Temporal Learning for Edge AI: Hyperdimensional Computing with Legendre Delay Networks

Md Rubel Sarkar, Ramashish Gaurav, Govind Sunil Kumar, Shirazush Salekin Chowdhury, Yang Yi
Bradley Department of Electrical and Computer Engineering, Virginia Tech
Blacksburg, Virginia, USA
rubel@vt.edu, rgaurav@vt.edu, gskumar@vt.edu, salekin@vt.edu, yangyi8@vt.edu

***Abstract*—Hyperdimensional computing (HDC) offers a brain-inspired computational paradigm well-suited for robust, low-power machine learning (ML) at the edge, particularly in resource-constrained devices that demand real-time performance. To enhance HDC's capabilities in processing time-series signals, we integrate the Legendre Delay Network (LDN)—a state-space model that provides a compact, low-pass filtered representation of recent input history. This work proposes an HDC architecture empowered by the LDN model to improve classification performance on temporal data streams, which supports continuous-time feature encoding and efficient transformation into high-dimensional representations, optimized for HDC inference. We design and evaluate the proposed architecture on an Intel Altera Cyclone-V field programmable gate array (FPGA) and synthesize it as an application-specific integrated circuit (ASIC) using GlobalFoundries 12nm Low Power (LP) technology. The ASIC implementation consumes $27.31\,\mathrm{mW}$ of power and occupies $0.5\,\mathrm{mm}^2$ of silicon area, highlighting its suitability for edge deployment, while the FPGA version reports $0.17\,\mathrm{W}$ static and $2.746\,\mathrm{W}$ dynamic power at $50\,\mathrm{MHz}$. Experimental evaluations across multiple datasets show that the LDN-integrated HDC architecture achieves significant improvements in classification accuracy over traditional HDC approaches—making it a compelling solution for energy-efficient, real-time edge intelligence.**

***Index Terms*—Hyperdimensional Computing, Legendre Delay Network, Neural Network, Machine Learning, Artificial Intelligence.**

I. Introduction

With the rising demand for low-power, real-time machine learning in edge and embedded applications, interest is shifting toward computational paradigms that differ from traditional deep learning models [1]. Hyperdimensional Computing (HDC), inspired by the brain's use of high-dimensional distributed representations, offers an attractive alternative [2]. It encodes data into high-dimensional hypervectors and employs simple, hardware-friendly operations such as addition, multiplication, and permutation for learning and inference [3]. These characteristics make HDC highly suitable for hardware implementations with tight constraints on power, area, and latency [4].

While HDC has shown great promise in domains like classification, associative memory, and signal processing, its effectiveness is often limited by the quality of the input representations, especially in the case of time-varying signals [5]. Conventional methods use handcrafted or heavy neural extractors, limiting their hardware efficiency.

To address this, we propose the integration of a Legendre Delay Network (LDN) [6] as a temporal feature extractor within an HDC pipeline. LDNs, rooted in control theory and derived from orthogonal Legendre polynomials, offer a compact and structured way to represent the recent history of input signals using a continuous-time state-space model [7], [8]. This makes them ideally suited for capturing temporal patterns without resorting to costly recurrent architectures.

In this work, we design and implement a fully digital HDC architecture with embedded LDN model. We evaluated our design in Altera FPGA board and implemented (automatic place and route (APR)) a custom ASIC chip. The proposed HDC architecture converts input streams into a compact temporal state using the LDN, projects them into high-dimensional space through HDC's random encoders and performs inference via associative memory operations. This architecture is optimized for pipelined and parallel computation, allowing for efficient encoding, training, and inference in different real-world applications.

We evaluate the performance of our design on a time-series classification and image recognition tasks, demonstrating the benefits of LDN-based feature extraction. Implemented with GlobalFoundries low-power 12nm FinFET commercial CMOS process. Our contributions in this work focus on the following areas:

- **LDN empowered Feature Extraction:** HDC's sim-

plistic encoding often fails to capture temporal dependencies or complex patterns in time-series datasets. Integrating our LDN model addresses this by enabling richer temporal feature extraction, enhancing performance on time-series and streaming data. This approach improves inference performance compared to a baseline HDC system that lacks dedicated feature extraction capabilities.

