CONV-SRAM: An Energy-Efficient SRAM With In-Memory Dot-Product Computation for Low-Power Convolutional Neural Networks

Citation: Biswas, Avishek and Anantha P. Chandrakasan. "CONV-SRAM: An Energy-Efficient SRAM With In-Memory Dot-Product Computation for Low-Power Convolutional Neural Networks." IEEE Journal of Solid-State Circuits 54, 1 (January 2019): 217 - 230 © 2018 IEEE

As Published: http://dx.doi.org/10.1109/jssc.2018.2880918

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: https://hdl.handle.net/1721.1/122468

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Conv-SRAM: An Energy-Efficient SRAM with In-Memory Dot-product Computation for Low-Power Convolutional Neural Networks

Avishek Biswas, *Member, IEEE,* and Anantha P. Chandrakasan, *Fellow, IEEE*

***Abstract*—This work presents an energy-efficient SRAM with embedded dot-product computation capability, for binary-weight convolutional neural networks. A 10T bit-cell based SRAM array is used to store the 1-b filter weights. The array implements dot-product as a weighted average of the bit-line voltages, which are proportional to the digital input values. Local integrating ADCs compute the digital convolution outputs, corresponding to each filter. We have successfully demonstrated functionality ($> 98\%$ accuracy) with the 10,000 test images in the MNIST hand-written digit recognition dataset, using 6-b inputs/outputs. Compared to conventional full-digital implementations using small bit-widths, we achieve similar or better energy-efficiency, by reducing data transfer, due to the highly parallel in-memory analog computations.**

***Index Terms*—in-memory computation, dot-product, energy-efficient SRAM, machine learning, edge-computing, analog computing, convolutional neural networks, binary weights**

I. Introduction

ARTIFICIAL intelligence (AI) and machine learning (ML) are changing the way we interact with the world around us. Speech recognition [1] allows us to interact with "smart" devices more naturally using our voices. Facial recognition [2] enables using our faces to get access to devices in a more intuitive manner, instead of traditional passcodes. As we think about extending machine intelligence to more and more devices around us, in the "Internet-of-Things" (IoT), "edge-computing" i.e. computing on these edge devices vs. the "cloud" becomes increasingly important. There are a multitude of reasons for this. Firstly, "edge-computing" enables the devices to make fast decisions locally, without having to wait for the "cloud". Secondly, it can significantly reduce the communication traffic to the "cloud", by only sending the critical/relevant information and filtering out the rest of the massive amount of data the edge-devices may collect. Furthermore, "edge-computing" helps in improving the security of the data by keeping it local (within the devices), rather than having to send sensitive information to the "cloud". While "edge-computing" promises significant benefits for IoT devices, it also has certain requirements. The circuits to run the compute algorithms must be very energy-efficient to extend the battery-life of these IoT devices, most of which have a very limited energy budget and would be "always-ON". Additionally, in many applications, the local decision-making has to be done in real-time (e.g. self-driving cars), to make them practical.

Convolutional neural networks (CNN) provide state-of-the-art results in a wide variety of AI/ ML applications, ranging from image classification [3] to speech recognition [1]. However, they are highly computation-intensive and require huge amounts of storage. Hence, they consume a lot of energy when implemented in hardware and are not suitable for energy-constrained applications e.g. "edge-computing".

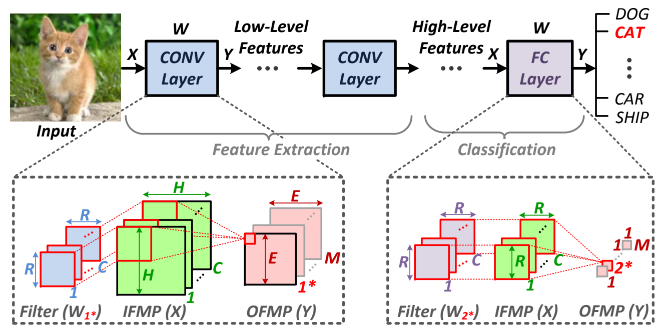

Fig. 1. Basics of a typical convolutional neural network (CNN) for a classification problem, showing the structure for the CONV and FC layers [4], [5].

CNNs typically consist of a cascade of convolutional (CONV) and fully-connected (FC) layers (Fig. 1) [4], [5], with some non-linear layers in between (not shown in the figure). The CONV layers extract different features of the input and the FC layers combine these features to finally assign the input to one of the many pre-determined output classes. For each of the CONV/FC layers, there is a set of 3-D filters (W_k), which are applied on the 3-D input feature-map (IFMP) to that layer and generate its 3-D output feature-map (OFMP). Each 3-D filter/input consists of mutliple 2-D arrays, each of which corresponds to a different "channel" (1 to C). When a 3-D filter (W_k) is applied on the input (X), an element-wise multiplication is performed, followed by an addition of the partial products to compute the convolution output (Y_k). For CONV layers the 3-D filter is applied on the shifted input to compute the next element in the 2-D OFMP. Each individual filter corresponds to a different channel in the 3-D OFMP. Therefore, the fundamental operation for both the CONV and FC layers can be simplified to a dot-product or a multiply-and-accumulate (MAC) operation, as shown in equation (1).

A. Biswas is with Kilby Labs, Texas Instruments Incorporated, Dallas, TX 75243, USA (email: avishek.biswas@alum.mit.edu).

A. P. Chandrakasan is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

$$\begin{gathered} Y_{x,y,k} = \sum_{c=1}^{C}\sum_{j=1}^{R}\sum_{i=1}^{R} W_{i,j,c,k} \times X_{x+i,y+j,c} \\ 1 \leq (x+i),(y+j) \leq H, \\ 1 \leq x,y \leq \lfloor (H-R)/S \rfloor + 1 \ (=E), \\ 1 \leq k \leq M \end{gathered} \tag{1}$$

where H is the width/height of the IFMP (with padding), E is the OFMP width/height (for a stride S), R is the filter width/height, C is the number of IFMP/filter channels and M is the number of filters/OFMP channels for a given CONV/FC layer. The width and height of the feature-maps/filters are assumed to be same for simplicity and also because it is very common in most of the popular CNNs.

In general, CNNs use real-valued inputs and weights. However, in order to reduce their storage and compute complexity recent works have strived towards using small bit-widths to represent the input/filter-weight values. [6] proposed a binary-weight-network (BWN), where the filter weights (w_i's) can be trained to be +1/-1 (with a common scaling factor per 3-D filter: α). This leads to a significant reduction in the amount of storage required for the w_i's, making it possible to store them entirely on-chip. BWN's also simplify the MAC operation to an add/subtract operation, since α is common for a given 3-D filter and it can be incorporated after finishing the entire convolution computation for that filter. As shown in [6], this algorithm does not compromise much on the original classification accuracy of the CNN, obtained using full precision weights. BWN performs better than binary-connect [7], which does not incorporate the scaling factor of α per filter, and also binarized-neural-networks [8], where both weights and activations are constrained to ± 1.

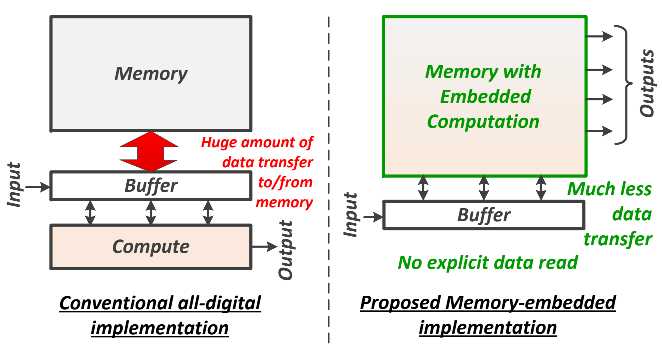

Fig. 2. Comparison of conventional approach vs. proposed approach of memory-embedded convolution computation, for processing of CNNs.

In the conventional all-digital implementation of CNNs [4], [5], [9], with the memory and the processing elements being physically separate, reading the w_i's and the partial sums from the on-chip SRAMs lead to a lot of data movement per computation [10] and hence, make them energy-hungry. This is because, in modern CMOS processes, the energy required to access data from memory can be much higher than the energy needed for a compute operation with that data [11]. To address this problem, we present an SRAM-embedded convolution computation architecture [12], conceptually shown in Fig. 2. Embedding computation inside memory has two significant benefits. Firstly, data transfer to/from the memory is greatly reduced, since the filter-weights are not explicitly read and only the computed output is sent outside the memory. Secondly, we can take advantage of the massively parallel nature of CNNs to access multiple memory addresses simultaneously. This is because we are only interested in the result of the computation using the memory data and not the individual stored bits. Therefore, a much higher memory bandwidth can be achieved with this approach, overcoming some of the major limitations posed by the conventional "von-Neumann bottleneck".

This paper is organized as follows. Section II explains the concept of memory-embedded convolution computation as voltage averaging in SRAMs. Section III presents the overall architecture. Section IV highlights the key contributions of this work, compared to prior in-memory computing approaches. Section V discusses the details of the different circuitry involved in the embedded convolution computation. Section VI presents the measurement results. Finally, concluding remarks are discussed in section VII.

