A Flash-based Current-mode IC to Realize Quantized Neural Networks

1st Kyler R. Scott
Department of ECE
Texas A&M University
College Station, USA
kylerrscott@tamu.edu

2nd Cheng-Yen Lee
Department of ECE
Texas A&M University
College Station, USA
cylee@tamu.edu

3rd Sunil P. Khatri
Department of ECE
Texas A&M University
College Station, USA
sunilkhatri@tamu.edu

4th Sarma Vrudhula
Department of ECE
Arizona State University
Tuscon, USA
vrudhula@asu.edu

***Abstract*—This paper presents a mixed-signal architecture for implementing Quantized Neural Networks (QNNs) using flash transistors to achieve extremely high throughput with extremely low power, energy and memory requirements. Its low resource consumption makes our design especially suited for use in edge devices. The network weights are stored in-memory using flash transistors, and nodes perform operations in the analog current domain. Our design can be programmed with any QNN whose hyperparameters (the number of layers, filters, or filter size, etc) do not exceed the maximum provisioned. Once the flash devices are programmed with a trained model and the IC is given an input, our architecture performs inference with zero access to off-chip memory. We demonstrate the robustness of our design under current-mode non-linearities arising from process and voltage variations. We test validation accuracy on the ImageNet dataset, and show that our IC suffers only 0.6% and 1.0% reduction in classification accuracy for Top-1 and Top-5 outputs, respectively. Our implementation results in a $\sim$50$\times$ reduction in latency and energy when compared to a recently published mixed-signal ASIC implementation, with similar power characteristics. Our approach provides layer partitioning and node sharing possibilities, which allow us to trade off latency, power, and area amongst each other.**

***Index Terms*—Quantized Neural Networks, Floating-gate Transistors, Current-mode Circuits**

I. Introduction and Previous Work

Deep Neural Networks (DNNs) have demonstrated state-of-the-art performance in a wide range of tasks including image classification [1], object detection [2], and text recognition [3]. Modern DNNs have significant power and memory requirements, making them unsuitable for use in edge computing contexts such as mobile and embedded devices. Consequently, there are many recent works that attempt to reduce the resource consumption of DNNs using various strategies like network pruning [15], [16], compact network designs [17], [18], and low-bit quantization [19], [20]. Quantization remains the most effective technique. It involves reducing the representation of network weights and activations to a small number of bits, yielding a Quantized Neural Network (QNN). A Binarized Neural Network (BNN) [4] is the extreme case of quantization, where network weights and activations are binary values. BNNs can achieve up to a 58x speedup and a 32x reduction in memory bandwidth requirements [19] on CPUs versus full-precision networks.

Because most BNN computation consists of binary operations, FPGAs are a good platform for implementing BNNs. The work of [21] presents an FPGA framework for BNN implementation that achieves high inference throughput on small images, using less than 25W total system power. The FPGA-based accelerator LP-BNN [24] achieves ultra-low-latency inference of the ImageNet dataset [32] on an FPGA by optimizing network structure. However, this approach must sacrifice some functionality such as batch normalization. Unlike our implementation which uses flash transistors, the resource consumption of any FGPA implementation is much higher and will scale as the bitwidth of network weights is increased.

ASICs or custom ICs are another promising platform for both analog and digital BNN implementation. In [22], the authors present a digtal ASIC for text interpretation. Although this work is proposed for use in edge devices, it cannot achieve energy efficiencies comparable to our approach, and like FPGA implementations, the resource consumption will scale with the bitwidth of weights. The work of [23] proposed an analog accelerator that uses a resistive RAM (RRAM) non-volatile memory (NVM) array. Unlike our architecture, NVM arrays can be problematic due to sneak currents [35], which require additional hardware to mitigate. Our novel architecture avoids the problem of sneak currents by using only one transistor per stack. Also, since we apply input voltages only to transistor gates, there is no opportunity for current flow in a direction other than that which is intended.

TULIP [14] is a recent mixed-signal CMOS ASIC for realizing BNNs, designed with the goal of maximizing energy efficiency. It consists of PEs that implement threshold logic functions to perform the necessary computations of a BNN. We use TULIP as a reference for comparison of performance metrics such as latency, area, and power, since TULIP had the best performance at the time it was published. We show that with comparable power consumption, our implementation requires about 50$\times$ lower energy and latency compared to TULIP, when performing inference on ImageNet.