- **A Fully Digital HDC Architecture:** In this work, we present a fully digital HDC architecture. The design utilizes a 7P5T-RVT standard-cell logic library, enabling a hardware-friendly implementation using only digital logic elements. This approach ensures compatibility with standard CMOS processes, simplifying integration and scalability by eliminating the need for custom analog circuitry.
- **FPGA Prototyping and ASIC Implementation:** By using universally compatible digital components, the design remains portable, enabling swift FPGA prototyping and easy migration to high-performance, low-power ASICs for final deployment.
- **Comprehensive Evaluation:** A comprehensive evaluation includes both FPGA and ASIC implementations to assess the design's performance, area, power efficiency, and scalability across platforms. This dual analysis ensures the solution is not only functionally robust but also optimized for real-world deployment in diverse hardware environments.

II. Overview of Legendre Delay Network (LDN)

The Legendre Delay Network (LDN) is a type of Recurrent Neural Network (RNN) or a dynamical system that approximates a continuous-time delay of an input signal [9]. The delay of a signal $u(t)$ by θ time-steps duration is written as:

$$y(t) = u(t - \theta) \tag{1}$$

This delay can be implemented by leveraging the sifting property of the Dirac delta function, where convolution with a shifted impulse $\delta(t-\theta)$ (henceforth, $\delta_\theta(t)$) yields:

$$y(t) = u(t) * \delta_\theta(t) = u(t - \theta) \tag{2}$$

Taking the Laplace transform of both sides, we get:

$$\frac{Y(s)}{U(s)} = \mathcal{L}[\delta_\theta(t)] = e^{-\theta s} \tag{3}$$

The corresponding transfer function $F(s) = e^{-\theta s}$ cannot be directly converted into a finite-order state-space model. However, Voelker et al. [8] provide an approximation of $F(s)$ through a Linear Time Invariant (LTI) system denoted by the following state-space equations:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{4}$$

$$y(t) = Cx(t) + Du(t) \tag{5}$$

In our LDN, we use only the Eq (4); there, the values of matrices A and B can be obtained as follows:

$$A_{i,j} = \frac{(2i+1)}{\theta} \begin{cases} -1, & i < j \\ (-1)^{i-j+1}, & i \geq j \end{cases} \tag{6}$$

$$B_i = \frac{(2i+1)}{\theta}(-1)^i \tag{7}$$

for $i, j \in [0, d-1]$, where d is the order (or dimensionality) of the state vector $x(t) \in \mathbb{R}^d$. Note that the Eqs (4) and (5) are in continuous time, therefore, for discrete-time implementation, we use the following discretized equation (of Eq (4)):

$$x[t+1] = \mathbf{A}x[t] + \mathbf{B}u[t] \tag{8}$$

where $\mathbf{A}$ and $\mathbf{B}$ are obtained from A and B respectively using Zero Order Hold method (with $\Delta t = 0.001s$). Henceforth, we call the d-dimensional state-vector $x[t]$ as the temporal features extracted/computed by the LDN from the input signal $u[t]$.

Thus, the tunable parameters of the discrete-time LDN are:

- d: the dimensionality of the LDN's state vector, and
- θ: the length of the temporal/memory window

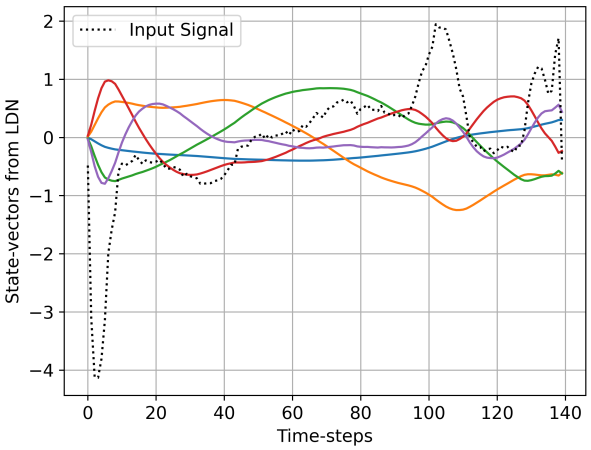

Fig. 1: ECG Signal after passing through LDN Model.

