ARCTIC: Agile and Robust Compute-In-Memory Compiler with Parameterized INT/FP Precision and Built-In Self Test

Hongyi Zhang†, Haozhe Zhu†*, Siqi He†, Mengjie Li†, Chengchen Wang‡,
Xiankui Xiong‡, Haidong Tian‡, Xiaoyang Zeng†, Chixiao Chen†
† State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
‡ State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation, Shenzhen, China
* E-mail: zhuhz@fudan.edu.cn

***Abstract*—Digital Compute-in-Memory (DCIM) architectures are playing an increasingly vital role in artificial intelligence (AI) applications due to their significant energy efficiency enhancement. Coupling memory and computing logic in DCIM requires extensive customization of custom cells and layouts, thus increasing design complexity and implementation effort. To adapt to the swiftly evolving AI algorithms, DCIM compiler for agile customization is required. Previous DCIM compilers accelerate the customization process but only focus on integer computation. Moreover, with technology node scaling down, design-for-test circuits are critical for robust chip design, while previous built-in-self-test (BIST) schemes for traditional memory fail to offer support for DCIM. This paper presents ARCTIC, an agile and robust DCIM compiler supporting parameterized integer/floating-point formats with corresponding BIST circuits. To support variable precision formats (including integer and floating-point), ARCTIC applies adaptive topology and layout optimization schemes for optimal performance. The compiler is also equipped with DCIM-friendly MarchCIM BIST circuits for efficient post-silicon tests with negligible area overhead. The energy efficiency of the generated DCIM macros remains competent with the state-of-the-art counterparts.**

***Index Terms*—Compute-in-memory, Floating-point, BIST, Agile customization**

I. Introduction

Deep neural networks (DNN) have been trending in various artificial intelligence (AI) applications. However, the massive data movement of DNN models has become an obstacle towards their deployment on energy-sensitive edge devices. Compute-in-memory (CIM) technique integrates computing logic into memory, alleviating the data communication bottleneck. Compared with near-memory computing architectures, CIM harvests remarkable energy efficiency improvement.

Among diverse CIM architectures, SRAM-based digital CIM (DCIM) has been intensively studied as a compelling candidate for DNN deployment [1]–[4]. DCIM implements PVT-insensitive lossless computation with in-memory Boolean logic, possessing strong technology scaling ability. However, due to the tight coupling of memory and computing logic, custom-designed cells are required to conduct multiply-and-accumulation (MAC) computation during DCIM read-out. These custom cells are unsupported by traditional digital very-large-scale-integration (VLSI) flow. Digital VLSI circuit design benefits from standardized libraries. The libraries accelerate the customization process

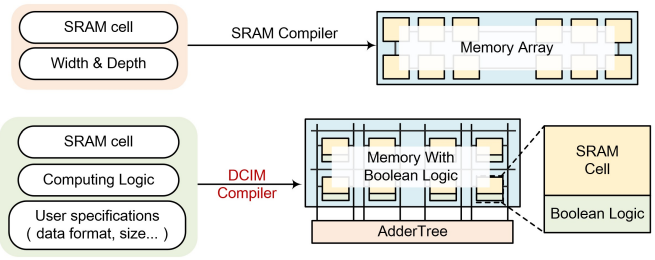

Fig. 1: SRAM compiler versus DCIM compiler

by providing standard-cell timing and power information, omitting transistor-level simulation. Featuring cells of the same height, they also pave the way for efficient placing and routing. Apart from standardized libraries, DCIM is also lacking in specific synthesis and layout constraints for meeting user requirements, and the manual optimization effort further postpones the development process.

Furthermore, with technology scaling down and design complexity improving, chip failure probability increases, and makes design-for-test (DFT) circuits crucial for robust chip design. Previous memory built-in-self-test (BIST) circuits help to swiftly verify conventional SRAM arrays [5], but they only compare the read-out vector with the pre-stored test vector. As the read-out values of the DCIM memory array implement MAC operation before outputting, it is difficult to identify and locate faults with direct comparison. Thus, the standard memory BIST circuits provided by electronic design automation (EDA) tools are unable to offer DCIM BIST support.

