ARYABHAT: A Digital-Like Field Programmable Analog Computing Array for Edge AI

Pratik Kumar, *Graduate Student Member, IEEE*, Ankita Nandi, Ayan Saha, Kurupati Sai Pruthvi Teja, Ratul Das, Shantanu Chakrabartty, *Senior Member, IEEE*, and Chetan Singh Thakur, *Senior Member, IEEE*

***Abstract*— Recent advances in margin-propagation (MP) based approximate computing have resulted in analog computing circuits that exhibit scaling properties similar to that of digital computing circuits. MP-based circuits allow trading off energy-efficiency with speed and precision, endow robustness to temperature variations, and make the design portable across different process nodes. In this work, We leverage these scaling properties to design ARYABHAT, a field-programmable analog machine learning processor that can be synthesized like digital field-programmable gate arrays (FPGAs). ARYABHAT features a fully reconfigurable tile-based modular analog architecture with adjustable throughput and configurable energy requirements, making it suitable for various machine-learning computations. The architecture can perform computations at variable accuracy and different power-performance specifications and can simultaneously leverage near-memory computing paradigms to improve computational throughput. We also present a complete programming and test ecosystem for ARYABHAT called ARYAFlow and ARYATest. As proof of concept, we showcase the implementation of machine learning algorithms at different performance specifications.**

***Index Terms*— Analog machine learning, analog accelerator, neural array, field programmable, margin propagation.**

I. Introduction

ANALOG computing has emerged as an attractive paradigm for machine learning applications, promising exceptional computational density and energy efficiency [1]. However, the inherent approximate nature and dependence on physical models make the approach vulnerable to transistor non-idealities. Unlike digital computing, analog computing lacks attributes like noise margin, which endows noise immunity to the system, and bit truncation and expansion, which endows variable computational precision and energy efficiency to the system. Furthermore, analog designs are not inherently portable and face challenges when scaling across advanced process technology nodes.

To address these limitations associated with analog computing, a Margin Propagation (MP) based approximate analog computing framework was proposed in [2]. This framework allows analog circuits to demonstrate bias-scalability, a property that enables trading off speed with energy efficiency [2]. Recently, a generalized version of the Margin Propagation principle has been introduced [3], enabling scalable current-mode analog circuits across process nodes and their operation at different computational precision. This framework referred to as shape-based analog computing (S-AC) [3], [4], aimed to create robust, non-linear computational shapes that depend only on the generic properties of transistors.

In light of these challenges, this work introduces a field-programmable analog computing architecture called ARYABHAT (Analog Reconfigurable technologY And Bias-scalable Hardware for AI Tasks) which incorporates a design flow akin to that of a digital field-programmable gate arrays (FPGA). Additionally, we propose ARYAFlow, a compiler to map neural network algorithms onto ARYABHAT, and a corresponding test infrastructure called ARYATest. The ARYABHAT design has been prototyped in CMOS 180nm, and the ARYAFlow and ARYATest frameworks were employed to program and demonstrate various machine learning and signal processing applications. Fig. 1 showcases the digital equivalent of the analog properties introduced in ARYABHAT, along with different compute styles, establishing a pathway for incorporating digital design properties into the analog computing environment for improved scalability and robustness.

Key Contributions of this work include:

- **MAC based near-memory crossbar computational tile:** Design of near-memory crossbar computational tile for analog multiply and accumulate (M-AC) operation, enabling scalable computations in analog under diverse operational conditions.
- **Digital-like Analog Synthesis Flow:** Proposition of a standard cell-based digital-like analog synthesis flow to rapidly design analog machine learning systems.
- **Re-configurable Interconnect Matrix:** Design implementation of a digitally programmable current-mode analog

Manuscript received 30 July 2023; revised 11 November 2023 and 4 December 2023; accepted 2 January 2024. This work was supported in part by the Department of Science and Technology of India under Grant SERB CRG/2021/005478 and Grant DST/IMP/2018/000550 and in part by the Indian National Academy of Engineering under Grant INAE SP/INAE-22-2106. This article was recommended by Associate Editor Y. Tang. *(Pratik Kumar and Ankita Nandi contributed equally to this work.) (Corresponding author: Chetan Singh Thakur.)*

Pratik Kumar, Ankita Nandi, Ayan Saha, Kurupati Sai Pruthvi Teja, Ratul Das, and Chetan Singh Thakur are with the NeuRonICS Laboratory, Department of Electronic Systems Engineering, Indian Institute of Science, Bengaluru 560012, India (e-mail: csthakur@iisc.ac.in).

Shantanu Chakrabartty is with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: shantanu@wustl.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2024.3349776.

Digital Object Identifier 10.1109/TCSI.2024.3349776

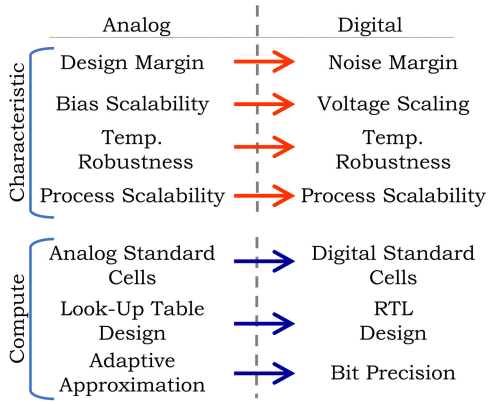


Fig. 1. Digital equivalent properties of generalized margin propagation (GMP) based analog computing system and its design methodology.

interconnect fabric to enable reconfigurability and programmability across multiple tiles.

- **ARYAFlow Mapper:** Developing ARYAFlow, a mapper designed for mapping and optimizing machine learning algorithms for use on accelerator chipsets, generating bitstreams for the chip's programmability.
- **ARYATest Framework:** Developing ARYATest, an open-source test framework built with the PyVISA package, facilitating automated functional verification of machine learning algorithms on the accelerator chipset.
- **Algorithms Implementation:** Demonstrating the implementation of a standard 4-layer neural network, a variant of SVM classifier, and a parallel FIR filter bank using the accelerator chipset.

The rest of the sections are organized as follows. Section II describes the prior work in the field of bias scalable computing. Section III shows the architecture ofcomputational tile, processing element, analog interconnect fabric, and activation sparsity. In Section IV, we show the design mapping, optimization and synthesis methodology for digital-like analog design. Section V presents the performance evaluation of the computational core and interconnect matrix. Section VI presents three case studies implemented on the ARYABHAT chipset. Section VII concludes the paper with discussions and final remarks.

II. Related Work

This section provides an overview of related work in the field of analog accelerators and field-programmable analog arrays (FPAAs). Simultaneously, it also provides an overview of the related works in Margin Propagation (MP) computing principles [2], [3], [4] and presents an understanding that leverages these findings for a comprehensive system design, introducing novel modular architectures for build

The underlying principle of field-programmable analog computing architecture is to be able to integrate analog computing circuits (operating using current or charge conservation [2], continuous-time dynamics [5] or translinear principles [6]) within an interconnect fabric that is digitally programmable. Previously, this paradigm has led to field-programmable synaptic arrays [7], trainable classifiers [8] and field-programmable analog arrays (FPAA) [9]. The paradigm has recently been used to implement dynamical systems [10], matrix-vector multipliers, and AI accelerators [11]. However, all these approaches required proper biasing of the analog computing circuits so as to exploit the physics of that operational regime (sub-threshold or above-threshold characteristics). This not only limited the dynamic range of the system but, also made the design susceptible to analog artifacts like temperature variations. Furthermore, unlike their digital counterparts, analog computing circuits are more susceptible to fan-out/fan-in (loading) artifacts, which makes embedding analog circuits within a programmable interconnect more challenging. Different interconnect loads will require different driving currents and hence, it is difficult to limit the biasing of transistors within an operational regime. MP-based analog computing addressed this limitation by using primitives (like KCL and thresholding) that scaled across different transistor biasing conditions (weak, moderate and strong inversion). In [4], a reconfigurable MP-based analog computing architecture was reported and was demonstrated for small-scale ML architectures. In [3], a generalized version of MP-based computing was proposed which was used to extend the framework to a large repertoire of analog computing modules.

Fig. 2 showcases a top-down design flow based on the Generalized Margin Propagation (GMP) principles, demonstrating the process of mapping machine learning algorithms to their corresponding circuit synthesis.GMP, discussed in detail in [3], is a multi-spline approximation framework for approximating non-linear monotonic functions and has been utilized as a basic framework for S-AC domain. The diagram clearly distinguishes the previous works, which focused on approximation algorithms and their circuit-level implementations [2], [3], [4], from the complete mapping flow, architectural implementation of ARYABHAT, mapper, and test framework presented in this work. In Fig. 2, Block A illustrates the initial mapping of a standard machine learning algorithm into the S-AC domain, represented as a combination of addition and thresholding operations. These mapped operations, along with other mathematical functions, are implemented using S-AC-based analog computation modules. Subsequently, a custom mapper creates a dataflow graph, performs multiple optimizations, and generates a functional map of the S-AC-based multiply-and-accumulate unit and various S-AC-based analog standard cells. Additionally, the mapper determines the level of computational accuracy based on the specific application requirements. Finally, the mapping is executed on the S-AC circuit shown in Block D. This systematic approach enables the design of digital-like large-scale analog ML systems, leveraging the benefits of MP principles and presenting a pathway for enhanced efficiency and scalability in the analog computing domain.