II. Concept of SRAM-Embedded Computation

The basic operation involved in evaluating convolutions (Y) for CNNs is the dot-product of the 3-D IFMP (X) and the filter-weights (W), as shown in equation (1). It can be re-written by flattening the 3-D tensor into a 1-D vector to obtain equation (2), where the 2-D subscripts (x, y) have been omitted for simplicity.

$$Y_k = \sum_{i=1}^{R\times R\times C} W_{k,i} \times X_i \tag{2}$$

Equation (2) can be further simplified for the case of binary filter-weights (w_i's) to get equation (3a), where α_k is the common coefficient for the k^{th} filter. If α_k is expressed as a ratio of two integers (M_k, N: number of elements added per dot-product in 1 clock cycle), then we get equation (3b).

$$Y_k = \alpha_k \sum_{i=1}^{R\times R\times C} w_{k,i} \times X_i, \quad w_i \in (+1,-1) \tag{3a}$$

$$= \frac{M_k}{N} \sum_{i=1}^{R\times R\times C} w_{k,i} \times X_i, \quad M_k, N \in I^+ \tag{3b}$$

Now, if we separate out the scaling factor of M_k (which can be incorporated after computing the entire dot-product), we get the expression for the effective convolution output (Y_{OUT}) as:

$$Y_{OUT,k} = \frac{1}{N} \sum_{i=1}^{R\times R\times C} w_{k,i} \times X_{IN,i} \tag{4}$$

where X_{IN} is the effective convolution input, i.e. scaled version of the original input X, limited to 7-b (includes 1-b sign). For energy-efficient computation with multi-bit values inside the memory, equation (4) has to be implemented in the analog domain, as shown in equation (5).

$$V_{Y\_AVG,k} = \frac{1}{N} \sum_{i=1}^{R\times R\times C} w_{k,i} \times V_{a,i} \tag{5}$$

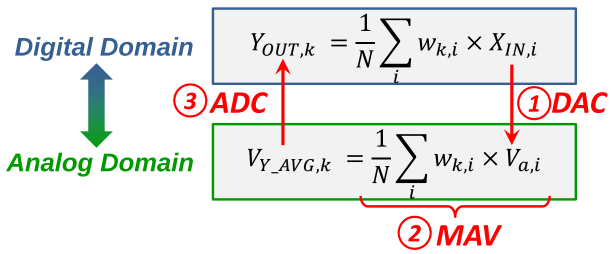

Fig. 3. Concept of embedded convolution computation as averaging in SRAMs for binary-weight convolutional neural networks.

The equivalence of equations (4) and (5), conceptually shown in Fig. 3, becomes apparent in 3 key steps. First, the digital inputs (X_{IN}'s) are converted into analog voltages (V_a's) using digital-to-analog converters (DAC). Then, the analog voltages are multiplied by the corresponding 1-bit filter-weights (w_i's), which are stored in a memory array. This is followed by averaging over N terms to get the analog-averaged convolution output voltage (V_{Y_AVG}). These constitute the second step: multiply-and-average (MAV). Finally, in the last step, the analog-averaged voltage is converted back into the digital domain (Y_{OUT}) using an analog-to-digital converter (ADC), for further processing. It may be noted that if the 3-D filter size ($R \times R \times C$) is greater than N, the above-mentioned 3-step process is repeated multiple (N_r) times using $R\times R\times C'$ ($\leq N$) elements in each cycle, where $N_r = C/C'$. The partial outputs (from the ADC) can then be further added digitally (outside the memory) to get the final convolution output.

III. Overall Architecture

Fig. 4 shows the overall architecture of the 16 Kb Conv-SRAM (CSRAM) array, consisting of 256 rows by 64 columns of SRAM bit-cells. It is divided into 16 local arrays, each with 16 rows. Each local array is meant to store the binary filter-weights (w_i's) for a different 3-D filter in a CONV/FC layer. w_i is stored in a 10T SRAM bit-cell as either a digital '0' or a digital '1', depending on whether its value is $+1$ or -1 respectively. The 10T bit-cell consists of a regular 6T bit-cell and 2 decoupled read-ports. Each local array has its analog averaging circuits (MAV_a's) and a dedicated ADC to compute the partial convolution outputs (Y_{OUT}'s). Sharing these circuits for 16 rows in a local array reduces the area overhead. The IFMP values (X_{IN}'s) are fed into column-wise DACs, which convert the digital X_{IN} codes to analog input voltages on the global bit-lines ($GRBL$'s). The $GRBL$'s are shared by all the local arrays, implementing the fact that in CNNs each input is shared/processed in parallel by multiple filters. With this architecture, the 16 Kb CSRAM array can process a maximum of 64 convolution inputs and compute 16 convolution outputs in parallel.

Fig. 5(a) shows the simulated test error-rates for the MNIST dataset with the LeNet-5 CNN, consisting of 2 CONV layers

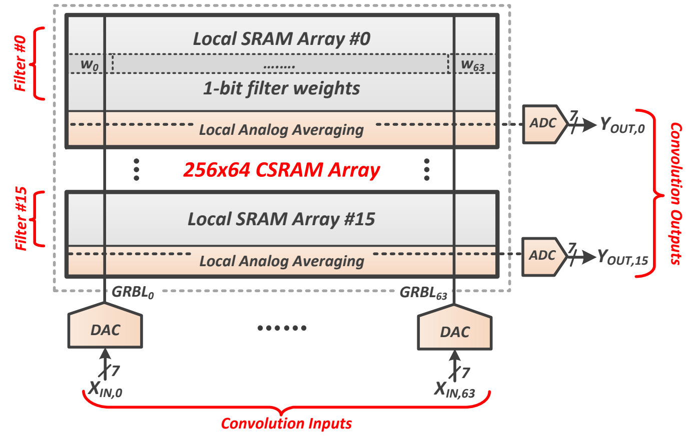

Fig. 4. Overall architecture of the Conv-SRAM (CSRAM) showing local arrays, column-wise DACs and row-wise ADCs to implement convolution as weighted averaging.

(C1, C3) and 2 FC layers (F5, F6). The number of bits to represent the IFMP/OFMP values are varied from 8 to 4. Lower bit-width helps in reducing the area/power costs of the DAC and ADC circuits involved for the convolution computations. However, as seen from Fig. 5(a), the error rate starts to increase steeply for <7-b. Hence, 7-b is chosen as the target bit-width for the DAC/ADC circuits. With 7-b (including the sign bit) the voltage resolution needed on a 1 V scale is 1 $LSB = 1/2^6 \approx 15.6$ mV. Next, the effect of the averaging factor ('N') on the test error-rate is observed. A high value of 'N' would decrease the area/power overhead of the ADC by amortizing it over more MAV operations per clock-cycle. However, higher 'N' can also degrade the computation accuracy due to increased quantization by averaging. This is more crtiticial for CNN layers with smaller filter sizes. As shown in Fig. 5(b) for layer F6, with a 3-D filter size of 120, the error-rate steeply increases as 'N' is varied from 15 to 120. For the other 3 layers of LeNet-5, the 2-D filter size is 5×5. Hence, a minimum $N = 25$ is required to fit at-least one full filter channel per CSRAM row. We chose $N = 64$ to fit 2 channels for 5×5 filters, without sacrificing much on the error rate.

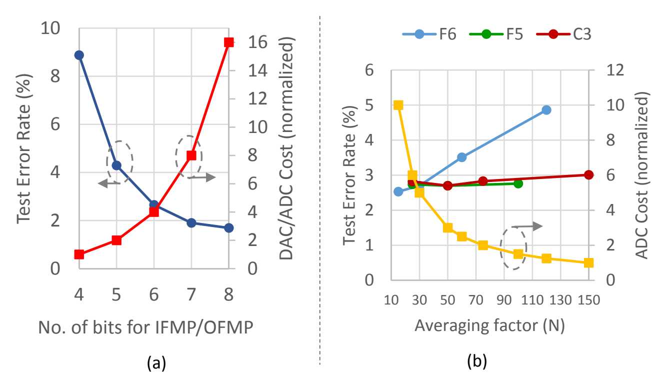

Fig. 5. Simulated results for the MNIST dataset with the LeNet-5 CNN by varying: (a) bit-width to represent IFMP/OFMP values, (b) averaging factor (N).

The number of rows (N_{rows}) per local array in the CSRAM determines the unit capacitance (C_{LBL}), which is used for all the analog operations required for the in-memory convolution computation. For every column in a local array, there is a corresponding $\mathrm{MAV_a}$ circuit. Hence, a higher value of N_{rows} would decrease the area-overhead of $\mathrm{MAV_a}$, by amortizing it over multiple rows. It also reduces variation of the C_{LBL} value, which helps in improving accuracy of the computations. However, a high value of N_{rows} means a high C_{LBL}, which translates to increased energy costs. It would also lead to less throughput for a given SRAM size, since fewer outputs would be computed per cycle. Therefore, $N_{rows} = 16$ is chosen as a trade-off. It may be noted that with $N_{rows} = 16$, the thermal noise ($\frac{kT}{C}$) is < 1 mV, which is well below 1 LSB = 15.6 mV.

IV. Key contributions of this work

While there are a few different approaches [13]–[17] for in/near-memory computing, the proposed architecture has some key contributions, which provide significant benefits over prior work. The first key feature of our approach is the robustness to SRAM bit-cell V_t variations. SRAM bit-cells use near-minimum transistor sizes available in a given CMOS process and hence, suffer from transistor mismatch and variation. For example, if we consider the discharge current (I_{cell}) through an SRAM bit-cell (shown in Fig. 6) we can observe that it has a significant spread from its mean value ($\sigma \approx 30\%\mu$). Now, when I_{cell} is used to modulate the analog voltage (V_a) on the bit-line [13]–[15], [17], there is a wide variation in the V_a value and it cannot be controlled very well. This compromises the computation accuracy and extra algorithmic techniques might be required to compensate for that. [13] uses the 'AdaBoost' technique, in which the results of many weak classifiers are combined to get a more accurate final result. However, this would lead to an increase in the number of computations and the energy required. [15] proposed an on-chip training to compensate for chip-to-chip variations. However, this would incur energy and timing penalty required to re-train the network corresponding to every single chip. In our approach (Fig. 6), the analog voltage (V_a) is directly sent to the bit-lines using global DACs at the periphery. Since the global DACs can be upsized, with their area being amortized over mutliple rows (256 in this case), the variation due to it is significantly less compared to that of the bit-cell. Furthermore, the SRAM bit-cell is only used to multiply V_a by the 1-b filter weight (w_i) stored in it, using full signal swing locally. That means, the purpose of the SRAM bit-cell is to discharge one of its local bit-lines to 0, it is not used to control V_a. Hence, given enough time for the worst-case bit-cell discharge, the computation accuracy does not suffer from local bit-cell V_t variations.

