Escaping Flatland: A Placement Flow for Enabling 3D FPGAs

Cong Hao
Georgia Tech
Atlanta, GA, USA
callie.hao@gatech.edu

Andrew B. Kahng
UC San Diego
San Diego, CA, USA
abk@ucsd.edu

Bodhisatta Pramanik
UC San Diego
San Diego, CA, USA
bopramanik@ucsd.edu

Ismael Youssef*
Georgia Tech
Atlanta, GA, USA
ismael.youssef@gatech.edu

Abstract

3D field-programmable gate arrays (FPGAs) promise higher performance through vertical integration. However, existing placement tools, largely inherited from 2D frameworks, fail to capture the unique delay characteristics and optimization dynamics of 3D fabrics. We introduce a 3D FPGA placement flow that integrates partitioning-based initialization, adaptive cost scheduling, refined delay estimation, and a simulated annealing move set — all targeted at 3D FPGA architecture. Together, these enhancements improve timing estimates and the exploration of *layer* assignments during placement. Compared to Verilog-To-Routing (VTR), our experiments show geometric-mean (max) critical-path delay reductions of ~3% (~7%), ~2% (~4%), ~3% (~8%), and ~6% (~18%) for four 3D architectures: *3D CB, 3D CB-O, 3D CB-I, and 3D SB*, respectively. We also achieve geometric-mean (max) routed wirelength reductions of ~1% (~3%), ~2% (~8%), < 1% (~5%), and ~5% (~10%), respectively. Our work will be permissively open-sourced on GitHub.

ACM Reference Format:
Cong Hao, Andrew B. Kahng, Bodhisatta Pramanik, and Ismael Youssef. 2026. Escaping Flatland: A Placement Flow for Enabling 3D FPGAs. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804424

1 Introduction

FPGAs are central to modern computing, and enable domain-specific acceleration in applications ranging from AI and cloud services to medical imaging and wireless communication [4–6]. As computational demands grow, improving FPGA efficiency—e.g., computation density, total compute capacity, and achievable clock frequency—has become increasingly important. 3D integration offers a compelling path forward. By vertically stacking logic layers, 3D FPGAs can reduce wirelength and delay, increase logic density, and alleviate congestion and I/O limitations [3, 7, 8]. Recent advances in monolithic integration, Through Silicon Vias (TSVs), and hybrid bonding have made such architectures practically realizable [9, 10, 14]. However, Computer-Aided Design (CAD) support for 3D FPGAs remains limited. Existing 3D placement approaches [16–18] do not accurately model inter-layer delay, and perform only limited inter-layer placement exploration, as emphasized by [15]. To address these gaps, this work introduces a 3D FPGA placement flow that explicitly models inter-layer delay, has a tailored move set for 3D placement, and optimizes a dedicated placement cost for 3D FPGAs. The key contributions of our work are as follows.

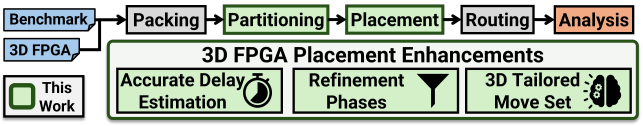


Figure 1: Overview of our 3D FPGA placement flow.

Open-source 3D FPGA placement flow. We propose a 3D FPGA placement flow (Figure 1) that generates high-quality placement solutions across multiple 3D architectures. In particular, we test our flow on the *3D connection box (CB), 3D CB-O, 3D CB-I*, and *3D switch box (SB)* architectures defined using the open-source 3D FPGA architecture exploration framework, LaZagna [15], as well as hybrid variants of these. Our flow is integrated into the open-source FPGA implementation framework VTR [26]. At its core, the flow leverages the hypergraph partitioner *TritonPart* [2, 39] to perform layer *assignments*, and augments VTR's placement algorithm with 3D-specific operators (move set) (Section 4). Our implementation is available on GitHub under a permissive open-source license [40].

3D placement enhancements. We introduce several enhancements to VTR to improve 3D placement quality and enable broader 3D architecture exploration. First, *TritonPart*'s layer assignment serves as the initial solution for VTR's simulated annealing (SA)-based placer. Second, we propose a delay model that accurately captures vertical interconnect delay, providing stronger timing guidance during 3D placement. Third, we add refinement phases during 3D placement: in early iterations, wirelength and congestion are prioritized and inter-layer moves are penalized to preserve layer structure, while in later iterations inter-layer penalties are relaxed and timing is prioritized to optimize critical paths across layers. Finally, we extend VTR's placement move operators to enable better layer exploration during 3D placement (Section 4).

Evaluation and architecture exploration. We evaluate our proposed flow on the Koios design suite [1] across representative 3D FPGA architectures proposed by LaZagna [15]. Experimental results demonstrate that compared to the default VTR flow, our flow achieves geometric-mean (max) critical-path delay reductions of up to ~6% (~18%). We also obtain geometric-mean (max) routed wirelength reductions of up to ~5% (~10%) (Section 5). Routability studies show that with our enhancements, the flow can find lower minimum channel widths than the default VTR implementation (Section 4). Moreover, our flow enables more accurate and effective architecture exploration than prior work [15, 26]. Finally, our studies show that (i) our flow can be used for architectural and performance exploration for 3D FPGA designs, and (ii) some 3D architectures previously considered underperforming relative to other 3D variants can in fact significantly outperform them when evaluated with our framework.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 presents background on 3D FPGA architectures. Section 4 describes our proposed approach. Section 5 presents experimental results. Section 6 concludes the paper.

*Lead author.

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804424

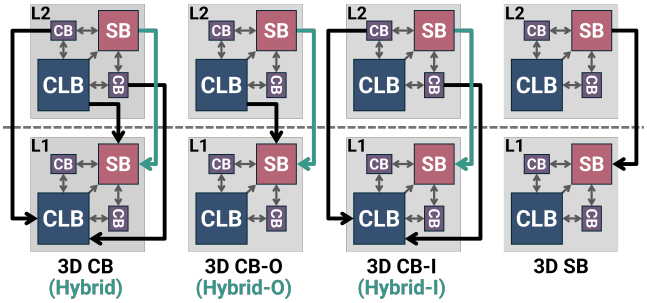


Figure 2: 3D FPGA architectures based on connection types. Connections are only shown from layer "L2" to layer "L1". Green arrows show hybrid variations of each architecture.

2 Related work

FPGA placement has been extensively studied for 2D fabrics [13, 19–21]. In contrast, 3D FPGA placement remains relatively underexplored. Existing works can be broadly grouped into two categories.

Layer assignment via partitioning. This strategy decomposes the placement problem into two stages: first, a layer-assignment phase that partitions the netlist into the target number of layers; and second, a conventional 2D placement flow applied independently within each layer [17, 18, 22–24]. However, these works are evaluated primarily on small benchmark circuits and simpler *homogeneous* FPGA fabrics that lack hard blocks[1] such as DSPs and BRAMs. As a result, they are not directly applicable to modern *heterogeneous* FPGA architectures that combine *configurable logic blocks* (CLBs) with embedded memory and DSP resources, and cannot be compared with this work. Moreover, with the exception of [18], all previous implementations remain closed-source.

