

KAIROS: Fast Microarchitectural Design Space Exploration by Bridging Architectural Semantics and Logic Gates

Shilin Tian

University of Central Florida
Orlando, FL, USA
Shilin.Tian@ucf.edu

Sanjay Gandham

University of Central Florida
Orlando, FL, USA
Sanjay.Gandham@ucf.edu

Lingxiang Yin

University of Central Florida
Orlando, FL, USA
Lingxiang.Yin@ucf.edu

Mingjie Lin

University of Central Florida
Orlando, FL, USA
Mingjie.Lin@ucf.edu

Hao Zheng

University of Central Florida
Orlando, FL, USA
Hao.Zheng@ucf.edu

Abstract—The proliferation of generative AI and Large Language Models (LLMs), with the availability of open-source register-transfer-level (RTL) libraries, has drastically shortened the time needed for RTL development. Hardware design is poised to transition from traditional manual coding to parameterized microarchitectural compilation. However, this rapid acceleration in RTL generation has inadvertently exposed a severe bottleneck in microarchitectural Design Space Exploration (DSE). Effective DSE requires evaluating candidate architectures in terms of functional performance (e.g., instructions per cycle) and physical performance (e.g., achievable clock frequency). However, while functional performance can be assessed using cycle-accurate simulation, evaluating physical performance still relies on gate-level logic synthesis, which often takes hours or even days to refine and optimize microarchitectural configurations.

In this paper, we identify the primary bottleneck in current chip design workflows as the semantic gap between microarchitectural design and physical design, through which the original design intent is irreversibly lost after logic synthesis. To address this challenge, we introduce a new Circuit Intermediate Representation (CIR) that preserves microarchitectural design intent by connecting RTL functional semantics with the corresponding synthesized gate-level netlist. On top of CIR, we build KAIROS, an optimization framework for efficient microarchitectural exploration. By retaining word-level RTL operators and signal names, KAIROS supports traceable operator-level critical path analysis without suffering from the semantic loss inherent in gate-level synthesis. In addition, we construct And-Inverter Graphs (AIGs) for operator-level timing prediction and develop CIR-based fan-out optimization and retiming techniques to identify Pareto-optimal physical implementations. This framework thus enables fast microarchitectural design space exploration with RTL-level accuracy. Evaluated on 62 design points spanning datapath, control, and memory components, from embedded cores to industrial-scale designs (the NVDLA accelerator and the CVA6 application-class core), KAIROS preserves post-synthesis Fmax rankings with Spearman $\rho=0.96$, achieves 7.1–13.4% frequency improvement on datapath circuits with 9.4% average area overhead, and completes timing evaluation $113\times$ faster than synthesis on average across 41 designs, up to $213\times$ on industrial-scale designs.

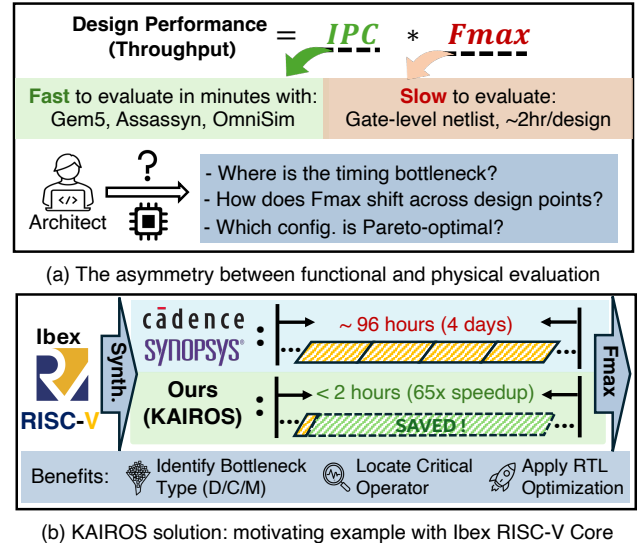


Fig. 1. Overview of the DSE evaluation gap (a) and the KAIROS solution (b).

I. INTRODUCTION

The relentless demand for higher computational performance in the Post-Moore era has intensified the focus on domain-specific acceleration and application-specific integrated circuits (ASICs) [1], [2]. In this landscape, time-to-market has emerged as a critical competitive advantage, placing unprecedented pressure on hardware design methodologies to achieve desired performance and energy efficiency. Recent advances in high-level synthesis (HLS) tools (e.g., Bambu [3] and Kiwi Compiler [4]), parameterized register-transfer-level (RTL) generation frameworks (Chisel/Chippyard [5], [6], Bluespec [7], and PyMTL3 [8]), as well as large language model (LLM)-based RTL generation approaches [9]–[16], have explored a variety of strategies to shorten the path from microar-

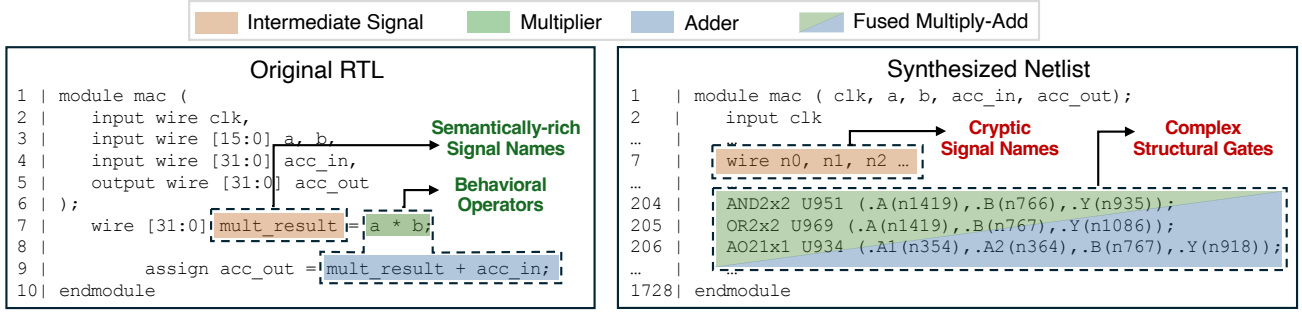


Fig. 2. RTL (left) preserves operator semantics and named signals. The synthesized netlist (right) expands 10 lines into 1,728 lines, losing design intent.

chitectural design to RTL implementation. Moreover, LLM-generated RTL is usually optimized for functional correctness rather than physical quality. Even functionally correct designs reach only about 64% of the area, delay, and power efficiency of optimized reference implementations [10], [17], and functional-correctness rates remain low on realistic designs [18], so many candidate designs must be generated and screened. Much of this efficiency gap can be recovered at the RTL level [19], [20], which makes fast, traceable identification of timing bottlenecks across many candidate designs increasingly valuable. However, contemporary hardware design flows address timing optimization through a multi-stage process that spans from initial RTL development to final physical implementation. The critical challenge of timing closure remains predominantly manual. Industry experience indicates that iterative refinement, along with logic and physical synthesis, remains the primary bottleneck for achieving timing and design closure in complex digital hardware projects [21], [22]. Repeated timing optimizations (e.g., register relocation and optimization of high-fan-out nets) can extend schedules by weeks or even months [21]–[26]. This tedious optimization process not only delays product schedules but also restricts the exploration of microarchitectural solutions due to the extensive cost of timing validation. Figure 1(a) illustrates this limitation: while functional performance (IPC) can be evaluated in minutes, physical timing evaluation requires hours of gate-level synthesis per design point.

The fundamental challenge lies in the semantic gap between RTL design and gate-level timing optimization. RTL designs are naturally expressed using behavioral operators and meaningful signal names that facilitate human understanding and reflect microarchitectural design intent. However, existing timing optimization techniques, such as retiming [27], [28], buffer insertion [29], and fan-out management [30], operate exclusively on synthesized netlists where this semantic information has been irreversibly lost through bit-blasting [31] and Boolean optimization [32]. Consequently, timing improvements discovered at the gate level cannot be easily mapped back to RTL modifications, forcing designers to manually interpret static timing analysis reports and apply ad-hoc register repositioning strategies without holistic visibility into interdependent timing paths. Moreover, gate-level optimization in EDA is usually a black-box process. If the optimized design fails to meet the target frequency, the entire synthesis flow

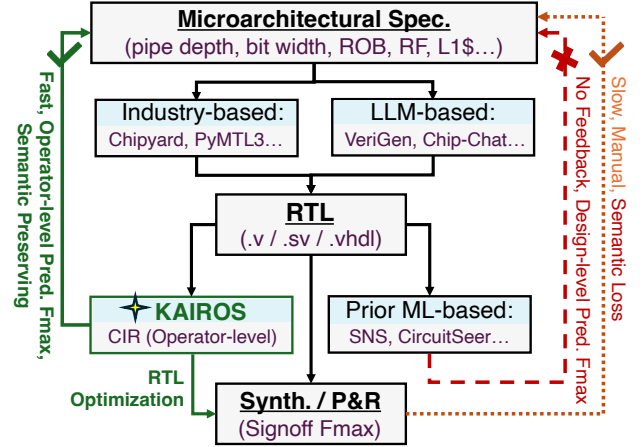


Fig. 3. Proposed KAIROS Framework for Fast Microarchitectural Design Space Exploration.

must be repeated, with limited insight into where further improvements can be made. This limits the potential for fast and automated microarchitectural design space exploration (DSE) in future chip design flows, particularly in an era of rapid RTL development.

Despite significant efforts to facilitate this process, prior optimizations are still performed largely in isolation at each stage of chip design. For example, post-synthesis performance prediction [43]–[45], [55] has been studied to provide timing, power, and area (PPA) estimation. However, these methods still operate largely at the netlist level, where improving PPA often requires substantial manual effort and relies heavily on designer experience to refine microarchitectural decisions. Motivated by this limitation, recent work has sought to elevate PPA optimization to the RTL level, thereby improving the interpretability of PPA behavior with respect to microarchitectural design choices. RTL-Timer [47], for instance, annotates timing slack at each register endpoint in RTL source code, while Fake Timer [48] predicts signal arrival times at the logic-block level. In parallel, compiler-driven frameworks such as ROVER [50], [56] perform verified equivalence-preserving rewrites on RTL datapaths, and AutoBridge [51] combines floorplanning with pipelining for HLS-generated designs. On the other hand, pre-RTL analytical frameworks such as Aladdin [38] and Charm [39] estimate microarchitectural performance and cost from high-level behavioral descriptions, but they are unable to capture detailed PPA characteristics.

TABLE I
LANDSCAPE OF TIMING EVALUATION APPROACHES FOR HARDWARE DESIGN.

Method Category	Representative Works	Timing Method	Fmax Pred.?	RTL Traceable?	RTL Optim.?
Cycle-accurate sim.	gem5 [33], Assassyn [34], OmniSim [35]	Perf. Simulation	✗	✗ (arch. params)	✗
Analytical modeling	McPAT [36], CACTI [37], Aladdin [38], Charm [39]	Analytical	Partial	✗ (arch. params)	✗
ML-assisted DSE	BOOM-Explorer [40], ArchGym [41], DOSA [42]	Surrogate	✗	✗ (arch. params)	✗
RTL-level ML pred.	MasterRTL [43], SNS [44], SNS v2 [45], ASAP [46]	GNN / ML	Partial (WNS)	✗ (design agg.)	✗
RTL timing annot.	RTL-Timer [47], Fake Timer [48], CircuitSeer [49]	GNN / ML	Partial (slack)	Partial (reg. endpts)	✗
Compiler-driven opt.	ROVER [50], AutoBridge [51], Calyx [52]	N/A	✗	✓ (RTL/HLS)	✓
Gate-level synthesis	Design Compiler [53], Genus [54]	STA	✓ (exact)	✗ (netlist only)	✓
RTL timing pred.+opt.	KAIROS (this work)	AIG depth	✓ (struct.)	✓ (RTL operators)	✓

Unfortunately, prior frameworks are unable to preserve high-level design intent through to the netlist level, limiting their ability to enable fast and traceable microarchitectural design space exploration.

