DAMIL-DCIM: A Digital CIM Layout Synthesis Framework with Dataflow-Aware Floorplan and MILP-Based Detailed Placement

Chuyu Wang, Ke Hu, Fan Yang, Keren Zhu* and Xuan Zeng*
State Key Laboratory of Integrated Chips and Systems, School of Microelectronics, Fudan University, Shanghai, China

***Abstract*—Digital computing-in-memory (DCIM) systems integrate complex digital logic with parasitic-sensitive bitcell arrays. Conventional physical design strategies degrade DCIM performance due to a lack of dataflow regularity and excessive wirelength. As a result, current DCIM design often relies on manual layout, which is time-consuming and a bottleneck in the design cycle. Existing layout synthesis frameworks for DCIM often mimic the manual approach and employ a template-based method for DCIM placement. However, overly constrained templates lead to an excessive core area, resulting in high costs in practice. In this work, we introduce DAMIL-DCIM, a novel placement framework that bridges template-based techniques with optimization-based placement methods. DAMIL-DCIM utilizes a global dataflow-aware floorplan inspired by template methods and further optimizes the layout using MILP(Mixed Integer Linear Programming)-based detailed placement. The combination of global floorplanning and placement optimization reduces total wire length while maintaining dataflow regularity, resulting in lower parasitic and enhanced performance. Experimental results show, on a practical 28nm DCIM circuit, our approach improves frequency by 25.2% and reduces power consumption by 19.6% compared to Cadence Innovus, while maintaining the same core area.**

***Index Terms*—Electronic Design Automation, Digital Computing-in-Memory, Physical Design**

I. Introduction

The rapid growth of AI and the rise of edge computing have introduced new power and performance challenges for edge devices. Traditional Von Neumann architecture struggles to keep up due to high data movement costs. Computing-in-memory (CIM) has emerged as an efficient solution, with SRAM-based digital CIM (DCIM) offering zero accuracy loss and high adaptability, making it ideal for AI edge applications [1].

Unlike conventional digital AI accelerators, DCIM circuits combine digital components with parasitic-sensitive bitcell arrays. Like SRAM arrays, their read and write operations depend on bitline charging, requiring strict control of parasitic capacitance and resistance. Inferior DCIM layouts introduce excess parasitics, degrading performance and potentially compromising functionality. Conventional digital place-and-route (P&R) disrupts bitcell array structure, leading to performance overhead. On the other hand, memory compilers like OpenRAM [2] are limited to SRAM array generation and lack the ability to synthesize complete DCIM circuits. As a result, current DCIM designs often rely on manual layout design to carefully place and route circuits, minimizing parasitics introduced by the layout. This process is time-consuming and has become a bottleneck in the design cycle, significantly slowing down development.

Recent research has explored customized placement strategies tailored to specific circuit architectures. AutoDCIM [3] employs a template-based approach for DCIM placement, mimicking manual layout design by arranging circuit components through a predefined procedure. Similar methodologies are widely used in layout automation for analog and mixed-signal circuits, as demonstrated by OpenSAR [4] and BAG [5]. Template-based methods ensure a regular global floorplan and maintain the array structure of key modules. However, overly constrained templates can lead to significant area overhead when applied to circuits with minor modifications or different technology nodes. On the other hand, emerging placement methods for digital accelerators treat datapath design as a fundamental methodology [6]. They attempt to augment optimization-based placement framework with additional constraints or objectives on regularity of datapaths [7]–[16]. Cells on critical datapaths are placed following the dataflow or regular topology, ensuring that regularity is maintained in the final layout. However, these optimization-based placement methods for digital circuits may struggle to meet the stringent parasitic requirements of DCIM circuits.

A potential solution combines template-based and optimization-based placement. For DCIM circuits, the strategy should leverage the global structure while optimizing layout area and wire length. A global floorplan template guides signal flow, and local optimization techniques minimize the area overhead of the templates.

In this work, we present DAMIL-DCIM, a DCIM placement framework that bridges template-based methods with dataflow-aware placement techniques, optimizing parasitic-sensitive paths while enhancing the power, performance, and area (PPA). Unique global dataflow-aware floorplan templates are generated based on circuit specifications, and a mixed-integer linear programming (MILP)-based detailed placement is employed for certain components to achieve a more compact layout. Our main contributions are summarized as follows.

- We present DAMIL-DCIM, a novel DCIM placement

*Corresponding authors: {krzhu, xzeng}@fudan.edu.cn.

978-3-9826741-0-0/DATE25/© 2025 EDAA

framework that simultaneously handles parasitic-sensitive bitcell arrays and digital components, achieving high-quality layout placement.

