DNN Engine: A 28-nm Timing-Error Tolerant Sparse Deep Neural Network Processor for IoT Applications

Paul N. Whatmough, *Member, IEEE*, Sae Kyu Lee, *Member, IEEE*, David Brooks, *Fellow, IEEE*, and Gu-Yeon Wei, *Senior Member, IEEE*

***Abstract*— This paper presents a 28-nm system-on-chip (SoC) for Internet of things (IoT) applications with a programmable accelerator design that implements a powerful fully connected deep neural network (DNN) classifier. To reach the required low energy consumption, we exploit the key properties of neural network algorithms: parallelism, data reuse, small/sparse data, and noise tolerance. We map the algorithm to a very large scale integration (VLSI) architecture based around an single-instruction, multiple-data data path with hardware support to exploit data sparsity by completely eliding unnecessary computation and data movement. This approach exploits sparsity, without compromising the parallel computation. We also exploit the inherent algorithmic noise-tolerance of neural networks, by introducing circuit-level timing violation detection to allow worst case voltage guard-bands to be minimized. The resulting intermittent timing violations may result in logic errors, which conventionally need to be corrected. However, in lieu of explicit error correction, we cope with this by accentuating the noise tolerance of neural networks. The measured test chip achieves high classification accuracy (98.36% for the MNIST test set), while tolerating aggregate timing violation rates $>10^{-1}$. The accelerator achieves a minimum energy of 0.36 μJ/inference at 667 MHz; maximum throughput at 1.2 GHz and 0.57 μJ/inference; or a 10% margined operating point at 1 GHz and 0.58 μJ/inference.**

***Index Terms*— Deep neural networks (DNNs), hardware accelerators, Internet of things (IoT), machine learning (ML), razor, system-on-chip (SoC), timing error detection and correction, timing error tolerance.**

I. Introduction

THE capability to interpret the complex, noisy real-world data arising from multi-modal sensor-rich systems is a critical enabling technology for the Internet of things (IoT).

Manuscript received January 10, 2018; revised April 21, 2018 and May 16, 2018; accepted May 22, 2018. Date of publication June 18, 2018; date of current version August 27, 2018. This work was approved by Associate Editor Marian Verhelst. This work was supported in part by the U.S. Government under Program DARPA CRAFT HR0011-16-C-0052 and Program PERFECT HR0011-13-C-0022, in part by Intel Corporation, and in part by Arm Inc. *(Corresponding author: Paul N. Whatmough.)*

P. N. Whatmough is with Arm Research, Boston, MA 02451 USA, and also with the School of Engineering and Applied Science, Harvard University, Cambridge, MA 02138 USA (e-mail: paul.whatmough@arm.com).

S. K. Lee was with the School of Engineering and Applied Science, Harvard University, Cambridge, MA 02138 USA. He is now with IBM Research, Yorktown Heights, NY 10598 USA (e-mail: saekyu.lee@ibm.com).

D. Brooks and G.-Y. Wei are with the School of Engineering and Applied Science, Harvard University, Cambridge, MA 02138 USA (e-mail: dbrooks@eecs.harvard.edu; guyeon@eecs.harvard.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2018.2841824

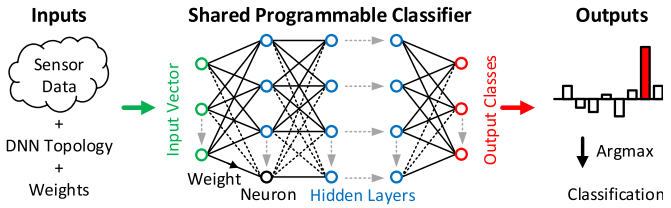


Fig. 1. Overview of inference task using an accelerator with programmable DNN computation graph. The host microcontroller provides input sensor data, topology, and weights. The accelerator processes the DNN graph and returns the output classes.

Machine learning (ML) techniques such as deep neural networks (DNNs) excel at tackling challenging classification and regression problems on sensor data from important application domains such as audio, vision, and medical [1].

As ML capabilities rapidly develop, the application scope for intelligent, autonomous IoT devices will mature. However, the system-on-chips (SoCs) in these devices are typically heavily constrained by form factor, cost, energy, and security limitations. As a result of these constraints, significant research is required across many fields to realize practical products. For the silicon platform, one of the biggest concerns is achieving sufficient energy efficiency to execute ML workloads locally on the device. The micro-controller platforms typically employed in energy-constrained devices are woefully under-powered for many ML techniques. This limitation can be overcome by adding a specialized hardware accelerator (Fig. 1). By efficiently accelerating DNN graph inference, with programmable network topology and model parameters (weights), we can support a wide range of sensor signals for a variety of potential applications with a single dedicated hardware intellectual property (IP) block.

Many of the fundamental neural network architectures that have been presented in the ML literature [1], including spiking neural network (SNN), fully connected neural network (FCNN), convolutional neural network (CNN), and recurrent neural network (RNN), have now been implemented in silicon. Several works have previously described specialized architectures in silicon for CNNs, often focusing on computer vision tasks [2]–[7], such as classifying the ImageNet data set. These require high-compute performance to provide real-time frame rates, and typically require off-chip DRAM storage to process such a large model. This paper complements these results, by focusing on the efficient implementation

0018-9200 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

of much simpler classification problems commonly encountered in autonomous IoT applications without communication capabilities [28]. For these cases, it is typically necessary to minimize the power consumption, along with die size and off-chip components (e.g., off-chip memory).

There is currently a lot of interest in neuromorphic computing, which attempts to more closely mimic the operation of biological systems. For example, a number of implementations of SNNs in silicon have been presented in [8]–[11]. However, to date, SNNs seem to offer very limited accuracy, even on small data sets (e.g., 84% on MNIST in [11]).

In contrast to previous work on CNNs and SNNs, we implement the FC DNN algorithm. FC DNNs can support a wide range of general sensor data classification/regression tasks [12]–[15], which we use as the basis to explore a range of low-power design techniques. However, they are challenging to implement for heavily energy constrained applications due to their large compute and memory footprint. We exploit three desirable properties of FC layers to achieve an energy efficient implementation.

1) *Highly Parallel Graph With Data Reuse:* Efficient and lightweight single-instruction, multiple-data (SIMD) data path, optimized to balance data reuse and local memory bandwidth.
2) *Sparse and Low Range Data:* Graph pruning schedule that removes redundant operations and loads/stores, without restricting parallelism and support for a small 8-bit fixed-point data type.
3) *Noise Tolerance:* Lightweight timing violation detection (Razor) scheme to minimize voltage guard-bands, without a costly explicit error correction mechanism.

This paper [12] is the state of the art in energy efficiency, with a minimum energy of 0.36 μJ/inference at 98.36% accuracy on the MNIST data set. We also demonstrate aggregate timing violation rates $>10^{-1}$, without degrading classification accuracy.

