iMCU: A 28-nm Digital In-Memory Computing-Based Microcontroller Unit for TinyML

Chuan-Tung Lin, *Graduate Student Member, IEEE*, Paul Xuanyuanliang Huang, *Graduate Student Member, IEEE*, Jonghyun Oh, *Member, IEEE*, Dewei Wang, and Mingoo Seok, *Senior Member, IEEE*

***Abstract*— Tiny machine learning (TinyML) envisions executing a deep neural network (DNN)-based inference on an edge device for improving battery life, latency, security, and privacy. Toward this vision, recent microcontroller units (MCUs) integrate in-memory computing (IMC) hardware to leverage its high energy efficiency and throughput in vector–matrix multiplication (VMM). However, those existing works require large IMC hardware, severely increasing the area overhead. In addition, most existing works use analog–mixed-signal (AMS) IMC hardware, exhibiting limited robustness over process, voltage, and temperature (PVT) variations. Finally, none can support a practical software development framework such as TensorFlow Lite for Microcontrollers (TFLite-micro). Due to these limitations, those MCUs did not present the performance for the standard benchmark MLPerf-Tiny, which makes it difficult to evaluate them against the state-of-the-art neural (not necessarily IMC-based) MCUs. In this article, we design a new IMC-based MCU, titled iMCU, for TinyML to address those challenges. In the design process, we: 1) define the optimal set of acceleration targets and 2) devise an area-efficient computation flow that requires the least amount of IMC hardware yet still provides a significant acceleration. In addition, we develop: 1) state-of-the-art digital IMC macros and 2) create the accelerator based on the macros, which can support the proposed computation flow in a fully pipelined manner. Combining those innovations, we prototyped the iMCU in a 28-nm CMOS. The measurement results show that the iMCU significantly outperforms the prior IMC-based MCUs in compute density, energy efficiency, and SRAM density (total SRAM size/total SRAM area). It also achieves a compact footprint of 2.73 mm^2.**

***Index Terms*— Deep learning, hardware/software co-design, in-memory computing (IMC), microcontroller units (MCUs), neural network accelerators, tiny machine learning (TinyML).**

I. Introduction

The advancements in machine learning (ML) have made us envision ultra-low-power microcontroller units (MCUs) with limited power and memory budget to perform ML tasks at the edge. Tiny ML (TinyML), which aims to collect data, execute ML models, and analyze the data in real-time on ultra-low-power devices near sensors, provides critical benefits, such as security and privacy. TinyML can also reduce latency and extend the battery life by avoiding the cost of transmitting data wirelessly [1], [2], [3], [4], [5], [6], [7].

The intensive computation required by ML inference motivates much research in custom hardware design. SRAM-based in-memory computing (IMC) has been proposed for improving energy efficiency and throughput in vector–matrix multiplication (VMM) [8], [9], [10], [11], [12], [13], [14], [15]. The conventional digital accelerator architecture requires accessing data in on-chip SRAM, one row at a time, which limits the throughput and energy efficiency. On the other hand, by combining the memory cells and computation elements inside a memory macro, IMC can perform multiple multiply-and-accumulate (MAC) operations without row-by-row accesses. The custom hardware design also reduces the macro's area and the lengths of critical wires, resulting in reduced dynamic power consumption.

Most existing IMC-based MCUs use analog–mixed-signal (AMS) IMC macros, which use capacitors and transistors for computation and analog-to-digital converters (ADCs). AMS IMC is capable of achieving high energy efficiency and area efficiency. However, analog computing hardware may produce incorrect VMM results across process, voltage, and temperature (PVT) variations, thereby degrading the accuracy of inferences [16]. Digital IMC hardware, on the contrary, uses digital arithmetic circuits, such as compressors, adders, and accumulators, performing MAC operations robustly across PVT variations [16], [17], [18], [19], [20], [21], [22]. But digital IMC hardware tends to consume more silicon area.

On the other hand, in developing an MCU, we must co-optimize its hardware and software stack to map ML tasks to resource-constrained devices flexibly. Supporting a software development flow to port ML models onto an MCU is important. Such a development flow should include model development (data engineering, model selection, and hyperparameter tuning/neural architecture search) and model deployment (software suite, model compression, and code generation). TensorFlow Lite for microcontrollers (TFLite-micro)

Manuscript received 7 October 2023; revised 14 December 2023 and 15 January 2024; accepted 31 January 2024. This article was approved by Associate Editor Meng-Fan Chang. This work was supported by NSF under Grant 1919147. *(Corresponding author: Mingoo Seok.)*

The authors are with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: ms4415@columbia.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3362274.
Digital Object Identifier 10.1109/JSSC.2024.3362274

is one of such flows. It can optimize TensorFlow models and convert the model file into a reduced-size binary file with less complexity. Unfortunately, none of the existing IMC-based MCUs can support such a practical software development framework [1], posing a significant limitation.

To address these challenges, in this article, we present a new digital IMC-based MCU, titled iMCU [23], which integrates a 32-b RISC-V-based MCU with a digital IMC accelerator. We architect the iMCU to improve energy efficiency, latency, and silicon area. Specifically, we optimally choose acceleration targets and devise an area-efficient computation flow that requires the least amount of additional hardware yet still provides a significant acceleration. We also design the state-of-the-art digital IMC circuits (titled D6CIM, [24]) and a fully pipelined accelerator based on them. In addition, we developed a custom iMCU library, and the iMCU supports a standard software development flow for the standard benchmark MLPerf-Tiny. We also investigated the performance of the iMCU while sweeping various microarchitecture parameters such as IMC sizes, scratchpad sizes, bus widths, and clock speeds.

We prototyped the iMCU in a 28-nm CMOS. The measurement results show that the iMCU significantly outperforms the best prior IMC-based MCU [22] by 288$\times$ in the figure-of-merit (FoM), which is defined as a product of compute density, energy efficiency, and SRAM density (SRAM size/SRAM area). Using only a small amount of IMC hardware, it also achieves a compact footprint of 2.73 mm^2.

The rest of this article is organized as follows. Section II introduces the existing MCUs for TinyML and IMC-based MCUs. Section III presents the hardware architecture and software development framework of the iMCU. We then describe the proposed computation flow and microarchitecture optimization in Section IV. Section IV details the IMC accelerator architecture and the digital IMC macro. We discuss the measurement results in Section V and conclude this article in Section VI.

II. Related Works

We have been seeing different companies and research groups propose neural accelerators to improve computing performance for on-device inference. ARM presents the Ethos-U55 micro neural processing unit (microNPU), which can work with the Cortex-M processor to offer better energy efficiency for MAC operations [25]. Syntiant offered the neural decision processor (NDP), which contains a deep neural network (DNN) datapath for always-on applications in battery-powered devices [26]. Silicon Labs proposed the matrix–vector processors (MVPs), which work as co-processors to accelerate the matrixed floating-point (FP) multiplications and additions, offloading intensive computations from an MCU core.

