A 55-nm, 1.0–0.4V, 1.25-pJ/MAC Time-Domain Mixed-Signal Neuromorphic Accelerator With Stochastic Synapses for Reinforcement Learning in Autonomous Mobile Robots

Anvesha Amaravati, *Student Member, IEEE*, Saad Bin Nasir, *Member, IEEE*, Justin Ting, *Student Member, IEEE*, Insik Yoon, *Student Member, IEEE*, and Arijit Raychowdhury, *Senior Member, IEEE*

***Abstract*— Reinforcement learning (RL) is a bio-mimetic learning approach, where agents can learn about an environment by performing specific tasks without any human supervision. RL is inspired by behavioral psychology, where agents take actions to maximize a cumulative reward. In this paper, we present an RL neuromorphic accelerator capable of performing obstacle avoidance in a mobile robot at the edge of the cloud. We propose an energy-efficient time-domain mixed-signal (TD-MS) computational framework. In TD-MS computation, we demonstrate that the energy to compute is proportional to the importance of the computation. We leverage the unique properties of stochastic networks and recent advances in Q-learning in the proposed RL implementation. The 55-nm test chip implements RL using a three-layered fully connected neural network and consumes a peak power of 690 μW.**

***Index Terms*— Accelerator, autonomous robot, edge computing, low power, reinforcement learning (RL), stochastic synapse.**

I. Introduction

MOST of the recent hardware demonstrations in deep neural networks (DNNs) and convolutional neural networks (CNNs) have addressed image classification applications [1]–[5] and demonstrated breakthrough improvements in energy efficiency. However, autonomous systems, such as mobile robots, which continuously interact with the environment, need to make decisions in real time. Although this problem shares some similarities with image classification, we understand that truly autonomous systems need to be able to make such decisions in real time and also learn from its experiences. Hence, the next generation of hardware systems that can enable autonomy in mobile robots must enable decision-making, such as path planning, obstacle avoidance, and so on. Fig. 1 provides a broad overview of the following three principal classes of machine learning (ML) paradigms.

1) *Supervised Learning:* It involves learning from large sets of labeled data. Typical examples include training image classifiers using the Image-Net database [6] or the MNIST database [7]. Supervised learning involves a training phase and a classification or inference phase.
2) *Unsupervised Learning:* In unsupervised learning, the neural network is not trained using labeled data. Instead, the system is trained to create clusters from the training set [8]. This learning approach can be more bio-mimetic, and recently, it has been used to train spiking neural networks.
3) *Reinforcement Learning:* In reinforcement learning (RL), an autonomous agent trains itself in real time by interacting with a dynamically changing environment and learning from its many experiences.

In a pursuit of learning, the agent first senses the environment using vision sensors, depth sensors, location/position sensors, and so on. Next, it takes an action and calculates a reward based on the action. The reward allows the agent to quantify the so-called quality of its action in the given state space. Eventually, it learns to maximize the notion of cumulative reward in the long run. Thus, RL is a biologically inspired algorithm, which can provide true autonomy in intelligent systems. Recently, it has enjoyed considerable success both

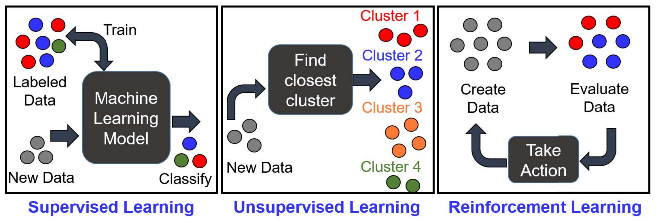


Fig. 1. Three types of ML paradigms: supervised learning, unsupervised learning, and RL.

Manuscript received May 7, 2018; revised August 30, 2018 and November 3, 2018; accepted November 5, 2018. Date of publication December 7, 2018; date of current version January 14, 2019. This work was supported in part by the Semiconductor Research Corporation (SRC), through Extremely Energy Efficient Collective Electronics (EXCEL), an SRC-NRI Nanoelectronics Research Initiative, under Grant 2698.002, and in part by the Center for Brain-inspired Computing Enabling Autonomous Intelligence (C-BRIC) under Grant 2777.005. This paper was approved by Guest Editor Wim Dehaene. *(Corresponding author: Anvesha Amaravati.)*

The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: aamaravati3@gatech.edu; saadbinnasir@gatech.edu; jting31@gatech.edu; iyoon@gatech.edu; arijit.raychowdhury@ece.gatech.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.2018.2881288

in control [9] and gameplay [10]. Notably, RL and other ML techniques were used to beat the best human player in Alpha-Go and play a suite of ATARI games [10].

From a hardware perspective, increased energy efficiency is a key enabler for true autonomy in edge nodes. As researchers are exploring hardware accelerators for ML applications, it has been shown that in most networks, 5–6 bits of resolution is sufficient to enable accurate and robust inference [11]–[13]. This has led to innovative circuit architectures that target low power and reduced area [2], [11], [12]. At lower precision, analog- and mixed-signal computing blocks have shown promise. In particular, most of the demonstrations have used voltage-mode circuits [11], [13], enabled by the switched capacitor designs to implement linear algebraic kernels. For example, [14] demonstrates 4–8$times$ improvement in energy efficiency. However, it is worth noting that even with the relatively lower precision of 5–6 bits, a suitable dynamic range (DR) and acceptable linearity need to be maintained. This has resulted in designs, where the supply voltage for mixed-signal circuits is limited to 1–1.2 V [11], [13]. On the other hand, algorithm-hardware co-design has resulted in efficient digital implementations of dynamic-accuracy-voltage–frequency-scaling (DAVFS) [2], wide and variable precision CNNs [15], [16], and even binary neural networks [15], [17], [18]. However, the applicability of binary networks in successfully solving the complex tasks needs to be further ascertained. References [19] and [20] have further demonstrated on-chip learning using neuromorphic sparse coding techniques. Thus, the landscape of energy-efficient ML has seen considerable advances, and in particular, voltage-based analog-/mixed-signal computing has recently gained well-deserved attention.

To address the problem of supply scalability in voltage-mode analog circuits, we propose a different approach, namely, encoding information and computing in the time domain. An earlier demonstration of time-domain compressive data conversion can be found in [21]. Time-domain mixed-signal (TD-MS) designs inherit the advantages of analog computing, including high energy efficiency at target bit resolutions. However, it also allows us to relax the supply voltage requirement since the DR is expressed in time. Our proposed time-based multiplication-and-accumulation (MAC) technique provides an energy-efficient alternative to digital implementation, exhibits seamless interfaces with digital memory circuits, and also demonstrates voltage and process scalability like digital logic. Fig. 2 shows the average compute energy for digital and time-based MAC implementations. We note that the proposed time-based implementation has better energy efficiency compared to the digital MAC operation for a bit resolution of less than 7 (30% lower at 6 b).

In spite of the success of RL in the algorithm front, the hardware demonstrations of RL are limited to early demonstrations in object recognition [22] and noise shaping [23]. However, for energy-constrained mobile robots that are used for surveillance and exploration, advances in RL algorithms need to be matched with efficient circuit and hardware designs. In this paper, which is a follow-up of [24], we demonstrate a neuromorphic accelerator for RL, which: 1) implements

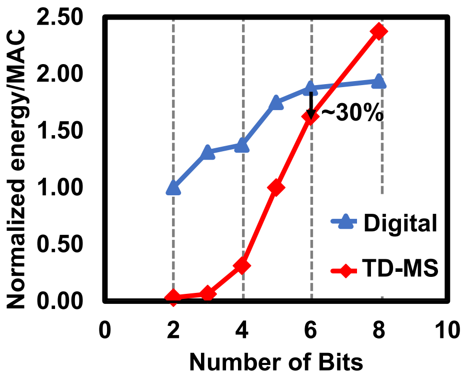


Fig. 2. Average computational energy/MAC for a digital and a time-domain implementation as a function of data resolution at $V_{CC} = 0.6$ V.

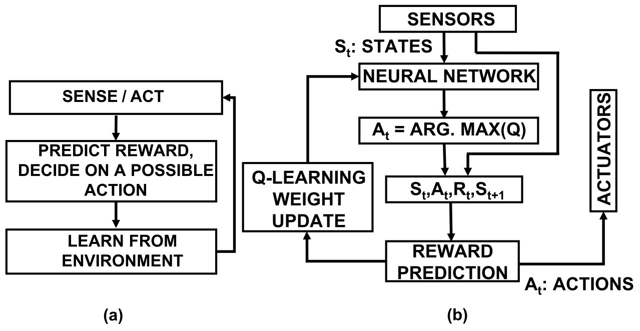


Fig. 3. (a) Basic steps in RL. (b) Q-learning as an implementation of RL that has been implemented in this test chip.

Q-learning using a fully connected (FC) neural network; 2) demonstrates energy-proportional computation using TD-MS circuits; and 3) uses a stochastic network of synapses to reduce data over-fitting. The proposed accelerator consumes 690 μW of peak power at 1.2 V. The accelerator can operate down to a low supply voltage of 0.4 V, making it a voltage-scalable mixed-signal neural network. Sections VII and VIII discuss measurement results and conclusion, respectively.