- A global dataflow-aware floorplan template is introduced, leveraging the unique intrinsic tree-like dataflow of DCIM circuits to provide overall guidance for the entire layout.
- A MILP-based detailed placement method is employed to further minimize the local wirelength, reduce the overall area while maintaining local regularity.
- Experimental results show that our framework reduces total wire length by 37.4%, improves frequency by 25.2%, and decreases power consumption by 19.6% compared to layouts generated by commercial tools, all while maintaining the same core area. Post-layout simulations confirm effective management of parasitic-sensitive paths, ensuring successful bitcell read and write operations.

II. Preliminaries

A. Typical DCIM Architecture

As shown in Fig. 1, a typical DCIM circuit can be broadly categorized into three main components: the bitcell bank, the adder tree, and the output processing module.

Bitcell Bank: Similar to a traditional SRAM memory bank, a bitcell bank comprises an array of bitcells along with peripheral circuits and control logic. Each bitcell integrates an SRAM cell with a one-bit multiplier logic cell, handling both data storage and single-bit multiplication. Peripheral and control circuits, such as address decoders, bitline precharge circuits, and sense amplifiers, ensure proper read and write operations. Bitcells are typically sensitive to parasitics and require a regular placement pattern to ensure reliable performance.

Adder Tree: In a typical DCIM architecture, each bitcell array row serves as a parallel input branch, generating a partial sum each cycle. To efficiently accumulate these sums, adders are arranged in a $\log_2 N$-stage binary tree, where N is the number of partial sums. The adder tree occupies the majority of the chip area and often includes critical paths, making its placement crucial in DCIM physical design.

Output Processing Module: The output processing module primarily consists of a shift accumulator and optional digital functional blocks. These modules typically include sub-blocks composed of multiple identical cells for parallel multi-bit processing. Maintaining local regularity in these modules enables higher area utilization and improved performance.

A typical DCIM circuit was presented by [17]. In this design, the bitcell consists of a 6T SRAM cell combined with a 4T NOR cell. The adder tree includes 8 stages to support 256 parallel inputs from the bitcell array, utilizing modified systolic array adders. Another example was presented by [18], featuring a 12T bitcell array and a 6-stage adder tree with optimized full adders.

B. Templated-Based DCIM Layout Generation

DCIM circuits feature a unique structure with memory-like components, requiring a placement strategy that capitalizes on

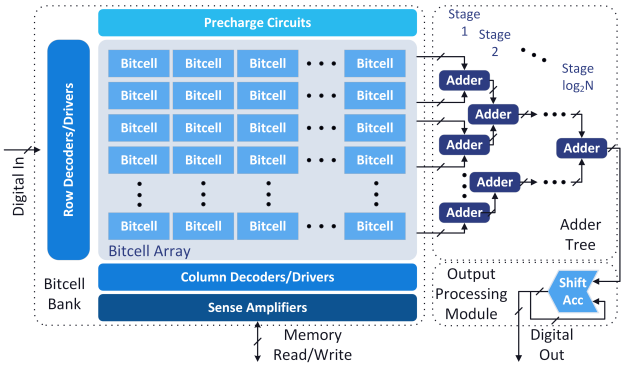


Fig. 1. Typical DCIM circuit architecture.

their specific architecture. AutoDCIM [3] employs a template-based DCIM macro generation method, using a customized template to guide overall circuit placement. In AutoDCIM, bitcells are arranged within a single bank, while the adder tree is placed adjacent to the bank. However, in modern DCIM designs, the digital implementation of wide adder trees occupies significant area [19], causing a layout size mismatch with the bitcell bank. AutoDCIM mitigates this mismatch by introducing a bitcell active interleaving strategy. The basic building block in AutoDCIM consists of SRAM-based weight cells and 1-bit multipliers in an $m : 1$ ratio. However, only one weight cell can be activated at a time to perform a multiplication with a multiplier cell. This means that only $1/m$ of the weight cells are active simultaneously, significantly reducing the number of partial sums and the width of the adder tree. While this decreases the adder tree size, it also results in low circuit resource utilization and substantially lowers throughput.

However, AutoDCIM's overly constrained handcrafted layout templates lack adaptability, resulting in low area utilization for circuits with minor modifications or different technology nodes. However, it reduces critical layout parasitics and enables automatic layout synthesis with correct circuit functionality. Consequently, template-generated DCIM layouts can incur up to a 60% area overhead compared to commercial PNR tools.