On the other hand, some of the recent processors have integrated IMC-based accelerators to perform efficient VMM for ML inference. Jia et al. [27] integrated a mixed-signal SRAM-based IMC accelerator for VMM and a digital near-memory-computing accelerator for vector elementwise computation. Ueyoshi et al. [28] use an analog SRAM-based IMC core for high energy efficiency for low-precision computation and a digital custom datapath for higher precision computation. Such an approach can execute some layers with the analog IMC while other layers with the digital custom datapath, depending on the precision requirement of each layer. Chang et al. [29] proposed a Cortex M3 core with embedded RRAM-based IMC and vector modules. The RRAM module is a high-density and non-volatile data storage, and the vector module supports vector addition, multiplication, and activation functions. Wang et al. [22] presented a hybrid in-/near-memory compute SRAM using an 8T transposable bitcell.

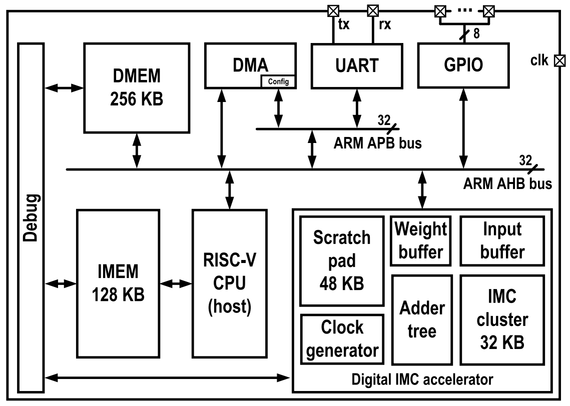


Fig. 1. Proposed iMCU architecture.

III. Architecture Design

A. Hardware Architecture and Software Framework

Fig. 1 details the overall organization of the iMCU, which consists of: 1) a 32-b RISC-V CPU core (host processor); 2) a digital IMC accelerator which contains the IMC macro cluster; 3) instruction memory (IMEM); 4) data memory (DMEM); 5) a direct memory access (DMA) module; 6) a universal asynchronous receiver–transmitter (UART); 7) a general-purpose IO (GPIO); and 8) a 32-b ARM AHB/APB bus.

Fig. 2 shows the matching software development framework for the iMCU. It starts with training an 8-b DNN model via TensorFlow, which produces a TF file. Then, we convert the TF file into the TFLite file by fusing a batch norm layer into a convolution layer. This helps avoid adding explicit hardware support for batch-norm-related computation. The next step is to convert the TFLite file into the C header file (model.cc). Then, we compile the header file with the input data file (input.cc) and the TFLite-micro library file using the RISC-V g++ compiler, which produces a binary file. We convert the binary file into instruction and data hexadecimal files and store them in IMEM and DMEM, respectively.

Using the framework, we develop the software for the following DNN models: tiny-conv, tiny-embedding-conv (both from TFLite-micro [1]), and ResNetv1 (from MLPerf-Tiny, an open-source benchmark suite, provides a set of DNNs in C++ to evaluate MCUs [2]) (see Fig. 3). ResNetv1 achieves 86.96% on CIFAR-10, and tiny-conv achieves 91.34% on GSCD (four keywords).

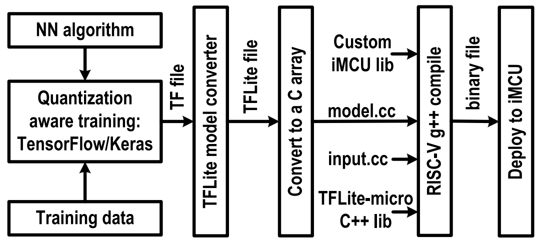


Fig. 2. Proposed software development framework for the iMCU.

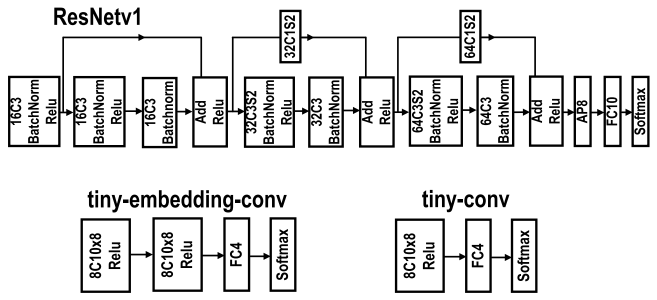


Fig. 3. Target neural network models: ResNetv1, tiny-embedding-conv, and tiny-conv. Abbreviations: 16C3: 16 features $3times3$ convolution, AP8: $8times8$ average pooling, FC10: ten fully connected.

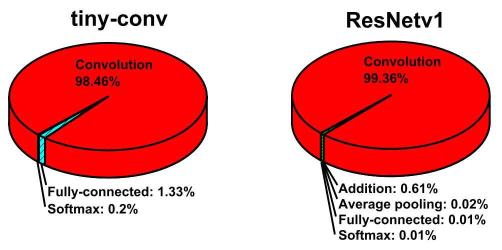


Fig. 4. Workload profiling: tiny-conv and ResNetv1.

B. Workload Profiling and Division

We begin our iMCU architecture design by identifying the parts of a DNN workload worth acceleration. This allows us to incorporate minimal hardware in the accelerator to support those parts only. We first profiled the computation complexity of each layer using SPIKE (a functional simulator for RISC-V processors [30]). The simulation results show that the convolution layer is the most dominant, followed by the addition layer (see Fig. 4). From these data, we can estimate that if we accelerate convolution layers by 500×, we can reduce the total cycle count by 119×. If we accelerate addition layers, as well, we estimate to gain an additional 3.6× speedup (total 434×) (see Fig. 5). We have found that all other layers (pooling, fully connected, softmax) are not worth accelerating since they only marginally reduce the total cycle count.

C. Area-Efficient Computation Flow and Memory Sizing

We develop the computation flow (sequences) that requires the least amount of IMC hardware yet still delivers a significant acceleration. Many existing works use arbitrarily large amounts of IMC hardware to store more than one (potentially all) layer of weight data of a DNN model before starting computation [22], [27], [29]. Such architecture, however, severely increases area overhead since IMC hardware is generally larger than regular SRAM.

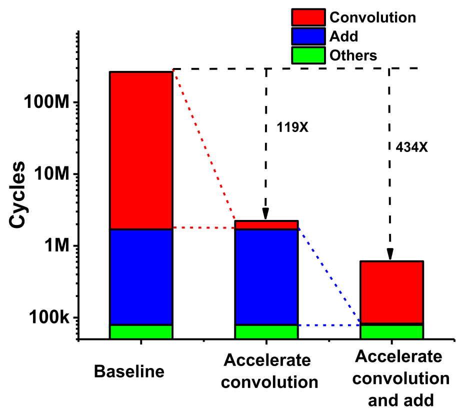
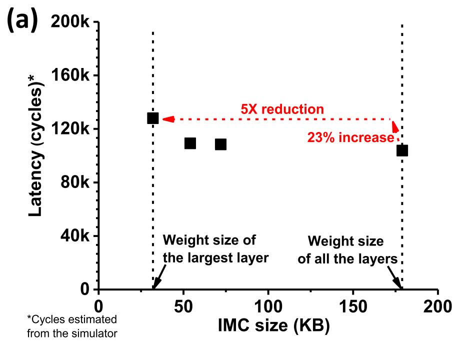


Fig. 5. Latency improvement estimation shows that convolution and add layers are worth accelerating.