Motivated by traditional digital circuit design flow and SRAM compilers, several automated DCIM compilers have been proposed to ease the manual effort for DCIM customization [6], [7], as shown in Figure 1. The compilers generate DCIM macros with a template-based compilation framework according to user specifications, and produce optimized layout with competitive performance compared with handcrafted counterparts. But to date, previous DCIM compilers feature integer (INT) MAC computation only, while floating-point (FP) precision is urgently demanded in diverse scenarios.

To tackle the aforementioned challenges, this paper presents ARCTIC, an agile and robust SRAM-based DCIM compiler with INT/FP precision parameterization and BIST

979-8-3503-4859-0/DATE24/© 2024 EDAA

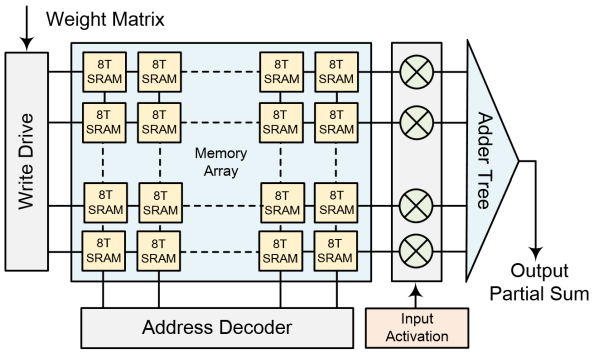

Fig. 2: A typical integer DCIM architecture

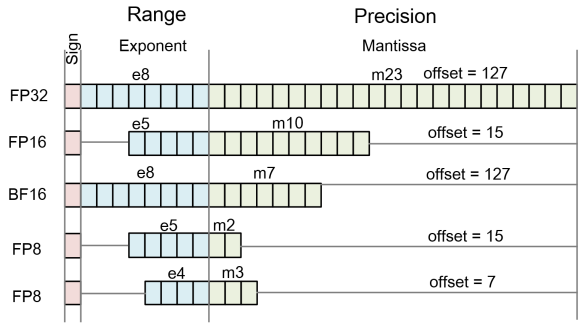

Fig. 3: Commonly Used Floating-point formats

support. The main contributions of this article are as follows:

- We propose a parameterized DCIM compiler with flexible precision formats (including INT1∼32, FP8/16/32, and BF16 etc.). The compiler produces DCIM macros featuring INT/FP MAC operations, and of diverse widths and depths according to the user specifications. ARCTIC is also equipped with an adaptive placing and routing scheme for layout optimization.
- We propose MarchCIM BIST scheme to offer BIST support for both SRAM memory array and computing logic cells in DCIM. The scheme has a high fault coverage rate while possessing a low complexity of $O(6N)$.
- We validate the energy efficiency competence of ARCTIC generated DCIM macros, in comparison with state-of-the-art counterparts. We also evaluate the area overhead of the BIST circuit. Evaluation result shows that the BIST circuit brings about negligible area and power consumption overhead.

The remains of this paper are organized as follows. In Section II, we introduce the preliminaries of DCIM, floating-point computation, and memory BIST. In Section III, we illustrate our compiling framework's implementation. Our evaluation and comparison with other works is displayed in Section IV, and the conclusion is drawn in Section V.

II. Preliminaries

A. INT DCIM Architecture

DCIM architectures have been trending in recent years owing to their remarkable energy efficiency. With technology nodes scaling from 28nm to 5nm, DCIM continues to exhibit great scaling potential [1], [2].

Figure 2 depicts the architecture of a typical DCIM macro, which consists of a write driver, address decoder, memory array, read-out compute logic, and adder tree. For DNN applications, the weights are pre-stored in memory arrays, and implement MAC operation with input activation upon read-out. During integer MAC computation for DCIM, the address decoder produces one-hot encoding with the given address to select a specific column of the memory array. The write driver implements row-wise weight writing of the memory array as well as pre-charging for the read-out computing logic, for performing multiplication upon read-out. The adder tree further accumulates the multiplication results after fixed-bit-width shifting and outputs the MAC result.