Each combination of (d, θ) defines a unique LDN with its own characteristic dynamics. In summary, the LDN provides a principled and efficient mechanism to extract temporal features, enabling high-performance, low-complexity spiking models for time-series tasks [9], [10]. In Fig. 1, we illustrate a time-varying input signal, and the 5-dimensional signals/temporal features obtained after passing it through an LDN model. Note that these

features can be encoded to spikes using Rate Encoder spiking neurons. In the next section we explain our rate encoding scheme along with the Leaky Integrate-and-Fire (LIF) neuron equations.

A. Rate Encoding LDN's temporal features

Since the temporal features extracted by the LDN are continuous-valued, they must be encoded to spikes for further processing. To achieve this, we employ the LIF neuron as rate encoder. Given that the LDN outputs can contain both positive and negative values (as illustrated in Fig. 1), we offset the LDN extracted features by a value to make all the LDN signals positive (more details in Sec IV-1). Additionally, we concatenate the offset LDN signals one after the other; this way, only one LIF neuron sensitive to positive values is needed to encode the concatenated signal to binary spikes (see Fig. 2). The equations of the LIF neuron's dynamics are below:

$$I[t] = (1-\alpha)I[t-1] + x[t] \tag{9}$$

$$V[t] = (1-\beta)V[t-1] + I[t] \tag{10}$$

$$S[t] = \Theta(V[t] - V_{\text{thr}}) \tag{11}$$

In Eqs (9) and (10), $0 \leq \alpha < 1$ and $0 \leq \beta < 1$ respectively are the encoding neuron's current decay and voltage decay. Likewise, $I[t]$ and $V[t]$ in Eqs (9) and (10) respectively are LIF neuron's current and voltage. In Eq (11), Θ denotes the Heaviside function that accepts the difference of $V[t]$ and the voltage threshold V_{thr}, thereby producing a value of 1 (i.e., a spike) if the difference is positive. Also, once the LIF encoding neuron spikes, its voltage $V[t]$ is reset to 0.

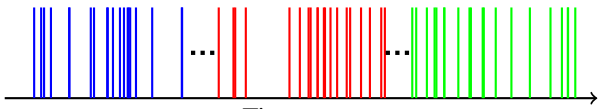

Fig. 2: Concatenated LDN signals' spike trains

III. Overview of Hyperdimensional Computing

HDC is a brain-inspired computational framework that uses high-dimensional vectors (hypervectors) to represent and manipulate information in a distributed manner [11]. Hypervectors, typically of dimension $D \gg 1000$, offer properties such as robustness to noise, fault tolerance, and massive parallelism. The fundamental operations in HDC are binding, bundling, and permutation.

Binding encodes the association between entities by component-wise multiplication:

$$\mathbf{H}_{A=a} = \mathbf{A} \circ \mathbf{a} \tag{12}$$

where $\mathbf{A}$ is a variable hypervector, $\mathbf{a}$ is a value hypervector, and $\circ$ denotes element-wise multiplication:

$$(\mathbf{A} \circ \mathbf{a})_i = A_i \cdot a_i, \quad \forall i \in \{1, 2, \ldots, D\} \tag{13}$$

Bundling combines multiple hypervectors into one using addition and majority voting:

$$\mathbf{S} = \sum_{i=1}^{n} \mathbf{h}_i \tag{14}$$

$$\mathbf{H}_{\text{bundle}} = \text{sign}(\mathbf{S}) = \begin{cases} +1, & S_i > 0 \\ -1, & S_i < 0 \\ \text{random}(\pm 1), & S_i = 0 \end{cases} \tag{15}$$

Permutation encodes sequential or positional information by rearranging dimensions. A simple form is cyclic shift:

$$\rho(\mathbf{x}) = [x_D, x_1, x_2, \ldots, x_{D-1}] \tag{16}$$

Temporal encoding of sequences becomes:

$$\mathbf{S} = \sum_{t=1}^{T} \rho^t(\mathbf{x}_t) \tag{17}$$

N-gram encoding captures order-sensitive patterns across multiple time steps. For a sequence of n vectors $\mathbf{x}_{t-n+1}, \ldots, \mathbf{x}_t$, the n-gram representation is constructed as:

$$\mathbf{G}_t = \rho^{n-1}(\mathbf{x}_{t-n+1}) \circ \rho^{n-2}(\mathbf{x}_{t-n+2}) \circ \cdots \circ \mathbf{x}_t \tag{18}$$

This composition preserves both identity and order of elements in the sequence window.