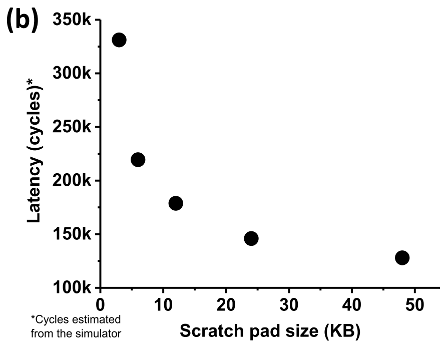


Fig. 6. (a) Proposed computation flow enables 5× IMC area reduction at a 23% latency penalty. (b) Small scratch pad cannot buffer the largest output data, worsening latency.

In this work, we devise an alternative flow in which DMEM stores all the weights. Since DMEM is implemented in the dense foundry 6T bitcells, it has a more compact area. The IMC hardware buffers the weight data of only one layer right before the accelerator computes the layer. This can largely reduce the size requirement of the IMC hardware. The cost is to increase data movement costs between DMEM and IMC hardware. However, we found that the area savings largely outweigh the cost: it reduces the IMC hardware's area by 5× at a 23% increase in the cycle count [see Fig. 6(a)]. This is because we perform 100–10 000 VMMs for each layer, thereby amortizing the data movement cost over those many VMMs.

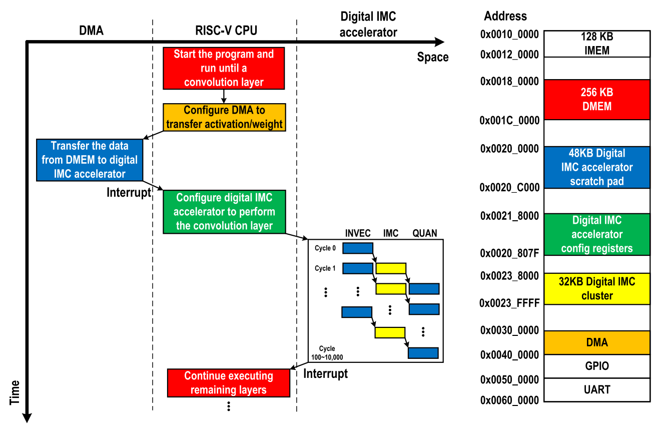

DMA
RISC-V CPU
Digital IMC accelerator
Address
Space
Start the program and run until a convolution layer
Configure DMA to transfer activation/weight
Transfer the data from DMEM to digital IMC accelerator
Interrupt
Configure digital IMC accelerator to perform the convolution layer
INVEC
IMC
QUAN
Cycle 0
Cycle 1
Cycle 100~10,000
Interrupt
Continue executing remaining layers
Time
0x0010_0000
0x0012_0000
128 KB IMEM
0x0018_0000
256 KB DMEM
0x001C_0000
0x0020_0000
48KB Digital IMC accelerator scratch pad
0x0020_C000
0x0021_8000
Digital IMC accelerator config registers
0x0020_807F
0x0023_8000
32KB Digital IMC cluster
0x0023_FFFF
0x0030_0000
DMA
0x0040_0000
GPIO
0x0050_0000
UART
0x0060_0000

Fig. 7. Proposed computation flow and the memory map.

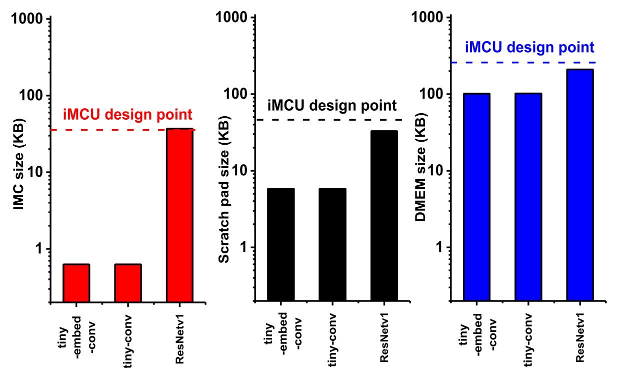


Fig. 8. Based on the proposed computation flow, we set the sizes of the IMC macro cluster, the in-accelerator scratch pad, and the DMEM to support the target workloads.

We also envision the scratch pad in the accelerator to fully buffer the output of one layer to be used as the next layer's input. This helps avoid costly DMEM accesses. As shown in Fig. 6(b), if the scratchpad size is smaller than the largest output activation data among layers, we must temporarily buffer the data in the main memory (DMEM), increasing latency. We found that the largest output data size is 32 kB, setting the scratchpad size to 48 kB.

Fig. 7 shows the proposed computation flow for an end-to-end inference. First, the host starts the program. When it reaches a convolution layer (or an addition layer), it configures DMA to transfer the weight data of the layer from the DMEM to the IMC cluster. Only for the input layer does it transfer the input data from the DMEM to the in-accelerator scratchpad. Upon data transfer completion, the host configures layer-related configurations such as input, filter, and output dimensions, stride and padding sizes, input and output offsets, and the starting addresses of the input, weight, output data accesses, etc. In addition, the host configures which digital IMC macros to use so the accelerator can clock-gate unused macros. Then, the accelerator starts to compute on the layer, which involves many iterations of three sub-tasks: input vector preparation, IMC operation, and output quantization. Finally, it stores the layer output in the scratchpad and then interrupts the host. The host resumes executing the program.

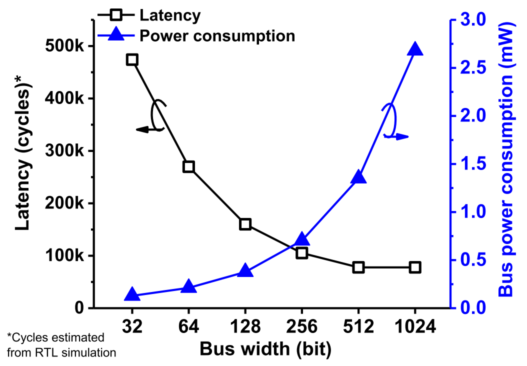


Fig. 9. Impact of the bus width on latency and bus power consumption.

Based on the computation flow, we determine the sizes of the memory blocks and create the memory map (see Fig. 8). Again, the IMC accelerator must buffer only one layer at a time. We choose ResNetv1 as our target model since it is the most complex model among the MLPerf-Tiny benchmark. Therefore, we can set the IMC cluster size to 32 kB (= 512 × 512 b), roughly matched to the largest layer of the target model. The largest layer has 64 output channels and thus requires the column size of the IMC cluster to be equal to or greater than 512 b (= 64 × 8 b) for optimal mapping. Hence, we set one dimension of the cluster to 512 b. Similarly, we set the scratchpad size to 48 kB, slightly larger than the largest output data size (32 kB). In addition, we found that the largest model we target has 179 kB of weight data. Thus, we set the DMEM size to 256 kB. In addition, we set the size of IMEM to 128 kB to store the largest program among the targets. The size of each memory is summarized in the memory map (see Fig. 7).

D. Bus Width

The proposed computation flow moves weight data from DMEM to the accelerator (specifically the IMC cluster) via an on-chip bus. A wider bus improves the latency but increases power consumption. To investigate this tradeoff, we sweep the bus widths from 32 to 1024 b and simulate the latency and power consumption. As shown in Fig. 9, regarding the computation of one convolution and one addition layer, a 1024-b-wide bus reduces the latency over a 32-b bus by 6×. Still, it increases the power consumption of the bus by 21×. For the iMCU, we chose a 32-b bus to reduce bus power consumption and silicon area.