B. Floating-point Computation and Formats

MAC computation is the fundamental operation in DNN algorithms. Apart from INT MAC, FP MAC is also required in various DNN applications (i.e., DNN training and fine-tuning). The FP formats commonly used in DNN are shown in Figure 3. An FP number consists of a sign bit, exponent bits, and mantissa bits. With a certain exponent offset, the value of the number can be calculated with the formula:

$$\mathrm{Num} = (-1)^{\mathrm{Sign}} \times 2^{(\mathrm{Exp.}-\mathrm{Offset})} \times 1.\mathrm{Mant}. \quad (1)$$

During an FP MAC operation, first, the operands' exponents are added and sorted to find the maximum sum of exponents. Then, the exponents conduct subtraction with the maximum exponent, for the mantissa multiplication results to shift accordingly. The shifted results are then fed into an adder tree to implement further accumulation.

The different computation flows of mantissa and exponent hinder FP MAC's deployment on DCIM. In DCIM architectures, the DNN weights' exponents and mantissa are stored in different memory arrays. Mantissa shifting and adding operations have to wait for the exponents to generate shifting numbers, and the serial computation flow brings about timing inconsistency. Sharing exponents among a group of weights provides an energy-friendly compression opportunity, and eliminates the delay of weight exponent comparison with negligible accuracy loss [8].

Several FP DCIM macros have been proposed in recent years, exhibiting satisfactory energy efficiency [9], [10], but the tedious handcrafting processes are lacking in agility.

C. Memory BIST

In robust chip designs, memory BIST is adopted to thoroughly verify the functionality of memory arrays. The common faults within memory arrays include stuck-at-fault (SAF), transition fault (TF), address decoder fault (AF), and coupling fault (CF) etc. SAF indicates a fixed 0/1 value for a typical cell, and TF reveals a cell's incapability of switching from 0 to 1 or vice versa. AF identifies the malfunctioning

of decoders, and CF is the case when the changing of a cell's value alters the storage of another cell.

To identify the faults above, memory BIST circuits generate test vectors, and sweep through the memory with pre-stored test patterns. During test pattern traversing, with direct test vector and read-out vector comparison, the malfunctioning cell can be located, and the faulty type can be determined.

Various memory BIST schemes have been proposed for thorough verification, including MSCAN, GALPAT, Walking 1/0, Checkerboard and March [5]. The test schemes differ in test patterns and traversing orders. Among the schemes, the March algorithm stands out with a high fault coverage rate and low testing complexity. Multiple adapted versions of the March algorithm have been established, including MATS, MATS+ and March-X, etc.

However, current memory BIST schemes fail to adapt to DCIM architectures. The DCIM input consists of weight and input activation, and conducts MAC computation upon read-out. The SRAM memory faults and computing logic faults are both possible contributors to test comparison failure, and the peripheral MAC computation impedes the deployment of direct test vector and read-out vector comparison.

III. The Proposed ARCTIC DCIM Compiler

A. Overall Compilation Framework

We propose ARCTIC, an agile and robust DCIM compiler. With the user-provided parameters and constraints, the compiler can customize DCIM structures of different precision formats and sizes, finally generating area-timing-co-optimized macro layouts. The generated macros are also equipped with DCIM-adaptive BIST circuits. The overall framework of the ARCTIC compiler is depicted in Figure 4. It consists of: (1) custom cell characterization (colored in green), (2) template-based DCIM macro customization and MarchCIM BIST circuit generation (colored in yellow), (3) constraint-aware synthesis, placing and routing (colored in blue).

To activate the compilation flow, the input user specifications include timing and area constraints, depth and width of the memory array, mantissa width, exponent width, and exponent offset.

During compilation, first, the 8T memory cell and computing cells are handcrafted to be of the same height as foundry-provided cells, to root for a compact layout design. The custom cells' post-layout resistance and capacitance parasitics are extracted with commercial tools. Libraries featuring timing and power, as well as Verilog models are characterized afterward, and ARCTIC performs a formality check for the user-input schematic and corresponding Verilog models. With the libraries and models, the custom cells are incorporated into standard cell libraries; thus commercial EDA tools can be utilized for synthesis as well as layout optimization.