The second key feature of our approach is the improvement of the dynamic voltage range for the analog computations without disturbing any bit-cell. In the conventional approach (with 6T SRAM bit-cells) [13], [14], [17], where multiple word-lines (WL) are activated for the same bit-line, there might be a situation where one of the accessed bit-cells in that column is in pseudo-write mode (Fig. 7). This is because multiple activated bit-cells in that column can discharge a bit-line to a very low voltage, which could overwrite the data

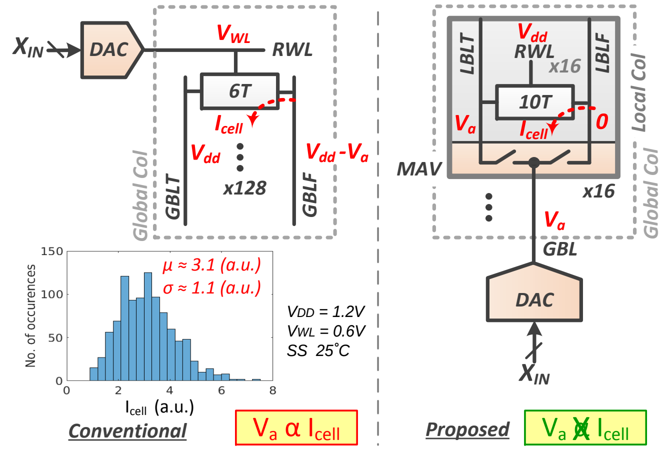

Fig. 6. Comparison of the conventional and proposed approaches of using SRAM bit-cells for embedded analog computations.

stored (Q_k = '1') in the disturbed bit-cell. Hence, the bit-line voltage range has to be limited to prevent any write-disturb. In our approach, 10T bitcells are used which de-couple the read and the write ports, to prevent any write-disturb. Furthermore, each bit-cell is read independently in parallel without sharing any bit-lines. And hence, the discharge on one bit-line cannot affect another accessed bit-cell. Thus, we can utilize a wide voltage range (close to full-rail) for the analog computations, without disturbing any bit-cell. It may be noted that, although a 10T bit-cell has more transistors than a 6T, it can be designed using smaller sized devices, compared to a 6T bit-cell. This is because, unlike a 6T, a 10T bit-cell does not have conflicting sizing requirements to achieve high margins for both read and write operations. In addition, due to the limitations of 6T bit-cells for in-memory analog computations, network augmentation, i.e. larger sized neural networks, might be required to compensate for lower computation accuracy. Larger neural networks translate to increased storage requirement for filter-weights on-chip and hence, increased SRAM size. Therefore, overall, our 10T bit-cell based in-memory architecture is not necessarily higher area than a 6T based design.

The third key feature of our work, which distinguishes it from other "in-memory" computing approaches [14], [16], is the use of the inherent bit-line capacitances in the SRAM array to implement the computations. This precludes the need for extra area-intensive capacitors, which would be otherwise required at the SRAM periphery [14] to implement some of the analog computations.

Finally, this work supports multi-bit resolution for the inputs and outputs of the dot-products, compared to [13] (output: 1-b) and [16], [17] (both input/output: 1-b). This helps in achieving higher classification accuracy for a neural-network of a given size.

All the key features, described above, make our proposed architecture scalable i.e. multiple CSRAM arrays can operate in parallel to run larger neural networks.

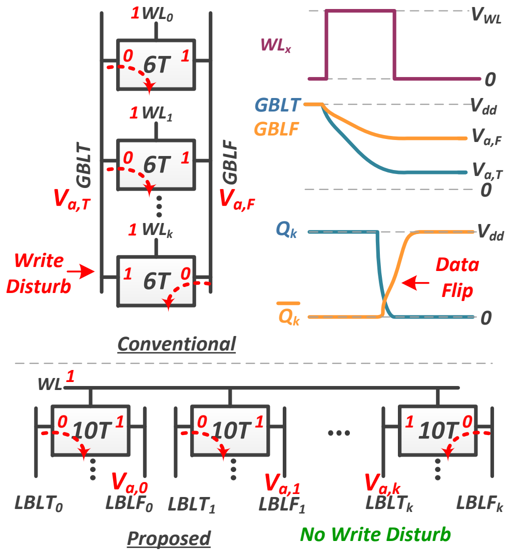

Fig. 7. Comparison of the conventional and proposed approaches on write-disturb issue of SRAM bit-cells during compute mode.

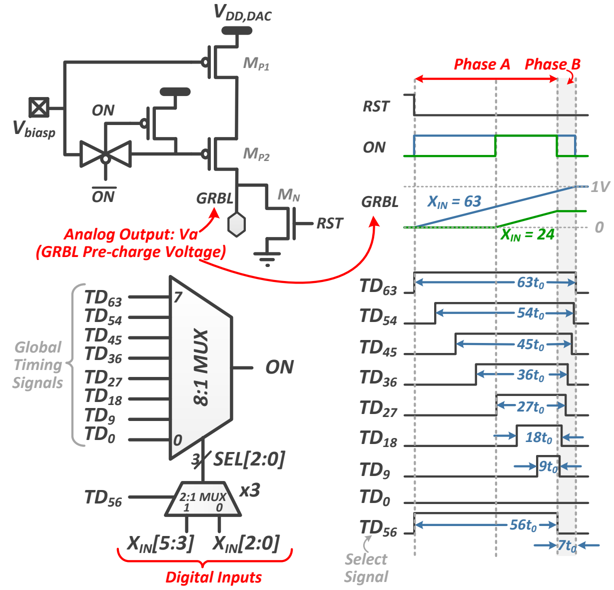

Fig. 8. Schematic of the column-wise GBL_DAC circuit, showing the digital-to-time converter (bottom-left) and time-to-analog converter (top-left). Also shown are the timing signals and operation waveforms for 2 input codes (right).

V. Circuits for the 3-Phase Conv-SRAM Operation

A. Phase-1: DAC

During the first phase of the Conv-SRAM (CSRAM) operation the digital convolution input (X_{IN}) is converted into an analog voltage (V_a) using a column-wise digital-to-analog converter (GBL_DAC). The analog voltage is used to pre-charge the global read bit-line ($GRBL$) and the local bit-lines (not shown in Fig. 4) in the SRAM array. Each $GRBL$ is shared by all the 16 local arrays and hence, they get the same value of the analog pre-charge voltage. This implements the fact that in a given CNN layer (CONV/FC) each input is processed simultaneously by multiple filters. Furthermore, all the 64 column-wise GBL_DACs operate in parallel and can send a maximum of 64 analog inputs to the CSRAM array in one clock cycle.

Fig. 8 shows the schematic of the proposed GBL_DAC circuit. It consists of a cascode PMOS stack biased in the saturation region to act as a constant current source. The $GRBL$ is charged with this fixed current for a time t_{ON}, which is determined by the ON pulse-width. t_{ON} is modulated based on the digital input code ($X_{IN}[5:0]$), using a digital-to-time converter. To achieve a very good linearity of V_a vs X_{IN} or t_{ON} vs X_{IN}, there should be a single continuous ON pulse for every input code, to avoid non-linearities due to multiple charging phases. This is not possible to generate by simply using 6 timing signals with binary-weighted pulse-widths. However, it may be generated using 2^6 or 64 timing signals and a 64:1 mux. But that would consume a lot of area, which is not ideal for a circuit that needs to be replicated for each column of the SRAM array. To address this issue, we present a 2-phase architecture in which the 3 MSBs of X_{IN} are used to select the ON pulse-width for the first half of charging and the 3 LSBs for the second half. A control signal (TD_{56}) is used to choose between the 2 phases. In this way an 8-to-1 mux, with 8 timing signals, can be shared during both the phases, to reduce the area overhead and the number of timing signals to route. A tree-based architecture, using 2:1 unit mux's, is used for the 8:1 mux to equalize the mux-delay for different control bits.

To design the pulse-widths of the 8 timing signals, we need to express X_{IN} in terms of its 2 components:

$$\begin{aligned} X_{IN,dec} &= 8 \times k_A + k_B, \\ k_A &= Decimal(X_{IN}[5:3]), \\ k_B &= Decimal(X_{IN}[2:0]) \end{aligned} \tag{6}$$

where k_A and k_B are the decimal values for the 3 MSBs and the 3 LSBs of X_{IN} respectively. Since k_A and k_B can have any integer values from 0 to 7, the pulse-widths of the timing signals (TD's) are chosen as:

$$\begin{aligned} t_{TD_{9k}} &= 8 \times kt_0 + kt_0 = 9 \times kt_0 \\ k &\in (0, 1, .., 7) \end{aligned} \tag{7}$$

where t_0 is the minimum time resolution. A delay-line architecture, with a controllable unit delay of t_0, is used to generate 64 time-delayed signals from the input clock. Then the appropriate signals are combined using NOR gates to generate the TD's. This is done at the global level and the generated TD's are buffered and routed to all the GBL_DACs. A higher value of t_0 reduces the non-linearities from the timing generation circuitry, at the cost of increased clock cycle time.

To understand how the 2-phase charging technique works, let us consider two X_{IN} values of 24 and 63, as shown in Fig. 8. For $X_{IN} = 24 = 8 \times 3 + 0$, k_A is 3 and k_B is 0. Hence, $TD_{9\times3}$ or TD_{27} is used in phase A and TD_0 is used in phase B, to select the pulse-width of the ON timing signal. Similarly, for the code $X_{IN} = 63 = 8\times7+7$, both k_A and k_B are 7 and hence, TD_{63} is used in both the charging phases.

In addition to the linearity aspect of the DAC transfer function, this architecture also performs better in terms of device mismatch, compared to binary-weighted PMOS charging DACs [13]. This is because, here, the same PMOS stack is used to charge the global bit-line for all input codes, rather than having to use smaller PMOS devices for small input values. Furthermore, the pulse-widths of the globally generated timing signals have less variations typically, compared to those arising from local V_t-mismatch in the PMOS devices [13].