The remainder of this paper is organized as follows. Section II gives an overview of the challenges of implementing DNNs for embedded applications. Section III describes the DNN engine accelerator architecture. Section IV presents further techniques to exploit data reuse and data sparsity to reduce compute and data movement. Section V discusses the application of timing error tolerance techniques to DNNs. Section VI gives the hardware implementation, and Section VII presents the measurement results. Section VIII concludes this paper.

II. Fully Connected Deep Neural Networks

A. Inference Algorithm

The "deep" designation in DNN refers to a model involving more than one hidden layer between the input layer and the output (softmax) layer (Fig. 1). The additional hidden layers allow for much more complex non-linear functions to be learned. A DNN is typically represented as a simple weighted directed acyclic graph consisting of multiple layers of neurons (nodes) and weights (directed edges), as shown in Fig. 1. Specifically, the activation value of the jth neuron in the kth layer, $x_j(k)$, is given as

$$x_j(k) = \varphi\left(b_j(k) + \sum_i w_{ij}(k) \cdot x_i(k-1)\right) \tag{1}$$

where $w_{ij}(k)$ is a unique weight that defines a connection in the graph, and $b_j(k)$ is a bias term. The non-linearity (activation function) used is the rectified linear unit (ReLU)

$$\varphi(x) = \max(x, 0). \tag{2}$$

The weights and bias, $\boldsymbol{W}(k)$ and $\boldsymbol{b}(k)$, for each layer are optimized during the training process [1], which is not described in this paper.

The intrinsic data reuse of the FC graph (Fig. 1) is high, as denoted by the large number of edges that fan-out from each neuron output. In Section IV, we will describe a schedule that allows this reuse to be exploited. However, this only extends to the activation data; there is no fundamental reuse of the weights. By batching multiple inference tasks, it is possible to introduce reuse, but this comes at the cost of increased latency.

B. DNN Hardware Accelerators

A dedicated hardware accelerator typically offers significantly better efficiency than using software running on a general-purpose microprocessor [15]. We are essentially trading efficiency for flexibility. DNNs represent a particularly good target for hardware acceleration, because we can accelerate a single algorithm, which can then be programmed to do a range of different tasks by changing only the graph topology and the weights (i.e., the DNN model). This is the approach we follow in this paper.

This inner-product form of (1) is similarly a very common idiom in digital signal processing (DSP) algorithms, e.g., finite-impulse response (FIR) filters [16]. However, here, the kernel is typically much wider and involves much more data storage and movement than is required in a filter. Nonetheless, many common optimizations used for DSP algorithms are also relevant here. Just like with the FIR algorithm, FC DNNs map very well to simple SIMD data paths.

III. Accelerator Architecture

A. SIMD Architecture

The DNN engine is a five-stage SIMD-style programmable sparse matrix-vector machine for processing arbitrary DNNs (Fig. 2). The architecture is based around an eight-way SIMD multiply-accumulate (MAC) data path. Each lane of the data path processes a single neuron to completion, with eight in flight at any time. Each lane is fed by the same activation value read from IPBUF[1], when processing the input layer, or from the XBUF SRAM, when processing hidden layers. A vector of eight unique weights is loaded from the on-chip WMEM SRAM. Once a set of in-flight neurons are complete, the activation stage adds the bias term to the accumulator register, applies the activation function, and writes the eight

[1]The input data are held in a separate SRAM to XBUF, so that it is not overwritten. This allows us to run the accelerator continuously for accurate power measurement without a lull in activity due to the host CPU interaction.

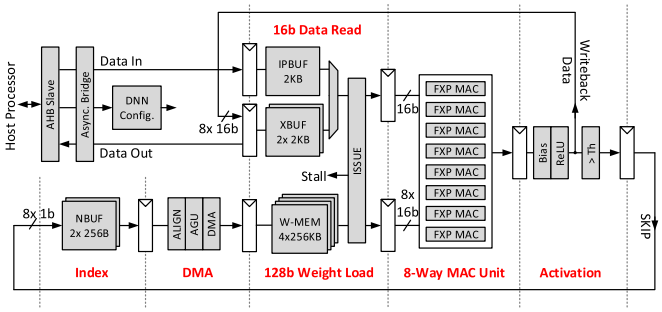


Fig. 2. Block diagram of the five-stage accelerator pipeline.

activations back to XBUF. XBUF is double buffered to allow new activation data to be written for the k^{th} layer, while simultaneously reading data from the $(k-1)^{\text{th}}$ layer in the same cycle. A sequencer schedules the operations based on the configuration register settings, for different DNN topologies of up to eight FC layers with 1–1024 nodes per layer.

B. Weight Storage

A distinctive property of neural networks is the requirement for a large memory footprint to store the weights. The storage requirement for activations is modest as it grows equal with the number of output neurons in the layer. The weight matrix, however, grows as the product of the number of input and output neurons. For inference, we read the weights in the model in their entirety, and hence, we must minimize the distance this data travels to keep energy consumption in check. For heavily constrained applications, it is essential to store the weights as close as possible and ideally, this should be on-chip. Access to off-chip SRAM, FLASH, or DRAM memory incurs orders of magnitude greater energy cost; therefore, moving weight storage on-chip generally is the most important optimization. Of course, this imposes limitations on the model size.

C. Data Types

The accelerator data paths use small fixed-point data types throughout, rather than 32-bit integer or floating-point types, which is a key advantage over a software implementation. In particular, weights are stored as either 8- or 16-bit signed types, which greatly reduce the storage required for each parameter. Support for both is included, as some networks cannot tolerate heavily quantized weights. Weights can be represented as two's complement (TC) or sign-magnitude (SM), which is discussed in Section V. The MAC data paths use 16-bit unsigned activation data, with 32-bit accumulators. The output of the accumulator is rounded back down to 16 bit unsigned after the activation function, with support for programmable rounding modes. The data types used here were chosen to maximize the utility over a range of different networks, as required over the life of the product, rather than over-optimizing to suit a single DNN. For example, the MNIST data set can tolerate weights as small as 1 bit [30], and something in the vicinity of 4 bits per weight is probably energy optimal in many cases [31], [39], but this may not be sufficient for some other (possible unforeseen) networks.