This paper introduces KAIROS, a framework for rapid microarchitectural design space exploration by bridging microarchitectural semantics down to the logic-gate level as shown in Fig. 1(b), completing timing evaluation in minutes rather than hours. KAIROS builds a Circuit Intermediate Representation (CIR) that preserves RTL operators and signal names, enabling critical-path analysis without the semantic loss introduced by synthesis. A proposed AIG-based delay model then assigns timing estimates to CIR nodes based on their structural complexity, covering datapath operators, control primitives, and memory interfaces. Beyond prediction, KAIROS also supports RTL-level optimizations, such as fan-out management and retiming, directly on the CIR, providing both timing estimation and design improvement in a single pass that completes within minutes. The framework can be integrated into existing design tools to support fast microarchitectural design space exploration. The key contributions are as follows:

- **Circuit Intermediate Representation with AIG delay model.** We propose a Circuit Intermediate Representation that preserves RTL operators and signal names. In addition, we develop a structural delay model based on And-Inverter Graphs (AIGs) that estimates operator delays without requiring technology-specific netlists and generalizes across technology nodes without retraining.
- **RTL-level Timing Optimization.** We propose a tool that allows the implementation of retiming algorithms based on the proposed CIR, enabling automated RTL rewrites that preserve the original design intent while improving timing performance.
- **Fast and Accurate Microarchitectural Design Space Exploration.** The proposed framework bridges microarchitectural semantics and logic-gate representations, enabling fast and accurate microarchitectural DSE. We evaluate 41 designs, from the Ibex RISC-V core and OpenTitan IP blocks to industrial-scale designs (the NVDLA deep-learning accelerator and the CVA6 application-class core). The results show that the proposed framework accelerates DSE by 113× on average, up to 213×,

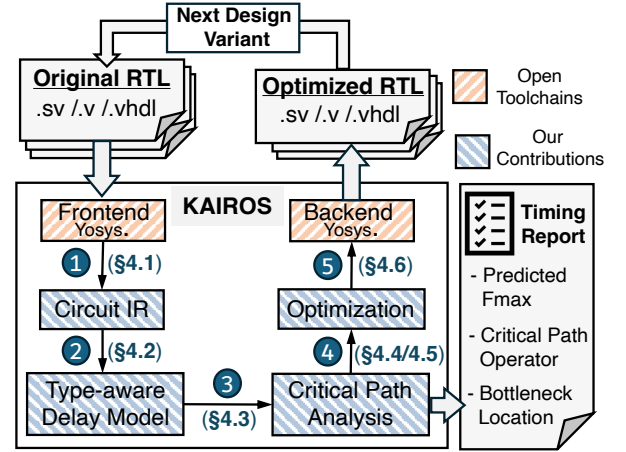


Fig. 4. Overview of the KAIROS framework.

compared with traditional design flows while preserving ranking quality.

II. BACKGROUND AND MOTIVATION

Microarchitectural design space exploration requires evaluating each candidate design along two key axes: functional performance, often measured in instructions per cycle (IPC), and physical performance, captured by the achievable clock frequency. The ultimate performance of a processor depends on both. KAIROS focuses on the timing optimization of specification-level DSE, while workload-level performance, including IPC, can be evaluated using cycle-accurate architectural simulators. These two analyses are complementary and can be combined through overall throughput, i.e., $IPC \times F_{max}$. In this workflow, the timing bottleneck reported by KAIROS, including datapath, control, and memory-system bottlenecks, identifies the microarchitectural optimization targets that should be further evaluated under workload-level IPC studies. Thus, KAIROS provides early timing guidance for specification-level design exploration, rather than replacing cycle-accurate workload evaluation. Over the past decades, functional performance has been widely assessed using cycle-accurate simulators [33]–[35] and FPGA-based emulation frameworks [57]. By contrast, physical timing evaluation remains a key bottleneck, since estimating clock frequency typically requires gate-level synthesis with commercial tools such as Design Compiler [53].

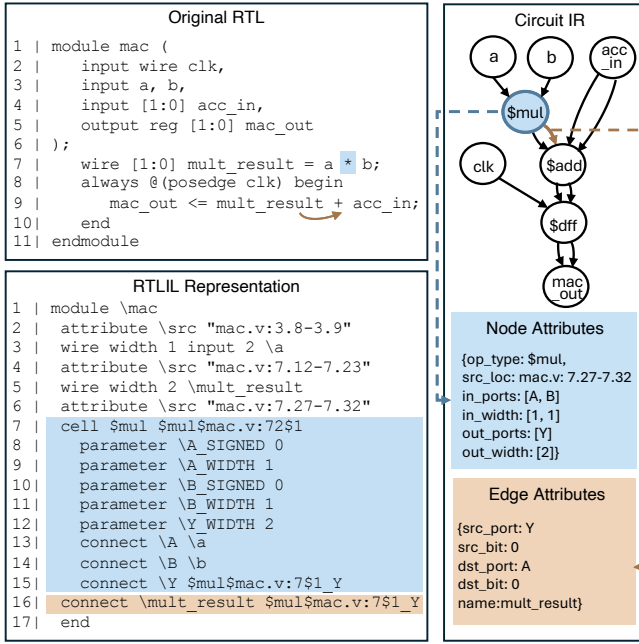


Fig. 5. RTL and its corresponding RTLIL and Circuit IR.

Table I summarizes prior approaches for fast microarchitectural design exploration, timing prediction, and optimization. Cycle-accurate simulators, for instance, enable efficient functional evaluation and report metrics such as instructions per cycle, but existing frameworks [33]–[35] cannot predict timing performance or offer direct guidance for microarchitectural optimization. Analytical models [36]–[38] estimate area and power from architectural parameters, yet they remain insufficient for predicting clock frequency or enabling automated RTL and microarchitectural optimization. In addition, several DSE frameworks [40]–[42] have been proposed for domain-specific accelerators and processor designs. However, these frameworks still have a limited ability to identify the optimal clock frequency and to provide RTL-level traceability for optimization and design refinement. This is because netlist timing prediction operates on a low-level representation where the semantic correspondence between RTL operators and synthesized gates has been irreversibly lost. For example, Fig. 2 shows an RTL for a 32-bit fixed-point multiply-accumulate functional unit. In RTL, the $*$ and $+$ operators describe a 16-bit multiplication and a 32-bit addition, and the named intermediate signal *mult_result* explicitly captures the output of the multiplication before accumulation. Its corresponding synthesized netlist shows this intermediate signal was removed, because the multiplier and adder can be structurally fused to form a fused multiply-add unit optimized at the gate level. Furthermore, the 10 lines of RTL describing the MAC unit expand into 1,728 lines of structural Verilog in the netlist, replacing a concise, human-readable arithmetic description with a low-level representation of interconnected gates.

With recent advances in shortening and automating RTL development [5]–[16], new opportunities have emerged to support fast microarchitectural design space exploration together

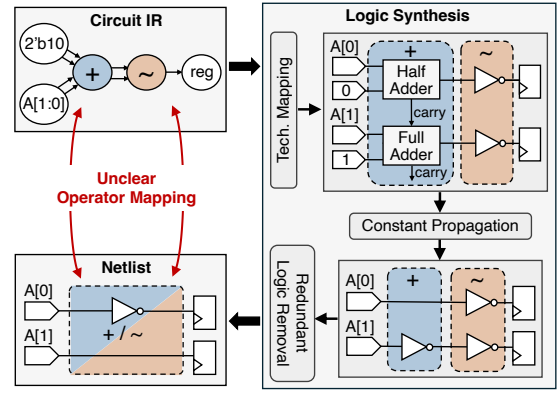


Fig. 6. Challenges with RTL Operator Modeling.

with physical performance evaluation. As shown in Fig. 3, the stages of current ASIC design flows are optimized largely in isolation, limiting the ability to perform fast and accurate microarchitectural design space exploration. At the core of this limitation is the semantic disconnect between high-level microarchitectural design intent and the synthesized netlist. This motivates us to develop an intermediate representation that preserves high-level design intent while enabling prediction of physical characteristics.

III. KAIROS FRAMEWORK OVERVIEW

Figure 4 presents the KAIROS workflow. The framework takes RTL as input and produces two outputs: a timing report (predicted Fmax, critical path operators, bottleneck component type) and an optimized RTL design. The entire flow is elaborated as follows.

1 The frontend converts input RTL into a Circuit Intermediate Representation (CIR) via the Yosys RTLIL infrastructure [58], preserving word-level operators and designer-specified signal names. Unlike gate-level netlists, where synthesis decomposes behavioral operators into bit-level gates and eliminates named signals [59], the CIR retains a one-to-one mapping between RTL operators and graph nodes, enabling operator-level analysis and traceability [60], [61].

2 Each CIR node is classified by component type (datapath, control, or memory) based on its RTLIL cell type, and an AIG-based delay model assigns a timing estimate appropriate to that type. The delay model uses AIG depth as a technology-independent proxy for combinational delay, with per-type scaling: linear for datapath operators, logarithmic for control primitives, and analytical lookup for memory macros.

3 KAIROS computes the W (minimum register weight) and D (maximum delay) matrices on the delay-annotated CIR using the Floyd-Warshall algorithm. The longest combinational path (where $W=0$) determines the predicted Fmax. The operators on this path are the critical operators, and their type classifications identify the bottleneck component type. Because the CIR operates at the word level (hundreds of nodes) rather than the gate level (millions of nodes), this analysis completes in seconds to minutes per design point.

4 KAIROS also improves the RTL through two optimization passes. Structural optimization targets high-fan-out CIR

TABLE II

CIR OPERATOR TAXONOMY WITH DELAY MODEL ASSIGNMENT. OPERATORS ARE CLASSIFIED BY COMPONENT TYPE AND MAPPED TO THE APPROPRIATE AIG SCALING MODEL.