It may be noted that, a one-time calibration is required to set the maximum value of the analog pre-charge voltage for the maximum input code ($X_{IN,max}$). The maximum pre-charge voltage should be kept lower than the supply voltage of the GBL_DAC, to ensure that the PMOS cascode stack is operating in the saturation region as a constant current source. For a given t_0, the calibration can be achieved by tuning the externally provided bias voltage (V_{biasp}) of the PMOS stack. During calibration, all DACs are fed the same input value of $X_{IN,max}$. In a given clock cycle, first, the GBL_DAC pre-charges the $GRBL$ to an analog voltage (V_a). Then, V_a is compared to an externally provided reference voltage V_{ref} (typically kept at 1 V in this work). The comparison is done by the column-wise sense-amplifiers (SA), which are already present for normal read-out of the SRAM. All the 64 SAs operate in parallel and use the same V_{ref} to provide 64 comparison outputs simultaneously. V_{biasp} is monotonically increased from 0 V until majority of the SAs ($> 50\%$) flip their outputs ('1' to '0'), at which point the calibration is achieved. In this work, a 5 mV step-size is used to tune V_{biasp}.

B. Phase-2: Multiply-and-Average

The second phase of the Conv-SRAM operation involves the mutliplication of the analog input voltages (V_a's) with the 1-b filter weights (w_i's) and averaging over N values. This multiply-and-average (MAV) operation is done in parallel for all the 16 local arrays, each storing the w_i's for a different 3-D filter when running a CONV/FC layer.

$$V_{Y\_AVG,k} = \frac{1}{N}\sum_{i=1}^{N} w_{k,i} \times V_{a,i}, \;\; 0 \le k \le 15, N \le 64$$
$$V_{Y\_AVG} = Vp_{AVG} - Vn_{AVG} \tag{8}$$

Fig. 9 shows the details for the MAV operation for one local array. It starts by turning on the read word-line (RWL) for the selected row in the local array. This leads to discharging of one of the local bit-lines ($LBLT, LBLF$) in each column, depending on the w_i stored in the corresponding 10T bit-cell. A positive w_i (+1) is stored as a digital '0' and a negative w_i (−1) as a digital '1'. It may be noted that, the local bit-lines have been pre-charged to the same analog voltage ($V_{a,i}$) as its corresponding global bit-line ($GRBL$) during phase-1. Therefore, at the end of weight evaluation, the difference between the local bit-line voltages represents the product of the analog voltage ($V_{a,i}$) and the 1-b weight (w_i). For example, the bit-cell in the '0^{th}' column stores a −1 and hence, $\Delta V_{LBL,0} = V_{LBLT,0} - V_{LBLF,0} = -V_{a,0}$.

The weight multiplication/evaluation step is completed by turning off the RWL. After that, the appropriate local bit-lines are shorted together horizontally to evaluate the average. The

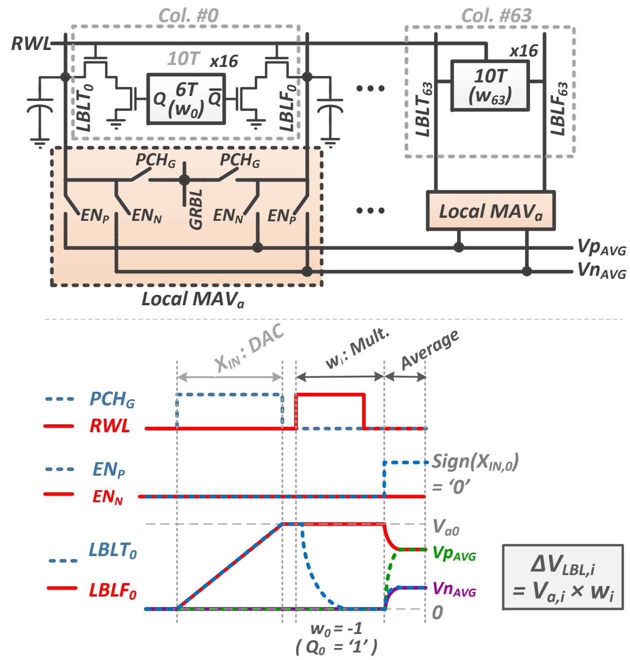

Fig. 9. Architecture of a 16×64 local array of the Conv-SRAM, showing the 10T bit-cells storing the filter weights and local analog multiply-and-average ($\mathrm{MAV_a}$) circuits. Also shown are typical operation waveforms (bottom) for one column.

positive and negative parts of the average as obtained on two separate voltage rails, Vp_{AVG} and Vn_{AVG}, respectively. This is implemented by the local $\mathrm{MAV_a}$ circuits, which pass the voltages of the $LBLT$'s and $LBLF$'s to either the Vp_{AVG} or the Vn_{AVG} voltage rails, depending on the sign of the input X_{IN}. If the input for the particular column is positive ($X_{IN} > 0$) EN_P is turned on, otherwise EN_N is on. EN_P and EN_N are digital control signals which are globally routed and shared column-wise by all the 16 local arrays. The switches controlled by EN_P and EN_N are implemented using NMOS pass transistors, since the final Vp_{AVG}, Vn_{AVG} voltages would be closer to 0 V than V_{ref}. On the other hand, the switches controlled by PCH_G use CMOS transmission gates, since they need to pass a wide range of voltages from 0 V to V_{ref} ($\sim$ 1 V), during phase-1 (DAC pre-charge).

The fully-differential nature of the averaging architecture helps in mitigating many common-mode noise issues, e.g. clock coupling noise from the control switches, capacitance variation of the local bit-lines and the voltage rails due to different process corners, etc. This helps in improving the accuracy of the dot-product computations with our approach.

It may be also noted that during this phase, when the SRAM bit-cell is actually used for weight-evaluation, the time required does not have a huge variation. Fig. 10 shows the simulated local bit-line discharge time ($t_{dis,LBL}$) in the slowest process corner (SS). As seen from the figure, even the 6σ value of $t_{dis,LBL}$ is merely 500 ps, which is much smaller than the total clock period ($\approx$ 100 ns). This shows that bit-cell V_t variations do not dominate the overall computation time. The longer clock period is justified due to the highly parallel processing in the compute mode.

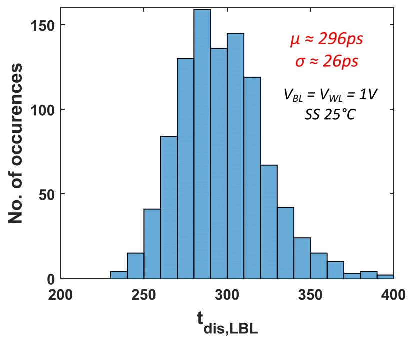

Fig. 10. Variation of the local bit-line discharge time for weight evaluation/multiplication in phase-2.

C. Phase-3: ADC

The third and last phase of the Conv-SRAM operation is the analog-to-digital conversion of the dot-product outputs, with multi-bit resolution. The difference of the analog average voltages (Vp_{AVG} and Vn_{AVG}) is fed to an ADC to get the digital value of the computation (Y_{OUT}). This is done in parallel for all the 16 local arrays, producing outputs corresponding to 16 different filters simultaneously.

Choosing the ADC architecture is crucial since it would be replicated multiple times in the CSRAM array. Hence, area and power consumption are key metrics to consider. In addition, the typical distribution of the ADC outputs (Y_{OUT}'s) should also be considered to find the more appropriate architecture. As seen from simulation results in Fig. 11, for a typical CONV layer with a full scale input range of ± 31, Y_{OUT} has an absolute mean value of ± 1.3 and is typically limited to ± 7. Hence, a serial integrating ADC architecture is more suitable in this scenario, compared to other area-intensive (e.g. SAR) and more power-hungry ones (e.g. flash). In spite of its serial nature, in most cases we can expect the ADC to finish its operation within a few cycles, due to the particular Y_{OUT} distribution.

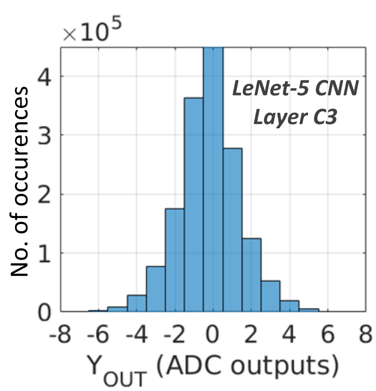

Fig. 11. Simulated distribution of the partial convolution output from the ADC (Y_{OUT}), for a typical CONV layer (C3) in the LeNet-5 CNN.

Fig. 12 shows the architecture of the proposed integrating ADC (CSH_ADC). It consists of 3 main parts: a charge-sharing based integrator, a sense-amplifier (SA) and a logic block. Capacitive charge-sharing with replica bit-lines is used to implement the integration. The use of replica bit-lines helps to track the local bit-line capacitance better in the presence of process and temperature variations. The SA has a standard StrongARM latch-type architecture. PMOS devices are chosen for the input differential pair of the SA, since the common mode voltages of Vp_{AVG} and Vn_{AVG} signals are expected to be closer to the GND rail. The logic block provides the timing signals for the charge-sharing (PCH_R, EQ_P, EQ_N) and the SA comparison (SA_EN), using the globally provided timing signals (ϕ_1, ϕ_2). It also has a counter to count the number of cycles it takes to finish the ADC operation and that provides the digital output of the dot-product computation.