Extending "2D" cost function and move space. A more recent line of research augments conventional 2D placement by introducing a z-coordinate into the cost and delay models [27], effectively treating the "third dimension" as an additional spatial axis. [27] integrates such 3D support into the VTR framework, extending its wirelength and timing models to capture inter-layer connectivity[2] while retaining VTR's 2D annealing schedule and move set.

In this work, we combine the benefits of both categories. We use a *partitioning-based initial layer assignment*, but do not restrict logic blocks to their assigned layers. Instead, blocks are allowed to move across layers under a dynamically adaptive cost function that preserves the initial layer structure (i.e., partitioning-based layer assignment) in early placement iterations and enables timing-driven optimization across layers in later iterations (Section 4.1).

3 3D FPGA Architectures

3D FPGAs can be architected in different ways depending on how logic blocks and routing resources are distributed across layers and how vertical connections are inserted between them. Prior architectural studies [27–29] have explored a range of inter-layer connectivity patterns and shown that these choices strongly affect routability and timing. Amagasaki et al. [28] propose architectures where logic blocks have dedicated inter-layer pins; Boutros et al. [27] examine architecture variants that restrict vertical connections to logic block outputs; and Le et al. [29] investigate switch-block-based vertical routing schemes. More recently, Youssef et al. [15] perform a comprehensive sweep across different 3D connectivity models and evaluate their benefits. To capture this diversity, and to build on the architectures established by LaZagna [15], we consider the same four representative architecture types (Figure 2): (i) *3D CB:* inter-layer connections are made through *input and output pins* of logic blocks; (ii) *3D CB-O:* inter-layer connections are made through *output pins* of logic blocks; (iii) *3D CB-I:* inter-layer connections are made through *input pins* of logic blocks; and (iv) combinations of *3D SB* connectivity with a *3D CB*-based architecture (*3D CB-O, 3D CB-I, 3D CB*), where inter-layer connections are made through both *logic block pins and SB resources* (green arrows in Figure 2).

The above configurations span the vertical-connectivity paradigms studied in previous work [15, 27]. More architectures can be expressed as combinations of the above types. Our evaluation focuses on the above types with *two-layer* fabrics. This aligns with near-term 3D FPGA integration scenarios [32] and captures the key placement challenges introduced by vertical connections. [3]

Table 1: Notations and terminologies.

Symbol	Definition
$b, \mathcal{B}$	A block in the netlist and set of blocks in the netlist, respectively, where $b \in \mathcal{B}$.
l_{src}, l_{dst}	Source and destination layer indices for a move.
Δx, Δy	Absolute Manhattan (horizontal) displacement between source and destination grid locations.
T, T_{init}, T_{exit}	Annealing temperature variables: current, initial (Section 9.1 in [16]), and exit temperature (Section 3 in [31]), respectively.
delay	Lookup table entry storing precomputed interconnect delay for a connection with bounding-box from (l_{src}, x, y) to $(l_{\text{dst}}, x+\Delta x, y+\Delta y)$.
$C_{\text{bb}}(b)$	Bounding-box wirelength cost contribution of block b (Equation 2.2 in [35]).
C_{bb}	Total bounding-box wirelength cost of the placement (Equations 4.2, 4.3 in [35]).
$C_{\text{timing}}(b)$	Timing cost contribution of block b (Equation 1).
C_{timing}	Total timing cost of the placement (Equations 7, 8 in [21]).
$C_{\text{total}}(b)$	Total placement cost per block b (Equation 5).
C_{total}	Aggregated total placement cost (SA cost function): $\sum_{b\in\mathcal{B}} C_{\text{total}}(b)$.
N_{moves}	Total number of moves per temperature step.
α	Move-acceptance rate, i.e., the fraction of attempted moves (out of N_{moves}) that are accepted.
w	Arithmetic average of move-acceptance rates in the last five temperature steps.
k	Number of layers in the 3D FPGA fabric.
N_{blk}	Number of movable logic blocks in the netlist.
$\theta(w)$	Our *dynamic* timing vs. wirelength weighting factor.
$\zeta(w)$	Our *dynamic* scaling factor that increases penalty for inter-layer timing cost.
p_ζ, p_θ	Exponential scaling factors (Equations 2, 4).
θ_{min}, θ_{max}	Preset bounds (min, max) of $\theta(w)$.
ζ_{min}, ζ_{max}	Preset bounds (min, max) of $\zeta(w)$.
w_{min}, w_{max}	Preset bounds (min, max) of w.

4 Our approach

We now describe our 3D placement flow (see notations and terminologies in Table 1). The overall flow is shown in Figure 1, with pseudocode in Algorithm 1. To enable reliable evaluation of 3D FPGA architectures, the placement engine must (i) accurately capture vertical delay, (ii) adapt its optimization operators to layered fabrics, and (iii) efficiently explore the 3D search space. This section details the modifications introduced into the VTR placement engine to meet these goals. Each enhancement contributes incrementally to placement quality; together, they form a cohesive 3D-aware flow (see ablation studies in Section 5.3). VTR uses simulated annealing [34] as its core placement algorithm, and we implement all of our enhancements on top of the reference implementation in [26].

Lines 1–14: The algorithm first partitions the netlist into k layers using *TritonPart* (Line 2) and constructs an initial 3D placement that respects this layer assignment (Line 3). Lines 4–10 then initialize all

[1] A block in an FPGA is a placeable logic or memory resource (IO, CLB, DSP, BRAM). During placement, every block corresponds to an element in the *packed* netlist [16] that must be mapped to a physical site on the FPGA fabric.

[2] In this work, we use the terms (*vertical, inter-layer, 3D*) connectivity interchangeably.

[3] Extending the flow to more than two layers is non-trivial and is left for future work. Unlike the two-layer case, deeper stacks introduce fundamental reachability constraints. For example, if only certain pins support vertical connections, nets between non-adjacent layers (e.g., layer 1 to layer 3) may be unroutable without intermediate resources. Addressing this would require changes to either or both of the routing architecture and placement formulation, which are beyond the scope of this work.

Algorithm 1: 3D-Aware Simulated-Annealing Placement.