III. The DAMIL-DCIM Framework

The entire flow of the DAMIL-DCIM framework is illustrated in Fig. 2. DAMIL-DCIM takes circuit features, component cell layouts, and user-specific characteristics as input and generates a DRC-clean placement for further routing and verification. It first applies a tree-structured, dataflow-based DCIM template to create a global floorplan, which follows the unique binary tree-structured dataflow of DCIM and defines the relative locations of various components. Next, a MILP-based module layout generation method is used to produce the detailed placement of digital modules. We introduce customized MILP constraints for several modules, such as Brent-Kung adders and shift accumulators. These additional constraints accelerate MILP solving while preserving local regularity.

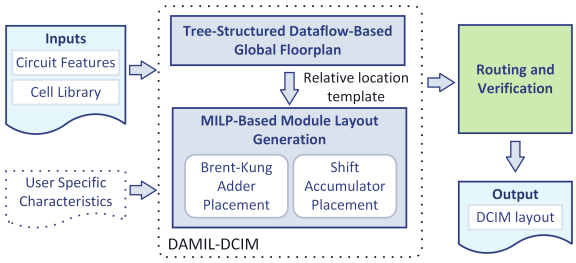


Fig. 2. Overview of the proposed DAMIL-DCIM flow.

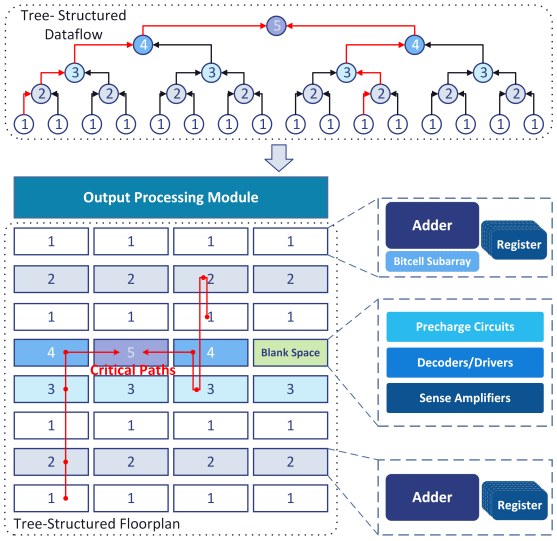


Fig. 3. Overview of the tree-structured dataflow-based global floorplan flow.

A. *Tree-Structured Dataflow-Based Global Floorplan*

DAMIL-DCIM employs a novel tree-structured global floorplan strategy. Fig. 3 shows the overall floorplan, where the adder tree serves as the top-level structure. Each node in the tree-structured floorplan contains an adder, along with an optional bitcell subarray and associated output registers. The bitcell banks are divided and integrated with their corresponding first-stage adders. Peripheral circuits and control logic modules are placed in specific blank spaces within the floorplan template. The output processing module is positioned adjacent to the top of the tree structure, ensuring an efficient and compact layout.

The adder tree-centric floorplan optimizes critical signal paths and enhances global routability. In modern DCIM circuits, the wide adder tree often dominates the layout, causing mismatches with the bitcell bank. Our placement framework adopts the adder tree's binary structure as the top-level floorplan, generating a configurable tree-structured template based on its dataflow, as shown in Fig. 3 for a 5-stage adder tree. Each node in the tree-structured dataflow is mapped to a block in the template, and the shape of the template can be configured according to the number of nodes per row. For instance, the template in Fig. 3 has 4 nodes per row, while Fig. 4a illustrates another template generated from the same 5-stage adder tree with 2 nodes per row. The tree's physical floorplan template is carefully designed as a binary recursive structure in both vertical and horizontal