Fig. 12 also shows the waveforms for a typical CSH_ADC operation. It starts by sending a SA_EN pulse from the ADC logic block to the SA. The SA compares Vp_{AVG} and Vn_{AVG}, and sends its outputs (SAO_P, SAO_N) to the ADC logic block. The first comparison determines the sign of the output, e.g. for the case shown in Fig. 12, Y_{OUT} is positive since Vp_{AVG} is higher than Vn_{AVG}. After the first comparison, the lower of the 2 voltage rails (Vn_{AVG}) is integrated by charge-sharing it with a reference local bit-line (BLN_{ref}), using the equalize signal (EQ_N in this case). The reference bit-line, which replicates the local bit-line capacitance, was pre-charged during the SA comparison using the PCH_R signal to V_{ref} (= 1 V in this work). Therefore, the step-size of the integration is $\approx \frac{V_{ref}}{N}$, where N is the number of SRAM local columns that were averaged. The pre-charge and equalize/integrate operations, along with the SA comparison, continue until the lower voltage rail (Vn_{AVG}) exceeds the higher one (Vp_{AVG}). When this happens, the SA outputs flip indicating the end-of-conversion (EOC). After this, no more timing pulses are generated. A counter in the ADC logic block counts the number of equalize pulses (EQ_N) it takes to reach EOC and that generates the digital value of the convolution/dot-product output (Y_{OUT}), which is $+4$ for the example shown in Fig. 12.

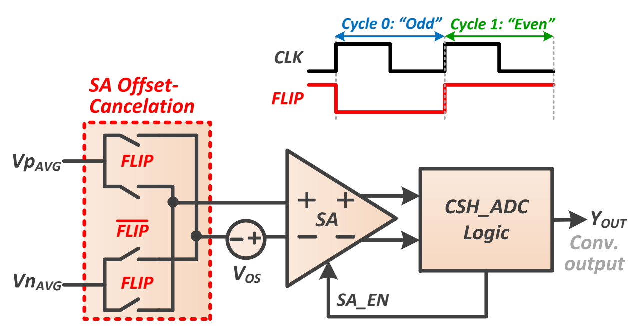

Fig. 13. Circuit for the 2-cycle offset-cancellation technique for the SA in CSH_ADC.

It may be noted that Y_{OUT} is directly affected by the SA offset voltage (V_{OS}), which can degrade the overall computation accuracy due to incorrect ADC outputs. To address this issue, we propose a simple 2-cycle offset-cancellation technique, using a flipping mux at the input of the SA (Fig. 13). During the first/even cycle of this 2-cycle period, $FLIP$ = '0'. Hence, Vp_{AVG} and Vn_{AVG} are passed to the positive and negative input terminals of the SA respectively. Therefore,

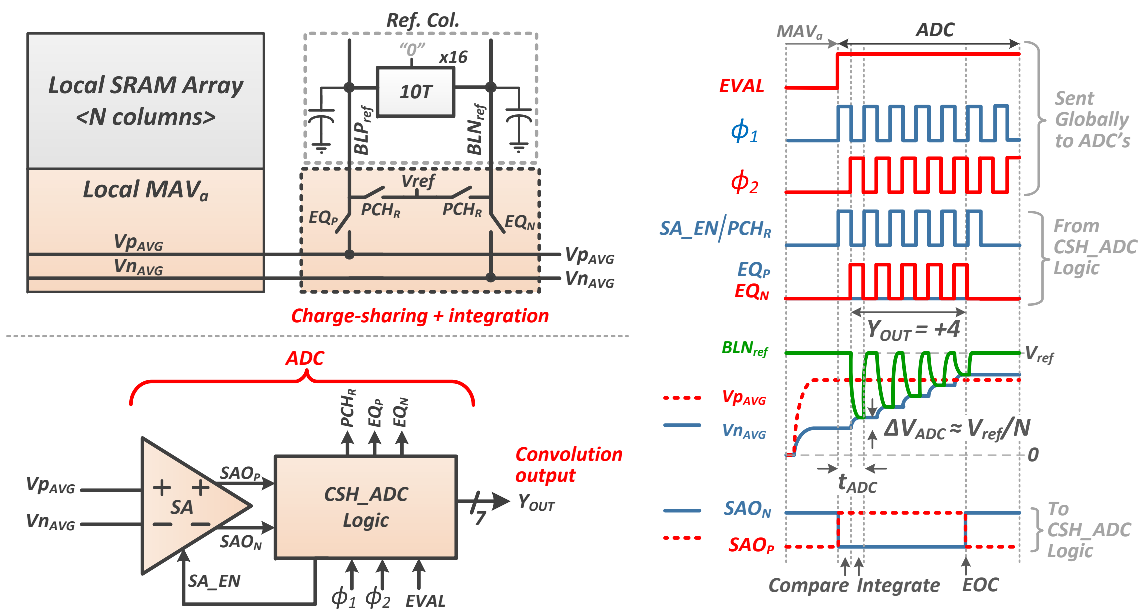

Fig. 12. Architecture (left) for the charge-sharing based ADC (CSH_ADC) for 1 local array of the Conv-SRAM and typical waveforms (right) for the digital output (Y_{OUT}) computation for the convolution (dot-product) operation.

$Y_{OUT,0} = ADC(Vy_{AVG,0} - V_{OS})$. The output in this cycle is exactly same as in the conventional case (without the flipping mux). However, during the next odd cycle, the inputs to the SA are flipped by setting $FLIP$ = '1'. Hence, a differential voltage of $(Vn_{AVG} - Vp_{AVG}) = -Vy_{AVG}$ is applied to the input of the SA. To get the correct polarity at the output of the ADC, another negation is applied by the ADC logic block. This results in $Y_{OUT,1} = -ADC(-Vy_{AVG,1} - V_{OS}) = ADC(Vy_{AVG,1} + V_{OS})$, as compared to $Y_{OUT,1} = ADC(Vy_{AVG,1} - V_{OS})$ for the conventional case. Finally, we add the Y_{OUT}'s for the 2 consecutive cycles to accumulate the partial results for the convolution. With our proposed 2-cycle approach the effect of V_{OS} is inherently canceled since:

$$\begin{aligned} Y_{OUT} &= Y_{OUT,0} + Y_{OUT,1} \\ &= ADC(Vy_{AVG,0} + Vy_{AVG,1}) \end{aligned} \tag{9}$$

On the other hand, for the conventional case the effect of V_{OS} adds up since:

$$Y_{OUT} = ADC(Vy_{AVG,0} + Vy_{AVG,1} - 2 \times V_{OS}) \tag{10}$$

and this makes the accumulation result further inaccurate. It may be noted that, the benefits of this offset-cancellation technique comes without any extra timing and power penalty, as long as an even number of cycles are required to finish a full convolution computation. This can be easily expected for most CNNs.

VI. Measured Results

The 16 Kb CSRAM array was implemented in a 65-nm LP CMOS process. The die photo in Fig. 14 shows the relative area occupied by the different key blocks. The bit-cell array (ARY) along with its peripheral circuitry occupies 73.1% of the total CSRAM area, 8.2% is occupied by the GBL_DACs, 8.6% by the local $\mathrm{MAV_a}$ circuits, 7.3% by the CSH_ADCs and the rest by global timing circuits. The test-chip summary is shown in Table I.

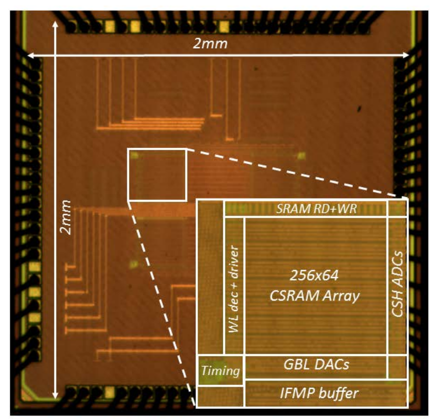

Fig. 14. Die micro-photograph in a 65-nm CMOS process.

A. Circuit Characterizations

Fig. 15 shows the measured transfer function of the 6-b GBL_DAC. With V_{ref} = 1 V and $t_0 \approx 250$ ps, a 5-b voltage resolution is effectively achieved in measurements. Hence, we set the LSB of X_{IN} to '0'. To estimate the DAC analog output voltage (V_a), V_{GRBL} for the 64 columns of the CSRAM are compared to an external voltage (V_{ref}) by column-wise SAs, as explained before for DAC calibration (Section V-A). For each X_{IN}, the V_{ref} at which more than 50% of the SA outputs flip is chosen as an average estimate of V_a. As mentioned before, an initial one-time calibration is needed to set $V_{a,max} = 1$ V for X_{IN} = 31 (max. input code). The supply voltage for the DACs is fixed at 1.2 V, to keep the PMOS

TABLE I
TEST-CHIP SUMMARY

Technology	65-nm
CSRAM size	16 Kb
CSRAM area	0.063 mm^2
Array organization	256×64 (10T bit-cells)
# column DACs	64
# row ADCs	16
Max. # MAVs	64×16
Supply voltages	1.2 V (DAC), 0.8 V (array), 1 V (rest)
Main clock freq. (compute mode)	5 MHz
ADC clock freq.	250 MHz

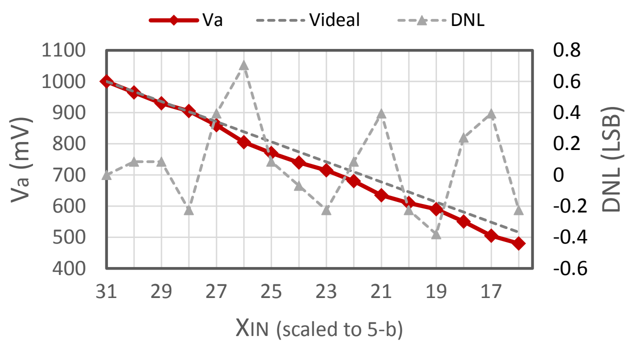

Fig. 15. Measured transfer function of GBL_DAC at $V_{dd,DAC}$ = 1.2 V, with V_{ref} = 1 V and $t_0 \approx$ 250 ps.

stack in them operating in the saturation region (as a constant current source). It can be seen from Fig. 15 that there is a good linearity in the DAC transfer function with $DNL < 1$ LSB. Since the SAs have NMOS input-pair, low values of V_a cannot be properly estimated. Hence, the characterization is done till X_{IN} = 16 (or $V_a \approx$ 500 mV).