In this work, we propose a novel mixed-signal architecture for realizing QNNs, with the goal of minimizing power, energy, and latency. We report an IC which can be programmed with any QNN whose hyperparameters do not exceed the maximum provisioned in our IC. Our design uses flash transistors to store network weight parameters and performs most computations in the analog current domain.

We summarize our key contributions below.

- We introduce a novel single-chip solution for implementing QNNs on edge devices, based on flash technology, that results in extremely low power, latency, and energy, with minimal cost to QNN classification accuracy.
- We use flash transistors to perform operations in-memory, significantly reducing both memory bandwidth require-

978-3-9819263-6-1/DATE22/©2022 EDAA

ments, as well as energy. Unlike other NVM technologies such as RRAM, flash is a well understood, high yielding technology, making it a practical approach.
- Unlike existing mixed-signal approaches that use 2D arrays of transistors and are subject to sneak currents, our approach uses a single flash transistor per stack, thereby obviating sneak currents and resulting in a tight control of current.
- Additionally, our design uses a novel circuit architecture to support activation values in {-1, 1}. Further, we have conducted a thorough Monte Carlo analysis of the variability of our output current, and based on this, we show that the classification accuracy is minimally degraded.

The rest of this paper is organized as follows. Section II presents a brief background on CNNs and flash technology. Section III presents the design for all our CNN components, and Section IV reports results of experiments we conduct on classification accuracy, power, energy, area, latency, and robustness. Section V presents our conclusion.

II. Background

1) Convolutional Neural Networks: Convolutional Neural Networks (CNNs) have achieved great success for a variety of tasks, most notably image and video processing. A representative CNN architecture is shown in Fig. 1b. CNNs are built partially from convolutional (CONV) layers, whose nodes implement a 3D filter W, sometimes referred to as a kernel. CONV filters will scan over a 3D input matrix, called an Input Feature Map (IFM), X. At each position of W over the IFM, a convolution operation is performed, yielding the result $W \circledast X$. At each position of W over the IFM, this operation computes $\sum w_i x_i$, where $w_i \in W$ and $x_i \in X$, and the indices i are over the unrolled matrices W and X. In CNNs, node outputs are passed through an activation function. In BNNs, where activations are binary, the activation function is a step function, yielding $+1$ if $\sum w_i x_i > 0$, and -1 otherwise. The outputs of a CONV layer form a 3D matrix of outputs called the Output Feature Map (OFM). Following CONV layers, the OFMs are fed to one or more fully-connected layers, whose nodes compute inner products followed by an activation step. A detailed description of CNNs may be found in [26].

2) Flash Transistors: A flash device is a field effect transistor (FET) with two gates: a control gate (like in CMOS transistors), and an uncontacted *floating gate* which can store electric charge, as shown in Fig. 1a. The charge on the floating gate can be altered with a programming scheme [5], in order to change the threshold voltage (V_{TH}) of the flash transistor. Historically, flash transistors have only been used for non-volatile memory (NVM) technologies, but recent results demonstrate their promise for use in digital design, due to their programmable V_{TH} [27].

3) BNN Architectures: In this paper we focus on two BNN architectures: Binary AlexNet and BinaryNet. Binary AlexNet is a binarized form of the AlexNet architecture [25], implemented by the Larq Python library [13], and trained with the methods proposed in [4]. It is used for inference of ImageNet, and is depicted in Fig. 1b. BinaryNet is an architecture used for inference of the Cifar-10 dataset [33]. We implement BinaryNet as in [14].

(a) Cross Section of a Flash Transistor

(b) Binary AlexNet Architecture

Fig. 1: Flash Transistor and Binary AlexNet Architecture

III. Implementation

A. Overview

In our IC, we implement all aspects of the CNN needed for inference. Convolutional (CONV), max pooling (MAXPOOL) and fully-connected (FC) layers, batch normalization, intermediate data storage, and control flow are all implemented on-chip. At programming time, all network hyperparameters can be chosen up to the maximum allowed. The maximum values of these hyperparameters are summarized in Fig. 2. Note that not all layer types need be used, resulting in a highly flexible architecture.

Each layer is comprised of circuits implementing the nodes of that layer. The nodes in the first CONV layer accept 8-bit inputs. Nodes in all other layers accept 1-bit inputs in {-1, 1}. The nodes in all layers can be programmed with any weight programming scheme from binary up to 9-valued, at zero additional cost to delay, area, or power. All nodes have 1-bit outputs in {-1, 1}.

In the rest of this section, we describe our implementation of every aspect of a CNN. First, we discuss the designs of the FC node (Section III-B), CONV node (Section III-C), and MAXPOOL node (Section III-D). Then we describe how batch normalization is implemented (Section III-E). Finally, we describe our dataflow architecture (Section III-F).