II. Basics of Reinforcement Learning

For a detailed overview of RL techniques, algorithms, and applications, interested readers are referred to [9]. In this section, we will provide a short overview of Q-learning that has been implemented in the current design. The problem space in RL consists of agents, states, actions, and cumulative rewards. The agent transitions between states (S_t and S_{t+1}) via an action A_t. From a neurobiology standpoint, RL finds similarities when compared to the operation of the basal ganglia in the human brain [25]. It has been observed that the cerebral cortex performs state/action representations, the striatum performs reward predictions, the palladium assists in action selection, and the dopamine neurons perform complex operations akin to the temporal difference method. Q-learning inherits these neuromorphic properties and continues to be a well-studied RL technique. This has been illustrated in Fig. 3(a) and (b), where the different parameters have been shown. Q-learning works on the principle of the action-value function $[Q(S_t, A_t)]$

that can implement an optimal action policy for a finite set of states. This is an off-policy learning technique [9] and has been shown to exhibit faster convergence compared to on-policy learning algorithms, such as SARSA [26]. The core of the algorithm is based on the iterative update of the Q value, as an agent navigates through a series of (state, action, reward) tupules. This iterative scheme is derived from the Bellman equation [27] for optimal control. The iterative algorithm can be summarized as

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha(R_t + \gamma \max Q_t(S_{t+1}, A_t) - Q_t(S_t, A_t)) \quad (1)$$

where γ is the discount factor and α is the learning rate. α is reduced from 1 to 0 in the steps of 0.001. γ is used to prioritize the rewards; immediate rewards are given higher priority, whereas the distant rewards are discounted. It is easy to understand that the size of the state-action space grows as $(S \times A)$. To enable real-time Q-learning, one can store the Q values corresponding to the state, action, reward, and next state in a lookup table. However, with a growing $(S \times A)$, a huge memory capacity is required. Therefore, the Q value is typically approximated as a neural network output [10]. The sensor provides the states (S_t). States act as inputs to the neural network. The neural network provides the Q values corresponding to the set of possible actions. In our demonstration system, we use three ultrasonic sensor inputs, and the mobile robot can move in three possible directions (left, center, or right). Once the maximum argument of the Q value is determined, the robot moves in that particular direction with a finite probability, ϵ. The value of ϵ is fixed in the current design. By taking a series of actions in the state space, the robot calculates the reward for each action and trains the neural network via backpropagation, thus creating a robust functional mapping from the state space to the action space. The principal steps carried out by the neural network constitute the following.

1) For the current state S_t, we perform a forward pass of the neural network to obtain the Q values for all actions.
2) We take an action, A_t performs a forward pass for the next state, S_{t+1}, and calculate maximum overall network outputs $\max_{A_{t+1}} Q(S_{t+1}, A_{t+1})$.
3) We set the target Q value as a sum of the immediate reward, R_t and $\gamma\, max_{A_{t+1}} Q(S_{t+1}, A_{t+1})$, using the maximum value calculated in the previous step.
4) We define a loss function based on the target and update the model weights using backpropagation.

We will describe mapping of the algorithm to the current hardware design in Section V.

III. System Components and Architecture

In this section, we provide a brief overview of the system components and the system architecture.

A. Ultra-Sonic Sensors as State Estimates

We use ultra-sonic sensors for measuring the distance of obstacles. The ultra-sonic sensors receive periodic trigger signals from an associated raspberry PI-based micro-controller. The sensor emits a sonic signal in response to the trigger signal. The echo pulse is raised to high at the end of the sonic burst. Once the reflected wave is received, the echo pulse is pulled down. The pulsewidth of the echo signal is proportional to the distance from the obstacle. The distance from obstacle is evaluated as

$$d = \frac{t * c}{2} \quad (2)$$

where d is the distance from the obstacle, c is the velocity of sound in air, and t is the pulsewidth of the echo pulse. In traditional digital processing, the echo pulse needs to be converted to a digital value. Hence, it requires a pulse-to-digital converter. However, in the current TD-MS design, the neural network can directly process the pulse-based echo signal, thus reducing both the latency and energy of analog (time)-to-digital conversion. Our current system uses three ultra-sonic sensors, and the vector of the three distances $[d_{\text{left}}, d_{\text{center}}, d_{\text{right}}]$ defines the state space. The left and right sensors are placed at an angle of 30° with respect to the center.

B. System Architecture

Fig. 4 shows the system architecture and illustrates four main components: 1) three ultra-sonic sensors; 2) the proposed RL test chip; 3) a Raspberry PI-based micro-controller; and 4) motor drivers and motors for each of the four wheels. The ultrasonic sensors are used to find the distances from three directions, as described in Section II, and are periodically clocked by the Raspberry PI. The echo signals from the ultrasonic sensors are fed to the test chip and the Raspberry PI. During RL training, the Raspberry PI stores (S_t, A_t, R_t, S_{t+1}) in a scratchpad memory and feeds the data to the test chip. The test chip first performs inference (from sensed states to actions). Next, it updates the reward-based neural-network model via backpropagation and gradient descent. The test chip sends a 2-bit control word indicating one of three possible actions (*move left*, *move forward*, or *move right*) to the Raspberry PI, which in turn sends the control command to the motor controllers.

C. Neural Network Implementation

A FC, three-layered neural network provides a functional mapping from the state space to the Q values of the action-space. The first layer obtains pulse-domain data from the three ultra-sonic sensors ($D = [d_{\text{left}}, d_{\text{center}}, d_{\text{right}}]$) as well as a bias value for the input layer ($BIAS$). The inner products of (D) and the weights from the input layer to hidden layer (w^{IH}) are computed in the time domain and accumulated in a 15-bit counter. A digital word, consisting of the first seven MSBs of the counter, is fed to a digital-to-pulse converter (DPC). The DPC natively implements a rectified linear unit (ReLU)-based activation function and produces neuron outputs as pulses. We choose ReLU, as opposed to other activation functions, because of faster and more stable convergence. The outputs to the DPC represent the outputs of the hidden layer of neurons (H)

$$H = ReLU(D * w^{IH}). \quad (3)$$

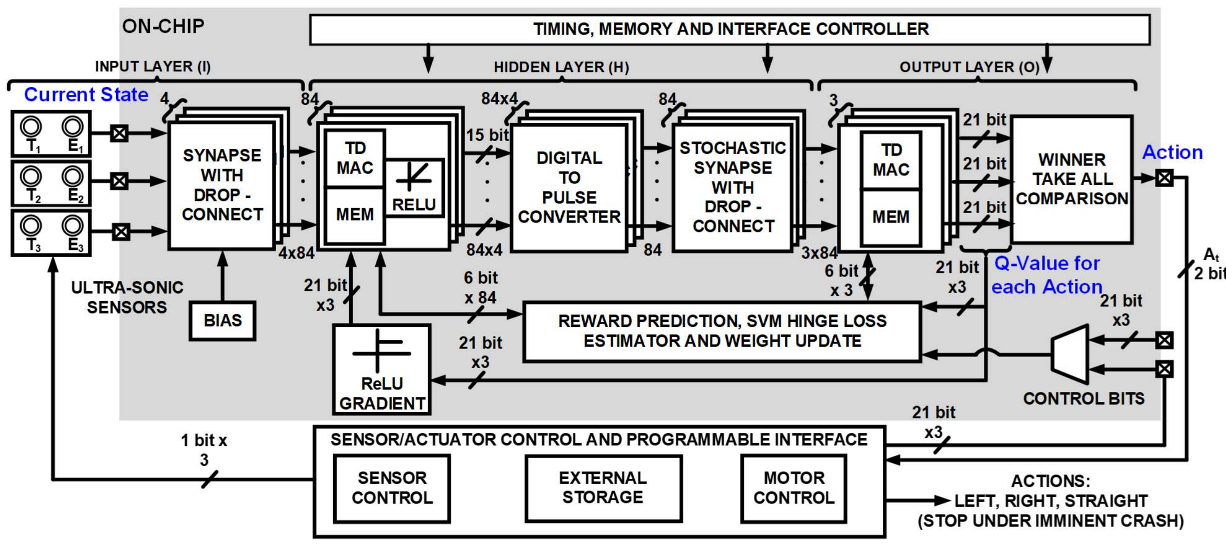

ON-CHIP
TIMING, MEMORY AND INTERFACE CONTROLLER
INPUT LAYER (I)
HIDDEN LAYER (H)
OUTPUT LAYER (O)
Current State
SYNAPSE WITH DROP - CONNECT
TD MAC
MEM
RELU
DIGITAL TO PULSE CONVERTER
STOCHASTIC SYNAPSE WITH DROP - CONNECT
WINNER TAKE ALL COMPARISON
Action
Q-Value for each Action
ULTRA-SONIC SENSORS
BIAS
ReLU GRADIENT
REWARD PREDICTION, SVM HINGE LOSS ESTIMATOR AND WEIGHT UPDATE
CONTROL BITS
SENSOR/ACTUATOR CONTROL AND PROGRAMMABLE INTERFACE
SENSOR CONTROL
EXTERNAL STORAGE
MOTOR CONTROL
ACTIONS: LEFT, RIGHT, STRAIGHT (STOP UNDER IMMINENT CRASH)
1 bit x 3
21 bit x3
6 bit x 84
84x4
15 bit
4x84
3x84
6 bit x 3
21 bit
A_t 2 bit

Fig. 4. System architecture illustrating the different circuit blocks and the interface to the external micro-controllers (Raspberry PI) and motor drivers.