E. Performance Analysis

We analyze the computation speedup that the proposed accelerator enables, compared with the case that only the host processor fulfills all computation needs. We use the functional simulator (SPIKE) for this analysis. Overall, it achieves 434× speedup for ResNetv1. As shown in Fig. 10, it achieves higher speedup for the computation of layers 1–5 because those layers contain small weight data yet large input and output data (see Fig. 11). Such layers can reuse the same weights across more inputs, amortizing the performance overhead of moving

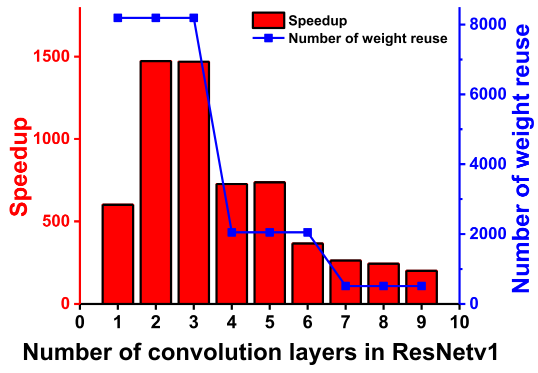


Fig. 10. Speedup and weight reuses across the layers of ResNetv1.

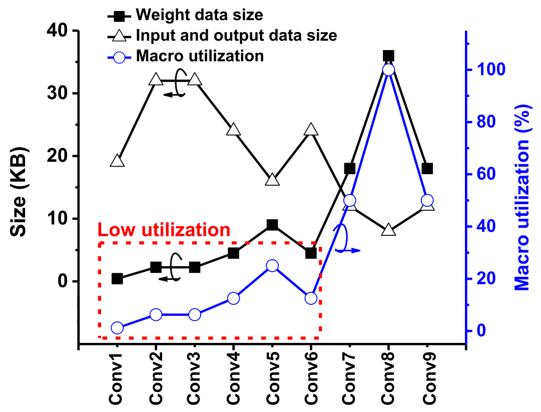


Fig. 11. Weight, input, and output data size across the layers of ResNetv1. The macro utilization is low for the layers having small weight data and large input and output data.

weight data from the DMEM to the IMC cluster. On the other hand, computing layers with large weight data but small input and output data pronounces the overhead of the data transfer between the DMEM and the IMC cluster.

On the other hand, Fig. 11 shows the low macro utilization for some of the layers. The two root causes of this low utilization are twofold: 1) most CNN models have an imbalanced weight data size among layers and 2) the iMCU uses the IMC cluster that can fit the weight data of the largest layer. Therefore, when the iMCU computes the smaller layers, it inevitably exhibits low utilization. We could improve the macro utilization by reducing the IMC cluster size. However, it can significantly increase the latency for computing the layers that cannot fit in the IMC cluster.

IV. IMC Accelerator Architecture

A. Pipeline Architecture

We devise the microarchitecture of the IMC accelerator to support the computation flow in a fully pipelined manner. Fig. 12 shows the proposed microarchitecture of the accelerator. It has three stages, each designed to take the same 64 cycles for the fully pipelined operation.

The first stage (INVEC) prepares input vectors. It uses two 512-B buffers operating in a ping-pong fashion to hide the latency. Specifically, one buffer grabs 8-B data per cycle from the scratchpad over 64 cycles. In parallel, the other buffer feeds an input vector, again 8 B per cycle, to the IMC macro cluster. The second stage (IMC) performs VMM using the 4×4 IMC macro cluster (512×512 b). The cluster can complete one multiplication between an 8-b 512d (dimension) vector and an 8-b 64×512d matrix in 64 cycles. Because the data width from the AHB bus is 32 b while the data width into the IMC macro is 128 b, the weight buffer stores the weight data temporarily before writing into the IMC macro.

The last stage (QUAN) performs the quantization. The IMC stage's result can have up to 25 bits, but we need to quantize them to 8 b before storing them in the scratchpad. Simply removing the LSBs is not optimal for inference accuracy. Instead, we adopted a quantization scheme from TFLite-micro [1], where the quantized value q is defined as

$$q = 2^n \cdot M_0 \cdot (r + Z) \qquad (1)$$

where n, M_0, and Z are the offline-computed hyperparameters and r is the IMC stage's result. Since QUAN has 64 cycles to quantize a 64-D vector, QUAN uses only one two-input 32-b adder, one two-input 64-b multiplier, and one 32-b shifter. Note that although not ideal, the iMCU can support a layer having the input and output size larger than the scratchpad. Since it requires additional data loading, it incurs a latency penalty.

B. Digital IMC Macro

Building upon [24], we designed a digital IMC macro for the accelerator. Fig. 13 shows its circuit schematics. It uses a time-sharing architecture to improve area efficiency. Specifically, the macro uses 128×128 compact 6T bitcells to store the NN weights. Every 16 bitcells share two multiplication units (NOR gates), and every 128×8 bitcells time-share a set of eight 15-4 compressors and an adder tree.

The macro performs an 8-b 128×16d (dimension) VMM in 64 clock cycles. It first activates two consecutive MAC wordline (MWL) in each sub-module, which transfers two weight bits to the two NOR gates in that sub-module. At the same time, the row peripheral feeds the corresponding two input activation bits via INbs to the NOR gates. Every eight columns generate 16 8-b partial products. The compressors and adder tree then add up 16 partial products and produce partial sums, and the shift-accumulator performs shift-and-accumulate of the partial sums. We repeat this process eight times while feeding the rest of the input bits in the bit-serial fashion and then again eight times for providing the rest of the inputs corresponding to the weights in each sub-module. Finally, the macro produces the VMM result, a 23-b 16-D vector. The output is 23 b since the output bitwidth is equal to the summation of the number of row address bits (7), weight precision (8), and activation precision (8).

The macro achieves the state-of-the-art energy efficiency, compute density, and weight density [24]. The macro achieves an excellent weight density of 126 kB/mm^2. In addition, the macro achieves a compute density of 1.25 TOPS/mm^2 (0.12 TOPS/mm^2) at 1 V (0.6 V) and an energy efficiency of 20.78 TOPS/W (43.24 TOPS/W) at 1 V (0.6 V) with a

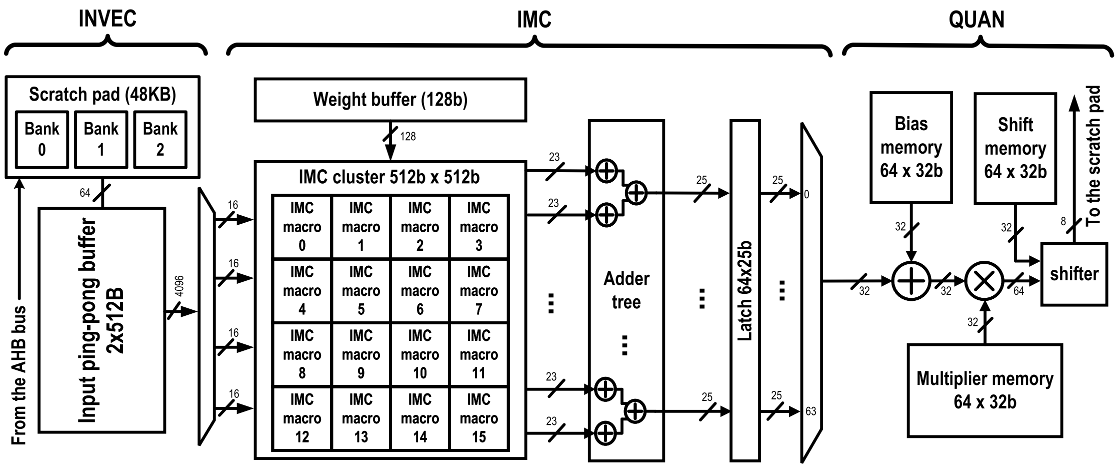


Fig. 12. IMC accelerator microarchitecture.