B. Fully-Connected (FC) Node Design

1) Input Network (IN): The FC node accepts inputs through an Input Network, consisting of two halves. We refer to these halves as the Left Input Network (LIN) and the Right Input Network (RIN), as shown in Fig. 3a. Each branch in the IN is made from one flash FET and two MOSFETs, as shown in Fig. 3b. For IN branch i, the flash FET threshold voltage is programmed to either 0.862V, 0.908V, 0.962V, 1.037V, or 2V according to the magnitude of weight parameter w_i such that the $I_{DS} \propto |w_i|$ when the FET is conducting. The LIN (RIN) stores positive (negative) weight parameters. For a given weight w_i, $sign(w_i)$ will determine whether it is programmed to branch i of the LIN or RIN, yielding 9 weights in total. The programming of these branches is mutually exclusive; if branch i of the LIN (RIN) is programmed with a weight w_i, branch i of the RIN (LIN) will be programmed with a weight of zero.

The FC node accepts a digital voltage input x_i and its complement $\overline{x_i}$. These signals are connected as shown in Fig. 3b, which can encode an input in {-1, 1} as follows. Recall that if weight $w_i > 0$, it will be programmed to the LIN. Now, if $x_i = 1$ ($x_i = 0$), the branch current will flow to the node I_{IN+} (I_{IN-}). Similarly, recall that if weight $w_i < 0$, it

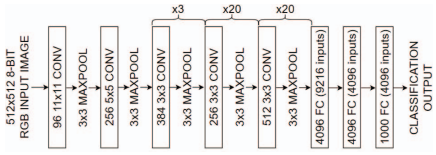
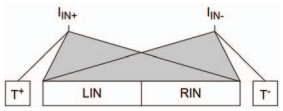
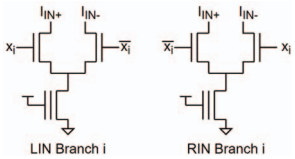


Fig. 2: Maximum Architecture Provisioned On-Chip



(a) IN and Threshold Blocks



(b) IN Branch Design and Connectivity

Fig. 3: FC and CONV Node Input Network (IN)

will be programmed to the RIN. Then, if $x_i = 1$ ($x_i = 0$), the branch current will flow to I_{IN-} (I_{IN+}). This effectively computes $w_i x_i$ in the current domain, for a given branch i, with 9 possible weight values.

At I_{IN+} and I_{IN-}, all branch currents are summed by Kirchoff's Current Law. The current flowing to I_{IN+} (I_{IN-}) will be $\sum w_i x_i$ over all i for which $w_i x_i$ is positive (negative). Both the LIN and the RIN have a number of input branches equal to the maximum number of node inputs. To realize a node with a smaller number of inputs than the maximum, the unused branches will be programmed with zero weight.

Each LIN/RIN branch pair can be programmed with any 9-valued weight parameter. Our experiments prove reliable node performance for weights with up to and including 9 values. However, it should be noted that we can use a larger number of weights as well. It is known that one can choose flash V_{TH} with very fine precision ([36], [37]), and hence we could have hundreds of unique weight values within a typical V_{TH} range (a few hundred mV). Importantly, this choice of weights comes at no memory cost, because the weight value is stored in the flash FET alone. That said, our paper uses 9 weight values as described above.

2) Current Mirrors and Comparator: Each node uses two current mirrors to convert the differential currents from the nodes I_{IN+} and I_{IN-} to a common node V_{CMP}, as shown in Fig. 4. On the positive (Left) side of the figure, the current I_{IN+} is transferred to V_{CMP} through a current mirror connected to V_{DD}, so that an increase in this current causes V_{CMP} to increase. On the negative (Right) side of the figure, we use a two-stage current mirror, with the second stage connected to GND, so that an increase in current I_{IN-} causes V_{CMP} to decrease. The left and right current mirrors have identical scaling factors, and hence $V_{CMP} > V_{DD}/2$ when $I_{IN+} > I_{IN-}$, and $V_{CMP} < V_{DD}/2$ otherwise. Thus, $V_{CMP} > V_{DD}/2$ if $\sum w_i x_i > 0$ and $V_{CMP} < V_{DD}/2$ otherwise.

The comparator (CMP) compares V_{CMP} against $V_{REF} = V_{DD}/2$, to produce the node output bit. This results in an output bit of 1 when $\sum w_i x_i > 0$, and zero otherwise. The output bit encodes a value in {-1, 1}.