The number of neurons in the hidden layer is 84. In our demonstration platform, we simulated various grid-world examples, and for a grid world of 10^4 nodes, we found that 84 neurons in the hidden layer performed satisfactorily both in terms of convergence speed and accuracy. More complex environments with more complex dynamics (such as moving obstacles) will require a larger number of neurons and hidden layers. Furthermore, it is worth mentioning that navigation via ultra-sonic sensors has limited accuracy and is also computationally less intensive. At-scale systems that can navigate using camera-based images require significantly larger neural networks models. However, the current prototype demonstrates an RL system enabled by scalable hardware primitives, capable of learning to navigate in a real environment, albeit with limited complexity. The outputs of the hidden layer (H) are multiplied by the synaptic weights of the hidden-to-output layer (w^{HO}) to produce the final output (Q) of the neural network. We can express this as

$$Q = H * w^{HO}. \tag{4}$$

The output (Q) consists of three Q values corresponding to the three directions for the motor control (Q_{left}, Q_{center}, and Q_{right}). The argument of the maximum Q value among the three outputs determines the direction of motor control (A_t) via a winner-take-all (WTA) circuit. Thus, the agent can move either left, forward, or to the right. The flow of input pulses from the ultra-sonic sensors to the hidden layer outputs and finally to the output of the network is sequential. A_t is encoded in a 2-bit word and sent off-chip to the Raspberry PI for motor control. We use a scan chain to initialize the neural network weights to random values. Alternatively, the neural network can be initialized to pre-defined weights obtained from simulations using transfer learning [9].

IV. TD-MS Circuit Architecture and Design

In this section, we introduce the design of TD-MS circuits and corresponding circuit architectures.

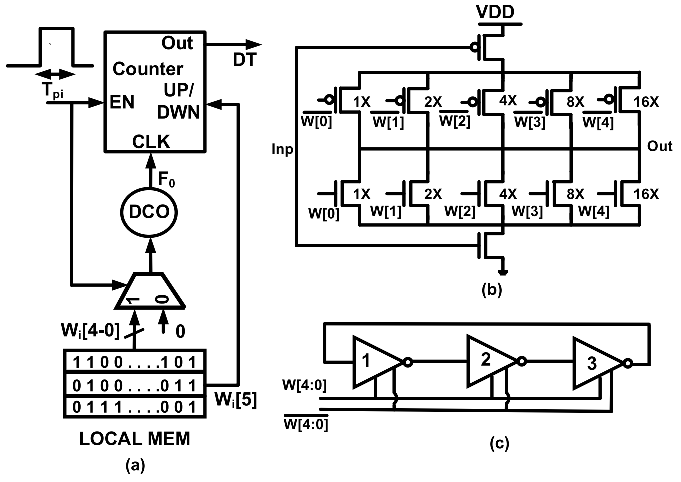


Fig. 5. (a) Proposed TD-MS MAC unit. (b) DCO unit cell. (c) Three-stage DCO design.

A. Multiply-and-Accumulate in a TD-MS Architecture

Fig. 5(a) illustrates the proposed time-based MAC circuit. It has a pulse input (T_{p_i}) used as the "Enable" signal to an up–down counter. This pulse is obtained from the ith ultrasonic sensor ($i = 1, 2, 3$) in case of the hidden layer neurons or from the ith hidden layer neuron ($i = 1, 2, \ldots, 84$) in the case of the output layer. We store the synaptic weights of all the fan-ins locally at each neuron. The weight is a 6-bit value expressed in signed-magnitude format, where the MSB is the sign bit. The five LSBs (W[0:4]) are connected to a multiplexor. T_{p_i} controls the select bit of the multiplexor. If the digital pulse is high, the 5-bit word is passed through the multiplexor, or else, a 0 is passed to a digitally controlled oscillator (DCO), thus creating a gated-DCO. The three-stage DCO converts the digital value to a frequency proportional to W[0:4]. Each stage of the DCO consists of a bank of parallel binary-sized inverters controlled by the digital value (W[0:4]), as shown in Fig. 5(b) and (c). The frequency of the DCO for the ith word (W_i), ignoring the second-order effects, such as non-linearity, is given by

$$F_{W_i} = W_i * F_0 \tag{5}$$

where F_0 is the unit frequency of the DCO corresponding to a $code1$ when $W = 00001$. The clock to the counter is driven by the DCO, and the enable signal is controlled by the pulsewidth (T_{p_i}). Hence, the counter output is given by

$$DT_i = F_{W_i} * T_{p_i} = W_i * F_0 * T_{p_i}. \tag{6}$$

From (6), we can observe that the counter output is proportional to the product of the weight (W_i) and the pulsewidth (T_{p_i}) of the gating signal. This is a true mixed-signal multiplier, where one input is a pulse (analog) and other input is a digital value stored in the local memory. The counter provides accumulation natively. In the current design, all the pre-synaptic neurons (fan-ins) send pulses (T_{p_i}) one after another. For each of these pulses, the TD-MS MAC loads the corresponding synaptic weight (W_i) and computes $W_i * T_{p_i} * F_0$. For N such pre-synaptic pulses, at the end of an update, the counter output represents

$$DT = \sum_{i=0}^{N} DT_i = \sum_{i=0}^{N} W_i * T_{p_i} * F_0. \tag{7}$$

Before the beginning of the update, the counter is reset to all zeros expect for the MSB that is reset to 1. This allows the counter to count up (for $W_i > 0$) or count down (for $W_i < 0$). At the end of the update cycle, the sign of the MSB indicates whether $DT > 0$ or $DT < 0$, a key threshold that is subsequently used by the non-linear activation function, which will be discussed in Section V. It should also be mentioned that the enable signals for the counter and the local DCO clock are asynchronous and can lead to a 1 LSB of error due to a metastable capture. We did not find this to be a significant issue either in simulations or in measurements. However, if this is a concern, clock-crossing circuits can be used to synchronize between the two signals and prevent any metastable capture.

B. Digital-to-Pulse Converter-Based Non-Linear Activation

After the linear MAC operation, a DPC is used to implement the non-linear activation function of the neuron. The schematic of the DPC is shown in Fig. 6(a). It consists of a digital-to-time converter (DTC) followed by a phase–frequency detector (PFD). It receives a 7-bit input (DT), which is obtained from the output of the TD-MS MAC. The DTC consists of cascaded delay chains, as shown in Fig. 6(b). Fig. 6(c) shows the schematic of the PFD. The PFD receives clock signal (CLK) and a delayed version of CLK (CLKD) as inputs. Each delay chain consists of two paths: one path directly feeds the input to the output through NOR-based buffers, and another path transmits the input through a bank of binary-sized inverters, as shown in Fig. 6(d). The buffers are sized to produce binary-weighted delays that scale as 2^i for $i = 0$–6. The control word for the DPC is DT (output of the MAC), whose MSB is 1 when the accumulated value of the MAC is non-negative or 0 otherwise. We use the MSB to gate all the lower bits before they are applied as a control word to the DTC, as shown

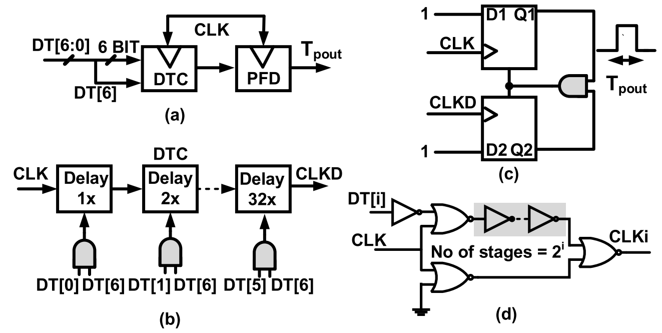


Fig. 6. (a) Circuit implementation of the DPC illustrating (b) cascade of delay elements, (c) PFD, and (d) delay cell for DT[i] input [31].

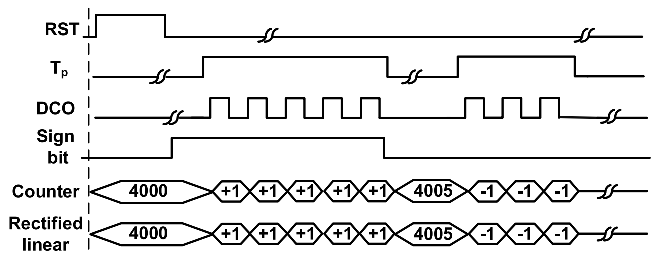


Fig. 7. Timing diagram for the time-domain MAC logic illustrating accumulation over two cycles, one with positive synaptic weight and one with a negative synaptic weight.

in Fig. 6(b). As a result, the DPC produces an output given by

$$T_{p_{\text{out}}} = \begin{cases} 0 & \text{if } DT \le 0 \\ DT * T_0 & \text{if } DT > 0. \end{cases} \tag{8}$$

This shows that the final output ($T_{p_{\text{out}}}$) of the TD-MS neuron implements an ReLU activation function

$$T_{p_{\text{out}}} = ReLU(DT). \tag{9}$$

To generalize, in a cascaded design where the output of one pre-synaptic neuron, i, drives the input of the postsynaptic neuron, j, the time-domain pulses, T_{pij}, represent propagation of information through the network. In ML literature, this is typically denoted by X_{ij}. Here, $T_{pij} = X_{ij} * T_0$. The synaptic weight between the two neurons is W_{ij}. By combining (7) and (8), we can write down the output of the jth neuron ($T_{p_{\text{out}j}}$) simply as

$$T_{p_{\text{out}j}} = T_0 * ReLU\left(\sum_j W_{ij} * X_{ij} * F_0 * T_0\right). \tag{10}$$