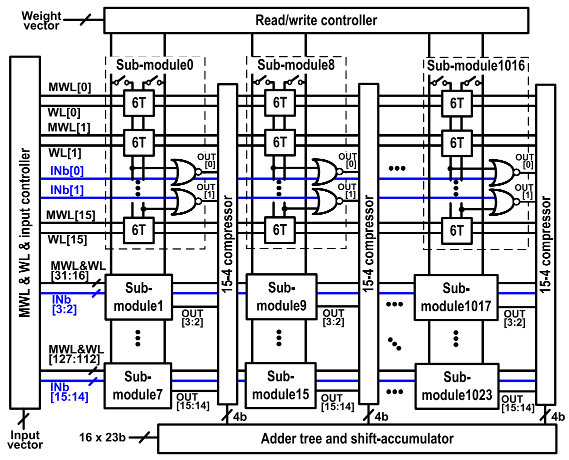


Fig. 13. Proposed 128 × 128 digital IMC macro.

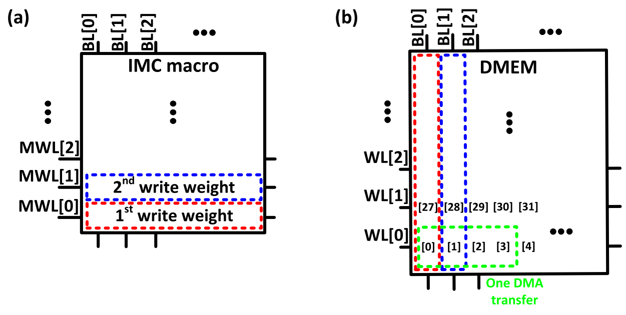


Fig. 14. (a) Our IMC macro has MWLs orthogonal to BLs. (b) Regular SRAMs, such as the DMEM and the scratchpad, store and access data in a row-major fashion.

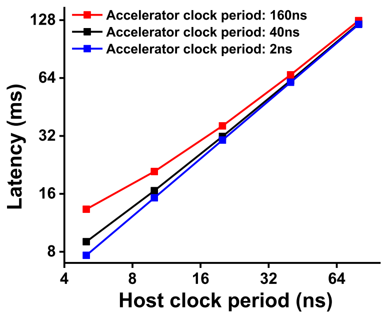


Fig. 15. Inference latency across the host and accelerator clock periods.

TABLE I
Configuration Registers of the Accelerator

Name	Description	Bitwidth
en_cpu_access	Enable cpu to read or write data	1b
Start: clk_sel	Select: no clock, host clock, IMC local clock	2b
sel_input_addr	Select the address where input data is stored	2b
sel_output_addr	Select the address where output data is stored	2b
output_partial_sum	Output is the partial sums of the next computation	1b
bias_partial_sum	Bias is the partial sums of the next computation	1b
ack_layer_done	Acknowledgement of the interrupt	1b
compute_add_layer	Enable executing add layer	1b
imc_macro_usage	Indicate which IMC macros are used	16b

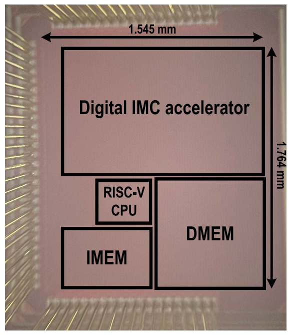


Fig. 16. Die micrograph.

50% input sparsity. Note that our model does not exhibit 50% sparsity. We used the 50% sparsity data to evaluate the IMC macro's energy efficiency, which has been commonly done in prior digital IMC works [19], [20]. We implement fully digital circuits, including compressors and adders, to ensure high robustness over PVT variations. The macro uses exact arithmetic (no approximation), thus exhibiting 0% root mean square percentage error (RMSPE).

We investigated the impact of MWL direction on the data transfer between the IMC macros and the DMEM. As shown

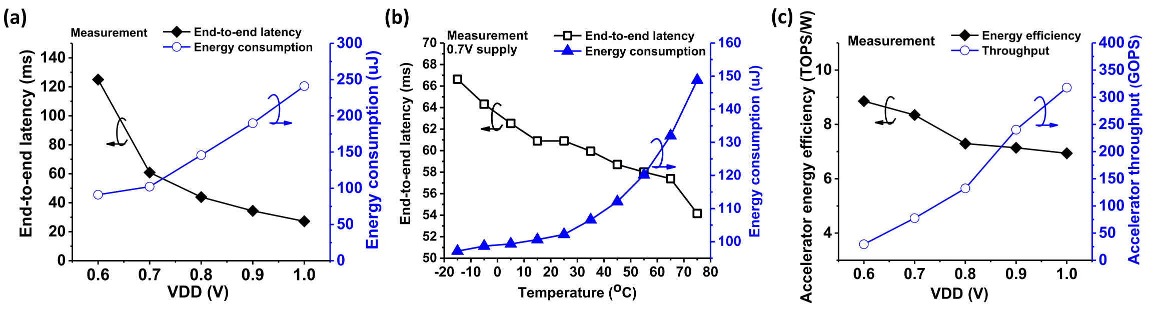


Fig. 17. Measurement results. End-to-end latency and energy consumption in performing ResNetv1 (a) across VDDs and (b) temperatures. (c) Accelerator energy efficiency and throughput measurement across VDDs.

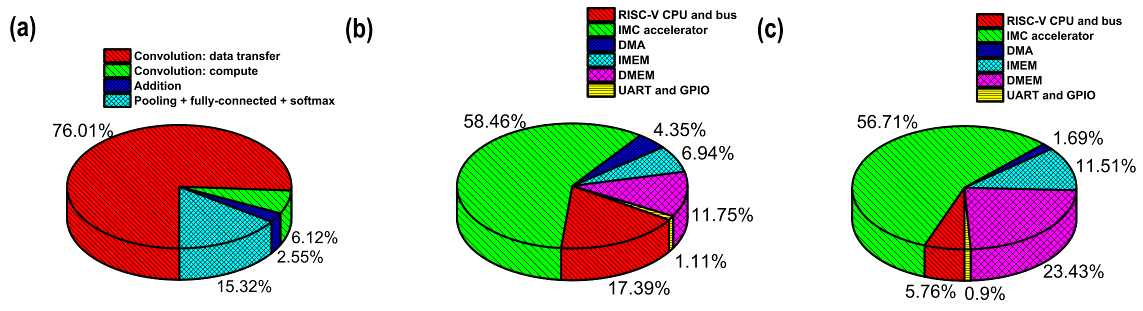


Fig. 18. (a) Latency, (b) energy consumption, and (c) area breakdown of the iMCU.

in Fig. 14(a), we chose our macro to have MWLs orthogonal to BLs, like some of the prior macros [8], [16], [17], [18]. This orientation is also more suitable to the proposed macro's data flow. Therefore, our macro has to store the weight data in a column-major fashion, but it receives an input vector in a row-major fashion. On the other hand, other SRAMs, such as the DMEM and the scratchpad, store and access the weight and input data in a row-major, as shown in Fig. 14(b). As a result, the IMC macro and the DMEM have a mismatch in the direction of storing the weight data.