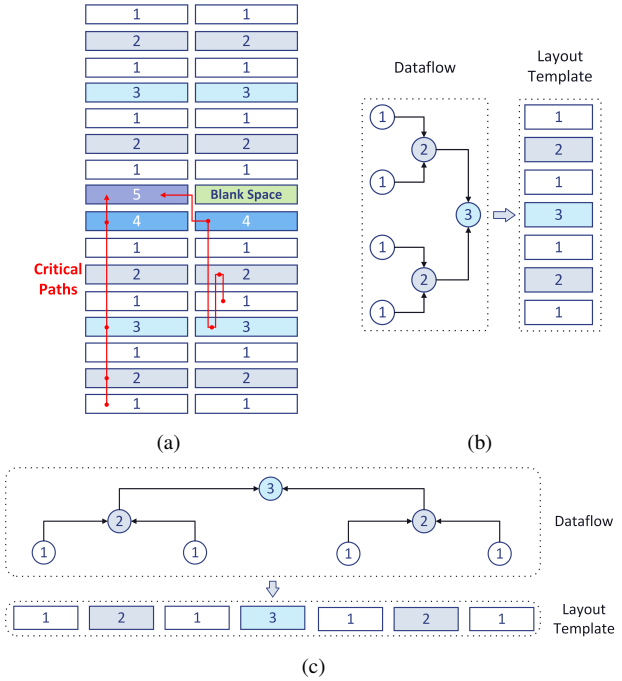


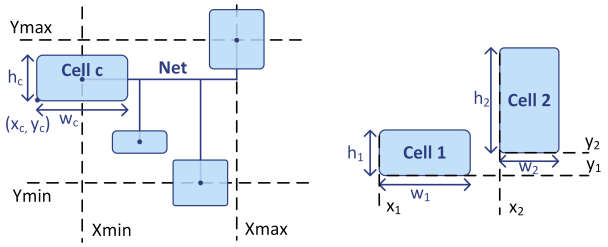
Fig. 4. Details in global floorplan template generation. (a) Template with 2 nodes per row. (b) Vertical floorplan template generation scheme. (c) Horizontal floorplan template generation scheme.

dimensions, as depicted in Fig. 4b and 4c, respectively. This design ensures that each critical path from the leaf adders to the top adder achieves approximately the same wire length, promoting balanced signal propagation and improving timing performance across the entire adder tree, while also reducing congestion during the routing stage.

DAMIL-DCIM splits the bitcell array into subarrays and merges them with corresponding first-stage nodes to prevent area mismatches with the adder tree. By having first-stage adders directly receive outputs from the bitcells, this strategy reduces the global wire length between bitcells and adders. Inspired by [18], which divides the bitcell array into smaller banks near the first and second-stage adders, DAMIL-DCIM assigns each first-stage node an adder, a bitcell subarray, and optional output registers. In contrast, nodes in other stages do not include bitcell subarrays. The peripheral circuits of the bitcell bank can be placed within the blank spaces of the floorplan template, as shown in Fig. 3. This floorplan strategy allows DAMIL-DCIM to maximize core area usage and prevent mismatches between the bitcell bank and adder tree, avoiding low area utilization and long wire lengths.

B. *MILP-Based Module Layout Generation*

After generating the global floorplan template, DAMIL-DCIM uses a MILP-based method for detailed placement. To speed up the time-consuming MILP process for large modules while maintaining local regularity, we introduce customized constraints for two common digital blocks. We first explain the basic MILP formulation, followed by our customized layout methods for Brent-Kung adders and shift accumulators.



(a) (b)

Fig. 5. Basic MILP placement formulation. (a) HPWL formulation in MILP style. (b) Overlap constraint formulation in MILP style.

1) Basic MILP Placement Formulation: The MILP formulation for the basic placement problem is formulated in Equation (1).

$$\begin{aligned} &\textbf{Minimize} \quad \sum_{n \in Nets} HPWL_n \\ &\textbf{Subject to} \\ &\qquad overlap(c_i, c_j) = 0, \\ &\qquad outOfBound(c_i) = 0, \\ &\qquad \forall c_i, c_j \in Cells, c_i \neq c_j \end{aligned} \tag{1}$$

The primary objective of the placement problem is to minimize total wire length while ensuring that modules do not overlap and are positioned within the given boundary. We use half-perimeter wirelength (HPWL) to estimate the wirelength in the placement.

HPWL is formulated in a linear form in Equation (2), where $Xmax_n$, $Ymax_n$, $Xmin_n$, and $Ymin_n$ represent the upper and lower x and y boundaries of the centers of each cell connected by net n, as illustrated in Fig. 5a. Here, x_c, y_c, w_c, and h_c represent the x-coordinate and y-coordinate of the bottom-left corner, width and height of cell c, respectively.

$$\begin{gathered} HPWL_n = Xmax_n + Ymax_n - Xmin_n - Ymin_n, \\ Xmax_n \geq x_c + w_c/2, Ymax_n \geq y_c + h_c/2, \\ Xmin_n \leq x_c + w_c/2, Ymin_n \leq y_c + h_c/2, \\ \forall c \in n, \forall n \in Nets \end{gathered} \tag{2}$$

We use four binary auxiliary variables, representing the relative locations of each pair of cells, to formulate non-overlap constraints, as shown in Equation (3). Only one auxiliary variable needs to be true, indicating that the two cells are separated in the specified direction. Fig. 5b provides an example where cell 1 is positioned to the left of cell 2, making aux_{left} true while the other auxiliary variables false.

$$\begin{gathered} x_i + w_i \leq x_j + M \cdot (1 - aux_{left}), \\ x_j + w_j \leq x_i + M \cdot (1 - aux_{righ}), \\ y_i + h_i \leq y_j + M \cdot (1 - aux_{below}), \\ y_j + h_j \leq y_i + M \cdot (1 - aux_{above}), \\ aux_{left} + aux_{right} + aux_{above} + aux_{below} \leq 1, \\ aux_{left}, aux_{right}, aux_{above}, aux_{below} \in \{0, 1\}, \\ \forall i, j \in Cells, i \neq j, \\ M \text{ is sufficiently large} \end{gathered} \tag{3}$$