Equation (10) illustrates that the TD-MS neuron can successfully implement an ReLU-based neuron and encodes both the inputs from pre-synaptic neurons and output in time domain. It should be noted that the portion of the equation within the parenthesis is dimensionless. An example of the timing diagram is shown in Fig. 7, where the cascaded operation of the DCO, the counter-based accumulator, and the output of the ReLU are shown with specific values. The above-mentioned

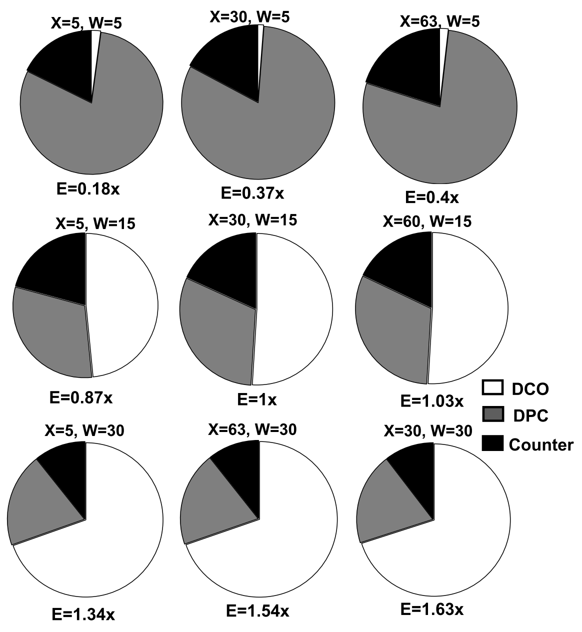


Fig. 8. Breakdown of the energy to compute for TD MAC across various input and weight codes. The energy is normalized with respect to a centrally placed code.

treatment captures the first-order effects only. Second-order effects, such as non-linearity of DCOs, lead to error in calculating the MAC, which can lead to loss of accuracy. However, RL algorithms like many other online ML techniques are forgiving to such non-idealities, albeit within limits.

C. Energy and Power Analysis of the Proposed TD-MS Design

For the application in hand, sensor data (state estimates) are acquired at regular intervals. Hence, the amount of compute needed per unit time is approximately constant, depending on the speed of the mobile robot and the complexity of the environment. Hence, we wish to compute at the lowest possible energy while being cognizant of the range and resolution of the data. TD-MS designs provide some unique characteristics, as we describe here. From simulations on the 55-nm technology node at $V_{CC} = 0.6$ V, we explore the energy consumed by the various blocks (after synthesis with tight constraints on power and area) in the time-based neuron and plot them in Fig. 8. Since the number of switching events in the TD-MS design depends on the magnitude of the operands (X_{ij} and W_{ij}), the energy consumed also shows a strong dependence on X_{ij} and W_{ij}. The nine pie charts (for W = 5, X = 5 to W = 30, X = 63) illustrate the normalized energy per MAC. For lower values of the DCO code ($W_{ij} < 15$), the energy consumption is dominated by the DPC. Subsequently, the DCO power becomes dominant when the DCO code increases beyond 15. Furthermore, with the increasing magnitudes of the operands, the total energy to compute a MAC also increases. This is an interesting design choice because of the two observations specific to the algorithm. First, larger operands correspond to more important parts of computation in the network and typically represent the flow of critical information. Second, in trained neural networks, the magnitudes of the weights tend to cluster around low values. We take advantage of both of these observations in TD-MS designs because of its energy scalability with the importance of the computation, which is a bio-mimetic approach. We capture this in Fig. 9 by plotting the energy to compute as a function of the input (X_{ij}) and the weight (W_{ij}). In binary-coded arithmetic, we do not see any such trend since the number of switching events is not correlated with the magnitudes of X_{ij} or W_{ij}. It is worth noting that a digital design is expected to produce higher performance than a TD-MS design, which inherently uses time for data encoding. However, as long as the time required to complete the data processing is acceptable, as is the case in this application, TD-MS provides a bio-mimetic and energy-efficient approach. We note that on an average, where all the combinations of X_{ij} and W_{ij} are equally likely, TD-MS-based-MACs spend 30% lower energy at iso-V_{CC} compared to an array-based digital MAC. We also compare the post-synthesis area of the TD-MS design vis-à-vis a digital design and observe: 1) 45% lower system area and 2) 47% lower interconnect power. The lower interconnect power comes from the fact that the outputs of the neurons travel through single bit interconnects and undergo only one $1 \hookrightarrow 0$ and one $0 \hookrightarrow 1$ transition. This saves both leakage and switching power compared to a bit-parallel digital design, albeit at the cost of performance. This also consumes less dynamic energy compared to a bit-serial digital interconnect, where a 6-bit word will typically undergo multiple transitions. The compact area of the TD-MS design is further reflected in Fig. 11, where the total simulated leakage power of TD-MS and digital designs across three temperatures 0 °C, 50 °C, and 100 °C corners are shown. On an average, the leakage power in TD MAC is 35% less compared to the digital implementation.

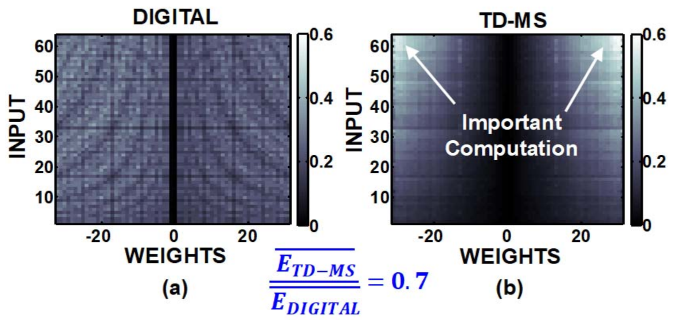


Fig. 9. 2-D plot of the energy surface illustrating the energy/MAC for (a) digital implementation and (b) TD-MS design with 6-b inputs (X and W) at $V_{CC} = 0.6$ V. We have note that more energy is consumed for important operations, where $|X|$ and $|W|$ are large. The average energy/MAC for the TD-MS implementation is 0.7× that of the digital implementation.

D. Time-Domain Stochastic Synapses

It is well studied in the ML community [28] that the introduction of stochasticity in synaptic connections avoid over-fitting of data. Along with stochasticity, drop-connect is also used to eliminate the random synaptic connections during training. Both of these regularization techniques assist

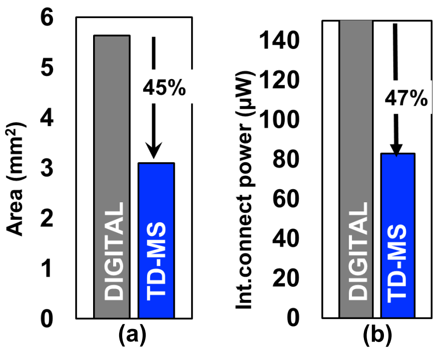


Fig. 10. Results from the synthesis and PnR of the complete design show that the TD-MS consumes (a) 45% lower area and (b) 47% lower interconnect power, compared to a digital design.

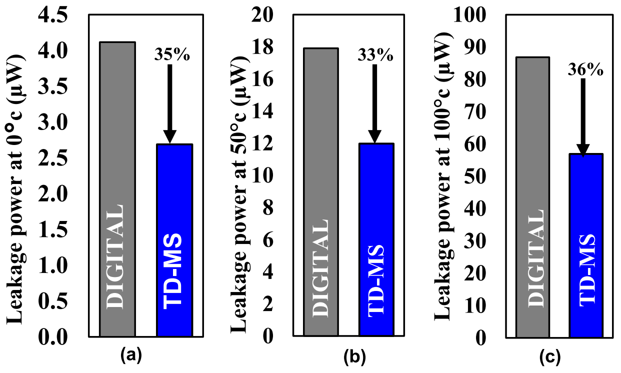


Fig. 11. Simulated post-synthesis leakage power at three different temperatures: (a) 0 °C, (b) 50 °C, and (c) 100 °C for the digital and TD-MS design.

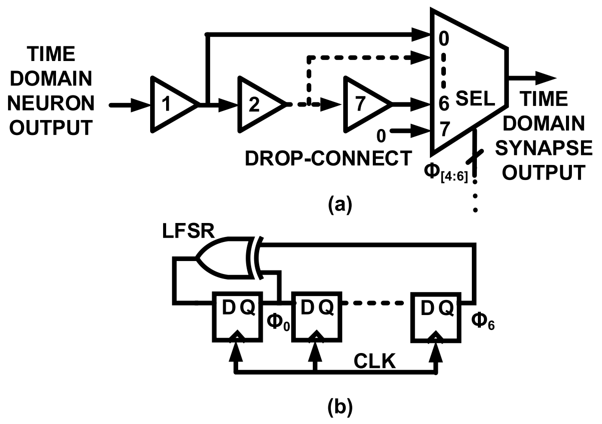


Fig. 12. Stochasticity and drop-connect are implemented by intentionally introducing randomly varying delays between the $1 \hookrightarrow 0$ and the $0 \hookrightarrow 1$ transitions. The randomness is introduced by (a) LFSR that drives (b) MUX-select.

in generalizing the learned neural network model to unknown environments. Here, we introduce stochasticity in the synaptic connections in the time domain, as shown in Fig. 12(a). High-speed local linear-feedback-shift registers (LFSRs) drive multiplexers that tap various nodes of a synaptic buffer chain. A part of the buffer chain provides a fixed delay, and a part of the buffer chain has a stochastic delay. Stochasticity is introduced by changing the delay between the $1 \hookrightarrow 0$ and the $0 \hookrightarrow 1$ transitions. Fig. 12(b) depicts the LFSR circuit and how the multiple output bits enable stochasticity as signals propagate. Since these synaptic connections propagate X_{ij}, the variable delay introduces controlled randomness in the operand X_{ij}. Furthermore, for a fixed LFSR code, the multiplexer selects a 0, thereby preventing any signal propagation and enabling drop-connect.