Type	Verilog Construct	CIR Node	AIG Depth Scaling	DSE Knob
Type D — Datapath-Dominated				
Arith.	A + B	\$add	$\alpha \cdot b + \beta$	bitwidth
	A * B	\$mul	$\alpha \cdot b + \beta$	bitwidth
	A - B	\$sub	$\alpha \cdot b + \beta$	bitwidth
	A << B / A >> B	\$shl / \$shr	$\alpha \cdot b + \beta$	bitwidth
Red.	&A	\$reduce_and	$\gamma \lceil \log_2 b \rceil + \delta$	bitwidth
	A == B	\$eq	$\gamma \lceil \log_2 b \rceil + \delta$	bitwidth
	A < B	\$lt	$\gamma \lceil \log_2 b \rceil + \delta$	bitwidth
Type C — Control-Dominated				
Select	? : / case	\$mux / \$pmux	$3 \lceil \log_2 N \rceil$	fan-in N
	1 << sel	decoder	$O(\log_2 N)$	width N
FSM	priority if	priority enc.	$O(\log_2 N)$	width N
	round-robin	arbiter	$O(N) + O(\log_2 N)$	ports N
FSM	always FSM	next-state	encoding dep.	# states
			(one-hot << binary)	encoding
Type M — Memory-Dominated				
Mem.	SRAM macro	\$mem / macro	$f(\text{cap, ports})$	capacity
	register file	\$mem_v2	$f(\text{entries, R/W})$	# ports

of the tree. In this flow, the number of AIG nodes and the number of levels serve as heuristics for the circuit area and delay, respectively. As the example in Fig. 7(a) shows, the node (2-bit adder) delay can be represented with 4 levels. Based on this insight, we generate combinational RTL templates for each operator with varying bitwidth. The inputs and outputs of the operators are connected to the module I/O ports. We then use ABC to convert the operator to an AIG and record the number of levels as pseudo-operator delay. Mathematically, for each operator op with bitwidth b , we construct an And-Inverter Graph $G(op, b) = (V, E)$ via ABC synthesis using the transformation sequence $\mathcal{T} = \langle \text{strash, balance, rewrite, refactor} \rangle$ [74]. The AIG depth of each operator (e.g., mul or add) is defined by the following level function based on its AIG graph.

$$\ell(v) = \begin{cases} 0 & \text{if } v \in \text{PI} \\ \max\{\ell(u) : u \in \text{fanin}(v)\} + 1 & \text{if } v \in N \cup \text{PO} \end{cases} \quad (1)$$

where PI denotes the inputs of the operator AIG, N denotes internal AND gate nodes in the AIG, PO denotes the outputs of each operator, and $\text{fanin}(v)$ is the set of immediate predecessors of AND gate nodes v . The operator pseudo-delay is the maximum level across all AIG nodes:

$$\hat{d}(op, b) = \max_{v \in V} \ell(v) \quad (2)$$

To characterize operator (node) delays, combinational RTL templates were generated for each operator type with bitwidth $b \in [1, 32]$, and their AIG levels were measured as shown in Fig. 7(b). We then use ordinary least squares regression [76] to find the best-fitting linear or logarithmic model for each operator. Linear operators such as *add* and *mul* fit the form $\hat{d}(op, b) = \alpha b + \beta$, while reduction and comparison operators (*reduce_and*, *eq*) follow logarithmic scaling $\hat{d}(op, b) = \gamma \lceil \log_2 b \rceil + \delta$. As illustrated in Fig. 7(b), the

Algorithm 1: DP-Based OPT Selection Strategy

Input : CIR graph $G(V, E)$, fan-out threshold τ_f , fan-out budget B , register-weight and delay matrices W, D , target period T , and delay-to-load conversion coefficients (α_{iso} , α_c , α_n)

Output: Optimized CIR G' with reduced critical fan-out bottlenecks

- 1 **Phase I**: Critical pairs
 $\mathcal{P}_{\text{critical}} \leftarrow \{(u, v) : W(u, v) = 0 \wedge D(u, v) > T\}$;
- 2 Candidate drivers $\mathcal{C} \leftarrow \{d \in V : |\text{fanout}(d)| > \tau_f, d \text{ lies on a critical pair path, and all edges of } d \text{ are combinational}\}$;
- 3 **Phase II**: Per-driver option evaluation;
- 4 **foreach** driver $d \in \mathcal{C}$ **do**
5 // Partition fan-outs by timing criticality;
6 $\mathcal{F}_c \leftarrow \{f \in \text{fanout}(d) : \exists(u, v) \in \mathcal{P}_{\text{critical}}, W(f, v) = 0\}$,
7 $\mathcal{F}_n \leftarrow \text{fanout}(d) \setminus \mathcal{F}_c$;
8 **if** $|\mathcal{F}_c| = 0 \vee |\mathcal{F}_n| = 0$ **then**
9 **continue** // Skip if uniform criticality;
10 **end**
11 $\lambda \leftarrow |\mathcal{F}_n| / |\text{fanout}(d)|$ // Load reduction ratio;
12 **Option 1 (none)**: $\text{gain}_{\text{none}} \leftarrow 0$, $\text{cost}_{\text{none}} \leftarrow 0$;
13 **Option 2 (isolation)**: $\text{gain}_{\text{iso}} \leftarrow D[d] \cdot \lambda \cdot \alpha_{\text{iso}}$, $\text{cost}_{\text{iso}} \leftarrow 1$;
14 // Buffer insertion for $\mathcal{F}_n$, reduce load on critical paths;
15 **Option 3 (splitting)**:
16 $\text{gain}_{\text{split}} \leftarrow D[d] \cdot (\alpha_c \frac{|\mathcal{F}_c|}{|\mathcal{F}_c| + |\mathcal{F}_n|} + \alpha_n \frac{|\mathcal{F}_n|}{|\mathcal{F}_c| + |\mathcal{F}_n|})$,
17 $\text{cost}_{\text{split}} \leftarrow 2$;
18 // Driver duplication: fresh driver for $\mathcal{F}_c$, buffered for $\mathcal{F}_n$;
19 **end**
20 **Phase III**: Budgeted DP over drivers $d_1, \dots, d_k \in \mathcal{C}$;
21 $\text{best}[0][j] \leftarrow D_{\text{max}}(G)$ for $j = 0, \dots, B$ // Current critical-path delay;
22 **foreach** $i = 1, \dots, k$ **and** $j = 0, \dots, B$ **do**
23 $\text{best}[i][j] \leftarrow \min_{o \in \{\text{none, iso, split}\}, \text{cost}_o \leq j} (\text{best}[i-1][j - \text{cost}_o] - \text{gain}_o(d_i))$, recording the chosen option;
24 **end**
25 Select the option assignment that attains $\min_j \text{best}[k][j]$ // A driver is transformed only if its option has gain > 0 ;
26 Apply load isolation: $b \leftarrow \text{createBuffer}(d)$; $\text{rewire}(d \rightarrow b \rightarrow \mathcal{F}_n)$;
27 apply load splitting: $d' \leftarrow \text{duplicate}(d)$;
28 $\text{rewire}(d' \rightarrow \mathcal{F}_c, d \rightarrow b \rightarrow \mathcal{F}_n)$;
29 **return** Structurally optimized G' with improved fan-out distribution

measured AIG levels (burnt orange points) closely match the fitted model (green curves), reflecting underlying circuit structures: linear growth for addition (ripple-carry adders) and multiplication (array multipliers with reduction trees), logarithmic growth for reduction/comparison operators (balanced tree implementations). The high regression accuracy confirms that AIG metrics reliably capture RTL operator complexity. Empirically, addition follows $\hat{d}(\text{add}, b) = 2.00b$ ($R^2=1.00$), multiplication follows $\hat{d}(\text{mul}, b) = 4.19b + 0.54$ ($R^2=0.997$), and reduction/comparison operators follow logarithmic scaling ($R^2=1.00$), where R^2 is the coefficient of determination.

a) *From AIG Depth to Delay Estimation*: The AIG depth $\hat{d}(op, b)$ is a unit-less structural metric. To convert it to a delay estimate in time units for Fmax prediction, we apply a technology-specific scaling factor τ (picoseconds per AIG level):

$$d_{\text{physical}}(op, b) = \tau \cdot \hat{d}(op, b) \quad (3)$$

For a given technology library, τ is determined by a one-time linear regression of AIG depths against post-synthesis operator delays for a small calibration set. Importantly, τ needs to be fitted only once per technology node, not per design. For the ASAP7 7nm library used in our evaluation, this yields a single scalar that maps the entire AIG-based delay model to physical delay estimates. Because retiming and

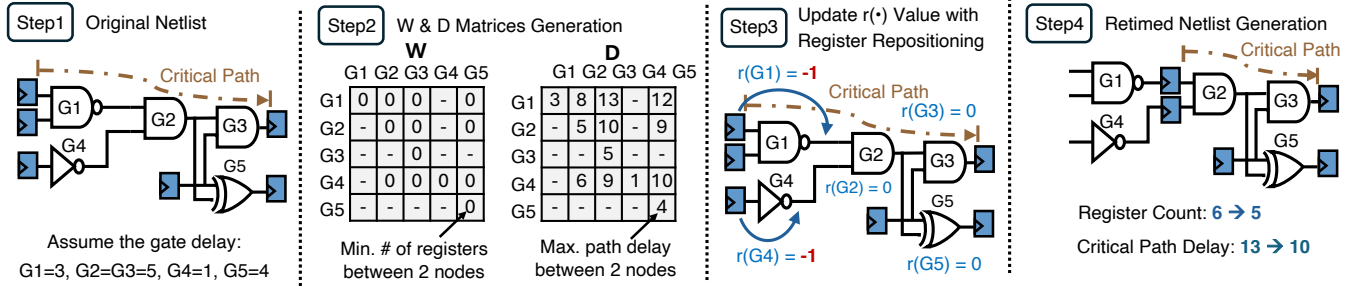


Fig. 9. Retiming: W/D matrix computation and register redistribution via $r(\cdot)$.

fan-out optimization require only *relative* delay ordering, the AIG depth metric is effective for optimization even without calibration; τ is needed only for absolute Fmax prediction. We now extend this methodology to Type C (control) and Type M (memory) components, completing a unified delay model across all component types. Table II summarizes the full operator taxonomy with delay model assignments and DSE knobs.

1) *Type C: Control Primitive Delay Models*: Control primitives differ from datapath operators in their scaling behavior: depth grows with fan-in width N rather than operand bitwidth b . We apply the same AIG characterization methodology used for datapath operators, generating parameterized Verilog templates, synthesizing them with ABC [74], and fitting regression models to the measured AIG depths.

a) *Multiplexer Trees*: An $N:1$ multiplexer is implemented as a balanced binary selection tree. We generate templates sweeping $N \in \{2, 4, 8, 16, 32, 64\}$ with data widths $W \in \{1, 8, 16, 32\}$. The measured AIG depth follows:

$$\hat{d}(\text{mux}, N) = 3 \lceil \log_2 N \rceil \quad (4)$$

The factor of 3 arises because each 2:1 MUX selection requires approximately 3 AIG levels (two AND gates and one inverter for the select logic). Depth is independent of data width W , since each bit of the output is selected independently. This model is applied to both $\$mux$ and $\$pmux$ (parallel MUX) RTLIL cells. In architectural terms, widening instruction decode from 1-wide to 4-wide increases MUX fan-in from 2 to 16, adding approximately $3 \times (\lceil \log_2 16 \rceil - \lceil \log_2 2 \rceil) = 9$ AIG levels to the control path.

b) *Decoders, Encoders, and Arbiters*: We apply the same template-and-fit procedure to decoders (N -bit input to 2^N -bit one-hot), priority encoders (N -input to grant index), and round-robin arbiters (N request ports). Decoders and priority encoders exhibit $O(\log_2 N)$ depth after ABC balancing; round-robin arbiters scale as $O(N) + O(\log_2 N)$ due to the rotating priority logic.

c) *FSM Next-State Logic*: We generate parameterized finite-state machines (FSMs) with state counts $N_s \in \{4, 8, 16, 32, 64, 128\}$, three encoding styles (binary, one-hot, Gray), and varying transition densities. We observed that one-hot encoding results in substantially shallower logic than binary encoding because each state transition activates a single state bit.