Similarity search is done using cosine similarity or Hamming distance:

$$\text{cosine}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{19}$$

$$\text{Hamming}(\mathbf{x}, \mathbf{y}) = \frac{1}{D} \sum_{i=1}^{D} \mathbb{I}[x_i \neq y_i] \tag{20}$$

Finally, HDC enables classification via prototype vectors. For class c with prototype hypervector $\mathbf{P}_c$, the prediction is:

$$\hat{y} = \arg\max_c \text{cosine}(\mathbf{H}, \mathbf{P}_c) \tag{21}$$

These simple yet powerful operations allow HDC systems to learn and infer rapidly with minimal training, low memory usage, and strong generalization, making them well-suited for deployment on energy-efficient, resource-constrained hardware.

IV. Proposed LDN Empowered HDC Architecture

In this section, we present a detailed explanation of the proposed HDC architecture by examining its hardware implementation, timing diagram, and operational aspects, while decomposing the design into its various integral sub-blocks. The top-level block diagram is illustrated in Fig.-3, and its timing diagram can be found in

Fig.-4. The proposed HDC architecture operates with a dimension (D) of 1024-bit, meaning that each calculation or signal flow between blocks processes 1024-bit binary data.

*1) **Dataset preprocessing for Simulation in FPGA and ASIC** :* Any time series dataset may contain signals with positive and negative values. In step-1, the LDN transforms input signals into four state-space vectors (see Fig.-1). Each of the four state-space vectors is processed by LIF neurons later at encoder stage, resulting in a total of four spike trains. In step-2, we identify the most negative value across all signals in the dataset. For example, in ECG5000 the maximum negative value is -5.79. Offsetting the input signal is necessary because CMOS technology is optimized for positive supply ranges (0 to VDD). Applying negative inputs risks forward-biasing internal P–N junctions, which can induce latch-up, compromise signal integrity, and accidentally engage ESD protection elements. In step-3, a uniform offset is applied to shift the entire dataset into the positive range before feeding the signals into the LDN encoder. In step-4, the class-wise signals are passed through the encoder to generate binary bit streams. Finally, in step-5, the signals generated by encoder block are concatenated (see Fig.-2) based on their respective classes to pass to the HDC system for training and testing.

*2) **LDN Encoder Design**:* For the encoder, we designed a LIF neuron [12]. The LDN provides time-varying input signals, which are fed into the neuron's `I_NRN` terminal. The firing threshold of the neuron can be configured using the `VTH` terminal. The optimized value for `VTH` is 14, and `V_LEAK` is 0.6 for the dataset we tested. When the voltage across the capacitor exceeds the threshold, the neuron generates a spike at the `SPK_OUT` terminal. The neuron is then reset and gets ready to integrate next input signals. The neuron works only when it gets the `CLK` signal high. Digital block `serial-in-parallel-out (SIPO)`, and `first-in-first-out(FIFO)` is used to collect and process the spike streams.

*3) **Base HV Generation**:* This subsection describes the generation of the Random Number Hypervector (RNHV), which serves as a foundational component in the HDC system. In the HDC architecture, `Base Hypervectors (BHVs)` generated by the `Random Number Generator (RNG)` block is utilized for the encoding or binding of `feature hypervectors (FHVs)` and `query hypervectors (QHVs)` (Fig.-3- **A**). These BHVs provide essential properties such as maximal orthogonality and uniform distribution within the vector space, which are critical for efficient associative pattern recognition and robustness against noise.

In this work, we propose a 32-bit RNG block that performs regular sampling of an irregular waveform, where the irregular signal is generated by XORing `Galois and Fibonacci Ring Oscillators (GFRO)`. To enhance randomness, the proposed RNG design introduces a two-layered approach: a chained ring oscillator generates seed data, which is then passed through a SIPO before feeding into the main RNG block. This layered structure improves the quality of randomness and ensures the generation of distinct patterns from the GFROs due to the use of unique seed values. Table-I represents the qualification standards of the proposed RNG.