E. Information Flow in the Feed-Forward Direction

The timing diagram for information flow in the feed-forward network is shown in Fig. 13. The input signals from the ultrasonic sensors (E_1, E_2, and E_3) are processed in a sequential manner; 84 TD-MS neurons perform MAC operations with weights ($W^{IH}_{1,1}$–$W^{IH}_{1,84}$) and the E_1 pulse, all in parallel. Next, for the E_2 pulse from the ultrasonic sensor, the next set of 84 weights ($W^{IH}_{2,1}$–$W^{IH}_{2,84}$) is used. This is followed by the pulse from E_3. Next, the time-domain outputs of the 84 neurons in the hidden layer are generated sequentially. The pulse generated by the first neuron of the hidden layer is multiplied by $W^{HO}_{1,1}$, $W^{HO}_{1,2}$, and $W^{HO}_{1,3}$ in parallel at the three output neurons, respectively. Next, the output pulse of the second neuron of the hidden layer is generated, multiplied by $W^{HO}_{2,1}$, $W^{HO}_{2,2}$, and $W^{HO}_{2,3}$ in parallel, and accumulated. This continues till all the 84 neurons of the hidden layer have fired. Finally, the outputs of the three output neurons are generated, and they encode the Q values corresponding to the three actions. A WTA decides on the most favorable action and sends a 2-bit command to the Respberry-PI, which in turn enables the corresponding motor controls. The test chip runs on a system clock, which maintains synchronicity among sensor activation, data flow, memory read operations, and output generation.

V. System Architecture for Learning via Backpropagation

In Section IV, we have described data flow in the forward direction. For online RL, we perform backpropagation using gradient descent. Backpropagation involves the following discrete steps: 1) evaluation of the output gradients (GD_O); 2) evaluation of the gradients at the hidden layer (Δ_H); 3) evaluation of the gradients for the weights from the input to the hidden layer (w^{IH}) and from the hidden to the output layer (w^{HO}); and 4) updating all the weights.

The output gradient is computed using a support vector machine (SVM) hinge-loss function. Fig. 14(a) illustrates the steps implemented in the current system. (GD_0) takes two cycles. Let us denote the Q value corresponding to the label stored in the state-action table by Q_{Y_I}. Here, Y_I is the action corresponding to maximum Q value. Q_{Y_I} can be expressed as

$$Q_{Y_I} = R_t + \gamma \cdot \max(Q(S_{t+1}, A_t)). \quad (11)$$

Next, the SVM hinge-loss is evaluated as

$$L_j = \sum_{j \neq Y_I} \max(0, Q_j - Q_{YI} + 1). \quad (12)$$

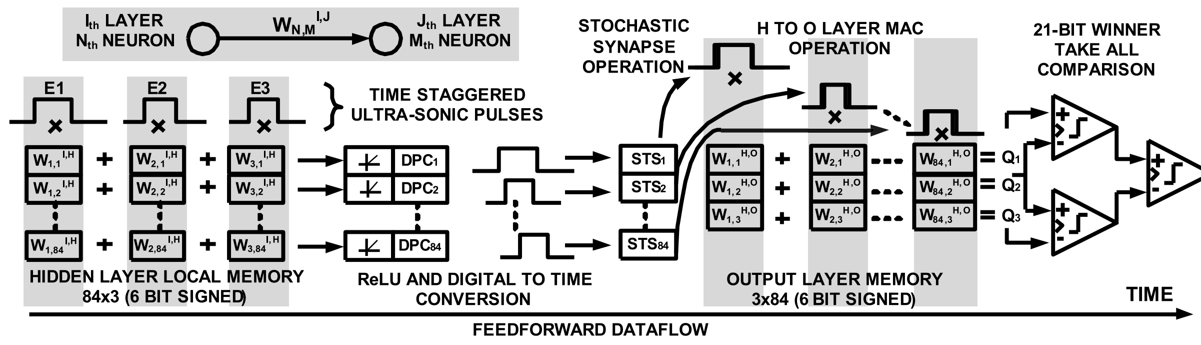

Ith LAYER Nth NEURON
Jth LAYER Mth NEURON
TIME STAGGERED ULTRA-SONIC PULSES
STOCHASTIC SYNAPSE OPERATION
H TO O LAYER MAC OPERATION
21-BIT WINNER TAKE ALL COMPARISON
HIDDEN LAYER LOCAL MEMORY 84x3 (6 BIT SIGNED)
ReLU AND DIGITAL TO TIME CONVERSION
OUTPUT LAYER MEMORY 3x84 (6 BIT SIGNED)
TIME
FEEDFORWARD DATAFLOW

Fig. 13. Timing diagram for the feed-forward neural network illustrating the parallel and sequential components of the data flow.

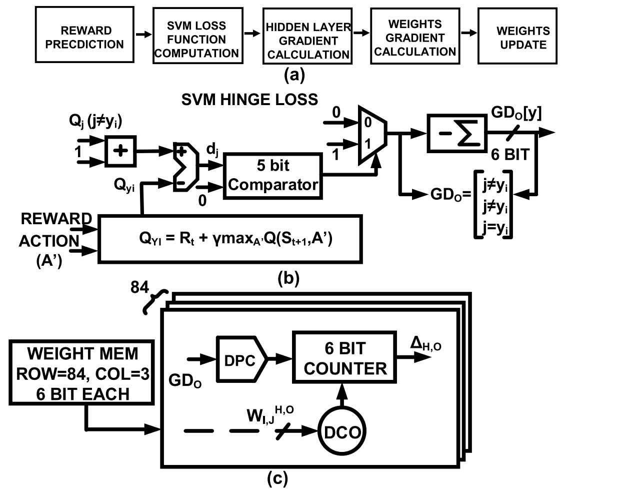

REWARD PRECDICTION
SVM LOSS FUNCTION COMPUTATION
HIDDEN LAYER GRADIENT CALCULATION
WEIGHTS GRADIENT CALCULATION
WEIGHTS UPDATE
(a)
SVM HINGE LOSS
5 bit Comparator
6 BIT
REWARD
ACTION (A')
(b)
WEIGHT MEM ROW=84, COL=3 6 BIT EACH
6 BIT COUNTER
(c)

Fig. 14. (a) Flowchart showing the different components of backpropagation. (b) Implementation of SVM hinge-loss estimation during backpropagation. (c) Schematic of the circuit that implements the update of the hidden layer gradients ($\Delta^{H,O}$).

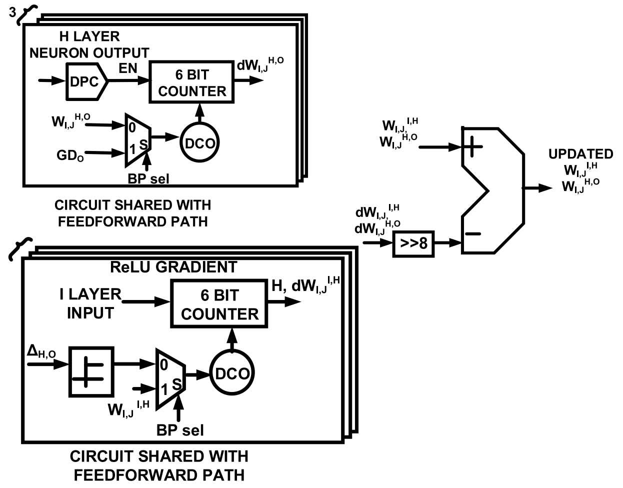


Fig. 15. Circuit components showing the process of weight update across layers during backpropagation.

The gradient of loss for (Y_I) is calculated using [29]

$$GD(Y_I) = \sum_{j \neq Y_I} ((Q_j - Q_{Y_I} + 1) > 0?1 : 0). \quad (13)$$

Finally, we calculate the gradient of loss for $(j \neq Y_I)$ as

$$GD(j) = ((Q_j - Q_{Y_I} + 1) > 0?1 : 0). \quad (14)$$

It takes two cycles to calculate the gradient. Fig. 14(b) depicts the schematic of the corresponding circuit. Q_j is added with 1 and subtracted from Q_{YI}. The subtracted value is compared with 0. The comparator output is used as the select bit for selecting 0 or 1 in the accumulator. Hidden layer gradients are computed using the dot product between the output gradient (GD_0) and w^{HO}. This is illustrated in Fig. 14(c). It can be mathematically written as

$$\Delta^{H,O} = GD_O * w_{I,J}^{H,O}. \quad (15)$$

The 84 parallel units of DCOs and DPCs are used for calculating the gradients of the hidden layer. Fig. 15 shows how the gradient of the hidden to the output layer is calculated. The hidden layer output is generated from the feed-forward iteration and multiplied with the output gradients to finally obtain the gradient of the hidden to the output layer weights, dW^{HO}

$$dW^{H,O} = H * w_{I,J}^{H,O}. \quad (16)$$

Next, the hidden layer gradients are passed through the gradient of the ReLU and multiplied with the input signals to compute dW^{IH}. The circuit that implements dW^{IH} is schematically shown in Fig. 15. dW^{IH} is obtained as

$$dW^{I,H} = X * ReLU_G(\Delta^{H,O}). \quad (17)$$

Finally, the update rules for w^{IH} and w^{HO} can be expressed as

$$w^{H,O} = w^{H,O} - \frac{dW^{H,O}}{8} \quad (18)$$

$$w^{I,H} = w^{I,H} - \frac{dW^{I,H}}{8}. \quad (19)$$

The design of hardware-based backpropagation uses linear algebraic kernels that are similar to the feed-forward path. This allows us to share hardware resources. As a matter of fact, the 84 neurons at the hidden layer, consisting of DCOs and DPCs, are fully programmable to compute both in the feed forward (inference) as well as the feedback (backpropagation and gradient-descent) directions.