The boundary constraints are defined in Equation (4), where X_{min}, Y_{min}, X_{max} and Y_{max} specify the limits of the x and y coordinates for each cell, with X_{min} and Y_{min} generally set to 0.

$$\begin{gathered} x_i \geq X_{min}, x_i + w_i \leq X_{max}, \\ y_i \geq Y_{min}, y_i + h_i \leq Y_{max}, \\ \forall i \in Cells \end{gathered} \tag{4}$$

2) Customized Brent-Kung Adder MILP Placement: The Brent-Kung adder (BK adder) features a unique tree-like dataflow consisting of three basic functional cells: the unit cells of *pg*, *dot* and *sum*. Fig. 6a shows the typical dataflow of a 12-bit BK adder.

The placement process is divided into four steps, to optimize layout regularity and align dataflow, as illustrated in Fig. 6b. 1) **Input and Output Cell Array Generation:** We generate unit cells of *pg* and *sum* and align them vertically bit-by-bit. We further fold the input and output cell arrays to achieve a desired layout aspect ratio for adders with large input bit widths. Then we generate a virtual grid for *dot* cells for the later steps. 2) **Dataflow Based Cell Pre-Placement:** We apply a regular digital placement of *dot* cells to obtain a pre-placement, as marked with red dashed box in Fig. 6a. The obtained positions of cells are used to formulate MILP relative position constraints in the later step. The relative position constraints empirically speedup the downstream MILP process by more than 10×. 3) **MILP Based Cell Placement:** We complete the *dot* cell placement with the MILP problem formulated in Section III-B1. The coordinates of each cell are constrained to ensure that all dot operation cells are placed within the grid. 4) **Dummy Cell Filling:** After all the cells are placed, corresponding dummy cells are inserted into unoccupied grid spaces to ensure a consistent density and maintain manufacturability.

3) Customized Shift Accumulator MILP Placement: DAMIL-DCIM introduces array constraints for shift accumulator submodules. Shift accumulators generally include submodules with multiple identical cells for parallel multi-bit processing. Additional array constraints enforce local regularity, improving signal flow and speeding up the MILP solving process. For multi-bit cell arrays, various shapes with different row and column counts can be selected. The shape is automatically determined within the MILP placement formulation and is represented by a row and column pair (r, c). The set of potential shapes, denoted as $Shapes$, can be easily determined based on the bit width. For each multi-bit cell array submodule, we introduce binary auxiliary variables corresponding to the number of possible array shapes $|Shapes|$. Only one of these binary variables is true, representing the selected shape of the cell array. The actual width and height of the submodule are then calculated, as formulated in Equation 5. Fig. 7 illustrates an example of four potential configurations of a 6-bit inverter array, with shapes of 1r6c (1 row by 6 columns), 2r3c, 3r2c and 6r1c, respectively.

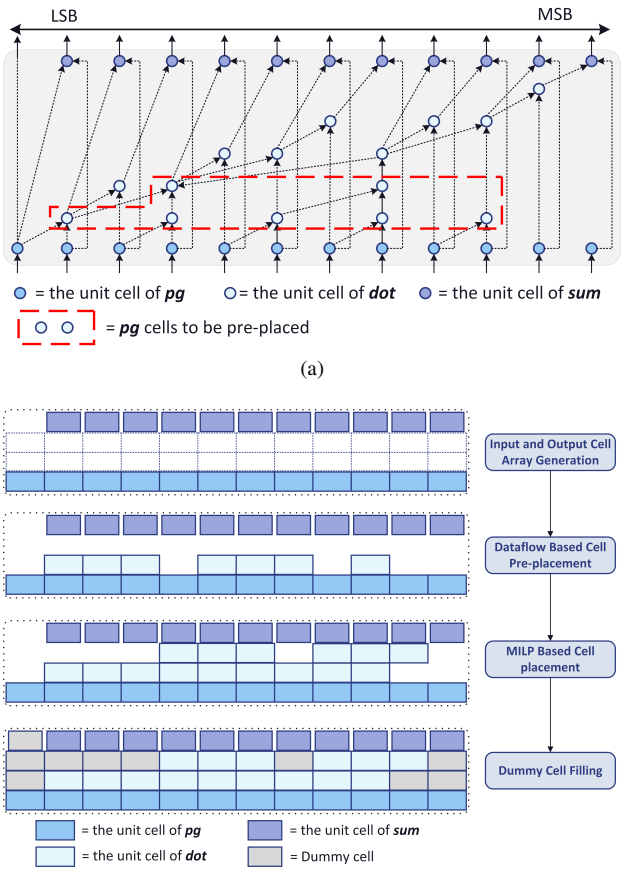


Fig. 6. BK Adder detailed placement strategy. (a) typical BK adder dataflow. (b) Customized BK adder placement flow.