Input: $\mathcal{N}$: netlist, k: number of layers, $\mathcal{M}$: SA move set, T_{init}: initial temperature, T_{exit}: exit temperature
Output: $place_{best}$: best 3D placement found

1 // Phase 1: Initialization and layer assignment
2 $\mathcal{N}_{\text{part}} \leftarrow$ partition $\mathcal{N}$ into k layers using *TritonPart*
3 $place_{curr} \leftarrow$ construct an initial placement consistent with $\mathcal{N}_{\text{part}}$
4 $T \leftarrow T_{\text{init}}$; // current temperature
5 $\alpha \leftarrow 1.0$; // initial value of move-acceptance rate
6 $\theta(w) \leftarrow \theta_{\min}$; // initial timing weight (Section 4.1)
7 $\zeta(w) \leftarrow \zeta_{\max}$; // initial 3D timing scale (Section 4.1)
8 $w \leftarrow 1.0$; // initial value of average move-acceptance rate
9 $N_{\text{blk}} \leftarrow$ number of movable blocks in $\mathcal{N}$
10 $N_{\text{moves}} \leftarrow 0.5 \cdot N_{\text{blk}}^{4/3}$; // moves per temperature step (VTR heuristic)
11 $place_{best} \leftarrow place_{curr}$
12 $C_{\text{BB}} \leftarrow$ get bounding-box wirelength cost of $place_{curr}$ (see Table 1)
13 $C_{\text{timing}} \leftarrow$ get timing cost of $place_{curr}$ (Table 1)
14 $C_{\text{best}} \leftarrow$ create placement cost tuple $\langle C_{\text{BB}}, C_{\text{timing}} \rangle$
15 // Phase 2: Simulated-annealing-based 3D placement
16 **while** $T > T_{exit}$ **do**
17 **for** $i \leftarrow 1$ **to** N_{moves} **do**
18 $m \leftarrow$ sample a move from $\mathcal{M}$ using the 3D move-selection policy (Section 4.2)
19 $\Delta C_{\text{total}} \leftarrow$ compute change in C_{total} (see Table 1, Equation 5)
20 **if** $\Delta C_{total} < 0$ **or** $e^{-\Delta C_{total}/T} > rand(0,1)$ **then**
21 $place_{best} \leftarrow$ apply move m to the current placement
22 $C_{\text{BB}} \leftarrow$ get bounding-box wirelength cost of $place_{curr}$ (Table 1)
23 $C_{\text{timing}} \leftarrow$ get timing cost of $\text{place}_{\text{curr}}$ (see Table 1)
24 $C_{\text{curr}} \leftarrow$ create placement cost tuple $\langle C_{BB}, C_{timing} \rangle$
25 **if** $C_{curr}.C_{BB} < C_{best}.C_{BB}$ **and** $C_{curr}.C_{timing} < C_{best}.C_{timing}$ **then**
26 $place_{best} \leftarrow place_{curr}$
27 $\alpha \leftarrow$ fraction of attempted moves (out of N_{moves}) that were accepted
28 $w \leftarrow$ arithmetic average of α across previous five temperature steps
29 $\zeta(w) \leftarrow$ adjust the 3D timing scale using w to control vertical timing emphasis (Section 4.1, Equation 1)
30 $\theta(w) \leftarrow$ update the timing weight using w to rebalance timing vs. wirelength (Section 4.1, Equation 4)
31 $T \leftarrow$ decrease the temperature according to the cooling schedule [26]
32 // Phase 3: Quenching / final refinement
33 $place_{best} \leftarrow$ perform a final quench (low-temperature refinement) on $place_{best}$
34 **return** $place_{best}$

simulated-annealing parameters. Lines 11–14 evaluate this initial placement by computing its bounding-box wirelength cost C_{BB} and timing cost C_{timing}, and store the corresponding cost tuple C_{best} as the current best solution.

Lines 16–31: For each temperature level, until the exit temperature is reached (Line 16), exactly N_{moves} move attempts are performed (Line 17). Each attempt samples a 3D-aware move operator from the move set $\mathcal{M}$ using the move-selection policy of Section 4.2 (Line 18), and computes the change in total cost ΔC_{total} (Lines 18–19).[4] The move is accepted if it improves the total cost or satisfies the Metropolis criterion [33] at the current temperature (Lines 20–21). When a move is accepted, it is applied to the current placement; the updated C_{BB} and C_{timing} are recomputed; and if the new cost tuple improves upon C_{best} in both dimensions, $\text{place}_{\text{best}}$ is updated (Lines 22–26). After all move attempts have been made at a given temperature level, α is set to the fraction of moves that were accepted (Line 27), w is updated as the arithmetic average of α over the five most recent temperature steps (Line 28), and $\zeta(w)$ and $\theta(w)$ are adapted based on w to rebalance timing versus wirelength and to control the emphasis on inter-layer timing effects (Lines 29–30).

Lines 33–34: After the annealing process completes, a final low-temperature refinement (quench) is applied to the best placement to remove residual suboptimal moves. The resulting placement is returned as the final 3D solution.

[4]We use the same total-cost formulation as in [26], except that the static timing weight t_f is replaced by the dynamic scaling factor $\theta(w)$, where w is the average move-acceptance rate from the previous five temperature steps (see Section 4.1).

Partition-based layer assignment. In 3D FPGA placement, the assignment of logic to layers strongly affects both timing and routability. A poor initial assignment can force the annealer to spend early iterations repairing poor cross-layer placements, limiting its ability to explore better 3D configurations later. To avoid this, we initialize the placement using a structured layer assignment that reflects logical connectivity and timing criticality. We obtain this assignment by partitioning the packed netlist produced by VTR's packing stage [16].[5] We build a hypergraph where each vertex represents a placeable logic block (CLBs, DSPs, BRAMs, I/Os) and each hyperedge represents a net; hyperedges are weighted using timing criticalities from the packer so that strongly connected or timing-critical groups tend to remain within the same layer. We then apply *TritonPart* to partition this weighted hypergraph into k layers ($k = 2$ in this work), using a 5% imbalance factor [2]. Finally, blocks are assigned to legal grid locations in decreasing-criticality order; if a layer becomes temporarily over-utilized, the block's layer assignment is relaxed and it is placed on the alternate layer, ensuring feasibility while preserving the partition-guided structure.

Placement delay model. Timing-driven placement and routing in VTR rely on a delay lookahead model: a precomputed table that estimates the minimum routing delay required to traverse a given Manhattan offset on the FPGA grid [11]. This model guides both the router's A* search [38] and the placer's timing evaluation, so its accuracy directly affects final timing quality. In 2D, VTR builds a delay table $delay[\Delta x][\Delta y]$ using Dijkstra's shortest-path search from a tile[6] near the bottom-left corner, recording for every reachable tile the minimum delay observed for that offset [11].

For 3D fabrics, this structure extends to a four-dimensional lookup table $delay[l_{\text{src}}][l_{\text{dst}}][\Delta x][\Delta y]$. However, VTR constructs this table using a single *average* delay per routing-segment type, causing intra-layer wires, vertical wires, and inter-layer connections (e.g., TSVs or monolithic (inter-layer) vias (MIVs)) to be treated uniformly—even though TSVs can be 3–20× slower and monolithic vias much faster than horizontal wires [25]. To address this, we modify the delay-model construction to record the *exact per-edge delays* encountered during Dijkstra's search rather than relying on segment averages. This allows intra-layer and inter-layer transitions to be evaluated separately, ensures that vertical hops reflect true TSV/MIV delays, and yields more consistent timing estimates for multi-layer nets—all with negligible runtime overhead.