III. Architecture Core

Machine learning algorithms are often represented as a series of transformations organized in a graph structure, where each node processes and transforms the output of the previous node. These transformations are parallelized to increase efficiency by mapping the graph onto specialized hardware designed to perform the operations in parallel. ARYABHAT

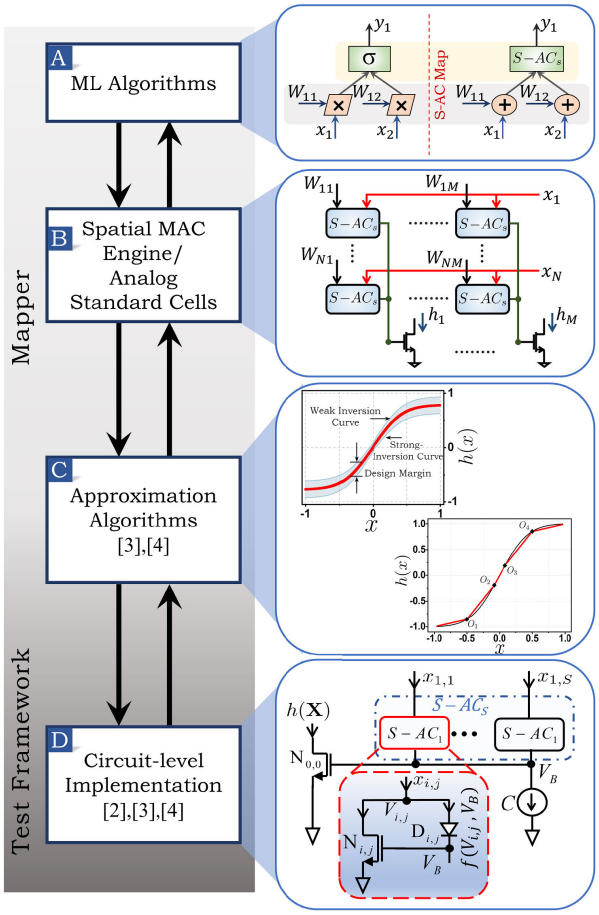


Fig. 2. Top-down flow from mapping a machine learning algorithm to its circuit-level implementation. Here, Block A shows the representative mapping of a generic computational operation used in a standard neural network to its equivalent S-AC computational map; Block B shows the dataflow mapping of the computational map to the S-AC multiply-and-accumulate engine and other S-AC analog standard cells; Block C shows the multi-spline hardware approximation [3] for variable accuracy computation controlled thereby deciding the hardware complexity, and Block D shows the circuit level implementation of S-AC modular block [3]. The mapper performs a dataflow map, computational map, and other optimization while a generic test framework interacts with all the blocks from A-D.

implements parallelism using a multi-tile-based design that can be reconfigured to perform multiple functions in parallel. The system also includes the ARYAFlow mapper that converts ML algorithms into a form that can be understood by the ARYABHAT chip and performs optimization based on user-defined specifications, an analog interconnect fabric and an ARYATest framework for automated testing of an analog chip.

A. Computational Tile

Matrix multiplications dominate neural network calculations. To run the neural network calculations, one needs to multiply the neuron inputs x by the neuron weights W_{ij} where the inputs are a vector, and the neurons are a matrix. The resultant is the vector created by multiplying these two together. Here, the matrix multiply comprises much smaller multiply-accumulate (MAC) operations where multiplication in such MAC operations is not only resource-intensive but energy and area-inefficient too.

ARYABHAT addresses the limitations of resource-intensive and inefficient matrix multiplications by implementing all operations as additions, subtractions, mirroring, and rectification using S-AC-based analog standard cells. These cells maintain robust functionality across different operational conditions, process nodes, and temperature variations. A key difference between traditional neural networks (NNs) and S-AC-based networks implemented in ARYABHAT is the way in which they handle multiplication and non-linearity in the form of addition and thresholding operations.

ARYABHAT's near-memory computational architecture is based on a reconfigurable, tile-based design that utilizes S-AC processing elements and memory elements that work in close proximity to each other, as shown in Fig. 3. The computational tile consists of an array of $N \times M$ processing elements, as shown in the first inset of Fig. 3. One or more computational tiles, such as the one shown in Fig. 3, can work in parallel to perform a variety of computations, including MAC operations and generic functions like non-linearity, integration, and differentiation. The tiles can be programmed to work separately or in tandem and can be configured to operate in different operating regimes for optimal performance. The architecture includes a grid of compute tiles that communicate via a custom interconnect module and a runtime configuration bus that allows for efficient runtime reconfiguration of the hardware. Each S-AC tile consists of arrays of scalable processing elements: modular, bias scalable, and process scalable. Each processing element can store 8 bits of binary data converted to current using an S-AC-log compressive DAC discussed in detail in [4] and then fed to the analog S-AC module. Each column in the computational tile shown as the first inset of Fig. 3 implements the MAC operation between the input element and the weight. The mathematical formulation implementing the operation has been explained in detail in Appendix. The loop nest representation is an alternate way to view the processing near memory in this architecture. Algorithm 1 illustrates a processing-in-memory design for a FC layer with M output channels and where the input activations are flattened along the input channel, height, and width dimension. The computation takes place in one cycle computing all the results in a single cycle in line 12. It can be noted that line 12 in Algorithm 1 implements the S-AC mathematical function explained in detail in [3] and [4].

B. Processing Element

Each processing element of ARYABHAT is based on the MOS implementation of generalized margin propagation (GMP) function [3]. The second inset in Fig. 3 shows the basic architecture of the in-memory processing element (PE), arrays of which are utilized in each computational tile. The processing element consists of MOS-based S-AC unit and an S-AC-based near-memory log compressive DAC. It can be noted that S-AC unit is a current-mode circuit implementing the GMP principle and has been discussed in [3]. The summarized mathematical formulation implementing its operation has been explained in detail in Appendix. It can further be noted that the processing element apart from receiving inputs w_{ij} and x_i also receives the constant offsets O_j which is generated internally simply using the multiple of reference mirrors. The number of offsets decides the computation accuracy (here

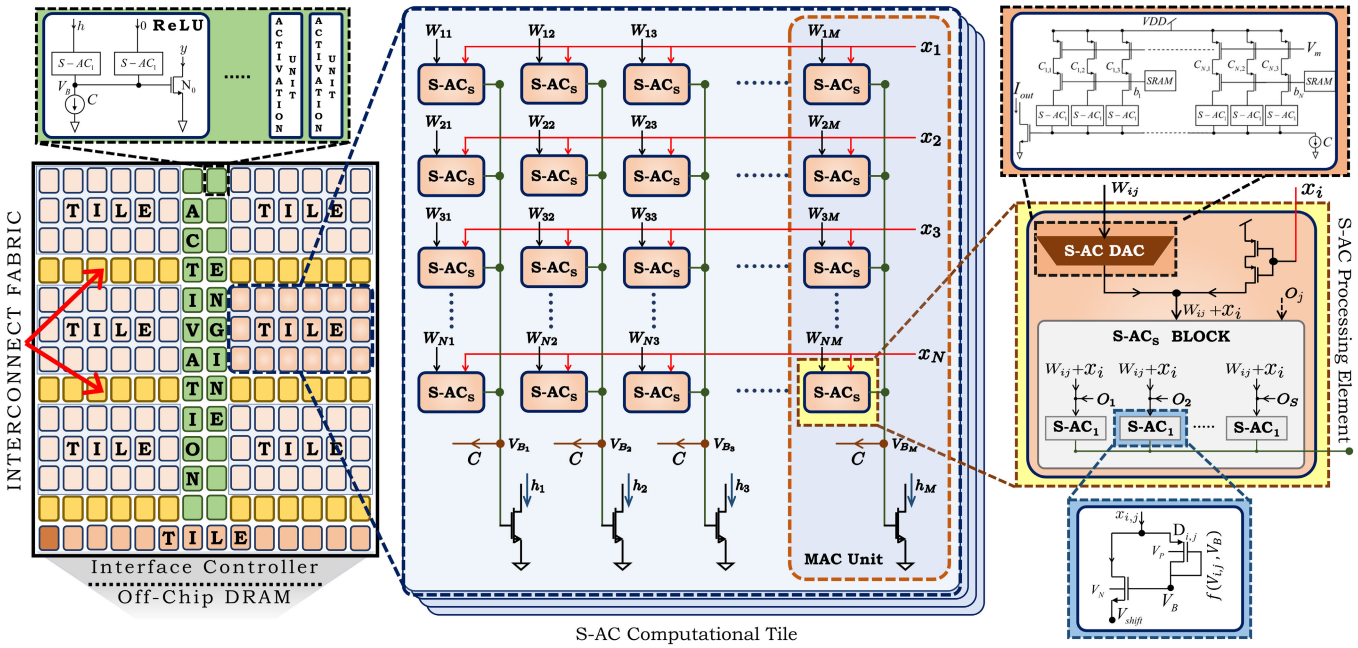


Fig. 3. Block diagram showing S-AC-based near-memory analog machine learning tiles where the zoomed-inset (S-AC Computational Tile) shows an array of $N \times M$ computational tile implementing analog multiply and accumulate (M-AC) unit. Each column has a current sink C, a design parameter, and an output NMOS transistor. The second zoomed-inset (S-AC Processing Element) shows the circuit implementation of a near-memory processing element consisting of S-AC units and a near-memory S-AC DAC whose CMOS circuit implementation have been previously reported in [3] and [4].

Algorithm 1 Computational Tile Operation

Input: x, w, O

$\mathbf{x} = Array(N)$ *# Input Activations*
$W = Array(M, N)$ *# Weights*
$\mathbf{h} = Array(M)$ *# Output*
$\mathbf{O} = Array(S)$ # Spline Offsets
f : S-AC proto function.

par-for $j \leftarrow 1$ to M **:**
  par-for $i \leftarrow 1$ to N **:**
    par-for $k \leftarrow 1$ to S **:**
      $h_j \mathrel{+}= f(w_{i,j},\ x_i,\ O_k)$ *# Refer [3], [4]*
    end-for
  end-for
end-for

Output: $\boldsymbol{h} \in [h_1,\ h_2, \cdots, h_M]$

multiplication) [3] and is equal to the number of splines. The choice of offset depends on the application requirement and user specification. Appendix of [3] explains the detailed mathematics of calculating the offset points.

C. Interconnect Fabric

In literature, many systems have been proposed based on two-dimensional arrays of processing elements interconnected by a reconfigurable routing fabric, such as those present FPGAs. [13] discusses a Field-Programmable Analog Array (FPAA) architecture with a 6×6 grid of programmable Configurable Analog Blocks (CABs), eliminating the need for fixed analog sub-circuits. Meanwhile, [14] introduces a novel routing fabric composed of multiplexers and unidirectional point-to-point connections controlled by configuration bits, offering enhanced flexibility at a lower silicon cost. Despite the simple architecture, the drawback is the reconfiguration time (in milliseconds) and the synthesis time (in minutes to hours). Furthermore, the designs are standard cell-based above threshold design, meaning users only have the privilege to tune once the transistor device physics and hence is mostly the energy factor is fixed for such interconnect. At the other end are the recent multi-core processors. In general, no synthesis is involved. Instead, extensions to existing programming languages are used to describe parallelism explicitly.