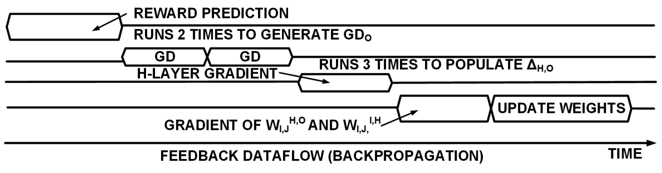

REWARD PREDICTION
RUNS 2 TIMES TO GENERATE GD$_O$
GD
GD
RUNS 3 TIMES TO POPULATE Δ$_{H,O}$
H-LAYER GRADIENT
UPDATE WEIGHTS
GRADIENT OF W$_{I,J}^{H,O}$ AND W$_{I,J}^{I,H}$
FEEDBACK DATAFLOW (BACKPROPAGATION)
TIME

Fig. 16. Timing diagram for backpropagation during RL.

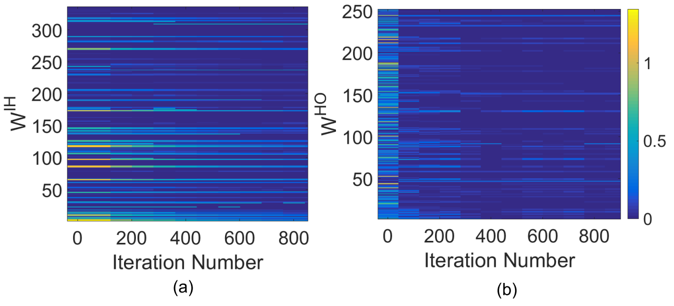


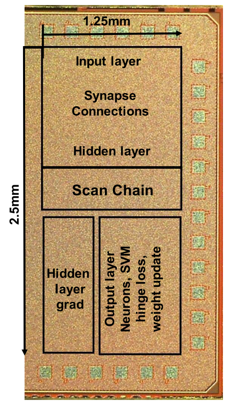
Fig. 17. 2-D surface plot illustrating the process of learning (weight update) for (a) input to the hidden layer weights w^{IH} and (b) hidden to the output layer weights w^{HO}. The model weights are shown in column vectors, and their evolution in time (number of iterations) is shown by color coding.

A. Data Flow During Backpropagation

Fig. 16 illustrates the timing diagram for performing backpropagation and gradient descent. The principal algorithmic steps have been shown here. After each update, the corresponding Q values and hidden layer weights are updated in each registers. Fig. 17 shows an example of how the weights are updated during RL. Here, the entire weight vector is shown as a vertical column, and its evolution with time (or iteration number) is plotted. We note that as the system learns from its environment, the weights settle down to optimal value. It can also be noted that the magnitude of the majority of the weights is low, which has been discussed earlier in Section IV-C.

VI. Design Flows for TD-MS Circuits

One of the main advantages of TD-MS circuits is our ability to map them to the traditional synthesis and place-and-route (PnR) flows. As an example, we first design one stage of the DCO, characterize it, and create the corresponding physical design. Next, we design the entire multistage DCO for target speed and DR. One of the takeaways from the exercise was that the entire three-stage DCO is needed to be presented as a single block to the PnR flow, to avoid mismatches among the VCOs and timing closure. Similarly, the DPCs are characterized as single macros and used in the back-end flows. By doing so, we minimize the within-die variation, which results in better tracking of T_0 and F_0. Functional verification during synthesis is completed using behavioral RTL for the digital blocks, clock circuits, and structurally modeling the DCO and DPCs. Once functional verification is complete, we combine the structural RTL with the rest of the digital blocks. This allows us to use back-end tools for the final design closure. TD-MS circuits use digital gates, which allow us to use both front-end and back-end tool chains as long as the mixed-signal blocks synthesized and laid out as distinct macros.



Chip Characteristics	
Technology	55nm 1P8M CMOS
Die area	1.25mm *2.5mm
Testing Interface	QFN package
Pin Count	24

Fig. 18. Chip micro-graph and characteristics.

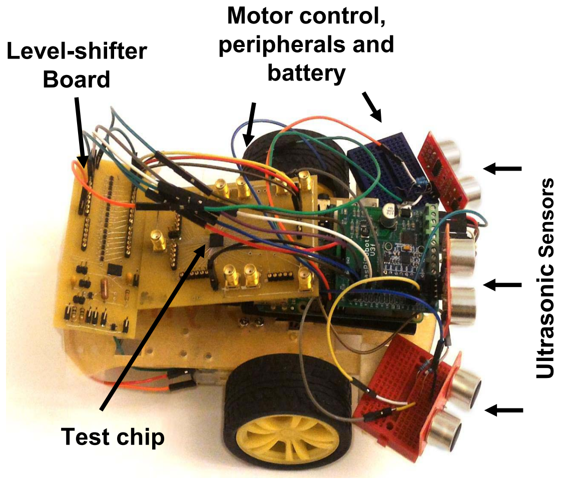


Fig. 19. Overall system of the mobile robot.

VII. Measurement Results

The test chip is fabricated in a 55-nm CMOS process and occupies an active area of 1.25 mm × 2.25 mm. The chip characteristics are shown in Fig. 18. Before going into the measurement results from the test chip, let us define the application space. We use the test chip in a mobile robot that can perform obstacle avoidance in a limited space. The design is scalable and can be extended to more complex tasks and more complex obstacle maps. The reward function $[R(t)]$ is defined as follows:

$$R_t = \begin{cases} -100 & \text{if the robot collides} \\ 30 - (d_{\text{left}} + d_{\text{center}} + d_{\text{right}}) & \text{otherwise.} \end{cases} \tag{20}$$

Initially, the robot explores the environment by taking random steps and updates the weights of its neural network (exploration phase). Progressively, the learning rate decreases, and the robot harnesses its knowledge and experience to make optimal decisions on the fly (exploitation phase). Fig. 19 depicts the system setup with the test chip along with the Raspberry PI and the motor drivers. We use an external MEMS oscillator for clock generation on the board. Level shifters are used at the interface between the test chip (1-V supply) and the Raspberry PI (3.3-V supply).

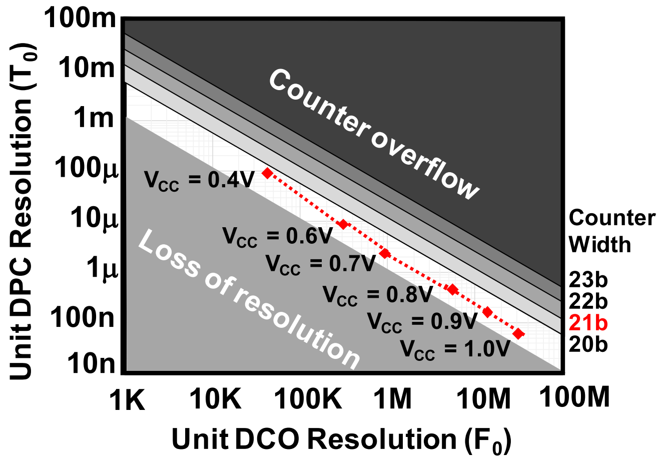


Fig. 20. Design space of DCO and DPC showing scalability till 0.4 V.

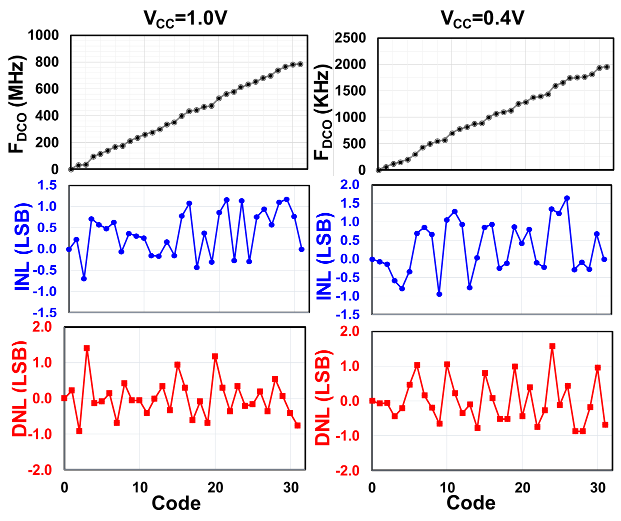


Fig. 21. Measured linearity of the DCO.