3) Analog-Digital Converter: For FC nodes that produce a 1-bit output, only the CMP is needed. However, the final FC layer must produce integer outputs, because the outputs will

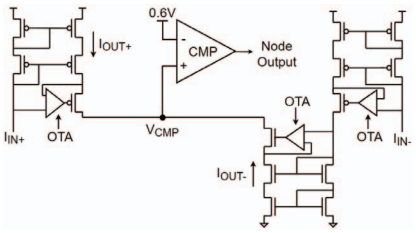


Fig. 4: Node Current Mirrors and Comparator

be used to choose the final classification result of the network. Therefore, for the FC nodes in the final layer, we use an 8-bit SAR ADC [28] to produce the output. The ADC consists of a sample-and-hold circuit (S/H), a digital-analog converter (DAC), a comparator (CMP), and successive approximation registers (SAR). Our ADC is not shown for brevity, since it uses a traditional SAR ADC design.

C. Convolution (CONV) Node Design

1) Input Network: The input network (IN) of CONV nodes is exactly the same as in FC nodes. This is because the convolution operation is also a summation $\sum w_i x_i$, over a portion of a 3D feature map. To realize a CONV node with a $k \times k$ kernel and c channels, the node will have $k^2 c$ inputs. In our IC, each CONV node will implement a different filter. Our design can implement different numbers of filters, with different filter sizes as shown in Fig. 2.

The IN design is different for the first CONV layer however, because this layer accepts 8-bit image inputs with three (RGB) channels. We accomplish this by duplicating the IN block into 8 blocks IN_0-IN_7.

2) Current Mirrors and Comparator: In a CONV node with 1-bit inputs, the current mirror design is identical to that in FC nodes. For a CONV node with 8-bit inputs, the current through each block IN_j must be given twice the magnitude of IN_{j-1}, so that the bits of x_i are given their proper significance. To accomplish this, we duplicate current mirrors and design them to have binary-weighted scaling factors.

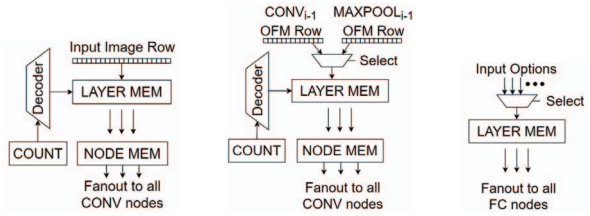
3) ADC: There is usually no need for an ADC in CONV nodes, because they only ever produce 1-bit outputs in a typical QNN. However, if the user wants a network with only CONV layers and an 8-bit output, this is possible to realize, by programming identity filters to the final FC layer, such that the layer simply transfers the CONV layer outputs to the ADC.

D. MAXPOOL

We implement MAXPOOL as an OR operation on binary values in a feature map. We use 9-input OR gates to accomplish MAXPOOL of a 3x3 kernel or smaller.

E. Batch Normalization

We implement batch normalization as described in [6], by changing the threshold T used in the node output computation $\sum w_i x_i > T$. We do this with threshold blocks T^+ and T^-, as depicted in Fig. 3a. T^+ and T^- are implemented with flash transistors (as in the node IN branches in Fig. 3b), but without any MOSFETs. Setting the V_{TH} of the FETs in T^+ and T^- allows for setting the node threshold to $-Max \leq T \leq Max$, where Max is the maximum $\sum w_i x_i$ value of that node.



(a) First CONV Layer (b) Other CONV Layers (c) FC Layers

Fig. 5: Node Input Multiplexing for Different Layers

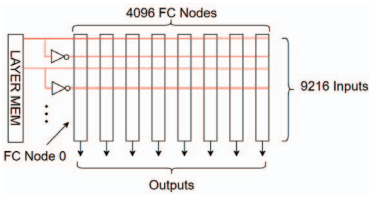


Fig. 6: FC Layer Interconnect

F. Dataflow Architecture

We use a dataflow architecture to pipeline data through our chip. Our architecture allows for a variable number of layers, for each layer type, to be programmed onto the chip. We supply the input image to the CNN in a row-by-row manner. Each layer produces its output feature map (OFM) pixel-by-pixel. This allows all CONV layers to compute in parallel, because each layer can begin computing once it has enough inputs for a single output pixel to be produced. Depending on the choice of the user, some number of layers may be disabled. We route data around unused layers, giving the user a flexible choice for the number of CONV, MAXPOOL, and FC layers, up to the maximum for each (see Fig. 2). Because layers can be disabled, some layers can receive their data from different source layers, as listed here.