After the custom cell library characterization process, based on the user-specified INT/FP data format and memory

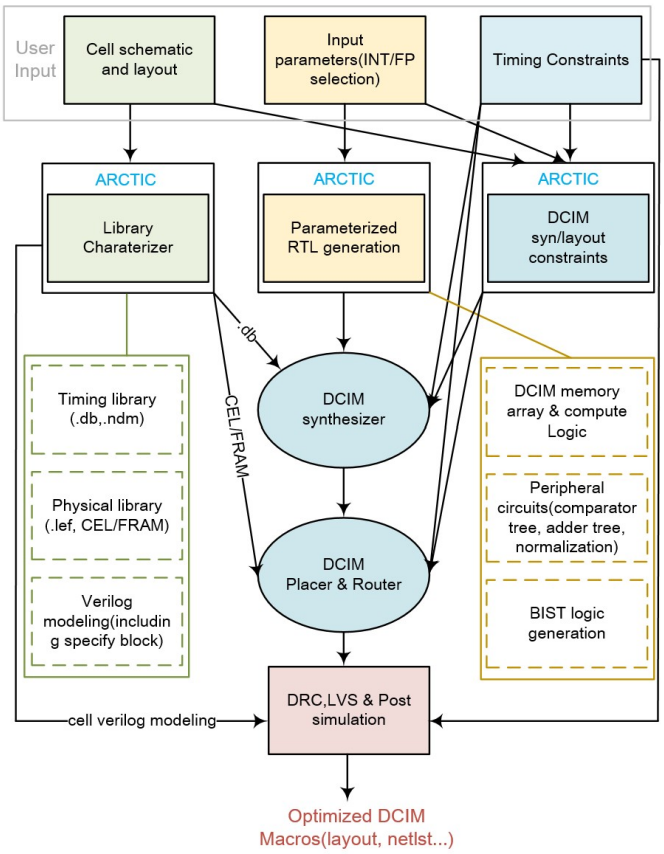

Fig. 4: The overall ARCTIC compilation framework

array size, a low-level RTL file of the DCIM macro is customized with parameterized and template-based generic framework. The corresponding MarchCIM-based BIST circuits are also generated. The BIST circuits equipped with CIM-friendly test patterns are later inserted into the RTL, and is capable of locating memory and computing logic faults during post-silicon test.

Upon RTL generation, synthesis based on the input timing and area constraints is implemented with the characterized custom cell library. Compact template-based placing and routing are then carried out for ideal latency and energy efficiency performance. After DRC and LVS verification, post-layout simulation is performed with the custom and standard cells' Verilog models. While performing timing checks and power analysis, the post-simulation result gives feedback to optimize the synthesis and place and route constraints.

B. Topology of the ARCTIC-generated INT/FP DCIM Macro

Figure 5 shows the topology of ARCTIC-generated macros, which adds FP customization extensions to a state-of-the-art DCIM macro structure [2]. During customization, the memory array, adder tree, and comparator tree are generated with a parameterized template. For the multi-bit multiplication of INT/FP formats, a read-out latch is coupled with multiple AND gates to accomplish bit-parallel MAC operation. The latch is also equipped with an enable signal