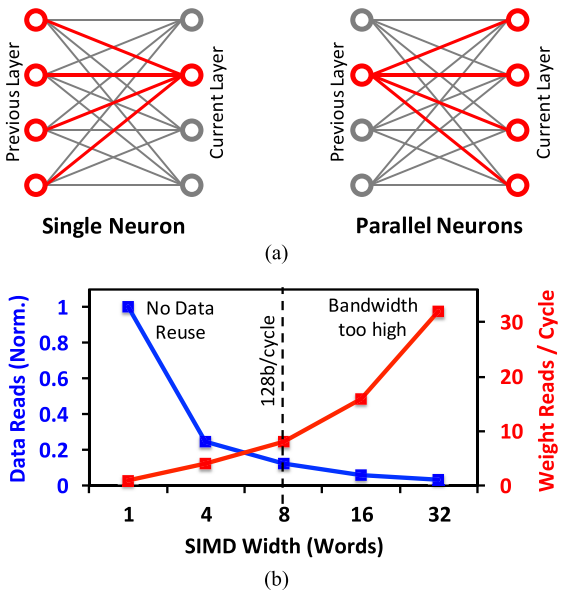


Fig. 3. Data path scheduling tradeoffs. (a) Two example schedules for a four-way parallel MAC unit, both processing four graph edges. The parallel neurons version requires only a single activation read, which is reused across four edges. (b) Advantage of the parallel schedule is apparent as the SIMD width increases, but is limited by memory bandwidth for weights.

D. SoC Interfacing

The accelerator attaches to the SoC via a single 32-bit AHB interface, a prevalent Arm interconnect standard. An asynchronous bridge is incorporated to allow the accelerator to run from a faster clock, asynchronous to the interconnect. A higher bandwidth 128-bit AXI stream interface is used to connect to WMEM. The host CPU on the SoC initiates offloading a DNN inference job by writing the memory-mapped configuration registers in the accelerator. These define the characteristics of the network architecture and various other hardware settings. Next, the input data vector is written into IPBUF, and the accelerator is started with another register write. Once the graph has been fully computed, an interrupt is sent to the host CPU, which can retrieve the output vector from XBUF. For a classification problem, the host CPU must additionally find the largest activation in the output vector, which represents the most likely class.

IV. Data Reuse and Sparsity

A. Data Reuse

Neural networks operate on large data structures stored in SRAM. The cost of data movement to/from SRAM can easily eclipse the compute itself. Therefore, reusing data already loaded from SRAM is a key to reducing power. The FC DNN graph provides scope for reuse of activation data, as shown by the high fan-out from neuron nodes in the graph (Fig. 1).

There are two ways to schedule the FC graph onto an N-way parallel SIMD architecture consisting of multiple MAC lanes [Fig. 3(a)]. The first approach is to compute a single neuron in the current layer at a time. This schedule involves reading N neuron values from the previous layer, along with

N weights [Fig. 3(a)]. Hence, the single neuron schedule does not exploit reuse. The alternative, is to calculate N neurons in parallel (i.e., one per SIMD lane), and read a single activation value from SRAM per cycle. This parallel neuron schedule reduces the memory bandwidth for activations by $1/N$ through reusing a single activation data value over the N lanes. In other words, with the parallel neurons schedule, activation data bandwidth is fixed at one independent of N. The cost of this is that we now require N accumulators (c.f. only a single accumulator is required for the single neuron schedule); however, registers are much cheaper than SRAM access, so this is a good tradeoff.

Unfortunately, there is no reuse opportunity in the weights, so bandwidth for the weights increases with N. Therefore, although there are no data dependencies whatsoever within a single FC layer, the degree of parallelism is still limited by memory bandwidth to WMEM. Fig. 3(b) shows an analysis of the tradeoff between the normalized data reads (reciprocal of reuse factor), and the memory bandwidth requirement, for the parallel neuron schedule. For our design point, we use an eight-way SIMD data path, which represents an $8\times$ reuse factor at a reasonable memory bandwidth that can be satisfied by a standard 128-b AXI interconnect channel.

B. Sparsity

The input data and intermediate activation data in many neural network applications exhibit abundant sparsity, which we exploit to lower the workload. The activation data contain a large number of zero values because of the ReLU activation function which convert negative accumulations to zeros. There are also a large number of small non-zero values [17]. Operations and data movement associated with sufficiently small activations can be ignored, as they are unlikely to significantly influence the final accumulator value.

In previous work, it has been shown that it is possible to save power in activation units by clock-gating registers either side of a MAC data path lane, when either or both of the input operands are zero [2], [4], [5]. Although this reduces power, the resultant pipeline bubbles reduce the data path utilization.

In the DNN engine, bubbles are eliminated by dynamically eliding all zero operands in the activation stage before writeback. In fact, we can skip not only the zero values, but also small non-zero values without degrading the prediction accuracy, which leads to further savings. We implement this in the activation stage (Fig. 2). After the ReLU operator is applied to the neuron accumulations, a comparator is used to compare the values with a per-layer programmable threshold. An optimal threshold is determined for each layer empirically by sweeping the threshold and observing the accuracy degradation [17]. Any rectified activations smaller than the threshold generate a SKIP control signal that predicates writeback to XBUF (Fig. 2). In the following layer, the elided neuron values are completely ignored, and no computation is performed on them. A small double-buffered 512-B SRAM (NBUF in Fig. 2) maintains a list of indexes for the "significant" neurons stored in the previous layer. NBUF then drives the sequencing of the graph traversal and ensures that only "significant" neurons in the previous layer are processed as inputs to the current layer. The NBUF indexes are also used to generate the addresses for WMEM accesses. The address stream becomes non-contiguous due to the data sparsity. For the MNIST data set, the average number of loads, ops, and total cycles are each reduced by approximately $4\times$, significantly improving the energy and throughput, as we see in the results in Section VII.

V. Timing Error Tolerance

A. Background

Razor systems allow excessive worst case V_{DD} guardbands to be safely minimized down to the point where timing violations start to occur, when setup and hold time constraints are occasionally disturbed [18]–[26]. By adding *in situ* circuits to detect the timing violations at critical paths, we are able to track the timing violation rate as it changes over time due to process, voltage, temperature, and aging (PVTA) variations. However, timing violations may result in bit flips at compromised flip-flops, and hence, measures must be taken to correct or otherwise ensure that logic errors do not propagate in the system resulting in functional failure. Therefore, Razor systems typically consist of two key parts: 1) a mechanism to detect timing violations and 2) a mechanism to correct any potential logic errors.

The Razor concept was first applied to RISC CPUs [18]–[20]. In a CPU pipeline, it is generally necessary to correct any potential logic errors to guarantee the correct execution of the software.[2] This is implemented by reusing the checkpoint and replay mechanism in the pipeline to perform exact error correction by reissuing instructions that have potentially been affected by timing violations. Despite good results, the additional complexity of adding Razor to an already complex CPU leads to significant verification challenges. It is also difficult to cleanly isolate the timing violations in the CPU pipeline from the complex memory system, which tends to be increasingly tightly integrated.

Razor and related approaches have also been applied to hardware accelerators [21]–[26]. Accelerators appear to be a better fit, as they tend to be relatively simple designs with a very simple control plane and a predominance of data path logic coupled and simple/regular data movement. This can make error correction mechanisms simpler to design and easier to verify [21].

In fact, in some applications, it is not always necessary to completely correct logic errors. In many cases, a limited number of occasional bit errors will not result in a noticeable degradation according to some performance metric. For example, in many DSP algorithms, it is sufficient to merely limit the worst case magnitude of a logic error in a mean-squared error sense. Therefore, instead of error correction through replay, we can use a cheaper error mitigation technique since the algorithm itself is inherently robust to noise [23]–[26].