4.1 Adaptive cost weighting and refinement

VTR's placer uses simulated annealing [26], beginning with broad exploration at high temperature and gradually refining the placement as temperature decreases. In 3D placement, two challenges arise: (i) if cross-layer moves are too permissive early on, the annealer can potentially disrupt a good initial layer assignment; and (ii) if such moves are over-penalized, the search becomes trapped within layers, preventing effective 3D exploration. Additionally, during the initial annealing stages, timing estimates are unreliable because inter-layer connectivity and routing demand are still evolving; over-weighting timing at this stage can potentially lead to "suboptimal" moves, while under-weighting it slows convergence. To balance these effects, our cost function (Equation 5) adapts continuously with annealing progress. Specifically, we (i) modulate the

[5]We partition the packed netlist, rather than the primitive netlist, because (i) resource usage per layer becomes explicit after packing, and (ii) timing criticalities are more accurate post-packing.

[6]In VTR, a *tile* is a physical grid location that may contain a logic block, switch box (SB), or connection box (CB); it is the fundamental unit used for placement and routing.

effective timing cost of cross-layer movement based on annealing state, and (ii) dynamically adjust the relative emphasis on timing versus wirelength objectives throughout the process.

Inter-layer timing cost.

We introduce a scaling factor $\zeta(w)$ that dynamically adjusts the timing penalty of vertical transitions during annealing. Early in the process, a large $\zeta(w)$ discourages excessive vertical movement and preserves the partition-guided layer assignment established at initialization. As annealing progresses and the placement stabilizes, $\zeta(w)$ gradually decreases, allowing more cross-layer movement.

This adaptive scaling helps to achieve a smooth transition from partition-preserving to fully 3D timing-driven optimization. We apply $\zeta(w)$ to the *incremental* vertical timing cost $C_{3D}(b)$ relative to the intra-layer baseline cost $C_{2D}(b)$:

$$C_{\text{timing}}(b) = C_{2D}(b) + \zeta(w)\, C_{3D}(b), \tag{1}$$

where

$$C_{2D}(b) = \text{delay}[l_{\text{src}}][l_{\text{src}}][\Delta x][\Delta y],$$
$$C_{3D}(b) = \text{delay}[l_{\text{src}}][l_{\text{dst}}][\Delta x][\Delta y] - \text{delay}[l_{\text{src}}][l_{\text{src}}][\Delta x][\Delta y].$$

The scale $\zeta(w)$ is driven by the averaged acceptance rate w (to avoid jitter[7]) and follows a monotone, nonincreasing schedule:

$$\zeta(w) = \zeta_{\max} + (\zeta_{\min} - \zeta_{\max})\left(\frac{w_{\max} - w}{w_{\max} - w_{\min}}\right)^{p_\zeta}, \quad p_\zeta \in \{1, 2\}. \tag{2}$$

At high acceptance rates (early, high-temperature stages), a large $\zeta(w)$ imposes strong penalties on vertical transitions, effectively constraining placement to a pseudo-2D regime. As w decreases, $\zeta(w)$ is gradually reduced toward $\zeta_{\min}$, relaxing penalties and enabling cross-layer moves that refine timing on critical paths.

Dynamic timing–wirelength modulation. While $\zeta(w)$ modulates local inter-layer timing penalties, the balance between timing and wirelength in the cost function must also evolve during SA. Early in the placement, timing estimates are noisy and the layout is still changing significantly, so too much weight on timing can cause the optimizer to chase spurious gradients. We adapt VTR's SA cost function [16] and use a dynamic schedule $\theta(w)$:

$$C_{\text{total}}(b) = (1 - \theta(w))\, C_{\text{BB}}(b) + \theta(w)\, C_{\text{timing}}(b), \tag{3}$$

$$\theta(w) = \begin{cases} \theta_{\max} - w^{p_\theta}\left(\theta_{\max} - \theta_{\min}\right), & \text{if } w > 0.15, \\ \theta_{\max}, & \text{otherwise.} \end{cases} \tag{4}$$

$$C_{\text{total}} = \sum_{b \in \mathcal{B}} C_{\text{total}}(b). \tag{5}$$

The SA cost function is Equation 5. At high acceptance rates (α) (exploratory phase), $\theta(w)$ is small, prioritizing wirelength. As SA progresses and timing stabilizes, $\theta(w)$ increases smoothly toward $\theta_{\max}$, shifting emphasis to timing-driven optimization. When w falls below 15%, we fix $\theta(w) = \theta_{\max}$, consistent with [13] that characterizes this phase as final annealing where the "true objective" should dominate. $\theta(w)$ is driven in the same manner as $\zeta(w)$ to avoid jitter and is constrained to be non-decreasing during placement to ensure stable convergence.

[7]Using a moving average for the acceptance rate prevents sharp, step-to-step changes in $\zeta(w)$, which would otherwise cause abrupt shifts in the cost landscape and destabilize annealing. The smoothed w lets $\zeta(w)$ track the annealer's progress without noisy fluctuations.

4.2 Expanded SA move set

The VTR placer uses a *reinforcement learning (RL) agent* [16] to adaptively select among several move types—*uniform*, *centroid*, *median*, and *feasible-region*—each relocating a single logic block according to a heuristic rule. In centroid moves, a block is displaced toward the geometric center of its connected blocks; median moves target the median position along each axis; and weighted variants bias these targets by timing criticality to emphasize critical connections. Each proposed move is evaluated by the simulated-annealing cost function (Equation 5), and only accepted moves update the placement, so directional bias in the move generator directly shapes the explored search space. In VTR, these moves are augmented with a z-coordinate: uniform moves may target any legal location on any layer; centroid and median moves use the centroid or median of their neighbors' layer indices; and feasible-region moves remain confined to the source layer of the candidate block. However, centroid and median moves yield very limited vertical exploration, since an imbalanced layer distribution among neighbors pulls the averaged z-coordinate toward the majority layer (e.g., six neighbors on layer 1 vs. three on layer 2 keeps the target near layer 1), making cross-layer transitions unlikely unless many neighbors move first to the target layer, under the same biased scheme.

To overcome this limitation, we extend the SA move set to explicitly encourage vertical exploration and add a dedicated cross-layer perturbation operator. Centroid and median moves now select the destination layer *probabilistically* based on the distribution[8] of a block's neighbors across layers—e.g., if a block has three neighbors on layer 1 and two on layer 2, it moves to layer 1 with probability 60% and to layer 2 with probability 40%.

In addition, we introduce a *Layer-Swap* move that exchanges the layer assignments of two blocks while preserving their (x, y) coordinates. Together, these extensions broaden the action space for 3D placement: adaptive cost weighting determines when vertical moves become favorable, and the expanded move set enables beneficial cross-layer perturbations to be realized once the cost landscape permits them.