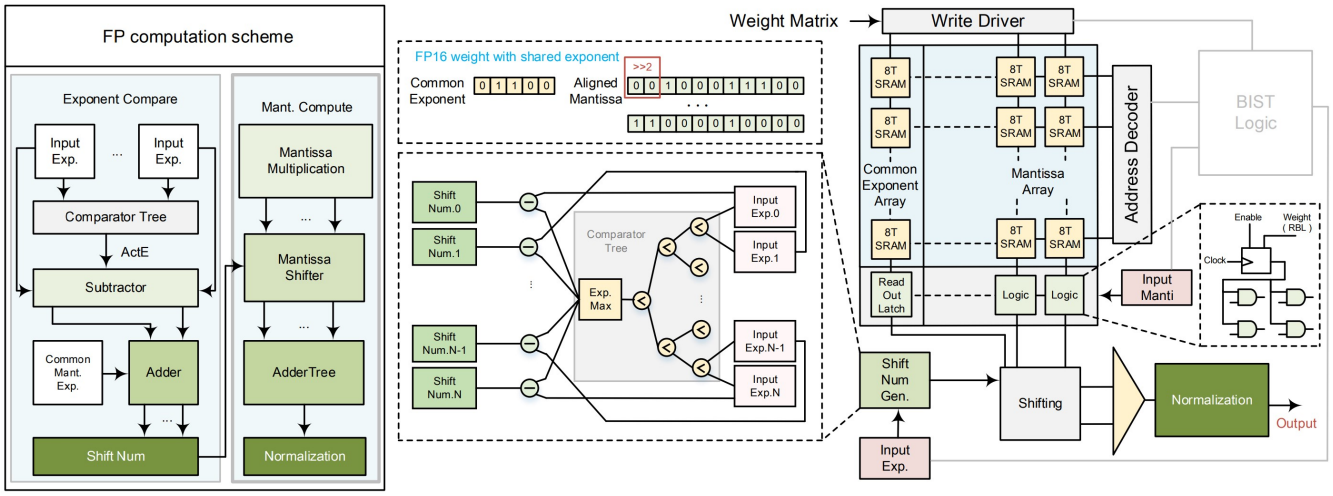

Fig. 5: ARCTIC DCIM macro architecture for FP precision

to offer gating support in specific sparsity scenarios. Weights are directly stored in the memory arrays for INT DCIM macros. Concerning FP DCIM, macros are customized with the shared-exponent scheme for better latency and energy efficiency. Thus, only the common exponent is written in the exponent array for the stored mantissa to share during computation.

For INT MAC operations, computation is accomplished by in-memory multiplication and fixed bit width shifting. For FP MAC operations, first, the exponents of input activation are compared to find the maximum activation exponent (`ActE`). After comparison, the `ActE` is subtracted by the activation exponents, and the subtraction results are then added to the unified weight exponent, so as to determine the unfixed shifting number for multiplication results. An additional shifting window is provided to preserve the accuracy of shifted mantissa (i.e., 10 bits for FP16). The shifted results are then fed into an adder tree for accumulation. To present the MAC result in standard floating-point formats, a normalization module is designed based on the user-provided data width, exponent width, and exponent offset.

For layout optimization, concerning the driving capacity of the write diver and compute logic, we divide the bit lines to adjust the storage-computation ratio. Clock-gating of sub-arrays and adder tree sharing are utilized to trade-off between TOPS/area and TOPS/energy performance. For FP DCIM macros, adaptive mapping is deployed by placing the common exponent bits in the middle of the mantissa array for optimal routing. The above topology is shown in Figure **??**. Write drivers and compute logic are arranged next to the memory array for minimal data transfer cost, and the address decoder is located amid the memory array to pave the way for routing. The address decoder selects a certain column during computation.

C. Architecture of MarchCIM-based BIST circuits

To pave the way for DCIM post-silicon tests, MarchCIM BIST scheme is proposed. The scheme is adapted from

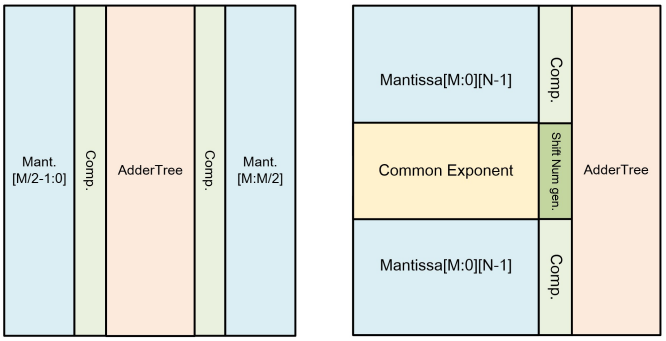

Fig. 6: Adaptive mapping for (a) INT, (b) FP data formats

March-X, which possesses a high fault coverage rate with minimal complexity among the memory BIST schemes. Extensions are added to test vectors for computing logic verification, and test patterns are modified to accommodate the peripheral MAC computation. The read-out values of the DCIM memory array follow the formula:

$$Output = w_0 \times a_0 + w_1 \times a_1 + ... + w_n \times a_n \quad (2)$$

As the output is affected by both input activation and weight, tests for both memory cells and read-out computing cells are required for fault detection. While SRAM cells build up the memory array, the DCIM read-out computing logic generated by ARCTIC features a latch-based structure, and the operating principle is similar to SRAM. Thus, identical test cases can be deployed for memory and logic verification.

While testing several rows of memory/logic cells of a certain mantissa/exponent, the weight is written with the test pattern, and the corresponding input activation is set to numerical 1 to enable direct write and read comparison, and vice versa. Cells of other exponents and mantissa are masked as zero to avoid interleaved interference caused by MAC computation. To accomplish the masking process, addresses for both column index and row index are demanded.

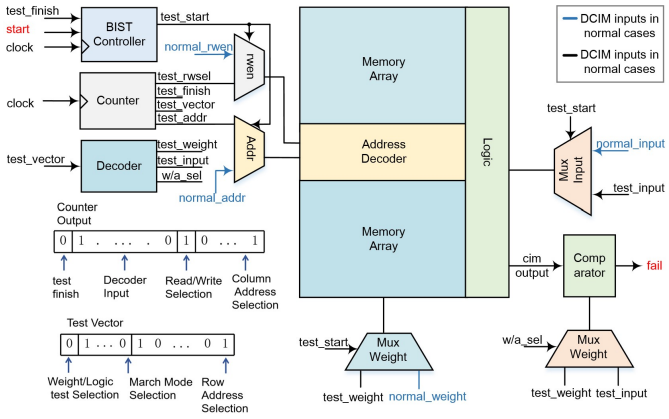

Fig. 7: MarchCIM BIST architecture

TABLE I: MarchCIM BIST Test Vector

Weight Sel	Mode Sel[2:0]	Weight	Activation
0	000	00000000	Numerical 1
	001	00001111	
	010	00110011	
	011	01010101	
	100	11111111	
	101	11110000	
	110	11001100	
	111	10101010	
1	000	Numerical 1	00000000
	001		00001111
	010		00110011
	011		01010101
	100		11111111
	101		11110000
	110		11001100
	111		10101010

Circuits and corresponding test vectors of the proposed MarchCIM BIST scheme is illustrated in Figure 7. The circuit features a state-machine-based controller, a counter, a test vector decoder, and a test pattern comparator. During a MarchCIM BIST procedure, the controller and counter functions with the start signal. The counter input data contains test ending, read/write enable, memory column address and test vector signals. The decoder receives the test vector, which consists of memory/logic test selection, test mode selection and element-wise row address signals. Then, the decoder produces the read/write state, memory column address and data pattern to be fed into the memory array. During the process, the comparator identifies the difference between the test vector and the read-out vector, in search of inoperative memory/logic cells. After going through all cases, the counter gives out the finish signal to the controller and ends the test.

The test vector of MarchCIM for INT/FP DCIM macros is shown in Table I for both INT8 and FP8 cases. During test vector traversing, the memory/logic cells go through the following cases: (1) initialization with all "0"s, (2) read out zero values and write "1" to the test cell in ascending order, (3) read out "1" and write "0" to each cell in descending order, (4) read out "0" of each test cell in descending order. After traversing the 0/1 value storage test and flipping scenarios in both orders, the faults previously mentioned can be detected.