Implementing a multi-layer neural network requires many pipelined and parallel computations in any generic ML algorithm. The chip design should be re-configured as per application demands. ARYABHAT interconnect fabric encompasses custom high-speed interconnect to define multi-tile connections. The interconnect architecture can implement programmability and gating simultaneously in the design. Gating here refers to powering off the modular computational section to save power and energy. The simultaneous reconfigurability and gating of the ARYABHAT are implemented by a custom, efficient, and high-speed analog Interconnect fabric. At the core the interconnect used modified high-precision switches [12] which reduces channel charge injection, clock feedthrough errors, and the voltage drop induced by the leakage currents considerably.

The designed interconnect architecture allows any P number of input rows to be routed to any Q number of output columns, as briefed in Fig. 4. It also gives the flexibility to route the other way from Q input columns back to P output rows. For $P > Q$, Q needs to be a factor of P. All the P lines are

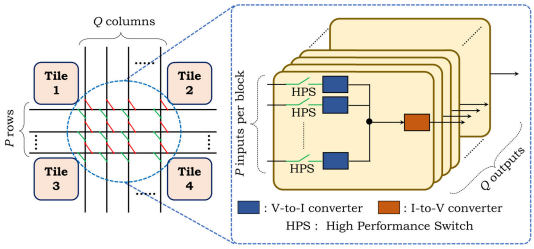


Fig. 4. Block diagram showing $P \times Q$ current mode analog interconnect matrix implemented using modified high precision switch [12] where the inset shows the interconnect fabric implementation of P input nodes and Q output nodes with all-to-all connectivity.

grouped into P/Q sub-groups for uniformity, each followed by an analog High Precision Switch to pass on to the output. For $P < Q$, Q needs to be a multiple of P, with each Q/P group of outputs will share one line of P. The circuit chain remains the same for both cases.

D. Activation Engine

ARYABHAT is designed to configure different types of bias scalable non-linearity depending on the application and user requirements. Depending on control signals, switches can be configured to implement various non-linearity as per the user applications or bypass the activation module or can block the activation input to implement sparsity. M such blocks are integrated at each computational tile and can be configured through ARYAFlow mapper (described in Section VI). Fig. 5 shows the S-AC implementation of four different activations namely Hard ReLU, Soft ReLu, Soft Plus, and Normalized Soft ReLU. The generalized equation for all the computational module can be written as

$$I_{out} = max(h, C) \tag{1}$$

(1) implements Hard ReLU function for ($C \rightarrow 0$, S=1), the Soft ReLU function for ($C \rightarrow 0$, S=3) and the SoftPlus function for ($C \rightarrow k$, S=3). Here, C is the hyper-parameter, S is the number of splines, and k is some constant value. The Normalized Soft ReLU function could be expressed as

$$I_{out1} = h_1 + MP(h_1 - h_2) \tag{2}$$

$$I_{out2} = h_2 + MP(h_1 - h_2) \tag{3}$$

where h_1, h_2 and I_{out1}, I_{out2} are the differential representation of the input h and I_{out} respectively to the activation unit. Some examples of other non-linearities using the S-AC unit have been shown in [3]. The activation engine also implements activation sparsity in the sense that during runtime the activations modules can be configured to pass only non-zero values by dynamically terminating the forward propagation connection to the next PE or the computational tile. In ARYABHAT this is achieved by tuning the hyperparameter C of the S-AC block and programming the register memory present in each activation module of the computational tile. Implementing sparsity in the chip allows ARYABHAT to improve energy efficiency and throughput further.

IV. Design Mapping, Optimization and Synthesis

This section briefly presents the various optimization and mapping implemented in ARYABHAT. In addition, we also present the possible synthesis flow for shape-based analog design for ML applications.

A. ARYAFlow: A Mapper for ARYABHAT

ARYAFlow, as shown in Fig. 6b, is a custom mapper that converts machine learning algorithms into a representation that can be interpreted by the control and logic unit and other sub-units of the ARYABHAT hardware. It takes the algorithms from input libraries and user-defined specifications and translates them into firmware binaries that the reconfigurable chip can use. The mapper considers the hardware specifications and limitations and uses an iterative search process to optimize the execution graph to meet the user specifications. ARYAFlow has four computational stages: Quantizer, Graph Mapper, Optimizer, and Runtime Generator. These stages work together to process the algorithms and generate the necessary binaries efficiently.

- **Quantizer:** The quantization flow converts the floating point numbers into 8-bit fixed points. The quantizer uses a feed-forward process where the floating points are re-scaled into integer values, with the word length set per the hardware specs. It can be noted that reducing the precision of computation would result in higher energy-efficiency (in terms of pJ/MAC) and reduced area. However, once the weights are pre-programmed, the system latency remains invariant and is determined by settling time and slew-rate of the analog computing circuits. In the current architecture of ARYABHAT, the inputs and weights generated by the runtime binaries of ARYAFLOW are 8-bit quantized. The same can be modified to operate with 6-bit or 4-bit precision.
- **Dataflow and Mapping:** To leverage the high performance of a neural network, this stage of the compilation flow aims to maximize parallelism and data reuse with a low-cost memory hierarchy. Analog accelerators face implementation challenges due to the complexity of analog circuits and continuous-time processing, which need more discrete processing units and clock boundaries. Unlike digital accelerators that rely on dataflow taxonomy methods to reuse partial sums or input weights, analog accelerators face limitations due to their dependence on physical properties. ARYABHAT's architecture can mimic input and weight stationary dataflows and potentially support Fully-Connected and Convolutional Neural Networks. In addition, computational tiles can be utilized to implement single-layered and multi-layered fully connected neural networks. Depending on the data type, the user can switch between the two dataflows by programming the mapper, which maps the software-trained engine into its hardware equivalent and loads the network weights into the chip memory. The mapper splits the network across multiple tiles to accommodate higher computational requirements while considering the tile-interconnection constraints within the design flow.
- **Post mapping:** Post mapping, the architecture needs to be optimized. The **optimizer** stage considers the non-linearities of the hardware, which arise from circuit mismatches, and incorporates them into the API. The model is further optimized using S-AC approximations, which include re-training

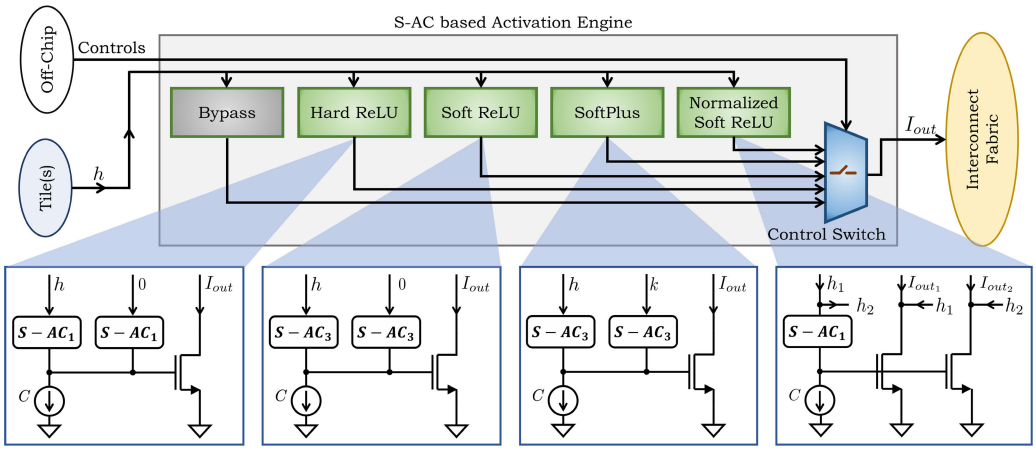


Fig. 5. Block diagram of S-AC standard cells based different activation units (Hard ReLU, Soft ReLU [3], SoftPlus [3] and Normalized Soft ReLU) along with control network implemented inside Activation Engine of ARYABHAT chip.

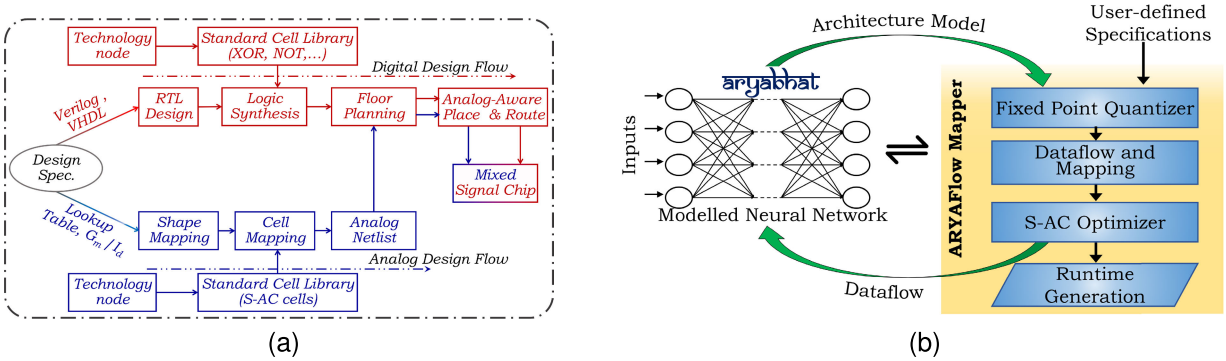


Fig. 6. (a) Block diagram depicting digital-based analog synthesis flow for a mixed-signal analog neural accelerator chip designed using S-AC analog standard cells; (b) ARYAFlow: A custom mapper architecture depicting different compilation and execution flow during algorithm map to prototyped AI chip.

to obtain the optimal spline value, reference current magnitudes at desired operating regions, and suitable activation functions for maximum accuracy.

- **Runtime generator:** The **runtime generator** stage generates patterns for selecting the activation function, scaling magnitudes, and deploying the necessary interconnect and routing networks according to the hardware architecture. These patterns are saved in binary form and are synchronized with the clocks and control lines. The binaries are then fed into the chip for further computations.