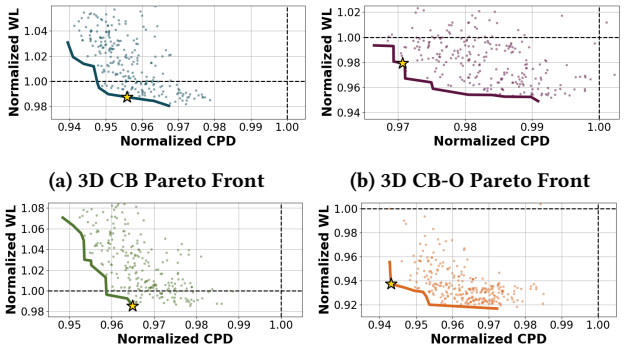
5 Experimental Evaluation

We implement our flow in C++ and integrate it into the VTR codebase [26], using *TritonPart* [39] as the hypergraph partitioner. All experiments are performed on a server equipped with two 4.1 GHz Intel Xeon Gold 6548Y+ CPUs and 512 GB RAM. We evaluate our flow using the Koios benchmark suite [1] and the four 3D FPGA architectures based on [15]. The FPGA fabric includes I/O blocks along the device periphery, with interior tiles comprising approximately 75% CLBs, 12.5% DSPs, and 12.5% BRAMs. Each benchmark is mapped onto the smallest feasible FPGA grid to ensure high utilization.[9] Our baselines consist of the VTR implementation in [26] set in two configurations: (i) *3D-baseline*, VTR run on the above 3D architectures, and (ii) *2D-baseline*, VTR run on the corresponding *2D-equivalent* architectures.[10] Results for critical-path delay (CPD) and wirelength (WL) in all tables and plots are reported relative to the *2D-baseline* (exceptions are Figures 3, 4) and averaged over 10 independent runs with distinct random seeds. For clarity, our evaluation is structured as follows: (i) QoR evaluation of our flow (Section 5.1);

[8]For a block b and $b \in \mathcal{B}$, let $N_b^{(i)}$ be the number of neighbors on layer i and $N_b^{\text{tot}} = \sum_{j=1}^{k} N_b^{(j)}$. We choose layer i with probability $P(L = i \mid b) = N_b^{(i)}/N_b^{\text{tot}}$.

[9]Grid dimensions used are available in the architecture XML files in [40].

[10]A 2D-equivalent architecture preserves the logical and routing capacity of its 3D counterpart while flattening all resources into a single layer.



(a) 3D CB Pareto Front (b) 3D CB-O Pareto Front

(c) 3D CB-I Pareto Front (d) 3D SB Pareto Front

Figure 3: Pareto fronts across the four architectures. CPD and WL are normalized to *3D-baseline*. Stars = chosen configs.

(ii) hyperparameter exploration (Section 5.2); (iii) ablation studies (Section 5.3); (iv) routability evaluation (Section 4); and (v) architecture exploration (Section 5.5). Full details, data for reproducibility, and an analysis of layer reassignment behavior are given in [40].

5.1 QoR comparison

Table 2 presents CPD and WL results compared to the *3D-* and *2D-*baselines. We run 10 placement trials with different random seeds and report the arithmetic mean, with all values normalized to the *2D-baseline*. Across all four 3D architectures, our flow consistently outperforms both baselines.

Geometric-mean improvements of our flow compared to the *3D-*(*2D-*) baselines are as follows: (i) 3D CB: CPD improves by 3.32% (6.80%) and WL by 0.86% (19.70%); (ii) 3D CB-O: CPD improves by 2.23% (7.80%) and WL by 5.52% (9.40%); (iii) 3D CB-I: CPD improves by 2.72% (7.00%) and WL by 0.25% (19.60%); and (iv) 3D SB: CPD improves by 6.21% (7.90%) and WL by 4.74%.

The 3D SB architecture benefits the most from our flow because its vertical connections are assigned dynamically; without an accurate 3D delay model, it is particularly sensitive to lookahead inaccuracies, since the delay lookahead matrix [26] also guides routing-path exploration (Section 4).

5.2 Hyperparameter exploration

We tune the hyperparameters controlling the adaptive cost functions $\zeta(w)$ and $\theta(w)$ (Section 4.1) using `Optuna`'s Tree-structured Parzen Estimator (TPE) sampler [36], and jointly optimize WL and CPD. Each trial evaluates one configuration on nine representative Koios designs across five random seeds, and we perform 300 trials per architecture; for each seed, CPD and WL are measured, normalized to the *3D-baseline*, and aggregated via geometric mean as the trial objective. The resulting Pareto fronts are shown in Figure 3. For each architecture, we choose the Pareto-efficient configuration that maximizes combined gain in CPD and WL (yellow star in each plot). The selected configurations are summarized in Table 3.

5.3 Ablation studies

To quantify the contribution of each component in our 3D placement flow, we perform ablation studies on the CB architecture using the same Optuna-based hyperparameter tuning setup described in Section 5.2, but enabling only one enhancement at a time. *Partition-based layer assignment* and the *probabilistic move-set*

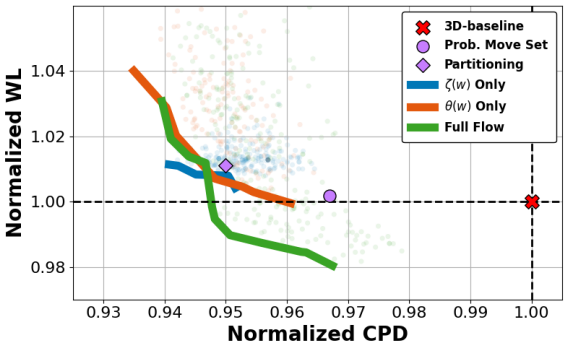


Figure 4: Ablation studies on a subset of the Koios benchmarks.

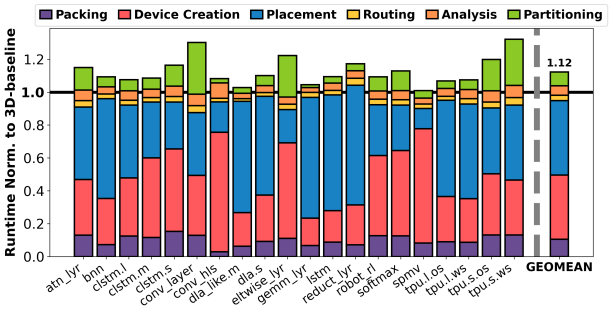


Figure 5: Runtime profiling on 3D CB. Results are normalized to *3D-baseline* and averaged over 10 seeds.

expansion are "binary features" and therefore appear in Figure 4 as individual points, whereas the *3D timing-scaling factor* $\zeta(w)$, the *timing–wirelength balance* $\theta(w)$, and the *full flow* produce Pareto curves as their hyperparameters vary. Results show that tuning $\zeta(w)$ (blue) and $\theta(w)$ (orange) can achieve CPD improvements comparable to those of the *full flow*, but only the *full flow* simultaneously improves both CPD and WL relative to the *3D-baseline*. This indicates that the *combination* of our proposed enhancements is needed to obtain consistent timing and wirelength gains.

Runtime remarks. Figure 5 illustrates the runtime overhead of our flow on the Koios benchmarks, normalized to *3D-baseline*. Overall, our flow incurs a 12% increase in total runtime (geometric mean). As the stacked breakdown shows, the runtime increases above the baseline (i.e., exceeding the $y = 1$ line) is dominated by the partitioning stage, which accounts for most of the additional runtime overhead.