TABLE I

POWER CONSUMED BY DIFFERENT SYSTEM COMPONENTS

Component	Current (A)	Voltage (V)	Power (W)
Ultrasonic sensor	0.015	5	0.075
Motor Drivers	1.2	6	7.2
Raspberry-PI	2.5	5	12.5
Test-chip	0.575m	1	0.69m

To functionally characterize the test chip, we need to understand how the design space is constrained by the two fundamental design parameters, F_0 and T_0. This is shown in Fig. 20. If the DCO runs too fast (high F_0), the counters overflow. If the DCO runs too slowly (low F_0), then we lose resolution in the MAC units. For the current design, we choose a 21-b counter, and the measured results show that the F_0 and T_0 track each other over a wide range of supply voltages. This provides correct operation as well as voltage scalability from 1.0 down to 0.4 V. We suspect that the test chip ceases to function at 0.4 V when the local register and memory circuits start to fail.

The measured frequency of the DCO, F_{DCO} (of the hidden layer) shows peak performance of 780 MHz (at 1 V). Fig. 21 shows the measured linearity of the DCO. We measure a peak integral non-linearity (INL) of 1.2 LSB (1.6 LSB) at 1 V (0.4 V) and a peak differential non-linearity (DNL) of 1.5 LSB (2 LSB) at 1.0 V (0.4 V). Fig. 22 shows the measured linearity of the DPC. The measured T_{DPC} exhibits

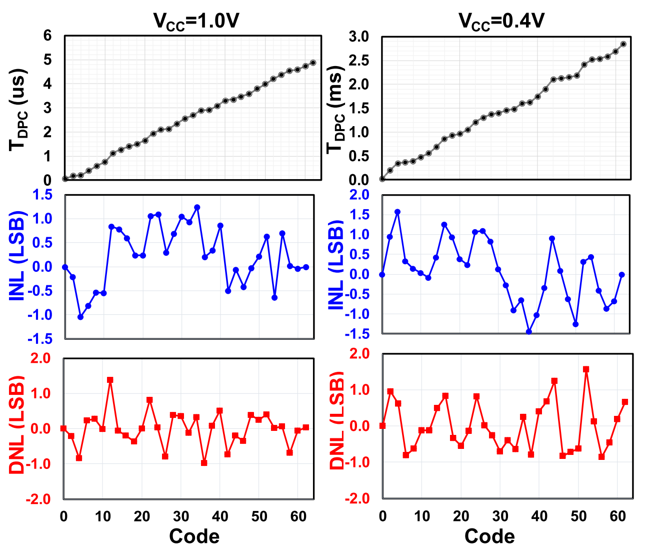


Fig. 22. Measured linearity of the DPC.

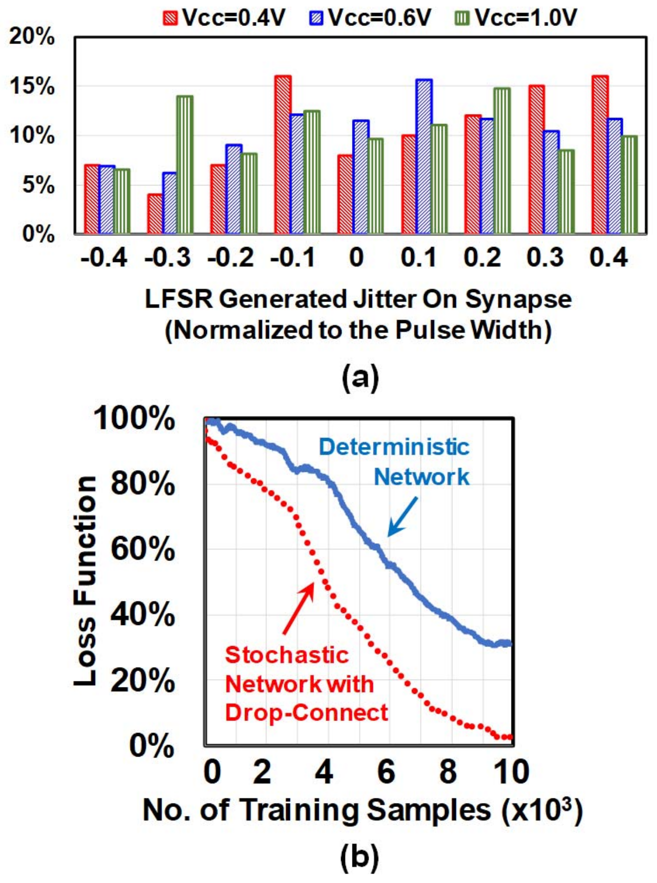


Fig. 23. (a) Distribution of stochasticity on the synaptic connections as measured through the jitter on a mean synaptic pulse at V_{CC} of 0.4, 0.6, and 1.0 V. (b) Role of stochasticity on RL illustrating faster convergence compared to a deterministic network.

TABLE II
COMPARISON WITH THE OTHER HARDWARE IMPLEMENTATIONS OF NEURAL NETWORKS

	This work	[22]	[3]	[2]	[4]	[1]	[20]	[19]	[30]
ML System	Reinforcement Learning	Object Recognition	CNN-RNN	CNN	DNN	DNN	Spiking LCA	Spiking LCA	DNN
Database	Sensor states	ImageNet	ImageNet speech	ImageNet	MNIST/ CIFAR	ImageNet	MNIST	MNIST	MNIST
Technology	55nm	65nm	65nm	180nm	65nm	65nm	65nm	65nm	28nm
Circuit style	Time domain Mixed-signal	Digital	Digital	Digital	Digital	Digital	Analog	Digital	Digital
Area	$3.4mm^2$	$4mm^2$	$16mm^2$	$3.3mm^2$	$16mm^2$	$16mm^2$	$1.3mm^2$	$1.85mm^2$	$5.76mm^2$
Learning/ Training	Online in real time	Offline	Offline	Offline	Offline	Offline	Online	Online	Offline
Stochasticity	Present	Absent	Absent	Absent	Absent	Absent	Absent	Absent	Absent
Resolution	6b	16b	16b	4b-16b	16b	16b	NA	4b, 5b, and 16b	8b, 16b
Power	690μW at peak performance	121mW	63mW	7.5-300mW	45mW	278mW	87mW	268mW	63.5mW
Supply voltage	0.4-1V	1.2V	0.77-1.2V	Unavailable	1.2V	0.82-1.17V	0.9V	0.45-1V	0.6-1.1V
No. inferences/sec	254K	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	1.7M	9.9M	30K
No. training/sec	118K	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported
Performance (TOPS/W)	3.12	1.24	8.1	0.26-10	1.42	0.21	3.43	Not Reported	Not Reported
Application	Autonomous mobile-robotics	Object Recognition	General	Visual recognition	CNN processor	Vision	Vision	Vision	Hand-written recognition
Min. Energy/ inference	0.69nJ	Not Reported	Not Reported	3000nJ	Not Reported	Not Reported	50.1nJ	Not Reported	580nJ
Min. Energy/ training	1.5nJ	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported	Not Reported

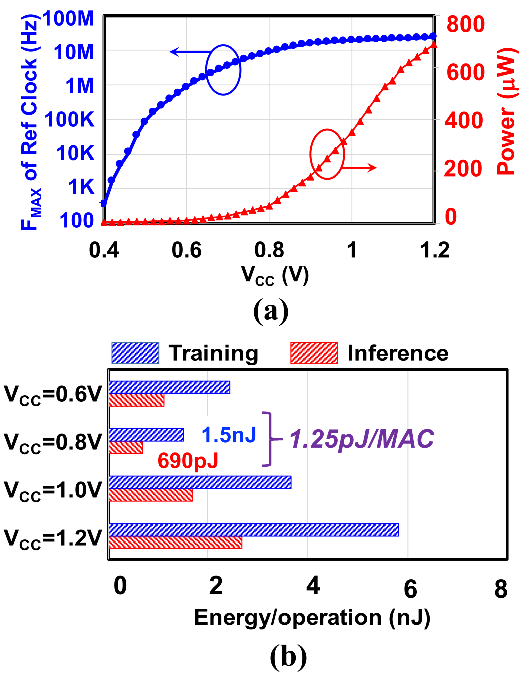


Fig. 24. (a) Measured performance and power as a function of the supply voltage. (b) Measured energy efficiency as measured by the energy to perform a single inference and training on the network.

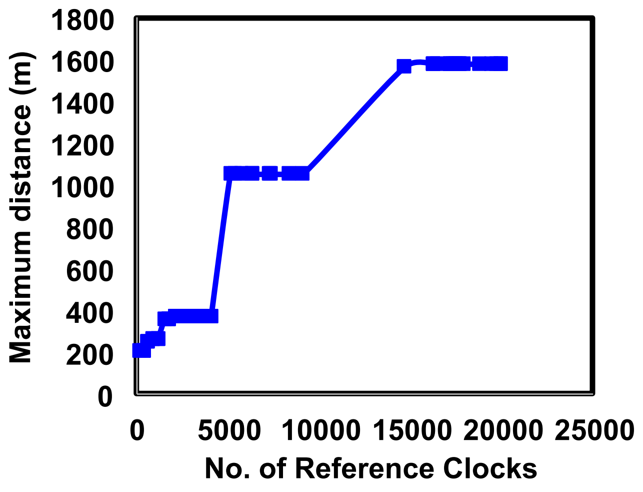


Fig. 25. Distance covered by the robot via RL as a function of the number of clock cycles or iterations.

acceptable linearity with a peak INL of 1.3 LSB (1.6 LSB) at 1 V (0.4 V) and DNL of 1.5 LSB (at 1 V) and 1.8 LSB (at 0.4 V). The DR of the DPC is 4.9 μs and 2.8 ms for supply voltages of 1 and 0.4 V, respectively. We note that: 1) the DCO and the DPC are both composed of programmable buffer chains and their delays track each other across V_{CC} and 2) the programmability of the DPC and the DCO depends on the number of stages of buffer delay, which results in high linearity. Fig. 23(a) shows the intentionally introduced LFSR jitter normalized to a mean pulsewidth. We can observe that the jitter variance is within 40% and can be further tuned by programing the fixed and stochastic parts of the synaptic delay chains. To evaluate the effectiveness of the stochasticity in the network, we train the robot for obstacle avoidance with and without stochasticity. Fig. 23(b) illustrates the loss function over time, and we observe that compared to a deterministic network, a stochastic network converges faster. From inference measurements in different environments, we have further validated that the system generalizes better with a stochastic network and prevents over-fitting on the training environment.