2) *Type M: Memory Component Delay Models*: Memory components (SRAM macros, register files) differ fundamentally from logic operators: their delay is determined by physical configuration parameters rather than combinational structure, and cannot be captured by AIG analysis. Instead, we use analytical delay models parameterized by the memory configuration.

a) *SRAM Macros*: For SRAM-based memory structures (caches, buffers), the access time depends on capacity, port count, and banking factor. We use published characterization data from the ASAP7 memory compiler and CACTI [37] to construct a lookup table:

$$\hat{d}(\text{sram}) = f(\text{capacity}, \text{ports}, \text{banking}) \quad (5)$$

where f is interpolated from the characterization table. This approach avoids rebuilding CACTI internals while providing sufficient accuracy for relative ranking during DSE.

b) *Register Files*: For synthesized register files (common in RISC-V cores like Ibex), access time scales with the number of entries and read/write ports. We characterize register file timing by synthesizing parameterized register file modules with Design Compiler for a one-time characterization, then storing the results as a lookup table indexed by entry count and port configuration.

c) *Integration into CIR*: Memory components appear in the CIR as nodes with fixed delay values drawn from the lookup tables above. These fixed-delay nodes participate in the same critical path analysis and retiming constraint computation as logic operators. The overall timing constraint becomes:

$$T_{\text{clock}} \geq \max(T_{\text{datapath}}, T_{\text{control}}, T_{\text{memory}}) \quad (6)$$

where each component's critical path delay is computed using its corresponding delay model. This unified formulation enables KAIROS to identify which component type is the timing bottleneck at each design point.

C. Critical Path Analysis

Once the CIR is annotated with per-node delays, KAIROS computes the W (minimum register weight) and D (maximum delay) matrices across all node pairs using the Floyd-Warshall algorithm:

$$W(u, v) = \min_{p: u \rightsquigarrow v} \sum_{(i, j) \in p} w_{\min}(i, j) \quad (7)$$

$$D(u, v) = \max_{p: u \rightsquigarrow v} \sum_{i \in p} d(i) \quad (8)$$

$w(p) = W(u, v)$

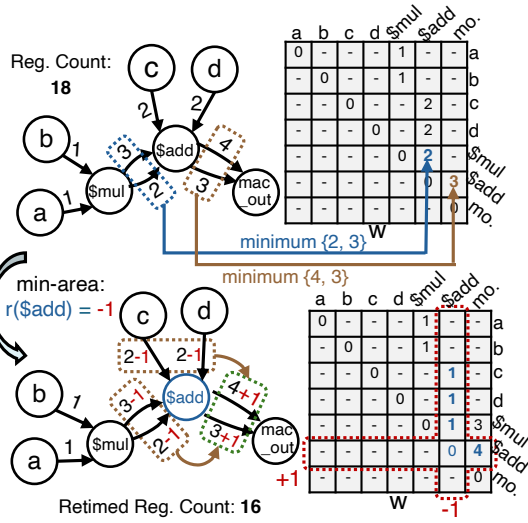


Fig. 10. An illustration of the edge weight selection for creating W matrix and updating $r(\cdot)$.

where $d(i)$ is the combinational delay of node i from the AIG-based delay model, and $w_{min}(i, j)$ is the minimum edge weight (register count) between nodes i and j . A combinational path exists between nodes u and v when $W(u, v) = 0$, meaning no register separates them. Register edges are not treated as zero-delay. Each edge weight $w(e)$ encodes the number of registers in the W and D matrices. Clock-gating and multi-cycle paths are represented as additional timing constraints on the edges. The predicted clock period is the longest such path:

$$T_{pred} = \max_{\{(u,v): W(u,v)=0\}} D(u, v) \quad (9)$$

and the predicted Fmax is $1/T_{pred}$. The operators along this longest path are the critical operators, and their D/C/M classifications identify the bottleneck component type. This analysis produces the timing report: predicted Fmax, the critical path operators, and the bottleneck location.

For optimization (Sections IV-D–IV-E), we identify all critical paths that exceed a target clock period:

$$\mathcal{P}_{critical} = \{(u, v) : W(u, v) = 0 \wedge D(u, v) > T_{target}\} \quad (10)$$

For each path pair (u, v) in $\mathcal{P}_{critical}$, we reconstruct the actual path nodes using breadth-first search. Because the CIR operates at the word level (hundreds of nodes) rather than the gate level (millions of nodes), the Floyd-Warshall computation and path reconstruction complete in seconds.

D. Structural Optimization via Fan-out Management

High fan-out nets are a primary source of combinational delay that limits ASIC performance [77]. Commercial EDA tools address this through load isolation and load splitting (fan-out tree synthesis [78]–[80]), typically using dynamic programming algorithms such as Van Ginneken’s [65], [81], [82] at the gate level. KAIROS adapts these techniques to operate on word-level CIR operators through a dynamic programming-based algorithm that selects appropriate transformations for each node. Figure 8 illustrates both gate-level and CIR-level

variants. KAIROS lifts a number of late-stage optimizations, such as fan-out management and retiming, to the word level. This keeps netlist-level timing optimization traceable to the RTL and enables faster design space exploration. Technology mapping, gate sizing, and Boolean restructuring are still performed in the downstream synthesis flow based on KAIROS-optimized RTL.

a) *Driver Candidate Selection:* From the critical path identified in Section IV-C, candidate nodes for fan-out optimization are selected based on three criteria: (1) a fan-out count exceeding the threshold τ_f (typically 4, following the FO4 convention [83]), (2) location on a critical path, and (3) exclusively combinational connections, meaning this node is not connected to any registers (i.e., all incoming and outgoing edges have weight $w(e) = 0$), which ensures the optimization targets only the combinational delay on the critical path.

b) *Fan-out Classification:* For each candidate driver node, we partition its fan-out set F_{total} into critical and non-critical subsets:

$$F_{crit} = \{f \in F_{total} : \exists (u, v) \in \mathcal{P}_{critical}, W(f, v) = 0\} \quad (11)$$

$$F_{non-crit} = F_{total} \setminus F_{crit} \quad (12)$$

A fan-out node is identified as a critical-path node if it lies on a critical path or feeds into one through a purely combinational connection, i.e., $W(f, v) = 0$ for the endpoint v of some critical pair $(u, v) \in \mathcal{P}_{critical}$ (with $W(v, v) = 0$, this includes the nodes on the path itself).

c) *Dynamic Programming-based Optimization:* Algorithm 1 first identifies the critical pairs and the candidate drivers from W , D , and the target period T (Phase I). For each candidate driver, it then evaluates three options (Phase II): (1) no optimization, (2) **load isolation**, which inserts a buffer to drive non-critical fan-outs and reduces capacitive load on the critical path, and (3) **load splitting**, which duplicates the driver for critical fan-outs while routing non-critical fan-outs through the original driver with buffering. The gain for each option is computed from the driver delay $D[d]$, the fan-out partition sizes, and empirically calibrated coefficients ($\alpha_{iso}=0.2$, $\alpha_c=0.16$, $\alpha_n=0.08$) that capture the relative improvement from load reduction. Because they are ratios of delay improvements rather than absolute delays, these coefficients remain consistent across technology nodes. Each option also has a structural cost, namely the number of nodes it adds (none, one buffer, or a duplicate driver plus a buffer). Because the options of different drivers compete for the same structural budget, the selection is solved as a dynamic program over the candidate drivers and a fan-out budget B (Phase III), which returns the combination of options that minimizes the estimated critical-path delay while adding at most B nodes ($B=20$ in our experiments); a driver is transformed only if its selected option has positive gain. After selection, the CIR is transformed directly: buffer nodes or duplicate drivers are inserted, and edges are rewired while preserving all parallel edge attributes and register weights to

maintain semantic equivalence. Figure 8 (bottom) shows both CIR-level transformations.

E. Sequential Optimization via Retiming

Sequential retiming minimizes clock period through optimal register placement while preserving functional correctness. Building upon Leiserson and Saxe’s foundation [27], [66], we adapt classical retiming algorithms [66], [84]–[88] to operate on CIR’s multigraph structure. In addition, structural fan-out optimization must operate ahead of sequential retiming, as it could potentially add new structural nodes (e.g., duplicated drivers and inserted buffers) that will affect the retiming formulation. Figure 9 illustrates the standard retiming process that KAIROS adapts for CIR.

a) Directed Multigraph Retiming Formulation: Unlike conventional retiming approaches that assume simple directed graphs, CIR is a directed multigraph in which multiple edges may exist between the same pair of nodes, each representing a distinct signal path with potentially different timing characteristics. As shown in Fig. 10, for each node pair (u, v) with multiple parallel edges, the minimum weight edge $w_{\min}(u, v) = \min\{w_i : e_i \in E(u, v)\}$ is selected to construct the constraint matrices, ensuring the retiming formulation remains conservative and preserves the most restrictive timing relationships. Retiming then updates these edge weights throughout the multigraph. For example, when $r(\$add) = -1$, registers migrate from fan-in edges (weight decreased by 1) to fan-out edges (weight increased by 1), effectively shifting pipeline stages across the adder. This selection-and-update procedure prevents unintended register removal on critical control paths, allows optimization on less constrained data paths, and ensures that all edge weights remain non-negative for physical realizability of the retimed circuit.

The retiming function $r(\cdot) : V \rightarrow \mathbb{Z}$ redistributes registers according to the constraint: $w_{\text{new}}(u, v) = w_{\min}(u, v) + r(v) - r(u) \geq 0$. The W and D matrices computed in Section IV-C provide the timing constraints for retiming: $W(u, v)$ gives the minimum register count along any path, and $D(u, v)$ gives the maximum combinational delay along the minimum-weight path. After retiming, the matrices are recomputed to reflect the updated edge weights.

b) Minimum Clock Period Optimization: The core retiming optimization uses binary search over feasible clock periods T and applies Bellman-Ford relaxation to solve the difference-constraint system. For each candidate period T , the modified constraints are constructed as:

$$r(u) - r(v) \leq w_{\min}(u, v) \quad \forall (u, v) \in E \quad (13)$$

$$r(u) - r(v) \leq W(u, v) - 1 \quad \forall (u, v) : D(u, v) > T \quad (14)$$

We incorporate selective backward retiming control through configurable node-level constraints. When backward retiming is disabled, we enforce $r(v) \geq 0$ for all nodes to prevent register movement that could complicate reset logic or violate user-specified pipeline boundaries, essential for maintaining predictable initialization behavior in complex sequential designs.

c) Area Optimization Integration: After minimum period identification, we optionally apply constrained minimum-area optimization using linear programming. A new objective function is required to minimize the total register count as follows:

$$\text{minimize} \quad \sum_{v \in V} (|\text{FI}(v)| - |\text{FO}(v)|) \cdot r(v) \quad (15)$$

where $|\text{FO}(v)|$ and $|\text{FI}(v)|$ represent fan-out and fan-in degrees respectively, and Equation (15) is still subject to Constraints (13) and (14). This formulation enables area-constrained retiming that respects timing constraints derived from the minimum period.