Overall, the mapper generates an optimized bitstream from the algorithm and the user specification to generate a more compact representation for efficient computation. These bitstreams are passed on to the ARYATest Framework for subsequent testing flow.

B. Neural Architecture Map

ARYABHAT is capable of supporting both Fully-Connected and Convolutional Neural Networks (CNNs) layers. It can implement single-layered and multi-layered fully connected neural networks using the available computational tiles. Fig. 7 illustrates the mapping of a fully connected layer, as shown in Fig. 7a, onto the S-AC computational tile depicted in Fig. 7b. It can be observed how the parallel Multiply-Accumulate (MAC) operations ($\sum_i W_i^{(M)} x_i$) of the M^{th} node are executed along the M^{th} column of the computational tile. Each processing element within a tile store the pre-fetched weights, and inputs are continuously passed through in an amortized fashion to maximize reuse. Similar mapped operations are shown for different nodes as well.

The architecture of ARYABHAT can also be programmed to replicate different dataflow architectures such as input and weight stationary dataflows to facilitate efficient mapping and computation. Fig. 8a to 8d demonstrate how the computational tile can be configured to support CNN computations in various dataflow configurations according to specific application requirements. Fig. 8a illustrates the Weight-stationary Convolutional Neural Network and its equivalent mapping shown in Fig. 8c. The mapping in S-AC tiles in Fig. 8c demonstrates that when the weights (kernel map) are fixed through the input port in the S-AC tile, and different input feature maps are provided through the weight channels (storing the kernel in memory DAC), the architecture can operate in weight stationary mode. This mode reduces weight-read energy consumption while enabling weight amortization. Such a configuration significantly reduces computational complexity and memory requirements for operations like convolution. Similarly, Fig. 8b shows the Input-stationary Convolutional Neural Network and its equivalent mapping shown in Fig. 8d. The mapping in S-AC tiles in Fig. 8d illustrates that by providing input feature maps through the input port and weights (kernel map) through the weight channel, input stationary mode is achieved. This mode minimizes energy consumption for input per activation.

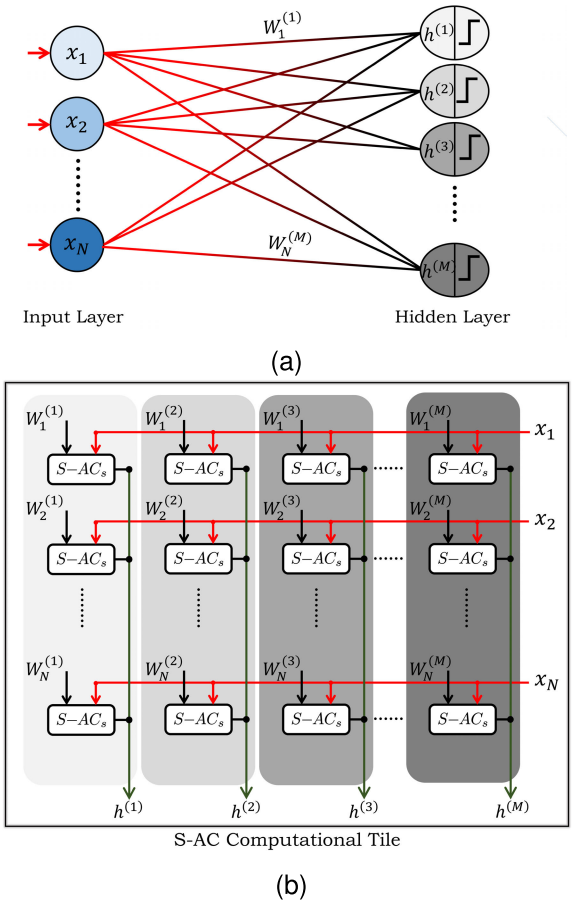


Fig. 7. (a) Block diagram depicting the Input and the Hidden layer of a Fully Connected Neural Network; (b) Mapping of the layers across the S-AC Computational Tiles of ARYABHAT.

In summary, the architecture of ARYABHAT offers flexibility in dataflow configurations, supporting both Fully-Connected and Convolutional Neural Networks while optimizing energy consumption and computational efficiency. The following provides a detailed description of how both weight stationary (WS) dataflow, and input stationary (IS) dataflow, along with various optimizations, are supported and implemented in the ARYABHAT architecture.

C. Dynamic Tile Optimization

In general, to achieve high performance in a system, the hardware and software must be co-designed to improve the system's performance parameters significantly. Thus it reduces trade-offs in mapping and implements an architecture specifically tailored to the target algorithms (such as using GPUs for MAC operations). Thereby reducing computational complexity and increasing throughput. ARYABHAT achieves this by mapping the algorithm into the logarithmic domain, which results in smaller hardware and simplified operations. ARYABHAT architecture can perform dynamic tile optimization as per application requirements. The cores can be programmed to work as

- Low power Low-speed cores;
- High power High-speed cores, or
- Optimum power Optimum speed cores.

Furthermore, multiple tiles can be programmed with the above three and switched off if not needed. This reduced trade-offs in mapping and the designed architecture for the specified algorithms. ARYAFlow mapper considers all these parameters at the tiles and system core levels to perform the required optimization. Thus, the overall system becomes more power, energy, and area-efficient.

D. Design Synthesis

To keep pace with the growing algorithmic complexity and implement machine learning tasks at scale, it is essential to approach analog design in a digital-like manner. One solution is breaking down the larger analog system into smaller, robust modules that can be independently designed and synthesized, akin to the approach used in digital design through digital standard cells. This approach enhances scalability, reduces design complexity, and saves time in the design process. An example of the standard digital-ASIC design flow along with the proposed similar analog design flow is shown in Fig. 6a, which incorporates various design advancements such as analog standard cells [3], lookup table-based design [15], and an automated placement and routing framework [16]. The detailed analog design flow starts with the user design specification and the targeted application. It is assumed that the targeted PDK in any technology node is first characterized in a lookup-based table as in [15]. Then, MATLAB can synthesize the complete computational module or system-level netlist for the remaining step.

- **Shape Map:** Fig. 9 depicts the elaborated flow step in the Shape Mapping block of Fig. 6a. To synthesize the basic shape $h(\cdot)$ starting from the assumed base function $\theta(x)$ (e.g., exponential, binary, or others), we iterate to find the desired count of S and their respective tangential vector points (**Q**) and tuning vector points (**T**), depending on the computational accuracy required by the algorithm. The process is repeated until the desired basic shape with the required accuracy is obtained. Once the basic shape is obtained, we map the desired algorithm to shape-based functions using MATLAB. Firstly, we generate design codes for the basic S-AC unit, shown as S-AC$_1$ in Block D of Fig. 2, using the g_m/I_d design methodology [15] and pre-computed lookup tables for the desired process technology node. A subset of new design codes was then used to synthesize S-AC-based analog standard cells, utilizing the previously synthesized S-AC unit and current mirrors. Finally, the mapper performs the mapping and routing of the S-AC unit to implement the desired mathematical function. It can be noted that various operations like, $cosh(\cdot)$, $sinh(\cdot)$, sigmoid, ReLU, soft-Plus, winner-take-All, N-of-M encoder, soft argmax, max, four quadrant multiplier, log DAC, log-based integration, division, addition, subtraction can be implemented such as shown in [3] and [4].
- **Cell Map:** Once the basic shape $h(\cdot)$ is synthesized, various mathematical functions used in ML can be implemented. For instance, the synthesis of Equation (7) would require four S-AC$_{S=4}$ units interconnected in cascade as shown in [3]. It can be noted that an S-AC$_{S=4}$ unit consists of