TABLE I: NIST SP 800-22 Test Results for Proposed RNG

Test	P-value	Pass Rate	P-value	Pass Rate
Frequency	0.4036	98/100	0.5111	94/100
Block Frequency	0.4108	93/100	0.4703	87/100
Cumulative Sums	0.4567	94/100	0.5173	81/100
Runs	0.5072	96/100	0.5007	97/100
Longest Runs	0.4178	97/100	0.4874	92/100
Rank	0.4843	89/100	0.2622	57/54
FFT	0.2738	52/54	0.52165	88/100

*4) **Spatio-Temporal Encoding (Training)**:* During the training phase, HVs corresponding to the same class are bundled using a bitwise majority operation. This operation selects the most frequent bit (either 0 or 1) at each position across the input HVs. For example, $\text{MAJ}(0,0,1) = 0$ and $\text{MAJ}(1,0,1) = 1$. Related HVs are first bundled and then bound with a BHV for encoding (Fig.-3- **B**).

For a classification task involving k classes, the HDC architecture generates k distinct class HVs. The bundled HV for the i-th class is computed as shown in Equation-22.

$$C_i = S_1^i + S_2^i + \cdots + S_k^i \tag{22}$$

The encoded hypervector is then formed by binding each individual HV S_j^i with a class identifier ID using a bitwise XOR operation:

$$HV_{En} = ID \oplus S_1^i + ID \oplus S_2^i + \cdots + ID \oplus S_k^i \tag{23}$$

To handle temporal or sequential data, HDC employs an n-gram training scheme, which encodes ordered patterns across time windows. Given a sequence of n hypervectors $\{H_{t-n+1}, \ldots, H_t\}$ extracted from time-dependent input features, an n-gram hypervector is generated by applying a cyclic permutation $\rho(\cdot)$ and binding:

$$G_t = \rho^{n-1}(H_{t-n+1}) \oplus \rho^{n-2}(H_{t-n+2}) \oplus \cdots \oplus H_t \tag{24}$$

This G_t hypervector represents a temporally ordered n-gram and captures both the identity and order of input

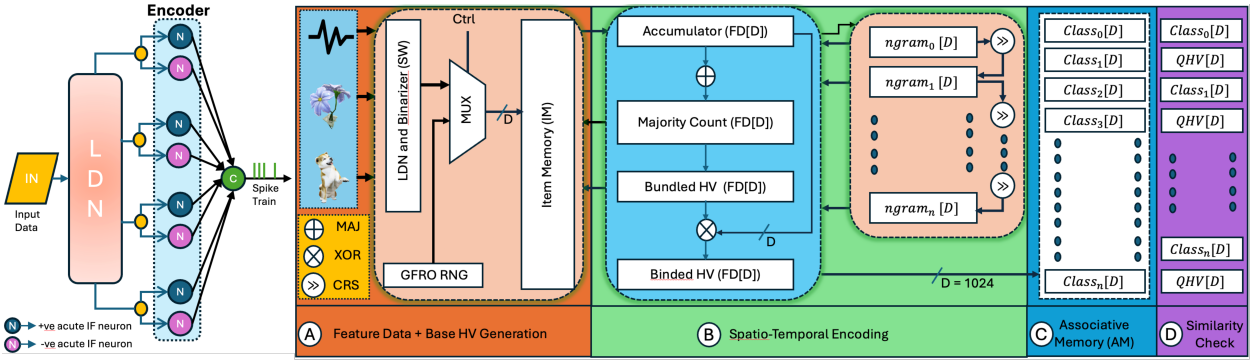

Fig. 3: Top-Level HDC Hardware Block Diagram.

signals across a fixed window. These n-gram HVs are then bundled together (using the majority operation) across all training examples belonging to a particular class, resulting in robust and order-aware class HVs:

$$C_i^{(n\text{-gram})} = \text{Majority}(G_1^i, G_2^i, \cdots, G_m^i) \tag{25}$$

The n-gram strategy significantly improves classification in tasks where sequence or time-order carries discriminative information, such as in biosignals, speech, or gesture recognition.

*5) **Associative Memory**:* After the training phase is completed, the resulting HVs corresponding to different classes are stored in the `ASSOCIATIVE_MEMORY` for inference operations. The `ASSOCIATIVE_MEMORY` is configured with a size of 1024×32, allowing storage for up to 32 class hypervectors, each with a dimensionality of 1024 bits (Fig.-3- C). Scaling the class count requires larger associative memory arrays, increasing both silicon area and power consumption.

