing closure is no longer confined to backend optimization stages but is a cross-hierarchical challenge requiring early-stage design co-optimization. The solutions explored from

RapidStream's island-based physical implementation and anchor-register methodology to FADO's co-optimization of HLS and floorplanning demonstrate that integrating timing constraints into upstream design processes can significantly enhance design convergence and operational frequency. Furthermore, the emphasis on architectural features such as NoC topologies and elastic dataflow partitioning points to a future where timing closure is not only automated but also architecturally embedded. These methodologies affirm the value of physically aware, hierarchical, and constraint-driven design flows as essential enablers for successful full-chip timing closure in adaptive SoCs.

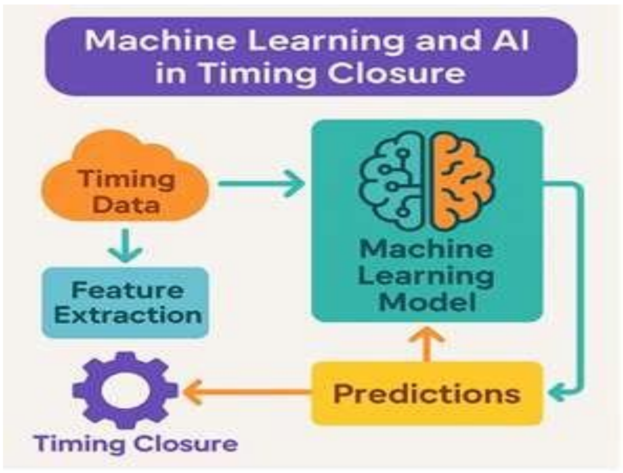


Figure 1: *A conceptual diagram illustrating how Machine Learning and AI are applied in timing closure, from timing data and feature extraction to predictive modeling and optimization.*

Table 1: *Comparison Between Traditional and Advanced Timing Closure Techniques*

Aspect	**Traditional Timing Closure**	**Advanced Techniques for Adaptive SoCs**
Analysis Scope	Full-chip flat STA	Hierarchical, partitioned STA
Design Abstraction	Gate-level only	Gate-level + IP-level black-box abstraction
Mode-Corner Handling	Single or a few modes	Multi-mode, multi-corner (MMMC)
Change Propagation	Manual updates after logic change	Incremental ECO support with automated feedback
Tool Intelligence	Rule-based	AI/ML-guided prediction and optimization
Closure Convergence Time	High	Lower due to localized re-analysis
Constraint Management	Static constraints only	Dynamic, scenario-aware constraints
Physical Awareness	Basic routing-based adjustments	Placement-aware ECO with RC extraction

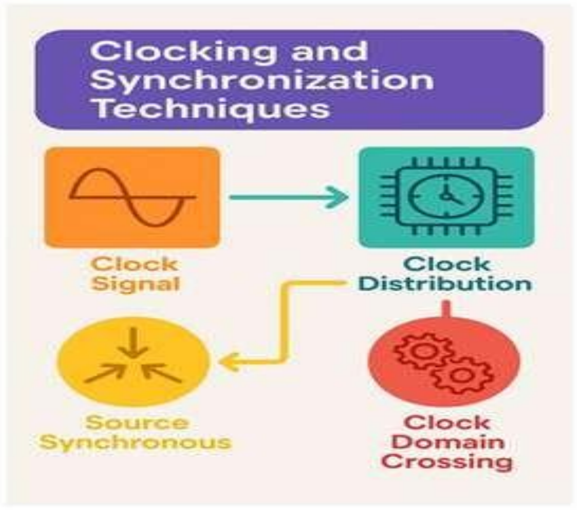


Figure 2: *An overview of clocking and synchronization techniques, illustrating key concepts such as clock signals, distribution, source synchronous communication, and clock domain crossing.*

10. Challenges and Future Directions

While the techniques discussed thus far provide a robust foundation, several emerging challenges threaten to outpace current timing closure capabilities. As semiconductor manufacturing enters sub-5nm nodes and system complexity grows with chiplet-based integration, the traditional assumptions of timing behavior and closure strategies begin to unravel [13].
Scaling to advanced nodes such as 5nm, 3nm, and beyond introduces unprecedented levels of variability. These include increased gate delay sensitivity to voltage and temperature, higher metal layer resistance-capacitance (RC) parasitics, and proximity effects from neighboring devices. Furthermore, manufacturing-induced variability, such as line-edge roughness and random dopant fluctuations, can significantly alter timing margins. Closure tools must incorporate statistical and probabilistic models to address these variations rather than relying solely on corner-based analysis [14]. The adoption of chiplet-based architectures introduces a new challenge: inter-die timing closure. Unlike monolithic SoCs, chiplets communicate via high-speed interposers or die-to-die links, which exhibit unique timing characteristics including skew, latency variability, and signal integrity issues. Timing closure in this context must incorporate cross-chip analysis and synchronized constraint propagation across heterogeneous dies [15]. Additionally, adaptive SoCs increasingly rely on software-defined behaviors, making co-verification of hardware timing and software execution paths critical. Dynamic scheduling decisions, reconfiguration events, and software-initiated power gating all impact timing paths. This necessitates hybrid timing verification that considers real-time software impact, a field still in its infancy [16]. Artificial Intelligence will continue to shape the future of timing closure. Advanced models such as graph neural networks (GNNs) and reinforcement learning agents are being researched to offer more precise path predictions, routing-aware closure suggestions, and automated constraint tuning. The incorporation of explainability in AI tools will also be essential to gain trust and adoption in signoff-critical flows [17]. Looking forward, timing closure must evolve from a signoff step into an embedded, continuous process throughout the design lifecycle from RTL synthesis to post-silicon validation. This holistic approach, underpinned by AI, hierarchical abstraction, and physical-aware flows, represents the direction of future SoC design methodology.