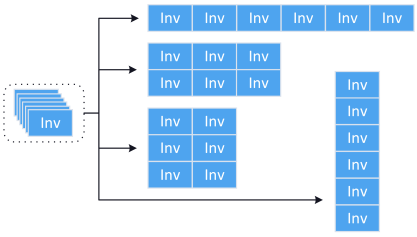


Fig. 7. An example of potential array configurations.

$$\begin{aligned} w_{submodule} &= \sum_{i=1}^{|shapes|} aux_i \cdot w_{cell} \cdot r_i \\ h_{submodule} &= \sum_{i=1}^{|shapes|} aux_i \cdot h_{cell} \cdot c_i \\ \sum_{i=1}^{|Shapes|} aux_i &= 1 \\ aux_i &\in \{0,1\}, 1 \le i \le |Shapes| \end{aligned} \tag{5}$$

To fully utilize the die area, segmented horizontal or vertical boundaries are sometimes necessary for accumulator placement. These boundaries are treated as standard rectangular boundaries with additional rectangular blockages. Fig. 8 provides an example where two rectangular blockages are introduced to model the boundary.

IV. Experimental Results

All experiments are conducted on a Linux workstation with a maximum of 16 Intel 3.0GHz CPU cores available for our process. Of all the steps in the placement flow, the MILP solving processes dominate the runtime. We observe that the optimization gap does not significantly decrease after a few hours. Therefore, we employ an early termination strategy for MILP solving, which stops the process after 12 hours and adopts the best solution available at that time. The MILP solving for specific modules is executed in parallel to further accelerate the overall process. Empirically, DAMIL-DCIM can outperform the baseline methods given limited runtime budget.

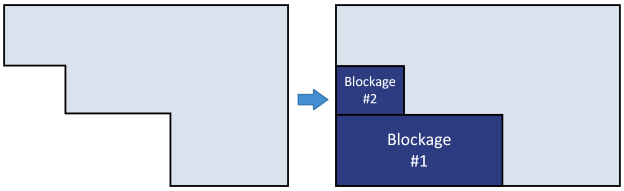


Fig. 8. Segmented boundary formulation.

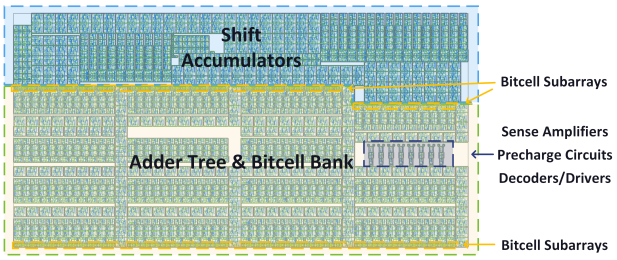


Fig. 9. Top level layout of the practical DCIM circuit.

Our experiments are conducted on a practical DCIM circuit following the architecture of [17]. The circuit consists of a bitcell bank with an 8×16 array size, a 4-stage adder tree using BK adders, and two shift accumulators, implemented with TSMC 28nm technology. Fig. 9 shows the completed layout using DAMIL-DCIM framework, with a core area of $1.132 \times 0.580 = 0.657\text{mm}^2$. The layout indicated that the adder tree occupies the majority of the core area, while the peripheral circuits and control logic of the bitcell bank are placed in the remaining blank spaces, guided by the tree-structured global floorplan. The two accumulators are positioned above the adder tree, utilizing a segmented horizontal boundary to maximize die area utilization.

A. Comparisons with Other Placement Methods

To evaluate the effectiveness of our proposed framework, we conducted a complete physical design flow on the DCIM circuit using different placement strategies. We compare DAMIL-DCIM against the following three placement methods: 1) **Innovus:** We compare with the placement in a commercial physical tool, Cadence Innovus. 2) **Template-Based Method:** We compare with the template-based AutoDCIM method. As we do not have access to their original implementation, we implement a fully template-based placement method following their methodology, serving as a template-based method. 3) **Innovus with SDP:** We compare with the structured data path (SDP) placement strategy of Cadence Innovus, and mark the bitlines in the bitcell bank as structured data paths. Innovus