5.4 Routability evaluation

Routability is a critical measure of placement quality, since poor spatial organization can lead to excessive routing demand or even unroutable designs. We assess routability by comparing the minimum channel width[11] (*min_CW*) required for successful routing between our 3D placement flow and the *3D-baseline*. As summarized in Table 4 for the Koios benchmarks on the 3D CB-O architecture (averaged over ten seeds), our flow reduces *min_CW* by 2.82% on average (geo-mean) relative to *3D-baseline*, with the `reduction_layer` showing the most (30.53%) reduction.

[11] Minimum channel width (*min_CW*) is the smallest number of routing tracks per routing *channel* that allows all nets to be routed; lower *min_CW* indicates better routability and more balanced congestion [37].

Table 2: QoR comparison: CPD and WL of our flow vs. *3D-baseline*, with all values normalized to *2D-baseline*. Δ is the percentage improvement of our flow compared to the *3D-baseline*. More negative values of Δ are better.

Benchmark	3D CB						3D CB-O						3D CB-I						3D SB					
	CPD			WL			CPD			WL			CPD			WL			CPD			WL		
	3D-base	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ	*3D-base*	Ours	Δ
clstm_like.large	0.952	0.917	**-3.68%**	0.815	0.794	**-2.58%**	0.989	0.914	**-7.58%**	1.030	0.890	**-13.59%**	0.913	0.906	**-0.77%**	0.813	0.802	**-1.35%**	1.112	0.944	**-15.11%**	1.082	1.042	**-3.70%**
clstm_like.medium	0.914	0.885	**-3.17%**	0.823	0.811	**-1.46%**	0.947	0.882	**-6.86%**	0.987	0.898	**-9.02%**	0.899	0.881	**-2.00%**	0.825	0.805	**-2.42%**	1.018	0.905	**-11.10%**	1.086	1.055	**-2.85%**
clstm_like.small	0.941	0.948	+0.74%	0.810	0.798	**-1.48%**	0.941	0.928	**-1.38%**	0.935	0.895	**-4.28%**	0.947	0.928	**-2.01%**	0.806	0.793	**-1.61%**	0.992	0.993	+0.10%	1.077	1.044	**-3.06%**
dla_like.medium	1.010	0.970	**-3.96%**	0.792	0.799	+0.88%	0.968	0.970	+0.21%	0.892	0.897	+0.56%	1.005	0.963	**-4.18%**	0.788	0.805	+2.16%	0.958	0.931	**-2.82%**	1.091	1.054	**-3.39%**
dla_like.small	0.972	0.942	**-3.09%**	0.805	0.802	**-0.37%**	0.953	0.938	**-1.57%**	0.904	0.908	+0.44%	0.961	0.949	**-1.25%**	0.797	0.805	+1.00%	0.931	0.935	+0.43%	1.152	1.085	**-5.82%**
lstm	0.907	0.910	+0.33%	0.789	0.779	**-1.27%**	0.870	0.833	**-4.25%**	0.964	0.953	**-1.14%**	0.895	0.911	+1.79%	0.778	0.787	+1.16%	0.931	0.809	**-13.10%**	1.091	1.076	**-1.37%**
tpu_like.large.os	0.893	0.842	**-5.71%**	0.814	0.825	+1.35%	0.887	0.858	**-3.27%**	0.861	0.902	+4.76%	0.885	0.832	**-5.99%**	0.820	0.839	+2.32%	0.827	0.797	**-3.63%**	1.096	1.084	**-1.09%**
tpu_like.large.ws	0.928	0.919	**-0.97%**	0.812	0.812	0.00%	0.903	0.888	**-1.66%**	0.984	0.917	**-6.81%**	0.921	0.893	**-3.04%**	0.801	0.805	+0.50%	0.961	0.895	**-6.87%**	1.079	1.050	**-2.69%**
tpu_like.small.os	0.913	0.893	**-2.19%**	0.854	0.844	**-1.17%**	0.887	0.856	**-3.49%**	0.912	0.936	+2.63%	0.923	0.886	**-4.01%**	0.836	0.846	+1.20%	0.900	0.850	**-5.56%**	1.152	1.122	**-2.60%**
tpu_like.small.ws	1.006	0.982	**-2.39%**	0.805	0.794	**-1.37%**	0.985	0.949	**-3.65%**	0.911	0.899	**-1.32%**	1.001	0.973	**-2.80%**	0.800	0.798	**-0.25%**	1.041	0.998	**-4.13%**	1.073	1.053	**-1.86%**
bnn	0.941	0.907	**-3.61%**	0.758	0.770	+1.58%	0.934	0.913	**-2.25%**	0.989	0.862	**-12.84%**	0.947	0.913	**-3.59%**	0.765	0.763	**-0.26%**	0.962	0.890	**-7.48%**	0.997	0.948	**-4.91%**
gemm_layer	0.824	0.838	+1.70%	0.801	0.790	**-1.37%**	0.862	0.837	**-2.90%**	1.135	0.927	**-18.33%**	0.855	0.853	**-0.23%**	0.786	0.788	+0.25%	0.967	0.798	**-17.48%**	1.080	1.024	**-5.19%**
attention_layer	1.050	0.973	**-7.33%**	0.830	0.823	**-0.84%**	0.977	0.975	**-0.20%**	0.882	0.882	0.00%	1.040	0.984	**-5.38%**	0.838	0.828	**-1.19%**	1.026	0.964	**-6.04%**	1.142	1.054	**-10.16%**
conv_layer	1.024	0.987	**-3.61%**	0.820	0.817	**-0.37%**	0.987	0.989	+0.20%	0.918	0.914	**-0.44%**	1.024	0.986	**-3.71%**	0.821	0.817	**-0.49%**	1.043	1.014	**-2.78%**	1.208	1.099	**-9.02%**
spmv	1.009	0.943	**-6.54%**	0.830	0.816	**-1.69%**	0.962	0.926	**-3.74%**	0.901	0.908	+0.78%	0.967	0.956	**-1.14%**	0.824	0.818	**-0.73%**	1.037	0.948	**-8.58%**	1.157	1.101	**-4.84%**
robot_rl	1.017	0.979	**-3.74%**	0.825	0.807	**-2.18%**	0.986	0.968	**-1.83%**	0.966	0.954	**-1.24%**	1.008	0.994	**-1.39%**	0.828	0.813	**-1.81%**	1.004	1.004	0.00%	1.259	1.155	**-8.26%**
reduction_layer	0.878	0.853	**-2.85%**	0.768	0.754	**-1.82%**	0.892	0.916	+2.69%	1.291	0.880	**-31.84%**	0.838	0.838	0.00%	0.758	0.763	+0.66%	0.876	0.863	**-1.48%**	1.128	1.092	**-3.19%**
softmax	1.043	0.979	**-6.14%**	0.816	0.823	+0.86%	0.976	0.974	**-0.20%**	0.926	0.932	+0.65%	1.066	0.976	**-8.44%**	0.812	0.824	+1.48%	1.032	0.988	**-4.26%**	1.173	1.068	**-8.95%**
conv_layer_hls	1.057	1.000	**-5.39%**	0.819	0.809	**-1.22%**	0.986	0.992	+0.61%	0.924	0.862	**-6.71%**	1.049	1.016	**-3.15%**	0.827	0.788	**-4.72%**	1.050	0.994	**-5.33%**	1.088	1.015	**-6.71%**
eltwise_layer	1.051	1.002	**-4.66%**	0.812	0.788	**-2.96%**	0.995	0.963	**-3.22%**	0.903	0.910	+0.78%	1.015	1.001	**-1.38%**	0.806	0.798	**-0.99%**	1.028	0.958	**-6.81%**	1.199	1.110	**-7.42%**
Geomean	0.964	0.932	**-3.32%**	0.810	0.803	**-0.86%**	0.943	0.922	**-2.23%**	0.956	0.906	**-5.52%**	0.956	0.930	**-2.72%**	0.806	0.804	**-0.25%**	0.982	0.921	**-6.21%**	1.119	1.066	**-4.74%**

Table 3: Optimal parameter set found for each architecture.