11. Conclusion

Adaptive SoCs represent a paradigm shift in semiconductor design, demanding new timing closure strategies that account for runtime variability, reconfiguration, and multidomain interactions. Traditional static analysis methods alone are inadequate for the dynamic timing requirements imposed by features such as DVFS, CDC, and programmable logic. This article has detailed a range of advanced timing closure methodologies, including hierarchical abstraction, incremental optimization, machine learning-enhanced path analysis, and physical-aware ECO implementation. Case studies have shown measurable improvements in timing convergence, closure time, and post-silicon accuracy, reinforcing the necessity and effectiveness of these approaches. Future trends suggest a convergence of AI, physical modeling, and cross-domain timing methodologies as the industry adapts to advanced nodes and chiplet integration. As timing becomes increasingly dynamic and multifactorial, continuous innovation in tools, algorithms, and design methodologies will be critical to meet the demands of next-generation adaptive SoCs.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

1. Sheng, D., Lin, H. R., & Tai, L. (2021). Low-Process–Voltage–Temperature-Sensitivity Multi-Stage Timing Monitor for System-on-Chip Applications. *Electronics*, *10*(13), 1587.

2. Jasmin, M., & Philomina, S. *Runtime adaptive Dynamic Voltage Frequency Scaling technique for reducing the power consumption in Multi-Processor System On Chip*.
3. Li, B., Chen, N., Schmidt, M., Schneider, W., & Schlichtmann, U. (2009, April). On hierarchical statistical static timing analysis. In *2009 Design, Automation & Test in Europe Conference & Exhibition* (pp. 1320-1325). IEEE.
4. N. Karimi and K. Chakrabarty, "Detection, Diagnosis, and Recovery From Clock-Domain Crossing Failures in Multiclock SoCs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 9, pp. 1395-1408, Sept. 2013, doi: 10.1109/TCAD.2013.2255127.
5. Damsgaard, H. J., Grenier, A., Katare, D., Taufique, Z., Shakibhamedan, S., Troccoli, T., ... & Nurmi, J. (2024). Adaptive approximate computing in edge AI and IoT applications: A review. *Journal of Systems Architecture*, *150*, 103114.
6. Kalapothas, S., Flamis, G., & Kitsos, P. (2022). Efficient edge-AI application deployment for FPGAs. *Information*, *13*(6), 279.
7. Reddy, K. V. UVM-BASED POWER-AWARE VERIFICATION CLOSURE USING DYNAMIC VOLTAGE AND FREQUENCY SCALING (DVFS) MODELS.
8. Jain, A. M., & Blaauw, D. (2004, February). Modeling flip flop delay dependencies in timing analysis. In *ACM/IEEE Timing Issues (TAU) Workshop, Austin, TX*.
9. Hatami, S., Abrishami, H., & Pedram, M. (2008, May). *Statistical timing analysis of flip-flops considering codependent setup and hold times*. In Proceedings of the 18th ACM Great Lakes symposium on VLSI (pp. 101-106).
10. Fu, C. (2023). *Machine Learning Techniques for Early Identification of Timing Critical Flip-Flops in Digital IC Designs* (Doctoral dissertation).
11. Madhuri, G. M. G., & Selvakumar, J. Machine-Learning Techniques for Predicting Post-CTS Worst Negative Slack.
12. Raha, A., Kim, S. K., Mathaikutty, D. A., Venkataramanan, G., Mohapatra, D., Sung, R., ... & Chinya, G. N. (2021, February). Design considerations for edge neural network accelerators: An industry perspective. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*(pp. 328-333). IEEE.
13. Li, B., Chen, N., Xu, Y., & Schlichtmann, U. (2013). On timing model extraction and hierarchical statistical timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *32*(3), 367-380.
14. Owen, A., & Davids, L. (2025). Timing Closure Verification of DDR PHY Interfaces Across PVT Corners Using Advanced STA Techniques.
15. Guo, G. (2023). *Parallel and heterogeneous computing for static timing analysis* (Doctoral dissertation, University of Illinois at Urbana-Champaign).
16. Zhao, Z., Zhang, S., Liu, G., Feng, C., Yang, T., Han, A., & Wang, L. (2022). Machine-learning-based multi-corner timing prediction for faster timing closure. *Electronics*, *11*(10), 1571.
17. Ganesan, S. (2025). Advanced Timing Closure Methodologies for High-Performance Neural Network Accelerators: A Comprehensive Framework. *Journal Of Engineering And Computer Sciences*, *4*(8), 158-166.
18. Hu, J., & Kahng, A. B. (2023, October). the inevitability of AI infusion into design closure and signoff. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)* (pp. 1-7). IEEE.
19. Jiang, W., Chhabria, V. A., & Sapatnekar, S. S. (2024, September). IR-aware ECO timing optimization using reinforcement learning. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD* (pp. 1-7).
20. Xing, W. W., Wang, L., Wang, Z., Shi, Z., Xu, N., Cheng, Y., & Zhao, W. (2024). Multicorner Timing Analysis Acceleration for Iterative Physical Design of ICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *43*(7), 2151-2162.
21. Simons, B. (2005). An overview of clock synchronization. *Fault-Tolerant Distributed Computing*, 84-96.
22. Buckler, M., & Burleson, W. (2014, March). Predictive synchronization for DVFS-enabled multi-processor systems. In *Fifteenth International Symposium on Quality Electronic Design* (pp. 270-275). IEEE.
23. Rahmani, A. M., Liljeberg, P., Plosila, J., & Tenhunen, H. (2010, May). Power and performance optimization of voltage/frequency island-based networks-on-chip using reconfigurable synchronous/bi-synchronous FIFOs. In *Proceedings of the 7th ACM international conference on Computing frontiers* (pp. 267-276).
24. MARCHESE, A. (2015). A DVFS-capable heterogeneous network-on-chip architecture for power constrained multi-cores.
25. Yeung, P., & Mandel, E. (2015, November). Multi-Domain Verification of Power, Clock and Reset Domains. In *Haifa Verification Conference* (pp. 245-255). Cham: Springer International Publishing.
26. Ye, Y., Xu, P., Ren, L., Chen, T., Yan, H., Yu, B., & Shi, L. (2024). Learning-driven physically-aware large-scale circuit gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
27. Bai, L., & Chen, L. (2018, October). Machine-learning-based early-stage timing prediction in SoC physical design. In *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)* (pp. 1-3). IEEE.
28. P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin and B. Yu, "DREAMPlace 4.0: Timing-driven Global Placement with Momentum-based Net Weighting," *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp,