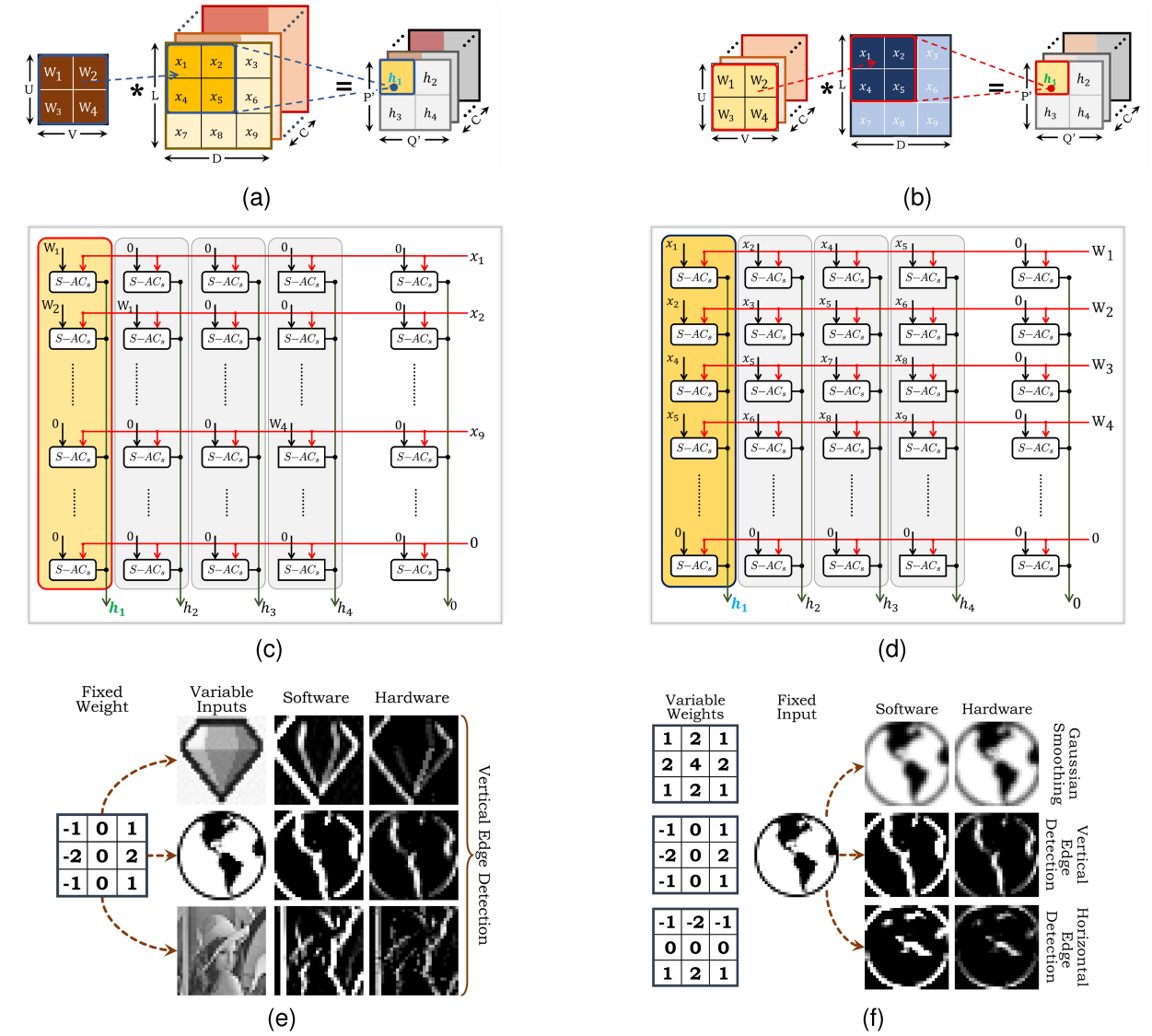


Fig. 8. (a) Block diagram depicting the Weight-stationary Convolutional Neural Network; (b) Block diagram depicting the Input-stationary Convolutional Neural Network ; (c) Mapping the architecture of the Weight-Stationary Convolution Filter across the ARYABHAT Computational Tiles; (d) Mapping the architecture of the Input-Stationary Convolution Filter across the ARYABHAT Computational Tiles; (e) Results of Weight Stationary Convolution Filter; (f) Result of the Input-stationary Convolution Filter.

four S-AC cells connected in parallel and each receiving inputs with some offsets$(O_1, \cdots, O_4)$. The mapper first connects the S-AC standard cells to implement an S-AC$_{S=4}$ unit and then connects them to implement the desired equation. Then constant currents and voltage offsets are applied to the inputs of the standard cells through current mirrors. The whole design was then synthesized as a netlist, which can be used as a black box in the digital synthesis floor planning step. Pre-computed lookup tables for $180nm$ CMOS and $7nm$ FinFETs were used in conjunction with g_m/I_d design methodology to generate synthesize shaped mapped equation using S-AC only standard cells. It can be noted that the same can also be implemented using a traditional GUI schematic viewer or Spice netlists.

V. Measurement and Test

We prototyped the ARYABHAT chip in a standard CMOS 180nm process technology. Fig. 10b shows the die microphotograph of the chip highlighting 8-Computational tiles (where a unit computational tile is shown in Fig. 3), analog interconnect fabric (shown in Fig. 4) and activation network (shown in Fig. 5). It may be noted that multiple copies of the computational tile were fabricated for test purposes. The functionality of the circuit modules has been verified using the test measurement setup shown in Fig. 10a. The test chip was mounted on a custom IC test board, and the test vectors were generated using a PYNQ-Z2 FPGA board which used a Python-based interface to control the digital inputs and outputs. High-precision analog test equipment was directly interfaced with the test chip using the ARYATest framework, also controlled by the PYNQ-Z2 FPGA board. To accurately determine the region of operation, the transistors were first characterized for some fixed circuit parameters (such as the aspect ratio of the transistor, spline count S, etc.). This section shows the results of the S-AC computational core and the interconnect fabric at different operating conditions.

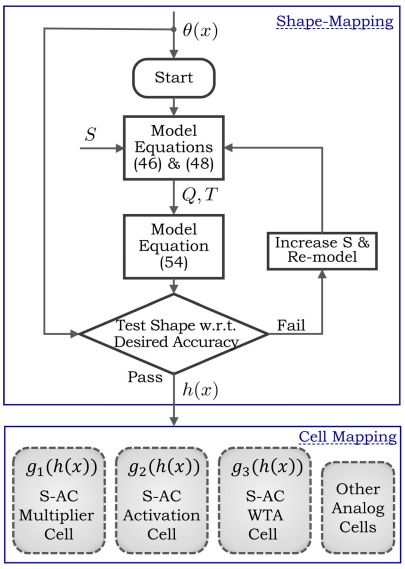


Fig. 9. Flow diagram depicting a step-by-step approach for S-AC based shape mapping and cell mapping blocks shown in Fig. 6a where model equations (46), (48) and (54) are the base design equations as in [3].

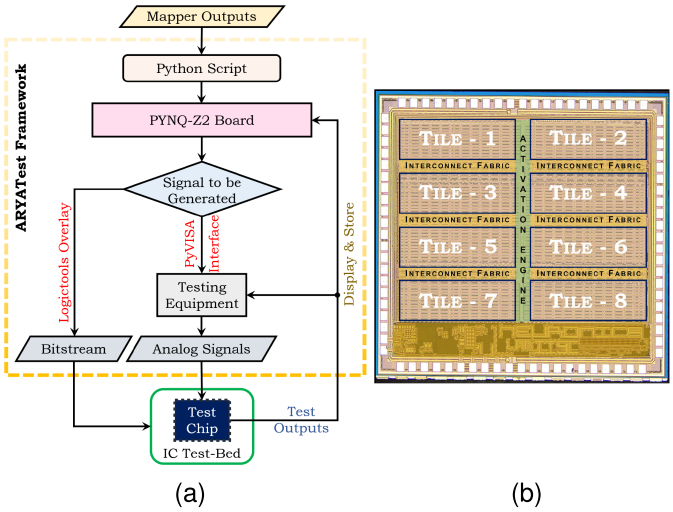


Fig. 10. (a) Flow diagram of ARYATest: An automated analog test framework; (b) Die micro-photograph of prototyped analog AI chip.

A. ARYATest: An Automated Analog Test Framework

Testing large-scale analog chips dedicated to ML and AI applications is an incredibly complex task that demands architectural expertise, expensive test equipment, and significant manual effort and time. Unlike digital chips, analog chips are not only sensitive to noise but also have a significantly larger number of test points compared to equivalent digital architectures. Therefore, it is essential to have a comprehensive test framework that ensures analog AI chipsets meet their design specifications and perform reliably on desired applications with minimal manual intervention and cost.

The ARYATest framework, illustrated in Fig. 10a, is an automated system developed using an open-source Python package. It facilitates the seamless connection between Python-compatible FPGAs, high-precision analog test equipment, and any generic chip. This framework takes the binaries generated by the ARYAFlow mapper as input and consists of efficient sub-units that generate both digital and analog signals for ARYABHAT. A CPU board, compatible with Python scripts, serves as the primary hardware for interacting with the hardware units. The framework is designed to be modular and generic, enabling users to execute various test cases within the limitations of the available controller and instrumentation.

The operation flow of the ARYATest framework involves taking input files and processing them using the available overlay on the board. Key components of this framework include:

- The ARYATest interface uses the *Logictool Overlays*, a hardware library available on PYNQ-Z2 to generate hardware functions like FSMs, Boolean logic functions, and digital patterns. The bitstream patterns for the Inputs, Weights, and Control Signals are generated with this overlay. It also allows the chip-output bitstreams to be ported back into the FPGA and displayed on the PC. In addition, it utilizes *TCL* script and *VISA* language for smooth connectivity and increases functionality.
- Additionally, the open-source PyVISA package is employed to interface the Keithley and Rigol test equipment with the test chip, offering users flexible control. The framework allows for instrument control via LAN and USB, and the configuration setup and operations can be managed through the Python script with minimal user interference. PyVISA provides a standardized API for communication with instruments that utilize various protocols such as GPIB, RS-232, Ethernet, and USB.

B. S-AC Computational Tile Performance

The unique core of ARYABHAT has a scalable design that allows it to function at different operating regimes, including weak, moderate, and strong inversion, without any loss in functionality. This design has been tested at different process nodes but is beyond the scope of this work. This feature gives users the ability to adjust the tile for optimal performance, energy efficiency, or a balance between the two by biasing it towards strong, weak, or moderate inversion, respectively.

The operational performance of the S-AC tile, shown in Fig. 3, is summarized in Table I at various operating conditions for a fixed test condition. We have calculated the performance of the S-AC tile for some commonly used metrics, including system efficiency, power efficiency, and computational efficiency at 50% utilization. The table shows that it is most power efficient in WI while operating frequency is highest in SI. A near-similar pJ/MAC operation can be achieved in both SI and WI. Additionally, it can be seen that the S-AC core has optimal energy performance in the moderate inversion regime, where there is a balance between power and speed, as well as good energy efficiency. It should be noted that the majority of the power consumption in the S-AC core is due to the MP blocks and the routing and interconnect mechanism, while the power consumed by the other supporting circuitries is minimal and can be ignored.

C. Analog Interconnect Performance

An analog interconnect fabric designed for machine learning (ML) applications should have specific characteristics to

TABLE I
OPERATIONAL PERFORMANCE PARAMETERS OF UNIT S-AC CORE

Parameters	†Results		
*Operating Regime	SI	MI	WI
Supply Voltage (V)	1.8		
Total Power (μW)	963.1	816.2	50.2
Operating Frequency (KHz)	109.6	38.4	5.3
System Efficiency (pJ/MAC)	58.6	133.2	60.2
Power Efficiency ($TOPS/W$)	1.7E-2	7.4E-3	1.6E-2
Computational Efficiency ($TOPS/mm^2$)	4.5E-5	1.7E-5	2.3E-6
Storage Efficiency	0.442 KB/mm^2		
Technology	180 nm		
Weight Precision	Int8		
Area	0.359 mm^2		
Sparsity	Fixed programmable sparsity		
Process Scalability	Yes		

* For operation in Strong Inversion (SI) regime $IC > 10$; for Moderate Inversion (MI) regime $0.1 < IC < 10$ and for Weak Inversion (WI) regime $IC < 0.1$ was maintained [17]. Measured results were performed for design parameter $S = 1$, including the cumulative effect of memory DAC, interconnect, Buffers and I/O PADs.
† The results are reported at 50% utilization of the unit S-AC Tile.