[2]An exception to this would be any speculative state, such as branch prediction results, which will be checked anyway at a later time.

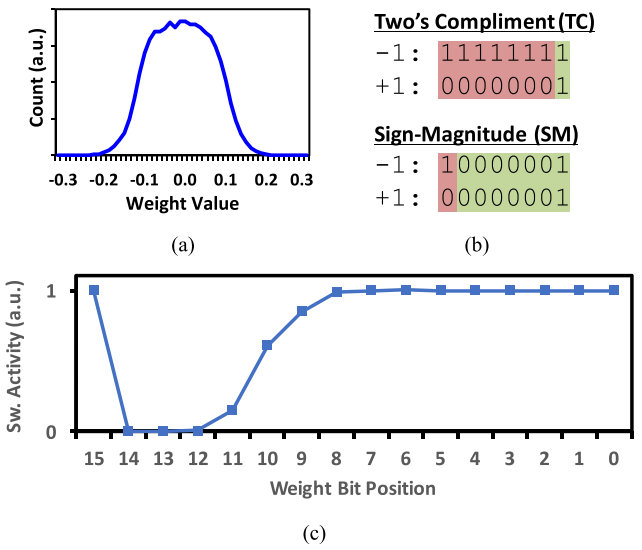


Fig. 4. Exploiting the switching activity of DNN weights is possible due to (a) their Gaussian distribution. (b) Using SM representation removes switching due to sign extension, resulting in (c) low activity in high-order bits (normalized to TC).

B. Error Tolerance in DNNs

Similar to previous work on error tolerant DSP accelerators [23]–[26], ML algorithms are inherently noise-tolerant [29] and are a natural fit for Razor systems, without the burden of exact error correction.

In our work, we use Razor timing error detecting flip-flops (RZFFs) to monitor the timing violations on two timing-critical stages: WMEM load and the MAC unit (Fig. 2). Rather than explicit error correction, we mitigate potential intermittent bit flips using two distinct mechanisms described follows.

1) Algorithm-Level Error Tolerance: Neural networks are remarkably tolerant to noise, which is an essential characteristic that allows them to operate on real-world signals which contain noise and distortion, such as audio and natural images. This inherent algorithmic resilience of DNNs is the basis for timing error tolerance in this paper [29].

To further enhance this effect, we exploit the switching statistics of the weights. The weight matrices have a zero-mean Gaussian distribution [Fig. 4(a)], owing to the L1/L2 regularization function used during the training process [1]. Small magnitude integers with random sign result in a high switching activity in the most significant bit (MSB) positions from one word to the next, when represented using TC, owing to sign-extension. Instead of the TC signed number system, we use a SM representation, which removes the switching activity due to sign-extension in small magnitude numbers, as shown in Fig. 4(b). Hence, using SM for the weights reduces switching activity, normalized to TC [Fig. 4(c)]. In turn, this directly reduces the probability of a timing violation causing a logic error.

Implementing the SM data path requires some changes. Although multiplication is straightforward in SM, addition/subtraction is difficult to implement. Fig. 5 shows our approach with two accumulators, one to sum all positive numbers, and one for the negative ones, which is quite similar to the implementation in [27].

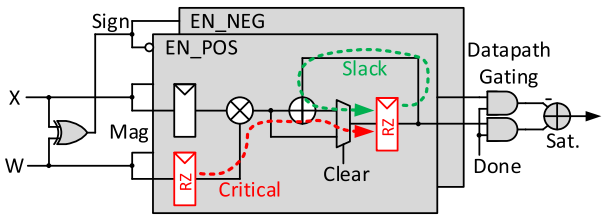


Fig. 5. Modified SM MAC data path, with split unsigned accumulators, data path gating, and output adder to convert back to TC. Also indicated are the critical path from operand registers to accumulator (red arrow) and the slack path from the accumulator through the adder (green arrow).

2) Circuit-Level Error Tolerance: Error tolerance in the data path is harder to achieve because bit flips persist in the accumulator, with no opportunity for masking. Although the SM format previously described offers some benefit in the data path, we additionally introduce the circuit-level technique of time borrowing (TB) in the data path. Generous TB from the accumulator is possible through the feedback path of the adder in the MAC unit, which is much shorter than the path from the operand register through the multiplier and the adder to the accumulator (Fig. 5). This represents a one-sided critical path. TB is enabled using a pulse latch, as described in Section VI. We also introduce a bit-masking (BM) technique that uses per-bit razor error data to mask individual bit errors in the weight word. This is implemented by simply ANDing the inverted error signal from the RZFF with the data bit, such that if a timing-error occurs in a bit position, the output data will be forced to zero, which may not be the correct bit value, but instead represents a round toward zero operations.

VI. Hardware Implementation

An SoC targeting IoT applications (Fig. 6(a)) was implemented in a 28-nm CMOS process and packaged in a 100-pin QFN package. The SoC is based around an ARM Cortex-M0 microcontroller cluster and associated peripherals, memories, and IO. The DNN engine IP itself connects to the M0 cluster through an asynchronous bridge, which allows independent F_{CLK} and V_{DD} scaling to balance throughput and energy efficiency. An on-chip digitally controlled oscillator (DCO) is used to generate the high-performance clock for the accelerator. A 1-MB on-chip SRAM (WMEM) stores the weights for the DNN model (up tro 1 MB) and provides high bandwidth access to the DNN engine through an AXI stream interface. The 1 MB is split into four banks to ease floorplanning, with each bank supplying up to 128-bits (8 weights) per clock cycle.

The Razor implementation uses a dual-mode RZFF constructed from standard cells, with no custom layout. This allows the design to be trivially ported to new process nodes. The basic circuit is a double sampling with time-borrowing design [19], but with the addition of a 2:1 mux that allows either the pulse latch or the flip-flop in the data path. This provides extra flexibility for experimental results, with either hard-edge clocking using the flip-flop in the data path, or alternatively can operate as a pulse latch with the latch in the data

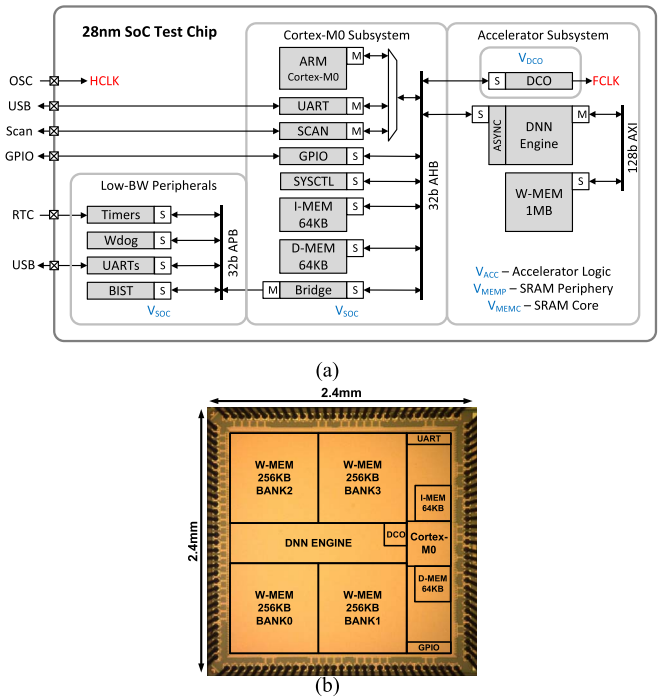


Fig. 6. 28-nm SoC targeting IoT applications. (a) Block diagram. (b) Die photograph.