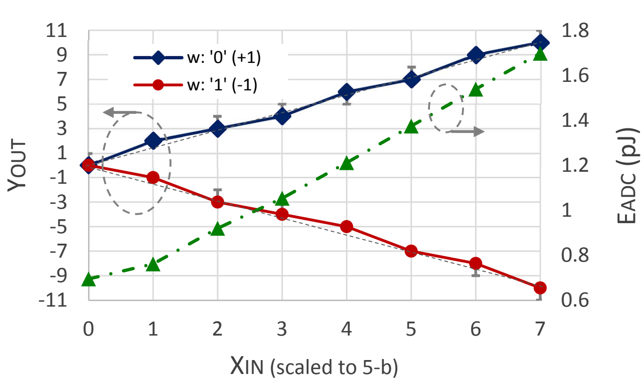

Fig. 16. Measured transfer function and energy consumption of CSH_ADC at $V_{dd,ADC}$ = 1 V, $V_{dd,ARY}$ = 0.8 V and f_{ADC} = 250 MHz.

Fig. 16 shows the transfer function of the CSH_ADC, operating at 1 V and a clock frequency of 250 MHz (generated on-chip with a free-running VCO). The array voltage ($V_{dd,ARY}$) is kept at 0.8 V to reduce the clock-coupling noise from the WL's, when reading the weights. To characterize the CSH_ADCs, all X_{IN}'s are fed the same input code, all w_i's are written the same value and then the ADC outputs (Y_{OUT} 's) are observed. The measurement results show a good linearity in the overall transfer function and low variation in the Y_{OUT} values, which is due to the fact that the variation in BL capacitance (used in CSH_ADC) is much lower than transistor V_t-variation. It can be also seen from Fig. 16 that the energy/ADC scales linearly with the input/output value, which is expected for the integrating ADC topology.

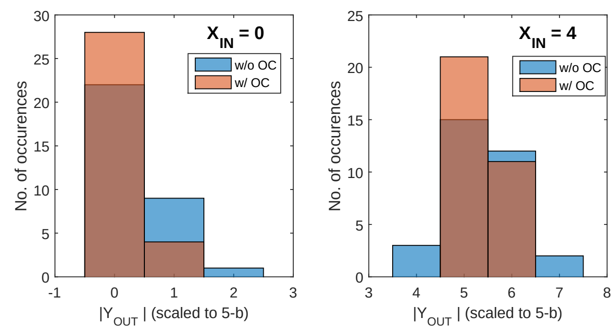

Fig. 17. Measured distribution of convolution output values (Y_{OUT}) from CSH_ADC with and without the offset-cancellation (OC) technique, for two values of the input code (X_{IN}).

The effect of the offset-cancellation (OC) technique for the SA (in the CSH_ADC) is also characterized, as shown in Fig. 17 for two different input codes. It can be clearly seen that the OC helps in reducing the variation of the Y_{OUT} values, leading to a better computation accuracy for the dot-products/convolutions.

B. Test Case: MNIST Dataset

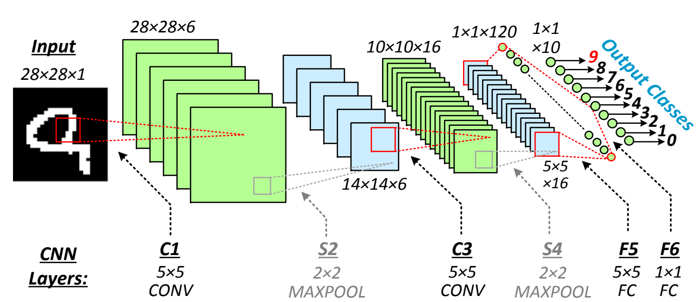

Fig. 18. Architecture of the LeNet-5 CNN, showing the sizes of the feature maps (top) and the filters (bottom).

To demonstrate the functionality for a real CNN architecture, the MNIST hand-written digit recognition dataset is used with the LeNet-5 CNN [18]. As shown in Fig. 18, LeNet-5 consists of 2 CONV layers (C1, C3) and 2 FC layers (F5, F6). In addition, there are 2 sub-sampling or max-pooling layers (S2 and S4, following layers C1 and C3 respectively) and a non-linear ReLU layer (R5 after layer F5). Only the CONV/FC layers, which involve majority of the computations, are implemented on-chip by the CSRAM array. The non-linear layers are implemented in software. Fig. 19 shows the test setup used to automatically run the 4 CONV/FC layers, one

TABLE II
PARAMETER MAPPING FOR THE CONV/FC LAYERS OF LENET-5 CNN TO THE CSRAM ARRAY

Parameters (↓)	**C1**	**C3**	**F5**	**F6**
3-D Filter size	5×5×1	5×5×6	5×5×16	1×1×120
# 3-D Filters	6	16	120	10
# Local ARYs used	6	16	15†	10
# IFMP channels/row	1	2	2	30*
Rows, Col.s/local ARY	1, 25	3, 50	8, 50	4,* 30*
# col.s for AVG (N)	32	50	50	32
# operations/cycle	25×6×2	50×16×2	50×15×2	30*×10×2

† Repeated 8 times to cover all the 120 filters
* 15 columns and 8 rows mapping used at V_{DD} = 0.8 V

after the other, on the test-chip. Data is transferred back and forth between MATLAB (running on a host PC) and the test-chip, via an FPGA board. Table. II shows the detailed mapping of the 4 CONV/FC layers to the CSRAM array to compute the convolutions. Let us first consider layer C3. It has a filter size of 5×5, with 6 input channels and 16 output channels (number of 3-D filters). Each of the 16 3-D filters are mapped to one of the 16 local arrays in the CSRAM. Since each row in the local array has 64 bit-cells, a maximum of 2 ($= \lfloor \frac{64}{5\times5} \rfloor$) input channels can fit per row. Therefore, 3 ($= \frac{6}{2}$) rows are required in each local array to fit the entire 3-D filter. In every clock cycle, 50 ($= 5 \times 5 \times 2$) X_{IN}'s are sent through a buffer (shift-registers) to the CSRAM array to compute 16 partial convolution outputs. Thus, the CSRAM array processes $50 \times 16 \times 2$ operations (1 MAV = 2 OPs: 1 multiply + 1 add/average) per clock cycle. For layer F5, the entire filter cannot fit at once in the CSRAM array (due to its limited 16 Kb size in the test-chip). Hence, the entire process, explained above, is repeated multiple times to finish all the computations. However, having multiple CSRAM arrays operating in parallel can easily alleviate this problem, by fitting all the filter weights together on-chip.

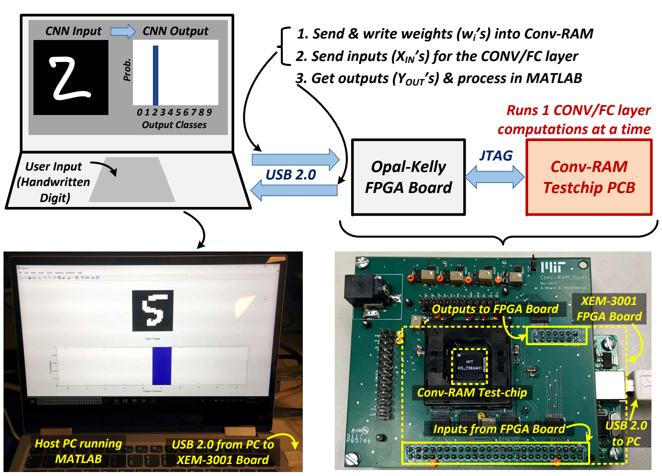

Fig. 19. Test setup for automatically running the 4 CONV/FC layers of LeNet-5 CNN on Conv-SRAM, for a given input image (28×28).

Fig. 20 shows the measured error rate for the 10,000 test images in the MNIST dataset, with the 4 CONV/FC layers being successively implemented on-chip. 3 different chips are measured, each experiment is repeated multiple times, and the average value of the error rate is reported. We tested 2 different versions of LeNet-5: with and without Batch-Normalization (BN) layers preceeding the CONV/FC layers. Without BN layers ('v1') we achieve a classification error rate of 2.5% after all the 4 layers. The error rate is improved to 1.7% by using the BN layers ('v2'). This is mostly because BN normalizes the convolution inputs for every layer, with a mean around 0 and also limits the maximum value of the inputs. Hence, after input quantization to 6-b, its features are better preserved compared to an un-normalized input distribution. The measured error rate, which is close to the expected value from an ideal digital implementation, shows the robustness of the CSRAM architecture to compute convolutions. The error rate for the MNIST dataset is improved by 8.3% compared to prior work on in/near-memory compute [13], [16], where a 10% error rate was achieved. Next, we tested functionality at a lower voltage setting of $V_{dd,DAC}$ = 1 V and the rest of the circuits operating at V_{dd} (rest) = 0.8 V, with a clock period of 400 ns. The maximum DAC pre-charge voltage ($V_{a,max}$), corresponding to the maximum input code, is calibrated to 0.8 V. Hence, the magnitude of 1 LSB is $\sim$ 26 mV (instead of 32 mV for the previous case with $V_{a,max}$ = 1 V). Fig. 21 shows the measured error rate for this set of voltages. Due to reduced analog voltage precision, the error rates are slightly higher, with 'v1' achieving 3.4% and 'v2' achieving 1.9% for the MNIST test dataset.

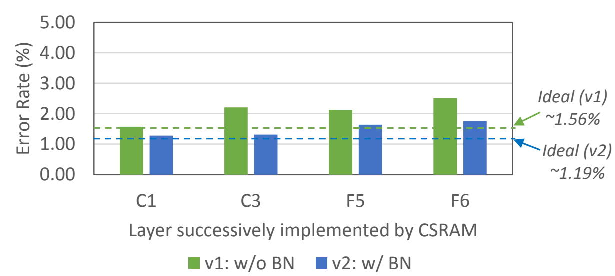

Fig. 20. Measured error rate for the 10K images in the MNIST test dataset using LeNet-5 CNN, with and without BN, at $V_{dd} = V_{a,max}$ = 1V.