F. CIR-to-RTL Reconstruction

The optimized CIR undergoes a reverse translation back to RTL through our semantic-preserving pipeline: CIR $\rightarrow$ RTLIL $\rightarrow$ RTL. Our backend systematically reconstructs RTLIL cells from CIR nodes by preserving all original operator types, parameter dictionaries, and connection attributes while applying weight changes as explicit register insertions or removals along signal paths. When edge weights increase due to retiming, the implementation generates intermediate pipeline registers with meaningful names (e.g., `mult_result_to_adder_retime_reg_0`) and proper clock domain assignments, ensuring that all timing optimization transformations remain traceable to their algorithmic origins. The reconstruction process adopts Yosys’s RTLIL-to-Verilog backend to generate the final RTL while maintaining complete port connectivity and hierarchical structure integrity through careful signal name preservation mechanisms.

G. Equivalence Checking

Once the RTL design is optimized with KAIROS, it is essential to ensure the functional correctness and equivalence between the optimized and the original RTL. A two-stage verification flow is employed. First, the optimized and original RTL are subjected to randomized simulation using shared testbenches and randomly generated input vectors (e.g., 10K vectors for large designs). We utilize Synopsys’s VCS [89] as the simulation tool. Second, formal equivalence checking is applied. Inspired by prior work [50], [90], [91], we adopt an off-the-shelf formal tool (i.e., VC Formal [92]) to prove the equivalence between the original and the optimized RTL.

V. EVALUATION

A. Experimental Setup

Software and tools. All RTL designs are synthesized using Synopsys Design Compiler NXT (DC) T-2022.03-SP2 with the ASAP7 7 nm predictive process design kit (PDK) [93], following industry-standard flows. Synthesis executes on a Linux server with 503 GB RAM and dual Intel Xeon Gold 6240 CPUs (144 threads total), utilizing up to 16 threads per instance. KAIROS is implemented in Python 3, with its frontend using Yosys [58], [94] to convert RTL into the RTLIL representation. NetworkX [95] is used for CIR graph

TABLE III
BENCHMARK SUITE SPANNING ALL COMPONENT TYPES (D/C/M).
“PARAM.” INDICATES PARAMETERIZABLE DESIGNS SWEEP ACROSS
MULTIPLE CONFIGURATIONS.

Design	Type	Source	Param.
AdderTree (16b FP)	D	HardFloat [97]	pipeline 1–8
DotProduct (16b FP)	D	HardFloat [97]	pipeline 1–8
MulAdd (16b FP)	D	HardFloat [97]	pipeline 1–8
MulAdd_add (16b FP)	D	HardFloat [97]	pipeline 1–8
MulAdd_mul (16b FP)	D	HardFloat [97]	pipeline 1–8
Systolic PE	D	Gemmini [98]	bitwidth 8/16/32
Ibex decoder	C	OpenTitan [99]	—
Priority encoder	C	Custom	$N=4/8/16/32$
Round-robin arbiter	C	Custom	$N=4/8/16/32$
eth_10ge_controller	C	OpenCores [100]	pipeline 1–8
ps2host	C	OpenCores [100]	pipeline 1–8
Register file (param.)	M	OpenTitan [99]	16/32/64 entries
Cache ctrl interface	M	OpenTitan [99]	—
Ibex ALU	D+C	OpenTitan [99]	—
Ibex execute stage	D+C	OpenTitan [99]	—
SERV RISC-V	D+C	SERV [101]	—
Ibex full core	D+C+M	OpenTitan [99]	pipeline, mul type
NVDLA (conv. pipeline)	D+C+M	NVDLA [102]	nv_full
CVA6 (Ariane)	D+C+M	OpenHW [103]	cv64a6, FPU

construction and SciPy [96] for the linear-programming (LP) and mixed-integer linear-programming (MILP) solvers. Our synthesis constraint strategy searches the clock period with an eight-pass binary search that converges within a tolerance of 10 ps. We use the Synopsys Design Compiler *compile_ultra* flow [53] to report area and delay numbers throughout the evaluation.

Benchmark suite. To evaluate KAIROS across datapath, control, and memory components, we assemble multiple designs from several open-source projects, spanning datapath (D), control (C), memory (M), and mixed (D+C) categories (Table III). Type D designs include five 16-bit FPUs from Berkeley HardFloat [97] and a Gemmini systolic PE [98]. Type C designs include the Ibex instruction decoder [99], [104], two OpenCores controllers [100], and parameterizable priority encoder and round-robin arbiter primitives. Type M designs include a parameterizable register file and the OpenTitan cache controller interface [99]. Mixed designs include the Ibex ALU, Ibex execute stage, and the SERV bit-serial RISC-V CPU [101]. The full Ibex core serves as the primary case study for progressive architectural exploration (Section V-D), containing all three component types; its progressive sweep is a separate experiment from the benchmark suite described here. For parameterizable designs, we sweep architectural parameters (pipeline depth, bitwidth, fan-in width, register file entries), yielding 60 distinct design points. We further add two industrial-scale designs, the NVDLA deep-learning accelerator and the CVA6 (Ariane) RISC-V core (Table III), which both include all three component types and bring the suite to 62 design points. In total, Table III lists 19 design families; sweeping their parameters yields the 60 design points used for prediction and optimization (62 with NVDLA and CVA6), while the runtime comparison in Table VI covers 41

distinct RTL designs, including additional OpenTitan IP blocks (e.g., uart, aes, kmac).

Synthesis baseline. Our primary baseline consists of the benchmark designs synthesized using Synopsys Design Compiler’s *compile_ultra* flow [53]. We evaluated several alternative synthesis flows [54], [105], [106], and open-source flows [107]–[109]. *compile_ultra* was selected because it integrates industry-grade aggressive optimizations that directly align with KAIROS’s scope, and because prior work [110] shows that commercial synthesis flows consistently achieve higher Fmax than open-source tools.

Evaluation methodology. To study how timing bottlenecks shift across component types as architectural parameters change, we sweep the Ibex core across three architectural dimensions: pipeline depth (1 to 6 stages; Fig. 13 shows depths 1–4, which suffice to expose the bottleneck transitions), multiplier type (1-cycle fast, 3-cycle slow), and register file entries (16, 32, 64), at three levels of design complexity (datapath only, datapath+control units, full core with memory macro added).

B. Prediction Accuracy

For KAIROS to be useful for DSE, its delay estimates must correlate with post-synthesis results well enough to preserve design rankings. We evaluate prediction quality using Spearman rank correlation (ρ), which measures whether KAIROS preserves the relative ordering of designs, and Mean Absolute Percentage Error (MAPE), which measures the magnitude of prediction deviation.

Figure 11(a) plots KAIROS’s predicted critical path delay against Design Compiler results. Across the 60 design points of the original suite, KAIROS achieves a Spearman rank correlation of $\rho = 0.96$ and MAPE of 14.0% (13.9% across all 62 points including CVA6 and NVDLA, as shown in Fig. 11(a)). Per-type accuracy confirms that the taxonomy-aware delay models generalize across component types: Type D designs achieve $\rho = 0.93$, Type C $\rho = 0.76$, and mixed Type D+C $\rho = 0.93$. The lower correlation for Type C is expected: control primitives have shallow combinational paths where AIG depth differences are small, and the ranking is more sensitive to technology mapping choices. Nonetheless, for architectural DSE the primary requirement is correct bottleneck type identification, i.e., determining whether the datapath, control, or memory component limits frequency at each design point. Section V-D shows that KAIROS correctly identifies these bottleneck transitions across the Ibex design space.

Figure 11(b) isolates the effect of the Type D/C/M taxonomy on prediction quality. When a uniform datapath AIG model is applied to all component types, accuracy degrades notably for Type C designs (ρ drops from 0.76 to 0.55) and mixed Type D+C designs (ρ drops from 0.93 to 0.77), while Type D accuracy remains largely unchanged. This confirms that control primitives (MUX trees, decoders) exhibit fundamentally different AIG depth scaling from arithmetic operators, and that the taxonomy-aware modeling in Section IV-B is a necessary design choice.

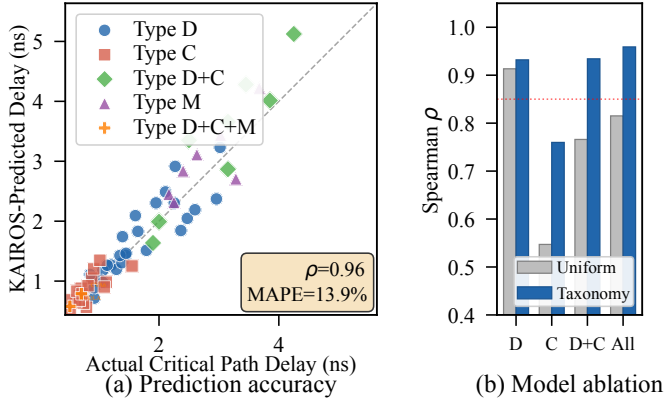


Fig. 11. (a) KAIROS-predicted vs. actual post-synthesis critical path delay. The dashed diagonal represents perfect prediction. (b) Prediction accuracy improvement from taxonomy-aware delay models vs. a uniform AIG model.

Table IV compares KAIROS against published pre-synthesis prediction methods. KAIROS achieves timing accuracy comparable to MasterRTL [43] (MAPE 14% vs. 14%) without requiring any training data. SNS [44] and SNS v2 [45] achieve higher speedups but produce only design-level aggregate metrics. The key difference is prediction granularity: KAIROS identifies which operator and which component type is the timing bottleneck, enabling microarchitectural feedback that design-level predictors cannot provide.

C. RTL Optimization Results

Figure 12 summarizes the optimization results based on the taxonomy. Datapath-dominated designs benefit most, with averaged Fmax improvement of 10.2% (ranging from 7.1% for DotProduct to 13.4% for AdderTree). The gains are highest at mid-to-deep pipeline depths (4–8 stages), where retiming has sufficient register slack to redistribute delay and fan-out management effectively reduces critical path loading. Control units show modest but consistent gains, averaging 2.4% across Type C (2.5% for eth_10ge_controller, 2.7% for ps2host), which aligns with the taxonomy prediction: control paths are inherently shallow (logarithmic AIG depth in fan-in width), leaving limited room for retiming to redistribute delay. Type M designs (register file and cache controller) gain 2.7% on average. The ablation in Fig. 12 confirms that retiming contributes the majority of the 7.1–13.4% gains while fan-out management provides complementary improvements, as global register repositioning is more effective than localized buffering for the deep combinational circuits on datapath. For the optimization study, NVDLA and CVA6 are evaluated as D+C designs, with memory paths excluded, as was done for the Ibex core, although the full designs contain all three component types (Table III). CVA6 achieves a 5.5% Fmax improvement, while the MAC-dominated NVDLA achieves an 8.0% improvement. Both improvements remain below the pure-datapath average because control- and memory-bound critical paths limit further gains.