To mitigate this difficulty, we decided to modify the program compilation process. Essentially, between compilation and program loading, we transpose the weight matrix. This modification enables the DMA to move weight data from the DMEM to the IMC macros in the same continuous address order. We have considered other options, such as modifying the DMA to support the data transfer between two memory blocks having different addressing orders. However, we did not choose them since they increase hardware overhead.

C. Accelerator Clock

We use dual clocks for the host and the accelerator. The IMC accelerator operates under either the host or IMC local clock. We can also gate its clock for power savings. We can select the clock by writing into the configuration register, which controls the multiplexer. Initially, the IMC accelerator's clock is gated. For the data transfer between DMEM and the accelerator, the IMC accelerator uses the host clock. Then, it switches to the local clock for the computation. We disable the host clock before clock switching to avoid metastability. Once the accelerator finishes the computation, it interrupts the host, which then gates the clock of the accelerator.

We sweep host and accelerator clock frequencies to investigate their impact on the latency performance. As shown in Fig. 15, if the host clock is slow, the accelerator clock has little impact on the overall latency. This is because the DMA operation dominates the execution time. If the host clock is fast (its period shorter than 10 ns), the accelerator clock will notably impact the latency performance. At the 5-ns host clock period, speeding up the accelerator clock from 160 to 40 ns (2 ns) improves the latency by 1.5× (1.74×). However, using fast clock indeed increases power consumption. It also increases the energy consumption when the host, DMA, and bus are idle (during the convolution/addition operation). Based on this tradeoff, we set the host clock period to 40 ns (25 MHz) and the accelerator clock to 5 ns (200 MHz).

In addition, we developed the clock gating for each macro to save the clock power consumption for unused macros. As shown in Fig. 11, for ResNetv1, except for the largest layer (conv8), all the other layers use less than 50% of the macros in the accelerator. Hence, we gate the clock for each idle macro by setting the imc_macro_usage register (see Table I). For ResNetv1, it can improve the energy consumption of the iMCU by 37%.

D. Configuration Interface

The host configures the accelerator via programming special registers in the accelerator. Table I summarizes the registers. The *start* register allows the host to select the clock sources or gate clock. We can configure the addresses of the input data and the output data to be stored through *sel_input_addr* and *sel_output_addr* registers. We can set the *output_partial_sum* and *bias_partial_sum* registers to support a layer whose weight data does not fit the IMC cluster size. They allow the accelerator to buffer the results of partial layer computation.

TABLE II
COMPARISON TO STATE-OF-THE-ART MCUs

		This work	JSSC20 [22]	JSSC20 [27]	JSSC23 [28]	syntiant-9120-1v1-98mhz Syntiant [2]	xG24-DK2601B Silicon Labs [2]	NUCLEO-H7A3ZI-Q STMicroelectronics [2]
Technology [nm]		28	28	65	22	n/a	n/a	n/a
Architectural features	Host processor	RISC-V 32b	Cortex-M0 32b	RISC-V 32b	RISC-V 32b	HiFi3 + Cortex-M0 32b	Cortex-M33 32b	Cortex-M7 32b
	Accelerator	Digital IMC	Digital IMC	Analog IMC	Analog IMC	Digital accelerator	Digital accelerator	n/a
	Activation precision [bit]	8	1-32	1-8	7	1-16	8	32
	Weight precision [bit]	8	1-32	1-8	1.5	1-16	8	32
	IMEM size	128KB	16KB	128KB	512KB	n/a	1.5MB5)	2.06MB5)
	DMEM size	256KB	16KB	128KB	512KB	304KB	256KB	1.4MB
	IMC size	32KB	96KB	73.75KB	72KB	n/a	n/a	n/a
	In-accelerator scratch pad size	48KB	0KB	0KB	320KB	1024KB	n/a	n/a
	Total SRAM size	464KB	128KB	329.75KB	832KB	1328KB	256KB	1.4MB
	Total SRAM area [mm^2]	0.933	1.225	4.626	5.414	n/a	n/a	n/a
	Total area [mm^2]	2.03	1.85	8.56	10.24	7.75	n/a	n/a
	IMC density [KB/mm^2] (IMC size/IMC area)	**125.8**	104.5	25.2	31.4	n/a	n/a	n/a
	SRAM density [KB/mm^2] (Total SRAM size/total SRAM area)	**497.4**	104.5	71.3	153.7	n/a	n/a	n/a
Hardware performance	Supply voltage [V]	0.6-1	0.6-1.1	0.85-1.2	0.55-0.9	0.9-1.1	n/a	0.74-1.3
	Operating frequency [MHz]	6-35 (host) 29-310 (accelerator)	**114-475**	40-100	50-340	30-98	40-78	280
	Macro compute density [TOPS/mm^2]	**1.25 (1V,8b,8b)**	0.0273 (1.1V,8b,8b)	0.0094 (1.2V,8b,8b)2)	**12.88 (0.8V, 7b,1.5b)**	n/a	n/a	n/a
	Macro energy efficiency [TOPS/W]	**43.24 (0.6V,8b,8b)1)**	0.56-5.27 (0.6V,8b,8b)	6.25 (0.85V,8b,8b)2)	**600 (0.55V,7b,1.5b)**	n/a	n/a	n/a
	Accelerator compute density [TOPS/mm^2]	**0.301 (1V,8b,8b)**	0.0273 (1.1V,8b,8b)	0.0094 (1.2V,8b,8b)2)	**6.65 (0.9V,7b,1.5b)**	n/a	n/a	n/a
	Accelerator energy efficiency [TOPS/W]	**8.86 (0.6V,8b,8b)**	0.56-5.27 (0.6V,8b,8b)	6.25 (0.85V,8b,8b)2)	**298 (0.6V,7b,1.5b)**	n/a	n/a	n/a
	FoM = acc. compute density × acc. energy efficiency × SRAM density	**1039 (1V, 8b, 8b)**	3.61 (0.6V, 8b, 8b)	2.01 (1.2V, 8b, 8b)	**179893 (0.9V, 7b, 1.5b)**	n/a	n/a	n/a
MLPerf-tiny: ResNetv1 on CIFAR104)	Latency [ms]	60.9	n/a	n/a	n/a	**5.1**	239.98	158.13
	Energy consumption [uJ]	**102.18**	n/a	n/a	n/a	139.4	2248.02	4151.13

1) Simulated. 2) Normalized to 8b weights and 8b activations. 3) n/a: not available. 4) The top-1 accuracy of all systems are above 85%, meeting the quality target in the benchmark suite.

