A 65-nm Digital Stochastic Compute-in-Memory CNN Processor With 8-bit Precision

Jiyue Yang, *Member, IEEE*, Tianmu Li, Wojciech Romaszkan, Puneet Gupta, *Fellow, IEEE*, and Sudhakar Pamarti, *Senior Member, IEEE*

***Abstract*— Compute-in-memory (CIM) is an emerging solution that embeds compute logic inside the memory array to achieve high energy efficiency and reduce memory access cost. However, current CIM solutions require bulky analog-to-digital converter (ADC)/digital-to-analog converters (DACs) that make the accelerator power hungry and unable to scale in the more advanced process with a low power supply. Stochastic computing (SC) is an attractive alternative digital compute scheme that uses tiny bit-serial logic, e.g., AND and OR, to perform the basic MAC operations on stochastic sequences. In this work, we propose a stochastic CIM (SCIM) approach that combines the benefits of SC and CIM accelerators. It eliminates the DACs/ADCs in CIM and adds CIM's benefits to SC. We demonstrate a highly programmable convolutional neural network (CNN) accelerator that achieves up to 7.96-TOPS/W peak energy efficiency in 65 nm and inference accuracy comparable to 8-b fixed point.**

***Index Terms*— Convolutional neural network (CNN), compute-in-memory (CIM), deep learning, hardware accelerator, stochastic computing (SC).**

I. Introduction

DEEP neural networks (DNNs) have been widely used in the applications of computer vision, voice recognition, and natural language processing [1], [2]. Deploying deep learning algorithms on mobile edge devices can significantly reduce decision latency and protect users' privacy, but the edge device's limited energy budget poses a big challenge to the hardware's energy efficiency. DNNs use a large number of convolution layers to improve accuracy, which leads to increasing model sizes. State-of-the-art models for ImageNet dataset require >100 MB of parameters and $>10^9$ of multiplication and accumulation (MAC) operations [3]. A general deep learning accelerator such as GPU and FPGA can process large DNN models on a workstation. It accommodates a wide range of applications and programmable computation precisions for training and inference. However, GPUs require a large power consumption that might exceed edge devices' power budget.

Recent works have proposed custom deep learning accelerators for resource-constrained edge devices. The accelerators can achieve much higher energy efficiency for low-precision and inference-only neural network applications. Chen et al. [4] proposed a digital deep learning accelerator that uses efficient data flows to maximize data reuses in DNNs and significantly reduces the off-chip memory access costs. The accelerator has an array of processing units with fixed-point computation logic and local scratch pads. However, data movement costs from the on-chip memory dominate the overall energy consumption.

Compute-in-memory (CIM) accelerator for deep learning is an emerging solution that provides high energy efficiency by embedding computation logic inside the memory array to reduce the data movement cost. The custom-designed CIM macros are used for both memory and computation. During computation, multiple rows are activated at the same time to perform matrix–vector multiplications. Many proposed CIM solutions are based on computing in the analog domain and converting to binary results by analog-to-digital converters (ADCs). Zhang et al. [5], Dong et al. [6], and Yue et al. [7] proposed accumulating the bit cell's current on the bitline. However, the computation accuracy is significantly limited by the transistor's local mismatch and nonlinearity. Although a large number (32∼64) of rows are activated in parallel, only 3∼4-bit accurate computation accuracy can be achieved per dot-product operation. ADCs also cause large area and energy overhead. Other researchers have proposed charge-based computing [8], [9], [10] using a metal-based capacitor embedded on top of the memory's bit cell at the back end. Multiplication results are stored as charges on the capacitor and then accumulated on the bitline. Due to the large size of the capacitor, a much better matching property is achieved. More than 2000 parallel rows and 8-bit computation accuracy are demonstrated. However, charging the bit cell's capacitor causes a significant amount of energy. Although capacitors have good matching properties, Wang et al. [11] found out that global process, voltage, and temperature variations can greatly degrade computation accuracy.

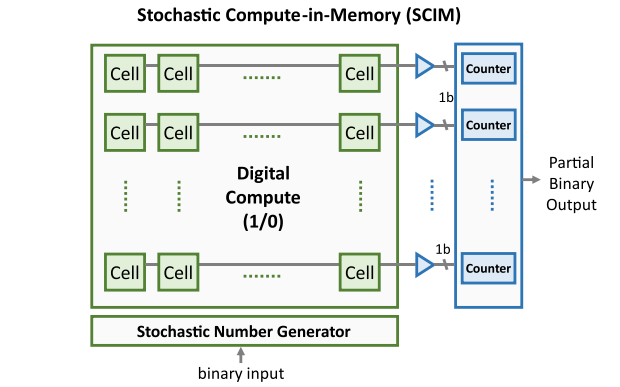
To alleviate analog computing's error and energy overhead problems, researchers have proposed robust digital CIM macro [11], [12]. The 1-bit multiplication is done in the bit cell and accumulated by an in-memory digital adder tree. It achieves accuracy close to fixed-point computation, but adder trees degrade the macro's area density. Stochastic computing (SC) is a digital probabilistic computing framework that uses random bit streams to represent numbers and simple bit-wise digital logic (AND and OR) gates to compute in the domain of probability. Previous work [13] has shown a deep learning accelerator using a standard cell's digital circuit to implement SC. It has demonstrated significantly

Received 31 March 2024; revised 16 October 2024 and 18 January 2025; accepted 15 March 2025. This work was supported in part by the Air Force Research Laboratory (AFRL) and in part by the Defense Advanced Research Projects Agency (DARPA) under Grant FA8650-18-27867. This article was approved by Associate Editor Jie Gu. *(Corresponding author: Jiyue Yang.)*

The authors are with the Department of Electrical and Computer Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: jyang669@ucla.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2025.3554554.

Digital Object Identifier 10.1109/JSSC.2025.3554554



	Stochastic CIM	Analog CIM
Number Conversion	Digital (SNG, Counter)	Analog (ADC, DAC)
Area Density	High (1b Digital logic)	Low (large ADC)
Accuracy	Match INT 8b, Limited by SC error	Match INT 8b Limited by noise, mismatch
Latency vs Precision	Exponential (match INT 6-8b)	Square (Bit Serial INT)

Fig. 1. High-level diagram of an SCIM macro, comparison between SCIM and analog CIM.

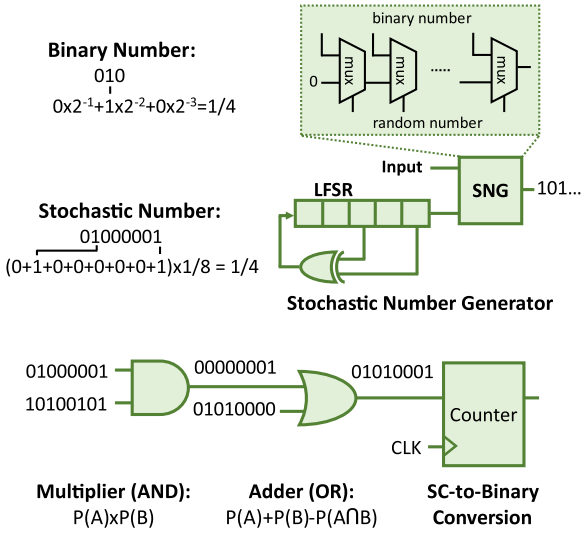


Fig. 2. Number representation and basic building blocks of SC.

improved computation density and energy efficiency compared to digital accelerators. In this work, we propose a stochastic CIM (SCIM) deep learning accelerator that combines the benefits of SC and CIM architecture. Due to the digital computation of SC, the CIM macro eliminates the large and power-hungry analog blocks: digital-to-analog converters (DACs)/ADCs. The OR-based SC accumulation is embedded as the memory's wired-OR structure on the compute line. The accumulation is a 1-bit operation and therefore does not cause area overhead. Efficient training for SC is performed and achieves comparable accuracy to INT8 on MNIST and CIFAR-10 classification. A high-level comparison between stochastic CIM and analog CIM is shown in Fig. 1. The DNN accelerator achieves energy efficiency up to 7.96 TOPS/W in 65 nm, which is 2–3× higher than analog CIM solutions. The main contributions of our works are: 1) a bit-parallel dataflow that maps bit-wise SC to SCIM macros in parallel; 2) storing pre-converted stochastic numbers in memory to achieve massive reuse of stochastic number generator (SNG); and 3) computation skipping technique to reduce SC stream length when the convolution layer is followed by average function.

A. Stochastic Computing

SC represents numbers and performs computation in the probability domain. The value of a number is represented by the fraction of ones in a random binary bit stream. For example, in a stochastic stream of 8 bits, $A = 01000001$, 2 bits are one, and 6 bits are zero. It represents the value of 2/8. Each bit represents an equal weight of $1/N$, where N is the stream length. The position of the one should be randomly located in the bit stream to avoid correlation between different bit streams during computation.

The conversion from binary to stochastic numbers is done by the SNG, as shown in Fig. 2. A randomness' source is required. True random number generator (TRNG) can generate uncorrelated bit streams, but the high energy and area make it impractical to implement. Pseudorandom number generators (PRNGs) such as linear feedback shift register (LFSR) are commonly used. To avoid correlations between different SNGs, LFSRs with different polynomial functions or seeds are used, which can be easily programmed on the chip. The conversion can be implemented by a comparator or a chain of multiplexers.