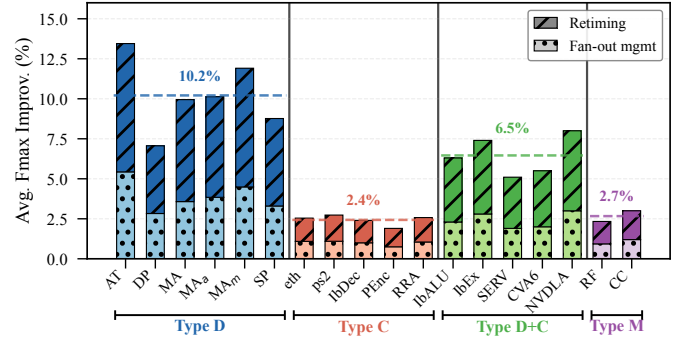


Fig. 12. Fmax improvement (%) organized by component type. Dashed lines indicate per-type averages. AT: AdderTree; DP: DotProduct; MA: MulAdd; MAa/MAm: MulAdd_add/MulAdd_mul; SP: Systolic PE; eth: eth_10ge_controller; ps2: ps2host; IbDec: Ibex decoder; PEnc: priority encoder; RRA: round-robin arbiter; IbALU/IbEx: Ibex ALU/execute stage; RF: register file; CC: cache controller. NVDLA and CVA6 are evaluated as D+C (memory paths excluded).

D. Progressive Architectural Case Study

Figure 13 presents the bottleneck shift map across the design space of the Ibex core. The map shows three distinct regions where different component types constrain Fmax, and the transitions between them depend on both pipeline depth and multiplier configuration. At shallow pipeline depths, the critical path lies exclusively within Type D arithmetic operators. Deeper pipelining improves Fmax consistently (304–840 MHz for the fast multiplier, 393–1018 MHz for the slow multiplier at depths 1–4), because inserted registers reduce the critical path length. KAIROS’s predicted Fmax tracks the post-synthesis ground truth with $\rho = 0.99$ in this region. The Ibex design variant with a slow multiplier achieves higher Fmax at every pipeline depth because its multi-cycle design breaks the multiply operation into shorter combinational segments. As pipeline depth increases, the length of the datapath critical path reduces and the control logic (decode MUX trees, control FSM) becomes the binding constraint. With the fast multiplier configuration, this bottleneck shift (D-to-C) occurs at pipeline depth 3 (604 MHz, control unit dominates). With the slower multiplier, the shift occurs earlier at pipeline depth 2 (629 MHz), because the 3-cycle multiplier already has a shorter critical path, which exposes the timing issue from the control path sooner. Discovering these crossover points would otherwise require running synthesis with Design Compiler for every configuration. When the register file and memory interface are added, Type M constraints dominate at larger configurations. With a 64-entry register file, the memory component is the bottleneck in 11 of the 12 configurations of the full sweep (six pipeline depths $\times$ two multiplier types); the only exception is the fast multiplier at pipeline depth 1, where the datapath still dominates. At pipeline depth 3 with the fast multiplier, Fmax drops progressively from 729 MHz (ALU only, D-bottlenecked) to 604 MHz (+control, C-bottlenecked) to 503 MHz (+RF16, M-bottlenecked) to 317 MHz (+RF64, M-bottlenecked).

Figure 14(a) shows the progressive exploration results, comparing KAIROS predictions against Design Compiler ground

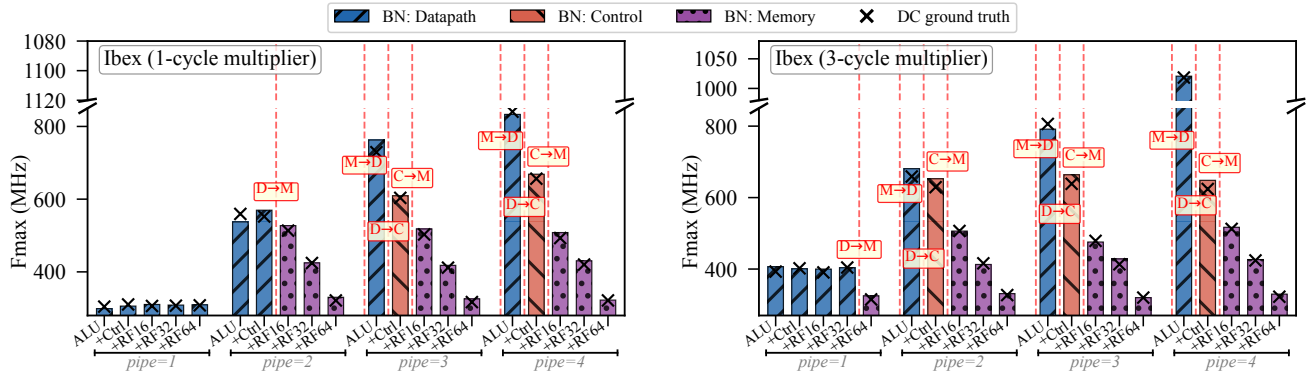


Fig. 13. Timing bottleneck shift map across the Ibex design space. Each bar is colored by the component type (D, C, or M) that determines the critical path. Crossover points (marked with arrows) indicate where the architectural bottleneck transitions between component types. $\times$ markers show post-synthesis Design Compiler Fmax for validation.

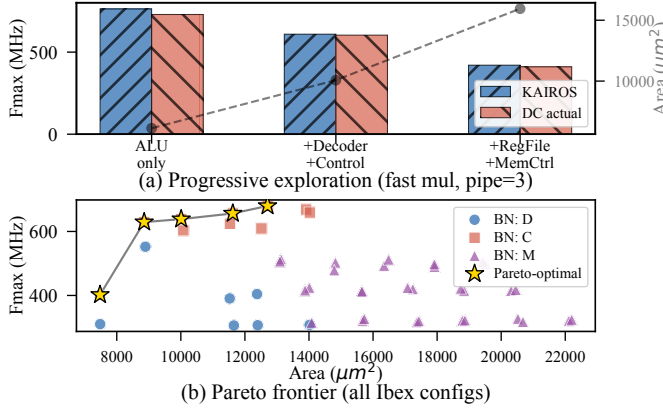


Fig. 14. (a) Fmax comparison: KAIROS prediction vs. DC ground truth across progressive design steps, with area trend on the secondary axis. (b) Pareto frontier of the full Ibex design space. Starred markers indicate Pareto-optimal configurations agreed upon by both KAIROS and DC. truth at each step. KAIROS correctly tracks the Fmax trend as complexity increases ($\rho = 0.99$ across all 60 configurations of the progressive sweep: five integration steps, six pipeline depths, and two multiplier types, of which Fig. 13 shows depths 1–4), and more importantly, correctly identifies at which step the bottleneck type changes. Figure 14(b) shows the Pareto frontier across the Ibex design space. KAIROS and Design Compiler agree on the Pareto-optimal configurations, confirming that the ranking quality demonstrated in Section V-B translates to correct DSE conclusions in practice. An architect using KAIROS can discover these crossover points in a few hours; observing them via synthesis with DC would require days of computation.

E. DSE Speedup Analysis

Figure 15 compares KAIROS’s per-design-point runtime against Design Compiler synthesis across all benchmark designs. Across the original 39-design suite, KAIROS completes timing prediction in 149 seconds per design point on average, while Design Compiler synthesis requires approximately 192 minutes, yielding a $77\times$ wall-clock speedup (ranging from $24\times$ for small datapath modules to $128\times$ for large SoC IP blocks). The speedup scales favorably with design complexity: larger designs that take longer to synthesize benefit from

TABLE IV
COMPARISON WITH PRIOR ML-BASED PRE-SYNTHESIS PREDICTION METHODS. EACH METHOD’S NUMBERS ARE AS PUBLISHED ON ITS OWN TECHNOLOGY NODE AND BENCHMARK SET; THE KAIROS ROW IS OUR MEASUREMENT ON ASAP7 (7 nm). MASTERRTL REPORTS PEARSON R ; KAIROS REPORTS SPEARMAN ρ .

Method	Tech Node	Timing Acc.	Training Data	Speedup	Bottleneck ID
SNS [44]	15 nm	MAPE=38%	Required	$760\times$	$\times$
SNS v2 [45]	32 nm	MAPE $\approx 17\%$	Required	$411\times$	$\times$
MasterRTL [43]	45 nm	$R=0.93$, MAPE=14%	Required	$\sim 20\times$	$\times$
KAIROS	7 nm	$\rho=0.96$, MAPE=13.9%	None	$113\times$	D/C/M

TABLE V
DSE CONCLUSION AGREEMENT BETWEEN KAIROS AND DESIGN COMPILER. THE TYPE D AND TYPE D+C SUBSETS YIELD IDENTICAL VALUES.

Metric	Type D	Type D+C	Overall
Spearman ρ	0.93	0.93	0.96
Pareto agreement	5/5	5/5	5/5
Top-3 match	2/3	2/3	2/3

proportionally greater speedup, because KAIROS’s runtime is dominated by the W&D matrix computation and retiming optimization, both of which scale more favorably than gate-level synthesis. Table V evaluates whether KAIROS’s speedup comes at the cost of DSE accuracy. KAIROS identifies all 5 Pareto-optimal design configurations found by full synthesis, and the top-3 ranked designs match in 2 out of 3 cases. This confirms that KAIROS’s ranking quality is sufficient to guide architects to the same Pareto-optimal design choices as synthesis, at a fraction of the computational cost.

Industrial-scale validation. We add two industrial designs: the NVDLA deep-learning accelerator (13.7M gates) and the CVA6/Ariane 64-bit core (340K gates), which are one to two orders of magnitude larger than the Ibex core. After including both large designs, the rank correlation remains $\rho=0.96$, with 13.9% MAPE across 62 design points; KAIROS predicts CVA6 and NVDLA within 10.7% and 11.5%, respectively. The speedup also increases with design scale. Achieving a target Fmax requires an iterative clock-period search that repeatedly re-runs synthesis until timing closure is reached; in our setup, this corresponds to an eight-pass binary search over the clock period. As a result, Design Compiler requires 28.6 hours for CVA6 and 4.4 days for NVDLA, whereas

TABLE VI
KAIROS RUNTIME VS. DESIGN COMPILER SYNTHESIS ACROSS
COMPONENT TYPES.

Type	Design	Gates	KAIROS (s)	DC (min)	Speedup
D	AdderTree	~4.5K	115.2	46.2	24×
D	SystolicPE-32b	~7.2K	74.6	53.3	43×
C	IbexDecoder	~8.2K	71.8	55.0	46×
C	uart	~9.6K	84.7	54.8	39×
C	eth_10ge	~12.5K	124.3	90.0	43×
D+C	IbexExec	~9.8K	148.5	180.0	73×
D+C	IbexALU	~32K	302.4	480.0	95×
M	RegFile-32	~3.2K	37.8	35.0	56×
M	CacheCtrl	~5.5K	61.4	48.3	47×
D+C+M	aes	~89K	335.7	530.1	95×
D+C+M	kmac	~138K	418.5	717.0	103×
D+C+M	Ibex_SoC	~207K	498.2	1060.5	128×
D+C+M	CVA6 (cv64a6)	~340K	650.0	1718.0	159×
D+C+M	NVDLA (nv_full)	~13.7M	1800.0	6382.9	213×
Average (all 41)			201	380	113×

KAIROS analyzes each design in a single word-level pass, taking only 11 and 30 minutes, respectively. This corresponds to 159× and 213× speedups. Adding these two industry-scale designs raises the maximum speedup from 128× to 213× and the suite-average speedup from 77× to 113×. KAIROS runtime scales with word-level CIR complexity rather than raw gate count: although NVDLA has roughly 40× more gates than CVA6, its highly replicated MAC array keeps the analysis time to 30 minutes, well within the minutes-scale budget required for DSE.

