A Sparse Coding Neural Network ASIC With On-Chip Learning for Feature Extraction and Encoding

Phil Knag, *Student Member, IEEE*, Jung Kuk Kim, *Student Member, IEEE*, Thomas Chen, *Student Member, IEEE*, and Zhengya Zhang, *Member, IEEE*

***Abstract*—Hardware-based computer vision accelerators will be an essential part of future mobile devices to meet the low power and real-time processing requirement. To realize a high energy efficiency and high throughput, the accelerator architecture can be massively parallelized and tailored to vision processing, which is an advantage over software-based solutions and general-purpose hardware. In this work, we present an ASIC that is designed to learn and extract features from images and videos. The ASIC contains 256 leaky integrate-and-fire neurons connected in a scalable two-layer network of 8 × 8 grids linked in a 4-stage ring. Sparse neuron activation and the relatively small grid keep the spike collision probability low to save access arbitration. The weight memory is divided into core memory and auxiliary memory, such that the auxiliary memory is only powered on for learning to save inference power. High-throughput inference is accomplished by the parallel operation of neurons. Efficient learning is implemented by passing parameter update messages, which is further simplified by an approximation technique. A 3.06 mm² 65 nm CMOS ASIC test chip is designed to achieve a maximum inference throughput of 1.24 Gpixel/s at 1.0 V and 310 MHz, and on-chip learning can be completed in seconds. To improve the power consumption and energy efficiency, core memory supply voltage can be reduced to 440 mV to take advantage of the error resilience of the algorithm, reducing the inference power to 6.67 mW for a 140 Mpixel/s throughput at 35 MHz.**

***Index Terms*—Feature extraction, hardware acceleration, neural network architecture, sparse coding, sparse and independent local network.**

I. Introduction

ONE key component in many classification algorithms involves developing and identifying relevant features from raw data. For some raw data types, e.g., image pixels and audio amplitudes, there is often a set of features that more naturally describe the data. Sparse feature encoding helps reduce the search space of the classifiers by modeling high-dimensional data as a combination of only a few active features and, hence, can reduce the computation required for classification.

Manuscript received August 26, 2014; revised November 06, 2014; accepted December 14, 2014. Date of publication January 20, 2015; date of current version March 24, 2015. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement HR0011-13-2-0015. This paper was approved by Guest Editor Masato Motomura.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: knagphil@umich.edu; jungkook@umich.edu; tcchen@umich.edu; zhengya@eecs.umich.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.2014.2386892

Sparse coding [1] is a class of unsupervised learning algorithms that attempt to both learn and extract the unknown features that exist within an input dataset under the assumption that any given input can be described by a sparse set of features that it learns. The original Sparsenet algorithm that attempts to find sparse linear codes for natural images develops a complete family of features that are similar to those found in the primary visual cortex [1]. (The features are also known as receptive fields, and we will use feature and receptive field interchangeably.) It was shown in [2] that a layer of Hebbian units connected with anti-Hebbian feedback connections learns a sparse code. Research in sparse coding has further evolved in recent years. The sparse-set coding (SSC) network forms efficient visual representations using a small number of active features [3]. The locally competitive algorithm (LCA) implements sparse coding based on neuron-like elements that compete to represent the input [4]. The sparse and independent local network (SAILnet) implements sparse coding using biologically realistic rules involving only local updates [5]. SAILnet was demonstrated to learn the receptive fields that closely resemble those of the primary visual cortex simple cells [5].

The latest sparse coding algorithms are capable of extracting biologically relevant features through unsupervised learning and use inference to encode image using a sparse set of features, therefore they accomplish the two important preprocessing tasks for object classification, namely, feature extraction and encoding. The sparse coding algorithms are naturally mapped to a network of neurons, where the neuron activity is kept sparse, which is an ideal property for low-power implementation. The sparse coding algorithms produce sparse representation of an input image for faster and lower power classification. The unsupervised learning of features, the sparse activation of neurons, and the biologically inspired sparse encoding are the key advantages of sparse coding compared to conventional methods, such as scale-invariant feature transform (SIFT) [6]. The primary objective of this work is to achieve high performance and low-power feature extraction and encoding, which will be important for emerging embedded vision applications ranging from personal mobile devices to micro unmanned aerial vehicles.

Sparse coding algorithms differ in their neural network implementation and learning rules. Some algorithms are nonspiking, i.e., neurons communicate via analog signals [2], [4] and require off-line computation [4], while some recent algorithms are spiking [5], [7], and the learning rules require only the knowledge of local information [5]. In this work, we take advantage of

the biologically plausible and implementation friendly SAILnet algorithm [5] for the design of the sparse coding ASIC [8]. The sparse coding ASIC is intended to be used in embedded vision applications, including image encoding, feature detection, and as a front end to a object recognition system [9], [10]. However, this work or variations of it can be potentially extended to nonvisual classification tasks such as speech recognition.

The design of the sparse coding ASIC leverages many prior works on neural network hardware, yet this ASIC is unique as all aspects of its design are optimized for low-power and high-throughput sparse coding. In the most recent literature, SpiNNaker [11] and Neurogrid [12] are general-purpose hardware. SpiNNaker is a massively parallel ARM processor based, packet-switched system designed to provide a flexible simulator for neuroscience experiments [11]. Neurogrid is designed to perform arbitrary mathematical computations using neurocores communicating via packets [12]. In comparison, our design is a dedicated ASIC that is optimized for sparse coding. In a related work, ConvModule is an event-driven 2-D convolution neural network processor for object recognition [13]. Despite the similarity of the application, our sparse coding ASIC uses a completely different algorithm that learns features to perform sparse image encoding. Mixed-signal neural netework designs have been presented in [14], [15] with highly efficient analog neurons and digital time-multiplexing bus, while digital designs [16], [17] exhibit software-equivalent deterministic behavior, better noise immunity, and scalability to newer technology nodes, though not necessarily as efficient as mixed-signal designs. The neurosynapic core [16] implements digital neurons and crossbar connectivity, and uses SRAM to store offline-trained weights. [17] uses a transposable SRAM array to implement crossbar connectivity and on-chip learning based on spike-timing-dependent plasticity (STDP). In this work, we propose a two-layer network to take advantage of the sparse spiking for a further simplification of the connectivity, and rate-based learning is used instead of time-based learning.

In parallel with neural network developments, significant advancements have been made in recent years in making object recognition processors. An object recognition processor in [18] uses a cellular neural network based visual attention engine, together with key point extraction and object database matching. A multi-object recognition processor in [19] was designed using a perception engine based on neural-fuzzy logic, SIFT descriptor and object database matching. A SIFT object recognition processor in [20] was proposed with a top-down visual attention feedback loop implementing neural-fuzzy inference to improve visual attention. The latest neural-fuzzy object recognition processor in [21] was designed to perform inference and learning using neural-fuzzy algorithms. Impressive performance and energy efficiency have been reported. In comparison, this work uses a completely neural network approach as a promising alternative to the state of the art for learning and extracting salient features and performing sparse encoding.