*6) **Similarity Check or Hamming Distance (Testing)**:* The inference process begins by encoding the QHV with the same BHV used for training. A similarity check is then performed between the encoded QHV and each class hypervector stored in the `ASSOCIATIVE_MEMORY`. As outlined in Equation-26, the predicted class is determined by identifying the class HV with the minimum Hamming distance to the QHV (Fig.-3- D).

$$\text{ClassPred}(j) = \arg \min_{i \in \{1,\ldots,k\}} \delta \left(\sum_{j=1}^{d} \mathbf{QHV}(j) \oplus \mathbf{S}_i(j) \right) \tag{26}$$

Here, δ denotes the normalized Hamming distance, computed via bitwise XOR. Algorithm-1 shows the similarity check ASIC implementation logic.

The QHV is XORed with each class HV, and the number of 1s' in the result is counted. The class with the minimum count of ones indicates the highest similarity and thus the predicted class.

Algorithm 1: Similarity Check Block Logic

```
Data: Ready_to_Execute
while SmC_En == 1 and CLK_En == 1 do
    for Class ≤ N do
        if Ready == 1 and count == 0 then
            Regs[N][D], X[D] ←
              class_HV[i] ⊕ QHV;
            Count[1s][N] ← Regs[N][D], X[D];
        end
        Pred_Label ← class[min(counts[1s]) :
          N − 1];
    end
end
```

V. FPGA and ASIC Implementation Details of the Proposed HDC Architecture

The entire architecture is developed using SystemVerilog HDL. Initially, the design is evaluated on the Altera Cyclone-V FPGA evaluation board. Following the FPGA evaluation, the design is implemented using commercial GlobalFoundries 12LP FinFET technology node for further optimization and performance analysis.

A. *Evaluation in FPGA*

FPGA-based evaluation offers rapid prototyping, agile hardware modifications, flexible and time-efficient environment for functional validation and performance analysis. In FPGA implementation, the proposed HDC system requires a total power of `2.922W` for operation, consisting of `0.177W` static power and `2.746W` dynamic power. With a clock period of `10ns` (corresponding to a frequency of 50 MHz) and a load capacitance of `5pF`, the design achieves a Worst Negative Slack (WNS) of `0.274ns`, indicating that all timing paths meet their required constraints. Table-II summarizes all the power and timing metrics.

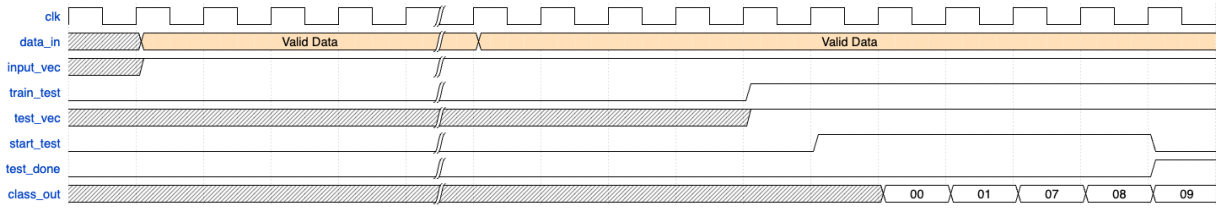

Fig. 4: Timing Diagram (CIFAR-10 Dataset Testing with Gate-Level Netlist).

TABLE II: FPGA Power and Timing Metrics

Parameters	Dynamic Power (W)	Timing (ns)
Clocks	0.175	WNS = 0.274 WHS = 0.132
Signals	1.199	WPWS = 2.00
Logic	1.368	TNS/THS/ TPWS = 0
IOs	0.005	Period = 20
Note: **WNS = Worst Negative Slack, TNS = Total Negative Slack** **WHS = Worst Hold Slack, THS = Total Hold Slack** **WPWS = Worst Pulse Width Slack** **TPWS = Total Pulse Width Negative Slack**		

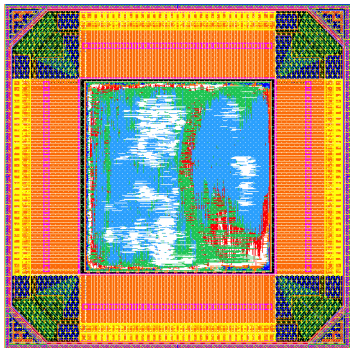

Fig. 5: APR View of the Proposed HDC in 12LP.