- The first CONV layer can only receive the input image
- MAXPOOL layers may only receive inputs from the preceding CONV layer
- All CONV layers after the first may receive inputs from the preceding MAXPOOL layer. If that layer is disabled, they will receive inputs from the preceding CONV layer.
- All FC layers may receive inputs from any preceding layer

These input options allow for a variable number of layers, but do not allow layer types to be arranged in arbitrary order. For example, FC layers may not be programmed to come before CONV layers. In this section, we describe how these different layer input options are realized by multiplexors, and how convolution is accomplished.

1) The first CONV layer: As depicted in Fig. 5a, we supply the input image into the first CONV layer's input memory (LAYER MEM) on a row-by-row basis. Once we have supplied a number of rows equal to the chosen filter width, the decoder feeds data column-by-column to the node memory block (NODE MEM). After NODE MEM has received enough inputs, the nodes in this layer produce an output, one-by-one. Next, the decoder will supply more data to NODE MEM, and the nodes produce more outputs. This emulates the sliding of the CONV filter from left to right over the top of the input image as columns of LAYER MEM are supplied to NODE MEM, and will continue until the first row of the OFM has been produced. After this, a new row of the input image is fed into LAYER MEM. This emulates the kernel moving down over the image, so it can produce the next row of the OFM. All memory is implemented with shift registers, to enable the "sliding" behavior. The outputs of the first CONV layer are stored in LAYER MEM of the next layer as they are produced.

2) MAXPOOL Layers: MAXPOOL Layer receive inputs and produce outputs identically to the first CONV layer.

3) Other CONV Layers: All CONV layers after the first have two options from where they can receive inputs. These two sources are multiplexed into LAYER MEM, enabling a choice between the two. This is shown in Fig. 5b.

4) FC Layers: Each FC layer can receive inputs from any previous layer. This is implemented using a 64:1 MUX feeding LAYER MEM, as depicted in Fig. 5c. Note that because FC nodes do not have a sliding kernel and operate on all inputs at once, the node inputs are wired directly to LAYER MEM.

Our FC layers have a very large number of inputs (up to 9216), connected to the input branches of up to 4096 nodes. This can pose a challenge for the interconnect between LAYER MEM and the nodes of the layer. We solve this challenge with a layout scheme that allows for simplified interconnect. The layout of the largest FC layer is depicted in Fig. 6. For every input x_i (and its complement $\overline{x_i}$), We use parallel wires which run horizontally over the inputs of each node of the FC layer. This results in $9216 * 2 = 18432$ horizontal wires. The outputs of the nodes of the FC layer are driven to the next FC layer in a vertical orientation, as shown in Fig. 6.

IV. Experiment and Results

We implement our designs in a 45nm process technology. We simulate circuit designs using Synopsys HSPICE [29], to test node correctness and measure performance metrics like latency, area, and power. All the CNN components described in Section III are accounted for in our computations. For CMOS devices, we use a 45nm PTM model card [30]. For flash devices, we use the model card derived in [27]. Our nominal $V_{DD} = 1.2V$ for all HSPICE experiments. We use the Python library TensorFlow [31] to measure the ImageNet classification accuracy of Binary AlexNet implemented with our approach.

We measure node output bit error rates in Section IV-A1, and node latency in Section IV-A2. The simulations for these sections are conducted in HSPICE. We then export node error rates to a Python simulation in Section IV-A3 to measure classification accuracy on ImageNet. In Section IV-B we quantify and compare metrics like layout area and inference latency, power, and energy, and we also discuss variations of our design that can allow us to trade off these metrics.

A. Error

1) Bit Error: This experiment tests node performance in the presence of process and voltage variation. We implement our largest node, 1-bit 9216-input FC node in HSPICE, and perform 18000 Monte Carlo simulations.