In the following, we present an introduction of the sparse coding algorithm in Section II, followed by a detailed discussion of the architectural features and chip design in Section III. The test chip measurement results are presented in Section IV. Section V concludes this work.

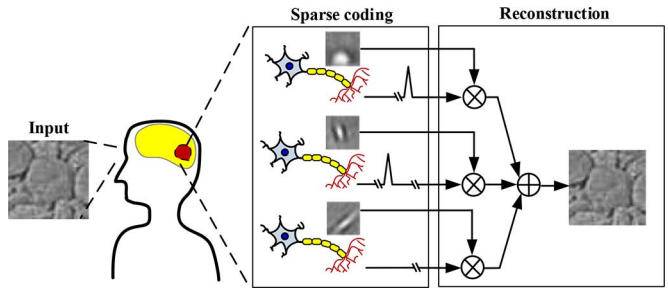


Fig. 1. Sparse coding mimicking sparse neural activities in primary visual cortex. The input is reconstructed by weighted sum of receptive fields of model neurons.

II. SAILnet Sparse Coding Algorithm

A conceptual illustration of the biologically inspired sparse coding processor is shown in Fig. 1 [8], [22]. The sparse coding processor mimics the feature extraction performed by the primary visual cortex. Each neuron in the sparse coding processor develops its receptive field, or feature, through unsupervised learning. A neuron is activated and generates a spike when its receptive field is highly correlated with the input. The spikes are kept very sparse through lateral inhibition. The spikes constitute the sparse code that represents the input image. To check the quality of sparse coding, the input image can be reconstructed by the sparse code and the receptive fields. Fig. 2 shows a whitened input image example, neuron receptive fields learned by the SAILnet algorithm, and the reconstructed image using the sparse code and the receptive fields. The close resemblance of the reconstructed image to the input image demonstrates the effectiveness of the SAILnet algorithm.

In this work, we quantitatively measure the quality of the reconstructed image using a normalized root mean square (NRMSE) metric. NRMSE is the root mean square error normalized to the range. It is mathematically defined by

$$\mathrm{NRMSE} = \frac{\sqrt{\frac{1}{N_p}\sum_{i=1}^{N_p}(\hat{X}_i - X_i)^2}}{\max_i \hat{X}_i - \min_i \hat{X}_i} \tag{1}$$

where X_i is the ith pixel of the input image, $\hat{X}_i$ is the ith pixel of the reconstructed image, N_p is the number of pixels in the image. As an example, the NRMSE of the reconstructed image in Fig. 2 is 0.085.

A. Algorithm Overview

The SAILnet sparse coding algorithm [5] tries to find a sparse set of basis vectors known as receptive fields or features to represent an input image. The SAILnet algorithm is naturally mapped to a network of neurons, and one basis vector is associated with one neuron. The SAILnet algorithm describes two operations, learning and inference [5]. In learning, the basis vectors are first initialized to random values, and through iterative gradient descent, the algorithm converges to a dictionary of basis vectors that allows for an accurate representation of images similar to the training images using a small number of the learned dictionary elements. Learning is done in the beginning to set up the weights and occasionally afterwards to update

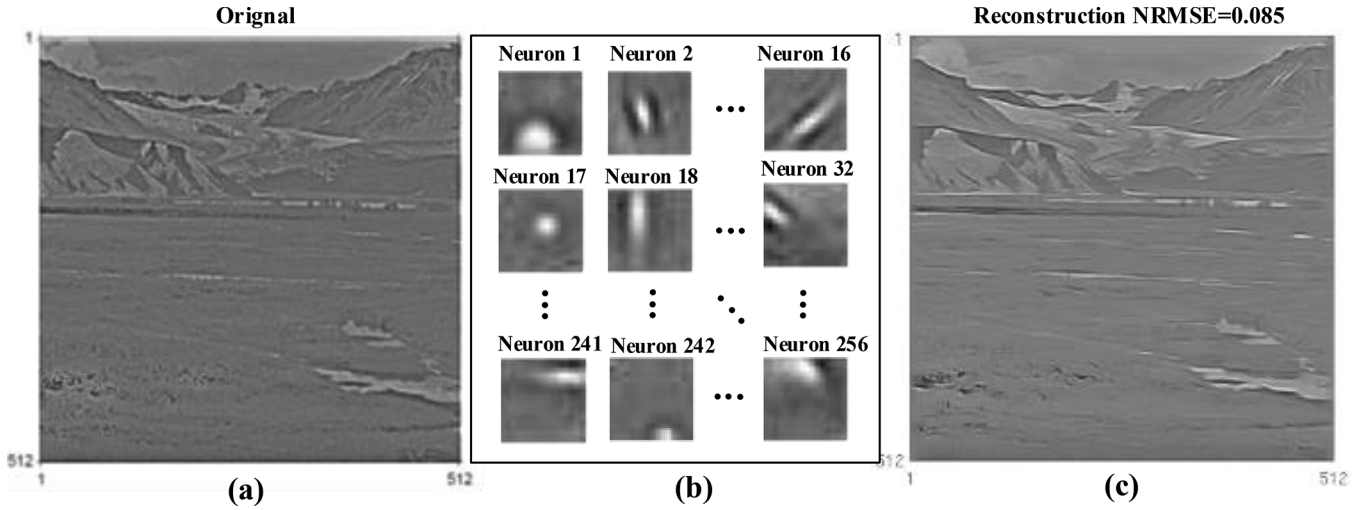


Fig. 2. Sparse coding of (a) an input image using (b) 256 receptive fields of model neurons, and (c) neuron spikes and receptive fields are used to reconstruct the input.

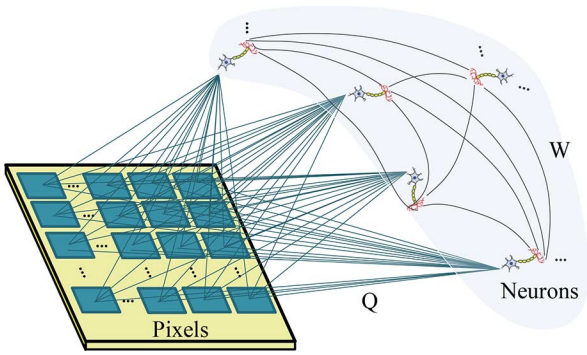


Fig. 3. Feedforward excitatory connections between neurons and pixels and feedback inhibitory connections between neurons.

the weights if the dictionary poorly models new input data, so no real-time constraint is placed on learning. However, inference needs to be done in real time. In inference, the algorithm generates neuron spikes to indicate the activated basis vectors from an input image. Generally, the library size, or alternatively the number of neurons needed by this algorithm, is no less than the number of pixels in the input image, as the overcomplete library tends to capture more intrinsic features and the sparsity of neuron activity improves with an overcomplete library [5].