Fig. 24(a) shows the measured clock frequency (F_{MAX}) as a function of V_{CC} from 0.4 to 1 V. Fig. 24(b) illustrates the corresponding energy measured separately for inference and training. At a supply voltage of 0.8 V, the energy consumptions for training and inferences are 1.5 nJ and 670 pJ, respectively. This corresponds to a peak energy efficiency of 1.25 pJ/MAC.

When the entire system is assembled, the supply voltages and power consumed by the various components are summarized in Table I.

Fig. 25 illustrates the distance moved by the robot in the presence of obstacles in a particular experimental setup.

In Table II, we compare the current test chip with the state-of-the-art silicon implementations of neural networks. We believe that this is the first implementation of RL particularly for edge robotics. Comparison with other reported results reveals that the TD-MS topology is energy-competitive and can work down to 0.4 V at a peak power efficiency of 3.12 TOPS/W.

VIII. Conclusion

This paper presents a neuromorphic accelerator for RL at the edge of the cloud suitable for mobile robotics. We introduce TD-MS computation as a means of achieving high energy efficiency as well as the scalability of digital logic. TD-MS computation is energy-proportional, where most of the energy is spent on computations that are more important to the network. We measure a peak efficiency of 3.12 TOPS/W and the average energy of 1.25 pJ/MAC.

Acknowledgment

The authors would like to thank the anonymous reviewers, whose comments have helped them to make this paper better.

References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2016.

[2] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10 TOPS/W sub word-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*, Feb. 2017, pp. 246–247.

[3] D. Shin, J. Lee, J. Lee, 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*, Feb. 2017, pp. 240–241.

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

[5] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "DNN engine: A 28-nm timing-error tolerant sparse deep neural network processor for IoT applications," *IEEE J. Solid State Circuits*, vol. 53, no. 9, pp. 2722–2731, Sep. 2018.

[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," Courant Institute, NYU Corinna Cortes, Google Labs, Microsoft Res., Redmond, WA, USA, Tech. Rep., 2010.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *Unsupervised Learning*. New York, NY, USA: Springer, 2009, pp. 485–585.

[9] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, 1984.

[10] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, 2013, pp. 1–9.

[11] J. Zhang, Z. Wang, and N. Verma, "A matrix-multiplying ADC implementing a machine-learning classifier directly with data conversion," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.

[12] A. Amaravati et al, "A 130 nm 165 nJ/frame CD smashed-filter based mixed-signal class. For smart cameras," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, to be published.

[13] E. H. Lee and S. S. Wong, "A 2.5 GHz 7.7 TOPS/W switched-capacitor matrix multiplier with co-designed local memory," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2016, pp. 418–419.

[14] D. Bankman and B. Murmann "An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS," in *Proc. IEEE-Asian Solid State Circuits Conf.*, Nov. 2016, pp. 21–24, no. 5.

[15] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μ J/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28 nm CMOS," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 222–224.

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

[17] C. Xiaofan, Z. Lin, and W. Pan, "Towards accurate binary convolutional neural network," *Comput. Sci.*, vol. 2, no. 5, pp. 1–14, 2018.

[18] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6t Sram array," in *Proc. Symp. VLSI Circuits*, 2016, pp. 1–2.

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

[20] 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*, 2017, pp. C30–C31.

[21] A. Anvesha, S. Xu, J. Romberg, and A. Raychowdhury, "A 65 nm compressive-sensing time-based ADC with embedded classification and INL-aware training for arrhythmia detection," in *Proc. IEEE-Bio-Med. Circuits Syst.*, Oct. 2017, pp. 1–4.

[22] J. Park *et al.*, "Online RL NoC for portable HD object recognition processor," in *Proc. CICC*, 2012.

[23] G. Cauwenberghs, "A nonlinear noise-shaping delta-sigma modulator with on-chip reinforcement learning*," *Analog Integr. Circuits Signal Process.*, vol. 18, pp. 289–299, Feb. 1999.

[24] A. Amravati, S. B. Nasir, S. Thangadurai, I. Yoon, and A. Raychowdhury, "A 55 nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 124–126.

[25] K. Doya, "Complementary roles of basal ganglia and cerebellum in learning and motor control," *Current Opinion Neurobiol.*, vol. 10, no. 6, pp. 732–739, Dec. 2000.

[26] D. Poole and A. K. Mackworth, "Artificial intelligence: Foundations of computational agents," *Stat. Process.*, 2017.

[27] R. Bellman *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 2003.

[28] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, "Stochastic synapses enable efficient brain-inspired learning machines," *Frontiers Neurosci.*, vol. 10, p. 241, Jun. 2016.

[29] Y. Tang, "Deep learning using linear support vector machines," in *Proc. ICML*, 2013, pp. 1–6.

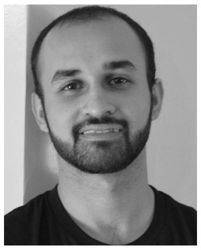
[30] P. N. Whatmough *et al.*, "A 28 nm SoC with a 1.2 GHz 568 nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2017, pp. 242–243.

[31] D. Miyashita *et al.*, "An LDPC decoder with time-domain analog and digital mixed-signal processing," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 73–83, Jan. 2014.

Anvesha Amaravati (S'12) received the M.Tech. degree in microelectronics from IIT Bombay, Mumbai, India, in 2012, with a focus on PVT tolerant analog and bio-medical circuits. He is currently pursuing the Ph.D. degree with Georgia Tech, Atlanta, GA, USA.

He has authored/co-authored 15 international conference/journal publications. His research interests include mixed-signal circuits and systems for machine learning.

Mr. Amaravati has won the national design contests organized by Analog Devices and Cadence.



Saad Bin Nasir (S'13–M'18) received the Ph.D. degree in electrical and computer engineering from Georgia Tech, Atlanta, GA, USA, in 2017.

His industry experience includes over three years as a Design Engineer with the Center for Advanced Research in Engineering (CARE), Islamabad, Pakistan, and a Research Intern with Intel Labs, Portland, OR, USA, and Qualcomm, Inc., USA. He is currently a Senior Design Engineer with Qualcomm, Inc. He has authored/co-authored over 25 journal and conference publications. His research interests include analog/digital/mixed-signal circuit design for power management in high-performance servers, mobile devices, and Internet of Things (IoT).

Dr. Nasir serves as a Technical Program Committee member for Design Automation Conference and VLSI Design conferences. He was a recipient of the 2013–2014 Fulbright Fellowship, the 2016–2017 International Solid-State Circuits Society Pre-Doctoral Achievement Award, and the Best Student Paper Awards at Hardware Oriented Security and Trust 2017 and TECHCON 2016, 2017.

Justin Ting (S'18) is currently pursuing two minors, music technology and computer science, and electrical engineering major degree at Georgia Tech, Atlanta, GA, USA.

In 2017, he joined ICSRL, where he worked on robotic motion controlled by neural networks, a project that he demonstrated at the International Solid State Circuits Conference 2018. In 2018, he was a Validation Engineer with Intel. He continues to work with ICSRL and aspires for a Ph.D. degree.

Insik Yoon (S'17) received the B.S. and M.S. degrees from Carnegie Mellon University, Pittsburgh, PA, USA, in 2009 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Georgia Institute of Technology, Atlanta, GA, USA.

From 2010 to 2015, he was with Memory and Display Interface Design, TLi and SK Hynix, Icheon, South Korea. His current research interests include emerging memory technologies and in-memory computing for machine learning acceleration.

Arijit Raychowdhury (SM'13) received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2007.

In 2013, he joined Georgia Institute of Technology, Atlanta, GA, USA, where he is currently an Associate Professor with the School of Electrical and Computer Engineering and holds the ON Semiconductor Junior Professorship. His industry experience includes five years as a Staff Scientist in the Circuits Research Lab, Intel Corporation, and one year as an Analog Circuit Researcher with Texas Instruments, Inc. He has published over 150 articles in journals and refereed conferences. He holds over 25 U.S. and international patents. His research interests include low-power digital and mixed-signal circuit design, device-circuit interactions, and novel computing models and hardware realizations.

Dr. Raychowdhury received the Dimitris N. Chorafas Award for Outstanding Doctoral Research in 2007, the Best Thesis Award from the College of Engineering, Purdue University, in 2007, the Intel Labs Technical Contribution Award in 2011, the Intel Early Faculty Award in 2015, the NSF CISE Research Initiation Initiative Award (CRII) in 2015, the IEEE Design Automation Conference Innovator Under 40 Award in 2018, the Georgia Tech's Outstanding Young Faculty Award in 2018, and multiple best paper awards and fellowships.