TABLE II
OPERATIONAL PARAMETERS OF THE ANALOG INTERCONNECT FABRIC

Parameters	Results					
† Matrix Size	P=Q=2			P=Q=24		
*Operating Regime	WI	MI	SI	WI	MI	SI
Rise Time	4.11 μs	51.60 ns	4.04 ns	4.47 μs	52.85 ns	3.67 ns
Fall Time	63.2 μs	682.81 ns	25.96 ns	48.03 μs	727.61 ns	25.72 ns
Settling Time	41.19 μs	826.30 ns	36.11 ns	41.19 μs	826.30 ns	36.11 ns
Propagation Delay	24.29 μs	463.15 ns	19.61 ns	24.29 μs	463.15 ns	19.61 ns
Operating Freq. (MHz)	0.024	1.21	27.6	0.024	1.21	27.6
Power Dissipation	7.48 nW	361.26 nW	17.65 μW	44.98 nW	4.29 μW	211.68 μW
Area (mm^2)	7.026 E-04			1.083 E-02		
Interconnect Spacing	16.2 μm					

† Results are taken when the matrix sizes (P.XQ) equals $2X2$ and $24X24$, with reference to Fig. 4.
* For operation in Strong Inversion (SI) regime $IC > 10$; for Moderate Inversion (MI) regime $0.1 < IC < 10$ and for Weak Inversion (WI) regime $IC < 0.1$ was maintained [17].

TABLE III
CASE STUDY RESULTS ON ARYABHAT

Non-Linear Classifier			
Feature Size	2		
No. of tiles used	1		
* Operating Regime	SI	MI	WI
Power Consumed (μW)	1061.81	899.69	55.34
†*Classification Accuracy (%)	91.67	90	86.67
Template SVM Classifier			
Feature Size	2		
No. of tiles used	1		
* Operating Regime	SI	MI	WI
Power Consumed (μW)	1481.64	1255.69	77.23
††Classification Accuracy (%)	86.6	81.6	78.3
In-memory FIR Filter			
Feature Size	30		
No. of tiles used	8		
†PE Utilization (%)	47.11		
Input Sampling Rate	16 KHz		
No. of filters	30		
* Operating Regime	SI	MI	WI
Power Consumed (mW)	8.96	7.59	0.47

* For operation in Strong Inversion (SI) regime $IC > 10$; for Moderate Inversion (MI) regime $0.1 < IC < 10$ and for Weak Inversion (WI) regime $IC < 0.1$ was maintained [17]; ** Measurements are taken for implementation of one octave at a time referred to Fig. 11c; † PE Utilization (%) denotes the utilization of the percentage processing element used for the corresponding application; †*Hardware accuracy calculated for baseline software accuracy of 95 %; †† Hardware accuracy calculated for baseline software accuracy of 88 %.

transmit data between ML computational cores efficiently. These characteristics may include high bandwidth for quickly transferring large amounts of data, low latency for timely data transmission, and low power consumption to save energy. Additionally, the interconnect fabric should be scalable and flexible to accommodate different ML algorithms and hardware architectures. Analog interconnect fabrics, which operate on continuous signals, can improve speed, accuracy, and energy efficiency compared to traditional digital designs. However, designing and implementing an analog interconnect fabric for ML can be difficult due to the need for precise control of analog signals and the potential for noise and interference.

The operational performance of the interconnect fabric, shown in Fig. 4, is summarized in Table II. It shows that when we go from WI to SI region the power consumption and the operating frequency increase. It is further observed that when the size of the interconnect matrix increases, there is relatively no change in the timing parameters, while the power consumption of the bigger network increases proportionately.

VI. CASE STUDY: NEURAL NETWORK & ALGORITHM IMPLEMENTATION

This section presents a few of the applications implemented on the ARYABHAT chip. As a proof-of-concept, we show the supported implementation of a non-linear classifier utilizing fully connected layer, template SVM machine learning algorithm (suitable for Edge-AI resource constrained applications), [18] and FIR filter bank implementation (this at the core implements CNN with limited layers) [19] for in-filter computing whose network diagrams and flow charts are shown in Fig. 11a, Fig. 11b and Fig. 11c respectively.

A. Application to Non-Linear Classifier

Non-linear classification using a multi-layer perceptron implementing XOR functionality is shown in Fig. 11a. It depicts the network structure, which includes an input layer, two hidden layers, and an output layer. For the input vector $\mathbf{x} \in \mathbb{R}^2$, the output function for a generic hidden node can be given as

$$h = g\left(f\left(\mathbf{x}\right)\right) \tag{4}$$

where $g(\cdot)$ is a non-linear activation function (here, $ReLU$), and $f(\mathbf{x})$ be the decision function given by

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{5}$$

where $\mathbf{w} \in \mathbb{R}^2$ is the trained weight vector and $b \in \mathbb{R}$ is the corresponding bias. For the sake of simplicity and understanding the mapping on the ARYABHAT chip, let us assume that for a one-dimensional input vector, the decision function $f(x)$ can be written as

$$f(x) = w \cdot x + \mathrm{b} \tag{6}$$

The mapped version of the above decision function in the shape-based form can be written as (7)

$$\begin{aligned} f(x) = h(2C + w + x) - h(2C + w - x) + \cdots \\ \cdots h(2C - w - x) - h(2C - w + x) + b \end{aligned} \tag{7}$$

In (7), standard multiplication is mapped to S-AC multiplication as described in detail in [3]. This multiplication is then

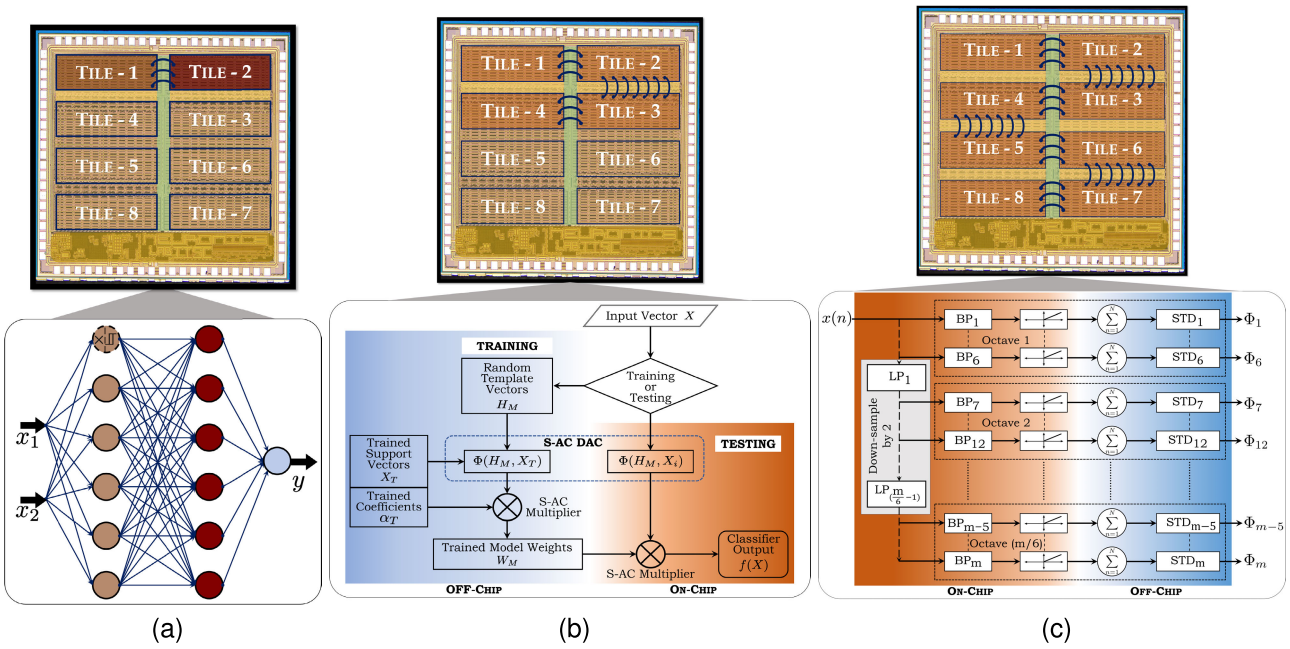


Fig. 11. (a) Neural network diagram with 2-hidden layers implementing a generic non-linearity function; (b) Flow chart describing the off-chip training and on-chip template SVM [18] implementation using S-AC circuits; (c) Flow chart depicting an implementation of FIR parallel filter bank for in-filter computing where low-pass, band-pass and half-wave rectifications are implemented on-chip [19].

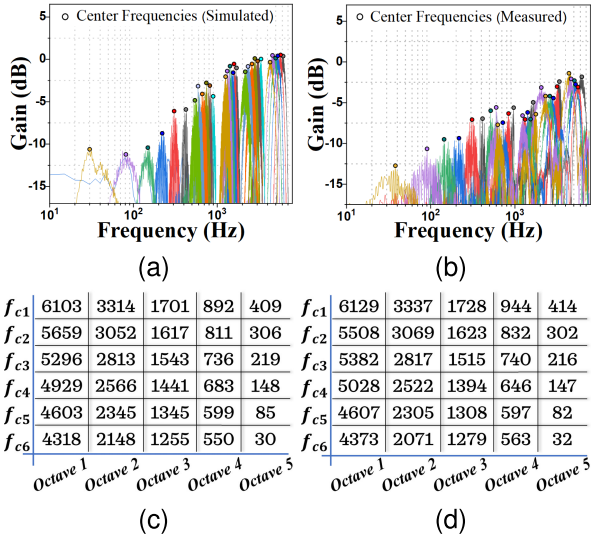
implemented by a column of the computational tile shown in Fig. 3. A PE also implements the bias without data input x. Fig. 11a shows the necessary on-chip routing configuration and tiles enabled to implement the network shown in Fig. 11a. Table III, Non-Linear Classifier sub-section shows the implementation parameter, design performance, and classification accuracy of the mapped network on the ARYABHAT chip. It can be inferred from Table III that the chip works in all operating regimes with reasonable accuracy. The power consumed by the chip and the speed of operation increases while going from WI to SI as expected.