B. Neuron Connectivity and Dynamics

The neurons are fully connected to each other and each pixel to implement the SAILnet algorithm. A weight is associated with each connection. The feed-forward connections between neurons and pixels are excitatory, and the associated weights are called Q weights. The feedback connections between neurons are inhibitory, and the associated weights are called W weights. An illustration is shown in Fig. 3 [5].

The neural network develops Q weights and W weights through learning. After learning converges, the Q weights of the feedforward connections from a particular neuron represent one basis vector in the dictionary. The W weights represent the strength of directional inhibitions between neurons, which allow neurons to dampen the responses of other neurons if their basis vectors are all highly correlated with an input. The lateral inhibition forces the neurons to diversify and differentiate their basis vectors and minimizes the number of active neurons.

The SAILnet algorithm is based on leaky integrate-and-fire neurons [23]. The neuron activity with respect to an input image is represented by the firing rate of the neurons. The synchronous digital description of a neuron's operation is given by [5]

$$V_i[n+1] = V_i[n] + \eta \left(\sum_{k=1}^{N_p} Q_{ik} X_k - \sum_{j=1, j\neq i}^{N} W_{ij} y_j[n] - V_i[n] \right). \tag{2}$$

where V_i is the voltage of neuron i, and n is the time index. η is the update step size, N_p is the number pixels in the input image patch, and N is the number of neurons in the network. X_k is the value of pixel k in the input image patch, and y_j is the binary output of neuron j. Q is a $N \times N_p$ matrix that stores the feed-forward connection weights, and Q_{ik} stores the weight of the feedforward connection between neuron i and pixel k. W is a $N \times N$ matrix that stores the feedback conection weights, and W_{ij} stores the weight of the feedback connection from neuron j to neuron i (directional). Neuron voltage increases due to input excitation through the feed-forward connections and decreases due to lateral inhibitions and a constant leakage term proportional to the neuron voltage. When the neuron voltage exceeds a threshold voltage, θ, the neuron generates a binary spike output and the neuron voltage is reset to a zero. The threshold voltage θ is a learned parameter specific to each neuron, given by

$$y_i[n] = \begin{cases} 1 \text{ (and } V_i[n] \text{ is reset to 0)}, & \text{if } V_i[n] \geq \theta \\ 0, & \text{if } V_i[n] < \theta. \end{cases} \tag{3}$$

C. Local Learning Rules

Q weights, W weights, and θ for each neuron are learned parameters. In practice, a batch of training images are given as

inputs to generate neuron spikes. The spike counts, s_i, where i is the neuron ID (NID), are then used in parameter updates following the equations below [5]:

$$\begin{aligned} Q_{ik}^{(m+1)} &= Q_{ik}^{(m)} + \gamma s_i \left(X_k - s_i Q_{ik}^{(m)} \right) \\ W_{ij}^{(m+1)} &= W_{ij}^{(m)} + \beta (s_i s_j - p^2) \\ \theta_i^{(m+1)} &= \theta_i^{(m)} + \alpha (s_i - p) \end{aligned} \tag{4}$$

where m is the update iteration number, and α, β, γ are tuning parameters to adjust the learning speed and convergence, p is the target firing rate in units of number of spikes per input image per neuron, and p is used to adjust the sparsity of neuron spikes. One key advantage of the SAILnet learning rules is their locality [5]. Q and θ updates for any particular neuron only involve the spike count and firing threshold of that neuron, and W update only involves the pair of neurons that are part of the lateral connection.

III. Scalable Network Architecture

The SAILnet algorithm can be mapped to a fully connected neural network that consists of simple homogeneous neurons [5]. It is straightforward to parallelize the neurons. However, the communication necessary for sharing the outputs of neurons is one limiting factor [22]. The direct implementation of a fully interconnected network will result in a routing nightmare. In this work, we present a scalable two-layer network architecture that cleanly fits the communication requirements of the sparse coding algorithm. In this architecture, the routing complexity is reduced by replacing all one-to-one connections within a small cluster of neurons with a single bus. The communications between clusters are carried by an upper-layer systolic ring connecting the clusters. The network architecture is described in Section III-A.

A further complication is that memory to store Q weights grows at $\mathcal{O}(N_p N)$ and W weights grows at $\mathcal{O}(N^2)$, where N_p is the number of pixels and N is the number of neurons. As a result, the memory costs significant area and power for a sufficiently large neural network. In this work, we optimize the word length of the weights to reduce the memory storage and partition the memory into two parts, so that, during real-time inference, only one part of the memory is powered on to reduce power consumption. The memory optimization is described in Section III-C.

The results of this work are demonstrated in a 256-neuron sparse coding processor for a 16×16 input image patch. For a larger image, the image is divided into overlapping patches for processing.

A. Two-Layer Sparse Spiking Neural Network

To implement the SAILnet algorithm, low-latency communication for broadcasting neuron spikes to all neurons needs to be done for each inference step. Since each step is directly dependent on the previous step, significant delays in communication will alter the dynamics of the algorithm and worsen the image encoding quality [22]. Interestingly, the sparse coding algorithm produces a very low spike rate, making it possible to use an efficient communication fabric.

In a conventional bus structure [24], [25], communication is a one-to-many broadcast and has low latency for small networks. However, a bus does not scale well with network size. The high fan-out and wire loading of a bus lead to large *RC* delays. Larger neural networks also produce more spikes and thus higher spike collision probability. Spike collisions need to be arbitrated [14], [15], and, to serve many simultaneous spikes in a large network, the bus needs to run at a higher speed than the neurons, increasing the power consumption.

In a conventional ring structure [26], the on-chip interconnects are all local, spikes propagate serially, and spike collisions are eliminated. Since there are no spike collisions, no arbitration is needed, fan-out is low, and the local wire capacitance does not grow with the network size. Therefore, a ring structure is highly scalable. However unlike the bus structure, the serial communication along a ring incurs high latency and alters algorithm dynamics. Significant communication latency degrades the image encoding quality and yields unacceptable results [22].

We create a two-layer hybrid structure, shown in Fig. 4 [8], to combine the unique advantages of the bus and ring structures. At the lower layer, a small cluster of neurons are connected in a bus. The size of the bus $N_1$is chosen to keep the fan-out and wire loading low, so that a low-latency broadcast bus can be achieved. A small bus also keeps the spike collision probability low, so that spike collisions can be discarded and arbitration removed with minimal impact on the image reconstruction error. At the upper layer, a ring is used to connect multiple buses together into a larger network. The length of the ring, N_2, is chosen to keep a low communication latency.

The sizes of the two layers of the hybrid architecture need to meet the requirement that $N_1 N_2 = N$, where N is the size of the neural network ($N = 256$ in this work). There is a tradeoff between N_1 and N_2. The image reconstruction error is measured in simulation as we sweep the size of each (N_1, N_2) pair as shown in Fig. 5. A large N_1 (small N_2) increases the error due to spike collisions, while a large N_2 increases the communication latency. We choose $N_1 = 64$ and $N_2 = 4$ to balance the trade-off. Note that in this and subsequent simulations, we used 1 million 16×16 image patches for training the network. The inference results (image reconstruction error) are based on 16 K 16×16 image patches.