TABLE I
COMPARISON WITH OTHER RELATED PLACEMENT FRAMEWORK

	Innovus	Ratio	Template-Based Method [3]	Ratio	Innovus with SDP	Ratio	Innovus with SDP*	Ratio	DAMIL-DCIM	Ratio
Core Area (mm^2)	**0.657**	**1.00**	1.052	1.60	**0.657**	**1.00**	0.685	1.04	**0.657**	**1.00**
Total Wire Length (mm)	495.377	1.00	**294.842**	**0.60**	823.771	1.66	548.285	1.11	310.02	0.63
Max Power (mW)	6.539	1.00	**3.951**	**0.60**	11.702	1.79	8.088	1.24	5.256	0.80
WNS (ns)	-0.798	-	0.079	-	-7.316	-	-1.184	-	**0.088**	-
TNS (ns)	-12.174	-	**0.000**	-	-216.127	-	-5.031	-	**0.000**	-
Equivalent Period (ns)	4.398	1.00	3.521	0.80	10.916	2.48	4.784	1.09	**3.512**	**0.80**
PPA ($mW \times ns \times mm^2$)	18.894	1.00	14.634	0.77	83.925	4.44	26.505	1.40	**12.128**	**0.64**

$*$ Run Innovus SDP flow with larger target area to avoid placement failure. Details in Section IV-A.

TABLE II
CLOCK TREE QUALITY COMPARISON WITH INNOVUS.

	Innovus	DAMIL-DCIM
Min ID (ns)	-0.471	**-0.553**
Max ID (ns)	0.017	**-0.416**
Avg ID (ns)	-0.329	**-0.493**
Std.Dev. ID (ns)	0.100	**0.019**
Skew (ns)	0.488	**0.137**

with SDP enforces regular structure to optimize the critical paths. After obtaining the placement results, we further route and complete the physical design flow using Innovus. The target clock period is set to 3.6ns. We use the same core area and aspect ratio except the Template-Based Method. We collect the total wire length, power consumption, and timing metrics of the routed circuits, and calculate the PPA scores. Table I compares the resulting layouts of DAMIL-DCIM and the baselines. DAMIL-DCIM achieves nearly the best results in terms of total wire length and timing metrics, and highest PPA score of all the methods.

Experimental results demonstrate that our method effectively reduces area overhead compared to template-based methods. Although the template-based method achieves slightly better power consumption and total wire length, it results in a core area of $1.570 \times 0.670 = 1.0519\text{mm}^2$, which is 1.60 times larger, and a standard cell density of only 56.05%. This sparse layout is impractical and significantly degrades the final PPA score.

Compared to Innovus, DAMIL-DCIM significantly reduces total wire length and improves timing. These timing enhancements are due to lower parasitics from shorter wires and the global floorplan's effective placement of registers within the adder tree, which enhances clock tree quality. Table II presents Insertion Delay (ID) and skew data for the clock tree in the setup-late timing corner. The results show that DAMIL-DCIM achieves lower and more tightly distributed IDs, reducing skew and directly enhancing timing performance.

Experimental results also show clear advantages of our method over the traditional datapath-aware placement. Innovus with SDP maintains the bitlines in the bitcell bank as structured data paths and approximately aligns the related cells. However, the strict constraints of structured data paths significantly limit the placement engine's optimization space, causing legalization failures and suboptimal placement. This leads to excessively long routing wires, increasing parasitics and degrading timing and power performance. To avoid these failures, we re-ran the Innovus SDP flow with a larger target core area. As shown in Table I under **Innovus with SDP***, our DAMIL-DCIM framework still outperforms the Innovus SDP flow in wire length, power, and timing metrics.

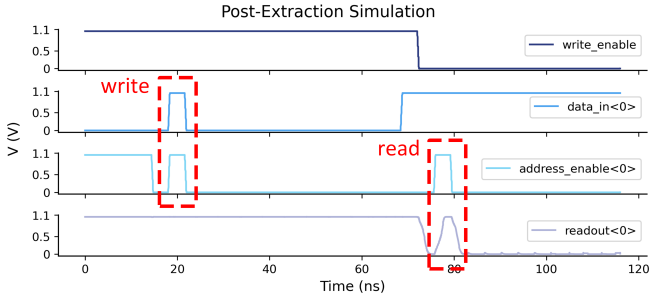


Fig. 10. Post-layout reading and writing simulation result on the first row cells.

B. Verification on Parasitic-Sensitive Paths

We conducted post-layout read and write tests on the bitcell arrays to verify DAMIL-DCIM's ability to manage parasitic-sensitive bitline paths. Simulation results show that all bitcells successfully performed read and write operations. Fig. 10 llustrates an example for a single bitcell. In the first half of the simulation, a write operation sets a high level in the first row, first column bitcell. In the second half, a read operation correctly retrieves the high value. The red dashed boxes in Fig. 10 highlight the critical write and read operations.