If the location of the ones in the SC bit stream is chosen randomly, the multiplication arithmetic of SC follows the rules of probability: $\text{Prob}(A \cap B) = \text{Prob}(A) \times \text{Prob}(B)$. This can be achieved by AND gate in hardware. Addition can be implemented by multiplexer and OR gate [14]. Multiplexer achieves the scaled addition between input streams. The control bit randomly selects between the inputs, which performs an average function. For addition between many inputs, mux-based addition leads to a small output amplitude due to the scaling factor. Although it performs accurate addition, the adder hardware costs a large area and energy consumption. The OR gate performs approximate addition following the union of two random variables $\text{Prob}(A \cup B) = \text{Prob}(A = 1) + \text{Prob}(B = 1) - \text{Prob}(A \cap B) \approx \text{Prob}(A) + \text{Prob}(B)$.

Although the OR-based accumulation does not implement the accurate addition function, previous works of SC proposed training methods to improve the computation errors in accumulation and stream generation when used in deep learning [15], [16]. SNGs using PRNGs create deterministic bit streams given the same inputs and stream. The training model can learn the patterns in the bit stream and allow moderate sharing of random numbers between SNGs without degradation to the accuracy. State-of-the-art classification accuracy has been demonstrated for MNIST/CIFAR10 datasets with DNNs that are comparable to 8-bit fixed-point implementations.

B. Motivation and Challenge of SC

Memory access consumes significantly more energy than conventional binary computation. Reducing the number of on-chip and off-chip memory access is critical to achieve higher energy efficiency. SC uses tiny bit-wise logic gates to achieve extremely high parallelism and reduce the on-chip memory access. The previous work [13] has built a highly

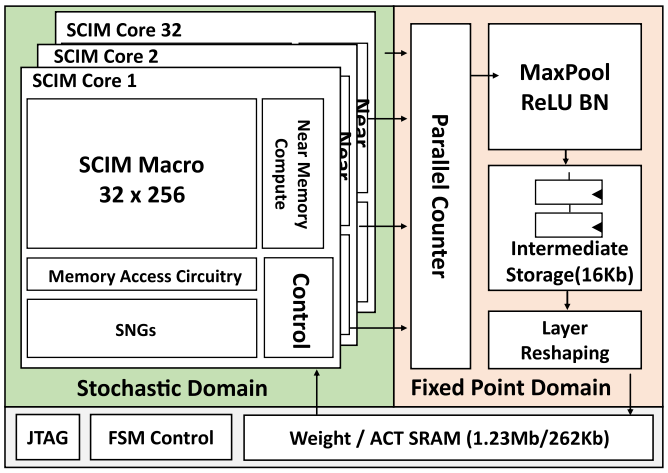


Fig. 3. Architecture of SCIM accelerator.

programmable SC deep learning accelerator using fully synthesizable standard cell digital gates. The MAC array achieves a density of 38.4 K/mm^2 (including SNG, buffers, and control logic), which is $>10\times$ higher than fixed-point logic.

The CIM architecture can achieve a large reuse factor and high density due to: 1) no memory access is required for weight since computation happens inside the memory and reloading of the CIM macro is necessary for a larger network, but the cost can be reduced by a large amount of reuse and 2) CIM breaks the bandwidth limits of the Von Neumann architecture by activating multiple rows or entire arrays for computation. The dense array can achieve very high parallelism. Combining the benefits of SC's lightweight digital computation with CIM's low data movement cost is an attractive solution. The SCIM accelerator embeds robust digital MAC circuits in the memory and does not suffer from analog noise. The accumulated output is a 1-b signal at the logic level and does not require large ADCs. Despite the benefits, a few challenges need to be overcome: 1) SNG's energy consumption is significantly larger than the SC's computation and a large SNG reuse factor is needed to reduce the average cost per operation; 2) the stream length of the stochastic number representation increases as a power of two with the number's precision and the long stream length degrades the energy efficiency and the throughput; and 3) SC's OR-based approximate accumulation is a nonlinear addition function. An efficient model of the accumulation function is needed during neural network training to achieve comparable accuracy as a fixed point.

II. SCIM Processor

Fig. 3 shows an overview of the convolutional neural network (CNN) processor based on the concept of SCIM. It has 32 SCIM macros that perform bit-parallel processing of convolution between activations and weights in the SC domain. The convolution results are converted to the fixed-point domain by the parallel counters. ReLU, max pooling, and batch normalization are performed at fixed point. The layer outputs are stored in the output SRAM until the next layer starts processing.

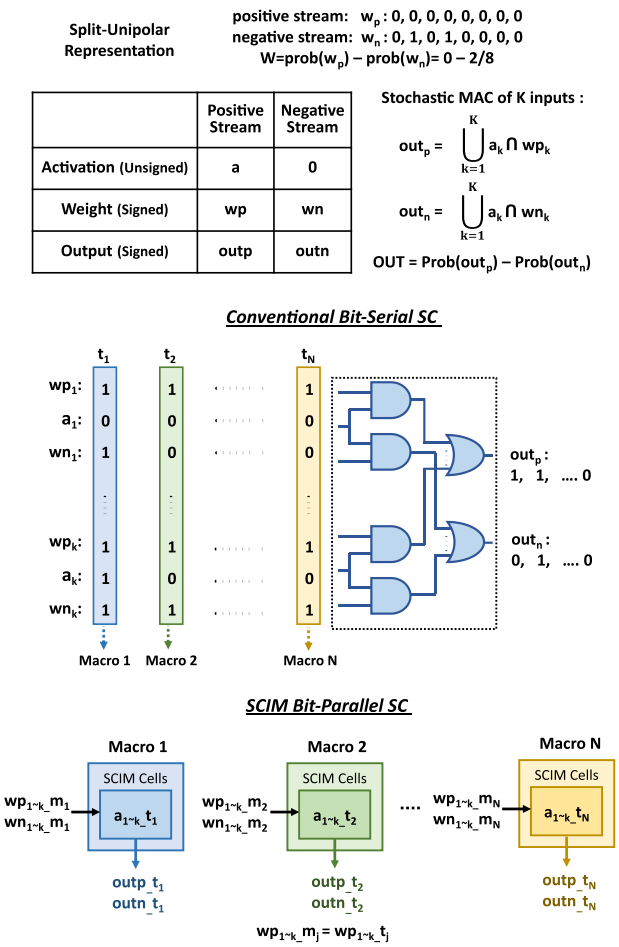


Fig. 4. Split-unipolar representation for inputs and weights. Bit-parallel processing and mapping to SCIM macros.

A. Bit-Parallel In-Memory Compute Data Flow

Conventional SC operates serially due to the sequential generation of stochastic bit streams. Embedding bit-serial SC computation in memory requires SNGs, which occupy a large area. To enable SC-in-memory, this work proposes to store entire bit streams in memory and computation will happen in parallel. A bit-parallel and in-memory SC data flow is designed to store input bit streams since activations are positive values and require less storage compared with weights. Unipolar stochastic representation directly maps the probability of ones to a range of [0, 1], therefore only representing positive numbers. Split-unipolar SC representation is a simple way to use the difference between two unipolar bit streams, positive and negative streams, to represent a signed number between -1 and 1. Fig. 4 shows an example. For a negative number, $W = -2/8$, and the magnitude 2/8 is encoded in the negative stream and the positive stream is zero. The difference in probability between positive and negative streams is $-2/8$. SCIM can also support negative inputs by storing split-unipolar streams in the macros and subtracting macros' outputs, but the proposed accelerator only implements data flows for positive inputs specific to vision-based applications.

The MAC arithmetic of the split-unipolar stochastic bit streams needs to account for the cross-product between the

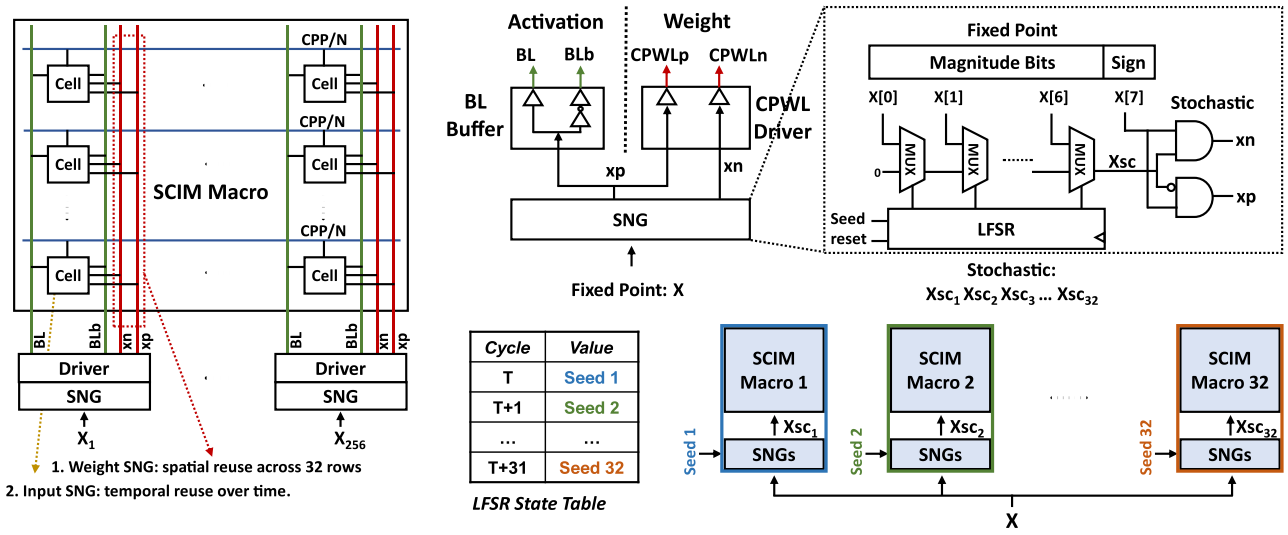


Fig. 5. SNG circuit. Dath path of SNG for input and weight. Seed schedule for parallel SNGs.