B. Local Grid Structure

In our implementation, each 64-neuron bus is further optimized into a grid structure [12], [14]. The fan-out and wire loading seen by each neuron is quadratically reduced compared to a flat bus. The grid is constructed of static combinational logic blocks, as opposed to a tri-state based approach that was found to be slower and more power consuming.

The spike outputs of the 8×8 grid of neurons are OR'ed together in every row and column as shown in Fig. 6. The OR structure simplifies encoding of spikes to NID to be transmitted to the network. A single spike results in one row and one column output to be activated. The spike is encoded using the address of the activated row and column

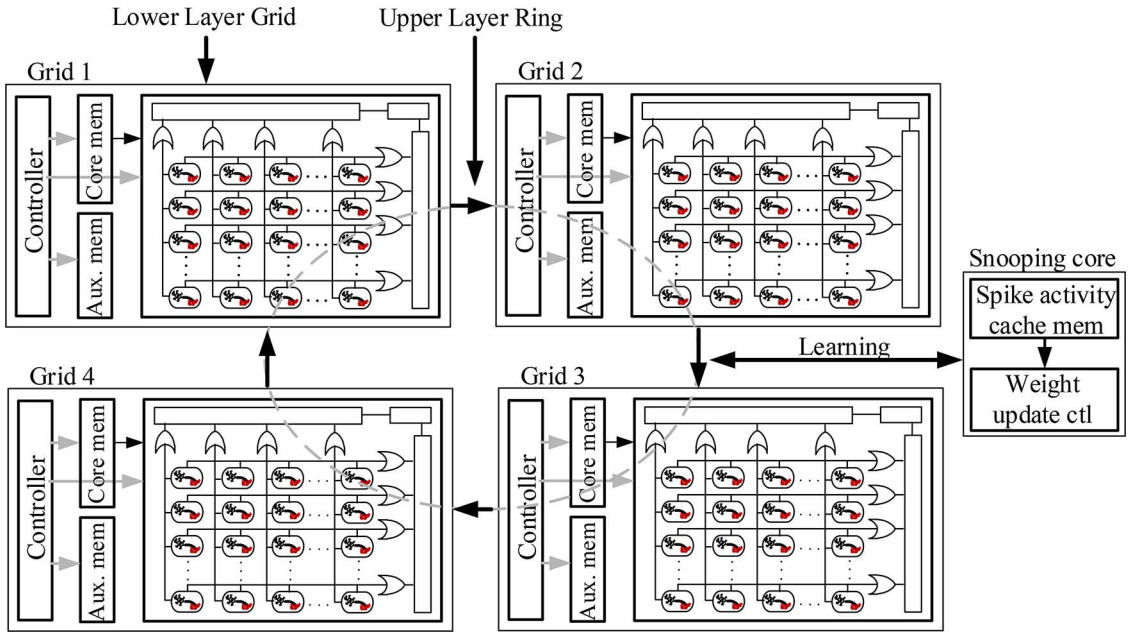


Fig. 4. Two-layer network. Four grids are connected in a 4-stage systolic ring, and the snooping core is attached to the ring to record spikes.

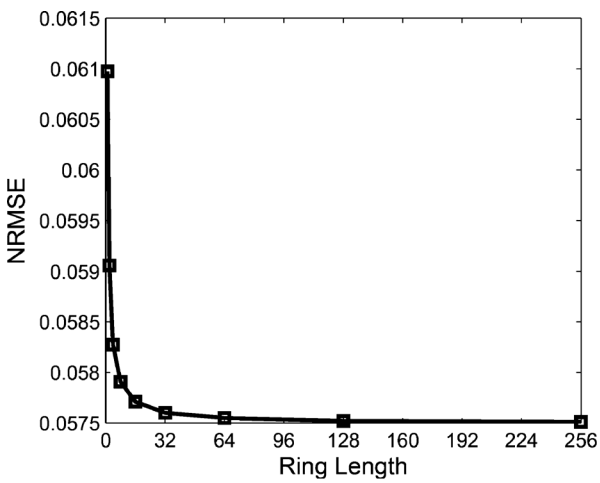


Fig. 5. Effect of ring length (N_2) on image encoding quality. Note that the bus size N_1 is chosen such that $N_1 N_2 = 256$.

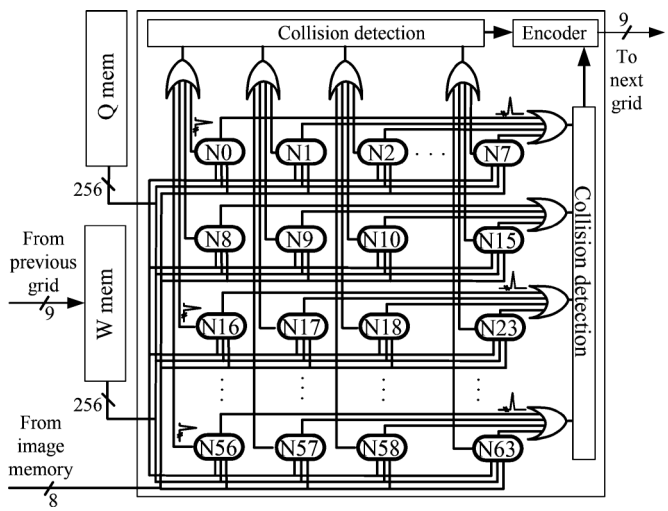


Fig. 6. Diagram of a 64-neuron 2-D grid connected with Q and W memory.

together with the grid ID and a request bit, i.e., NID = {[1b REQ] [2b grid ID] [3b row address] [3b column address]}.

The grid also allows the detection of spike collisions. Multiple spikes will result in two or more rows and columns to be activated. A simple collision detection logic is used to monitor the number of activated rows and columns. Since collisions occur very infrequently, detected collisions are discarded with negligible loss in image reconstruction error. Removing collision arbitration reduces the complexity and improves the throughput.

C. Core and Auxiliary Memory Partition

The 256-neuron network requires a 64 K-word Q memory to store Q weights and a 64 K-word W memory to store W weights. Memory size and power are constraining factors in the hardware implementation. To reduce the word length, we performed an empirical analysis of the fixed-point quantization effects on the image reconstruction error. Given that the input pixels are quantized to 8b, results show that the word length can be reduced to 13 b per Q weight and 8b per W weight for a good performance, as shown in Fig. 7. Longer word lengths produce only marginal improvements.