For each simulation, we choose a mean weight w_{mean} for that node, from the uniform distribution in Fig. 7 (Left). w_{mean}

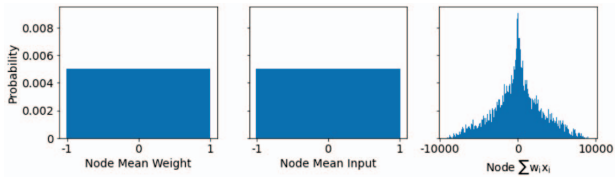
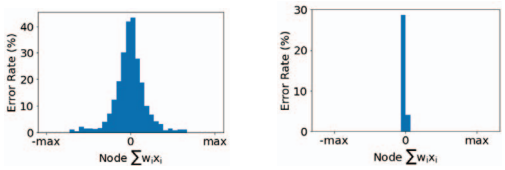


Fig. 7: Node mean weight, input, and $\sum w_i x_i$ distributions



(a) 9216-Input FC Bit Error (b) 1200-Input FC Latency Error

Fig. 8: FC Node Bit Error Rates

is the desired mean of all the node's weights, and corresponds to a probability for each weight value being chosen for any input branch. This allows us to test many possible states of the node. For instance, a node with a high w_{mean} may have different analog behavior than a node with a low w_{mean}, due to current mirror non-linearities. To ensure we sufficiently test the entire range of node weights, we sample each w_{mean} equally, with an increment of 0.01. We also select a mean input x_{mean} for each test, from the uniform distribution in Fig. 7 (Center). x_{mean} corresponds to a probability for each input value being chosen for any input branch. We sample each mean input equally, with an increment of 0.01.

In this experiment, we select node weights from {-1, 1}, i.e. 1-bit weights, so we can compare our ImageNet classification accuracy against that of a baseline BNN that also uses 1-bit weights in Section IV-A3. We note that performing this experiment with 9-valued weights did not result in a significant increase in error rates.

For the Monte Carlo simulations, we model process variation by randomizing the length (L), width (W), and threshold voltage (V_{TH}) of all MOSFETs. For L and W, we take the absolute value of the variation from [7], which was reported for a 65nm process. We use the V_{TH} variation presented in [8]. We derive our MOSFET average channel doping N_a from the average value of recent papers ([9]–[12]). For each simulation we also vary V_{DD}, with a 10% ($120mV$) standard deviation. Our CMP reference voltage V_{REF} is assumed to be constant, being generated by a precise bandgap voltage reference [34].

For each simulation, we simulate the node, and if the node output is incorrect we record this as a bit error for that $\sum w_i x_i$. We split the range of values for $\sum w_i x_i$ into bins of width 512. In Fig. 8a, we plot the node output error rates for each bin.

2) Latency Error: In our design, we bound the latency of a node, thereby incurring some error. We use our smallest, slowest nodes to measure latency and its associated error. For Binary AlexNet this is a 1200-input 1-bit FC node. For BinaryNet, this is a 27-input 8-bit CONV node. For both nodes, we perform 50000 Hspice transient simulations. Node weights and inputs are selected as in the Bit Error experiment.

For each simulation, we record the $\sum w_i x_i$ of the node, and the latency from the time the node inputs arrive to the time the node output bit reaches its final value. Just before the node inputs are applied, we precharge V_{CMP} to $V_{DD}/2$, using the CMP V_{REF}. We do not take the maximum latency as our clock period, but rather, we select the clock period $<$ maximum latency, and accept node output errors for situations when node latency exceeds the selected clock period. This gives additional error rates per bin, as shown in Fig. 8b. Our chosen clock period is 4.2ns for Binary AlexNet and 7.2ns for BinaryNet. The latency error rates for Binary AlexNet are shown in Fig. 8b. In general, a user can choose the clock period based on their tolerance to reduced classification accuracy.

	Binary AlexNet on ImageNet			
Setting / Metric	Variation 1	Variation 2	Variation 3	TULIP [14]
Max Power	6.47 mW	1.6 W	48.5 W	Unknown
Average Power	2.72 mW	614 mW	24.8 W	2.59 mW
Latency	3.07 ms	13.6 μs	336 ns	165 ms
Energy	8.35 μJ	8.35 μJ	8.35 μJ	428 μJ
Area	17.7 mm^2	17.7 mm^2	64.7 mm^2	1.80 mm^2
	BinaryNet on Cifar-10			
Setting / Metric	Variation 1	Variation 2	Variation 3	TULIP [14]
Max Power	4.59 mW	1.2 W	20.8 W	Unknown
Average Power	1.86 mW	433 mW	3.46 W	6.36 mW
Latency	1.99 ms	8.50 μs	1.07 μs	28.9 ms
Energy	3.71 μJ	3.71 μJ	3.71 μJ	184 μJ
Area	4.36 mm^2	4.36 mm^2	41.8 mm^2	1.80 mm^2

TABLE I: Chip Design Variations

3) CNN Classification Accuracy on ImageNet: This experiment will test how much the bit error rate (Section IV-A1) and latency error rate (Section IV-A2) impact classification accuracy of the CNN as a whole. We use the Binary AlexNet architecture, implemented in the Larq Python library [13]. We measure Top-1 and Top-5 validation accuracy on ImageNet. We model total node error behavior by summing the two histograms depicted in Fig. 8, thus adding the bit error and latency error rates. We apply error rates on a per-bin basis, to all nodes in the CNN. It should be noted that the two error histograms in Fig. 8 were measured for our most error-prone node and our slowest node respectively, and therefore give a conservative estimate for error rates of all nodes.