Belgium, 2022, pp. 939-944, doi: 10.23919/DATE54114.2022.9774725.

29. Joshua, C., Garcia-Gasulla, D., Walsh, I., & Kotsis, K. (2025). Automated Timing Closure with Machine Learning: Case Studies and Practical Results.
30. Chen, H. T., Chang, C. C., & Hwang, T. (2009). Reconfigurable ECO cells for timing closure and IR drop minimization. *IEEE transactions on very large scale integration (VLSI) systems*, *18*(12), 1686-1695.
31. S. Zheng, L. Zou, S. Liu, Y. Lin, B. Yu and M. Wong, "Mitigating Distribution Shift for Congestion Optimization in Global Placement," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-6, doi: 10.1109/DAC56929.2023.10247660.
32. Zuodong Zhang, Zizheng Guo, Yibo Lin, Runsheng Wang, and Ru Huang. 2022. AVATAR: an aging- and variation-aware dynamic timing analyzer for application-based DVAFS. In Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC '22). Association for Computing Machinery, New York, NY, USA, 841–846. https://doi.org/10.1145/3489517.3530530
33. Subhendu Roy, Pavlos M. Mattheakis, Laurent Masse-Navette, and David Z. Pan. 2014. Clock tree resynthesis for multi-corner multi-mode timing closure. In Proceedings of the 2014 on International symposium on physical design (ISPD '14). Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1145/2560519.2560524
34. A. B. Kahng, S. Kang, H. Lee, S. Nath and J. Wadhwani, "Learning-based approximation of interconnect delay and slew in signoff timing tools," *2013 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, Austin, TX, USA, 2013, pp. 1-8, doi: 10.1109/SLIP.2013.6681682.
35. T. -W. Huang, G. Guo, C. -X. Lin and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 776-789, April 2021, doi: 10.1109/TCAD.2020.3007319.
36. Guo, L., Maidee, P., Zhou, Y., Lavin, C., Hung, E., Li, W., ... & Cong, J. (2023). Rapidstream 2.0: Automated parallel implementation of latency–insensitive FPGA designs through partial reconfiguration. *ACM Transactions on Reconfigurable Technology and Systems*, *16*(4), 1-30.
37. Y. -C. Lu, Z. Guo, K. Kunal, R. Liang and H. Ren, "INSTA: An Ultra-Fast, Differentiable, Statistical Static Timing Analysis Engine for Industrial Physical Design Applications," *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2025, pp. 1-7, doi: 10.1109/DAC63849.2025.11132858.
38. W. Chen, H. Huang, M. Wei, P. Zou and J. Chen, "Virtual-Path-Based Timing Optimization for VLSI Global Placement," *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, Nangjing, China, 2022, pp. 1-3, doi: 10.1109/ICSICT55466.2022.9963291
39. Lecler, J. J., & Baillieu, G. (2011). Application-driven network-on-chip architecture exploration & refinement for a complex SoC. *Design Automation for Embedded Systems*, *15*(2), 133-158