TABLE II: MarchCIM BIST Area Overhead

Format	Element Size	Memory Size	Area Overhead
INT16	32x128	64kb	2.47%
INT8	32x128	32kb	3.41%
INT4	32x128	16kb	3.30%
FP16	32x128	45kb	2.30%
BF16	32x128	33kb	2.70%
FP8(E5M2)	32x128	0.8kb	3.77%

For INT macro memory test, an input activation element is located with the row address, and its least significant bit is set to one. The corresponding weight element is identified with both row and column addresses, and is written with the pattern indicated by the mode selection signal. The DCIM macro output is truncated for comparison.

For FP macro memory test, to adapt to varied floating-point formats, exponents and mantissa are managed separately. The exponent of the row-column-address-located input activation is set to the exponent's offset, and the mantissa is set to zeros to represent numerical 1, in correspondence to IEEE 754 standard. The identified weight is written with the test mode, and with the normalization module, the output can be directly utilized for comparison.

IV. Evaluation

With the ARCTIC compiler, DCIM macros of various INT/FP formats are generated with 28nm CMOS technology, and thorough experiments are conducted at 0.9V power supply, 200Mhz clock frequency. In Section A, we evaluated the BIST area overhead of the generated DCIM macros featuring diverse INT/FP data formats. In Section B, we experimented with the customized DCIM macros for energy efficiency, and compared the performance with state-of-the-art DCIM counterparts.

A. Area evaluation

We evaluated the area overhead of MarchCIM circuit for ARCTIC-generated DCIM macros with different INT/FP formats. It can be seen that the area overhead of MarchCIM BIST stays below 4%. The results are shown in Table II.

B. Energy efficiency evaluation

We simulated the power consumption of the generated macros with Synopsys PrimeTime PX tool, utilizing the characterized standard cell library and Verilog models for post-layout simulation.

Figure 8 depicts the layout of an INT4 DCIM macro and an FP16 DCIM macro customized with the compiler. The INT4 macro uses sub-array partitioning for storage-computation ratio adjustment, and the FP16 macro's exponent bits are placed in the middle of the macro. The energy breakdown of the INT4 and FP16 macros are depicted in Figure 9. As shown int the figure, the proposed BIST circuit's power overhead stays below 4%.

TABLE III: Comparison with Other Handcrafted DCIM Macro Works

	ISSCC'21 [3]	VLSI'21 [11]	ISSCC'22 [10]	ISSCC'23 [9]	**This Work**		
Technology	65nm CMOS	28nm CMOS	28nm CMOS	28nm CMOS	28nm CMOS		
Architecture	DCIM	DCIM	DCIM	DCIM	DCIM		
BIST Support	No	No	No	No	Yes		
Frequency	100MHz	40MHz	333MHz	147MHz[2]	200MHz		
Precision Format	INT2/4/8	BF16	INT4/8	INT8/BF16	INT4	INT8	BF16[3]
Memory Capacity	16kb	160kb	32kb	64kb	16kb	32kb	33kb
Energy efficiency (TOPS/W)[1]	6.05 @ INT8[2] 24.20 @ INT4[2]	2.80 @ BF16[3]	19.21 @ INT8[2] 42.00 @ INT4[2]	19.50 @ INT8[2] 14.04 @ BF16[3]	32.67	14.83	10.43

1.Evaluated at 0.9V voltage supply 2.Scaled from 1bit report or Estimated from figure 3.Evaluated with 50% input sparsity

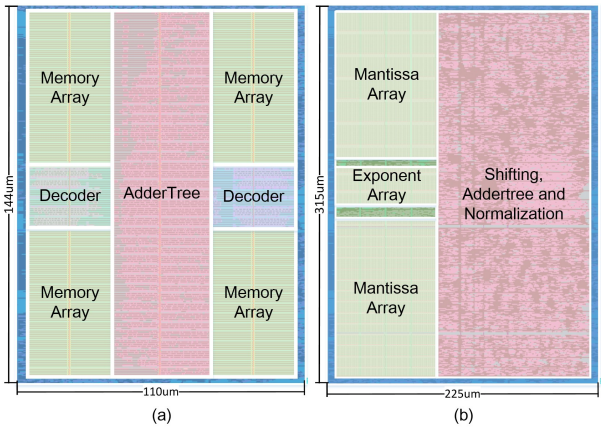

Fig. 8: Layout of the ARCTIC-generated DCIM macros with (a) INT4, (b) FP16 precision formats