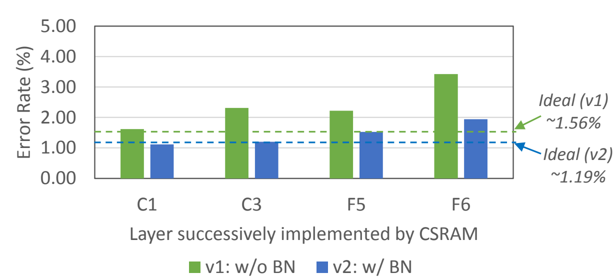

Fig. 21. Measured error rate for the 10K images in the MNIST test dataset using LeNet-5 CNN, with and without BN, at $V_{dd} = V_{a,max}$ = 0.8V.

The distributions of the partial convolution outputs from the ADC (Y_{OUT}'s) are shown in Fig. 22, for all the 4 CONV/FC layers. For each of these layers, Y_{OUT} has a mean around $\approx$ 1 LSB, which justifies the use of the serial ADC topology to compute it. Fig. 23 shows the distributions of the convolution inputs (X_{IN}'s) for the 4 layers. X_{IN}'s have been properly

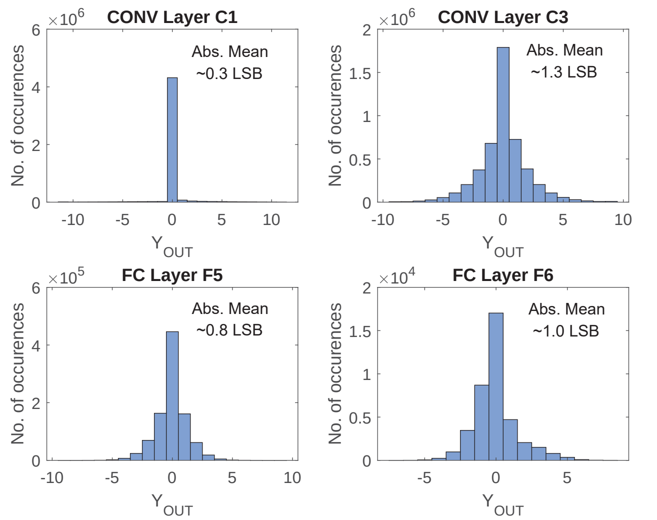

Fig. 22. Measured distribution of the partial convolution outputs (Y_{OUT}'s) for the 4 different CONV/FC layers of the LeNet-5 CNN.

scaled and quantized to 6-b (including sign bit) before being sent to the CSRAM array to compute the convolutions. As seen from the figure, all the layers have a high proportion of 0's for the X_{IN}'s. This helps in reducing the GBL_DAC energy to convert and send them to the columns of the CSRAM array.

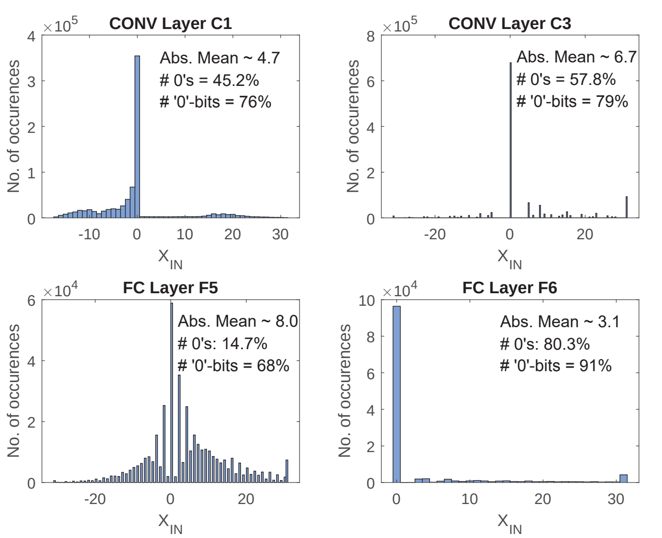

Fig. 23. Measured distribution of the 6-b convolution inputs (X_{IN}'s) for the 4 different CONV/FC layers of the LeNet-5 CNN.

Fig. 24 shows the overall energy consumption of the CSRAM array for running the different layers of LeNet-5, with $V_{dd,DAC}$ = 1.2 V, $V_{dd,ARY}$ = 0.8 V, V_{dd} (rest) = 1 V and f_{clk} = 5 MHz. Of the 4 CONV/FC layers in LeNet-5, the energy consumptions while running layers C1 and F6 are lower than that of layers C3 and F5. This is because layers C1 and F6 do not fully utilize the entire CSRAM array, due to their small filter sizes. However, that also translates to a lower energy-efficiency for these layers (Table III), since the energy is amortized over fewer MAV operations. A higher array utilization in layer C3 (all 16 local arrays) helps in achieving an energy-efficiency of 38.8 TOPS/W ('v1'), by consuming 41.3 pJ of energy for $50 \times 16 \times 2 = 1600$ operations. Whereas, for 'v2' (with BN), layer F5 achieves the best energy-efficiency of 40.3 TOPS/W, utilizing 15 of the 16 local arrays. Fig. 24 also shows the energy breakdown for the 3 major components: GBL_DAC, ARY+$\mathrm{MAV_a}$ and CSH_ADC. The energy for the GBL_DACs is limited by the bit-precision requirement for representing the IFMP values. Whereas, the energy for the ARY, $\mathrm{MAV_a}$ and CSH_ADC circuits can be scaled down by scaling their supply voltages while sacrificing speed. Fig. 25 shows the measured energy consumption of the CSRAM array, with $V_{dd,DAC}$ = 1 V, V_{dd} (rest) = 0.8 V and f_{clk} = 2.5 MHz. The reduced supply voltages help in decreasing the energy consumption, leading to better energy-efficiency numbers (Table IV).

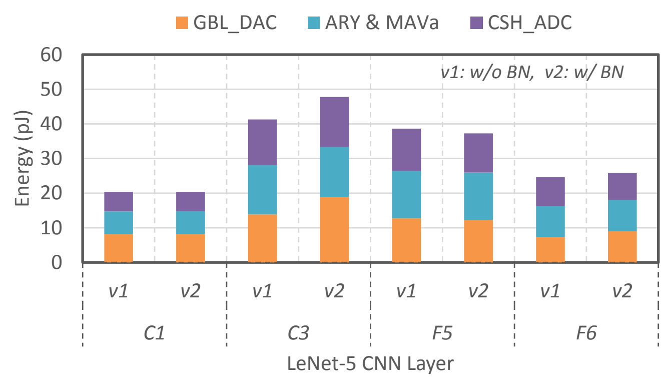

Fig. 24. Measured energy consumption of the CSRAM array when running the 4 different CONV/FC layers of the LeNet-5 CNN, with $V_{dd,DAC}$ = 1.2 V, $V_{dd,ARY}$ = 0.8 V, V_{dd} (rest) = 1 V and $f_{clk,main}$ = 5 MHz.

TABLE III
MEASURED ENERGY-EFFICIENCY* (TOPS/W) FOR THE CONV/FC LAYERS OF LENET-5 CNN, AT V_{dd} = 1V

Type (↓)	**C1**	**C3**	**F5**	**F6**
v1: without BN	14.8	38.8	38.8	24.3
v2: with BN	14.7	33.5	40.3	23.2

* 1 MAV = 1 multiply + 1 average = 2 OPs, with 6-b inputs and 1-b weights

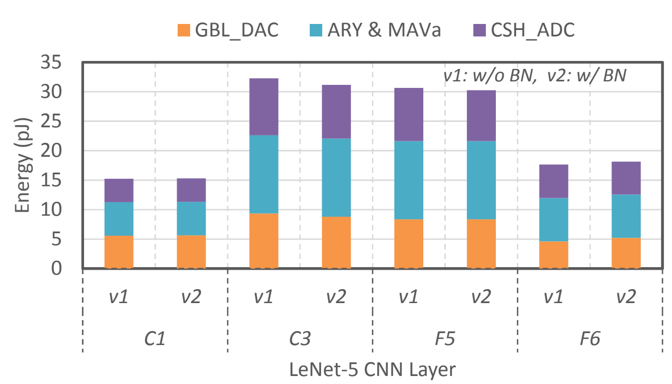

Fig. 25. Measured energy consumption of the CSRAM array when running the 4 different CONV/FC layers of the LeNet-5 CNN, with $V_{dd,DAC}$ = 1 V, V_{dd} (rest) = 0.8 V and $f_{clk,main}$ = 2.5 MHz.

Recent hardware implementations [5], [9], [16], [19]–[21] for NNs have focused on reduced bit-precisions to achieve higher energy-efficiency. Table V presents comparison with

TABLE V
COMPARISON WITH PRIOR WORK ON LOW BIT-WIDTH HARDWARE IMPLEMENTATIONS OF ML ALGORITHMS

Metric	This work	ISSCC'17 [9]	JSSC'17 [5]	JSSC'17 [13]	JSSC'18 [14]	JSSC'18 [16]	CICC'18 [19]	ISSCC'18 [20]
Tech. (nm)	65	28	40	130	65	65	28	65
Voltage (V)	1	0.715	0.8	-	1	0.55	0.66	0.65
Computation Mode	In-Memory, mixed-signal	Digital	Digital	In-Memory, mixed-signal	In-Memory, mixed-signal	Near-Memory, digital	Digital	Digital
ML Algo.	CNN (4-layer)	FC-DNN (4-layer)	CNN (5-layer)	SVM[1]	k-NN[2]	FC-DNN (12-layer)	CNN (5-layer)	CNN
ML Dataset	MNIST	MNIST	MNIST	MNIST	MNIST	MNIST	MNIST	FER2013
# MAC(V)'s per classification	406.8K	334.3K	406.8K	3.65K[1]	16.4K	768.1K	15M	-
Classification Accuracy	98.3% (1V) 98% (0.8V)	98.36%	98%	90%	92%[2]	90.1%	97.4%	-
# bits for IFMP/OFMP	6	8	6	5/1	8	2	1	16
# bits for Weights	1	8	4	1	8	1	1	1
SRAM Size (KB)	2	1024	144	2	16	102.1	328	256
Peak Throughput (GOPS)[3]	8 (1V) 4 (0.8V)	10.7	102	57.6[1]	10.2	380.2	90	368.6
Peak Energy Efficiency (TOPS/W)	40.3[4] (1V), 51.3[4] (0.8V)	1.86 (0.345)[5]	1.75 (0.663)[5]	11.51[1] (46.0)[5]	1.94	6.0	130 (24.12)[5]	50.6