F. Area Overhead and Runtime Analysis

KAIROS’s RTL optimization introduces an average area overhead of 9.4% across the optimized component designs. Fan-out management contributes 3.0–4.5% overhead through buffer insertion, while retiming adds 4.4–6.8% through register repositioning and occasional replication. These costs are acceptable for early-stage DSE, where the Fmax improvements of 7.1–13.4% substantially outweigh the area penalty. Table VI shows representative runtimes across all component types. The average total runtime (149 seconds per design point across the original suite; 201 seconds including CVA6 and NVDLA) scales with design complexity: most Type D and Type M designs complete in about two minutes or less (38–116 seconds), while the large SoC IP (89K–207K gates) takes 336–498 seconds. Internally, the runtime is dominated by structural optimization (28.6%), W&D matrix computation (28.7%), and sequential optimization (33.6%), with frontend and backend transformations negligible (<0.2% combined) and the remaining ~9% spent on setup and bookkeeping between phases. Further speedup is achievable through incremental W&D matrix updates when sweeping related design points, which we leave to future work.

Limitations. KAIROS models logic delay via AIG depth and does not model interconnect delay or technology mapping effects. For pre-synthesis exploration, logic depth is the dominant factor in relative Fmax ranking, as validated by our rank correlation results. Following the same evaluation methodology as SNS [44] and MasterRTL [43], we validate against post-synthesis timing from Design Compiler. Inter-

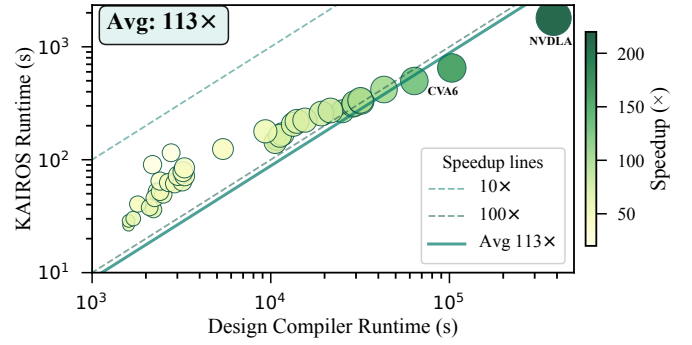


Fig. 15. KAIROS runtime vs. Design Compiler runtime on log-log scale. Bubble size indicates design complexity (gate count). Dashed diagonals mark 10×, 100× speedup.

connect and routing delays are accurately determined only after physical layout, i.e., place-and-route, and are therefore beyond the scope of KAIROS’s pre-synthesis analysis. KAIROS instead provides early Fmax ranking to guide design space exploration, complementing rather than replacing place-and-route timing closure. The AIG-based structural delay model is technology-independent; converting to absolute delay units requires a per-technology calibration factor (Section IV-B). ABC’s transformations are deterministic, ensuring reproducible AIG depth. Extending to additional technology nodes requires only re-fitting the calibration scalar, not re-training a model. We verify this across nodes: re-synthesizing 15 representative designs at NanGate 45 nm and refitting only τ preserves the Fmax ranking at $\rho=0.91$ (versus 0.96 on ASAP7), confirming that the single-scalar recalibration transfers across a large technology gap, from 7 nm to 45 nm. Moreover, KAIROS predicts timing, not power. Power-aware DSE can incorporate KAIROS alongside tools like McPAT [36]. KAIROS’s optimization operates at the RTL word level and is complementary to technology-aware gate-level optimization in commercial synthesis flows.

VI. CONCLUSION

We presented KAIROS, a pre-synthesis timing prediction and optimization framework for fast microarchitectural design space exploration. KAIROS operates on a Circuit Intermediate Representation that preserves RTL operators and signal names, enabling operator-level critical path analysis without gate-level synthesis. A component taxonomy assigns structurally derived delay models to datapath, control, and memory modules, making timing bottleneck shifts visible as architectural parameters change. Across 62 design points, from the Ibex RISC-V core to industrial-scale designs (NVDLA and CVA6), KAIROS preserves post-synthesis Fmax rankings with Spearman $\rho=0.96$, achieves 7.1–13.4% frequency improvement on datapath circuits, and completes timing evaluation 113× faster than synthesis on average across 41 designs, up to 213×. A progressive case study on Ibex confirms that KAIROS correctly identifies bottleneck transitions across component types at different pipeline depths, providing architectural insights previously available only after days of synthesis.

REFERENCES

- [1] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omer-nick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th annual international symposium on computer architecture (ISCA)*, 2017, pp. 1–12.
- [2] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, Adam Hutchin, Utku Diril, Krishnakumar Nair, Ehsan K. Ardestani, Martin Schatz, Yuchen Hao, Rakesh Komuravelli, Kunming Ho, Sameer Abu Asal, Joe Shajrawi, Kevin Quinn, Nagesh Sreedhara, Pankaj Kansal, Willie Wei, Dheepak Jayaraman, Linda Cheng, Pritam Chopda, Eric Wang, Ajay Bikumandla, Arun Karthik Sengottuvel, Krishna Thottempudi, Ashwin Narasimha, Brian Dodds, Cao Gao, Jiyuan Zhang, Mohammad Al-Sanabani, Ana Zehtabioskui, Jordan Fix, Hangchen Yu, Richard Li, Kaustubh Gondkar, Jack Montgomery, Mike Tsai, Saritha Dwarakapuram, Sanjay Desai, Nili Avidan, Poorvaja Ramani, Karthik Narayanan, Ajit Mathews, Sethu Gopal, Maxim Naumov, Vijay Rao, Krishna Noru, Harikrishna Reddy, Prahlad Venkatapuram, and Alexis Bjorlin, “MTIA: First generation silicon targeting Meta’s recommendation systems,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–13.
- [3] Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, and Antonino Tumeo, “Bambu: an open-source research framework for the high-level synthesis of complex applications,” in *Proceedings of 2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1327–1330.
- [4] Satnam Singh and David J Greaves, “Kiwi: Synthesis of FPGA circuits from parallel programs,” in *Proceedings of 2008 16th International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2008, pp. 3–12.
- [5] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzyniek, and Krste Asanović, “Chisel: constructing hardware in a Scala embedded language,” in *Proceedings of the 49th annual Design Automation Conference (DAC)*, 2012, pp. 1216–1225.
- [6] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić, “Chippyard: Integrated design, simulation, and implementation framework for custom SoCs,” *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.
- [7] Rishiyur Nikhil, “Bluespec System Verilog: efficient, correct RTL from high level specifications,” in *Proceedings of Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE’04.* IEEE, 2004, pp. 69–70.
- [8] Shunning Jiang, Peitian Pan, Yanghui Ou, and Christopher Batten, “PyMTL3: A Python framework for open-source hardware modeling, generation, simulation, and verification,” *IEEE Micro*, vol. 40, no. 4, pp. 58–66, 2020.
- [9] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie, “RTLML: An open-source benchmark for design RTL generation with large language model,” in *Proceedings of 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.
- [10] Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany, “Revisiting VerilogEval: A year of improvements in large-language models for hardware code generation,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 30, no. 6, pp. 1–20, 2025.
- [11] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg, “VeriGen: A large language model for Verilog code generation,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.
- [12] Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie, “RTLCode: Fully open-source and efficient LLM-assisted RTL code generation technique,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1448–1461, 2024.
- [13] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce, “Chip-Chat: Challenges and opportunities in conversational hardware design,” in *Proceedings of 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.
- [14] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri, “AutoChip: Automating HDL generation using LLM feedback,” *arXiv preprint arXiv:2311.04887*, 2023.
- [15] Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li, “ChipGPT: How far are we from natural language hardware design,” *arXiv preprint arXiv:2305.14019*, 2023.
- [16] Zehua Pei, Huiling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu, “BetterV: Controlled Verilog generation with discriminative guidance,” in *Proceedings of Forty-first International Conference on Machine Learning (ICML)*, 2024.
- [17] Manar Abdelatty, Maryam Nouh, Jacob K Rosenstein, and Sherief Reda, “Pluto: A benchmark for evaluating efficiency of LLM-generated hardware code,” *arXiv preprint arXiv:2510.14756*, 2025.
- [18] Pengwei Jin, Di Huang, Chongxiao Li, Shuyao Cheng, Yang Zhao, Xinyao Zheng, Jiaguo Zhu, Shuyi Xing, Bohan Dou, Rui Zhang, Zidong Du, Qi Guo, and Xing Hu, “RealBench: Benchmarking Verilog generation models with real-world IP designs,” *arXiv preprint arXiv:2507.16200*, 2025.
- [19] Kimia Tasnia, Alexander Garcia, Tasnuva Farheen, and Sazadur Rahman, “VeriOpt: PPA-aware high-quality Verilog generation via multi-role LLMs,” in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2025, pp. 1–9.
- [20] Yiting Wang, Wanghao Ye, Ping Guo, Yexiao He, Ziyao Wang, Bowei Tian, Shwai He, Guoheng Sun, Zheyu Shen, Sihan Chen, Ankur Srivastava, Qingfu Zhang, Gang Qu, and Ang Li, “SymRTLO: Enhancing RTL code optimization with LLMs and neuron-inspired symbolic reasoning,” *Advances in Neural Information Processing Systems*, vol. 38, pp. 50 093–50 118, 2025.
- [21] Andrew B Kahng, “New game, new goal posts: A recent history of timing closure,” in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 2015, pp. 1–6.
- [22] Sneha Saurabh, Hitarth Shah, and Shivendra Singh, “Timing closure problem: Review of challenges at advanced process nodes and solutions,” *IETE Technical Review*, 2019.
- [23] Rushabh Shah, Krishna Agrawal, G Anjaneyulu, and Vishnu Bhaskari, “Automated debugger for optimum physical clock structure targeting minimal latency,” in *Proceedings of 2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*. IEEE, 2022, pp. 80–85.
- [24] Vivek Bhardwaj, “Shift left trends for design convergence in SoC: An EDA perspective,” *International Journal of Computer Applications*, vol. 174, no. 16, pp. 22–27, 2021.
- [25] Tianhao Yang, Zhenyu Zhao, Ao Han, and Guoqiang Liu, “Automatic timing ECO using stage-based path delay prediction,” in *Proceedings of 2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 2022, pp. 455–459.
- [26] Linyu Zhu and Xinfei Guo, “Delay-driven physically-aware logic synthesis with informed search,” in *Proceedings of 2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 327–335.
- [27] Charles E Leiserson, Flavio M Rose, and James B Saxe, “Optimizing synchronous circuitry by retiming (preliminary version),” in *Proceedings of Third Caltech conference on very large scale integration (VLSI)*. Springer, 1983, pp. 87–116.