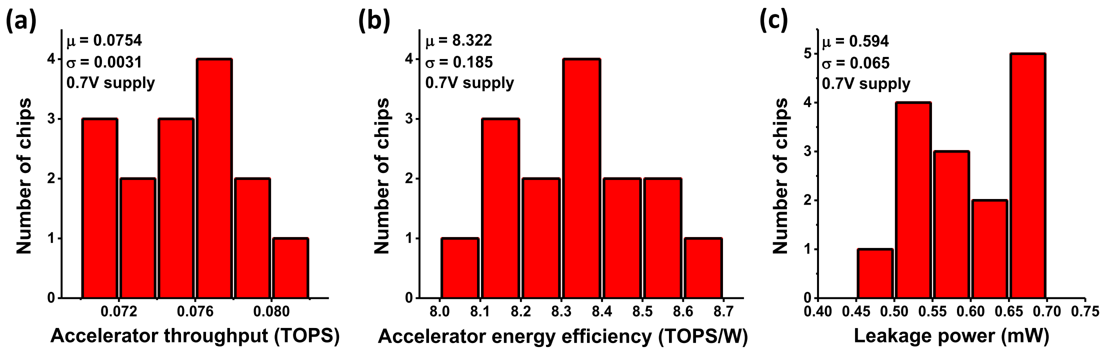


Fig. 19. 15-chip measurement at 0.7 V and room temperature. (a) Accelerator throughput, (b) energy efficiency, and (c) leakage power.

The *compute_add_layer* register configures the accelerator to compute an add or convolution layer. The host can clock-gate each IMC macro through the imc_macro_usage register. The host also sets the *ack_layer_done register* to acknowledge the accelerator when the accelerator interrupts the host.

V. MEASUREMENT RESULTS

We prototyped the iMCU in a 28-nm CMOS. It takes 2.73 mm^2. Fig. 16 shows the die photograph. The iMCU can perform an end-to-end inference with various TinyML models. For ResNetv1, it takes 60.9 ms and consumes 102.18 μJ per inference at 0.7 V. Fig. 17(a) shows the latency and energy consumption to perform ResNetv1 across VDDs. As shown in Fig. 17(b), we measure the end-to-end latency and energy consumption across different temperatures. When the temperature increases from -15 °C to 75 °C, the latency decreases by 19%, and the energy consumption increases by 1.53×.

Fig. 18 shows the latency, energy, and area breakdown. After the acceleration of convolution and addition, other operations such as pooling, fully connected, and softmax take a larger latency portion of 13%. The IMC accelerator consumes 59% and 57% of the total energy and area, respectively.

Fig. 19 shows the maximum accelerator throughput, energy efficiency, and leakage power across 15 dies at 0.7-V supply.

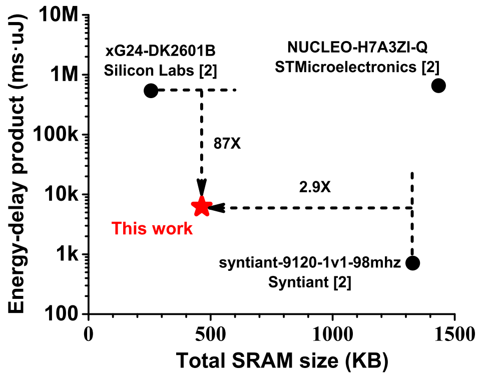


Fig. 20. EDP and SRAM size comparisons to the industrial MCU prototypes.

It gives the standard deviation over a mean (σ/μ) of 0.04 for throughput, 0.002 for energy efficiency, and 0.11 for leakage.

We also count the 8-b additions and multiplications that the IMC macros execute (OP_IMC) and divide it with the total energy consumption of the digital IMC accelerator, which gives the energy efficiency FoM of 8.86 TOPS/W at 0.6 V. At the same VDD, the accelerator achieves a throughput of 29.4 GOPS. Fig. 17(c) shows the accelerator-level energy efficiency and throughput across VDDs.

In addition, we divide the OP_IMC by the total energy consumption of the iMCU, presenting the MCU-level energy efficiency of 7.26 TOPS/W and the MCU-level compute density of 4.11 TOPS/mm^2 at 0.6 V. The prior IMC-based MCU works do not report the MCU-level, end-to-end inference energy efficiency. Therefore, we compare it to the ASIC [10], finding that the iMCU still provides 6.4% higher end-to-end inference efficiency and full programmability.

We compare the iMCU to the recent IMC-based MCUs. As shown in Table II, compared with the state-of-the-art digital IMC-based MCU [22], the iMCU achieves 1.7× better accelerator energy efficiency and 11× better accelerator compute density. The iMCU also achieves 5× better SRAM density since the iMCU uses the least amount of IMC hardware size and stores the weight data in the dense foundry SRAM. In the FoM, the product of those three metrics, the iMCU achieves 288× improvement over the prior state-of-the-art IMC-based MCU [22]. The analog IMC-based MCU [14] attains high energy efficiency and compute density, but the activation and weight precision are limited to 7 and 1.5 b, respectively. In addition, analog computing circuits are prone to PVT variations and may produce incorrect outputs.

We also compare the iMCU to the industry prototypes. The industry prototypes do not provide the aforementioned metrics, such as energy efficiency, compute density, and SRAM density. Still, they report the on-chip SRAM size, latency, and energy consumption of the end-to-end inference using ResNetv1. Therefore, we make a comparison based on those metrics. As shown in Fig. 20, when compared with syntiant-9120-1v1-98mhz, the iMCU achieves a similar energy-delay product (EDP) while using 2.9× less SRAM. Compared with xG24-DK2601B, the iMCU achieves an 87× better EDP while using a similar amount of on-chip SRAM.

VI. Conclusion

This article presents the iMCU for TinyML edge devices. It uses state-of-the-art digital IMC hardware for ensuring correct inference results across PVT variations. We propose a computation flow and several microarchitecture-level optimizations to use the least amount of IMC hardware but achieve significant acceleration. In addition, the iMCU is fully programmable, supporting an industrial software development framework. The test chip is prototyped in a 28-nm CMOS. The measurement shows the accelerator energy efficiency of 8.86 TOPS/W. The accelerator achieves a compute density of 0.301 TOPS/mm^2 and an SRAM density of 497.4 kB/mm^2. The iMCU achieves 288× better FoM over the prior art.