B. Application to SVM Classification

Given a training set $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, N$, where $\mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathbb{R}^c$, the generic form of the decision function for a template SVM formulation [18] is given as

$$f(\mathbf{x}) = \sum_{p=1}^{P} \mathbf{w}_p \Phi(\mathbf{m}_p, \mathbf{x}) \tag{8}$$

where $\mathbf{w}_p = \sum_{s=1}^{S} \alpha_s \Phi(\mathbf{m}_p, \mathbf{x}_s)$ can be thought of as the weight vector obtained after training, P is the number of template vectors, S is the number of support vectors obtained after training, α_s is the trained coefficient and $\Phi(\cdot, \cdot)$ is a non-positive definite function which gives an estimate of the similarity between the p^{th} template vector $\mathbf{m}_p$ and the i^{th} training vector $\mathbf{x}_i$. Fig. 11b shows the flowchart summarizing the design flow for the entire process. For $P = 48$ and kernel Φ being the Cauchy kernel, the implementation is carried out in the ARYABHAT chip, whose routing mechanism is shown in Fig. 11b. Table III (Template SVM Classifier sub-section) shows the implementation parameter, design performance, and classification accuracy of the mapped network on the ARYABHAT chip. It is seen that by using 1 tile, we can get



f_{c1}	6103	3314	1701	892	409
f_{c2}	5659	3052	1617	811	306
f_{c3}	5296	2813	1543	736	219
f_{c4}	4929	2566	1441	683	148
f_{c5}	4603	2345	1345	599	85
f_{c6}	4318	2148	1255	550	30

f_{c1}	6129	3337	1728	944	414
f_{c2}	5508	3069	1623	832	302
f_{c3}	5382	2817	1515	740	216
f_{c4}	5028	2522	1394	646	147
f_{c5}	4607	2305	1308	597	82
f_{c6}	4373	2071	1279	563	32

Fig. 12. (a) Simulated frequency responses of parallel FIR bandpass filters shown in Fig. 11c; (b) Measured frequency responses of parallel FIR bandpass filters shown in Fig. 11c; (c) Simulated center frequencies $f_{c1}, \cdots\cdots, f_{c6}$ of FIR bandpass filters for 5−Octaves$\left(\frac{m}{6} = 5\right)$ shown in Fig. 11c; (d) Measured center frequencies of the similar FIR bandpass filters.

the desired accuracy of the SVM classifier in all regions of operation. The power consumption and the speed of operation increase while going from WI to SI as expected.

C. Application to In-Filter Computation

In-filter computing employs a parallel FIR filter bank which combines feature extraction and nonlinear SVM kernel into a single function and has been described in detail in [19]. This filter bank employs a range of sampling rates to achieve the desired filtering effect. The idea behind decreasing the sampling rate, lower-order filters with a lower cutoff frequency can be implemented, resulting in cost and energy savings through less hardware. Fig. 11c shows the implementation

where the input signal is first filtered through a lowpass filter to eliminate aliasing. The filter bank is then divided into a number of octaves, each of which contains six filters. The center frequencies of the bandpass filters are determined using the Greenwood function [20] and arranged in decreasing order. Within the filter bank, each filter performs bandpass filtering and half-wave rectification (ReLU) on the chip. The resulting rectified samples are then summed and standardized off the chip. For an m^{th} bandpass filter shown in Fig. 11c, filtering equation is given by

$$BP_m(n) = \sum_{k=0}^{S} h_m(k)x(n-k) \tag{9}$$

where $x(n) \in \mathbb{R}^1$ is the input time series data sampled at 16kHz, $\boldsymbol{h}_m \in \mathbb{R}^{S+1}$ are precalculated coefficients of the bandpass filter, S is the order of bandpass filter, for this implementation S = 20, $m \in [1, 2, 3, \cdots, M]$, where M is the total number of bandpass filters in the filter bank, for this implementation M = 30. Similarly, lowpass filtering is implemented as

$$LP_q(n) = \sum_{k=0}^{T} g_q(k)x(n-k) \tag{10}$$

where $\boldsymbol{g}_q \in \mathbb{R}^{T+1}$ are the coefficients of low pass filter, T is the order of low pass filter, for this implementation T = 20, $q \in [1, 2, \cdots, Q]$, where Q is the total number of low pass filters in the filter bank and $Q = \frac{M}{6} - 1$. The filter coefficients are precalculated using the inbuilt MATLAB function. We use a chirp signal with a frequency increasing from 10 Hz to 8 kHz logarithmically to obtain the frequency response. Each filter's inputs and filter coefficients are scaled to map the values from the multiplication table generated from the prototype chip to the S-AC multiplication equation. After calculating the output response for every filter in the filter bank, we perform a low-pass filtering operation using the same procedure as the previous one. The output of the low-pass filter is then downsampled by two and given as input to the next filter bank. The frequency response of all bandpass filters in the filter bank is shown in Fig. 12a and 12b. It should be noted that the distortion in the frequency response in Fig. 12b is due to the S-AC approximation and the quantization noise caused by logarithmic DAC. To observe the center frequency peaks, the final plot is post-processed by smoothing the obtained frequency plot, removing high-frequency noise components that cause distortion. It can be observed in Fig. 12c and 12d that we can almost match the measured center frequencies of bandpass filters to the simulated. Now the half-wave rectification operation for $HWR_m(n) \in \mathbb{R}$ can be implemented as

$$HWR_m(n) = ReLU(BP_m(n)) \tag{11}$$

These half-wave rectified samples are summed according to (12). Since the sampling rate for input is 16 kHz, i.e., N = 16000, $S_m \in \mathbb{R}$. Standardization, commonly used neural network optimization, is performed as per [21].

$$S_m = \sum_{n=1}^{N} HWR_m(n) \tag{12}$$

The kernel function $\phi_m \in \mathbb{R}$ for m^{th} filter is the output of the standardization is given as

$$\phi_m = STD(S_m) \tag{13}$$

TABLE IV

CLASSIFICATION ACCURACY RESULTS FOR ESC-10 DATA SET

Class	Simulated (%)		*Measured (%)	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
Chainsaw	92	85	92	85
Clock Tick	90	82	89	88
Crying Baby	86	80	91	76
Dog	88	88	85	82
Fire Crackling	89	91	91	92
Helicopter	90	92	88	90
Person Sneeze	86	77	88	73
Rain	86	83	86	78
Rooster	92	89	90	85
Sea Waves	87	90	88	86

* Chip measurements are done for one octave at a time, and the training network is implemented off-Chip.

TABLE V

OPERATIONAL PERFORMANCE PARAMETERS OF ARYABHAT

Parameters	*Results					
Operating Regime	SI	MI	WI	SI	MI	WI
Supply Voltage (V)	1.8			1.3		
Frequency (Khz)	54.8	19.2	2.6	7.4	4.6	2.9
S.E (pJ/MAC)	112.7	271.2	121.5	7.2	42.6	28.2
P.E ($TOPS/W$)	8.9E-3	3.7E-3	8.2E-3	1.4E-4	3.1E-5	8.0E-5
C.E ($TOPS/mm^2$)	1.4E-5	5.1E-6	7.0E-7	1.9E-9	1.2E-9	7.7E-10
Storage Efficiency	0.442 KB/mm^2					
Technology	180 nm					
Available Precision	int-8					
On-chip Memory	1.42 kB					
Area	5.5 mm^2					
Algorithm	SVM, MLP, 8-10 Layer NN					
Sparsity	Fixed programmable sparsity					
Process Scalability	Yes					

* The results are reported for 50% utilization of all the ARYABHAT chip processing elements.

These kernel functions are further passed through the decision function for classification, which is given as

$$y = \mathbf{w}^T \Phi + b \tag{14}$$

where $\mathbf{w} \in \mathbb{R}^{30}$ is the weight vector and $b \in \mathbb{R}$ is bias which are obtained during training of model. Here, we used one versus all methodology on the ESC-10 data set with 10 classes. Table IV shows the accuracy results of the trained model. The accuracy results produced by the implemented classifier in hardware are close to the software model. It can be noted that in (9), (10), (11) and (14) standard multiplication is first mapped to S-AC multiplication as described in detail in [3]. This multiplication is then implemented by a processing element (PE) in the core column shown in Fig. 3. The implementation is done so that one octave is implemented on the chip at a time using 8-cores, utilizing 47.11% of processing elements. As a result, five iterations are required to implement the entire filter bank. Fig. 11c shows the necessary on-chip routing configuration and cores enabled to implement network shows in Fig. 11c. Table III, In-memory FIR Filter sub-section shows the implementation parameter, design performance, and the classification accuracy of the mapped network on the ARYABHAT chip.

VII. CONCLUSION & FUTURE WORK

In this study, we introduced a reconfigurable multi-tile analog chipset called ARYABHAT for use in ML and edge

computing applications. We demonstrated the chipset's performance under various power and performance specifications and simultaneously across different applications highlighting its scalability and reconfigurability. In addition, we provided a complete system stack, including an algorithm mapping tool (ARYAFlow) and an analog testing framework (ARYATest) to complete the computing ecosystem. We also presented the design flow and synthesis of analog machine-learning system similar to digital ASIC implementation. In Table V, we summarize the operational performance parameters of the overall chip and report some of the widely used metrics for ML accelerator. It can also be noted that the current version of ARYABHAT has constraints in terms of memory availability and reconfigurability. This leads to a restricted range of CNNs and DNNs that can be implemented. Due to these limitations, the FC neural network can implement up to 8 hidden layers, each having 156 hidden neurons. Finally, the output layer can have 256 neurons with all-to-all connectivity. Additionally, when utilizing the architecture for CNN network, it is possible to employ a 2×2 kernel on a 3×3 matrix and other similar configurations which depend on how the tiles are programmed, considering the restriction in kernel size. Nonetheless, if the chip possesses ample resources, its architecture is versatile enough to support a larger subset of ML and AI algorithms. Future versions of ARYABHAT will expand resources, improve interconnect bandwidth, upgrade the mapper to support more ML algorithms and implement a cluster configuration [22] for improved scalability of DNNs.