operands' positive and negative streams. Inputs to each neural network layer are typically positive numbers due to the nonlinear activation function such as ReLU, which clips negative values to zero. Inputs only need one unipolar bit stream to represent positive values. Weight parameters are signed numbers and require both positive and negative streams of split-unipolar representation. The cross product between input's positive stream: a and weight's split-unipolar streams: wp/wn is illustrated in Fig. 4. Assume that both input and weight are 1-D vectors with K elements. Each element of input a is multiplied with wp and wn by intersection AND logic: $a_k \cap wp_k$ and $a_k \cap wn_k$ and then accumulated by the union OR logic. The outputs of the two union operations produce positive and negative streams: out$p$ and out$n$, which are the split-unipolar representations of the dot product's result.

Fig. 4 (bottom) shows how bitwise processing of the conventional SC MAC operation is mapped to SCIM macros. Assume that the stream length is N. Conventional bit-serial SC arithmetic process 1 bit at a time through the AND and OR gates. To perform SC in memory, each SCIM macro stores the 1-bit representation of the input vector: macro-1 stores $a_1\_t_1 \sim a_k\_t_1$, where $a_1 \sim a_k$ denotes the k elements of vector a and t_1 denotes 1st bit of stochastic bit stream. Macro-2 stores $a_1\_t_2 \sim a_k\_t_2$ and macro-N stores $a_1\_t_N \sim a_k\_t_N$. Each 1-bit representation of the weight vector is applied to macros: $wp_1\_t_1 \sim wp_k\_t_1$ and $wn_1\_t_1 \sim wn_k\_t_1$ are applied to macro-1. $wp_1\_t_N \sim wp_k\_t_N$ and $wn_1\_t_N \sim wn_k\_t_N$ are applied to macro-N. The dot product between inputs and weights is performed in the memory macros together. The bit-parallel processing unrolls the bit-wise operation in space so that all the output bits are computed and available in one cycle.

B. Stochastic Number Generator

A bottleneck with conventional SC is the large energy cost of the conversion from binary to SNs. SNGs typically use PRNGs such as LFSR as the randomness source. It consumes 25× more energy than an SC MAC unit, which only performs 1-bit in-memory AND and OR operations. A large reuse factor of the SNG output is required to reduce the energy cost. Fig. 5 shows the diagram of the SNG circuit. Each register of the LFSR controls one multiplexer. The multiplexers are cascaded, and each control signal selects between a binary bit of the input and the output of the previous multiplexer (see Fig. 5). N multiplexers are required to convert an N-bit binary number to stochastic bits. It can be shown that the SNG can almost accurately convert binary numbers if two conditions are met: 1) maximal-length LFSR is used as the randomness source and 2) the order of the LFSR's characteristic polynomial matches the bit width of the input binary number [17]. An N-bit maximal-length LFSR will iterate over all the possible combinations of N-bit numbers except zero and therefore has a period of $2^N - 1$. Each N-bit combination is iterated exactly once. The most significant bit of the binary number is multiplexed to the SNG output exactly 2^{N-1} times. Other bits are multiplexed with binary weighted frequencies: 2^{N-2}, 2^{N-3}, ..., 2^0, and therefore, the frequency of one in the stochastic bit stream accurately represents binary input. Since the accelerator performs 8-bit computation with 1 bit for sign and 7 bits for magnitude. The LFSR size is chosen to be 7 bits. The demultiplexer at the output of the SNG selects between xp and xn based on the sign of the binary input.

1) SNG for Inputs: The SCIM macro stores the pre-generated stochastic bit stream of inputs. Fig. 5 shows how the per-column SNG supports both loading inputs and applying weights during computation. Each column has an SNG and driver for bit lines: BL/BLb and compute word lines: CPWLP/N. The SNG converts the binary number X to split-unipolar stochastic representation: xp/xn. Since the activations use unipolar representation, only xp needs to be stored in the memory. During input loading, xp is sent to the BL buffer, which drives BL/BLb and stores xp to the bit cell. Each macro only stores one stochastic bit of the activation. A parallel stochastic number generation scheme supports the

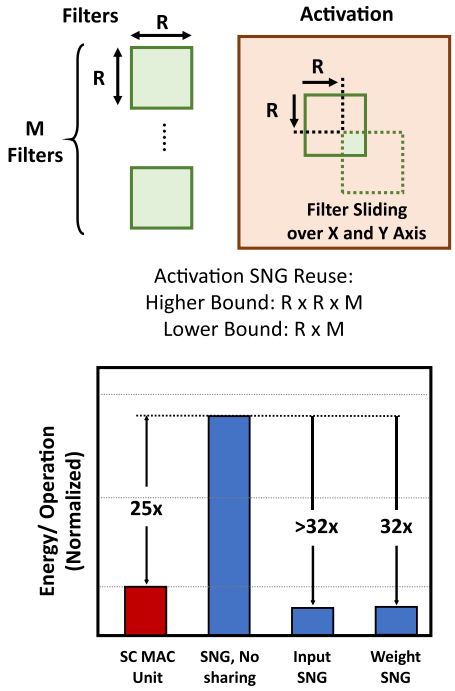


Fig. 6. Reuse of the stored activations (top). Energy cost of SNG with and without reuse (bottom).

bit-parallel SC processing shown in Fig. 5. The LFSR state table characterizes the changes in the shift registers at different cycles. To parallelize the SNG, each LFSR state is used to initialize the LFSR in different SCIM macros: seed 1 for the 1st macro and seed K for the Kth macro. The results of K SNGs in one conversion cycle are the same as a single SNG generating for K cycles.

The stored activation stochastic bits can be reused from filter's sliding windows and output channels, as shown in Fig. 6. Assume that the 2-D filter has the shape of $R \times R$ and M output channels, and the activation can be reused with the maximal factor of $R \times R \times M$. The SNG cost is lowered by the same factor since SNG only consumes energy when activations are loaded into the memory. The actual activation reuse factor depends on the size of each layer and whether the SCIM macro can support sliding windows across X- and Y-dimensions without reloading the activations. Our proposed accelerator computes Y-axis sliding windows in parallel due to the parallel operation of rows in the SCIM macro and X-axis sliding windows in serial. If the layer's X-dimension is too large, the SCIM macro might not support sliding windows on the X-axis without reloading the activations into the SCIM macro. The activation reuse over output channels can be easily supported with increased intermediate storage when the output channel size is large. Therefore, the lower bound of the activation SNG reuse factor is: RxM. In our experiment with CNNs supporting MNIST and CIFAR-10 classifications, the activation SNG reuse is >32.

2) SNG for Filters: The filters use the same SNGs as inputs but during the computation cycle. The binary filter coefficients are converted by SNGs in different SCIM macros in parallel to stochastic bits. The compute port word line (CPWLP/N) drivers buffer the split-unipolar bits *xp* and *xn*. All the rows in the SCIM macro operate in parallel, and therefore, the energy cost of filter SNG and CPWL driver is shared by the number of rows: 32. The comparison of the energy cost of the activation SNG, filter SNG, and SNG without any sharing is shown in Fig. 6. Although the SNG energy is 25× higher than a single SC MAC unit, the activation and filter SNG energy cost is reduced by more than 32× due to reuse.

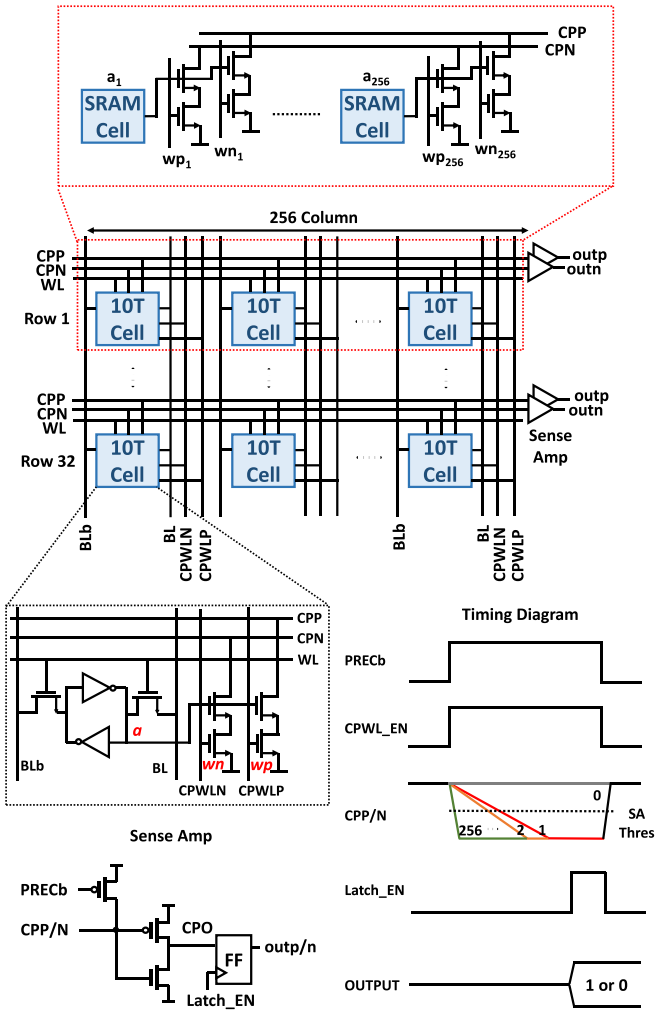


Fig. 7. SCIM bit cell array. Circuit implementation of 10T SCIM cell and sense amplifier. Timing diagram of a compute operation.

C. SCIM Array

The SCIM macro embeds simple digital SC computation logic inside the memory to achieve high energy efficiency for matrix multiplication operations: AND gates for multiplication and OR gates for approximate accumulation. Each 1-bit stochastic representation of the activation is stored in the 10-T bit cell before the computation. The storage cell uses the standard 6-T SRAM cell design. The bit cell structure is shown in Fig. 7 (bottom). Each pair of cascaded nMOS transistors performs an AND operation between the stored activation bit and a weight stochastic bit applied at the compute wordline (CPWLP/CPWLN). The two multiplier circuits in the bit cell perform multiplication between activation (a) and

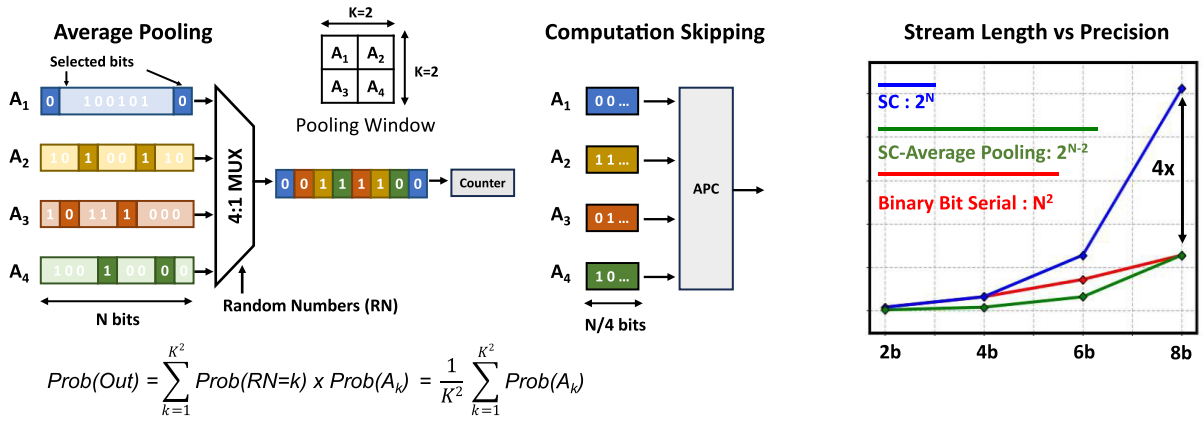


Fig. 8. Computation skipping of average pooling function. Stream length versus precision function.

weight (*wp*/*wn*), which correspond to one positive bit and one negative bit of a weight parameter. The shared compute port (CPP/CPN) across a row realizes a wired-OR operation between 256 cells. The CPWLP/N is routed in the vertical direction and shared by all the bit cells in the same column. A simplified diagram of SC MAC units of one row is shown in Fig. 7 (top). The computation follows the operation of the precharged pseudo-nMOS logic. The timing waveform of the MAC operation is shown in Fig. 7. The compute port is initially precharged to VDD and all the CPWLs are grounded. Once the computation starts, the compute port's precharging pMOS transistor is turned off and CPWL drivers are enabled. If the OR accumulation result is zero, no bit cell is conductive and only a small leakage current discharges the compute port. Simulation shows only 12-mV voltage drop due to 256 bit cells' leakage current for the worst PVT corner. If at least one bit cell is conducting, the OR accumulation result is one and the compute port will be discharged toward ground by active current. The timing constraint between pre-charging and sense amplifier's latch enable signal is decided by the propagation delay from the compute line to the output of the sense amplifier. The worst case propagation delay happens when only one bit cell is conductive and the rest of 255 bit cells are non-conductive. The data stored in the bit cells vary the total compute line's capacitance. When the stored bit (a) is 1, the top nMOS transistor is turned on and contributes extra gate capacitance compared to when the transistor is off. It is simulated by setting $a = 1$ and $wp/wn = 0$ for all 255 non-conductive bit cells and across PVT variation using parasitic-extracted netlist. The delay between pre-charging and latch enable signal is controlled by digital logic in a finite state machine that counts a programmable number of system clock. A simple inverter acts as a sense amplifier and buffer to provide tolerance to noise and coupling. Once the compute port evaluation is done, the 1-bit result is registered in a flip-flop. The macro is 32 rows tall and 256 columns wide. Each row performs two 256-long dot products, and one macro contains 16.4k MAC units.

D. Computation Skipping for Average Pooling

A computation skipping technique is developed to reduce stream length when the convolution layer is followed by an average pooling function. The pooling layer is a necessary component in neural networks to make the output features less sensitive to the location of the input and reduce layer dimension to save computation complexity. Pooling based on the maximal and average values are both highly effective: max pooling helps highlight the prominent pixel in the window, while average pooling can smooth out the images [18]. An averaging function can be implemented in SC using simple multiplexer logic. Consider an average pooling window of $K \times K$. It is realized by a K^2:1 mux that selects one of the K^2 inputs randomly (see Fig. 8). The probability of the multiplexer's output is the average input probabilities. For a four-input average pooling function, each stochastic input is selected for 1/4 of its sequence length. For each input stream, the unselected stochastic bits do not contribute to the output and the computation for those bits can be avoided to save energy and latency, which was originally proposed by [15]. Each input only needs to compute for $N/4$ bits for an original N-bit stream. The shorter stream length equivalently scales each input by 0.25 and the four inputs are added by accumulative parallel counters (APCs).

SC's energy consumption and latency increase exponentially versus computation precision due to its stochastic representation: N-bit binary number requires 2^N stochastic bits and 2^N evaluation of bitwise SC logic. In contrast, fixed-point digital logic's energy increases as N^2. An N-bit multiplier requires N^2 of full adders. Analog-based CIM commonly uses a bit-parallel bit-serial scheme to perform multi-bit multiplication [19], which also causes the energy to increase quadratically. Weight bits are stored as 1 bit per column and computed in parallel, but the input bit is serially applied to the CIM macro in N cycles. For an N-bit computation, CIM requires N^2 of in-memory MAC and ADC evaluation. For 4-bit or lower precisions, the number of MAC evaluations is comparable between SC and ADC-based bit-serial CIM ($4^2 = 2^4 = 16$). However, an 8-bit SC computation requires $2^8 = 256$ MAC evaluations, which is 4× larger than the ADC-based bit-serial CIM.

A comparison of the energy efficiency with other CIM types is shown in Fig. 9. The SCIM macro's energy cost is measured in post-layout simulation and includes SNGs for input stochastic number conversion, counters for output binary

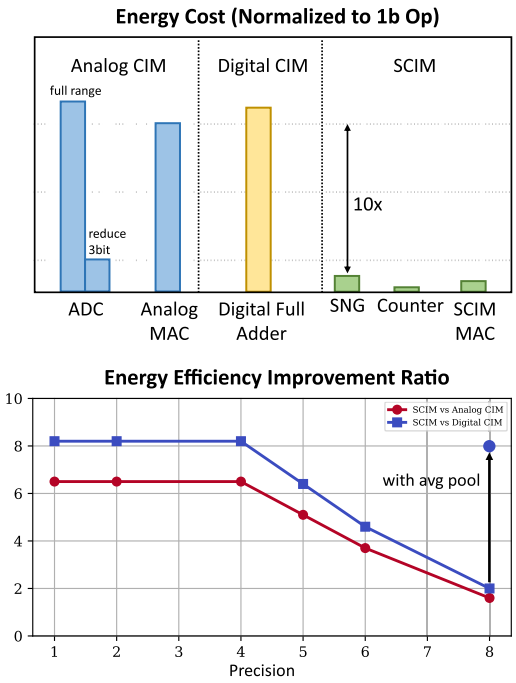


Fig. 9. Energy cost comparison of all components in CIM and SCIM macros normalized to 1-b Op. Energy efficiency improvement of SCIM over other CIMs versus precisions.