Furthermore, we found that the word length required by learning and inference differ significantly. Learning requires a relatively long word length, i.e., 13b per Q weight and 8b per W weight to allow for a small enough incremental weight update to ensure convergence, whereas the word length for inference can be reduced to 4b per Q weight and 4b per W weight for a good image reconstruction error as shown in Fig. 8. To save power, the memory is partitioned into a core memory to store 4b MSB of each Q weight and each W weight, and an auxiliary memory to store 9b LSB of each Q weight and 4b LSB of each W weight as shown in Fig. 9. This partition results in a 512 Kb main memory (256 Kb to store Q weights and 256 Kb to store

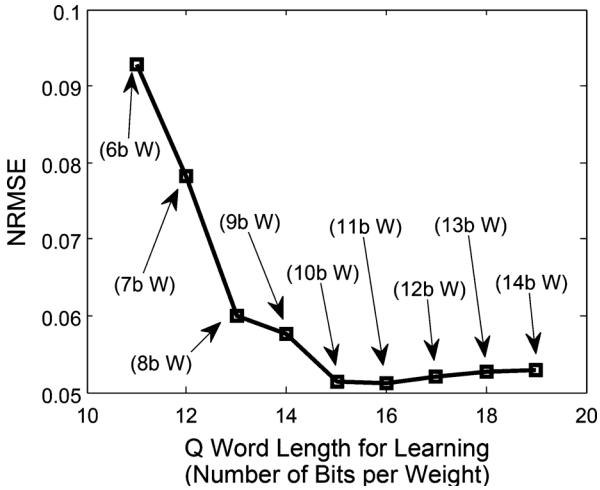


Fig. 7. Q and W weight quantization for learning.

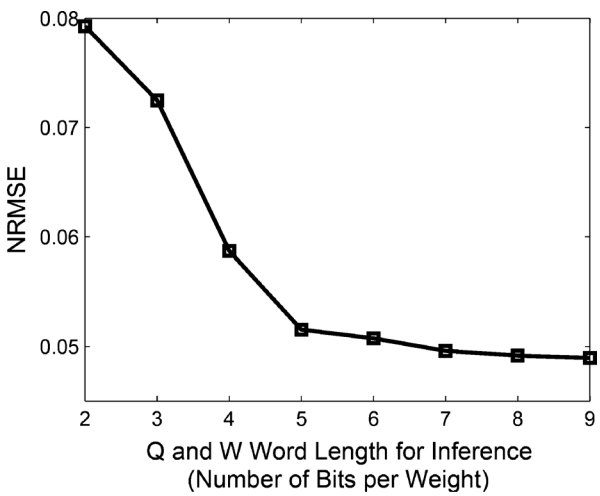


Fig. 8. Q and W weight quantization for inference.

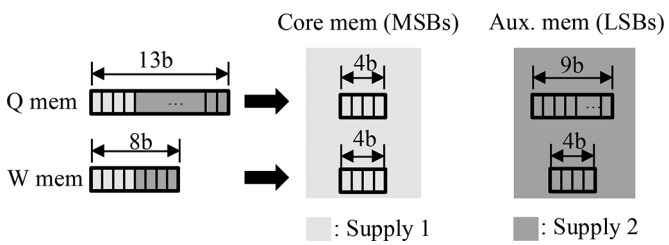


Fig. 9. Diagram of Q and W memory partition. MSB values are stored in core memory and used for both inference and learning. LSB values are stored in auxiliary memory that is powered on during learning.

W weights) and a 832 Kb auxiliary memory (576 Kb to store Q weights and 256 Kb to store W weights). Once the network has been properly trained, the larger auxiliary memory is powered down.

The access bandwidth of the core and auxiliary memory also differ. The core memory is needed for both inference and learning. In every inference step, a neuron spike triggers the simultaneous core memory access by all neurons to the same address corresponding to the NID of the spike. Therefore, the core memory of all neurons in a local grid are consolidated to support the wide parallel memory access by all neurons.

The auxiliary memory is powered on only during learning. Since learning does not need to be in real time, it is implemented in a serial way. Moreover, we implement approximate learning to update weights and thresholds only for the most active neurons, so the fully parallel random access to the auxiliary memory is unnecessary. Hence, the auxiliary memory of all neurons in a local grid are consolidated into a larger address space to improve area utilization.

D. Parallel and Pipelined Inference

A total of 256 neurons are used in this architecture to perform parallel leaky integrate and fire to generate spikes for inference. Inference is done over a number of inference steps n_s that is chosen based on the neuron time constant τ and the inference step size η, i.e., $n_s = w/(\eta\tau)$, where w is the inference period. For a low image reconstruction error, w is chosen to be sufficiently long, e.g., $w = 2\tau$, and the inference step size is chosen to be sufficiently small, e.g., $\eta = (1)/(32)$. With these choices, the number of inference steps is $n_s = 64$.

The leaky integrate and fire described by (2) has two main parts, namely excitation $\sum_{k=1}^{N_p} Q_{ik}X_k$ and inhibition $\sum_{j=1,j\neq i}^{N} W_{ij}y_j[n]$. Excitation computation is a vector dot product (256 4 b $\times$ 8 b multiplies in inference, 256 13 b $\times$ 8 b multiplies in learning) and it results in a constant scalar being accumulated in every inference step, so excitation is computed first using a multiply-accumulate in each neuron.

The inhibition computation is driven by spike events over the inference steps. Since the $y_j[n]$ term in (2) is binary, the inhibition computation is implemented with an accumulator, requiring no multiplication. The inhibition computation is triggered by neuron spikes, i.e., after receiving a spike NID. It takes up to three clock cycles for an NID to travel along the 4-stage ring to be received by every neuron, so a cycle-accurate implementation halts the inference for three cycles after an NID is transmitted. In this way, the inhibition computation over the 64-step inference requires up to $4 \times 64 = 256$ cycles, assuming one spike per inference step. To reduce the latency, we propose to remove the halt to implement approximate inference. In approximate inference, an NID will be received by neurons in different grids at different times, triggering inhibition computations at different times. Excessive spike latency may worsen the image encoding quality. However, since the latency is limited to three cycles, the fidelity is maintained as shown in Fig. 10. Using approximation inference, the inhibition computation over the 64-step inference requires exactly 64 cycles.

The inference operation of this chip is divided into two phases: loading and inference. For still images, loading would take 256 cycles and inference would take 64 cycles. However, in the case of streaming video, we assume that consecutive 16×16 frames can be well approximated by only updating 64 of the 256 pixels. Under this assumption, each step is done in 64 cycles, so that the two steps can be interleaved. The timing chart is shown in Fig. 11. The pipelined processing enables the inference of a 16×16 image patch every 64 cycles, or

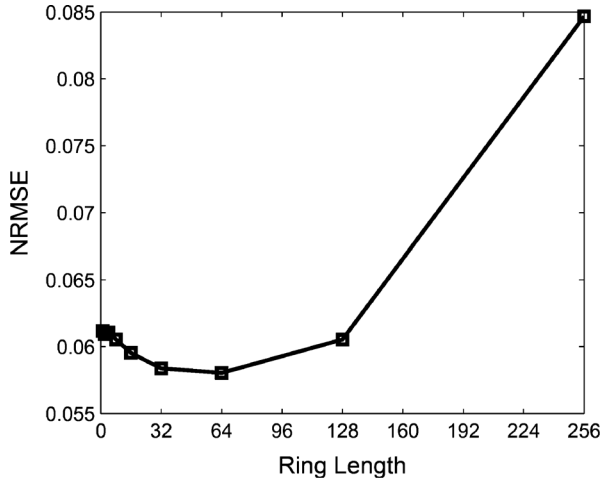


Fig. 10. Effect of spike communication latency (when no pipeline halt is implemented).