Compared to the digital implementation of Binary AlexNet, our mixed-signal implementation results in a minimal decrease in CNN classification accuracy. We measure a 0.6% dtop in Top-1 accuracy (from 36.3% to 35.7%) and a 1.0% drop in Top-5 accuracy (from 61.5% to 60.5%).

B. Latency, Power, and Layout Area

In this section we present results on performance metrics of latency, area, power, and energy of a single inference for the Binary AlexNet and BinaryNet architectures implemented with our approach. We compare the performance metrics of our approach against TULIP [14], a recent BNN implementation, which had the best performance at the time it was published. TULIP is implemented in a TSMC 40nm LP process. In our results, we include the contribution to latency, layout area, power, and energy of all the component circuits described in Section III (including digital and memory blocks). We note that TULIP does not report measurements for Binary AlexNet, but for a slightly different architecture, proposed in [19]. These two architectures are both binarized versions of AlexNet and are very similar, and therefore provide a fair comparison. We list TULIP entries under the Binary AlexNet heading in Table

I for simplicity. We implement BinaryNet as implemented in [14], for a direct comparison.

1) Architecture Modifications: As described in Section III-F, for each CONV filter, we pass different values over the node inputs to emulate the sliding of the CONV kernel over the IFM over time. We call this modification *Node Sharing*, and use it to achieve lower power and area requirements. In order to further reduce our power consumption, we only allow one node per layer to fire in any given cycle. We call this modification *Layer Partitioning*. For both CONV and FC layers, this means that only a single OFM pixel per layer will be computed in any given clock cycle. We present three combinations of these modifications while presenting performance metrics, in Table I. Variant 3 uses only CONV node sharing. Variant 2, in addition, uses FC layer partitioning. Variant 1, in addition to the modifications of Variant 2, uses CONV layer partitioning.

2) Latency and Power: We perform HSPICE simulations to measure the latency and maximum active power of all circuits described in Section III. To compute inference latency, power, and energy, we perform an architecture-level simulation using Python. For Binary AlexNet and BinaryNet, we use the clock periods determined in Section IV-A2. The inference latency, power, and energy are shown in Table I.

3) Layout Area: We estimate chip layout area. We first generate a layout design for all IN blocks, based on Fig. 6. For the digital blocks, we estimate their layout area using Synopsys Design Compiler [29]. The chip area used by our architectures adds the layout area of the component circuits and is shown in Table I. If we provision the "maximum" network of Fig. 2, our area is $27.1mm^2$, with a maximum power of 91mW per classification.

V. Conclusion

In this paper, we present a novel flash-based mixed-signal architecture for realizing QNNs, with the goal of minimizing power, energy, and latency. We can realize any architecture whose hyperparameters are provisioned by our design. We utilize flash technology for network weight storage and node computation. This enables us to achieve low-latency inference with very low power and energy requirements, and allows for storage of many weight bits in a single transistor. The number of weight values used in our architecture could be increased past the 9-valued weights we test in this paper, at no additional cost to latency, area, power, or memory.

We implement all our designs to quantify classification error, as well as performance metrics like latency, area, power, and energy. As a part of these experiments, we perform a Monte Carlo analysis of our designs to test robustness in the presence of process and voltage variation. We model the performance of our chip as applied to the Binary AlexNet architecture and demonstrate that ImageNet classification accuracy is minimally degraded (by $\sim$1%) across circuit variations. We show that our chip design has $\sim$50$\times$ lower latency and energy than a recent ASIC approach, for ImageNet classification. Our architecture has several variants to enable the tradeoff between latency, area, and power.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," In *NIPS'2012*, 2012.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[3] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Deep features for text spotting," In *Computer Vision - ECCV 2014*, Cham, 2014. Springer International Publishing.

[4] I. Hubara et al., "Binarized neural networks," In *Advances in neural information processing systems*, 2016.

[5] F. M. Bayat et al., "Model-based high-precision tuning of NOR flash memory cells for analog computing applications," *2016 74th Annual Device Research Conference (DRC)*, 2016, pp. 1-2, doi: 10.1109/DRC.2016.7548449.

[6] T. Simmons and D. Lee, "A review of binarized neural networks," In *Electronics*, 8(6):661, 2019.

[7] W. Zhao et al., "Rigorous extraction of process variations for 65nm CMOS design," In *Proc. of the European Solid State Device Research Conf.*, Sept. 2007.

[8] K. Bernstein et al., D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," In *IBM J. Res. & Dev*. 50, No. 4/5, 433-449, 2006.

[9] F. A. M. Rezali et al., "Scaling impact on design performance metric of sub-micron CMOS devices incorporated with halo," *2015 IEEE Regional Symposium on Micro and Nanoelectronics (RSM)*, 2015, pp. 1-4.

[10] N. Ali et al., "TCAD analysis of variation in channel doping concentration on 45nm Double-Gate MOSFET parameters," *2015 Annual IEEE India Conference (INDICON)*, New Delhi, 2015, pp. 1-6.

[11] P. Lemoigne, V. Quenette, A. Juge, and D. Rideau, "Monitoring variability of channel doping profile in the 45nm node MOSFET through reverse engineering of electrical back-bias effect," *2009 Proceedings of the European Solid State Device Research Conference*, 2009, pp. 383-386.

[12] H. M. Nayfeh et al., "Impact of lateral asymmetric channel doping on 45-nm-technology N-type SOI MOSFETs," In *IEEE Trans. Electronic Devices*, vol. 56, pp. 3097-3105, Dec 2009.

[13] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *J. Open Source Softw.*, vol. 5, no. 45, Jan. 2020, Art. no. 1746.

[14] A. Wagle, S. Khatri and S. Vrudhula, "A Configurable BNN ASIC using a Network of Programmable Threshold Logic Standard Cells," *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 433-440.

[15] S. Han, J. Pool, J. Tran, and W.J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *NIPS*, 2015.

[16] S. Han, H. Mao, and W.J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding," in *International Conference on Learning Representations*, 2016.

[17] A. Howard et al., "Searching for MobileNetV3," in *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[18] M. Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classifcation Using Binary Convolutional Neural Networks.," in *European Conference on Computer Vision*, pp. 525-542. Springer, 2016

[20] S. Zhou et al., "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," arXiv preprint arXiv:1606.06160, 2016

[21] Y. Umuroglu et al., "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* FPGA '17, pages 65–74, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Y. Li et al., "A 34-FPS 698-GOP/s/W Binarized Deep Neural Network-Based Natural Scene Text Interpretation Accelerator for Mobile Edge Computing," *IEEE Transactions on Industrial Electronics,* 66(9):7407–7416, 2019.

[23] X. Sun et al., "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)* pages 1423–1428, 2018.

[24] T. Geng et al., "LP-BNN: Ultra-low-Latency BNN Inference with Layer Parallelism," in *2019 IEEE ASAP* volume 2160-052X, pages 9–16, 2019

[25] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors *Advances in Neural Information Processing Systems 25* pages 1097–1105. Curran Associates, Inc., 2012.

[26] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning," MIT Press, 2016

[27] M. Abusultan and S. P. Khatri, "Implementing low power digital circuits using flash devices," *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 109-116, doi: 10.1109/ICCD.2016.7753268.

[28] H. Li and J. Hu, "The Research on SAR ADC Integrated Circuit," in *Journal of Physics: Conference Series*, (Vol. 1314, No. 1, p. 012022). October, 2019

[29] "Synopsys website," http://www.synopsys.com/.

[30] "PTM website," http://ptm.asu.edu/.

[31] "TensorFlow website," https://www.tensorflow.org/.

[32] "ImageNet website," https://www.image-net.org/.

[33] "Cifar-10 website," https://www.cs.toronto.edu/ kriz/cifar.html.

[34] Chi-Wah Kok; Wing-Shan Tam, "Bandgap Voltage Reference," in *CMOS Voltage References: An Analytical and Practical Perspective*, IEEE, 2013, pp.71-101.

[35] Y. Shang and T. Ohsawa, "Accurate Measurement of Sneak Current in ReRAM Crossbar Array with Data Storage Pattern Dependencies," *2019 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, 2019, pp. 1-2, doi: 10.1109/VLSI-TSA.2019.8804668.

[36] Tae-Sung Jung et al., "A 117-mm/sup 2/ 3.3-v only 128-MB multilevel NAND flash memory for mass storage applications," *IEEE Journal of Solid-State Circuits*, 31(11):1575–1583, Nov 1996.

[37] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, 91(4):489–502, April 2003.