References

[1] D. Robert et al., "Tensorflow lite micro: Embedded machine learning for tinyml systems," in *Proc. Mach. Learn. Syst.*, vol. 3, A. Smola, A. Dimakis, and I. Stoica, Eds., 2021, pp. 800–811. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2021/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf
[2] C. Banbury et al., "MLPerf tiny benchmark," 2021, *arXiv:2106.07597*.
[3] C. R. Banbury et al., "Benchmarking TinyML systems: Challenges and direction," 2020, *arXiv:2003.04821*.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
[6] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, vol. 16, 2016, pp. 770–778.
[8] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.
[9] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism," *IEEE J. Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, Jul. 2020.
[10] B. Zhang et al., "PIMCA: A programmable in-memory computing accelerator for energy-efficient DNN inference," *IEEE J. Solid-State Circuits*, vol. 58, no. 5, pp. 1436–1449, May 2023.
[11] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, Jan. 2019.
[12] N. Verma et al., "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Mag.*, vol. 11, no. 3, pp. 43–55, Aug. 2019.
[13] H. Jia et al., "Scalable and programmable neural network inference accelerator based on in-memory computing," *IEEE J. Solid-State Circuits*, vol. 57, no. 1, pp. 198–211, Jan. 2022.
[14] P. Houshmand et al., "DIANA: An end-to-end hybrid digital and analog neural network SoC for the edge," *IEEE J. Solid-State Circuits*, vol. 58, no. 1, pp. 203–215, Jan. 2023.
[15] B. Zhang et al., "A 177 TOPS/W, capacitor-based in-memory computing SRAM macro with stepwise-charging/discharging DACs and sparsity-optimized bitcells for 4-bit deep convolutional neural networks," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2022, pp. 1–2.
[16] D. Wang et al., "DIMC: 2219TOPS/W 2569F2/b digital in-memory computing macro in 28 nm based on approximate arithmetic hardware," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2022, pp. 266–268.
[17] B. Yan et al., "A 1.041-Mb/mm^2 27.38-TOPS/W signed-INT8 dynamic-logic-based ADC-less SRAM compute-in-memory macro in 28 nm with reconfigurable bitwise operation for AI and embedded applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 188–190.
[18] Y.-D. Chih et al., "16.4 an 89TOPS/W and 16.3TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22 nm for machine-learning edge applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 252–254.

[19] H. Fujiwara et al., "A 5-nm 254-TOPS/W 221-TOPS/mm^2 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) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[20] H. Mori et al., "A 4 nm 6163-TOPS/W/b 4790-TOPS/mm^2/b SRAM based digital-computing-in-memory macro supporting bit-width flexibility and simultaneous MAC and weight update," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 132–134.

[21] F. Tu et al., "A 28 nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 reconfigurable digital CIM processor with unified FP/INT pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[22] J. Wang et al., "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, Jan. 2020.

[23] C.-T. Lin, P. X. Huang, J. Oh, D. Wang, and M. Seok, "IMCU: A 102-μJ, 61-ms digital in-memory computingbased microcontroller unit for edge TinyML," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2023, pp. 1–2.

[24] J. Oh, C.-T. Lin, and M. Seok, "D6CIM: 60.4-TOPS/W, 1.46-TOPS/mm^2, 1005-Kb/mm^2 digital 6T-SRAM-based compute-in-memory macro supporting 1-to-8b fixed-point arithmetic in 28-nm CMOS," in *Proc. IEEE 48th Eur. Solid-State Circuits Conf. (ESSCIRC)*, 2023, pp. 1–11.

[25] *Ethos-U55*. Accessed: Feb. 10, 2024. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55

[26] *Syntiant Neural Decision Processors*. Accessed: Feb. 10, 2024. [Online]. Available: https://www.syntiant.com/hardware

[27] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

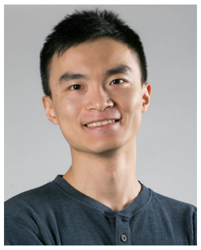
[28] K. Ueyoshi et al., "DIANA: An end-to-end energy-efficient digital and ANAlog hybrid neural network SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[29] M. Chang et al., "A 40 nm 60.64TOPS/W ECC-capable compute-in-memory/digital 2.25MB/768KB RRAM/SRAM system with embedded cortex M3 microprocessor for edge recommendation systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[30] *Spike RISC-V ISA Simulator*. Accessed: Feb. 10, 2024. [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim

Chuan-Tung Lin (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2019, and the M.S. degree in electrical engineering from Columbia University, New York, NY, USA, in 2021, where he is currently pursuing the Ph.D. degree.

His research interests include energy-efficient architecture for machine learning algorithms and VLSI circuit and system design.



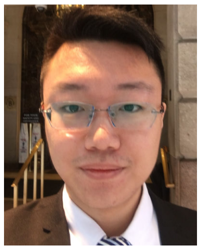
Paul Xuanyuanliang Huang (Graduate Student Member, IEEE) received the bachelor's degree in electrical engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2018, and the master's degree in electrical engineering from Columbia University, New York, NY, USA, in 2020, where he is currently pursuing the Ph.D. degree with the VLSI Laboratory led by Professor Mingoo Seok.

His research interests include design of high-performance accelerator circuits for computing applications.

Jonghyun Oh (Member, IEEE) received the B.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2015 and 2021, respectively.

In 2021, he was a Post-Doctoral Researcher with the Inter-University Semiconductor Research Center, Seoul National University. From September 2021 to March 2023, he was a Post-Doctoral Research Scientist at Columbia University, New York, NY, USA. Since July 2023, he has been working on a custom circuit design as a Hardware Engineer at Apple, Cupertino, CA, USA. His main research interests include high-speed interfaces, integrated voltage regulators, synthesizable digital architectures, neuromorphic hardware accelerator design, and digital-type computing-in-memory macro design.

Dr. Oh has served as a reviewer for various journals, including IEEE JOURNAL OF SOLID-STATE CIRCUITS, IEEE SOLID-STATE CIRCUITS LETTERS, IEEE ACCESS, and IEEE OPEN JOURNAL OF CIRCUITS AND SYSTEMS.



Dewei Wang received the B.S. degree in biomedical engineering (BME) from Xi'an Jiaotong University, Xi'an, China, in June 2017, and the M.S. degree in BME and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in December 2018 and May 2023, respectively.

His research interests include energy-efficient neuromorphic hardware and algorithm design.

Mingoo Seok (Senior Member, IEEE) received the B.S. degree (summa cum laude) from Seoul National University, Seoul, South Korea, in 2005, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 2007 and 2011, respectively, all in electrical engineering.

He was a Member of the Technical Staff with Texas Instruments Inc., Dallas, TX, USA, in 2011. Since 2012, he has been with Columbia University, New York, NY, USA, where he is currently an Associate Professor of electrical engineering. His current research interests include ultra-low-power SoC design for emerging intelligent systems, machine learning VLSI architecture and circuits, variation, voltage, aging, thermal-adaptive circuits and architecture, on-chip integrated power circuits, and nonconventional hardware design, including in-memory computing and analog–mixed-signal computing hardware.

Dr. Seok is/was the Technical Program Committee Member for several conferences, including the IEEE International Solid-State Circuits Conference (ISSCC) and the ACM/IEEE Design Automation Conference (DAC). He received the 1999 Distinguished Undergraduate Scholarship from the Korea Foundation for Advanced Studies, the 2005 Doctoral Fellowship from the Korea Foundation for Advanced Studies, and the 2008 Rackham Pre-Doctoral Fellowship from the University of Michigan. He received the 2009 AMD/CICC Scholarship Award for picowatt voltage reference work and the 2009 DAC/ISSCC Design Contest for the 35-pW sensor platform design. He received the 2015 NSF CAREER and 2019 Qualcomm Faculty Awards. He also received the Best Paper Award from the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS in 2022. He serves/served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS from 2013 to 2015, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS from 2015 to 2023, and IEEE SOLID-STATE CIRCUITS LETTERS from 2017 to 2022. He also served as the Guest Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS (JSSC). He is selected as the Solid-State Circuits Society (SSCS) Distinguished Lecturer for 2023–2025.