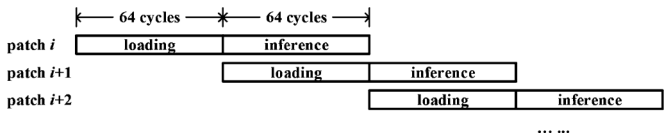


Fig. 11. Inference timing chart.

$\mathrm{TP} = (256 f_{\mathrm{clk}})/(n_s)$ pixel/s, where f_{clk} is the clock frequency and $n_s = 64$ in our design.

E. Learning Using Message-Passing Snooping Core

Learning is implemented on chip with a snooping core that is attached to the ring to snoop spike events. To improve efficiency, parameter updates in learning are done in a batch fashion—spike events are accumulated in a cache for a batch of up to 50 training image patches, followed by batch parameter updates based on the recorded spike counts [5].

Our experimental evidence indicates that active spiking neurons, i.e., neurons with high spike counts, affects learning the most, and active spiking neurons also tend to spike early on. We take advantage of this insight to approximate learning by allocating a small cache to store the spike counts of the first batch of neurons to fire. The approximation reduces the cache memory size and the frequency of parameter updates in order to speed up learning. Based on simulations, we chose a 10-word cache for the snooping core. It is also possible to use a larger cache to improve the image reconstruction error even further.

Of the three types of parameter updates done in learning, Q, W, and θ, Q update is the most costly computationally, as it involves updating the Q weights of all feed-forward connections from the active spiking neurons. To simplify the control of parameter updates, we use a message-passing approach. In the Q update phase, the snooping core sends a Q update message for each of the most active neurons recorded in the cache. The message takes the form of {[1 b REQ] [8 b NID] [4 b SC]}, where REQ acts as a message valid signal and SC is the spike count. Messages are passed around the ring and broadcasted through the grids. A small Q update logic is placed inside each grid to calculate the Q weight update based on (4) when the NID of the message belongs to the grid. The updated weight is saved in the 9 b wide Q auxiliary memory. Occasional carry out bit from the update will result in an update of the 4 b wide Q core memory. The Q updates in all four grids can execute in parallel to speed up the updates.

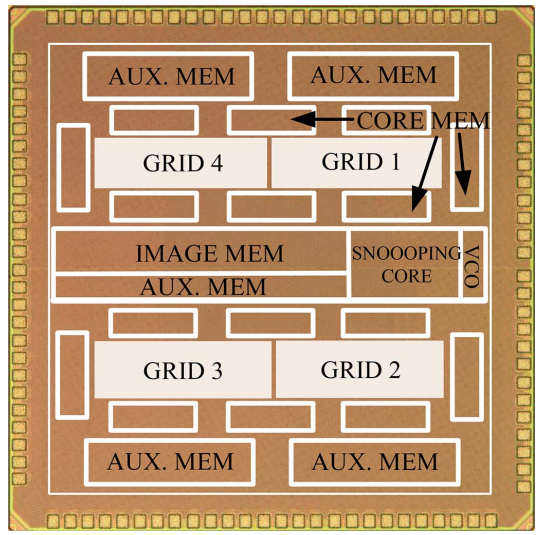


Fig. 12. Chip microphotograph.

W update involves calculating the correlation of spike counts between pairs of the active spiking neurons. The snooping core implements W update by generating a W update message for each active spiking neuron pair. The W update message is in the form of {[1b REQ] [8b NID_1] [8b NID_2] [4b SC_1] [4b SC_2]}, where NID_1 and NID_2 are the pair of active spiking neurons, and SC_1 and SC_2 are the respective spike counts. A small W update logic in the snooping core calculates the W weight update. The updated weight is saved in the 4 b wide W auxiliary memory, and the carry out bit is written to the 4 b wide W core memory.

Similarly, θ update is implemented by passing a θ update message in the form of {[1b REQ] [8b NID] [4b SC]}. θ updates are done by the respective neurons in parallel.

IV. Chip Measurement Results

We incorporate the architectural and algorithmic ingredients described above in an ASIC test chip implemented in a TSMC 65 nm CMOS technology [8]. The microphotograph of the test chip is shown in Fig. 12 with key parts of the design highlighted. The test chip has four separate power rails for four macro blocks: core logic (including neurons, grid and ring logic, and snooping core), 512 Kb core memory implemented in 16 256×128 b register files, and 832 Kb auxiliary memory implemented in four 2048×72 b SRAM to store Q weights and a 2048×128 b SRAM to store W weights, and a voltage-controlled oscillator as the clock source.

The test chip is limited in the number of input and output pads, therefore the input image is scanned bit-by-bit into the SRAM. After the scan is complete, the chip can operate in its full speed. We have made the implicit assumption that the throughput of the ASIC chip is not bounded by its input. We envision this ASIC chip to be integrated with an imager, so that the image input can

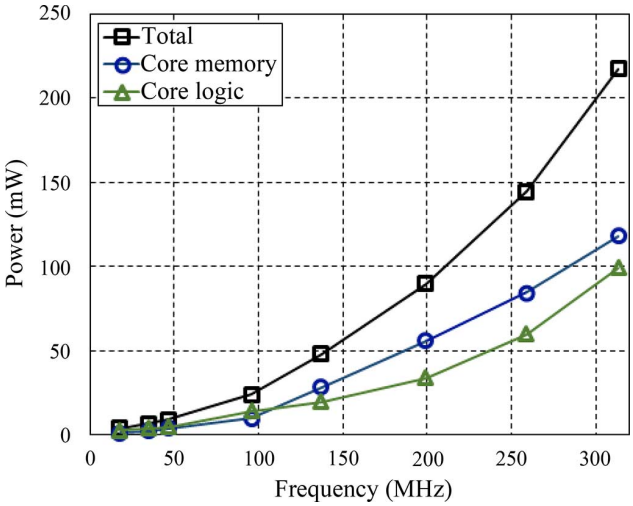


Fig. 13. Inference power consumption and breakdown.