TABLE I
TEST CHIP SUMMARY

Process Tech.	TSMC 28nm HPC 1P10M
Die Size	2.4mm x 2.4mm
Total SRAM	128KB (SoC), 1MB (WMEM), 6.5KB (DNN)
Total Flip-Flops	8460 (896/8460 RZFF)
ML Model	Fully-Connected Deep Neural Network
Data Types	8/16-bit Weights, 16-bit Act., 32bit Acc.
Model Support	Layers: 0-8, Nodes: 8-1024
Error Tolerance	$>10^{-1}$ @ 98.36% Accuracy
V_{DD}	0.9V (nom.) / 0.6 – 1.1 V (operational)
F_{MAX}	667MHz @ 0.9V (WC sign-off) 1.2GHz @ 0.9V (w/Razor DFS)
Total Power	33.7mW @ 667MHz/0.9V/8b (WC sign-off) 20.3mW @ 667MHz/0.715V/8b (DVS) 63.5mW @ 1.2GHz/0.9V/8b (DFS)
Leakage	3.03mW @ 0.9V

path, which allows for a window of TB. A global pulse clock (90–300-ps pulsewidth) defines the timing detection window, the latch transparency time, and the hold padding required at design time. For extra robustness, we use RZFF on all bits of the WMEM access and MAC stage. All other paths include 30% margin [24] to account for fast on-chip variation effects [38]. The area overhead of the standard cell-based RZFF is 3.24×, which applies to <11% of total flops. An area optimized RZFF cell can be implemented with area overhead as low as 13.6% [32].

The test chip includes both a conventional TC data path as well as the proposed SM data path on-chip to allow both to be measured and compared in silicon. Fig. 7(b) shows the die photograph, and Table I is a summary.

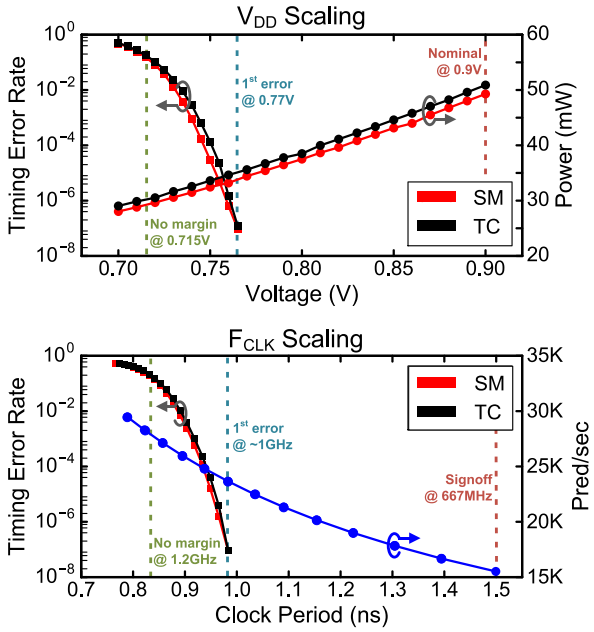


Fig. 7. Measured error rate and power for voltage scaling at 667 MHz (top) and error rate and throughput for frequency scaling at 0.9 V (bottom), with 16-bit weights.

VII. MEASUREMENT RESULTS

A. Voltage Frequency Scaling

Without any mechanism to track on-chip delay variation, it is necessary to always operate at nominal voltage (0.9 V) and the margined sign-off clock frequency of 667 MHz to guarantee correct operation at the worst case PVT conditions (SS, 0.81 V, 125 °C). However, with Razor, we can adaptively remove worst case margins. Fig. 7 shows the measurements from V_{DD} and F_{CLK} scaling experiments. For V_{DD} scaling, we fix the clock frequency at the worst case sign-off value of 667 MHz (SS, 0.81 V, 125 °C), and sweep supply voltage, from nominal voltage of 0.9 V down to 0.7 V. At each step, we measure the power dissipation and timing error rate observed over all 10 K vectors in the MNIST test set. For F_{CLK} scaling, the procedure is similar, but with supply voltage fixed at 0.9 V while sweeping the clock period. The timing violation rate given is the number of words with timing violations detected on one or more bits, as counted by on-chip performance counters. The workload is a DNN with three hidden layers and a topology of $784 \times 256 \times 256 \times 256 \times 10$. Results are given for both TC and SM number formats with 16-bit weights.

At 667 MHz, V_{DD} can scale from nominal (0.9 V) down to 0.77 V before on-chip counters record the first timing violations, which we refer to as the first error. By the time, we reach the first error, we have translated the worst case V_{DD} margin into a 30% power reduction (Fig. 7), for TT silicon running at nominal voltage and room temperature. We can scale V_{DD} below-mentioned the first error, until the zero-margin point, beyond which the accuracy starts to degrade. The region between the first error and the zero-margin point is wide enough to allow the system to operate with an error-rate

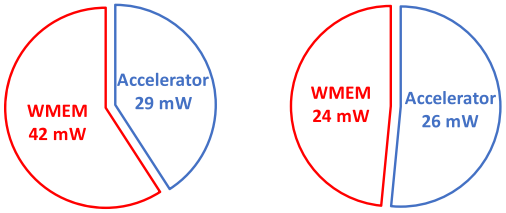


Fig. 8. Measured power breakdown of weights memory (WMEM) and the rest of the accelerator, for 16-bit weights (left) and 8-bit weights (right), at 0.9 V, 1 GHz.

driven dynamic voltage scaling (DVS) loop which targets a small non-zero error rate, while retaining sufficient V_{DD} margin to account for fast variation effects and high-frequency V_{DD} noise. Alternatively, instead of reducing V_{DD} at sign-off F_{CLK}, we can increase F_{CLK} at nominal V_{DD}. At 0.9 V, F_{CLK} can scale beyond 1 GHz before the first timing violations are observed. The zero-margin point is at 1.2 GHz.

At the high-throughput 1.2 GHz/0.9 V operating point, the effective compute efficiency is 1.209 TOPs/W, given typical activation sparsity levels. At the 667 MHz/0.715 V low-power operating point, the effective compute efficiency is slightly higher at 1.227 TOPs/W, including all memory power.

In Fig. 7, the timing error rate as a function of V_{DD} is almost identical for TC and SM, with TC having a marginally higher error rate. In terms of power consumption, the SM data path has slightly lower power consumption due to the reduction in switching activity, although the main motivation for SM is in reducing the impact of timing violations as we will see in the following.

The storage of the model parameters is a big cost for DNNs. Fig. 8 shows a breakdown of the power dissipation split between the WMEM and the rest of the accelerator. Switching from 16- to 8-bit weights reduces the power of the WMEM by almost half, leaving the WMEM power approximately equal to the rest of the accelerator.

B. Timing Error Tolerance

Further improvements are possible by leveraging the inherent resilience of the DNN algorithm. Fig. 9 plots measured classification accuracy versus timing violation rates for weights memory (i.e., WMEM loads), the data path MACs, and the combination. Fig. 9 also includes a summary of this data, with a degradation in absolute accuracy degradation of no more than 0.14% (98.36%).

For the weights memory, SM numbering exploits the zero-mean Gaussian distribution of the weights matrix to reduce switching activity in the MSBs and thus bit flips. On top of this, enabling the BM technique to mask individual bit errors in the weight word allows the accelerator to tolerate SRAM read timing violation rates in excess of 10^{-1}, at 98.36% accuracy.

Error tolerance in the data path is harder to achieve because bit flips persist in the accumulator. Although SM offers some benefit, we mainly rely on circuit-level TB (Section V) to tolerate the timing violation rates commensurate to levels achieved for the memory. As seen in the final summary plot

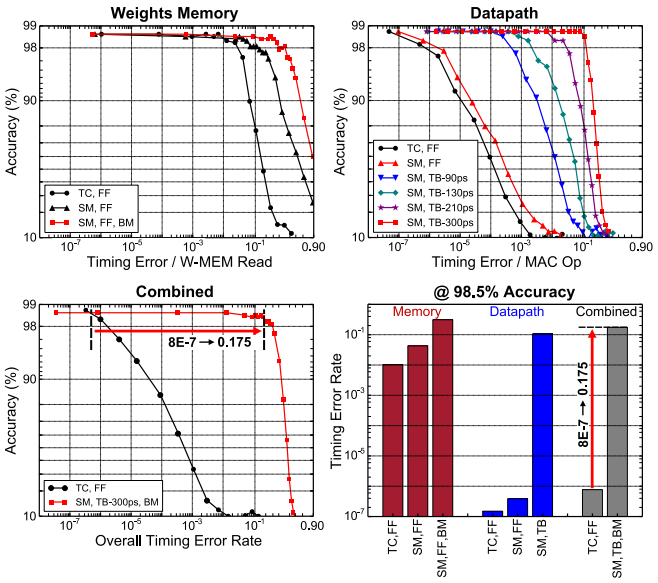


Fig. 9. Classification accuracy on MNIST test set versus error rates in the weights memory (WMEM), data path, and combined. Configurations include 16-bit weights, TC or SM number formats, Razor BM, and TB with multiple clock pulse widths. Clock frequency is fixed at 1 GHz, timing error rates are due to voltage scaling.

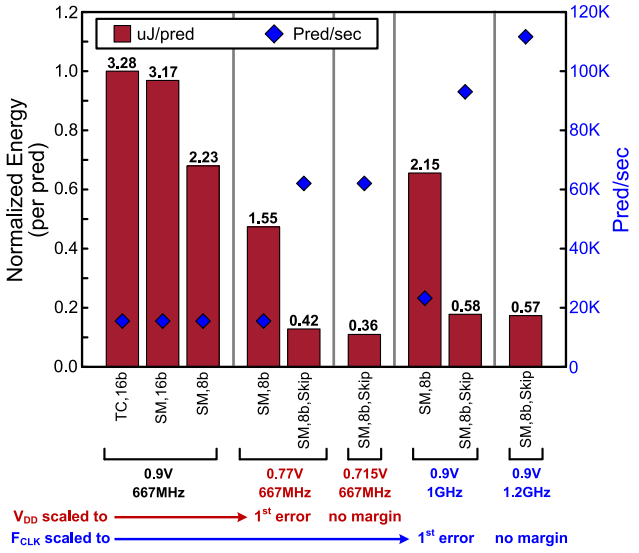


Fig. 10. Measured energy and throughput across different configurations at 98.36% or higher accuracy (MNIST), with DVFS. Configurations include 16- or 8-bit weights, TC or SM format, and sparsity optimization (skip).

in Fig. 9, the combined timing violation tolerance for the DNN engine improves by several orders of magnitude at 98.36% accuracy, over the whole 10 K vector MNIST test set, which supports further V_{DD} reduction down to 0.715 V (no margin).

C. Energy and Throughput

Fig. 10 summarizes energy and throughput improvements offered by different optimizations and techniques. Five V_{DD}/F_{CLK} operating points are given, starting with the worst case (WC) sign-off point of 0.9 V/667 MHz, then showing voltage scaling to the first error point (0.77 V), and the zero-margin point (0.715 V), beyond which the accuracy degrades.

TABLE II
COMPARISON OF TIMING ERROR TOLERANT ARCHITECTURES

	Tschanz ISSCC'10 [19]	Bull ISSCC'10 [20]	Kim CICC'11 [26]	Das CICC'13 [22]	Whatmough ISSCC'13 [24]	This work ISSCC'17 [12]
Tech.	45nm	65nm	180nm	65nm	65nm	28nm
Area	13.64mm^2		4mm^2	0.24mm^2	0.53mm^2	5.76mm^2
Pipeline	8-stage 32-bit LEON-3 CPU	8-stage 32-bit ARM CPU	256-tap correlator	Sobel Edge Detection	16-tap FIR filter	FC DNN
Error - Detection	Double Sampling + Latch	Transition Detector + FF	Algorithmic (FF)	Transition Detector + Latch	Transition-Detector + Latch	Double-Sampling + Latch
Error - Correction	Exact-Replay	Exact-Replay	Approximate-Mean/Median	Exact-Replay	Approximate-Interpolation + TB[c]	Approximate-Algo. + TB[c]
F_{MAX}	1.45 GHz	1 GHz	50 MHz	667 MHz	1 GHz	667 MHz
F_{MAX} Gain	41%[a]	50%[b]				50%[a] / 80%[b]
Energy Gain	22%[a]	52%[b]	87%[b]	34%	37%[a]	30%[a] / 40%[b]

[a] First error point, with margin remaining.
[b] Zero-margin point, beyond which accuracy degrades.
[c]TB – time borrowing.

The last two operating points show the frequency scaling up to the first error point (1 GHz) and finally the zero-margin point (1.2 GHz).

Overall, using Razor DVS and architectural techniques, energy was reduced by more than 9× down to 0.36 μJ/inference, with a simultaneous increase in throughput of about 4×. This improvement can be broken down into the optimizations of small data types ("8 bit"), number representation ("SM"), aggressive V_{DD} scaling due to error tolerance ("no margin"), and exploiting data sparsity ("Skip"). In particular, exploiting data sparsity results in the most significant savings in energy/inference (3.7×), which strongly motivates further research in this direction.

Alternatively, for maximum throughput, the clock frequency can be scaled up to 1.2 GHz (at 0.9 V), an improvement of about 80% at the zero-margin point, when including the architectural techniques. Classification latency (c.f. throughput), is an important consideration for real-time IoT applications. At 1.2-GHz F_{CLK}, the latency for this network is 35.8 μs.

We also ported a network for a 12-word keyword spotting (KWS) data set. The topology is 403 × 200 × 200 × 12, which achieves a 99% accuracy with a weighted f-score of 0.8 [37]. At the 667 MHz/0.715 V operating point, the KWS network achieves an energy efficiency of 135 nJ/inference.

D. Comparison with State of the Art

For energy constrained applications, comparing published results on the merits of operations per second, or operations per Watt, is not always constructive because there is a range of ML models in use, which achieve different accuracies and require a wildly varying number of operations. Ultimately, the most worthwhile comparison is of energy versus accuracy on a particular data set. To this end, Fig. 11 shows a comparison of hardware measurement results for previously published chips that report MNIST energy and accuracy results. A range of energy/accuracy points are shown for the DNN engine,[3]

[3]Includes accelerator and memory (WMEM) power, operating fully embedded, with no external memory.

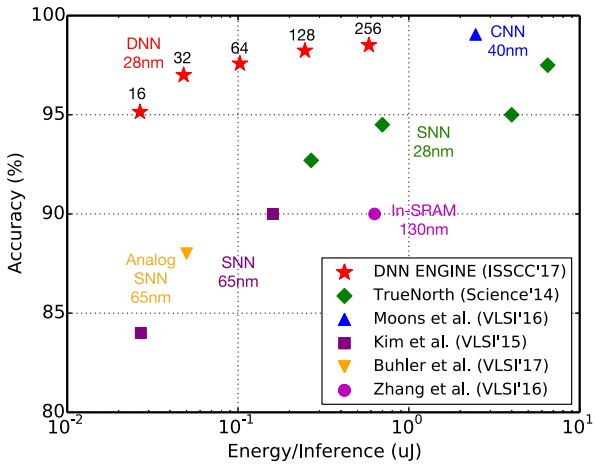


Fig. 11. State-of-the-art measured MNIST accuracy versus energy results. Measurements of this paper (DNN engine [12]) are shown for five DNNs, from 256 nodes/layer down to 16, with three hidden layers. Also shown the SNN [8]–[10], CNN [5], and in-memory (lower-resolution) [33] accelerators.

achieved by scaling the number of nodes in each of the three hidden layers.

Fig. 11 includes not only different hardware architectures but also different ML models. CNNs are well suited to image classification problems, and achieve slightly higher accuracy, albeit at higher power [5].[4] SNNs appear to occupy a frontier that is at least an order of magnitude greater in energy [8]–[10]. SNNs also seem to struggle to achieve high accuracy on MNIST, even with very large ensembles [11]. Buhler *et al.* [9] show an interesting result, which is an analog implementation of an SNN. In energy-accuracy terms, their results seem in line with digital SNNs. Bankman *et al.* [36] also implemented a mixed-signal approach for a CNN data path, with results for a different data set. Zhang *et al.* [33] present an ensemble weak classifier, where the computation is performed in a standard 6T SRAM. Results for a heavily cut-down version of MNIST (9 × 9 pixel images instead

[4]The reported CNN power in [5] does not include the off-chip DRAM.

of 28×28) show 90% accuracy at 630 pJ/inference. Similarly, Ando *et al.* [34] described an in-memory SRAM-based accelerator, this time for a binary FC network. Results on a cut-down version of MNIST (22×22 pixels instead of 28×28) show 90.1% accuracy, but energy/inference are not reported. Biswas *et al.* [35] describe a CNN implemented in a 10T SRAM structure, reporting an accuracy of 96% over 100 images (full test set is 10 000 images); energy/inference is not reported. Despite the interest in mixed-signal and in-memory architectures, they are not yet competitive. Due to vast differences in reported metrics, Fig. 11 only includes published chips that report energy/accuracy on the MNIST data set, which eliminates [13], [14], and [34]–[36].

Finally, Table II shows a comparison with previously published timing error tolerant test chips. Similar to the error tolerant DSP accelerators and in contrast to the CPUs, our work on DNNs does not need to perform exact error correction, which is a significant advantage. In fact, compared to the DSP accelerators, there are no additional pipeline stages at all in this design. The demonstrated timing error tolerance of greater than 10^{-1} without any replay is a compelling result. Comparing the energy savings for these designs is not really informative, as they have wildly different applications. Nonetheless, our reported numbers are compelling; voltage scaling provides around 30% energy gain, while retaining a healthy noise margin, or 40% gain at the zero-margin point.

VIII. Conclusion

Low-power hardware for processing DNNs is a key enabling technology for IoT applications. We presented a 28-nm SoC incorporating a specialized accelerator for FC DNN inference. The very large scale integration (VLSI) architecture and circuit implementation of the accelerator is optimized by exploiting inherent properties of neural networks, including: parallelism, data reuse, sparse/small-range data, and noise tolerance. The SoC includes on-chip SRAM to minimize the energy cost of accessing the model parameters.

Measurement results demonstrate the state-of-the-art energy efficiency, with a minimum energy of 0.36 μJ/inference at 98.36% accuracy on the MNIST data set. We also demonstrate the aggregate timing violation rates $> 10^{-1}$, without degrading classification accuracy.

References

[1] I. Goodfellow *et al.*, *Deep Learning*. Cambridge, MA, USA: MIT Press, Nov. 2016.

[2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2016, pp. 262–263.

[3] G. Desoli *et al.*, "A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 238–239.

[4] 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 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 246–247.

[5] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2016, pp. 1–2.

[6] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 240–241.

[7] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "A 1.42 TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan./Feb. 2016, pp. 264–265.

[8] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640 M pixel/s 3.65 mW sparse event-driven neuromorphic object recognition processor with on-chip learning," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2015, pp. C50–C51.

[9] F. N. Buhler, P. Brown, J. Li, T. Chen, Z. Zhang, and M. P. Flynn, "A 3.43 TOPS/W 48.9 pJ/pixel 50.1 nJ/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40 nm CMOS," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, 2017, pp. C30–C31.

[10] P. Merolla, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[11] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1117–1125.

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

[13] S. Bang *et al.*, "A 288 μW programmable deep-learning processor with 270 KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 250–251.

[14] M. Price, J. Glass, and A. P. Chandrakasan, "A scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 244–245.