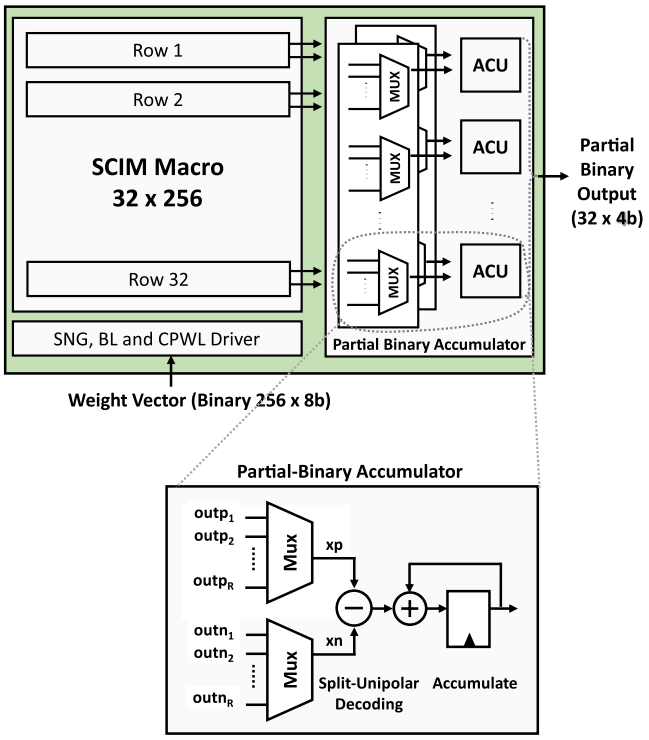


Fig. 10. Near-memory partial binary accumulator circuits.

conversion, and in-memory SCIM MAC. The charge-based CIM macro consumes energy from the input driver, charge-based CIM MAC, and column ADC. The energy consumption of each block is normalized to 1-bit operation and plotted in Fig. 9. The energy is derived from [19] demonstrated in 65-nm technology. The ADC accounts for 20% of the macro energy, but the ADC's precision is 3 bits less than the full dynamic range of the column dot product. Overall, SCIM achieves >6× higher energy efficiency than analog CIM for 1-bit Op. For higher precision, SC benefits diminish due to the longer stream length. For 8-bit precision, SC compute requires 256 1-bit Op, but CIM only requires 64 1-bit Op. The advantages of SCIM drop to 1.6× for 8-bit computation. The average pooling reduces the stream length by 4× and makes SCIM 6.4× more energy efficient than CIM. The energy cost of the digital CIM is mainly dominated by the full adder circuit, which requires switching 10∼20 extra transistors per stored bit. SCIM has 8× energy efficiency improvement over digital CIM at 1-b precision and 2× benefits at 8-b precision, as shown in Fig. 9.

III. Efficient Micro-Architecture Design for CNN

The proposed SCIM CNN accelerator supports end-to-end operation for convolutional neural network inferences and it is highly programmable to accommodate different layer topologies. In this section, we introduce different micro-architecture designs that support the efficient implementation of CNNs.

A. Near-Memory Partial Binary Accumulation

The near-memory accumulation circuit supports partial binary accumulation between rows (see Fig. 10). Partial binary accumulation is a technique to improve accuracy by breaking large SC dot products into several small ones and adding partial SC outputs using fixed-point adders. The in-memory dot product of two vectors in the SC domain uses AND logic as the multiplier, and then, the multiplication results are accumulated by the wired-OR structure of the SCIM macro's compute port. The partial binary accumulator block has 32 rows, matching the height of the SCIM macro, and a binary integrator circuit. The multiplexer can select every eight adjacent rows to the accumulator's input, which supports the row stationary dataflow to be discussed in Section III-B (see Fig. 10). One multiplexer selects among positive stochastic bits of the eight rows' output (out$p$), and the other multiplexer selects the negative stochastic bits (out$n$). The split-unipolar decoding circuits subtract 1-bit out$n$ from out$p$. The 2-bit subtraction result is accumulated by the fixed-point accumulation circuit.

B. IRS Dataflow for Conv Layer

The input activation's stochastic bits are stored in the SCIM macro due to the shorter stream length of the unsigned numbers compared to the signed weight parameters. An input and row stationary (IRS) dataflow is used to maximize the input reuse during the convolution. Instead of flattening the 3-D activation tensor into a 1-D vector, only the depth channel is fattened. The rows of the flattened activation can be reused during convolution sliding. Each activation row is stored in one row of the SCIM macro. A total of 32 activation rows are stored. The filter is flattened in the same manner into a 2-D map. Convolution operation prioritizes sliding across the Y-direction because the physical structure of the macro allows multiplication between rows to happen in parallel. Weight rows

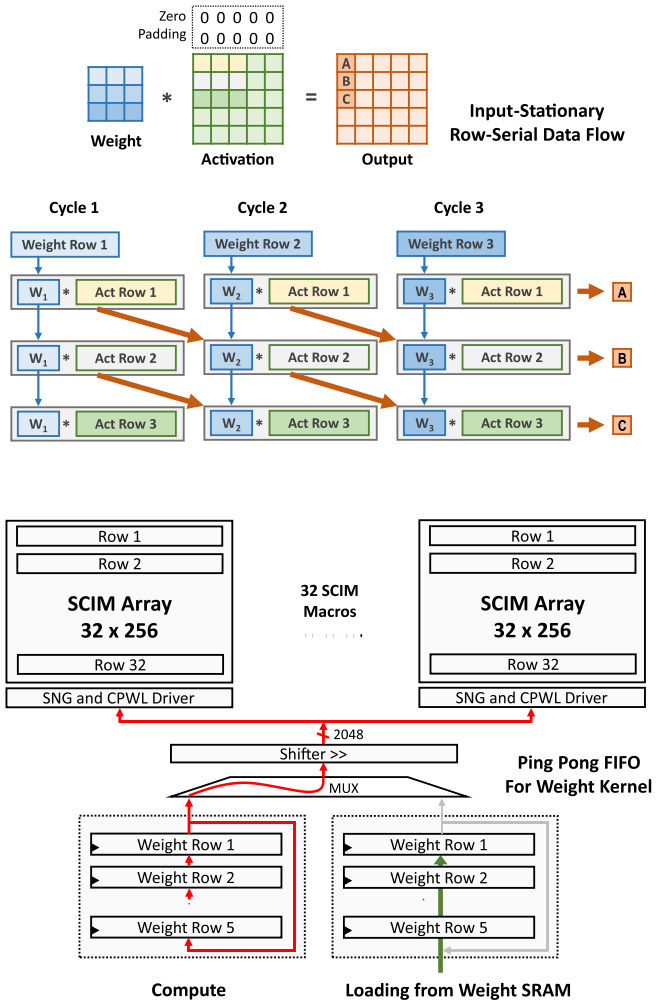


Fig. 11. Data flow of convolution. Ping-pong FIFO registers for weights.

are applied to the macro serially in different cycles, which is supported by a ping-pong structure of FIFO cyclic shift registers. Each register stores one row of the weight kernel. The ping-pong structure enables computation and loading new weight kernels simultaneously to hide the latency of SRAM access. One FIFO provides weight parameters to 32 SCIM macros and the other FIFO loads the next kernel from SRAM. Once the computation of one FIFO is finished, the roles of two FIFOs are switched and the multiplexer will select the new FIFO. Each SCIM row performs a dot product between an activation row and weight row. In the next cycle, the FIFO register shifts and the next weight row is applied to the macro. The row multiplexers shift the sense amplifier's outputs to the next row's partial binary accumulator such that the row outputs are accumulated diagonally, as shown in Fig. 11. The row-wise convolution is done until all the weight rows are shifted from the FIFO. The row multiplexers can select every seven rows' output, which supports 3×3, 5×5, and 7×7 convolution kernel size.

Convolution layer's dataflow is controlled by the on-chip finite state machine, which can be broken down into three recursive for loops: 1) convolution sliding across rows; 2) convolution sliding across columns; and 3) computing different output channels. Sliding across rows 1) is prioritized because of the row-parallel structure of the SCIM array. The order between column sliding 2) and output channel 3) depends on the layer and kernel dimensions.

1) If the SCIM Macro's Width Is Larger Than the Flattened Kernel Width: The macro's row can store activations of more than one convolution sliding window. The column sliding is scheduled first before loading a different weight kernel because sliding across the SCIM macro's columns avoids reading new weights from SRAM. Since the kernel width is smaller than the macro width, the weight kernel needs to be shifted to match the CPWL of a sliding window. This is achieved by a programmable barrel shifter at the output of ping-pong FIFO registers, as shown in Fig. 11. Other CPWLs not inside the sliding window are masked by zero. The shifting step for each sliding window is calculated by a controller based on the current sliding location and stride. Once a kernel has finished convolution over the stored input activation, the next kernel is applied by switching the ping-pong FIFO. If the activation map's size is larger than the SCIM array width, new activations need to load into the SCIM array to complete convolution over the entire activation map. Since the kernel width is smaller than the SCIM array width, the SCIM array is underutilized and the energy efficiency will degrade. This usually happens in the first few layers of neural networks when the number of output channels is small.

2) If the SCIM Array Width Is Equal to or Smaller Than the Flattened Kernel Width: The current activations stored in the SCIM array can only support one column-wise sliding window. Output channels 3) loop will be scheduled first because the energy cost of reading new weight kernels is smaller than loading inputs to the SCIM macros. To compute a different column-wise sliding window would require accessing input SRAM and loading 32 rows of input activation to the SCIM array. The kernel size (3×3 or 5×5) is much smaller than the height of activation map and therefore requires less SRAM accessing and communication energy. The output channels are scheduled to perform convolution over the current activations stored in the SCIM macros, and then, the SCIM array will be reloaded with new activations. Since the entire array is utilized for computation, the energy efficiency is the highest in this case. This usually happens for most of the hidden layers in DNNs where the flattened activation map's dimension is large.

C. Stochastic-to-Binary Conversion and Fixed-Point Domain Processing

The convolution outputs from the SCIM macros are converted to binary numbers to perform nonlinearity and scaling functions in the fixed-point domain. The stochastic outputs are converted to binary numbers by an array of parallel counters (see Fig. 12). Each SCIM macro produces 32 4-bit partial binary outputs. The outputs corresponding to the same convolution result in all 32 SCIM macros are passed to a parallel counter, which adds 32 binary numbers and produces a 9-b result. The parallel counter uses a binary tree architecture with two-stage pipelines to reduce the impact on the overall latency. The first pipeline register is placed after the third adder

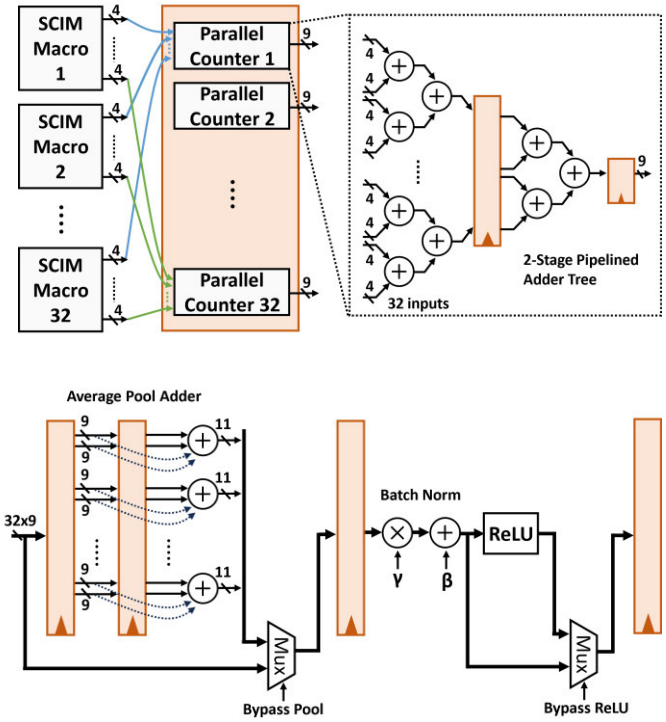


Fig. 12. Parallel counter for stochastic to binary conversion. Fixed-point domain processing.

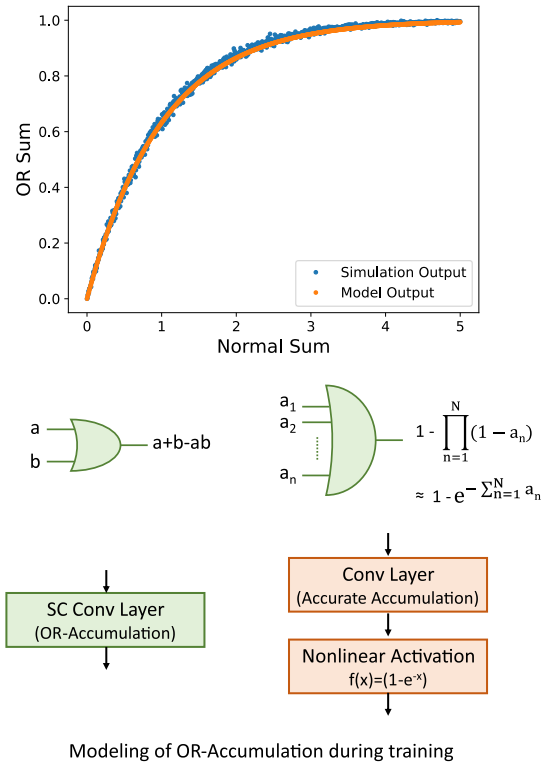


Fig. 13. OR accumulation's output versus accurate sum. Modeling of OR-accumulation during training.

stage, which makes the delay of the two pipeline sections about the same.

After the convolution outputs from SCIM macros are converted to binary numbers, the output processing functions such as pooling, batch normalization, and ReLU are done in the fixed-point domain (see Fig. 12). They are implemented in pipelined stages for maximum throughput. The first step of the fixed-point processing is average pooling. The computation skipping implemented in the SC domain reduces the stream length and therefore implicitly scales down the output value by the number of elements in the pooling window. To complete the average pooling function, the scaled outputs within pooling window need to be added. This design supports a pooling window of 2×2 and the inputs to the average pooling's adders are available at two different clock cycles. In the first clock cycle, the outputs from the parallel counters contain one column of the output feature map and they are stored in the shift registers next to the average pooling adders. The adjacent output feature map's column is computed after the kernel slides to the next column. Once the results of two adjacent output columns are available, the adder sums four inputs from the shift register. The average pooling can also be bypassed to support the layer without the pooling function. The results are scaled with batch normalization's scaling parameter and then subtracted with bias. The batch normalization's outputs are then processed by the ReLU function, which clips the negative values to zero. The outputs are loaded in the activation SRAM on chip until the start of the next layer.

D. OR-Accumulation and Neural Network Training

There are two main sources of random errors: 1) OR-based nonlinear accumulation function and 2) random bit stream representation generated from SNG. The OR-based accumulation is a nonlinear addition function. Take a two-input OR operation as an example. If the two input stochastic bit streams have the probability of a and b, the output probability is $a + b - ab$. When multiple inputs are accumulated by the OR-based accumulation, the output can be derived as shown in Fig. 13 and approximately equals $1\text{-}e^{-s}$, where s is the accurate sum of all the inputs. The plot of OR-accumulator's outputs versus accurate sum of the inputs is shown in Fig. 13 (top). The curve is close to a linear function in the range between 0 and 1. When the inputs become large, the accumulation saturates at 1 and becomes nonlinear. The backpropagation of the neural network training needs to account for the accumulation errors. We model the OR accumulation as a nonlinear activation function after an accurate accumulation, as proposed in [15]. The error in random bit stream representation is modeled by using the exact LFSR sequence and SNG during training. Multiplication is also modeled using the AND operation between stochastic bit streams of input and weight. The multiplication outputs are converted to floating point and accurately added. The details of modeling the errors in stream generation are discussed in [13].

IV. Measurement Results

The accelerator chip is fabricated in 65-nm CMOS technology, and a macro test chip is fabricated in 14 nm to better characterize the performance of the macro individually. The accelerator chip in 65 nm has an area of 9.36 mm^2. A die photograph is shown in Fig. 14. The activation and

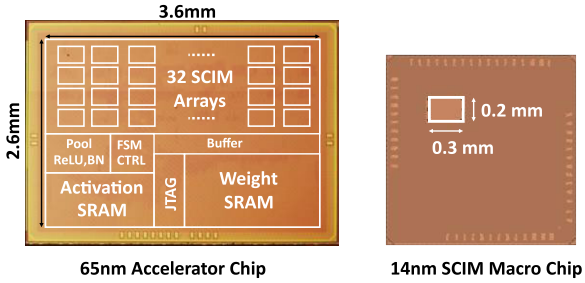


Fig. 14. 65- and 14-nm chip micrograph.

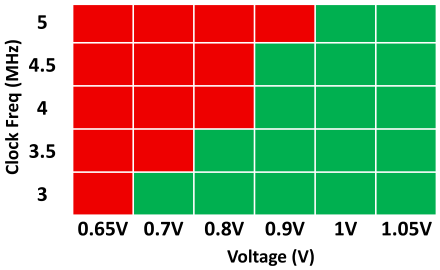


Fig. 15. 65-nm chip clock frequency versus voltage.

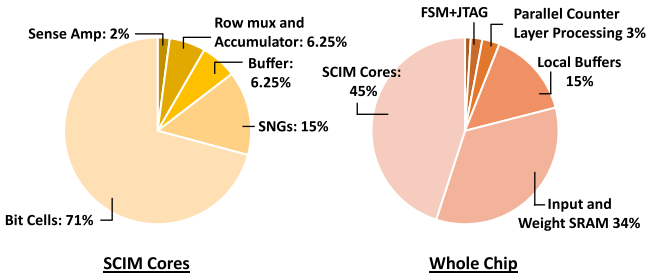


Fig. 16. Area breakdown of SCIM cores and whole chip.

weight SRAM are located at the bottom of the chip. The 32 SCIM cores are placed in a 4 × 8 array and contain 260-kB in-memory storage and 520 kB of MAC units in total, which achieves 130-kB/mm^2 MAC density. The nominal voltage of the 65-nm process is 1 V. The macro's read/write and dot product functions are verified at different supply voltages and operating frequencies, as shown in Fig. 15. The maximal clock frequency at 1 V is 5 MHz. It is limited by a problem of on-chip power delivery structure and could achieve a much higher frequency and throughput given a more optimized design. The macro is fully functional down to 0.7-V supply with a clock speed of 3.2 MHz. The area breakdown of the SCIM core and whole chip is shown in Fig. 16. The SCIM core area is dominated by the bit cells. The 1-bit sense amplifiers only occupy 2% of the area. SNG and row accumulator account for 15% and 6.25% of the SCIM core area, respectively. Input and weight SRAM accounts for a similar area as SCIM cores at the top level. The rest of the area is split among local register buffers, parallel counters, layer processing circuits, FSM, and JTAG interface.

CNNs for MNIST and CIFAR-10 datasets are demonstrated on the 65-nm accelerator chip, and the performance is summarized in Table I. We designed and trained CNN networks that can keep all parameters on the chip during computation. The measurement of energy consumption includes all components on the chip and there is no off-chip data movement during computation. For the MNIST dataset, the LeNet-5 topology is used. There are two CONV layers, each followed by average pooling to take advantage of SC's computation skipping and three fully connected layers. The classification results of the MNIST dataset achieve an accuracy of 99.1% over 1000 test images, which is close to the training accuracy. Activation and weight both have 8-bit precision. Due to the small size of filters used in LeNet-5, the peak macro utilization is only 5.1% and it causes significant degradation of energy efficiency. The peak system energy efficiency for MNIST classification is 0.35 TOPS/W.

For the CIFAR-10 dataset, we designed and trained a four-layer network: TinyConv-4. There are three convolution layers with 5 × 5 filters followed by 2 × 2 average pooling, batch normalization, and ReLU function. The last layer is a fully connected layer generating the classification results. TinyConv-4 is trained in both floating point and SC. The classification inference accuracy using INT8 is 78%. The training accuracy using SC achieves 75%. The test accuracy measured on the SCIM accelerator is 73.5% in 1000 images. Due to the limited on-chip memory, the accuracy is constrained by the network size and can be improved for a larger network. TinyConv-4 is larger than LeNet-5 and the utilization problem is slightly relieved, as summarized in Table I. The largest macro utilization is 31.3% at the second layer and the peak system energy efficiency is 2.2 TOP/S/W. The second layer's filter size is 5 × 5 × 32. After flattening to 2-D, the size becomes 5 × 160. The width of the filter, 160, is smaller than the width of the SCIM macro, 256.

A convolution layer with full macro utilization is tested to measure the peak energy efficiency of the accelerator. Activation and filter with a flattened width of 256 and no sparsity are applied in activation and weight. The energy efficiency at 0.8-V supply is 5.75 TOP/S/W. The energy efficiency at different supply voltages is tested and shown in Fig. 17. The peak energy efficiency at 0.7-V supply is 7.96 TOP/S/W. A breakdown of the accelerator's energy consumption is shown in Fig. 18. The most dominant components are SCIM cores (49%), clock communication (26%), controller, and local buffers (15%). The energy of sending 32 SCIM cores' output to parallel counters and loading weights from SRAM to 32 SCIM cores account for 11%. Parallel counter and fixed-point processing account for 2%.

Another test chip is fabricated in 14 nm to characterize the performance of the SCIM macros alone. The summary of the chip in comparison with the 65-nm accelerator chip is shown in Table II. It has 16 SCIM macros of 32 × 32 array. The bit cell and sense amplifier design are the same as the 65-nm chip. Each row performs a 32-long dot product and generates two stochastic output bits (out$p$/out$n$). The

TABLE I
NEURAL NETWORK DEMONSTRATION

Dataset	MNIST				CIFAR-10				Matrix Vector Multiply	
VDD/CLK	0.8V/4MHz									
Performance Details	LeNet-5				TinyConv-4				Layer	E Efficiency
	Input	Layer	Utilization	E Efficiency	Input	Layer	Utilization	E Efficiency		
	28x28x1	CONV (5x5)x6 Avg Pool 2x2	1.7%	0.15 TOPS/W	32x32x3	CONV (5x5)x32 Avg Pool 2x2	5.8%	0.4 TOPS/W	Matrix: 32x256 Vector: 1x256	5.75 TOP/S/W
	14x14x6	CONV (5x5)x16 Avg Pool 2x2	5.1%	0.35 TOPS/W	16x16x32	CONV (5x5)x32 Avg Pool 2x2	31.3%	2.2 TOPS/W		
	5x5x16	FC 25x120	4.8%	0.32 TOPS/W	8x8x32	CONV (5x5)x64 Avg Pool 2x2	15.6%	1.1 TOPS/W		
	120x1	FC 120x64	1.5%	0.13 TOPS/W	1028x1	FC 256x10	3.1%	0.2 TOPS/W		
	84x1	FC 64x10	1.1%	0.1 TOPS/W						
Bit Precision	8bit Weight, 8bit Act, 11bit Out									
INT Training Accuracy	99%				78%				Not Applied	
SC Training Accuracy	99.2%				75%					
Measured Accuracy	99.1%				73.5% (Top-1), 98.4%(Top-5)					
Energy/Classification	18.3μJ				35.6μJ					
Latency	5.85msec/frame				11.5msec/frame				1usec	

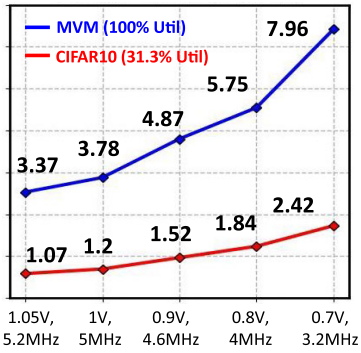


Fig. 17. Energy efficiency versus voltage.

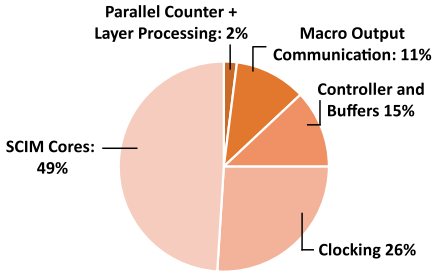


Fig. 18. Energy breakdown of the accelerator.

16 SCIM macros generate 32 stochastic bits for a given MAC operation, which is equivalent to 5-b computation precision. The macro's outputs are directly accumulated by the parallel counter without row mux and accumulator at a frequency of 130 MHz. The measured energy efficiency is 258 TOP/S/W. The energy efficiency scaled to 8-b operation is 35 TOP/S/W. If average pooling is used in combination with the convolution layer, the 8-b operation has an energy efficiency of 140 TOP/S/W.

A comparison table with other works is shown in Table III. The 65-nm chip packs 520 kB of in-memory MAC unit on chip, which is way larger than most of the other works

TABLE II
SUMMARY OF TWO PROTOTYPE CHIPS

	14nm Macro Test Chip	65nm Accelerator Chip
SCIM Macro Size	32 x 32	32 x 256
Number of Macro	16	32
Number of 1b SC MAC	32.7K	542.3K
MAC Density	850 Kb/mm^2	130 Kb/mm^2
Supply Voltage	0.65 V	0.7 V
Maximal Frequency	130MHz	5MHz
Operations	SCIM Matrix Multiplication	SCIM Matrix Multiplication, Row mux and accumulator Average Pooling (2x2), ReLU, Batch Norm, CNN FSM Controller
SC Stream Length	32	64 or 256 (with avg pool)
Equivalent Precision	5b	6b or 8b (with avg pool)
System Energy Efficiency	Not Applied	7.96 TOP/S/W
Macro Energy Efficiency	258 TOP/S/W	20 TOP/S/W

except [7]. With a smaller number of on-chip MAC units, more data movement is needed from on-chip buffer or off-chip DRAM, which might not be accounted for in the energy measurement. Our work builds a complete CNN inference process on the chip and keeps all the parameters on chip to account for all energy costs. The MAC density is also superior to other analog and digital CIM types. The 65-nm chip has a density of 130 kB/mm^2 and 14-nm chip has a density of 860 kB/mm^2. Digital CIM removes ADCs, but each stored bit requires ~20 extra transistors for multiplier and digital adder circuits. SCIM cell only adds four extra transistors for SC multiplier and uses the wired-OR structure for accumulation. We report energy efficiency for 8-b MAC operation and compare both system and macro efficiency. For SC, 8-b computation requires processing 256-long stochastic bit stream. Energy efficiency with and without average pooling is reported. The 65-nm accelerator has a peak system energy efficiency of 7.96 TOPS/W and a macro energy efficiency of 20 TOPS/W with average pooling. The 14-nm macro chip achieves 140 TOPS/W for 8-b compute with average pooling and 35 TOPS/W without average pooling.

TABLE III
COMPARISON WITH OTHER WORKS

	Digital	Analog CIM			Digital CIM			SC	SCIM	
	Chen, ISSCC 16	Dong, ISSCC 20	Jia, ISSCC 21	Yue, ISSCC 21	Yue, ISSCC 23	Chih , ISSCC 21	Lee, VLSI 22	Romaszkan SSCL 22	This Work	
Technology	65nm	7nm	16nm	65nm	28nm	22nm	12nm	14nm	65nm	14nm
Size	16mm^2	Not reported	25 mm^2	12 mm^2	3.75 mm^2	Not Reported	Not Reported	0.5mm^2	**9.4 mm^2**	**0.06 mm^2**
Macro Area	Not Applied	0.0032 mm^2	15 mm^2	1.7 mm^2	0.27 mm^2	0.2 mm^2	0.0323 mm^2	Not Applied	**4.2 mm^2**	**0.038 mm^2**
Voltage	0.8-1.2V	1V	0.8V	0.65V (Digital) 1V (CIM)	0.4-0.7V	0.7-0.9V	0.72V	0.6-0.9V	**0.7-1.05V (CIM), 0.8V(SRAM)**	**0.65 – 1V**
On-Chip CIM Size	Non CIM (168 PE)	4Kb	4.5Mb	64Kb	16.4Kb	64Kb	8Kb	Non CIM (19.2K MAC)	**520Kb**	**32.7Kb**
CIM MAC Density	Non CIM	1.25Mb/mm^2	300 Kb/mm^2	37.6 Kb/mm^2	60.7Kb/mm^2	320Kb/mm^2	248Kb/mm^2	38.4Kb/mm^2	**130 Kb/mm^2**	**860 Kb/mm^2**
Precision	INT 16	INT 4	INT 8	Wei: 4b, In: 8b	INT 8	INT 8	Wei: 4,8,12,16 In: 1~8	256b SC (INT8)	**256b SC (INT 8)**	**32b SC (INT 5)**
System Energy Efficiency (8b Op)	0.332 TOPS/W	Not reported	Not reported	2.3 TOPS/W	12.8 TOPS/W	Not Applied	Not Applied	4.4 TOPS/W	**7.96 TOPS/W**	**Not Applied**
Macro Energy Efficiency (8b Op)	Not applied	88 TOPS/W	30 TOPS/W	7.32 TOPS/W	68.7 TOPS/W	24.4 TOPS/W	24.7 TOPS/W	Not applied	**5~20 TOPS/W**	**35~140 TOPS/W**
Peak Throughput (Scale to 8b)	0.480 TOPS	0.093 TOPS	3 TOPS	0.025-0.79 TOPS	0.41 TOPS	0.03 TOPS	0.336 TOPS	0.15 TOPS	**0.06 TOPS**	**0.063-0.25 TOPS**
Throughput Density	0.3 TOPS/mm^2	29 TOPS/mm^2	0.2 TOPS/mm^2	0.014-0.46 TOPS/mm^2	1.52 TOPS/mm^2	0.15 TOPS/mm^2	10.4 TOPS/mm^2	0.3 TOPS/mm^2	**0.014 TOPS/mm^2**	**1.66 – 6.6 TOPS/mm^2**

V. CONCLUSION

In this work, we have demonstrated an ADC-less SCIM accelerator for convolutional neural networks. The accelerator has 32 SCIM macros storing the pre-converted stochastic numbers of activation and computing in a bit-parallel way. Since the OR-based SC accumulation is a 1-bit logic, the in-memory computation does not require an ADC. We have proposed a computation skipping technique to reduce the SC stream length by 4× when the average pooling layer is used. The accelerator has an efficient architecture supporting full neural network application on chip: convolution is performed in SCIM macros, and average pooling, ReLU, and batchnorm are processed after stochastic numbers are converted to binary. Classification of MNIST and CIFAR-10 datasets is demonstrated with all parameters remaining on the chip. The accelerator has a peak energy efficiency of 7.96 TOP/S/W for the system and 20 TOP/S/W for the macro. A test chip is fabricated in 14 nm with only SCIM macro and shows 35 TOP/S/W in 8-bit operation and 140 TOP/S/W with computation skipping in the average pooling layer.

REFERENCES

[1] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[2] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[4] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[5] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[6] Q. Dong et al., "15.3 A 351TOPS/W and 372.4GOPS compute-in-memory SRAM macro in 7 nm FinFET CMOS for machine-learning applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2020, pp. 242–244.

[7] 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) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2021, pp. 238–240.

[8] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[9] H. Jia et al., "15.1 A programmable neural-network inference accelerator based on scalable in-memory computing," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2021, pp. 236–238.

[10] 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.

[11] D. Wang, C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy, and M. Seok, "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*, San Francisco, CA, USA, Feb. 2022, pp. 266–268.

[12] J. Yue et al., "A 28 nm 16.9-300TOPS/W computing-in-memory processor supporting floating-point NN inference/training with intensive-CIM sparse-digital architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2023, pp. 1–3.

[13] W. Romaszkan, T. Li, R. Garg, J. Yang, S. Pamarti, and P. Gupta, "A 4.4–75-TOPS/W 14-nm programmable, performance- and precision-tunable all-digital stochastic computing neural network inference accelerator," *IEEE Solid-State Circuits Lett.*, vol. 5, pp. 206–209, 2022.

[14] Z. Li et al., "HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1543–1556, Aug. 2019.

[15] W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "ACOUSTIC: Accelerating convolutional neural networks through or-unipolar skipped stochastic computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Mar. 2020, pp. 768–773.

[16] T. Li, W. Romaszkan, S. Pamarti, and P. Gupta, "GEO: Generation and execution optimized stochastic computing accelerator for neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Feb. 2021, pp. 689–694.

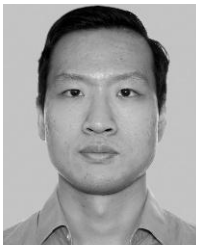
[17] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-36, no. 4, pp. 603–606, Apr. 1988.

[18] F. Bieder, R. Sandkühler, and P. C. Cattin, "Comparison of methods generalizing Max- and average-pooling," 2021, *arXiv:2103.01746*.

[19] 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.

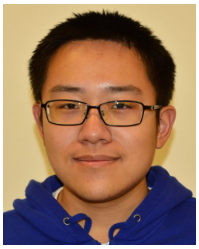
[20] Y.-D. Chih et al., "16.4 an 89TOPS/W and 16.3TOPS/mm2 all-digital SRAM-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *Proc. IEEE Int. Solid- State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, Feb. 2021, pp. 252–254.

[21] C.-F. Lee et al., "A 12 nm 121-TOPS/W 41.6-TOPS/mm2 all digital full precision SRAM-based compute-in-memory with configurable bit-width for AI edge applications," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, Honolulu, HI, USA, Jun. 2022, pp. 24–25.



Jiyue Yang (Member, IEEE) received the B.S. degree in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2016, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2018 and 2023, respectively.

He is currently a Hardware Research and Development Engineer with Broadcom Inc., Irvine, CA, USA. His research interests include hardware accelerators for efficient computing applications and circuit designs enabling emerging memory technologies.



Tianmu Li received the B.E. degree in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2017, where he is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department.

His research interests include efficient machine learning using approximate computing methods.

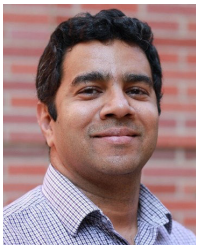
Wojciech Romaszkan received the B.S. and M.S. degrees in electronics and telecommunication from the AGH University of Science and Technology, Kraków, Poland, in 2012 and 2013, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of California at Los Angeles, Los Angeles, CA, USA, in 2023.

He is currently with Amazon Web Services, Seattle, WA, USA. His research interests include computer architecture, hardware design, and machine learning acceleration.

Puneet Gupta (Fellow, IEEE) received the Ph.D. degree from the University of California at San Diego, San Diego, CA, USA, in 2007.

He co-founded Blaze DFM Inc., Sunnyvale, CA, USA, in 2004. He is currently a Faculty Member with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA, USA. He currently leads the IMPACT+ Center. His research interests include design–manufacturing interface for lowered costs, increased yield, and improved predictability of integrated circuits and systems.

Dr. Gupta was a recipient of the National Science Foundation CAREER Award and the ACM/SIGDA Outstanding New Faculty Award.



Sudhakar Pamarti (Senior Member, IEEE) received the Bachelor of Technology degree from Indian Institute of Technology, Kharagpur, India, in 1995, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, San Diego, CA, USA, in 1999 and 2003, respectively.

He is a Professor of Electrical and Computer Engineering at the University of California, Los Angeles. His current research interests are in analog, mixed-signal, and RF integrated circuit design, specifically in developing signal processing techniques to overcome circuit impairments.

Dr. Pamarti is a recipient of the National Science Foundation's CAREER award. He was an IEEE Solid State Circuits Society Distinguished Lecturer and has served on the technical program committees of IEEE Custom Integrated Circuits Conference (CICC), IEEE International Solid State Circuits Conference (ISSCC), Design Automation Conference (DAC), and has been a guest or a regular Associate Editor for both Parts I and II of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and the IEEE JOURNAL OF SOLID STATE CIRCUITS.