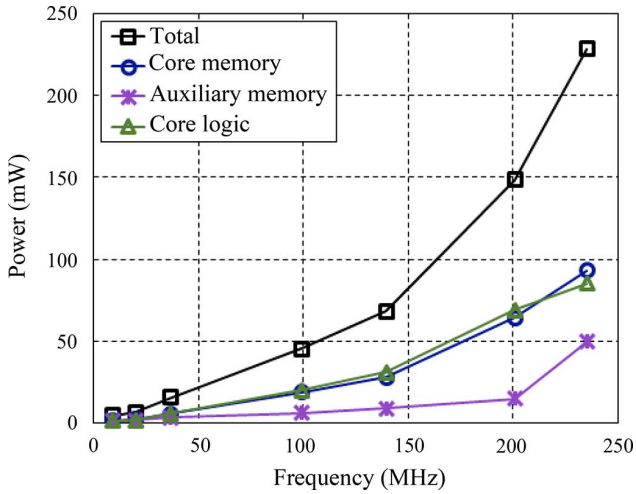


Fig. 14. Learning power consumption and breakdown.

be provided directly on-chip and not be limited by expensive off-chip input and output.

A. Power and Performance

The test chip is fully functional. The measured inference power consumption is plotted in Fig. 13, where each point in the plot corresponds to the power consumption at the lowest supply voltage at the given clock frequency. The auxiliary memory is powered down in inference to save power. At room temperature and 1.0 V core logic and core memory supply, the test chip operates at a maximum clock frequency of 310 MHz for inference, consuming 218 mW. At 310 MHz, the chip carries out inference at 1.24 Gpixel/s (Gpx/s) at 176 pJ/pixel (pJ/px). At 35 MHz and a reduced throughput of 140 Mpx/s, the core logic supply can be scaled to 530 mV and core memory supply can be scaled to 440 mV. The voltage and frequency scaling reduce the power consumption to 6.67 mW and improve the energy efficiency to 47.6 pJ/px.

The measured learning power is shown in Fig. 14. Similarly, each point corresponds to the power at the lowest voltage at the given frequency. The auxiliary memory is powered on in learning. At room temperature and 1.0 V core logic, core memory, and auxiliary memory supply, the test chip achieves a maximum clock frequency of 235 MHz for learning, consuming 228 mW. At 235 MHz, the test chip processes training images at 188 Mpx/s. A large training set of 1 million 16×16 image patches can be processed in 1.4 s. Learning requires writing to memories, which requires a minimum supply of 580 mV for the core memory and 600 mV for the auxiliary memory. At the minimum supplies, the learning power consumption is reduced to 6.83 mW at 20 MHz. The energy efficiency and performance metrics are summarized in Table I.

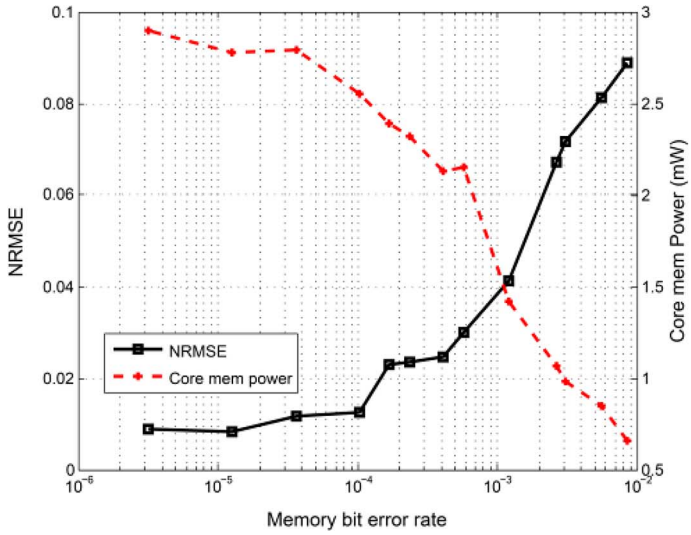
The test chip is the first reported work of dedicated silicon for sparse coding. As a fully digital ASIC implementation, it is most relevant to the prior works on fully digital neural networks [16], [17], both of which contain 256 neurons. Table II compares the key features. Note that the algorithms used are different, so a direct comparison is not very meaningful.

TABLE I
CHIP SUMMARY

Technology	TSMC 65nm GP CMOS			
Core Area	1.75mm × 1.75mm (Core logic: 1.16mm^2, Core mem: 1.01mm^2, Aux. mem: 0.89mm^2)			
Chip Area	2.11mm × 2.11mm (4.45mm^2)			
	Inference		Learning	
Frequency (MHz)	35	310	20	235
Core logic (V)	0.53	1.00	0.50	1.00
Core mem (V)	0.44	1.00	0.58	1.00
Aux. mem (V)	0.00	0.00	0.60	1.00
Throughput (Mpixel/s)	140	1240	16	188
Power (mW)	6.67	218	6.83	228.1
Energy Efficiency (pJ/pixel)	47.6	175.8	426.9	1213

TABLE II
COMPARISON WITH PRIOR WORKS

Reference	Mellola *et al.* [16]	Seo *et al.* [17]	This work
# Neurons	256	256	256
# Synapses	256K	64K	128K
Bitwidth of a Synapse	1 bit	4 bits	8 and 13 bits
Memory size	256Kbits	256Kbits	1.31Mbits
Interconnect	Crossbar	Crossbar	2-layer grid and ring
Algorithm	RBM	STDP	SAILnet
Learning	Off-chip	On-chip	On-chip
Application	Digit recognition	Pattern recognition	Image sparse coding
Technology	IBM 45nm	IBM 45nm	TSMC 65nm
Core Area	4.2mm^2	4.2mm^2	3.1mm^2
Energy metric	45pJ/spike	-	48pJ/pixel



Core mem (V)	Core mem (mW)	Core mem BER	NRMSE
0.330	0.66	8.48E-03	8.90E-02
0.340	0.85	5.63E-03	8.12E-02
0.350	0.98	3.10E-03	7.16E-02
0.360	1.07	2.64E-03	6.72E-02
0.370	1.42	1.22E-03	4.14E-02
0.390	2.15	5.77E-04	3.00E-02
0.395	2.13	4.09E-04	2.46E-02
0.400	2.32	2.34E-04	2.36E-02
0.405	2.39	1.68E-04	2.29E-02
0.420	2.55	1.03E-04	1.25E-02
0.425	2.79	3.64E-05	1.16E-02
0.440	2.78	1.26E-05	8.37E-03
0.450	2.90	3.16E-06	8.74E-03

Fig. 15. Tradeoff between image reconstruction error and memory power consumption.

B. Error Tolerance

One interesting aspect of the sparse coding algorithm is its resilience to errors in the stored memory weights. This resilience stems from the inherent redundancy of the network and the ability to correct errors through on-line learning. In order to explore the benefit of this error tolerance, we looked at voltage over-scaling of the core memory in inference for potential energy savings to exploit the potential tradeoff possible with this system.

Although no dedicated test structure was created in the test chip for the precise measurement of the error rate seen by the internal circuitry during runtime, we tried to approximate the memory bit error rate using the scan chain interface to first write and verify the correct known values at the nominal 1.0 V supply, and then lower the supply voltage, run inference, and read out the values for comparison. Fig. 15 shows the increase of the NRMSE and the reduction of memory power dissipation at supply voltages down to 330 mV and memory bit error rate up to about 10^{-2}. The NRMSE curve is relatively flat up to bit error rate of 10^{-4}. The rapid increase of NRMSE occurs when the bit error rate is above 10^{-3}. The error tolerance measurements, though approximate, highlight the potential for use of low-power unreliable memory elements in the implementation of sparse coding processors.