[15] B. Reagen, R. Adolf, P. N. Whatmough, G.-Y. Wei, and D. Brooks, *Deep Learning for Computer Architects* (Synthesis Lectures on Computer Architecture). San Rafael, CA, USA: Morgan & Claypool, Aug. 2017.

[16] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ, USA: Wiley, Jan. 1999.

[17] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 267–278.

[18] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2003, pp. 7–18.

[19] J. Tschanz *et al.*, "A 45 nm resilient and adaptive microprocessor core for dynamic variation tolerance," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2010, pp. 282–283.

[20] D. Bull, S. Das, K. Shivshankar, G. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2010, pp. 284–285.

[21] G. Dasika, S. Das, K. Fan, S. Mahlke, and D. Bull, "DVFS in loop accelerators using BLADES," in *Proc. 45th ACM/IEEE Des. Autom. Conf.*, Anaheim, CA, USA, Jun. 2008, pp. 894–897.

[22] S. Das, G. S. Dasika, K. Shivashankar, and D. Bull, "A 1 GHz hardware loop-accelerator with razor-based dynamic adaptation for energy-efficient operation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 8, pp. 2290–2298, Aug. 2014.

[23] P. N. Whatmough, S. Das, D. M. Bull, and I. Darwazeh, "Circuit-level timing error tolerance for low-power DSP filters and transforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 989–999, Jun. 2013.

[24] P. N. Whatmough, S. Das, and D. M. Bull, "A low-power 1 GHz razor FIR accelerator with time-borrow tracking pipeline and approximate error correction in 65 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2013, pp. 428–429.

[25] R. Hegde and N. R. Shanbhag, "A voltage overscaled low-power digital filter IC," *IEEE J. Solid-State Circuits*, vol. 39, no. 2, pp. 388–391, Feb. 2004.

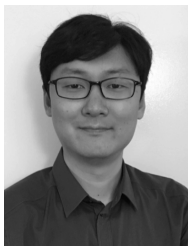
[26] E. P. Kim, D. J. Baker, S. Narayanan, D. L. Jones, and N. R. Shanbhag, "Low power and error resilient PN code acquisition filter via statistical error compensation," in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2011, pp. 1–4.

[27] A. P. Chandrakasan, "Ultra low power digital signal processing," in *Proc. 9th Int. Conf. VLSI Des.*, Bangalore, India, 1996, pp. 352–357.
[28] P. N. Whatmough, G. Smart, S. Das, Y. Andreopoulos, and D. M. Bull, "A 0.6 V all-digital body-coupled wakeup transceiver for IoT applications," in *Proc. Symp. VLSI Circuits (VLSI Circuits)*, Kyoto, Japan, 2015, pp. C98–C99.
[29] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. Des. Automat. Conf. (DAC)*, to be published.
[30] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 4107–4115.
[31] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, USA, 2017, pp. 1921–1925.
[32] Y. Zhang *et al.*, "iRazor: Current-based error detection and correction scheme for PVT variation in 40-nm ARM Cortex-R4 PROCESSOR," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 619–631, Feb. 2018.
[33] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.
[34] K. Ando *et al.*, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018.
[35] 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 *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2018, pp. 488–490.
[36] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2018, pp. 222–224.
[37] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G.-Y. Wei, "Applications of deep neural networks for ultra low power IoT," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, Boston, MA, USA, Nov. 2017, pp. 589–592.
[38] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. M. Bull, "Power integrity analysis of a 28 nm dual-core ARM Cortex-A57 cluster using an all-digital power delivery monitor," *IEEE J. Solid-State Circuits*, vol. 52, no. 6, pp. 1643–1654, Jun. 2017.
[39] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3007–3016.

Paul N. Whatmough (M'09) received the B.Eng. degree (Hons.) in electronic communications engineering from the University of Lancaster, Lancaster, U.K., in 2003, the M.Sc. degree (with distinction) in communications systems and signal processing from the University of Bristol, Bristol, U.K., in 2004, and the Ph.D. degree in electronic engineering from University College London, London, U.K., in 2012.

From 2005 to 2008, he was a Research Scientist with Philips/NXP Research Labs, Cambridge, U.K., working on hardware architecture and signal processing for software defined radio. From 2008 to 2015, he was with the Silicon Research and Development Department, ARM Ltd., Cambridge, U.K., working on research in the areas of hardware accelerators, digital signal processing (DSP), variation tolerance, supply voltage noise, and circuits and systems for emerging Internet of things applications. From 2015 to 2017, he was a Research Associate with Harvard University, Cambridge, MA, USA, leading the inter-disciplinary research on machine learning. He currently leads research on hardware for machine learning at Arm ML Research Group, Boston, MA, USA, and is a part-time Associate with Harvard University. He has co-authored the book *Deep Learning for Computer Architects* (Morgan & Claypool, 2017).

Dr. Whatmough was a recipient of the IET Student Project Award in 2003, the IEEE Communications Chapter Award in 2004, the European Wireless Technology Conference (EuWiT) Young Engineering Prize in 2008, and the IEEE/ACM International Symposium on Low-Power Electronic Design (ISLPED) Best Paper Award in 2015. He has served on the technical program committee of numerous conferences, including the ISLPED, the International Symposium for Computer Architecture (ISCA), the International Symposium on Microarchitecture (MICRO), and the International Symposium for High-Performance Computer Architecture (HPCA).



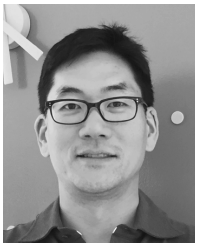
Sae Kyu Lee (M'10) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, the M.S. degree in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, and the Ph.D. degree from Harvard University, Cambridge, MA, USA.

He was with the Intel Corporation, Austin, TX, USA, and Advanced Micro Devices, Boxborough, MA, USA, where he worked on mobile microprocessor design. He is currently with IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. His current research interests include energy-efficient accelerator design for machine learning applications and very large scale integration (VLSI) design for efficient on-chip power delivery solutions.

David Brooks (F'16) received the B.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA.

He was a Research Staff Member with IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Haley Family Professor of computer science with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. His current research interests include resilient and power-efficient computer hardware and software design for high-performance and embedded systems.

Dr. Brooks was a recipient of several honors and awards including the ACM Maurice Wilkes Award and ISCA Influential Paper Award.



Gu-Yeon Wei (SM'00) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA.

He is currently a Robert and Suzanne Case Professor of electrical engineering and computer science with the Paulson School of Engineering and Applied Sciences (SEAS), Harvard University, Cambridge, MA, USA. His current research interests include span multiple layers of a computing system: mixed-signal integrated circuits, computer architecture, design tools for efficient hardware, and identifying synergistic opportunities across these layers to develop energy-efficient solutions for a broad range of systems from flapping-wing microrobots to machine learning hardware for Internet of things(IoT) devices to large-scale servers.