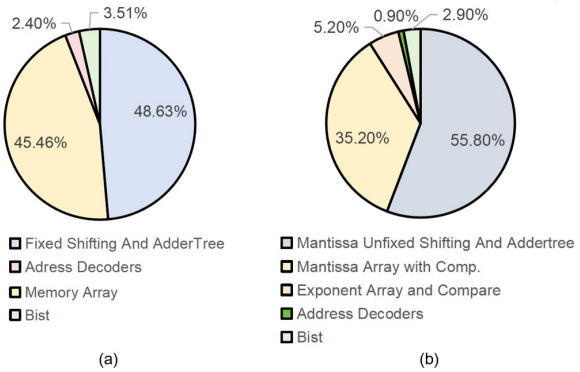

Fig. 9: Power Breakdown of ARCTIC-generated DCIM macros with (a) INT4, (b) FP16 precision formats

The energy efficiency comparison of the generated macros with state-of-the-art DCIM counterparts is shown in Table III. It can be seen that the generated macros possess competitive energy efficiency performance for diverse INT/FP data formats. Compared with previous DCIM compilers [6], [7], ARCTIC offers flexible INT/FP format support and an efficient post-silicon BIST scheme.

V. Conclusion

This paper presents ARCTIC, a DCIM compiler featuring parameterized INT/FP formats with MarchCIM-based BIST circuits. ARCTIC customizes macros with sub-array partitioning considering driving capacity, and employs an adaptive mapping scheme for routing and energy reduction. The generated macros are also equipped with MarchCIM BIST circuits for swift and thorough verification with negligible area and power consumption overhead. The energy efficiency of ARCTIC-generated macros is competent with state-of-the-art DCIM counterparts.

Acknowledgement

This work was supported by the National Key Research and Development Program of China under Grant No. 2022YFB4500100, the National Natural Science Foundation of China (NSFC) under Grant No. 62304047 and No. 62322404, and the Strategic Priority Research Program of CAS under Grant No. XDB44000000.

References

[1] H. Fujiwara *et al.*, "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 *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 186–187.

[2] H. Zhu *et al.*, "Comb-mcm: Computing-on-memory-boundary nn processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 250–251.

[3] J. Yue *et al.*, "15.2 a 2.75-to-75.9tops/w computing-in-memory nn processor supporting set-associate block-wise zero skipping and ping-pong cim with simultaneous computation and weight updating," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021.

[4] S. Liu *et al.*, "16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2023, pp. 250–252.

[5] G. Hantos, D. Flynn, and M. P. Y. Desmulliez, "Built-in self-test (bist) methods for mems: A review," *Micromachines*, vol. 12, no. 40, pp. 1–28, 2021.

[6] J. Chen *et al.*, "Autodcim: An automated digital cim compiler," in *ACM/IEEE Design Automa. Conf. (DAC)*, 2023.

[7] J. Zhang *et al.*, "Alpine: An agile processing-in-memory macro compilation framework," in *ACM/IEEE Great Lakes Symp. VLSI (GLSVLSI)*, 2021.

[8] W. Zhao *et al.*, "Optimizing fpga-based dnn accelerator with shared exponential floating-point format," in *IEEE Trans. Circuits Syst. I*, 2023.

[9] A. Guo *et al.*, "A 28nm 64-kb 31.6-tflops/w digital-domain floating-point-computing-unit and double-bit 6t-sram computing-in-memory macro for floating-point cnns," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2023.

[10] B. Yan *et al.*, "A 1.041-mb/mm2 27.38-tops/w signed-int8 dynamic-logic-based adc-less sram compute-in-memory macro in 28nm with reconfigurable bitwise operation for ai and embedded applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022.

[11] J. Lee *et al.*, "A 13.7 tflops/w floating-point dnn processor using heterogeneous computing architecture with exponent-computing-in-memory," in *IEEE Int. Symp. VLSI Circuits (VLSI-C)*, 2021.