Parameter	3D CB	3D CB-O	3D CB-I	3D SB
p_ζ	1	1	1	2
p_θ	1	1	1	1
θ_{min}	0.03	0.09	0.03	0.35
θ_{max}	0.51	0.80	0.51	0.79
ζ_{max}	1.6	1.4	2.8	2.0
ζ_{min}	1.0	1.0	1.0	1.0
w_{max}	0.41	0.31	0.79	0.26
w_{min}	0.32	0.16	0.61	0.15

Table 4: Comparison of *min_CW* of our flow vs. *3D-baseline*. Architecture is 3D CB-O; results are averaged over 10 seeds.

Benchmark	VTR 9	Ours	Δ	Benchmark	VTR 9	Ours	Δ
attention_layer	153.6	154.8	+0.78%	bnn	95.0	115.8	+21.89%
clstm_like.large	109.0	119.6	+9.72%	clstm_like.medium	107.6	102.6	**-4.65%**
clstm_like.small	111.6	110.8	**-0.72%**	conv_layer	199.6	189.8	**-4.91%**
conv_layer_hls	67.2	66.6	**-0.89%**	dla_like.medium	201.8	224.2	+11.10%
dla_like.small	182.4	171.0	**-6.25%**	eltwise_layer	101.4	94.2	**-7.10%**
gemm_layer	100.4	96.6	**-3.78%**	lstm	155.4	150.6	**-3.09%**
reduction_layer	90.4	62.8	**-30.53%**	robot_rl	120.2	102.4	**-14.81%**
softmax	72.8	76.6	+5.22%	spmv	89.2	87.6	**-1.79%**
tpu_like.large.os	148.8	135.6	**-8.87%**	tpu_like.large.ws	154.8	162.8	+5.17%
tpu_like.small.os	130.4	129.2	**-0.92%**	tpu_like.small.ws	155.3	139.2	**-10.37%**
Geomean							**-2.82%**

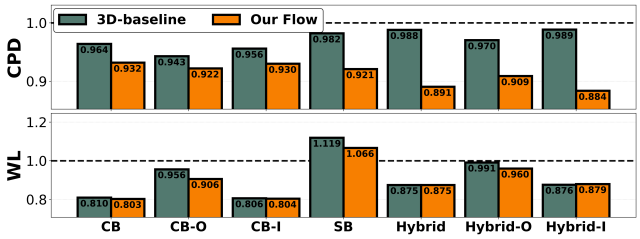


Figure 6: Results of various hybrid architectures on *3D-baseline* and our flow, normalized to *2D-baseline*.

5.5 Architecture exploration

To assess the generality of our flow and its utility for architectural studies, we evaluate a set of *hybrid architectures* that combine vertical connectivity through both logic pins and switch-box routing. Unlike the base CB, CB-O, CB-I, and SB architectures—where vertical links are restricted to a single connection type—hybrid architectures enable richer 3D connectivity (illustrated in Figure 2). We study three variants: (i) *Hybrid*, enabling both logic-pin and SB vertical connections; (ii) *Hybrid-O*, enabling output-pin and SB connections; and (iii) *Hybrid-I*, enabling input-pin and SB connections.

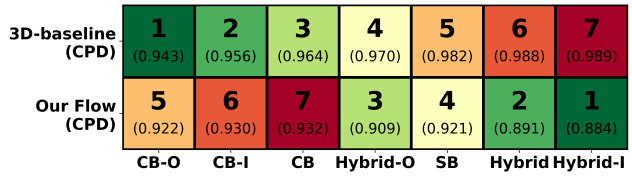


Figure 7: Relative CPD ordering of different 3D architectures on Koios benchmarks using *3D-baseline* and our flow.

Figure 6 shows normalized QoR (geometric mean across all Koios designs) using both *3D-baseline* and our flow. With *3D-baseline*, hybrid architectures appear inferior to the base 3D architectures, reflecting the placer's limited ability to exploit vertical flexibility. In contrast, our 3D-aware flow unlocks the potential of the hybrid architectures: CPD improves by up to 10.6% (Hybrid-I) and WL by up to 3.13% (Hybrid-O) relative to *3D-baseline*. Figure 7 further highlights this shift in architectural ranking. With *3D-baseline*, CB-O appears to be the strongest-performing architecture. With our flow, the hybrid architectures rise to the top, outperforming all base architectures. These results indicate that previous assessments understated the benefits of hybrid connectivity, and demonstrate that our placement framework provides the accuracy needed for meaningful and "directionally-correct" 3D FPGA architecture exploration.

6 Conclusion

We have presented an open-source 3D FPGA placement flow that combines partitioning-based initialization, refined 3D delay modeling, adaptive cost scheduling, and expanded 3D move sets. These enhancements improve timing guidance and enable effective exploration of the vertical design space. Across multiple 3D architectures, our flow achieves up to **10.6%** lower critical-path delay and **5.52%** shorter wirelength relative to VTR 9, while preserving routability. Ongoing efforts include extending the flow to multi-layer (> 2) architectures, incorporating thermal and power models, and exploring 3D routing optimizations. Our work thus provides foundations for the advance of 3D FPGA CAD and architecture research.

Acknowledgments

Research at UCSD is supported in part by Altera. Research at Georgia Tech was partially supported by the NSF under Grant Numbers 2338365 and 2317251.

References

[1] A. Arora, A. Boutros, S. A. Damghani, K. Mathur, V. Mohanty, T. Anand et al., "Koios 2.0: open-source deep learning benchmarks for fpga architecture and cad research", *IEEE Trans. on CAD* 42(11) (2023), pp. 3895–3909.

[2] I. Bustany, G. Gasparyan, A. B. Kahng, I. Koutis, B. Pramanik and Z. Wang, "An open-source constraints-driven general partitioning multi-tool for VLSI physical design", *Proc. ICCAD*, 2023, pp. 1–9.

[3] R. Reif, A. Fan, K.-N. Chen and S. Das, "Fabrication technologies for three-dimensional integrated circuits", *Proc. ISQED*, 2002, pp. 33–37.