[1] SVM: Support Vector Machine with 45 binary classifiers, each with 81 inputs, i.e. 81×45 MACs per 10-way classification
[2] k-NN: k-Nearest Neighbor, only 4-output classes (out of 10) were demonstrated with 100 test images
[3] We assume 2 operations (OPs) for 1 MAV (1 mult. + 1 avg.), similar to a MAC (1 mult. + 1 acc.)
[4] Does not include energy to access IFMP/OFMP memories
[5] Assuming a 65-nm implementation and Energy $\propto$ (Tech.)^2

TABLE IV
MEASURED ENERGY-EFFICIENCY* (TOPS/W) FOR THE CONV/FC LAYERS OF LENET-5 CNN, AT V_{dd} = 0.8V

Type (↓)	C1	C3	F5	F6
v1: without BN	19.7	49.6	48.9	17.0
v2: with BN	19.6	51.3	49.6	16.5

* 1 MAV = 1 multiply + 1 average = 2 OPs, with 6-b inputs and 1-b weights

prior work, both conventional digital [5], [9], [19], [20] and in-memory approaches [14], [16]. It should be noted that, while [5], [9], [16], [19], [20] are full systems, the main focus of this work was to demonstrate in-memory computation capability for CNNs. Hence, ours does not include the energy for IFMP/OFMP memories. However, for the MNIST dataset with LeNet-5 CNN, we estimate [22] those to have only small contributions to the overall energy-efficiency per MAV operation, due to the high parallelism supported by our in-memory approach. Furthermore, as seen from Fig.s 23 and 22, both the inputs (X_{IN}'s) and the partial outputs (Y_{OUT}'s) have a high proportion of '0's. Hence, in future work, data-dependent memory architectures e.g. 8T SRAMs, [23], [24] can be used to store and access the inputs/outputs. [23], [24] take advantage of data properties to significantly reduce memory-access energy, which would be highly useful here. Compared to [5], [9], we achieve $> 27\times$ improvement in energy-efficiency, due to the massively parallel in-memory analog computations. Our work achieves similar energy-efficiency numbers as [19] (considering a simplified technology scaling model), while using 6-b for IFMP/OFMP, compared to 1-b in [19]. Whereas, we achieve similar classification accuracy as [19] on MNIST, using $\sim 37\times$ less MAC/MAV operations per classification. Our numbers are also comparable to the energy-efficiency of [20] (not quoted for MNIST), which uses 1-b for weights. Next, we compare our results to an in-memory mixed-signal computing approach [13], which implements a support vector machine (SVM) algorithm with 45 binary-classifiers, for a 10-way MNIST digit classification. We achieve similar energy-efficiency as [13], while improving the classification accuracy from 90% to $> 98\%$. This is because, we mitigate the problem of degraded computation accuracy, caused by SRAM bit-cell variation. Although [13] can run at a higher speed, it only supports 1-b output resolution, compared to 6-b in our case. When compared to a near-memory approach [16], which uses only 1-b for IFMP/OFMP, we still achieve $8.5\times$ improvement in energy-efficiency. This is because, our approach exploits high parallelism of accessing multiple memory addresses simultaneously, without the need to sequentially and explicitly read out the data (filter weights) from the memory. We also achieve a higher classification accuracy compared to [16], because of using 6-b for inputs/outputs. Finally, our work also achieves better classification accuracy than prior in-memory approach [14], although it supports 8-b weights. This is because we reduce the effect of bit-cell variation when evaluating the weights. In addition, our approach benefits from more parallelism, by supporting 16 different dot-product computations per array per cycle, compared to 1 for [14].

VII. Conclusions

This paper presents an SRAM-embedded convolution (dot-product) computation architecture for running binary-weight neural networks. We demonstrated functionality with the LeNet-5 CNN on the MNIST hand-written digit recognition dataset, achieving classification accuracy close to digital implementations and much better than prior in-memory approaches. This is made possible by our variation-tolerant architecture and also the support of multi-bit resolutions of input/output values. Compared to conventional digital accelerator approaches using small bit-widths, we achieve similar or better energy-efficiency, by overcoming some of the major limitations of memories in traditional computing paradigms. This is because our architecture can significantly reduce data transfer by running massively parallel analog computations inside the memory. The results indicate that the proposed energy-efficient, SRAM-embedded dot-product computation architecture could enable low power ML applications (e.g. "always-ON" sensing) for "smart" devices in the Internet-of-Everything.

Acknowledgment

This project was funded by Intel Corporation. The authors thank Vivienne Sze and Hae-Seung Lee for helpful technical discussions.

References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1701–1708.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[4] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[5] B. Moons and M. Verhelst, "An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, April 2017.

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *ArXiv e-prints*, Mar. 2016.

[7] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 3123–3131.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 4107–4115.

[9] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G. Y. Wei, "A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with $>$0.1 timing error rate tolerance for IoT applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 242–243.

[10] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[11] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[12] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 488–490.

[13] J. Zhang, Z. Wang, and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, April 2017.

[14] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, Feb 2018.

[15] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip training," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 490–492.

[16] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "BRein Memory: A Single-Chip Binary/Ternary Reconfigurable in-Memory Deep Neural Network Accelerator Achieving 1.4 TOPS at 0.6 W," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, April 2018.

[17] W. Khwa, J. Chen, J. Li, X. Si, E. Yang, X. Sun, R. Liu, P. Chen, Q. Li, S. Yu, and M. Chang, "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3ns and 55.8TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 496–498.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[19] B. Moons, D. Bankman, L. Yang, and B. M. . M. Verhelst, "BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28nm CMOS," in *2018 IEEE Custom Integrated Circuits Conference - (CICC)*, April 2018.

[20] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 218–220.

[21] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 246–247.

[22] A. Biswas, "Energy-Efficient Smart Embedded Memory Design for IoT and AI," Ph.D. dissertation, Massachusetts Institute of Technology, June 2018.

[23] A. Biswas and A. P. Chandrakasan, "A 0.36V 128Kb 6T SRAM with energy-efficient dynamic body-biasing and output data prediction in 28nm FDSOI," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, Sept 2016, pp. 433–436.

[24] C. Duan, A. J. Gotterba, M. E. Sinangil, and A. P. Chandrakasan, "Energy-Efficient Reconfigurable SRAM: Reducing Read Power Through Data Statistics," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 10, pp. 2703–2711, Oct 2017.

Avishek Biswas (S'12 - M'18) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology (IIT), Kharagpur, India in 2012, and the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2014 and 2018 respectively.

Since June 2018, he has been with Kilby Labs, Texas Instruments Incorporated, Dallas, TX, USA. His research interests include circuit and algorithm co-design for low power neural networks, specifically focussing on in-memory computation architectures. Dr. Biswas has done internships with Intel Corporation, Hillsboro, Oregon, USA and also with CEA Leti, Grenoble, France, working on low-power, application-specific memory design.

Dr. Biswas was the recipient of the President of India Gold Medal from IIT Kharagpur in 2012 and the Merrill Lynch Fellowship from MIT in 2012.

Anantha P. Chandrakasan (M'95 - SM'01 - F'04) received the B.S, M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1989, 1990, and 1994 respectively. Since September 1994, he has been with the Massachusetts Institute of Technology, Cambridge, where he is currently the Joseph F. and Nancy P. Keithley Professor of Electrical Engineering.

He was a co-recipient of several awards including the 1993 IEEE Communications Society's Best Tutorial Paper Award, the IEEE Electron Devices Society's 1997 Paul Rappaport Award for the Best Paper in an EDS publication during 1997, the 1999 DAC Design Contest Award, the 2004 DAC/ISSCC Student Design Contest Award, the 2007 ISSCC Beatrice Winner Award for Editorial Excellence and the ISSCC Jack Kilby Award for Outstanding Student Paper (2007, 2008, 2009). He received the 2009 Semiconductor Industry Association (SIA) University Researcher Award. He is the recipient of the 2013 IEEE Donald O. Pederson Award in Solid-State Circuits.

His research interests include micro-power digital and mixed-signal integrated circuit design, wireless microsensor system design, portable multimedia devices, energy efficient radios and emerging technologies. He is a co-author of Low Power Digital CMOS Design (Kluwer Academic Publishers, 1995), Digital Integrated Circuits (Pearson Prentice-Hall, 2003, 2nd edition), and Sub-threshold Design for Ultra-Low Power Systems (Springer 2006). He is also a co-editor of Low Power CMOS Design (IEEE Press, 1998), Design of High-Performance Microprocessor Circuits (IEEE Press, 2000), and Leakage in Nanometer CMOS Technologies (Springer, 2005).

He has served as a technical program co-chair for the 1997 International Symposium on Low Power Electronics and Design (ISLPED), VLSI Design '98, and the 1998 IEEE Workshop on Signal Processing Systems. He was the Signal Processing Sub-committee Chair for ISSCC 1999-2001, the Program Vice-Chair for ISSCC 2002, the Program Chair for ISSCC 2003, the Technology Directions Sub-committee Chair for ISSCC 2004-2009, and the Conference Chair for ISSCC 2010-2014. He is the Conference Chair for ISSCC 2015. He was an Associate Editor for the IEEE Journal of Solid-State Circuits from 1998 to 2001. He served on SSCS AdCom from 2000 to 2007 and he was the meetings committee chair from 2004 to 2007. He was the Director of the MIT Microsystems Technology Laboratories from 2006 to 2011. Since July 2011, he is the Head of the MIT EECS Department.