- [28] Narendra Shenoy and Richard Rudell, "Efficient implementation of retiming," in *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*. Springer, 2003, pp. 615–630.
- [29] Charles Alpert and Anirudh Devgan, "Wire segmenting for improved buffer insertion," in *Proceedings of the 34th Annual Design Automation Conference (DAC)*, 1997, pp. 588–593.
- [30] William J Dally and Andrew Chang, "The role of custom design in ASIC chips," in *Proceedings of the 37th annual design automation conference (DAC)*, 2000, pp. 643–647.
- [31] Robert Brummayer and Armin Biere, "Boolelector: An efficient SMT solver for bit-vectors and arrays," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2009, pp. 174–177.
- [32] João P Marques-Silva and Karem A Sakallah, "Boolean satisfiability in electronic design automation," in *Proceedings of the 37th Annual Design Automation Conference (DAC)*, 2000, pp. 675–680.
- [33] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
- [34] Jian Weng, Boyang Han, Derui Gao, Ruijie Gao, Wanning Zhang, An Zhong, Ceyu Xu, Jihao Xin, Yangzhixin Luo, Lisa Wu Wills, and Marco Canini, "Assassyn: A unified abstraction for architectural simulation and implementation," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2025, pp. 1464–1479.
- [35] Rishov Sarkar and Cong Hao, "OmniSim: Simulating hardware with c speed and RTL accuracy for high-level synthesis designs," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2025, pp. 1334–1346.
- [36] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture (MICRO)*, 2009, pp. 469–480.
- [37] Naveen Muralimanohar, Rajeev Balasubramanian, and Norman P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.
- [38] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 97–108, 2014.
- [39] Weilong Cui, Yongshan Ding, Deeksha Dangwal, Adam Holmes, Joseph McMahan, Ali Javadi-Abhari, Georgios Tzimpragos, Frederic Chong, and Timothy Sherwood, "Charm: a language for closed-form high-level architecture modeling," in *Proceedings of 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 152–165.
- [40] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *Proceedings of 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.
- [41] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu, Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust, and Vijay Janapa Reddi, "ArchGym: An open-source gymnasium for machine learning assisted architecture design," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–16.
- [42] Charles Hong, Qijing Huang, Grace Dinh, Mahesh Subedar, and Yakun Sophia Shao, "DOSAs: Differentiable model-based one-loop search for DNN accelerators," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 209–224.
- [43] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie, "MasterRTL: A pre-synthesis PPA estimation framework for any RTL design," in *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [44] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills, "SNS's not a synthesizer: a deep-learning-based synthesis predictor," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 847–859.
- [45] Ceyu Xu, Pragma Sharma, Tianshu Wang, and Lisa Wu Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 167–179.
- [46] Yikang Ouyang, Sicheng Li, Dongsheng Zuo, Hanwei Fan, and Yuzhe Ma, "ASAP: Accurate synthesis analysis and prediction with multi-task learning," in *Proceedings of 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.
- [47] Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie, "Annotating slack directly on your Verilog: Fine-grained RTL timing evaluation for early optimization," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.
- [48] Daniela Sánchez Lopera, Robert Niklas Kunzelmann, Endri Kaja, and Wolfgang Ecker, "Fake timer: An engine for accurate timing estimation in register transfer level designs," in *Proceedings of 2024 25th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2024, pp. 1–8.
- [49] Sanjay Gandham, Joe Walston, Sourav Samanta, Lingxiang Yin, Hao Zheng, Mingjie Lin, and Stelios Diamantidis, "CircuitSeer: RTL Post-PnR delay prediction via coupling functional and structural representation," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024, pp. 1–9.
- [50] Samuel Coward, Theo Drane, and George A Constantinides, "ROVER: RTL optimization via verified e-graph rewriting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 12, pp. 4687–4700, 2024.
- [51] Licheng Guo, Yuze Chi, Jie Wang, Jason Lau, Weikang Qiao, Ecenur Ustun, Zhiru Zhang, and Jason Cong, "AutoBridge: Coupling coarse-grained floorplanning and pipelining for high-frequency HLS design on multi-die FPGAs," in *Proceedings of The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 81–92.
- [52] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson, "A compiler infrastructure for accelerator generators," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 804–817.
- [53] Synopsys, Inc., "Design Compiler user guide," Sunnyvale, CA, USA, 2022.
- [54] Cadence, "Genus Synthesis Solution," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, 2025.
- [55] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie, "Transferable presynthesis PPA estimation for RTL designs with data augmentation techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 1, pp. 200–213, 2024.
- [56] Samuel Coward, Theo Drane, and George A Constantinides, "Constraint-aware e-graph rewriting for hardware performance optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.
- [57] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović, "FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," in *Proceedings of 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.
- [58] Clifford Wolf, "Yosys manual: RTLIL specification," <https://yosyshq.readthedocs.io/projects/yosys/en/stable/>, 2024.
- [59] Pong P Chu, *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. John Wiley & Sons, 2006.
- [60] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Haoyang Zhang, Andrew Quinn, and Baris Kasikci, "Debugging in the brave new world of re-configurable hardware," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, pp. 946–962.
- [61] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.
- [62] Chin Ngai Sze, Charles J Alpert, Jiang Hu, and Weiping Shi, "Path based buffer insertion," in *Proceedings of the 42nd annual Design Automation Conference (DAC)*, 2005, pp. 509–514.

- [63] Huan Ren and Shantanu Dutt, "Algorithms for simultaneous consideration of multiple physical synthesis transforms for timing closure," in *Proceedings of 2008 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2008, pp. 93–100.
- [64] Wei Chen, Cheng-Ta Hsieh, and Massoud Pedram, "Simultaneous gate sizing and fanout optimization," in *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2000, pp. 374–378.
- [65] Lukas PPP Van Ginneken, "Buffer placement in distributed rc-tree networks for minimal Elmore delay," in *Proceedings of 1990 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1990, pp. 865–868.
- [66] Charles E Leiserson and James B Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.
- [67] Andrew V. Goldberg and Tomasz Radzik, "A heuristic improvement of the Bellman-Ford algorithm," *Applied Mathematics Letters*, vol. 6, no. 3, pp. 3–6, 1993.
- [68] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Proceedings of 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 209–216.
- [69] Amirali Sharifian, Reza Hojabr, Navid Rahimi, Sihao Liu, Apala Guha, Tony Nowatzki, and Arrvinth Shriraman, "μIR—an intermediate representation for transforming and optimizing the microarchitecture of application accelerators," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, pp. 940–953.
- [70] TI Kirkpatrick and Norman Ross Clark, "PERT as an aid to logic design," *IBM journal of Research and Development*, vol. 10, no. 2, pp. 135–141, 1966.
- [71] Norman P Jouppi, "Timing analysis and performance improvement of MOS VLSI designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 4, pp. 650–665, 1987.
- [72] Ahmed Agiza, Rajarshi Roy, Teodor Dumitru Ene, Saad Godil, Sherief Reda, and Bryan Catanzaro, "GraphSyM: Graph physical synthesis model," in *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [73] Daniela Sánchez Lopera and Wolfgang Ecker, "Applying GNNs to timing estimation at RTL," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–8.
- [74] Robert Brayton and Alan Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proceedings of Computer Aided Verification: 22nd International Conference (CAV), Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.
- [75] Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*. Springer Science & Business Media, 1984, vol. 2.
- [76] William H Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.
- [77] Yacoub M El-Ziq, "Logic design automation of MOS combinational networks with fan-in, fan-out constraints," in *Proceedings of 15th Design Automation Conference (DAC)*. IEEE, 1978, pp. 240–249.
- [78] Hirendu Vaishnav and Massoud Pedram, "Routability-driven fanout optimization," in *Proceedings of the 30th international Design Automation Conference (DAC)*, 1993, pp. 230–235.
- [79] Ulrich Brenner and Anna Silvanus, "Delay optimization of combinational logic by and-or path restructuring," in *Proceedings of 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 403–409.
- [80] Ghasem Pasandi and Massoud Pedram, "PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 4, pp. 1–14, 2019.
- [81] Weiping Shi and Zhuo Li, "A fast algorithm for optimal buffer insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 879–891, 2005.
- [82] Jason Jingsheng Cong and Kwok-Shing Leung, "Optimal wiresizing under Elmore delay model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 321–336, 1995.
- [83] Neil H. E. Weste and David Money Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley, 2011.
- [84] Xun Liu, Marios C Papaefthymiou, and Eby G Friedman, "Retiming and clock scheduling for digital circuit optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 2, pp. 184–203, 2002.
- [85] Narendra Shenoy, "Retiming: Theory and practice," *Integration*, vol. 22, no. 1-2, pp. 1–21, 1997.
- [86] Aaron P Hurst, Alan Mishchenko, and Robert K Brayton, "Fast minimum-register retiming via binary maximum-flow," in *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*. IEEE, 2007, pp. 181–187.
- [87] —, "Scalable min-register retiming under timing and initializability constraints," in *Proceedings of the 45th annual Design Automation Conference (DAC)*, 2008, pp. 534–539.
- [88] Peichen Pan, "Continuous retiming: Algorithms and applications," in *Proceedings of International Conference on Computer Design VLSI in Computers and Processors (VLSI)*. IEEE, 1997, pp. 116–121.
- [89] Synopsys, "VCS: Functional verification solution," <https://www.synopsys.com/verification/simulation/vcs.html>, 2024.
- [90] Alfred Koelbl, Reily Jacoby, Himanshu Jain, and Carl Pixley, "Solver technology for system-level to RTL equivalence checking," in *Proceedings of 2009 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2009, pp. 196–201.
- [91] Bin Xue, Prosenjit Chatterjee, and Sandeep K Shukla, "Simplification of C-RTL equivalent checking for fused multiply add unit using intermediate models," in *Proceedings of 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2013, pp. 723–728.
- [92] Synopsys, "VC Formal: Formal verification solution," <https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html>, 2024.
- [93] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandrasekaran Ramamurthy, and Greg Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
- [94] Clifford Wolf and Johann Glaser, "Yosys—a free Verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, vol. 97, 2013, pp. 1–6.
- [95] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy)*, 2008, pp. 11–15.
- [96] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
- [97] John R. Hauser, "Berkeley HardFloat: An open-source hardware library for floating-point arithmetic," <https://www.jhauser.us/arithmetic/HardFloat.html>, 2024.
- [98] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proceedings of 2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.
- [99] lowRISC contributors, "OpenTitan: Open source silicon root of trust," *lowRISC*, 2024, <https://opentitan.org>.
- [100] OpenCores, "OpenCores: Open source hardware community," <https://www.opencores.org>, 2024.
- [101] Olof Kindgren, "SERV – the serial RISC-V CPU," 2019, <https://github.com/olofk/serv>.
- [102] NVIDIA, "NVDLA: NVIDIA deep learning accelerator (open-source hardware)," <http://nvidia.org>, 2018, hardware RTL repository: <https://github.com/nvidia/hw>.

- [103] Florian Zaruba and Luca Benini, “The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm fdsoi technology,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019.
- [104] lowRISC contributors, “Ibex: A small 32-bit RISC-V core,” lowRISC, Tech. Rep., 2024, <https://github.com/lowRISC/ibex>.
- [105] Synopsys, “Fusion Compiler: RTL-to-GDSII design solution,” <https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html>, 2024.
- [106] Siemens EDA, “Oasys-RTL,” <https://eda.sw.siemens.com/en-US/ic/oasys-rtl/>, 2024.
- [107] Ahmed Ghazy and Mohamed Shalan, “OpenLane: The open-source digital ASIC implementation flow,” in *Workshop on Open-Source EDA Technology (WOSET)*, 2020.
- [108] Open Circuit Design, “Qflow 1.3,” <http://opencircuitdesign.com/qflow/welcome.html>, 2024.
- [109] LIP6, Sorbonne Université, “Coriolis VLSI CAD tools,” <https://coriolis.lip6.fr>, 2024.
- [110] Christian Lück, Daniela Sánchez Lopera, Sven Wenzek, and Wolfgang Ecker, “Industrial experience with open-source EDA tools,” in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2022, pp. 143–143.