APPENDIX
MULTIPLY-AND-ACCUMULATE PROCESSING UNIT

(i) Computational Tile

Each column in the computational tile (shown as the first inset of Fig. 3) implements the MAC operation between the input element and the weight given in GMP formulation and is given by

$$\sum_{i=1}^{N}\sum_{k=1}^{S}[x_{i,k} + w_{ij,k} - h_j]_+ = C, \ \forall j = 1, .., M \quad (15)$$

where, for a given j$^{\text{th}}$ column $x_{i,k}$ input element of a tile corresponding to the i^{th} row; $w_{ij,k}$ weight corresponding to $x_{i,k}$ element;h_j resultant shape of the prototype; C tuneable hyper-parameter of the model, and $[.]_+$ denotes thresholding operation. The equivalent representation of (15) in S-AC form can then be given by

$$\sum_{i=1}^{N}\sum_{k=1}^{S} f(V_{ij,k}, V_B) = C, \ \forall j = 1, .., M \quad (16)$$

where, for a j$^{\text{th}}$ column $V_{ij,k}$ is an internal parameter, V_B is the solution of the equation, and S is the design parameter. It can be noted that in (15) and (16), by varying design parameter S, different approximations in multiplications and other operations can be implemented.

(ii) Processing Element

The processing element consists of MOS based S-AC unit and an S-AC-based near-memory log compressive DAC. For a given i^{th} row and j^{th} column, each PE shown in second inset of Fig. 3 implements the equation given by

$$\sum_{k=1}^{S}[x_{i,k} + w_{ij,k} - h_j]_+ = C, \ \forall i = 1, .., N, \ \forall j = 1, .., M \quad (17)$$

The equivalent representation of (17) in S-AC form can then be given by

$$\sum_{k=1}^{S} f(V_{ij,k}, V_B) = C, \ \forall i = 1, .., N, \ \forall j = 1, .., M \quad (18)$$

ACKNOWLEDGMENT

The joint Memorandum of Understanding (MoU) between IISc and WashU is gratefully acknowledged by the authors for facilitating collaboration between the two institutions. The Fulbright Scholarship provided to one of the authors by USIEF and IIE is also acknowledged. Special acknowledgement to the research interns, Sanchari Das, Balagangadhar Vemula, Sudhanshu Gulavani, and Satyam Saha, for their valuable contributions during the development of this work.

REFERENCES

[1] K. Freund. (Sep. 23, 2021). *IBM Research Says Analog AI Will Be 100X More Efficient. Yes, 100X*. [Online]. Available: https://www.forbes.com/sites/karlfreund/2021/09/23/ibm-research-says-analog-ai-will-be-100x-more-efficient-yes-100x/?sh=268578c8129b

[2] M. Gu and S. Chakrabartty, "Synthesis of bias-scalable CMOS analog computational circuits using margin propagation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 2, pp. 243–254, Feb. 2012.

[3] P. Kumar, A. Nandi, S. Chakrabartty, and C. S. Thakur, "Process, bias, and temperature scalable CMOS analog computing circuits for machine learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 1, pp. 128–141, Jan. 2023.

[4] P. Kumar, A. Nandi, S. Chakrabartty, and C. S. Thakur, "Bias-scalable near-memory CMOS analog processor for machine learning," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 312–322, Mar. 2023.

[5] B. Schell and Y. Tsividis, "A continuous-time ADC/DSP/DAC system with no clock and with activity-dependent power dissipation," *IEEE J. Solid-State Circuits*, vol. 43, no. 11, pp. 2472–2481, Nov. 2008.

[6] B. A. Minch, "Translinear analog signal processing: A modular approach to large-scale analog computation with multiple-input translinear elements," in *Proc. 20th Anniversary Conf. Adv. Res. VLSI*, 1999, pp. 186–199.

[7] P. Hasler, C. Diorio, B. Minch, and C. Mead, "Single transistor learning synapses," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 7, 1994, pp. 1–10.

[8] S. Chakrabartty and G. Cauwenberghs, "Sub-microwatt analog VLSI trainable pattern classifier," *IEEE J. Solid-State Circuits*, vol. 42, no. 5, pp. 1169–1179, May 2007.

[9] J. Hasler, "Large-scale field-programmable analog arrays," *Proc. IEEE*, vol. 108, no. 8, pp. 1283–1302, Aug. 2020.

[10] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis, "A VLSI analog computer/digital computer accelerator," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 42–53, Jan. 2006.

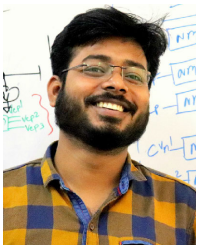
[11] *MYTHIC: A Groundbreaking Architecture Built for AI*. Accessed: Mar. 1, 2023. [Online]. Available: https://www.mythic-ai.com/technology/ and https://www.mythic-ai.com/

[12] S. Naghavi, N. Sharifi, and A. Abrishamifar, "A novel analog switch for high-precision switched-capacitor applications," *Int. J. Circuit Theory Appl.*, vol. 46, no. 4, pp. 764–778, Apr. 2018.

[13] Z. Chen and I. Savidis, "Reconfigurable array for analog applications," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 361–365.

[14] M. Bennebroek and A. Danilin, "Multiplexer-based routing fabric for reconfigurable logic," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2007, pp. 463–466.

[15] P. G. Jespers and B. Murmann, *Systematic Design of Analog CMOS Circuits*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
[16] K. Kunal et al., "ALIGN: Open-source analog layout automation from the ground up," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–4.
[17] D. M. Binkley, "Tradeoffs and optimization in analog CMOS design," in *Proc. 14th Int. Conf. Mixed Design Integr. Circuits Syst.*, Jun. 2007, pp. 47–60.
[18] P. Kumar et al., "Neuromorphic in-memory computing framework using memtransistor cross-bar based support vector machines," in *Proc. IEEE 62nd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2019, pp. 311–314.
[19] A. R. Nair, P. K. Nath, S. Chakrabartty, and C. S. Thakur, "Multiplierless in-filter computing for tinyML platforms," 2023, *arXiv:2304.11816*.
[20] D. D. Greenwood, "A cochlear frequency-position function for several species—29 years later," *J. Acoust. Soc. Amer.*, vol. 87, no. 6, pp. 2592–2605, Jun. 1990.
[21] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, Aug. 1996.
[22] D. García Moreno, A. A. Del Barrio, G. Botella, and J. Hasler, "A cluster of FPAAs to recognize images using neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 11, pp. 3391–3395, Nov. 2021.

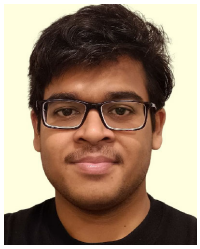


Pratik Kumar (Graduate Student Member, IEEE) received the M.Tech. degree from the Indian Institute of Technology, Dhanbad, in 2018. He is currently pursuing the Ph.D. degree with the Indian Institute of Science, Bengaluru, India. He is also associated with the NeuRonICS Laboratory, Department of Electronic Systems Engineering, and the Centre for Nano Science and Engineering, Indian Institute of Science, Bengaluru. His current research interests include the intersection of hardware-friendly machine-learning algorithms, high-performance mixed-mode machine-learning computational architectures, and multi-state emerging memory devices.

Ankita Nandi received the B.Tech. degree from the National Institute of Technology (NIT) Meghalaya, India, in 2018, and the M.Tech. degree from the Indian Institute of Technology (IIT) Gandhinagar, India, in 2020. She is currently pursuing the Ph.D. degree with the Department of Electronic Systems Engineering, Indian Institute of Science (IISc), Bengaluru, India. She was awarded the President's Gold Medal and the Institute Gold Medal for her outstanding performance during the B.Tech. studies. She was also a recipient of the Fulbright Nehru Doctoral Research Fellowship from 2022 to 2023 and the Prime Minister's Research Fellowship in 2021.

Ayan Saha received the B.E. degree from Jadavpur University, Kolkata, India, in 2021, and the M.Tech. degree from the Department of Electronic Systems Engineering (DESE), Indian Institute of Science (IISc), Bengaluru, in 2023. He was associated with NeuRonICS Laboratory, DESE, IISc.

Kurupati Sai Pruthvi Teja received the B.Tech. degree from the National Institute of Technology, Rourkela, in 2021, and the M.Tech. degree from the Department of Electronic Systems Engineering (DESE), Indian Institute of Science (IISc), Bengaluru, India, in 2023. He was associated with NeuRonICS Laboratory, DESE, IISc.



Ratul Das received the B.Tech. degree from the Institute of Engineering and Management, Kolkata (affiliated to MAKAUT) in 2021 and the M.Tech. degree from the Department of Electronic Systems Engineering (DESE), Indian Institute of Science (IISc), Bengaluru, India, in 2023. He was associated with NeuRonICS Laboratory, DESE, IISc.

Shantanu Chakrabartty (Senior Member, IEEE) received the B.Tech. degree from the Indian Institute of Technology, Delhi, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Johns Hopkins University, Baltimore, MD, USA, in 2002 and 2004, respectively. From 1996 to 1999, he was at Qualcomm Inc., San Diego, CA, USA. From 2004 to 2015, he was an Associate Professor with the Department of Electrical and Computer Engineering, Michigan State University (MSU), East Lansing, MI, USA. He is currently a Clifford W. Murphy Professor and the Vice-Dean of the Research and Graduate Education with the McKelvey School of Engineering, Washington University in St. Louis, St. Louis, MO, USA. He was a Catalyst Foundation Fellow, from 1999 to 2004. He is a fellow of the American Institute of Medical. He was a recipient of the National Science Foundation's CAREER Award, the University Teacher-Scholar Award from MSU, and the 2012 Technology of the Year Award from MSU Technologies. He served as an Associate Editor for IEEE TRANSACTIONS OF BIOMEDICAL CIRCUITS AND SYSTEMS.

Chetan Singh Thakur (Senior Member, IEEE) received the M.Tech. degree from the Indian Institute of Technology Bombay, India, in 2007, and the Ph.D. degree in neuromorphic engineering from the MARCS Research Institute, Western Sydney University, in 2016. He was a Post-Doctoral Researcher with Johns Hopkins University, Baltimore, MD, USA, and a Senior IC Design Engineer at TI Singapore. In 2017, he joined the Indian Institute of Science (IISc), Bengaluru, where he is currently an Associate Professor. His research interests include the computing principles of the brain and apply them to build intelligent VLSI systems. He received several awards, including the Pratiksha Trust Young Investigator Award, the Abdul Kalam Innovation Award from the INAE, and the Inspire Faculty Award for brain-inspired computing from the DST, India.