[4] "Navy orders diminishing manufacturing sources (dms) fpgas to keep f-35s and other military aircraft flying", Military + Aerospace Electronics, 2018, https://www.militaryaerospace.com/computers/article/16726578/navy-orders-diminishing-manufacturing-sources-dms-fpgas-to-keep-f-35s-and-other-military-aircraft-flying

[5] "Securing a silicon pathway: addressing supply chain risks in strategic field-programmable gate array chips", Schneier on Security, 2025, https://www.schneier.com/academic/archives/2025/08/securing-a-silicon-pathway-addressing-supply-chain-risks-in-strategic-field-programmable-gate-array-chips.html

[6] D. Kong, Q. Jia and H. Xu, "The design of an integrated guidance and control computer system based on multi-core DSP and FPGA", *Proc. CISP*, 2015, pp. 1625–1629.

[7] M. Lin, A. El-Gamal, Y.-C. Lu and S. Wong, "Performance benefits of monolithically stacked 3-d fpga", *IEEE Trans. on CAD* 26(2) (2007), pp. 216–229.

[8] K. W. Guarini, "Electrical integrity of state-of-the-art 0.13 um soi device and circuits transferred for three-dimensional (3d) integrated circuit (ic) fabrication", *Proc. IEDM*, 2002, pp. 943–945.

[9] S. K. Samal, D. Nayak, M. Ichihashi, S. Banna and S. K. Lim, "Monolithic 3d ic vs. tsv-based 3d ic in 14nm finfet technology", *Proc. S3S*, 2016, pp. 1–2.

[10] Y. Cheng, X. Guo and V. F. Pavlidis, "Emerging monolithic 3d integration: opportunities and challenges from the computer system perspective", *Integration the VLSI Journal* 85 (2022), pp. 97–107.

[11] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault et al., "Vtr 8: high-performance cad and customizable fpga architecture modelling", *ACM Trans. on RTS* 13(2) (2020), pp. 1–55.

[12] J. Kim, L. Zhu, H. M. Torun, M. Swaminathan and S. K. Lim, "Micro-bumping, hybrid bonding, or monolithic? A ppa study for heterogeneous 3d ic options", *Proc. DAC*, 2021, pp. 1189–1194.

[13] M. A. Elgammal, K. E. Murray and V. Betz, "Rlplace: using reinforcement learning and smart perturbations to optimize fpga placement", *IEEE Trans. on CAD* 41(8) (2022), pp. 2532–2545.

[14] "Xilinx and tsmc reach volume production on all 28nm cowos-based all programmable 3d ic families", TSMC News Archives, 2013, https://pr.tsmc.com/english/news/1792

[15] I. Youssef, H. Yang and C. Hao, "LaZagna: an open-source framework for flexible 3d fpga architectural exploration", *arXiv:2505.05579*, 2025, https://arxiv.org/abs/2505.05579.

[16] M. A. Elgammal, A. Mohaghegh, S. G. Shahrouz, F. Mahmoudi, F. Koşar, K. Talaei et al., "Vtr 9: Open-source cad for fabric and beyond fpga architecture exploration", *ACM Trans. on RTS* 18(3) (2025), pp. 1–53.

[17] H. Rahimi and H. Jahanirad, "Implementation of digital circuits on three-dimensional fpgas using simulated annealing", *arXiv:2304.07476*, 2023, https://arxiv.org/abs/2304.07476.

[18] C. Ababei, H. Mogal and K. Bazargan, "Three-dimensional place and route for fpgas", *Proc. ASP-DAC*, 2005, pp. 773–778.

[19] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer and D. Z. Pan, "Dreamplacefpga: an open-source analytical placer for large scale heterogeneous fpgas using deep-learning toolkit", *Proc. ASP-DAC*, 2022, pp. 300–306.

[20] F. Mahmoudi, M. A. Elgammal, S. G. Shahrouz, K. E. Murray and V. Betz, "Respect the difference: reinforcement learning for heterogeneous fpga placement", *Proc. FPT*, 2023, pp. 152–160.

[21] A. Marquardt, V. Betz and J. Rose, "Timing-driven placement for fpgas", *Proc. FPGA*, 2000, pp. 203–213.

[22] S. Chtourou, M. Abid, Z. Marrakchi, E. Amouri and H. Mehrez, "On exploiting partitioning-based placement approach for performances improvement of 3d fpga", *Proc. HPCS*, 2017, pp. 572–579.

[23] P. Danassis, K. Siozios and D. Soudris, "Ant3d: simultaneous partitioning and placement for 3-d fpgas based on ant colony optimization", *IEEE Embedded Systems Letters* 8(2) (2016), pp. 41–44.

[24] Q. Zhao, M. Amagasaki, M. Iida and T. Yoshida, "Architecture-aware cost function for 3d fpga placement using convolutional neural network", *Proc. CANDAR*, 2020, pp. 235–241.

[25] S. Huang, Y. Lin, K. Tsai, W. Cheng, S. Sunter, Y. Chou et al., "Small delay testing for tsvs in 3-d ics", *Proc. DAC*, 2012, pp. 1031–1036.

[26] Verilog-to-routing repository (commit hash: 8cb20aa). https://github.com/verilog-to-routing/vtr-verilog-to-routing

[27] A. Boutros, F. Mahmoudi, A. Mohaghegh, S. More and V. Betz, "Into the third dimension: architecture exploration tools for 3d reconfigurable acceleration devices", *Proc. FPT*, 2023, pp. 198–208.

[28] M. Amagasaki, Q. Zhao, M. Iida, M. Kuga and T. Sueyoshi, "A 3d fpga architecture to realize simple die stacking", *Information and Media Technologies* 10(3) (2015), pp. 425–431.

[29] R. Le, S. Reda and R. I. Bahar, "High-performance, cost-effective heterogeneous 3D FPGA architectures", *Proc. GLSVLSI*, 2009, pp. 251–256.

[30] M. I. Masud and S. J. E. Wilton, "A new switch block for segmented fpgas", *Proc. FPL*, 1999, pp. 274–281.

[31] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for fpga research", *Proc. FPL*, 1997, pp. 213–222.

[32] B. Hoefflinger, "Irds—international roadmap for devices and systems, rebooting computing, s3s", *NANO-CHIPS 2030*, 2020, pp. 9–17.

[33] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller "Equation of state calculations by fast computing machines", *Journal of Chemical Physics* 21(6) (1953), pp. 1087–1092.

[34] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing", *Science* 220(4598) (1983), pp. 671–680.

[35] A. Mohaghegh, "Expanding fpga cad and architecture exploration to new routing structures and stacked silicon", *Masters thesis*, 2009, University of Toronto.

[36] S. Watanabe, "Tree-structured parzen estimator: understanding its algorithm components and their roles for better empirical performance", *arXiv:2304.11127* , 2023, https://arxiv.org/abs/2304.11127

[37] V. Betz, J. Rose and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[38] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Trans. on SSC* 4(2) (1968), pp. 100–107.

[39] TritonPart partitioner repository. https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/par

[40] Our flow. https://github.com/IY2002/Beyond-Flatland