V. CONCLUSION

In this work, we present DAMIL-DCIM, a DCIM placement framework integrates global dataflow-aware floorplan with MILP-based detailed placement optimization. We evaluated the effectiveness of our framework on a practical DCIM circuit. Experimental results demonstrate that our framework not only achieves superior PPA metrics but also effectively handles the parasitic-sensitive paths inherent to DCIM circuits, significantly outperforming leading placement methods.

VI. ACKNOWLEDGEMENTS

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711901, partly by the National Natural Science Foundation of China (NSFC) Research Projects under Grant 62474050, Grant 62304052, Grant 62141407, Grant 92473207, Grant 62474051, Grant 92373207 and Grant 62090025, partly by the State Key Laboratory of Integrated Chips and Systems, Fudan University (No. SKLICS-Z202404), and partly by Science and Technology Commission of Shanghai Municipality Project (24JD1400800).

REFERENCES

[1] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of sram-based computing-in-memory for ai edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[2] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–6.

[3] J. Chen, F. Tu, K. Shao, F. Tian, X. Huo, C.-Y. Tsui, and K.-T. Cheng, "Autodcim: An automated digital cim compiler," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[4] M. Liu, X. Tang, K. Zhu, H. Chen, N. Sun, and D. Z. Pan, "Opensar: An open source automated end-to-end sar adc compiler," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[5] E. Chang, J. Han, W. Bae, Z. Wang, N. Narevsky, B. Nikolić, and E. Alon, "BAG2: A process-portable framework for generator-based ams circuit design," in *IEEE Custom Integrated Circuits Conference (CICC)*, 2018.

[6] Z. He, P. Liao, S. Liu, Y. Ma, Y. Lin, and B. Yu, "Physical synthesis for advanced neural network processors," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 833–840. [Online]. Available: https://doi.org/10.1145/3394885.3431625

[7] T. Tao Ye and G. De Micheli, "Data path placement with regularity," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers (Cat. No.00CH37140)*, 2000, pp. 264–270.

[8] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. J. Alpert, E. E. Swartzlander, and D. Z. Pan, "Structure-aware placement techniques for designs with datapaths," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 2, pp. 228–241, 2013.

[9] S. Chou, M.-K. Hsu, and Y.-W. Chang, "Structure-aware placement for datapath-intensive circuit designs," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 762–767. [Online]. Available: https://doi.org/10.1145/2228360.2228498

[10] S. Bae, H.-O. Kim, J. Choi, and J. Park, "Coarse-grained structural placement for a synthesized parallel multiplier," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ser. ISPD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 17–24. [Online]. Available: https://doi.org/10.1145/2717764.2717775

[11] Y. Wang and H. Shin, "Effective regularity extraction and placement techniques for datapath-intensive circuits," *IET Circuits, Devices & Systems*, vol. 11, no. 5, pp. 512–519, 2017. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cds.2016.0249

[12] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based cnns on fpgas," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–4.

[13] J.-M. Lin, W.-F. Huang, Y.-C. Chen, Y.-T. Wang, and P.-W. Wang, "Dapa: A dataflow-aware analytical placement algorithm for modern mixed-size circuit designs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–8.

[14] Y. Chou, J.-W. Hsu, Y.-W. Chang, and T.-C. Chen, "Vlsi structure-aware placement for convolutional neural network accelerator units," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1117–1122.

[15] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan, and J. Hu, "Global placement exploiting soft 2d regularity," in *Proceedings of the 2022 International Symposium on Physical Design*, ser. ISPD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 203–210. [Online]. Available: https://doi.org/10.1145/3505170.3506723

[16] H. Hu, D. Fang, W. Li, B. Yuan, and J. Hu, "Systolic array placement on fpgas," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[17] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori, W.-C. Zhao, D. Sun, M. E. Sinangil, Y.-H. Chen, T.-L. Chou, K. Akarvardar, H.-J. Liao, Y. Wang, M.-F. Chang, and T.-Y. J. Chang, "16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 252–254.

[18] H. Fujiwara, H. Mori, W.-C. Zhao, M.-C. Chuang, R. Naous, C.-K. Chuang, T. Hashizume, D. Sun, C.-F. Lee, K. Akarvardar, S. Adham, T.-L. Chou, M. E. Sinangil, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao, and T.-Y. J. Chang, "A 5-nm 254-tops/w 221-tops/mm2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous mac and write operations," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[19] J.-S. Seo, "Advances and trends on on-chip compute-in-memory macros and accelerators," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.