B. Implementation in ASIC

Following the FPGA-based evaluation, the design is implemented as an ASIC. The SystemVerilog HDL code is first synthesized with design constraints using `Cadence Genus`. The resulting gate-level netlist was then simulated using the same testbench used during golden RTL verification to ensure functional consistency. Additionally, `Cadence Conformal` is used to perform Logic Equivalence Checking (LEC) between the RTL and synthesized (gate-level) netlist. Subsequently, automatic place and route (APR) was carried out using `Cadence Innovus` to complete the physical design flow. For the ASIC implementation, we used the GlobalFoundries 12LP FinFET technology as the Process Design Kit (PDK), along with the 7P5T RVT Standard Cell Library provided by Synopsys IP vendor.

TABLE IV: Cadence Innovus HDC Power Metrics.

Category	Leakage	Internal	Switching	Total(mW)
Register	0.02046	7.96200	0.5371	8.5190
Logic	0.09712	11.1200	5.5660	16.780
Clock	0.00022	0.06590	1.9450	2.0110
Subtotal	**0.11780**	**19.15**	**8.048**	**27.31**

The chip operates with a total of 16 I/O pins and a single master clock. The design integrates FIFOs, counters, adders, and SPI interfaces to manage data flow efficiently. Flip-flops are employed for data storage and to support both training and testing operations.

From APR the measured total silicon area with PAD cells and seal-ring occupied by the chip is $0.5\ \mathrm{mm}^2$ (core area - $0.15\ \mathrm{mm}^2$). The design consumes total $27.31\ \mathrm{mW}$ power. Its detailed power metrics are shown in table-IV generated from Cadence Innovus ASIC physical design implementation tool.

The system's throughput is determined by the serial interface, which requires 1,024 clock cycles to load a single hypervector. At an operating frequency of 50 MHz, this results in a processing rate of approximately 48,800 inferences/second, with a latency of 20.48 μs per sample.

VI. Timing Diagram

As illustrated in Fig.- 4, a single clock (`clk`) drives the whole operation. Through `input_vec` it takes all the input require for training, testing and functionality verification. When `start` signal goes high it takes all the input values (HVs). The testing starts when `start_test` is set to high. For the training of CIFAR-10 dataset, we take 3 vectors of each class having a total of 30 FHVs. They get bundled and bound and saved to the associative memory. For testing we take one sample from each class (here class-9). Output port `test_done` confirms that the test has been completed. And the `class_out` port predicts the class to be a class-9 image.

VII. Experimentation with Different Datasets

To evaluate the classification improvement enabled by our integrated LDN-HDC system, we conducted experiments using ECG5000, CIFAR-10, and MNIST (with default train/test split) as these datasets offer practical

TABLE III: Comparison between LDN-HDC and other HDC hardware processors.

HDC Details	ISCAS'24 [13]	TCAD'23 [14]	JETCAS'19 [15]	This Work
Configuration	SNN + HDC	SCNN + HDC	HDC	LDN + HDC
Dimension	2K	4K	2K	1K
Accuracy	MNIST: 94.4% N-MNIST: 94.8% DVS-Gesture: 87.1%	N-MNIST: 95.2% DVS-Gesture: 85.3% DVS-ASL: 87.8%EMG: 96.6%	MNIST: 76% EMG: 96%	ECG5000: 93.33% (WoLDN-84.91%) CIFAR-10: 50.7 (WoLDN-46.29%)% MNIST: 96.4% (WoLDN-83.75%)
Technology	40 nm (Simulated)	28 nm (Simulated)	28 nm (Simulated)	12 nm (Simulated)
Chip Area	0.88 mm^2	2.70 mm^2	1.27 mm^2	0.5 mm^2

[a] Significant accuracy improvement despite having lower dimension (D=1024), and silicon area.

applications for edge devices. We used the sequential versions of the static CIFAR-10 and MNIST images, i.e., we flattened the individual image samples and fed one scalar per timestep to our LDN-HDC system (similar to [8], [16]). Note that such sequential image samples are frequently used to evaluate temporal processing models [16] - as means of stress-testing them; although, spatial processing models should be considered for better performance. For the ECG5000, MNIST, and CIFAR-10 datasets, we set $\theta = 120$, 360, and 480 respectively (chosen arbitrarily). Note that we have set the order d=4 in our LDN, although, increasing d can help further improve the accuracy of our LDN-HDC model [17]. In the initial phase, we tested our base HDC model without the LDN feature extractor and recorded the classification accuracies. Subsequently, we repeated the experiments with the LDN module integrated into the HDC system using the same datasets. Table-III presents the accuracy comparison among other HDC model (and their processors) and the LDN-enhanced HDC model (LDN-HDC) along with without LDN (WoLDN) model. Considering the non-HDC based works, we found that RNNs/LSTMs obtain 98.43% on ECG5000 [18], and 98.90% & 63.01% on MNIST and CIFAR-10 respectively [16].

VIII. Conclusion

In this work, we presented an LDN-empowered HDC architecture. We evaluated our design on the Intel Altera Cyclone-V FPGA evaluation board, followed by an ASIC implementation using GlobalFoundries 12nm Low Power (LP) technology. The FPGA evaluation indicates a static power consumption of `0.177W` and a dynamic power consumption of `2.746W`. The ASIC implementation utilizes approximately `564K` instances, consumes $27.31\,\mathrm{mW}$ power and occupies a silicon area of $0.5\ \mathrm{mm}^2$. Overall, our novel LDN-integrated HDC architecture demonstrates significant improvements in both power efficiency and classification performance.

References

[1] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 849–13 875, 2021.

[2] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent progress and development of hyperdimensional computing (hdc) for edge intelligence," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 119–136, 2023.

[3] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023.

[4] A. X. Manabat, C. R. Marcelo, A. L. Quinquito, and A. Alvarez, "Performance analysis of hyperdimensional computing for character recognition," in *2019 International Symposium on Multimedia and Communication Technology (ISMAC)*, 2019.

[5] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-dimensional computing challenges and opportunities for ai applications," *IEEE Access*, vol. 10, pp. 97 651–97 664, 2021.

[6] A. R. Voelker, "Dynamical systems in spiking neuromorphic hardware," 2019.

[7] N. R. Chilkuri and C. Eliasmith, "Parallelizing legendre memory unit training," in *International conference on machine learning*. PMLR, 2021, pp. 1898–1907.

[8] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre memory units: Continuous-time representation in recurrent neural networks," *Advances in neural information processing systems*, 2019.

[9] R. Gaurav, T. C. Stewart, and Y. Yi, "Reservoir based spiking models for univariate time series classification," *Frontiers in Computational Neuroscience*, vol. 17, p. 1148284, 2023.

[10] R. Gaurav and et al., "Legendre-snn on loihi-2: Evaluation and insights," in *NeurIPS 2024 Workshop Machine Learning with new Compute Paradigms*.

[11] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[12] F. Nowshin, Y. Huang, M. R. Sarkar, Q. Xia, and Y. Yi, "Merrc: A memristor-enabled reconfigurable low-power reservoir computing architecture at the edge," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 1, pp. 174–186, 2024.

[13] Xiao and et al., "Spiking-hdc: A spiking neural network processor with hdc classifier enabling transfer learning," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024.

[14] T. Zhang, J. Morris, K. Stewart, H. W. Lui, B. Khaleghi, Thomas, and et al., "Hyperspikeasic: Accelerating event-based workloads with hyperdimensional computing and spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3997–4010, 2023.

[15] S. Datta, R. A. G. Antonio, A. R. S. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, 2019.

[16] A. Gu, C. Gulcehre, T. Paine, M. Hoffman, and R. Pascanu, "Improving the gating mechanism of recurrent neural networks," in *International conference on machine learning*. PMLR, 2020.

[17] R. Gaurav, S. Agarwal, T. C. Stewart, and Y. Yi, "Benchmarking deep legendre-snn for time series classification – analysis and enhancements," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–12, 2025.

[18] J. Pereira and M. Silveira, "Unsupervised representation learning and anomaly detection in ecg sequences," *International Journal of Data Mining and Bioinformatics*, vol. 22, no. 4, 2019.