V. Conclusion

We present a 256-neuron ASIC design for sparse coding. To solve the communication bottleneck, a two-layer network is designed to link four 64-neuron grids in a ring to balance capacitive loading and communication latency. The sparse neuron spikes and the relatively small grid keep the spike collision probability low enough that collisions are discarded with only slight effect on the image reconstruction error. To reduce memory area and power, we divide memory into a core memory and an auxiliary memory that is powered down during inference to save power.

The parallel neural network permits a high inference throughput. Parameter updates in learning are serialized to save the implementation overhead, and the number of updates is reduced by an approximate approach that considers only the most active neurons. A message passing mechanism is used to run parameter updates without costly controls.

The test chip performs inference at 1.24 Gpx/s at 1.0 V and 310 MHz, and on-chip learning can be completed in seconds. The error resilience of the sparse coding algorithm provides extra margin for voltage over-scaling. At 440 mV core memory supply, the inference power consumption is reduced to 6.67 mW for an energy efficiency of 47.6 pJ/px.

Acknowledgment

The authors would like to thank W. Lu, M. Flynn, and G. Kenyon for helpful discussions.

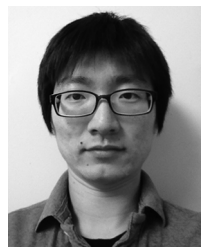
References

[1] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.
[2] P. Földiak, "Forming sparse representations by local anti-Hebbian learning," *Biological Cybern.*, vol. 64, no. 2, pp. 165–170, 1990.
[3] M. Rehn and F. T. Sommer, "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields," *J. Computat. Neurosci.*, vol. 22, no. 2, pp. 135–146, 2007.
[4] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, no. 10, pp. 2526–2563, 2008.
[5] J. Zylberberg, J. T. Murphy, and M. R. DeWeese, "A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields," *PLoS Computat. Biol.*, vol. 7, no. 10, p. e1002250, 2011.
[6] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. IEEE Int. Conf. Comput. Vis.*, 1999, vol. 2, pp. 1150–1157.
[7] S. Shapero, C. Rozell, and P. Hasler, "Configurable hardware integrate and fire neurons for sparse approximation," *Neural Networks*, vol. 45, pp. 134–143, 2013.
[8] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 6.67 mW sparse coding ASIC enabling on-chip learning and inference," in *Proc. Symp. VLSI Circuits*, 2014, pp. 61–62.
[9] L. M. Chalupa, J. S. Werner, and C. J. Barnstable, *The Visual Neurosciences*. Cambridge, MA, USA: MIT, 2004, vol. 1.
[10] D. J. Field, "What is the goal of sensory coding?," *Neural Computation*, vol. 6, no. 4, pp. 559–601, 1994.

[11] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.
[12] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *Proc. Int. Conf. Artif. Neural Netw. Mach. Learning*, 2012, pp. 121–128.
[13] L. Camunas-Mesa, C. Zamarreno-Ramos, A. Linares-Barranco, A. J. Acosta-Jimenez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE J. Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, Feb. 2012.
[14] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 253–265, Feb. 2007.
[15] G. Indiveri, E. Chicca, and R. Douglas, "A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 211–221, Feb. 2006.
[16] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2011, pp. 1–4.
[17] J.-S. Seo *et al.*, "A 45 nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2011, pp. 1–4.
[18] K. Kim, S. Lee, J.-Y. Kim, M. Kim, and H.-J. Yoo, "A 125 GOPS 583 mW network-on-chip based parallel processor with bio-inspired visual attention engine," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 136–147, Jan. 2009.
[19] J.-Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, "A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE. J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, Jan. 2010.
[20] S. Lee, J. Oh, J. Park, J. Kwon, M. Kim, and H.-J. Yoo, "A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 42–51, Jan. 2011.
[21] J. Oh, G. Kim, B.-G. Nam, and H.-J. Yoo, "A 57 mW 12.5 μj/epoch embedded mixed-mode neuro-fuzzy processor for mobile real-time object recognition," *IEEE J. Solid-State Circuits*, vol. 48, no. 11, pp. 2894–2907, Nov. 2013.
[22] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "Efficient hardware architecture for sparse coding," *IEEE Trans. Signal Process.*, vol. 62, no. 16, pp. 4173–4186, Aug. 2014.
[23] P. Dayan and L. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA, USA: MIT, 2001.
[24] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, M. Yamada, and A. Masaki, "Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed on 576 digital neurons," in *Proc. Int. Joint Conf. Neural Netw.*, 1990, pp. 527–535.
[25] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proc. Int. Joint Conf. Neural Netw.*, 1990, pp. 537–544.
[26] U. Ramacher, "SYNAPSE-A neurocomputer that synthesizes neural algorithms on a parallel systolic engine," *J. Parallel Distrib. Comput.*, vol. 14, no. 3, pp. 306–318, 1992.

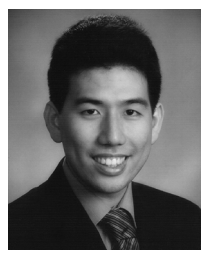
Phil Knag (S'11) received the B.S. degree in computer engineering and M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2010 and 2012, respectively, where he is currently working toward the Ph.D. degree in electrical engineering.

He received a GAANN fellowship in 2010 from the U.S. Department of Education for academic excellence. He was with Medtronic, Inc. as a Research Intern in 2010. His current research interests include nanoscale and neuromorphic computing systems.



Jung Kuk Kim (S'10) received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2009, and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2011, where he is currently working toward the Ph.D. degree in electrical engineering.

He received the 2009 Doctoral Fellowship from the Korea Foundation for Advanced Studies, and the 2012 Riethmiller Fellowship. His research interests are in low-power and high-performance VLSI systems with an emphasis on algorithm and architecture co-design for signal and image processing.



Thomas Chen (S'15) received the B.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2013, where he is currently working toward the M.S. and Ph.D. degrees in electrical engineering.

He received a Rackham Merit Fellowship in 2013 from University of Michigan. His research interests are in high-speed and low-power VLSI circuits and systems.

Zhengya Zhang (S'02–M'09) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, CA, USA, in 2005 and 2009, respectively.

Since 2009, he has been on the faculty of the University of Michigan, Ann Arbor, MI, USA, as an Assistant Professor with the Department of Electrical Engineering and Computer Science. His current research interests include low-power and high-performance VLSI circuits and systems for computing, communications and signal processing.

Dr. Zhang was a recipient of the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Honor Program Award in 2013, the David J. Sakrison Memorial Prize for outstanding doctoral research in EECS at UC Berkeley, and the Best Student Paper Award at the Symposium on VLSI Circuits. He is an associate editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERSand an associate editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS.