AFRL-RI-RS-TR-2019-171

RESERVOIR COMPUTING AND BENCHMARKING OF NEUROMORPHIC SYSTEMS FOR SWAP-CONSTRAINED AUTONOMOUS PROCESSING

ROCHESTER INSTITUTE OF TECHNOLOGY

AUGUST 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-171 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
CLARE D. THIEM
Work Unit Manager

/ S /
CHRISTOPHER FLYNN
Acting Technical Advisor,
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUGUST 2019	FINAL TECHNICAL REPORT	FEB 2016 – FEB 2019

4. TITLE AND SUBTITLE

RESERVOIR COMPUTING AND BENCHMARKING OF NEUROMORPHIC SYSTEMS FOR SWAP-CONSTRAINED AUTONOMOUS PROCESSING

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-16-1-0108

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)
Dhireesha Kudithipudi, Nicholas Soures, Abdullah Zyarah, Swatika Ramakrishnan, Humza Syed, Qutaiba Saleh, Ryan Brüll, Dan Christiani

5d. PROJECT NUMBER
T2DN

5e. TASK NUMBER
TR

5f. WORK UNIT NUMBER
IT

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rochester Institute of Technology
1 Lomb Memorial Drive
Rochester, NY 14623-5603

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2019-171

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Random projection networks are a class of learning algorithm, which use high-dimensional random projections of information as a basis for training simple linear classifiers. The advantage of this approach is the lite computational cost associated with learning and short training times. These advantages make random projection networks suitable candidates for implementing on-device learning systems for on-the-edge intelligence. In particular, we focus on a feedforward RPN known as the Extreme Learning Machine (ELM) as a baseline for spatial information, and two recurrent RPNs, known as the Liquid State Machine (LSM) and Echo State Network (ESN), for processing spatio-temporal information. The fist focus of this work was to investigate advancements at an algorithmic level to make the algorithms more computationally powerful and hardware friendly. Secondly, we developed several baseline architectures for stochastic, digital, and analog implementations for size, weight, and power (SWaP) constrained platforms with on-device learning.

15. SUBJECT TERMS
Reservoir Computing, Memristors, Random Projection Networks, Echo State Networks, Liquid State Machines

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**CLARE D. THIEM**
U	U	U	UU	92	19b. TELEPHONE NUMBER (*Include area code*) **NA**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1.0 SUMMARY

Random projection networks (RPNs) are a class of learning algorithm, which use high-dimensional random projections of information as a basis for training simple linear classifiers. The advantage of this approach is the lite computational cost associated with learning and short training times. These advantages make RPNs suitable candidates for implementing on-device learning systems for on-the-edge intelligence. In particular, we focus on a feedforward RPN known as the Extreme Learning Machine [1] (ELM) as a baseline for spatial information, and two recurrent RPNs, known as the Liquid State Machine [2] (LSM) and Echo State Network [3] (ESN), for processing spatio-temporal information. The fist focus of this work was to investigate advancements at an algorithmic level to make the algorithms more computationally powerful and hardware friendly. Secondly, we developed several baseline architectures for stochastic, digital, and analog implementations for size, weight, and power (SWaP) constrained platforms with on-device learning.

2.0 INTRODUCTION

The three RPN networks we investigate share the same basic three-layer architecture, which consists of an input layer, a hidden layer, and an output layer. The primary difference between the ELM shown in Figure 1 and the recurrent RPNs (LSM and ESN) shown in Figure 2 is the presence of recurrent connections in the hidden layer. The underlying intuition behind the operation of these three networks is that projecting the input space into a high-dimensional space will produce a linearly separable representation. This in turn allows a simple output layer to learn linear decision boundaries separating different classes, reducing the complexity of training.

The ELM only allows for information to propagate in a single forward direction during inference. This makes the network suitable for spatial or static tasks which do not depend on a temporal component. The initial projection from the input layer to the hidden layer is randomly initialized and kept fixed. This can be given by

$$H = f(W_{in} * u) \tag{1}$$

where W_{in} is a matrix which describes the random projection from the input signal u, to the hidden layer H and f represents a non-linear transformation such as the sigmoid function. The classifiers response can then be computed by

$$Y = f(W_{out} * H) \tag{2}$$

where W_{out} is the matrix describing the connections from the hidden layer H to the classifier. The connections represented by W_{out} can then be trained using the delta-rule or regression to produce the correct response Y. Due to the need to only train a single set of weighted connections (weights), the training complexity and time needed to optimize an ELM to a specific task is much less than conventional neural networks. In addition, because there is no backpropagation of error signals, there is not catastrophic interference between output nodes when learning different responses which allows the network to generalize to multiple applications at once.

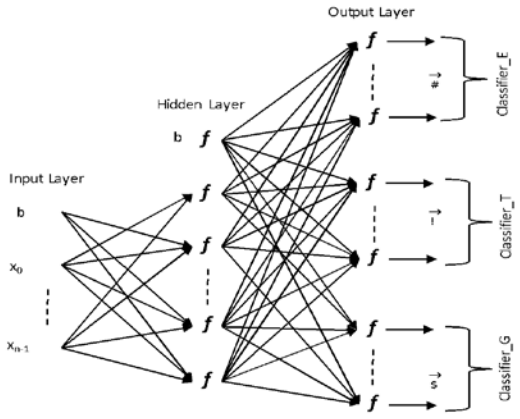

Figure 1. High Level ELM Architecture

We will refer to the recurrent RPNs as reservoir computing (RC) networks. RC networks are similar to the ELM, however the hidden layer is a 3D reservoir of neurons with recurrent connections rather than a single layer of feedforward neurons. The recurrent connections in the hidden layer provide RC networks

with a temporal feedback that does not exist in the ELM. This feedback gives the RC network a fading memory, allowing it to solve complex temporal applications. The reservoir (equivalent to the ELM's hidden layer) is now updated by

$$H(t) = f(W_{in} * u(t) + W_{rec} * H(t-1)) \quad (3)$$

where W_{in} is the connection matrix describing the random projection from the input layer to the reservoir and W_{rec} is the connection matrix describing the recurrent connections in the reservoir. Similar to the ELM, a classifier is then trained to produce a desired response Y(t) based on the state of the reservoir where

$$Y(t) = f(W_{out} * H(t)) \quad (4)$$

and W_{out} can be trained to produce the desired response through simple learning algorithms such as the delta rule or regression.

The two RC networks studied in this work are the ESN and LSM. Both networks employ the same architecture; however, the ESN is a rate-based model that uses sigmoid activation functions for the reservoir layer neurons. In the ESN, when the recurrent connections are initialized, they are normalized to have a maximum eigenvalue of one. This is done to make sure the activity caused by previous inputs will eventually fade out of the network. A modification over the original ESN is to use a leaky neuron in the reservoir layer so that the update equation becomes

$$H(t) = \alpha * f\big(W_{in} * u(t) + W_{rec} * H(t-1)\big) + (1-\alpha) * H(t-1) \quad (5)$$

where α is the leaky rate which determines how much of the previous reservoir state to forget and how much new information to write to the reservoir state.

The LSM is a spiking neural network that in this work is modelled using leaky integrate-and-fire (LIF) neurons in the reservoir layer. The major difference between the LIF neurons in the LSM and the sigmoidal neurons in the ESN is that the LIF neurons integrate information over time and have a binary output response. Aside from the neurons, there are also differences between how the LSM processes information and is initialized compared to the ESN. The inputs to the LSM can either be represented as analog current signals, or converted into binary spike trains. Another unique property of the LSM's implementation is that the recurrent connections are generated randomly based on the Euclidean distance between neurons.

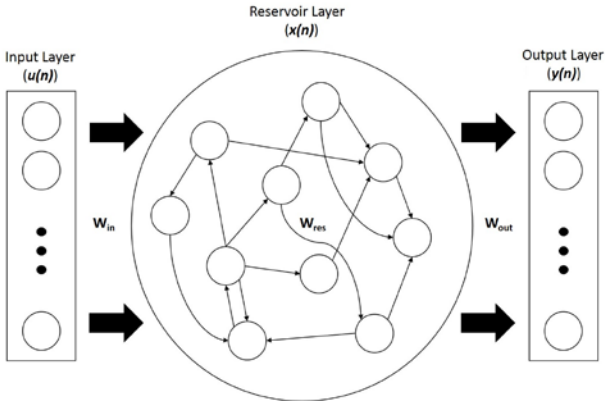

Figure 2. High Level RC Architecture

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Software Models

3.1.1 Extreme Learning Machine. The ELM algorithm has two forms of training which can be employed; i) online learning using regression or the delta-rule, and ii) solving for the optimal set of output weights with the Moore-Penrose pseudo inverse. For hardware efficiency and computational efficiency, we have investigated a modified delta-rule which uses a signed approximation [4]. Additionally, we have begun investigating techniques for simplifying the computational cost of the Moore-Penrose pseudo inverse through dimensionality reduction and iterative approximations.

3.1.2 Echo State Network. A common critique of RC models, including the ESN, is their sub-par performance in comparison to state-of-the-art (SOTA) models. This has been primarily attributed to the dependence on initialization and the inability of a shallow architecture to capture long-term temporal dependencies. To address this, many research groups have begun proposing deep ESN models such as [5] and [6]. In this research we have proposed two models for overcoming limitations of a traditional ESN.

The first model we developed is known as Convolutional Drift Networks (CDNs) [7]. CDNs use pre-trained spatial processing techniques, such as convolutional neural networks, as a feature extraction layer shown in Figure 3. The advantage of the CDN is that the feature extraction of the pre-trained CNN captures spatial dependencies and information that is often lost, or incapable of being represented by random projections. This allows the attached ESN to process much more meaningful information, resulting in higher performance while maintaining the low computational cost associated with training RC networks. Another feature of the proposed CDN to handle information over longer time-scales is the temporal averaging that occurs before passing information to the output layer.

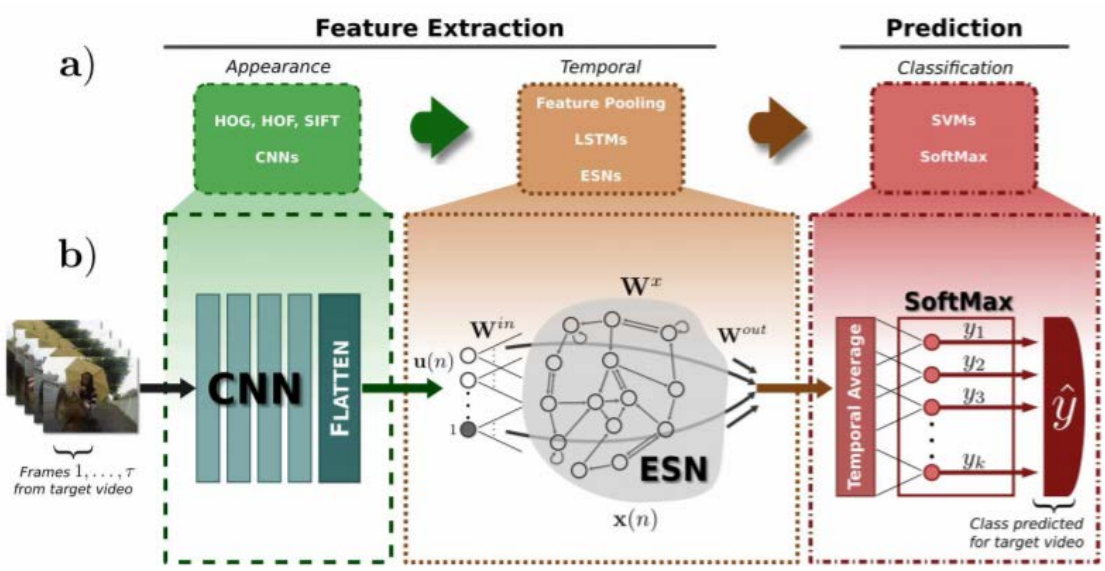

Figure 3. Architecture for the CDN on video activity recognition

The second model we proposed is the Mod-DeepESN [8], which is a modular architecture comprised of multiple reservoirs shown in Figure 4. The Mod-DeepESN is an approach that focuses on using modular reservoirs to build deep and/or wide architectures. Width, refers to multiple reservoir modules connected to the input space. This essentially builds an ensemble of reservoirs which can learn different temporal

features of the input, as well as share information with one another. An advantage of this approach is that it can reduce the computational cost of a traditional ESN by reducing the number of recurrent weight matrices and also act as an ensemble of reservoirs to mitigate the impact of random initialization. Depth refers to stacked reservoir layers which captures temporal information over multiple timescales. This increases the temporal memory capacity of the network and boosts performance on complex temporal applications.

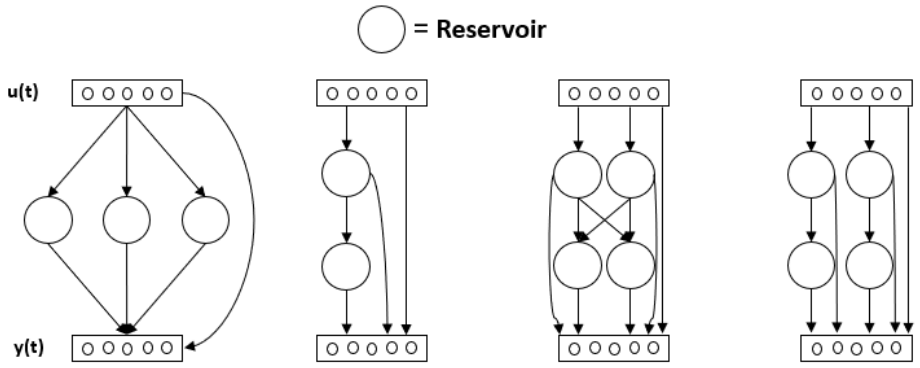

Figure 4. Different architectures for the Mod-DeepESN consisting of both deep and wide reservoir chains

There are also two features that are added to the reservoir implementation in the Mod-DeepESN that have been proposed in literature. For weight initialization, it is important to ensure the *Echo State Property* holds to have stability in the reservoir dynamics. In the Mod-DeepESN, three different initialization techniques can be chosen. The first technique is to ensure that the spectral radius of the reservoir weights is less than one (as done in the original ESN). The second approach is to choose a hyper parameter, σ, such that the Euclidean distance between weights is equivalent to σ. Lastly, the weights can be drawn from a normal distribution with a mean of zero where the standard deviation is determined by the fan in and fan out of each neuron as shown in (6).

$$\sigma = \sqrt{2/(n_{in} + n_{out})} \tag{6}$$

Intrinsic plasticity is applied by introducing a gain and bias term to the *tanh* function such that *tanh(x)* → *tanh(gx + b)*. The parameters g and b are dynamic and updated according to (7) and (8) respectively.

$$\Delta b = -\eta\left(\left(\frac{-\mu}{\sigma^2}\right) + \left(\frac{x}{\sigma^2}\right)(2\sigma^2 + 1 - x^2 + \mu x)\right) \tag{7}$$

$$\Delta g = \eta/g + \Delta\, bx \tag{8}$$

3.1.3 Liquid State Machines. The third model we investigated in this work is the liquid state machine shown in Figure 5. One part of our research has focused on developing a set of guidelines that can be used to streamline the process of applying the LSM to real-world applications [9]. The input layer is typically treated as a layer of linear neurons that directly passes external input and projects it into the high-dimensional reservoir layer. However, different forms of pre-processing can also occur at the input layer such as normalization and scaling of the input data or conversion into spike trains. The advantage of converting into spike trains is that information can be represented by binary signals denoting firing activity and allowing time for information to propagate through the network, but the downside is that it adds latency to the information processing of the LSM. The connections between neurons in the input layer to the hidden layer are random and sparse, where the degree of sparsity depends on the application. In [10], the authors state that the granule cells need to connect to a sparse number of inputs to produce a unique high-dimensional representation. Using combinatorial math, they show that for N inputs, the degree of synaptic connectivity can sufficiently create an over-complete high-dimensional representation with values as low as 3% of N.

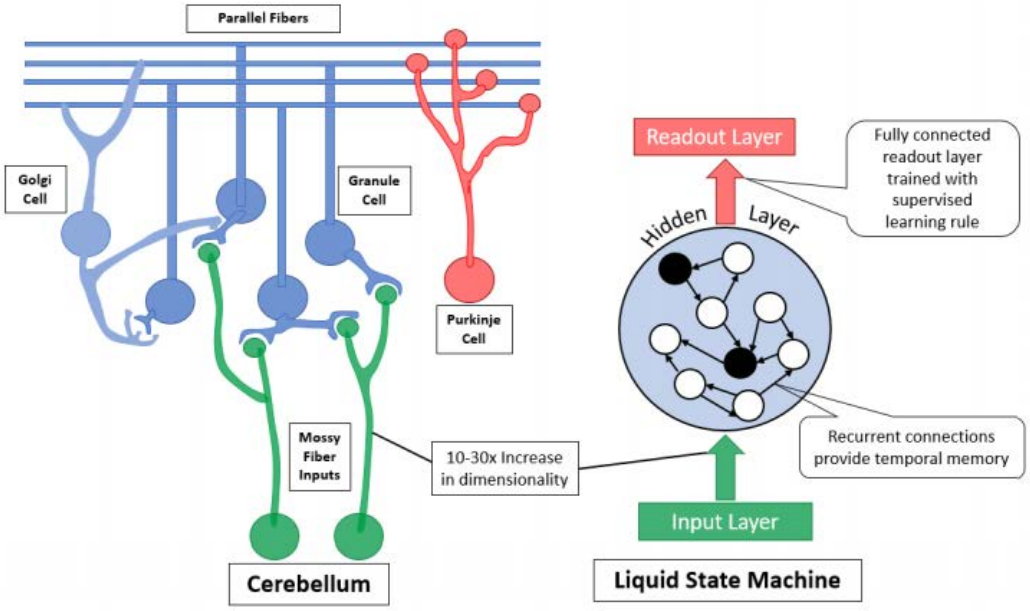

Figure 5. Similarity between information processing in the cerebellum and a Liquid State Machine

In the hidden layer, neurons are implemented as spiking neurons such as the LIF neuron model. LIF neurons model biological neurons by integrating information over time through the neurons membrane potential and emitting a spike when the potential reaches a specified threshold. The leakage term refers to a constant leak of charge regulated by the membrane time-constant as formulated in (9)

$$V_M = V_M - \frac{V_M}{\tau} + \frac{I*R}{\tau} \tag{9}$$

where V_M is the membrane potential, τ is the membrane time constant, R is the membrane capacitance, and I is the input current. As the hidden layer responsible for integrating temporal information and is the most critical for the LSM's performance, choosing appropriate neuron parameters is of paramount importance. The first parameter, the threshold voltage at which to emit a spike will regulate the firing rate of the hidden layer and the strength of stimulus required to emit a spike.

The hidden layer serves two primary roles; i) generating a high-dimensional and potentially linearly separable representation of the input, and ii) capturing temporal information about the dynamic nature of the input space. There are a few implementation details which are important when initializing the hidden layer. First, the LIF neurons in the hidden layer belong to two categories; excitatory neurons which emit positive spikes, and inhibitory neurons which emit negative spikes. The ratio of excitatory to inhibitory neurons is 4:1. The second component to the hidden layer initialization is the synaptic connections between neurons. One of the unique implementation methods utilized in the LSM is a probabilistic formation of connections based on the Euclidean distance between neurons and the neuron types. The probability of a connection is given by (10)

$$P = Ce^{\frac{D_{i,j}{}^2}{\lambda^2}} \tag{10}$$

where D represents the Euclidean distance between neurons, C is a scalar controlling the maximum probability of a connection forming based on the neuron types (*i.e.* it will be different for excitatory to excitatory connection than for excitatory to inhibitory connections), and λ controls how quickly the probability decreases with distance and also varies with neuron type. This allows a user to vary the network topology by controlling the number of connections formed and the dynamics of the network (long, and frequent excitatory connections will increase network activity whereas the same for inhibitory connections will decrease neuron activity). Once the connections are initialized, the they are assigned a random synaptic strength (or weight). This can be drawn from a variety of distribution such as normal or uniform and the magnitude can be dependent on the type of connection as well. However, we found maintaining an average homogeneous excitability among neurons to be beneficial for performance. This can be done through synaptic scaling where the sum of pre-synaptic excitatory and inhibitory connections for each neuron are normalized to be equal. Empirically, it was found that best performance occurs when the sum of excitatory synapses is twice that of the inhibitory synapses.

Though the recurrent connections are never trained, they are dynamic synapses regulated through short-term plasticity (STP). As another simplification proposed was to use a simplified model of STP given by (11)

$$S = S - \alpha(x - \beta) \tag{11}$$

where α and β control the recovery and depression of a synapse, resulting in a fixed increment or decrement every time-step based on the spiking activity and x is the LIF neuron output (1 if it spiked otherwise 0).

Neuron Connectivity: For the neuron parameters, ideally the neurons should match the timescale of the input stream and in some cases the target output. For the LIF neuron model described in (1), the steady-state firing rate can be computed by (12)

$$r(j) = \begin{cases} [\tau_{ref} - \tau_m \log\left(1 - \frac{V_{th}}{I*R}\right)]^{-1}, if\ I * R > V_{th} \\ 0, \quad Otherwise \end{cases} \tag{12}$$

where the firing rate r to a steady-state current j can be computed based on the refractory period τref, the membrane time-constant τm, and the threshold voltage Vth. To apply (12) to the LSM neurons, the steady-state current can be approximated by the mean current from the input data. This will allow one to predict the firing rate for different sets of parameters which can be tuned to match a desired firing rate or to ensure

that the neurons will be active enough to cause information to propagate through the hidden layer. However, once the hidden layer becomes active, the estimated firing rate will rise due to the recurrent connections increasing the average input current. There are different topologies that can be created based on (10). λ can be used to regulate, such as enforcing long excitatory neuron connections with short inhibitory neuron connections or vice-versa. The scalar C is used for fine-tuning the degree of sparsity in the reservoir. When choosing the parameters to control the formation of recurrent connections, typically one should choose a set of parameters that will generate a sparse reservoir. From a practical standpoint, sparse over-complete representations help facilitate learning and improve robustness to noise in the data [11], and sparsity also tends to result in better performance in terms of accuracy and cost of implementation [12]. In [10] and [11], synaptic degrees of connectivity less than 10% were found to be optimal.

Synaptic Strength and Plasticity: The strength of the synaptic connections, or sum of synaptic connections, is related to the choice of voltage threshold for the neurons and should be chosen such that the rate of excitability in the hidden layer (firing rate of LIF neurons) is on a desirable time-scale. The choice of STP parameters will also help to regulate the firing rate of neurons by weakening neurons with high firing rates. Using static synapses without STP has also been shown to have negligible impact on performance on small tasks [13].

Hidden Layer Size: Typically, the size of the hidden layer scales with the complexity of the problem, accounting for factors such as the separability of the data, the dimensionality of the input data, and the length of temporal information the network needs to remember. As stated in [10], an optimal increase in hidden layer dimensionality is ~ 10-50x the size of the input space, and resulting diminishing returns outside these bounds. Another guideline [12] is that the hidden layer should be large enough to capture the number of independent signals the hidden layer has to remember. The authors estimate this by DH = DI · t, where the dimensionality of the hidden layer (DH) is equal to the dimensionality of the input space (DI) times the number of time-steps it has to remember. The number of time-steps a network needs to remember is the number of 8 time-steps a prediction is dependent on (i.e. the length of a sequence). However, as the size of the hidden layer grows the LSM can become computationally intensive owing to the memory utilized by the recurrent synaptic connections and is also prone to overfitting.

The last component to the LSM is the readout layer which performs classification or prediction. When sending the activity of the hidden layer to a non-spiking, memoryless readout layer, several states collapse upon each other when using only the instantaneous firing. This impacts the networks ability to distinguish different patterns (*e.g.* 1,0,1 and 1,1,1 look the same if only looking at the last spike). Therefore, it is common to use some form of encoding or filtering in this case. For our research we use a synaptic trace operation computed by (13) to record the average activity of the hidden layer neurons before passing their activity to the readout later.

$$X_{trace} = X_{trace} + x - \frac{X_{trace}}{\tau_{trace}} \tag{13}$$

A summary of parameters to consider when designing an LSM along with their impact on the network and general insights into choosing appropriate values is listed in Table 1.

Table 1. Summary of Design Parameters for the LSM and Impact on Network Dynamics

Parameter	Description	Comments
V_{th}	Controls the rate of response of LIF neurons	Should have appropriate firing rate based on time-scale of input determined by (10)
τ_m	Controls the length of time over which a neuron integrates information and magnitude of changes in membrane potential	Should have appropriate firing rate based on time-scale of input (10). Manipulate R and C to adjust integration of new inputs and leakage rate separately (1)
Input Sparsity	Results in mixed-selectivity in hidden layer representations and forms and over-complete representation of input data.	Neurons should have small number of connections (*e.g.* less than 10%) based on studies in [7], [13], [15] to form sufficient representations and reduce computational complexity.
D^H	Larger hidden layers will capture more sets of information from the input and have longer temporal memory for complex problems.	A default is $D^H = 10 \cdot D^I$ [7] or $D^H = t \cdot D^I$ [13].
λ	Controls the likelihood of neurons forming recurrent connections with distant neurons.	Use different λs to specify reach of excitatory and inhibitory neurons.
C	Controls the sparsity of recurrent connections.	Small degrees of sparsity (*e.g* less than 10%) are optimal for performance [15] and computational efficiency [13].
Synaptic Strength	Determines the impact of a neuron firing on its neighbours membrane potential.	Magnitude should be similar to V_{th} to produce input currents that result in desired firing rates based on (10).
U	Resting value for fraction of synaptic efficacy used by an action potential.	Higher U will allow for an action potential to utilize more of the available synaptic efficacy.
τ_{facil}	The rate at which u decays to its resting value U.	Amount of synaptic efficacy an action potential can use, smaller τ_{facil} will have faster recovery (8).
τ_{rec}	The rate at which the available synaptic efficacy recovers from depression.	Total available synaptic efficacy, smaller τ_{rec} will have faster recovery (7).

Few metrics have been proposed as tools for assessing the quality of a reservoir as listed in Table 2. These metrics are designed to assess a reservoirs separation property [2], which is how well the reservoir produces separable representations for unique inputs and similar representations for similar inputs. The advantage of these metrics is that they can give some notion of how linearly separable a reservoir is which has been shown to be correlated with performance. Additionally, they do not require the training of the readout layer to be completed, and the kernel quality and Lyapunov exponent can be computed without class labels. This makes these metrics very easy tools for assessing a variety of reservoir parameters before selecting a model for training. However, a specific metric does not directly imply a good performance.

Table 2. Metrics for Assessing the Quality of a Reservoir

Metric	Equation	Description	Comments
Separation Property	$Sep_x(t) = \frac{C_d(t)}{C_v(t)+1}$	Ratio of inter-class variance to intra-class variance	Quantitative measure of separation and generalization
Kernel Quality	$KQ = rank(N)$	Rank of RC's representation of N unique inputs	Fast calculation to assess separation
Lyapunov Exponent	$\lambda = k \sum_{n=1}^{N} ln\left(\frac{\|\|x_j - x_j\|\|}{\|\|u_j - u_j\|\|}\right)$	Ratio of RC's representation of two similar inputs (x) to distance between inputs (u)	Measures how chaotic the reservoir is when inputs are very similar

In order to extend the LSM's computational capability to handle more complex tasks and increase the networks memory capacity we developed a model called the deep-LSM [14,15,16] as shown in Figure 6. There is a bottom-up flow of information between multiple hidden layers in the deep-LSM. The high-dimensional representation of information in the hidden layers is projected into a low-dimensional encoding through the introduction of a spiking winner-take-all layer (WTA). Similar to a vanilla LSM, the role of the hidden layers is to create high-dimensional representations of dynamic information in the input space. As a starting point for implementing the deep-LSM, one technique is to consider the size of the hidden layer needed for a vanilla LSM and divide that by the number of layers in the deep-LSM to get the layer size. In between the hidden layers are the WTA layers which extract temporal features and form a sparse, low-dimensional encoding of the hidden layer's dynamics. The WTA layer is implemented as a layer of feed-forward spiking LIF neurons trained with STDP (including synaptic scaling and intrinsic plasticity). To create the WTA nature, global inhibition is modelled by allowing an excitatory neuron to send a strong inhibitory signal to every other neuron if it emits a spike. This forces neurons to compete with one another to learn different patterns.

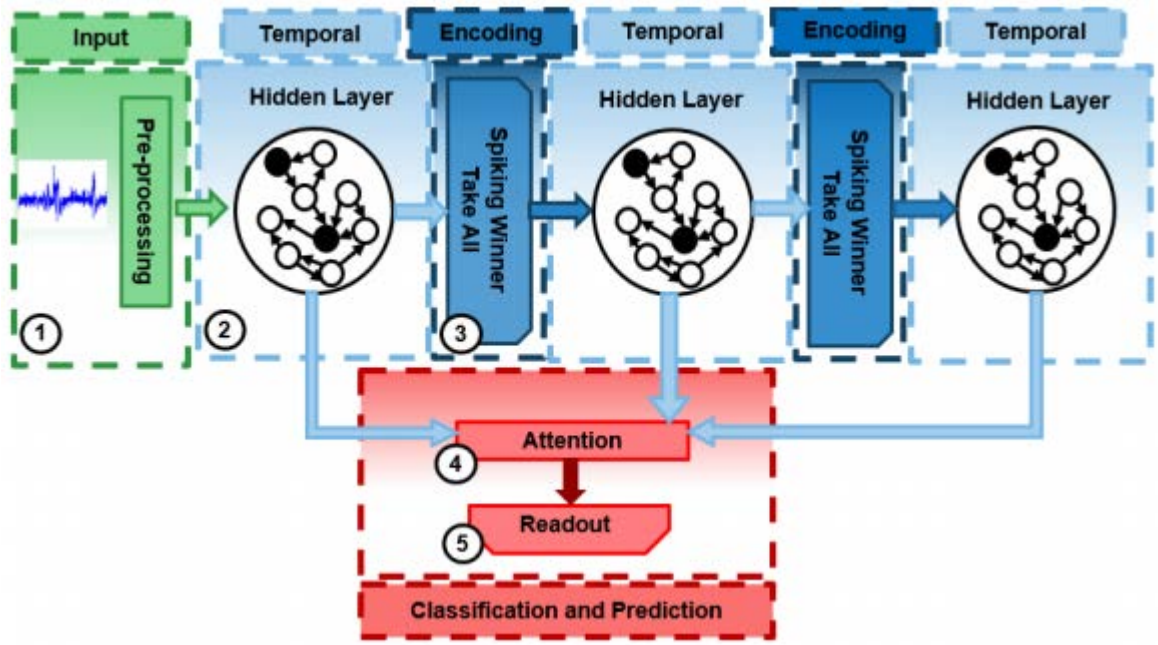

Figure 6. deep-LSM architecture. 1) Input Stage Consists of Pre-processing or Transfer Learning to Extract Features. 2) Hidden layers Consist of Random Reservoirs to Produce High-dimensional Dynamic Networks Which Capture Temporal Information. 3) WTA Layers are used to Extract Temporal Features and Condense the Hidden Layer Representation for Propagation through the deep-LSM. 4) Attention is used to Selectively Process Information in the deep-LSM with Limited Resources. 5) A Readout Layer is Trained to Perform Classification or Prediction

When implementing the WTA layers, a typical size is a tenth of the hidden layer size. Making the WTA layer larger extracts more features but starts to degrade performance because as the size of the WTA layer approaches the size of the hidden layer, the hidden layer loses its ability to produce a good high-dimensional representation of the WTA outputs. For synaptic scaling, the sum of synaptic connections is kept to one while for intrinsic plasticity the increase in firing threshold is dependent on the rate of learning desired and number of neurons. The increase should be large enough that during a short time window, the neuron is unlikely to fire again giving other neurons a chance to learn different features. Lastly, for the STDP learning rate, the magnitude of the weight change should be a few orders smaller than the magnitude of the weights to ensure stable but meaningful learning.

To process the information in the deep-LSM, a readout layer with attention is used to efficiently perform classification with limited resources. In the deep-LSM there are two attention mechanisms, deep attention which identifies important layers and spatial attention which identifies important neuron groups. Attention works by applying an external multilayer perceptron (MLP) to predict attention coefficients or a weighting for different parts of the deep-LSM. The attention networks receive the state of the hidden layers in the deep-LSM as input. In the case of the deep attention network, the output is a set of coefficients for each hidden layer assigning its importance. Whereas for the spatial attention, the output is a set of coefficients for each neuron in the hidden layer (or groups of neurons). To move the design towards a hardware friendly algorithm with a constant output size, the goal is to predict the attention coefficients from only the first layer's dynamics (like a single LSM) which showed negligible impact on performance in some cases as will be discussed later.

Once the attention coefficients are produced, the network first applies a weighted summation of all the hidden layers based on the deep attention coefficients and then applies the spatial attention to each neuron value as described by equations (14)-(16). This specific implementation utilizes soft attention which forms a summation of information in the deep-LSM, however future work can explore other models such as hard attention where we drop un-important portions of the deep-LSM or utilize concatenation rather than addition. Furthermore, the readout layer and attention networks are trained using backpropagation (only two layers), however different local learning rules can be tested as a replacement such as STDP, or random backpropagation and similar approximations.

$$A^{l}_{deep} = \sigma\left(W_{deep} * X_{deep-LSM}\right) \tag{14}$$

$$A^{n}_{spatial} = \sigma\left(W_{spatial} * X_{deep-LSM}\right) \tag{15}$$

$$X_F = \left(\sum_{l=1}^{L} A^{l}_{deep} * X_l\right) \odot A_{spatial} \tag{16}$$

3.2 Digital Architectures

3.2.1 Digital Extreme Learning Machine (ELM). A baseline digital architecture is shown in Figure 7. The dataflow and activation of all logical blocks are governed by a global controller. The network architecture has three layers, the input layer, the hidden layer, and the output layer. In this design the input layer has no dedicated neurons and only acts as a buffer to project the inputs to the hidden layer. In the hidden layer, all the weights are randomly generated using a 14-bit pseudo random number generator. The neuron weights are stored into memory dedicated to each neuron and the size is adjustable depending on the number of connections to the neuron. The weights are multiplied with the input features that are mapped to the hidden layer in a serial order to reduce the amount of needed hardware. The result is then stored in a 28-bit accumulator which relays the result to a sigmoidal activation function. The sigmoidal activation is implemented using a piecewise linear approximation. The output of the hidden layer is simultaneously generated and stored into a parallel-in-serial-out shift register. Same as the hidden layer, the neurons in the output layer receive the inputs serially which are multiplied by their weight and stored in a 28-bit accumulator and then passed into the activation function.

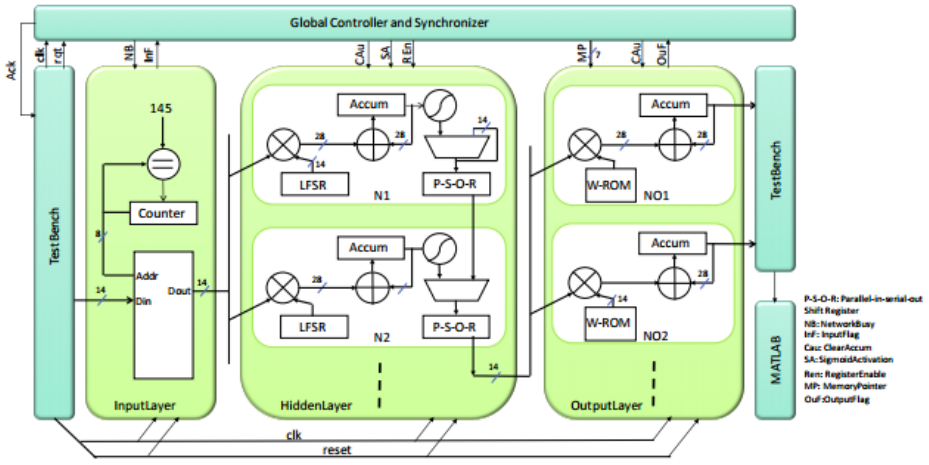

Figure 7. Digital ELM Architecture

For resource efficiency, we explored the trade-off between sharing computational resources by folding the network. A high-level representation of the baseline topology is shown in Figure 8a. The neurons of each layer are physically modelled in hardware, without any resource sharing. The classification rate (CR) can be computed as in (17). The CR is defined by three time elements including feature transfer time (T_{fetch}), feature extraction time ($T_{FeaturesExtract.}$), and classification time ($T_{Class.}$), defined in Table 3.

$$CR = \frac{1}{T\_fetch + T\_FeatureExtract. + T\_Class.} \tag{17}$$

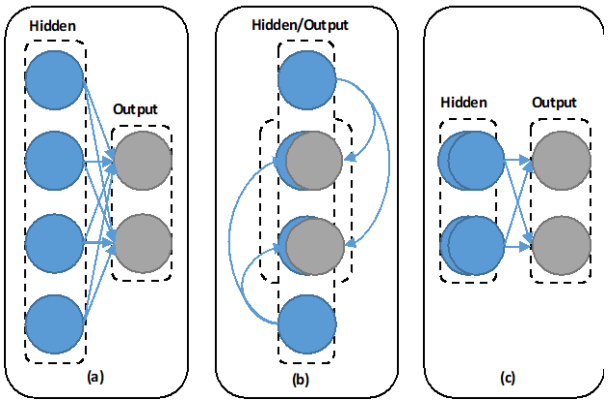

Figure 8. Proposed Topologies: (a) Baseline, (b) Coalescing, and (c) Folding

Two topologies are proposed for resource sharing in the digital ELM [17].

a) Coalescing: In this topology, the hidden and output layers are meshed together to share the neuron resources, as demonstrated in Figure 8b. For example, by coalescing M neurons in the output layer with N neurons in the hidden layer, the total number of neurons will reduce to N instead of N+M. This topology requires identical nonlinear functions for neurons in both the layers and similar physical models. The classification rate for this topology, according to (17) and Table 3, is slightly less compared to the baseline due to buffering and additional synchronization cycles.

Table 3. The Time Elements of CR is Defined in this Table. The final Expression of the Time Equation Should be Multiplied by the clk_{period}

Topology	T_{fetch}	$T_{featureExtract.}$	$T_{Class.}$
Baseline	Features, n[a]	Features, n+1	HiddenNeurons, NH[b]
Coalescing	Features, n	Features, n +c[c]	HiddenNeurons, NH
Folding	Features, n	k x n	Hidden Neurons, NH

[a]n = Number of input features
[b]NH = Number of hidden neurons
[c]c = Constant varying between 1-3 for layers time synchronization

b) Folding: In this topology, Figure 8c, the layer with large number of neurons is folded k times by time-sharing resources. Folding a layer leads to extra latency (see (17) and Table 3) in the network and throughput degradation. Thus, the value of k is application dependent.

c) Low Precision Folding: This topology is an extension of the folding topology combined with low precision. In all previous topologies a 14-bit resolution is utilized. In case of LP folding, 14-bit resolution is used in hidden layer, and 9-bit for output layer. 9-bit is chosen via observing the impact of resolution changing in the output layer on the overall network performance. As shown in Figure 9, 16-bit and 9-bit almost results the in same performance.

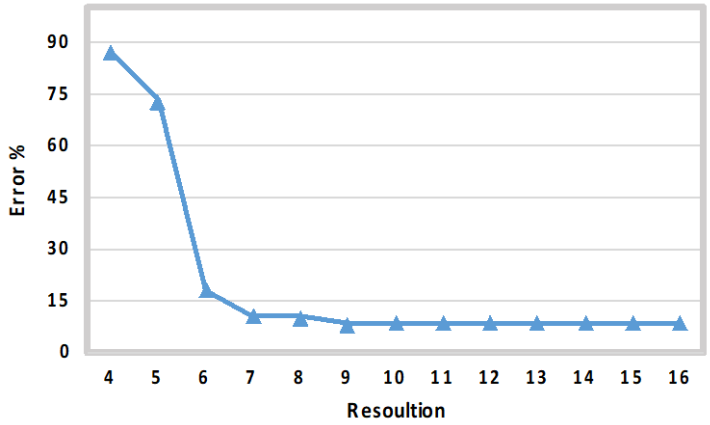

Figure 9. The Miss Classification Rate in MNIST Dataset When Changing the Data Resolution (Data Representation in Number of Bits) in the Output Layer of ELM

3.2.2. Digital Echo State Network. A digital implementation of the ESN was done in [18] for Floating Point Grid Array (FPGA) platforms. In order to improve the efficiency and scalability of a hardware based implementation, we implement a hybrid topology of a ring and a shared centered node shown in Figure 10a. This approach reduces the complexity of the interconnections among neurons, as well as reduced the delay of propagation of information between any two neurons to two time steps. On a reconfigurable platform, this architecture can be easily implemented with a doubly-twisted toroidal architecture as shown in Figure 10b. Through analysis over several benchmark applications we found using a tanh activation function to perform the nonlinear mapping in the reservoir as well as classification in the output layer to have the greatest performance. A Register-Transfer Level (RTL) description of the tanh function implemented in this work is shown in Figure 11. Each neuron has three inputs, the previous neuron and the center neuron from the hybrid topology as well as the external input to the network. These three inputs are multiplied by their respective weights which are stored in local registers. The final sum is then passed to the tanh activation function which is implemented by a piecewise linear function.

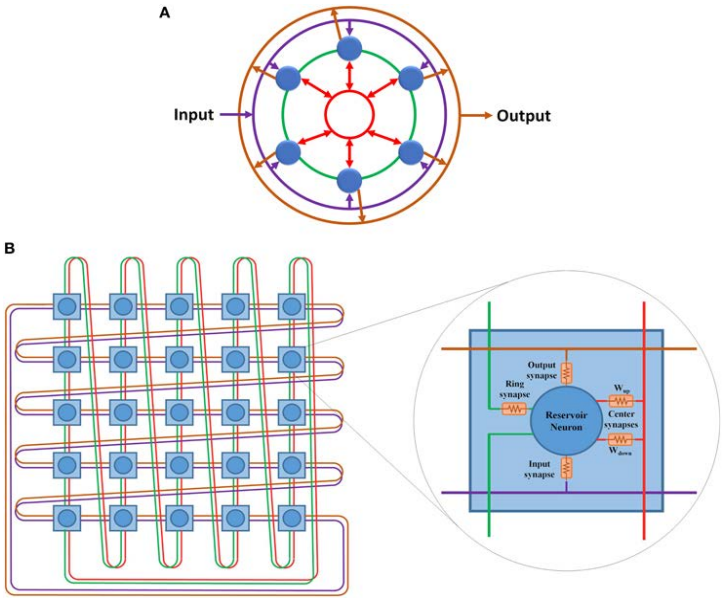

Figure 10. Digital ESN Architecture with hybrid topology (A) and interconnect and neuron architecture (B)

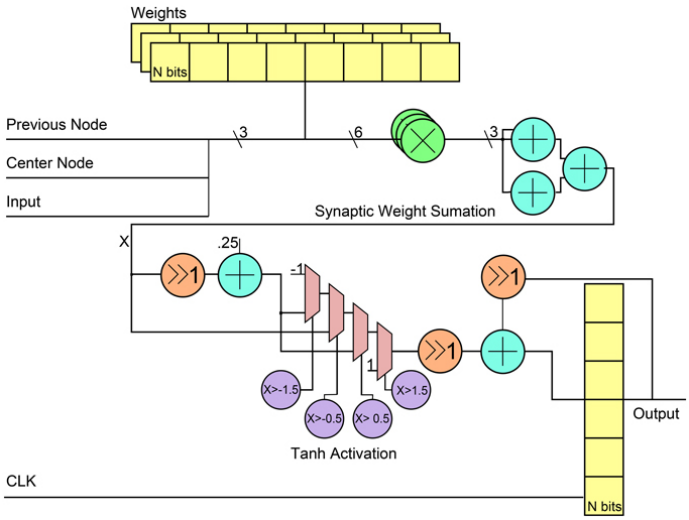

Figure 11. RTL Diagram of Digital tanh Activation Function for FPGA Platforms Using a Piecewise Linear Approximation

3.2.3. Digital Liquid State Machine. In [13] we developed a fully parallel reconfigurable digital architecture for implementing an LSM on FPGA platforms capable of processing information in real-time. The high-level architecture of the proposed design is shown in Figure 12.

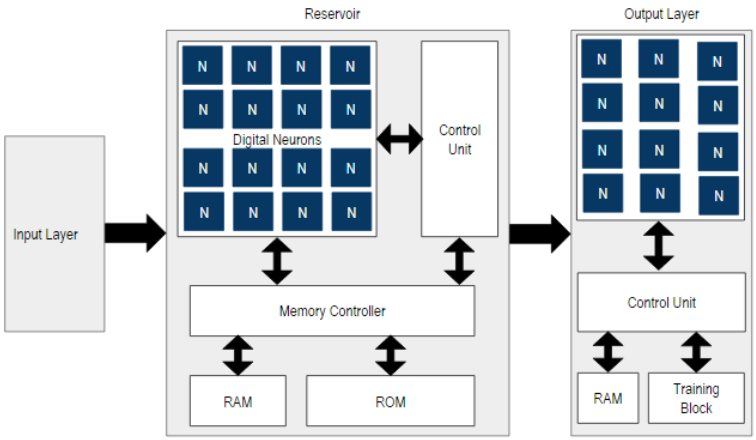

Figure 12. Digital LSM Architecture for FPGA Platforms

A new design approach taken in this research was the removal of an input layer, where instead the inputs are treated as synaptic current and fed directly into the liquid layer. This does not exclude the use of a pre-processing layer, however because pre-processing is application dependent it is not considered when discussing the general architecture. This reduces the design to two major components the reservoir (Liquid) layer and the output layer. The reservoir is comprised of a collection of Leaky Integrate and Fire neurons whose communication is managed by a digital controller. The LIF Neurons digital implementation is shown in Figure 13 and has four main sub-components; the spike generator, the spike counter, the weighted current block, and the integrator block.

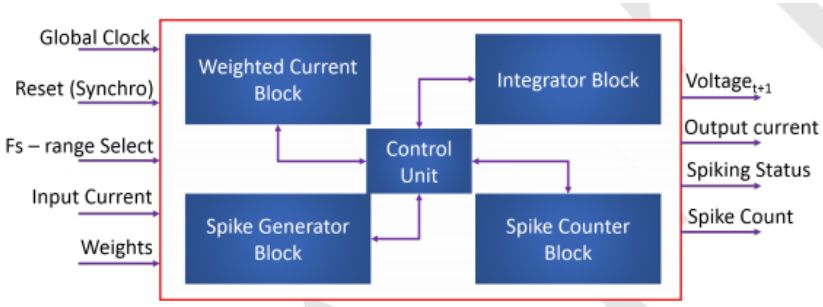

Figure 13. Block Diagram for Digital Leaky Integrate and Fire Neuron

The weighted current block in Figure 14 is responsible for the summation of the synaptic current flowing into an individual LIF neuron. The inputs to the neuron are stored in the input accumulator. The weights within the liquid are all powers of 2 therefore multiplication is performed as bit shift operations and the resultant current values are summed to calculate the total current. The total current is then passed to the integrator block in Figure 15 which keeps track of the membrane potential of the LIF Neuron at each time step. Once the integrator block reaches a threshold the spike generator block (Figure 16) is triggered to release a spike which is stored in memory. When the spike generator block is triggered it remains active for the duration of the spike and during this time the neuron is in its refractory period and cannot receive inputs. The last block, the spike counter, is responsible for counting the number of times an individual LIF neuron has emitted a spike in a specific time-window. This value is passed onto the output layer for classification.

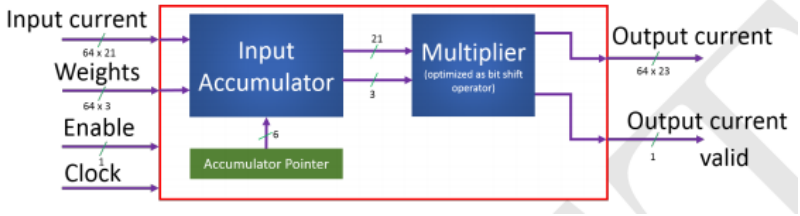

Figure 14. Block Diagram for Weighted Current Block Inside Digital Leaky Integrate and Fire Neuron

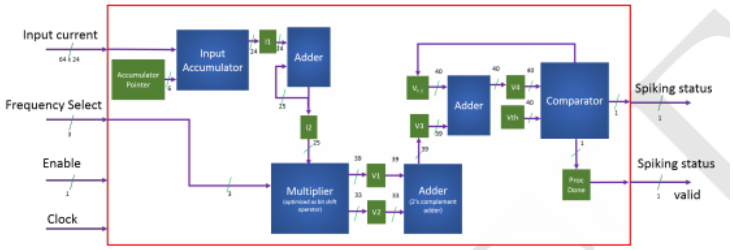

Figure 15. Block Diagram for Integrator Block Inside Digital Leaky Integrate and Fire Neuron

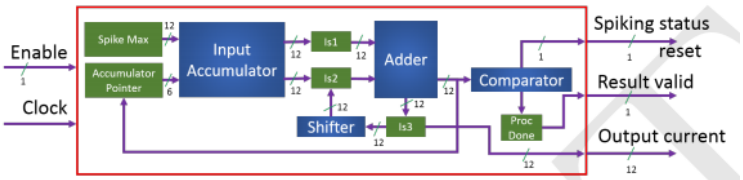

Figure 16. Block Diagram for Spike Generator Block Inside Digital Leaky Integrate and Fire Neuron

The output layer implemented is an ELM. The block diagram for the output layer is shown in Figure 17. where all the neurons are implemented by a piecewise linear approximation of the sigmoid function. In order to tune the second layer of weights, a block dedicated to performing stochastic gradient descent is incorporated into the output layer. The design is implemented using 32-bit single precision, further details about the digital architecture can be found in [13].

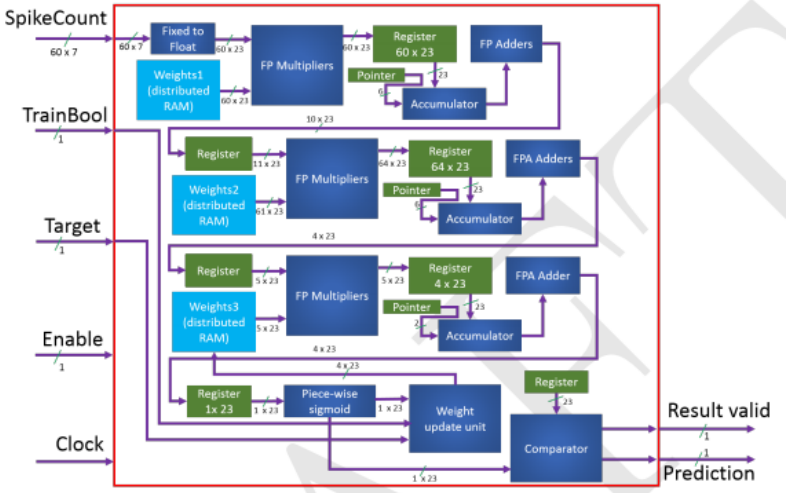

Figure 17. Block Diagram of Two-L MLP Implemented for the Output Layer of the Digital LSM

The design was extended to implement a deep-LSM. A digital architecture for the proposed deep-LSM was designed for FPGA platforms. The high-level architecture is shown in Figure 18. The configuration for communication between blocks is an enable→ done→ update handshaking mechanism. This is a synchronized design where the processing and communication between blocks is controlled by a high-level state machine.

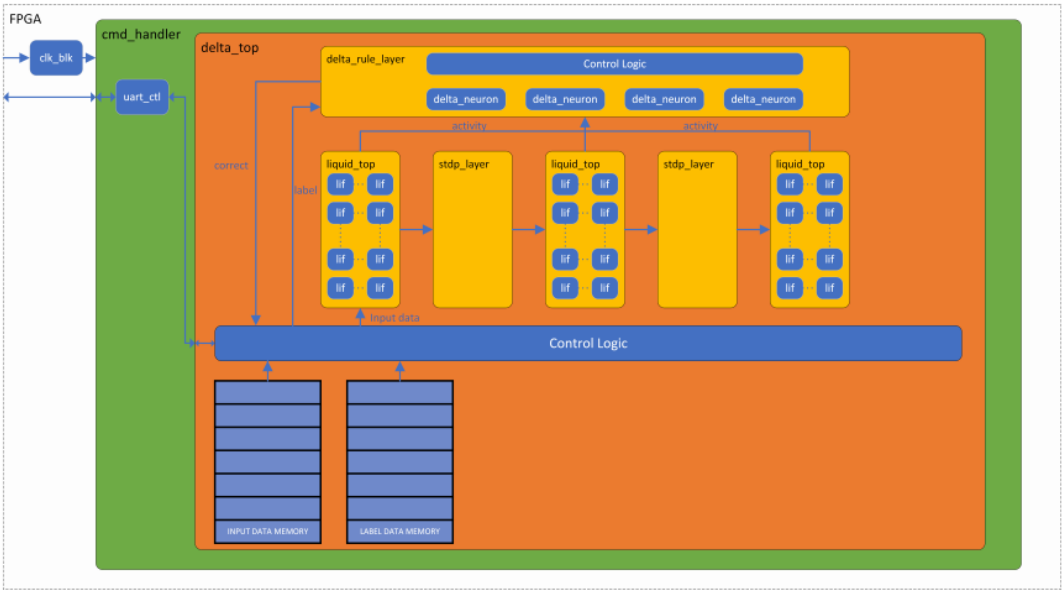

Figure 18. High-level FPGA Architecture for Digital Deep-LSM

The hidden layer consists of a block of LIF neurons using 32-bit precision. The inputs to the layer are the previous layer inputs and the control signals while the output is the spiking and rate-encoded activity of the neurons. The spike trains are passed to a Spike-Timing-Dependent Plasticity (STDP) layer which performs unsupervised learning using a trace-based STDP rule. Lastly, the hidden layer activities are passed to the delta rule layer which is where supervised learning occurs. The data flow for the proposed model is shown in algorithm 1.

while $training\_cnt < NUM\_TRAIN$ **do**
  while $timestep\_cnt < NUM\_TIMESTEP$ **do**
    Read input data from memory;
    Pass input data through layers;
  end
  Read label from memory;
  Train delta rule layer with label and activity values from liquid layers;
end
while $testing\_cnt < NUM\_TEST$ **do**
  while $timestep\_cnt < NUM\_TIMESTEP$ **do**
    Read input data from memory;
    Pass input data through layers;
  end
  Read label from memory;
  Test delta rule layer with label and activity values from liquid layers;
end
Report NUM_TRAINING_CORRECT and NUM_TESTING_CORRECT to host;

Algorithm 1. Data flow through digital deep-LSM

3.3 Stochastic Architectures

3.3.1 Stochastic Extreme Learning Machine. In stochastic computing values are represented by probabilities of ones in a bit-stream. Stochastic based designs offer very robust algorithms and allow for complex operations such as addition, multiplication, and the activation functions of many neural networks to be implemented with very little hardware. The downfall to this approach is the increased latency due to long bit-streams to achieve high precision. An early stochastic ELM architecture [19] is shown in Figure 19. In this design a stochastic number generator is used to generate all the stochastic bit streams for the inputs, the weights, and the target labels used for training. The threshold activation function is computed by counting the ones passed into the neuron, as long as the counted number of ones from the previous layer exceeds a threshold value, a one is sent to the next layer. This type of threshold function on stochastic bit-streams achieves a sigmoid like activation function as shown in Figure 20. The entire system is under the control of a single controller which regulates the network to pass one stochastic bit per clock cycle. A counter is used to keep track of the stochastic bit stream to raise a flag on a stable output or to enable training. This network is trained in an online approach using the stochastic least-means squared algorithm which is implemented using the circuit shown in Figure 21.

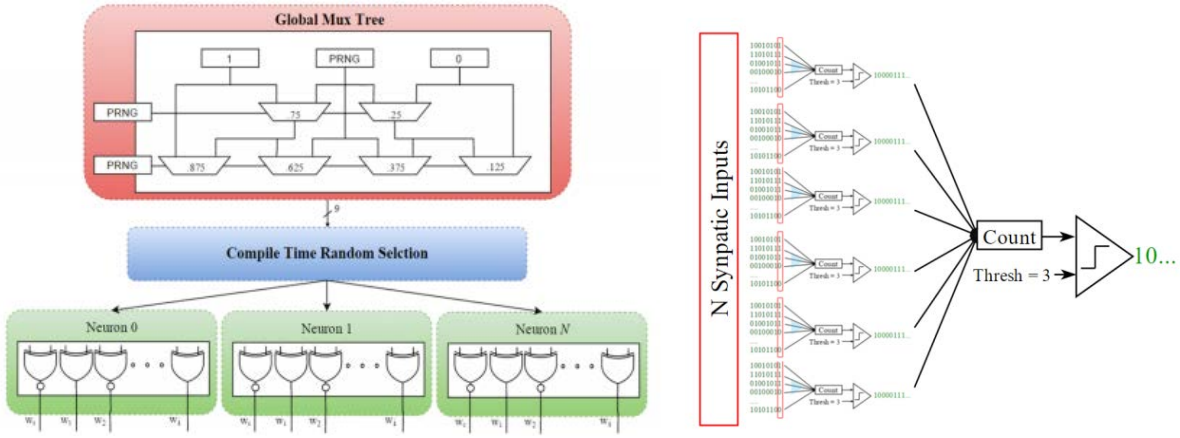

Figure 19. (Left) Hardware Implementation of the Hidden Layer in the ELM, (Right) Threshold Activation Function Connected to a Single Layer

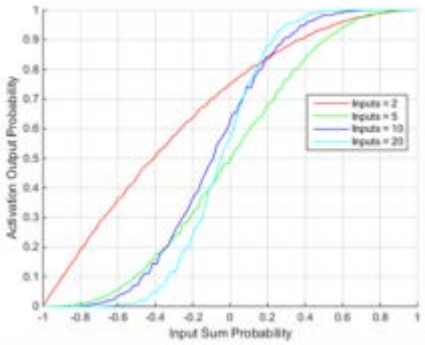

Figure 20. Threshold Function Transfer Function Achieves Sigmoid like Behavior for Long Bit-streams

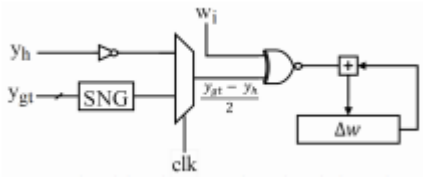

Figure 21. Stochastic Gradient Descent Circuit for Online Learning in the Stochastic ELM

The stochastic ELM involves several functional components which generate stochastic bits, perform arithmetic operations, non-linear functions, and conversion from stochastic to decimal. Due to the length of the stochastic bit streams, simulation time of stochastic architectures can be greatly accelerated with Graphics Processing Unit (GPU) based implementations of the different functional components [20]. The first component, the stochastic number generator block is done by comparing a binary number with a random number as shown in Figure 22.

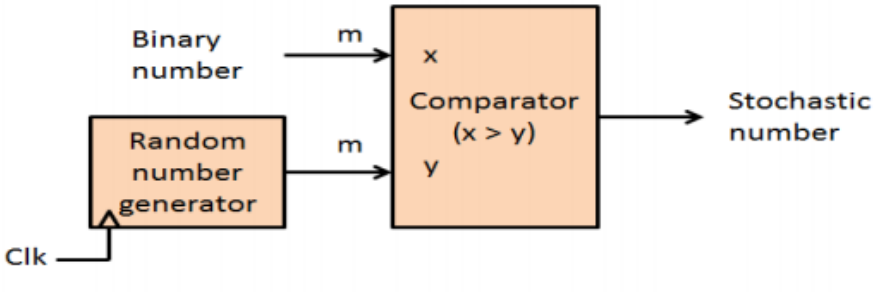

Figure 22. Stochastic Number Generator

This can be implemented on a GPU by comparing a vector of random numbers whose dimensionality is equivalent to the stochastic bit length with the decimal number being converted. The process can be performed in parallel to produce the stochastic representation in a single cycle. The random numbers in this process are drawn from a uniform random distribution bounded between 0 and 1.

The second function implemented on the GPU was the sigmoid function which performs a non-linear transformation on the information propagating through the ELM. The GPU implementation of the sigmoid function is done through two steps of parallel processes. The first step is to count the total number of ones in the input stochastic bit stream for each of the *m* input streams. If the number of ones is greater than half the number of inputs, then the output is a one. This operation is done in parallel for all *n* elements in the stochastic bit streams producing an n-bit stochastic bit stream as shown in Figure 23.

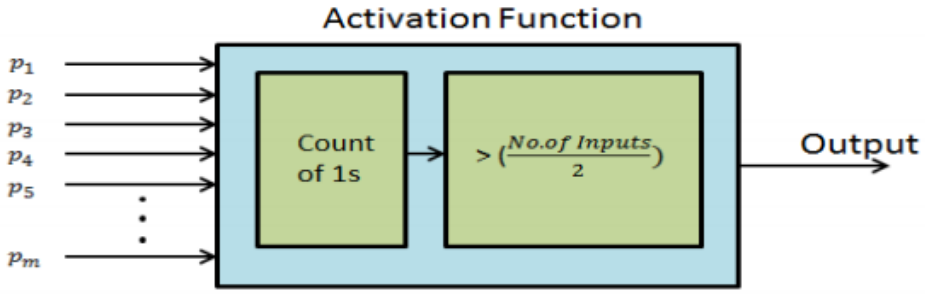

Figure 23. Stochastic Sigmoid Activation Function

There are two arithmetic operations used in the implementation of the ELM; multiplication and subtraction. The multiplication of inputs in stochastic logic can be implemented by a two-input XNOR gate. In the GPU this is done by performing the XNOR function for all *n* elements between two stochastic bit streams in parallel. Subtraction is performed by using a 2-1 MUX with an inverter as shown in Figure 24. The subtraction operation can be explained by

$$Z = (x.S + y'.S') \tag{18}$$

where Z is the subtraction between x and y and S is the select line. The function can be computed through an inversion, followed by the appropriate AND operation, and finally by an OR operation. These through steps can be computed in parallel for all *n* elements in the x and y stochastic bit streams.

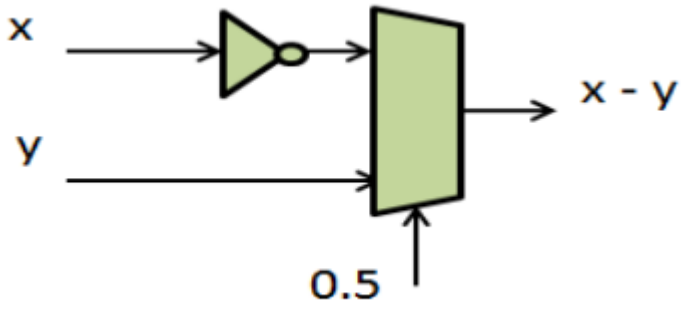

Figure 24. Stochastic Number Subtraction

The last component of the GPU implementation is the stochastic to decimal conversion process. For a bipolar stochastic representation, this can be done by counting the number of ones in a stochastic bit stream, dividing the result by the stochastic bit length, adding one, and the dividing the final result by two. This process in the GPU is shown in Figure 25. Where all *m* stochastic bit streams are processed in parallel in four steps which results in a decrease in conversion time from stochastic representations to a decimal representation.

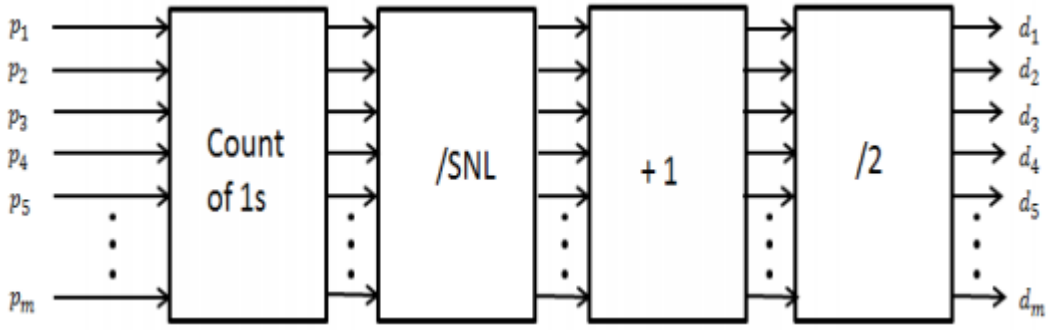

Figure 25. Stochastic to Decimal Conversion Process

Using these building blocks, it is possible to build a stochastic ELM as shown in Figure 26.

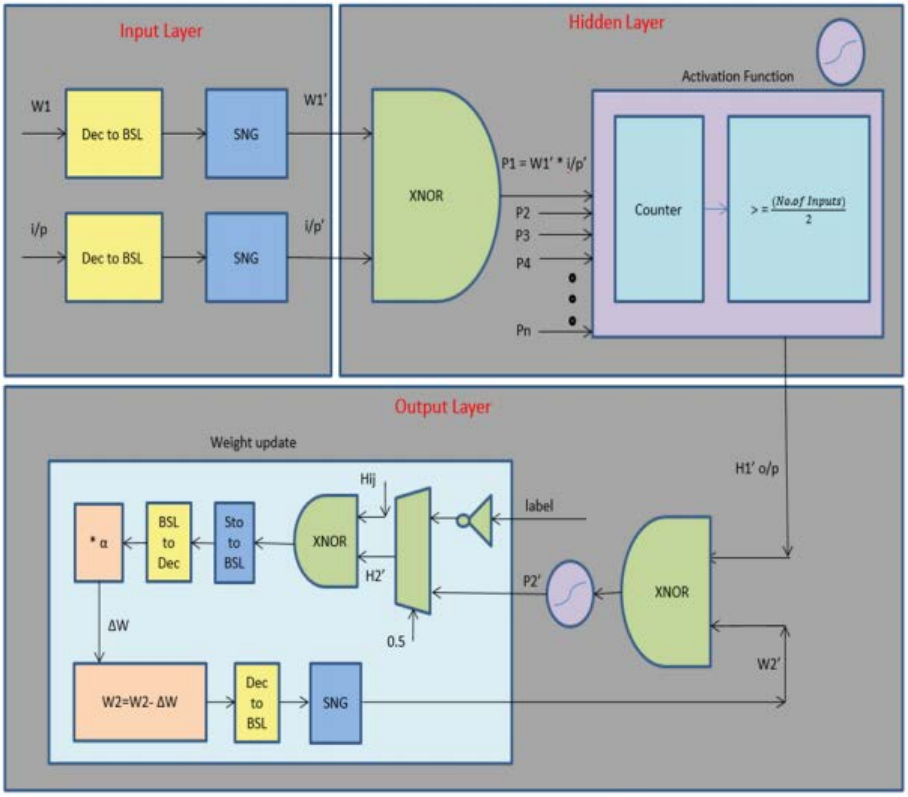

Figure 26. Full Stochastic ELM Architecture

The input layer consists of a set of input nodes equivalent to the number of features in the specific application and the first set of weights which performs the random projection in an ELM. These values are converted to stochastic bit streams which can be visualized in Figure 27 for the inputs and Figure 28 for the weights.

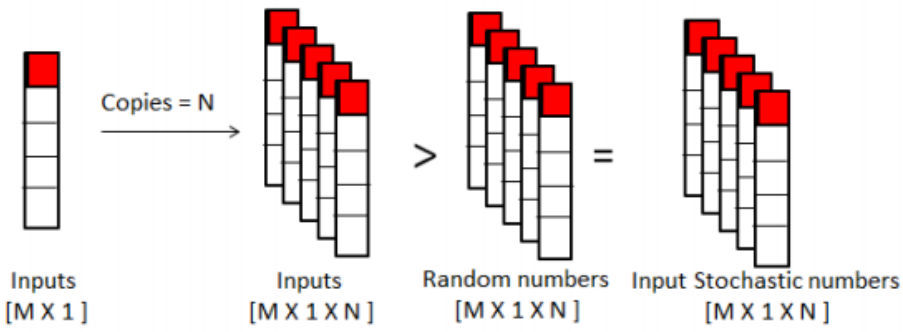

Figure 27. Conversion of Input Signals to Stochastic Bit Streams

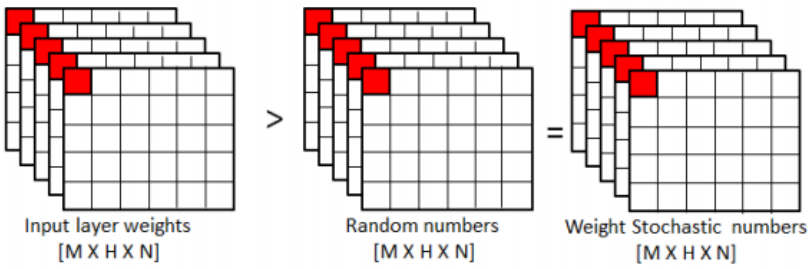

Figure 28. Conversion of Weights into Stochastic Bit Streams

The hidden layer computes the random projection and non-linear transformation of the input data by approximating the sigmoid with an XNOR operation and thresholding operation. This can be visualized in Figure 30 which shows the multiplication and Figure 31 which shows the sigmoid activation function.

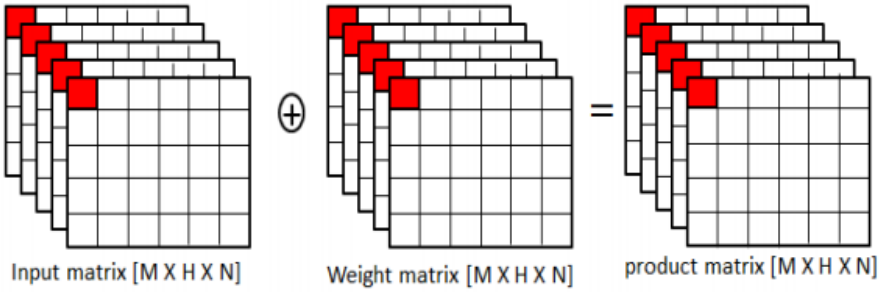

Figure 29. Neural Network Matrix operation Between Input Signal and Weights

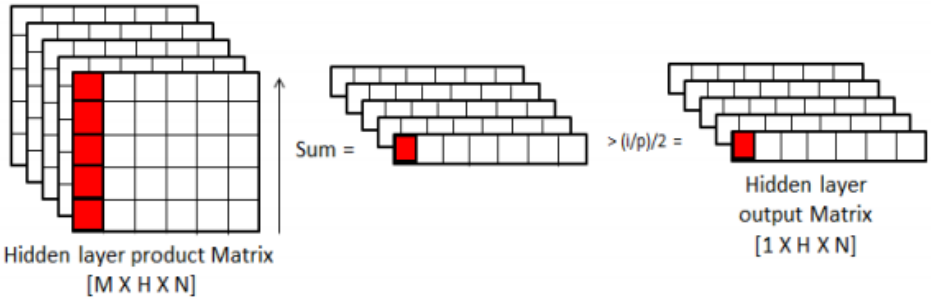

Figure 30. Summation and Non-linear Sigmoidal Function of Stochastic Inputs

Finally, the output layer performs the multiplication between a second set of randomly initialized weights and the output of the hidden layer. The second set of weights is then optimized through a stochastic training circuit. The update to the second weight matrix is computed by

$$\Delta W = \alpha(y'(k) - y(k))h(m) \tag{19}$$

where the output of the ELM y(k) is subtracted from the target value y'(k) and then multiplied by the output of the hidden layer. After this step, because α needs high-precision, the value $(y'(k) - y(k))h(m)$ is converted to a decimal and then multiplied by the scalar value α. Then the weights are updated and converted back into stochastic bit streams to be used in the ELM's operation.

3.3.2 Stochastic Echo State Network. Using the same building blocks of the stochastic ELM, a stochastic implementation of the ESN was also designed as shown in Figure 31. The main difference between the two architectures is that unlike the hidden layer of the ELM, the reservoir layer consists of a set of feedback connections.

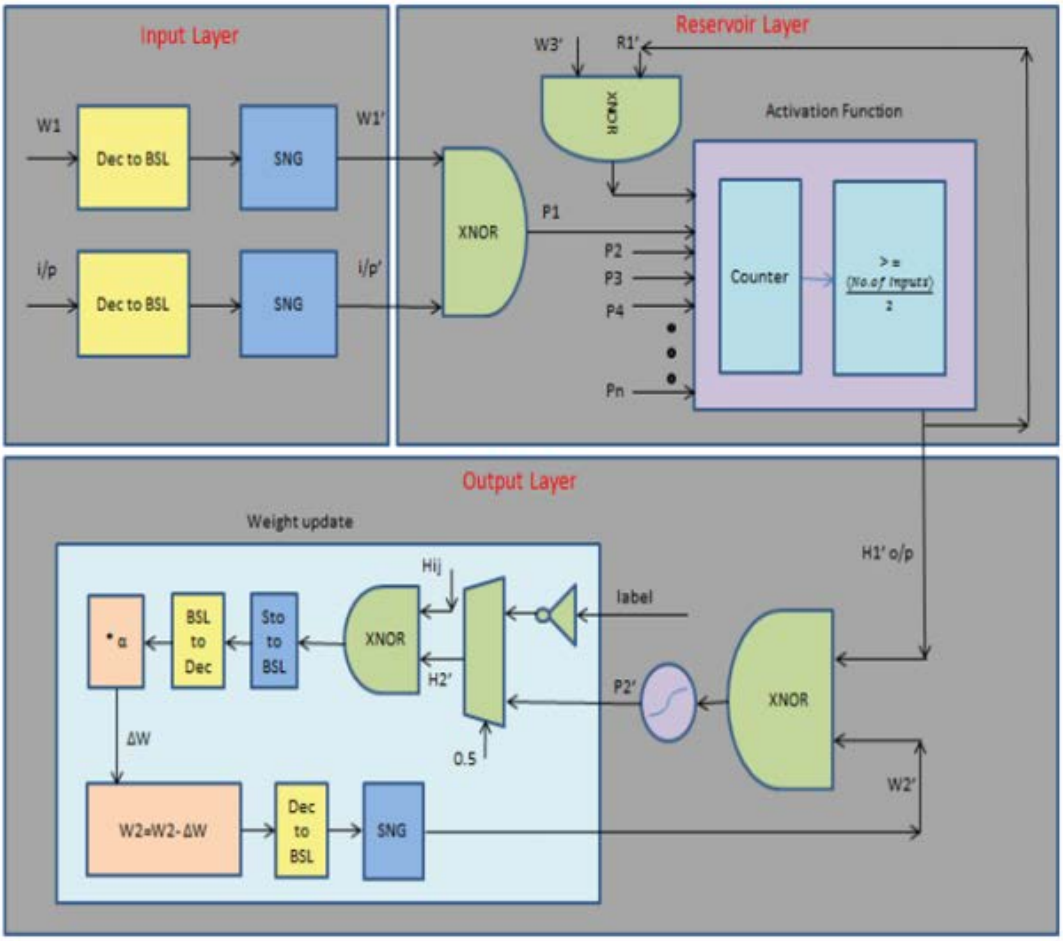

Figure 31. Architecture of a Stochastic ESN

3.4 Neuromemristive Architectures

3.4.1 Neuromemristive Primitives. RC systems consist of two layers of static weights. One method of implementing the networks synapses without the need of programming each individual synapse is to take advantage of device mismatch in the fabrication process to generate a range of fixed random strength synapses. A single synapse is shown in Figure 32 where the mismatch between the transistor pairs is the cause of the random synaptic strength. The fixed synapses could then feed directly into an analog neuron such as the *tanh* activation function implemented using a differential amplifier biased in weak inversion as shown in Figure 33. The outputs of these neurons can then be converted to digital signals and sent to a digital output layer, or we can use a memristor crossbar to implement the last layer of weights in the analog domain which for an ESN would then feed into a second layer of the *tanh* neurons.

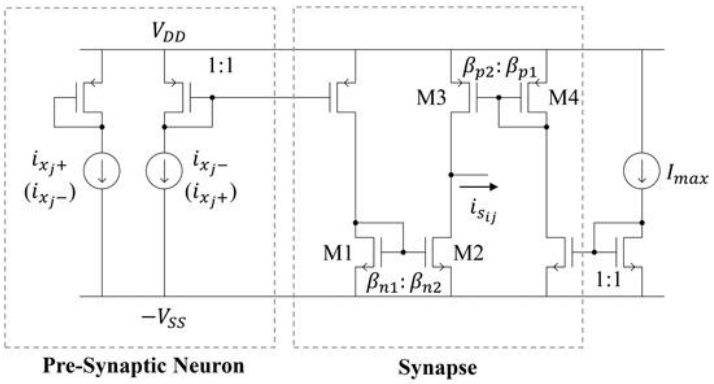

Figure 32. Fixed Synapse Design Using Transistor Mismatch for Random Weighting

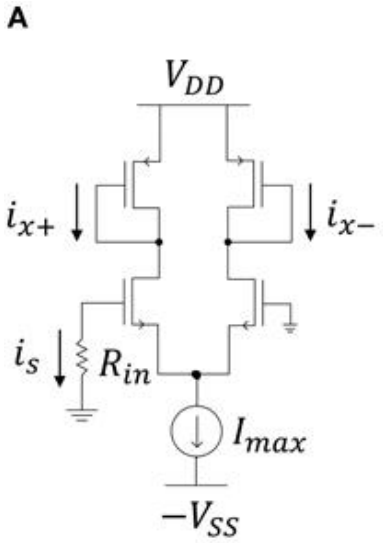

Figure 33. tanh Activation Function Using Differential Amplifier in Weak Inversion

Figure 34 shows how a crossbar translates to the synaptic connections between two layers of a neural network. In this year we have explored three different crossbar implementations. The first design in Figure 34 (right) is a current mode based design. With this approach the weight between a memristor of the i^{th} output neuron and j^{th} input neuron is a function of the ratio of all the passive memristance devices along a single column and can be computed by (20). This requires advanced methods to directly map desired weight matrices onto the crossbar, it also constrains the weight space that can be realized.

$$W_{i,j} = \frac{1}{G_i^{acc}} \frac{G_j^{ex} - G_{i,j}^{inh}}{G_j^{total}} \tag{20}$$

where G represents the memristors conductance while i and j denote the location of the memristor in the crossbar array by row and column respectively. Acc represents the conductance of the accessory memristor, ex represents an excitatory memristor, and inh represents an inhibitory memristor.

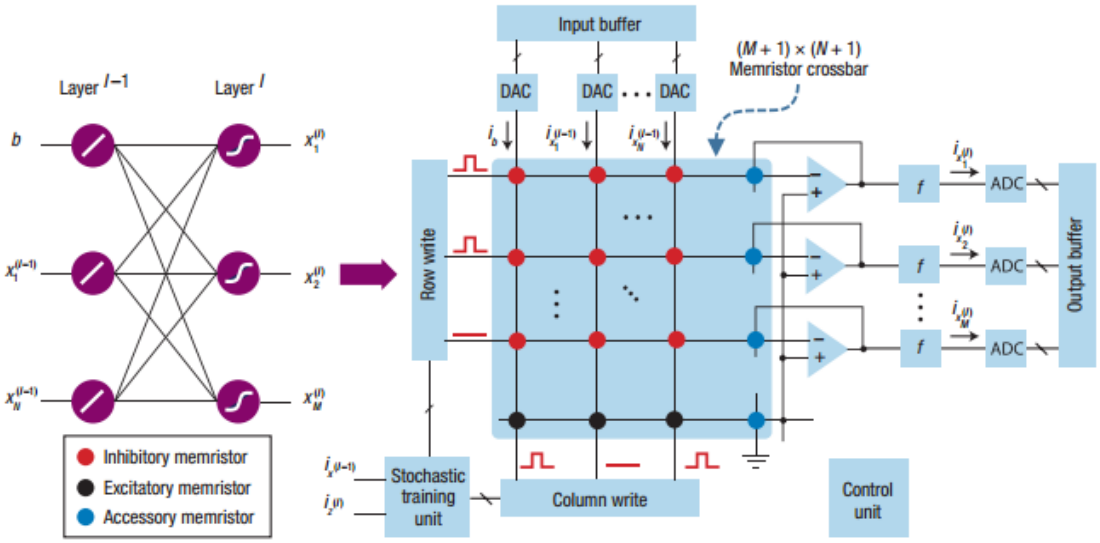

Figure 34. Translating a Neural Network (Left) onto a Current Mode Crossbar Design with Bipolar Weighting (Right)

A different approach is to implement a voltage mode memristor crossbar [21]. The first voltage mode design is advantageous over many crossbar implementations because it does not require the use of an operational amplifier on the crossbars output. In the design shown in Figure 35, each column is treated as a voltage divider. The drawbacks with this approach is that the weight between any input (j^{th} column) and output (i^{th} row) is given by a ratio of the memristance devices along the i^{th} row. This suffers from the same constraints as the current mode design due to the dependency between a memristors weight and all the other memristors. This requires mapping specific weight matrices onto the crossbar making sure that the weight matrices satisfy constraints based on the device specifications.

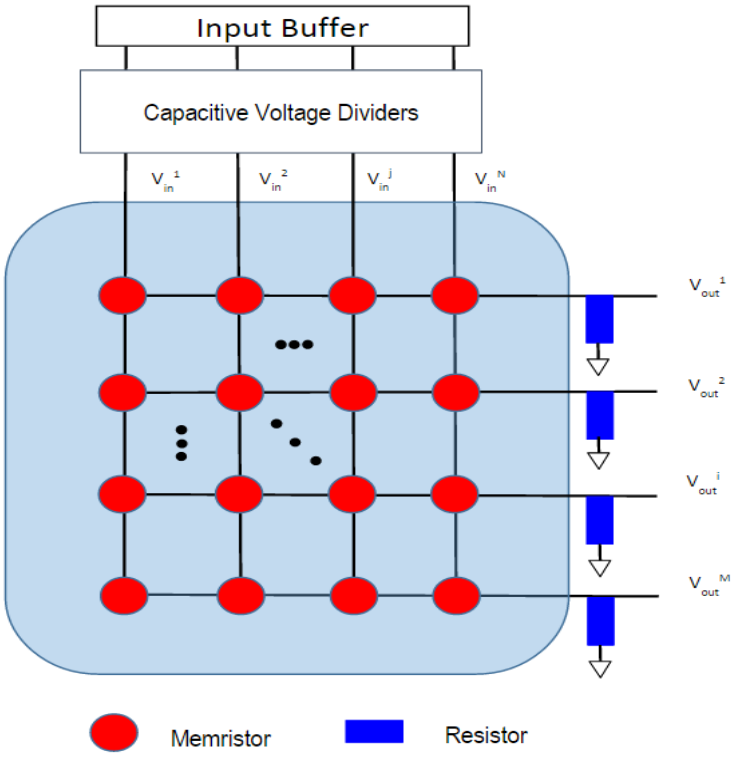

Figure 35. Voltage mode crossbar design without operational amplifiers by replacing them with a resistor to form a voltage divider between the N inputs and j^{th} output neuron

In order to achieve a direct mapping between the value of the memristor at a given cross point and the corresponding weight, an inverting amplifier can replace the output stage of the crossbar as seen in Figure 36. With this approach the weight between the ith input and jth output is given by (21), which can support a larger weight space than the previous two approaches and allows for the crossbar weights to be tuned by directly adjusting the memristors state using traditional gradient-descent based approaches.

$$W_{i,j} = \frac{R_i}{M_{i,j}} \tag{21}$$

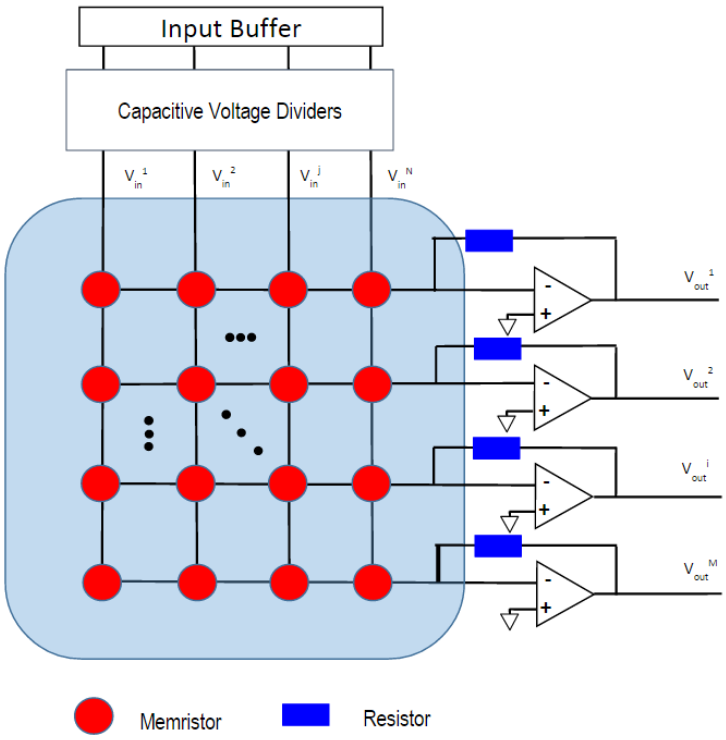

Figure 36. Voltage Mode Crossbar Design with Operational Amplifiers so that Each Synaptic Weight is Only a Function of a Single Memristor

One advantage to the current mode design is that the weight range of the crossbar can achieve bipolar weight values just by the inclusion of a single layer of inhibitory memristors. The voltage mode designs are only capable of achieving positive weight values. In order to address this problem two crossbars are used in parallel. One crossbar receives the input while the other receives the inverted input and the two results are summed together.

For on-chip training of neuromemristive architectures we have developed a mixed signal training core called Ziksa [22] capable of integrating with memristive crossbar designs to enable online learning. An example of Ziksa integrated with a crossbar is shown Figure 37. Ziksa consists of three transistors at each column and row, a global controller, and an error computation block. The error computation block determines the error of the network which is then passed to the global controller. This controller then calculates the weight update according to gradient descent and applies the appropriate signals to the transistors at each column and row to create the corresponding potential difference across the device to tune its state. As can be seen in Figure 37, four of the transistors (two on each column and row) form an H-bridge around every memristor and when turned on result in a potential across the memristor. On every row there is an additional transistor used to disconnect the crossbar from the input during training. On every column there is a transistor used to select which columns are being trained. When a column is not being trained it is driven with a potential equal to half the threshold of the memristor. In (20) the weight is dependent on the inverse of the memristance, therefore instead of performing gradient descent Ziksa updates the memristance according to (22).

$$M_{new} = \frac{M_{old} * R}{R + \alpha * M_{old} * U_i * (Y - V_{out,m})} \quad (22)$$

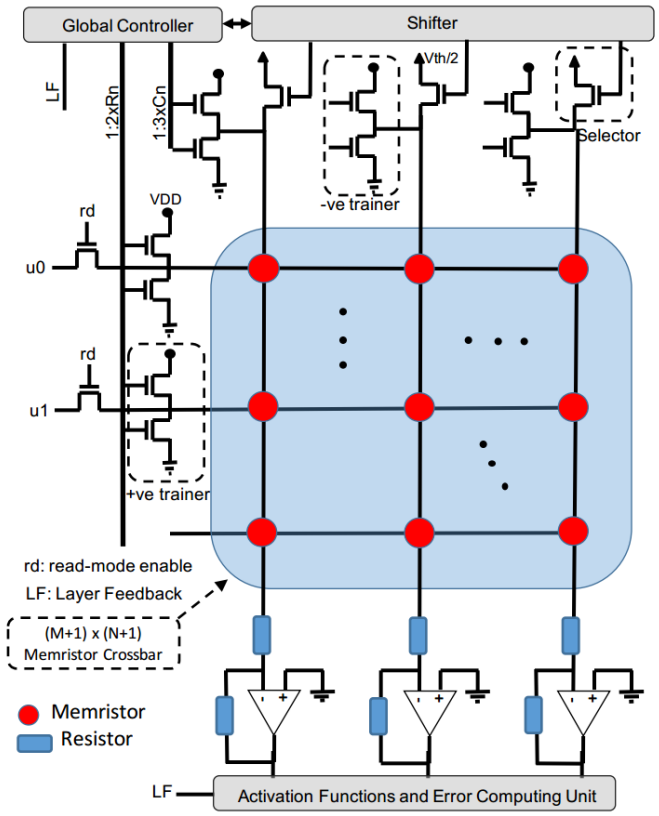

Figure 37. Ziksa On-chip Training Circuit Integrated with Memristor Crossbar for Updating the Memristors States

where R is the feedback resistance, V_{out} is the output of the post-synaptic neuron, U_i is the i^{th} pre-synaptic input, and α is the learning rate.

Each column is updated one at a time. Updating a single column requires two signals; one for increasing the memristance and one for decreasing the memristance. Ziksa is designed specifically for the last voltage mode design which uses an inverting amplifier, however the H-bridge can be used to tune the memristor state for any crossbar architecture, only the logic of the digital controller would need to be changed.

3.4.2 Neuromemristive Liquid State Machine. The primary focus of the past year has been the development of a neuromemristive architecture for implementing RC networks. Primarily the focus has been on developing circuitry for the LSM. The motivation for focusing on the LSM is due to the robustness, low precision communication, and low power consumption of spiking neural networks which makes them ideal for autonomous applications. An initial neuromemristive architecture is shown in Figure 38. We have focused our research efforts on the development of peripheral circuitry needed for network operations and on-device learning. The system consists of two separate crossbars, one to implement the reservoir and one for the output layer. The reservoir layer receives both the input to the network and the past reservoir response as inputs. The output of the reservoir layer is then fed as an input to the output layer crossbar which performs the desired task. With the ESN approach converters are necessary peripheral circuitry in between each crossbar output and buffer because of the high precision of information flowing through the network. The LSM in our model communicates with low precision information in the form of binary spike trains. This is one advantage in hardware implementations that the LSM has over the ESN because this removes the need of ADC/DAC's in between layers.

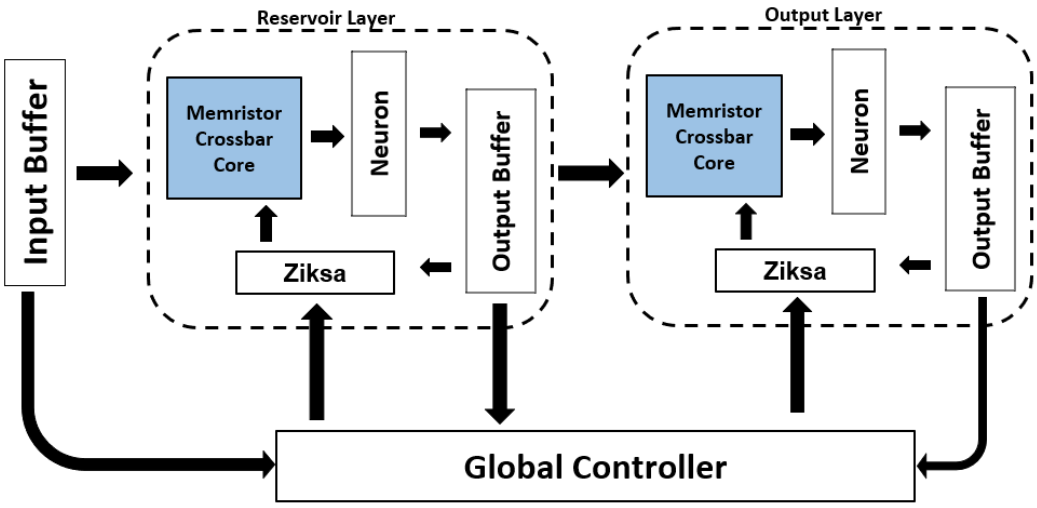

Figure 38. High-level Neuromemristive RC Architecture

Currently we have designed a reservoir layer shown in more detail in Figure 39 which consists of a memristor crossbar which connects to a current amplifier to perform the multiply and accumulate operations and feeds into an analog spiking neuron and integrated with Ziksa for on-device learning and plasticity.

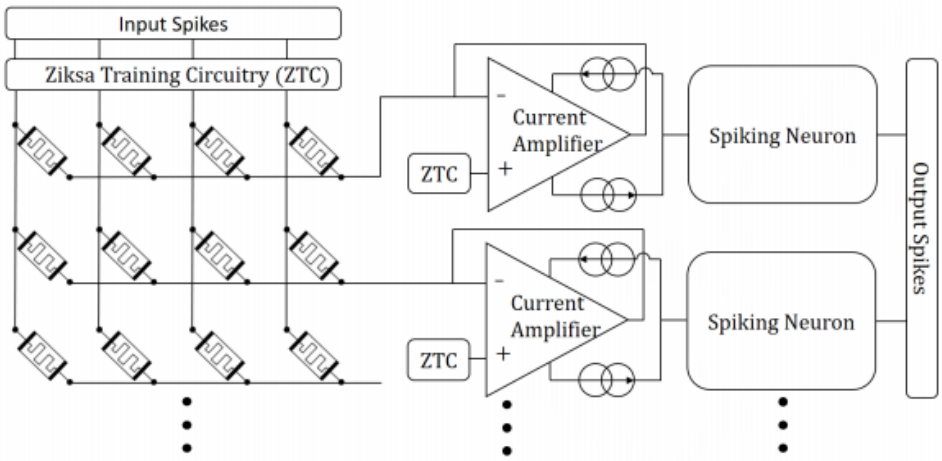

Figure 39. Neuromemristive Reservoir Layer in the LSM

The size of the peripheral circuitry needed to utilize a crossbar of memristive devices to accelerate the LSM's operation needs to be optimized for both area and energy efficiency as it will comprise a majority of the circuitry. In the proposed neuromemristive LSM model, there are three main circuits that have been developed in the last year; the Ziksa training circuitry, the current amplifier, and the spiking neuron.

The Ziksa training circuitry shown in Figure 40 will utilize transmission gates and transistors to form an H-bridge around every memristive device in the crossbar. Every column has its own ZCol and every row has its own ZRow. The first difference from the previous model is that the ZRow is integrated with the positive terminal of the current amplifier. This model draws less power during training and removes the need of a resistor which reduces area compared to the first implementation of Ziksa. A local digital control unit will be used to program the different transistors and control Ziksa's operation.

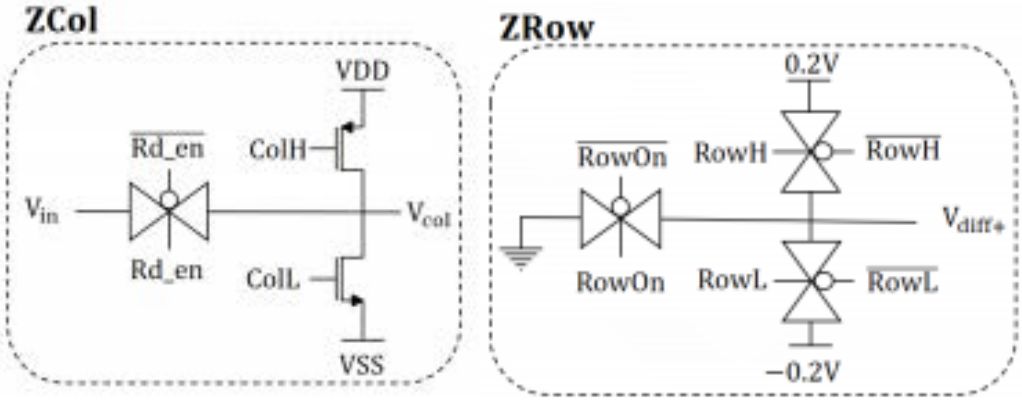

Figure 40. Ziksa Training Circuitry to Enable On-device Learning

The current amplifier (shown in Figure 41) has been designed to sink/source 20µA of current, have an input common mode range (ICMR) of -0.2V to 0.2V, and have a current gain of 1.

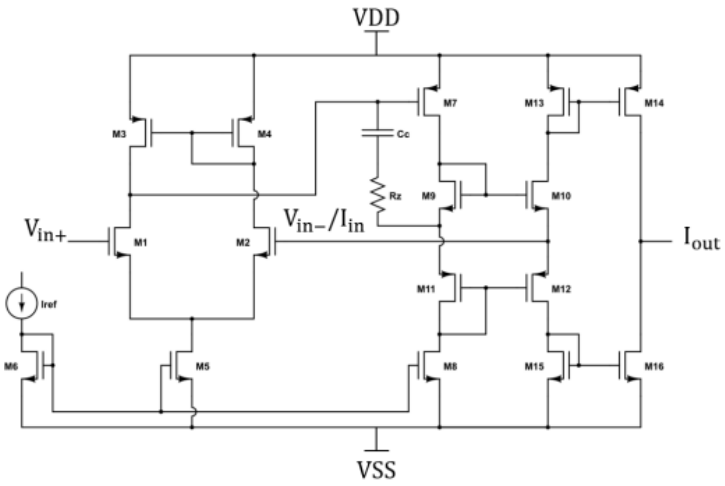

Figure 41. Schematic of Current Amplifier used in Neuromemristive LSM

The current reference for the current amplifier will supplied with the operational amplifier shown in Figure 42. The operation amplifier takes in a reference voltage in the positive terminal and the voltage across R1 in the negative terminal. A feedback loop through M7 will keep the voltage across R1 equal to V_{ref}. The current can then be modified by adjusting R1. This design will require and additional current mirror for every I_{ref} needed in the system.

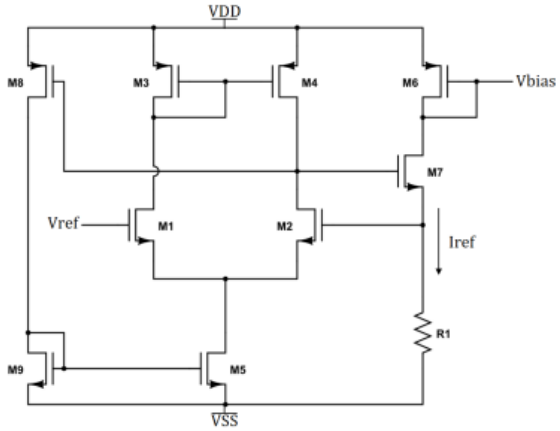

Figure 42. Schematic of Current Reference used in Neuromemristive LSM

The last component is the spiking neuron shown in Figure 43 which integrates charge over a capacitor to model the membrane potential of a biological neuron while the resistor provides a constant leakage of charge. If the potential across the capacitor exceeds a threshold, the comparators negative output will fire high. The SR latch then holds onto the result and performs a logical AND operation between the result and the clock signal. This will set the duration of the spike to be half the clock period. The reset (rst) signal will trigger the transmission gate to drain the capacitor, resetting the neuron. This operation is shown in Figure 44.

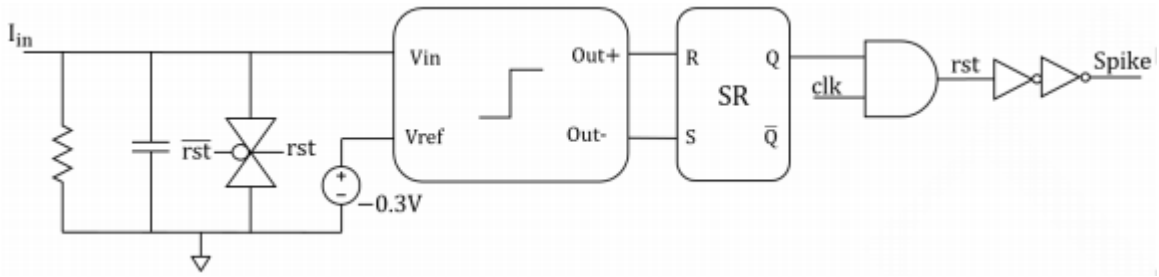

Figure 43. Schematic of Analog LIF Neuron used in Neuromemristive LSM

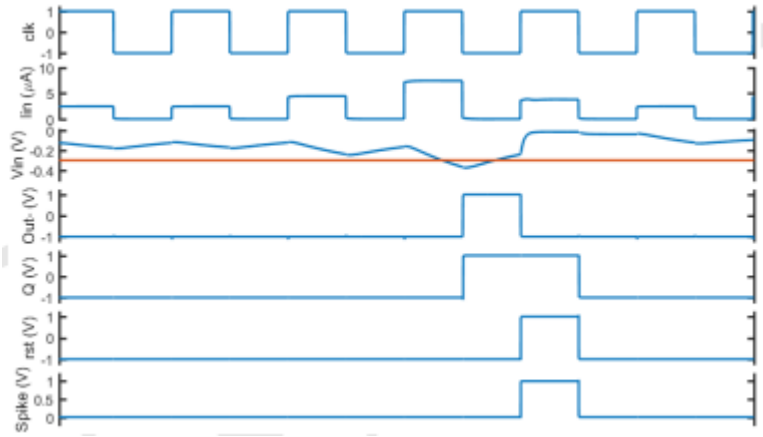

Figure 44. Operation of Analog LIF neuron. When the Potential Across the Capacitor Reaches the Threshold (-0.3V), the Out$^-$ Terminal of the Comparator Goes High Which Triggers a Spike and Resets the Potential Across the Capacitor to Ground.

In this architecture, the comparator is a double tail latched comparator which is shown in Figure 45. When the clock signal switches high, the differential amplifier made of M1 – M5 will output a high enough signal to trigger the latch (made of M7, M8, M11, and M12) to grab the difference and force the output to the rails. In a two clock cycle the differential amplifier would operate as follows. At t1, the clock is low allowing M3 and M4 to charge the top of the differential amplifier towards VDD while M5 is off preventing any computation from occurring. At t2, the clock will go high and if V_{in} is greater than V_{ref}, the drain of M3 will drop faster than the drain of M4 and the latch will cause Out^+ to rise higher than Out^-. By t4 the signals will be pulled to the rails as shown in Figure 46. In order for the differential amplifier to exhibit this behavior, the most important aspect is to ensure the difference between the outputs is large enough to allow the latch to decide correctly by t3. This implies that the gain of the differential amplifier must be high so the difference is large enough. Also, the time constants of the differential amplifier and latch must be small so they can charge quickly. Finally, the output loads of Out^+ and Out^- must be equal.

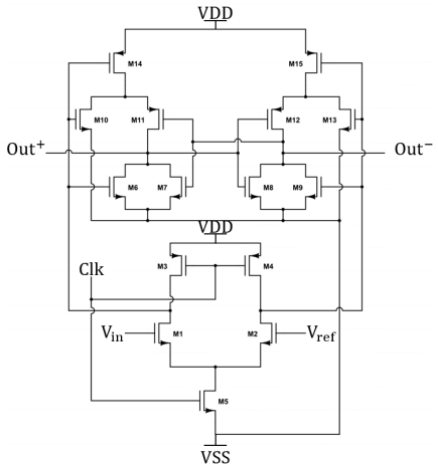

Figure 45. Schematic of Double Tail Latched Comparator used in the Neuromemristive LSM

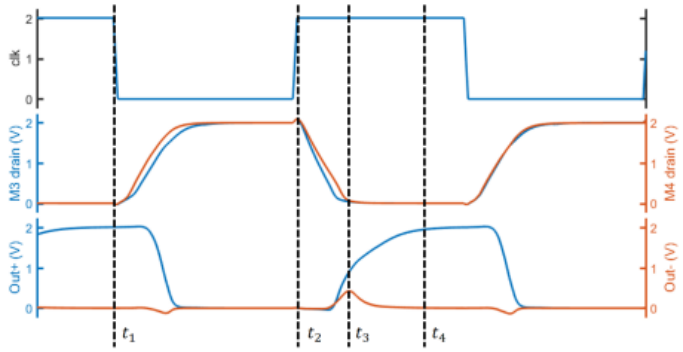

Figure 46. Operation of Double Tailed Latch Comparator

A Banba bandgap voltage reference is used to supply V_{ref} to the comparator (shown in Figure 47). The Banba bandgap was chosen because of its functionality at low voltages. One voltage reference can be used to supply the V_{ref} to all the spiking neurons in the design. The startup circuit is included to supply a small current when the chip is turned on to ensure that the nodes will not get stuck at the rails.

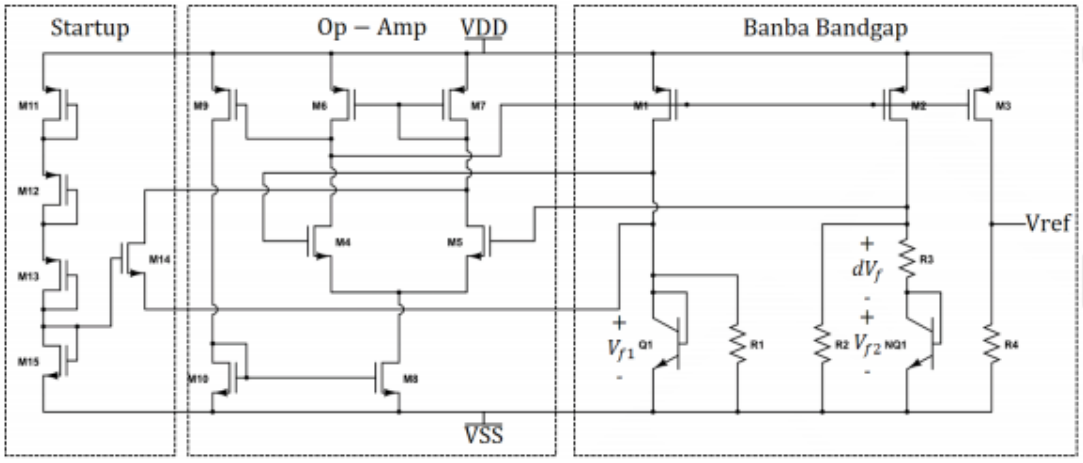

Figure 47. Schematic of Banba Bandgap Voltage Reference used in the Neuromemristive LSM

An improvement to the neuromemristive LSM design was the implementation of a spiking memristor neuron model and a memristor based synaptic trace circuit that can be used for rate-encoding and Hebbian based learning.The proposed spiking neuron circuit [23] is depicted in Figure 48. The circuit is comprised of three essential parts: a memristor, switched current source, and spike generator (tri-state inverter). The memristor changes its resistivity as it receives the pre-synaptic spikes (input current). The switched current source is used to read the voltage drop across the memristor as a function of the applied current. The tri-state inverter operates as a voltage comparator that generates a neuron output spike when the voltage drop across the memristor surpasses the neuron threshold.

The neuron circuit operates in two modes: In MODE-I, the neuron stores the pre-synaptic spikes in a memristor. This is achieved by allowing only the current received from the synapses to flow through the memristor i.e. T1 and T2 are set to be OFF. When the current flows in the memristor, it starts to increase its resistivity, and is modeled by (23):

$$\frac{\Delta w}{\Delta t} = \frac{\mu \times Ron}{D} \times \mathrm{i(t)} \tag{23}$$

Where w is the device state variable, μ is the dopant mobility, R_{on} and D denote the device low-resistive state (LRS) and its thickness, respectively. Once all the pre-synaptic spikes are stored, MODE-II operation starts. In this mode the spikes stored in the memristor get evaluated. This is accomplished by turning ON T1 and allowing $I_{capture}$ to flow through the memristor. If the voltage across the memristor terminals (V_{mem}) exceeds the neuron threshold (inverter threshold in this work), the neuron generates an output spike, otherwise, the memristor will be back to LRS via T1 and through I_{drain}. It is important to mention here that while neuron receives the pre-synaptic spikes, V_{mem} may go beyond the inverter threshold leading to an unintended output spike. However, these spikes will be blocked by the output tri-state buffer. The operation of the proposed spiking neuron design is shown in Figure 49.

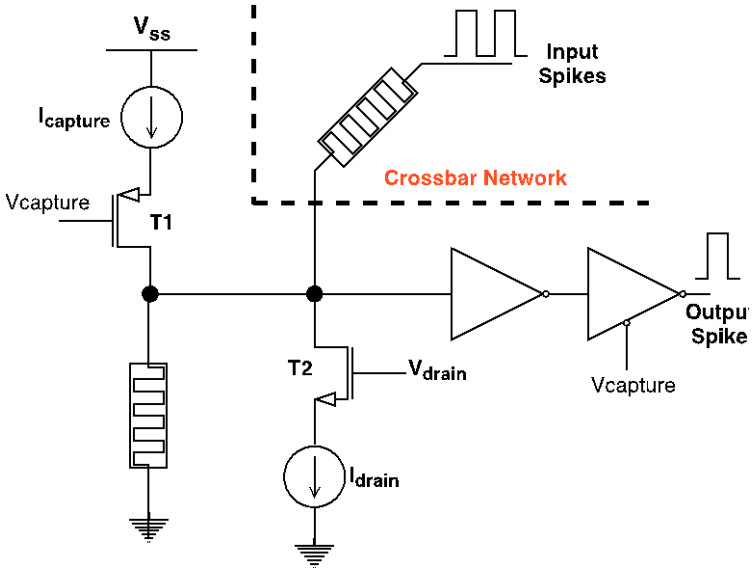

Figure 48. Memristor Spiking Neuron Circuit Integrated in a Crossbar Bit-line

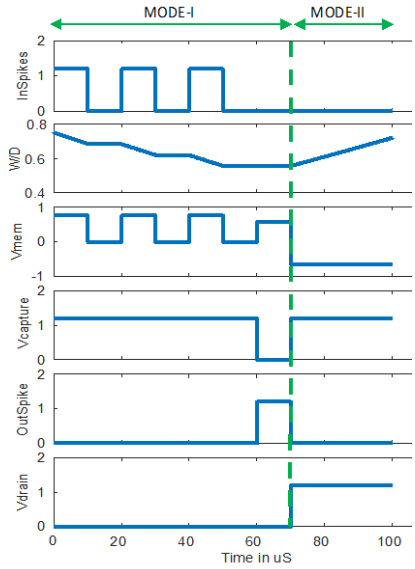

Figure 49. Spiking Neuron Timing Diagram Demonstrating the Operation of the Proposed Circuit while Receiving Input Stimulus, Generating Output Spikes, and Resetting the Memristor

In [23], an additional circuit, called synaptic trace circuit (shown in Figure 50), is proposed to record the activities of the synaptic connections. The synaptic trace circuit has the same components as spiking neuron with additional input buffers to eliminate any impact the circuit may impose on the crossbar with which it is integrated. Since the input buffer used in this circuit is an inverter with a threshold of 0.6v (IBM 65nm), the synaptic trace input inverter gate is biased to respond to input voltages as low as 0.45v. The synaptic trace circuit can be used in the LSM to track the activity of neurons in the reservoir and send their encoded firing activity to the output layer. This has been shown to improve performance in classification tasks with the LSM. Additionally, it can be used to implement a simplified spike-timing dependent learning rule given by

$$\Delta W = \alpha * sign(X_{trace} - X_{target}) \quad (24)$$

where the change in synaptic strength (ΔW) is determined by the difference between the synaptic trace (X_{trace}) and a target activity rate (X_{target}). This learning rule will strengthen any pre-synaptic connection whose trace value is above a certain threshold and decrease all other pre-synaptic connections.

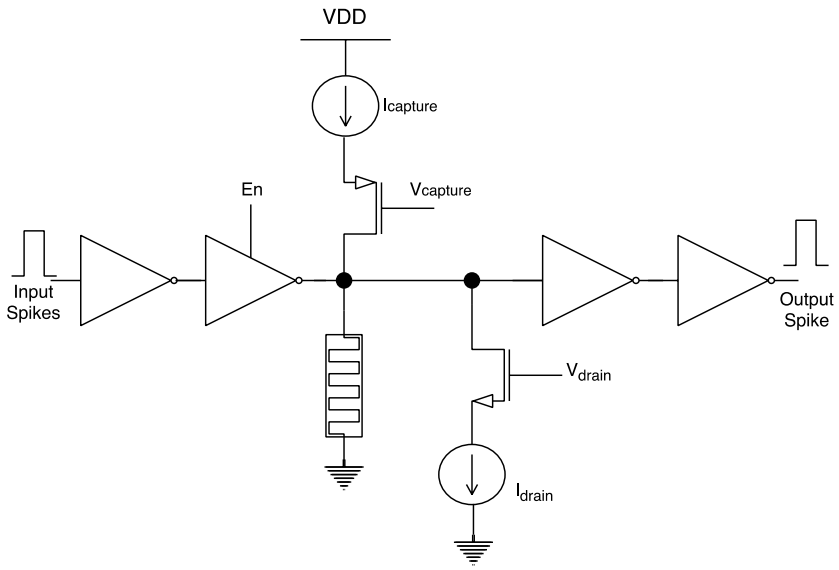

Figure 50. Synaptic Trace Circuit Associated with each Synapse to Record the Recent Spike History

An overall architecture of the spiking neural network (SNN) illustrated in Figure 51, is composed of memristive crossbar network, Ziksa trainer, and global and local controllers. The crossbar network represents one layer of SNN in which each column corresponds to one spiking neuron connected to *Rn* (*Rn* is the number of crossbar rows) number of synapses. The Ziksa trainer along with the local controller modulates the synaptic weights in the learning phase. The global controller is responsible for controlling data flow and network synchronization.

The SNN operates in two main phases: inference and training. During the inference phase, the input to the network is presented as a stream of spikes, where each spike is encoded with one digital bit (logic'1' for spike and logic'0' for non-spike). The input spikes are relayed to the crossbar network via pass transistors which are ON only during the inference phase. The input signals are scaled by the memristive synaptic weights. The output of the memristor will be a current signal summed at the bit-lines of the crossbar and delivered to the neuron. It is important to note that while the neurons are receiving pre-synaptic spike trains, the synapse activities are registered by the synaptic trace circuit. After the inference phase, the training phase will commence by capturing neurons outputs and storing it into a parallel-in-serial-out shift register. Typically, the training is performed column-by-column where two clock cycles are required for each column. In the first clock cycle all the synapses that are expected to be potentiated are modulated, while the synapses that are depressed will be modulated in the second clock cycle.

The training process is performed by Ziksa, composed of 3 transistors for each column and 2 for the crossbar rows, column-by-column on the crossbar. For instance, if the first column, C_1, is to be trained, the transistors (X_1, G_1, $P_{1\text{-}n}$, and $N_{1\text{-}n}$) are employed to form an H-Bridge across C_1 memristors. By using the row and column local control units, the H-Bridge transistors apply either a positive or negative voltage across the memristor to adjust its value. If a synapse needs to be potentiated, G_1 and $P_{1\text{-}n}$ are set to be ON such that the current flows from the rows towards the column and the opposite occurs when the synapses are depressed. During the tuning of C_1 memristors, the columns that are not involved in the training i.e. C_2- C_n are excluded by applying V_{DDD} to their bitlines via Y_2 - Y_m transistors. This ensures that the voltage across the columns C_2- C_n is always less than the memristor threshold. It is important to note that prior to synapse weight adjustment, spiking neuron activities (stored in the shift register) are sent to the local control units along with the synapse traces.

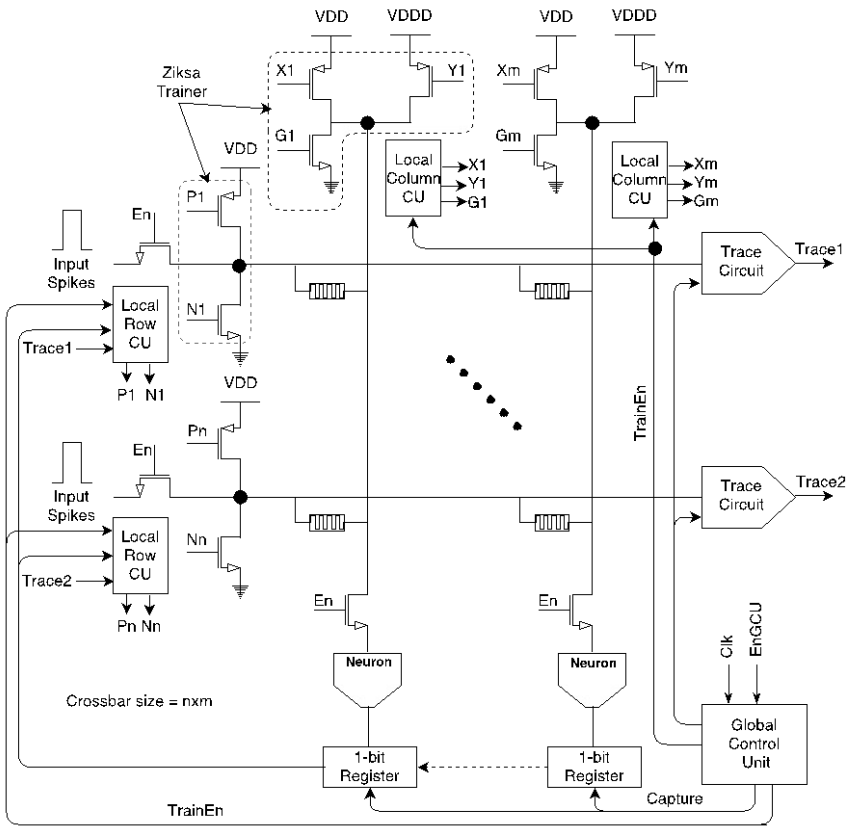

Figure 51. Architecture of a Single Layer Spiking Neural Network

In this work, A Verilog-A memristor model is employed, where the synaptic memristors are given high resistivity range (200 KΩ – 1 MΩ) to limit the amount of current going to the neuron circuit. In case of the neuron memristor, its high resistance state (HRS) is set to be 2.25x lower than the minimum resistance of the synaptic memristor. This is to suppress the influence of neuron memristance changing based on the amount of current flowing into the bit-lines.

The switched current sources used in the neuron circuit, should be designed such that $I_{capture}$ is less than the memristor threshold-current to avoid the undesired change in the memristor during MODE-II. At the same time, $I_{capture}$ must be large enough to cause a voltage drop across the inverter gate when neuron memristance reaches ~ R_{off} (R_{off} is memristor's HRS). The I_{drain} should be more than the memristor threshold-current to reset the memristor to its LRS, before the next inference iteration.

The following constraints must be fulfilled for the crossbar pass transistors:

- Limit the input voltage such that $V_{DS} << |V_{GS} - V_{Th}|$. This ensures that the transistor is working in the triode region and an undistorted signal reaches the memristors.
- Assume that $K(V_{GS} - 2V_{Th}) >> G_{mem}(w)$ such that the conductance of the transistor is higher compared to the memristor conductance. Thus, the voltage at the memristor is ~ input voltage.

4.0 RESULTS AND DISCUSSION

4.1 Software Models

4.1.1 Optimized LSM. Implementing both synaptic scaling in the initialization of the LSM and dynamic synapses with STP achieved 86.25% accuracy on a 4-phoneme classification task. Implementing a LSM with only the synaptic scaling but no STP resulted in 55.44% accuracy. Without using STP or synaptic scaling achieves around 40% in literature on the same task [24] on the TIMIT speech corpus [25].

4.1.2 Deep-LSM. The deep-LSM was tested on seizure detection, speech recognition, and the DogCentric dataset. The results on all three datasets were competitive with SOTA and are summarized in Table 4.

Table 4. Performance of deep-LSM on Several Benchmark Real-world Applications

Dataset	Accuracy
EEG Seizure Detection	93.20%
39 Phoneme Isolated Speech Recognition	92.36%
Video Activity Recognition	84.78%

For the seizure detection task [35], the proposed deep-LSM outperforms other reservoir-based methods [26], but is below SOTA which was 100% achieved using wavelet transformations and feedforward neural networks making the approach not real-time. For isolated phoneme recognition, the TIMIT speech corpus was split into 39 isolated phonemes and beat SOTA on isolated phoneme recognition. Lastly, for the DogCentric dataset the deep-LSM achieved SOTA outperforming all relative models (Table 5) with and without hand-crafted features. For the DogCentric task ResNet-50 was used as the pre-trained feature extractor and Principal Component Analysis (PCA) was used to reduce the dimensionality of the inputs to 100 features before being sent to the deep-LSM. All three models in these results achieved the listed performance with three hidden layers with 500 neurons each.

Table 5. Comparison of deep-LSM on DogCentric Dataset with Current SOTA Models

	Approach	Accuracy
HCF	GOFF + VIF + Log-C + Cuboids [3]	64.0%
	HOG+HOF+LBP+Cub.+Opt.Fl. ([14])	60.5%
	ITF[28, 41]	67.7%
	ITF+CNN[28, 17]	69.2%
	POT[31]	73.0%
	POT+ITF[31]	74.5%
	TDD[28, 42]	76.6%
	TDD+Temp. Fil. [28]	79.6%
	TDD+Temp. Fil.+LSTM[28]	81.4%
No HCF	VGG+Max Pooling[28]	≈ 57.2%
	VGG+Mean Pooling[28]	59.9%
	VGG+Sum Pooling[28]	59.9%
	VGG+Temp. Fil.-Learned[28]	≈ 65.0%
	VGG+Temp. Fil.-Learned+LSTM[28]	≈ 65.0%
	CDN (VGG-16) [9]	75.8 %
	CDN (ResNet-50) [9]	77.2%
	TCF (CaffeNet) [18]	72.19%
	TCF (VGG-16) [18]	77.79%
	TCS (VGG, TDD) [18]	82.24%
	LFP (G+SD+GS) [19]	82.5%
	deep-LSM (ResNet-50)	**84.78%**

The deep-LSM was also studied for the impact on sweeping the number of layers, hidden layer size, and WTA layer size as shown Figure 52-Figure 54.

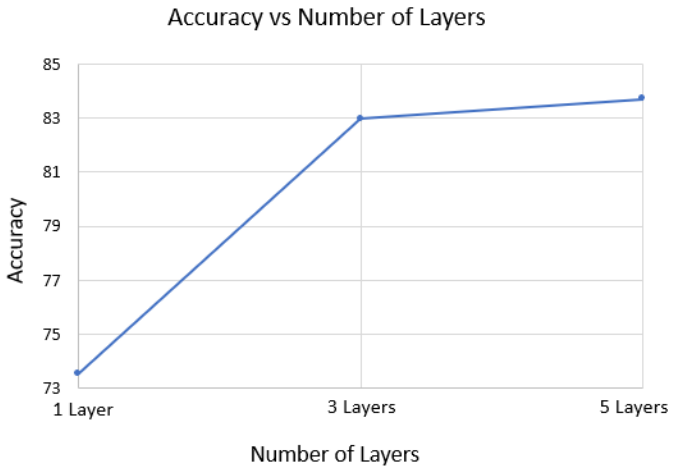

Figure 52. Deep-LSM Performance on DogCentric Dataset as the Number of Hidden Layers Increases

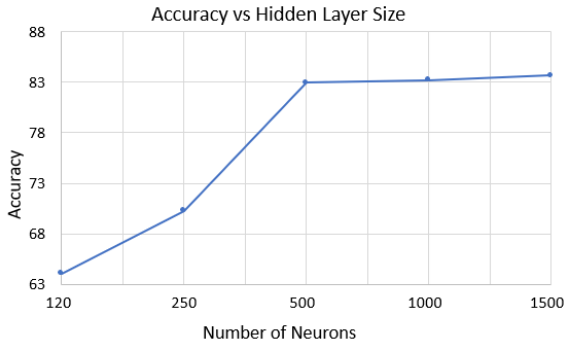

Figure 53. Deep-LSM Performance on DogCentric Dataset as the Size of the Hidden Layers Increase

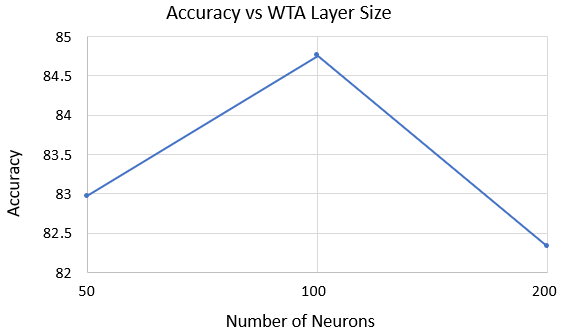

Figure 54. deep-LSM Performance on the DogCentric Dataset as the Size of the WTA Layer Increases

Table 6. Total Number of Synaptic Connections (Listed by Plasticity Mechanisms) and the Total Memory Consumption for the deep-LSM Compared to a Vanilla LSM and LSTM

	Backpropagation	Random	Unsupervised	Total	Memory (Gb)
Deep-LSM	759500	850000	50000	1659500	0.0531
LSM	15000	2400000	0	2415000	0.0773
LSTM	9615000	0	0	9615000	0.3077

Table 7. Total Number of Multiplication Operations and Weight Updates for a Single Training Sample. The deep-LSM Considered has Three Hidden Layers with 500 Neurons, Two WTA Layers with 50 Neurons, Two Attention Layers with a Total of 503 Neurons, and a Readout Layer with 10 Neurons. The Vanilla LSM and LSTM have a Hidden Layer with 1500 Neurons (Comparable to Three 500 Hidden Layers in the deep-LSM) and 10 Output Neurons. The LSTM is Considered for Under the Most Optimal Conditions where the Error is Backpropagated over a Single Time-step (i.e. no BackpropagationTthrough Time)

Network	# Multiplications (FP)	# Multiplications (BP)	# Weight Updates
deep-LSM	1,661,500	773,506	760,500
LSM	2,415,000	15,000	15,000
LSTM	9,619,500	9,675,000	9,615,000

Table 8. Total Energy Consumption for a deep-LSM, Vanilla LSM, and LSTM Under the Same Conditions Reported in Table 6. The Numbers are Estimated by Computing the Number of Multiplication and Additions Required for Inference and Training on a Single Set of Video Frames in the DogCentric Dataset, and an Estimated Energy Consumption of each Operation in a 32-bit Architecture

	Inference Energy(μJ)	Training Energy (μJ)	Weights Energy (μJ)	Total Energy (μJ)
Deep-LSM	7.01	3.10	1062.1	1072.2
LSM	11.11	0.069	1545.6	1556.88
LSTM	44.24	55.56	5866.6	5966.4

4.1.3 Mod-DeepESN. The Mod-DeepESN was tested on the Mackey-Glass time-series and a daily minimum temperature series dataset which contain variable nonlinearity at multiple timescales. The performance for different architectures on the datasets is reported in Table 9 and Table 10 respectively.

Table 9. Performance of Mod-DeepESN architectures on Mackey-Glass time series prediction compared to different baseline models

	Method	N_L	N_R	β	α	ρ	σ_{in}	σ_l	s_{in}	$\hat{s}_l$	s_l	IP	RMSE×e-3	NRMSE×e-3	MAPE×e-3
Baseline	ESN [1]	1	-	-	-	-	-	-	-	-	-	-	43.7	201	7.03
	ϕ-ESN [2]	2	-	-	-	-	-	-	-	-	-	-	8.60	39.6	1.00
	R^2SP [3]	2	-	-	-	-	-	-	-	-	-	-	27.2	125	1.00
	MESM [4]	7	-	-	-	-	-	-	-	-	-	-	12.7	58.6	1.91
	Deep-ESN [5]	3	-	-	-	-	-	-	-	-	-	-	**1.12**	**5.17**	**.151**
This Work	*Wide*	3	256	2e-8	.6	*X*	.1	*X*	.1	.1	.7	N	20.2	48.7	8.11
	Layered	3	256	2e-8	.6	*X*	.1	*X*	.1	.1	.7	N	57.8	96.2	19.9
	Criss-Cross	4 (2)	256	2e-8	.6	*X*	.1	*X*	.1	.1	.7	N	56.1	54.4	8.94
	Wide+Layered	6 (3)	256	2e-8	.6	*X*	.1	*X*	.1	.1	.7	N	41.1	55.4	11.2
	Wide	3	256	2e-8	.6	*X*	.1	*X*	.1	.1	.7	Y	7.22	27.5	5.55

Table 10. Performance of Mod-DeepESN Architectures on Daily Minimum Temperature Time Series Prediction Compared to Different Baseline Models

	Method	N_L	N_R	β	α	ρ	σ_{in}	σ_l	s_{in}	$\hat{s}_l$	s_l	IP	RMSE×e-3	NRMSE×e-3	MAPE×e-3
Baseline	ESN [1]	1	-	-	-	-	-	-	-	-	-	-	501	139	39.5
	ϕ-ESN [2]	2	-	-	-	-	-	-	-	-	-	-	493	141	39.6
	R^2SP [3]	2	-	-	-	-	-	-	-	-	-	-	495	137	39.3
	MESM [4]	7	-	-	-	-	-	-	-	-	-	-	478	136	37.7
	Deep-ESN [5]	2	-	-	-	-	-	-	-	-	-	-	473	135	**37.0**
This Work	*Wide*	2	1024	7e-4	1	*X*	.4	*X*	.6	.3	.6	N	473	135	38.6
	Layered	2	1024	7e-4	1	*X*	.4	*X*	.6	.3	.6	N	470	134	38.2
	Criss-Cross	4 (2)	1024	7e-4	1	*X*	.4	*X*	.6	.3	.6	N	472	135	38.7
	Wide+Layered	4 (2)	1024	7e-4	1	*X*	.4	*X*	.6	.3	.6	N	471	135	38.6
	Wide+Layered	4 (2)	1024	7e-4	1	*X*	.4	*X*	.6	.3	.6	Y	**459**	**132**	37.1

4.1.4 CDN. The CDN network was tested with VGG-16 and ResNet-50 for transfer learning of filter and a ESN with different reservoir sizes to capture temporal data. The two datasets used in this analysis were the UEC-Park and DogCentric datasets. As shown in Figure 55, increasing the reservoir size on this application was not necessary as 600 neurons converged to the same performance as up to 2400 neurons, the only difference being the larger reservoirs converged faster.

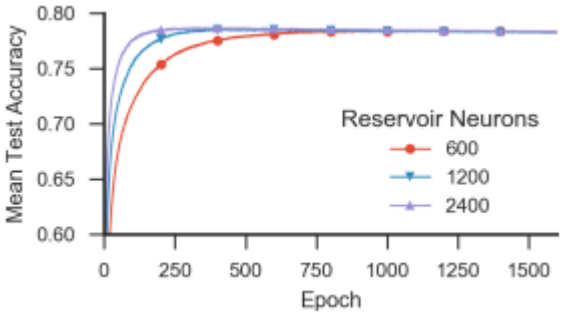

Figure 55. Performance of CDN Network on UEC-Park Dataset for Different Reservoir Sizes

In comparison to SOTA, the CDN at the time of publication outperformed SOTA for models with no hand-crafted features and near SOTA for models with hand-crafted features.

Table 11. Performance of CDN on DogCentric Dataset Compared to SOTA at Time of Publication

	Approach	Accuracy
HCF	HOG+HOF+LBP+Cub.+Opt.Fl. [21]	60.5%
	ITF [14], [24]	67.7%
	ITF+CNN [14], [25]	69.2%
	POT [13]	73.0%
	POT+ITF [13]	74.5%
	TDD [14], [26]	76.6%
	TDD+Temp. Fil. [14]	79.6%
	TDD+Temp. Fil.+LSTM [14]	81.4%
No HCF	VGG+Max Pooling [14]	≈ 57.2%
	VGG+Mean Pooling [14]	59.9%
	VGG+Sum Pooling [14]	59.9%
	VGG+Temp. Fil.-Learned [14]	≈ 65.0%
	VGG+Temp. Fil.-Learned+LSTM [14]	≈ 65.0%
	CDN (VGG-16)	**75.8 %**
	CDN (ResNet-50)	**77.2 %**

(**Bold**: Our approach) (≈: est. from graph)

Table 12. Performance of CDN on UEC-Park Dataset Compared to SOTA at Time of Publication

	Approach	Accuracy
HCF	STIP+IFV [13], [27]	69.1%
	Cubiod+IFV [13], [28]	72.3%
	ITF+CNN [25]	75.7%
	IFV+Pooling [13]	76.4%
	BoW+Pooling [13]	76.5%
	Inria ITF+IFV [13], [29]	76.6%
	POT [13]	79.4%
	POT+ITF [13]	79.5%
No HCF	**CDN (ResNet-50)**	**78.7%**

(**Bold**: Our approach)

4.2 Digital Architectures

4.2.1 Digital ELM. The digital ELM architecture was validated on MNIST for different hidden layer sizes as shown in Figure 56, with the power consumption listed in Table 13. The area for the entire stochastic ELM implementation was found to be 1,423,090.3μm^2 for 45nm Complementary Metal-Oxide-Semiconductor (CMOS) technology (258 Neurons, 256 in the hidden layer and 2 output neurons).

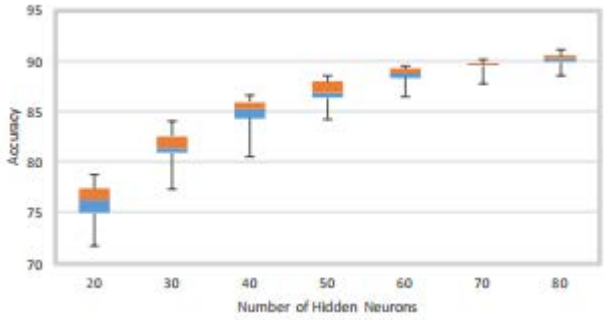

Figure 56. Digital ELM Accuracy for MNIST Dataset

Table 13. Power of Different Functional Blocks for a Digital ELM

High Level Functional Blocks	Power (mW)
Global controller and synchronizer	0.0147
Input Layer	2.34
Hidden Layer (80 Neurons)	9.29
Output Layer (12 Neurons)	1.96

In order to validate the proposed hardware for classification task, three datasets are identified. Two of them are binary classification benchmarks from UCI library, Diabetes and Australian Credit [27]. The third dataset is MNIST [28], a hand-written digit standard dataset with 10 classes. The MNIST dataset is validated on the full digital architecture implementation of ELM and the proposed topologies. The Diabetes and Australian Credit are validated on the MATLAB emulation of ELM architecture. An ELM architecture proposed earlier by our group is chosen as a baseline. Table-II depicts the classification accuracy (average of 10 runs) achieved in this work and compares it to [29] and [30]. It can be noticed that with fewer number of hidden neurons, comparable performance is achieved. Accuracy variation within ~1% can be attributed to the random initialization of the hidden layer.

Table 14. Summary of Classification Accuracy for Binary and Multi-class Datasets using ELM

Dataset	Dimen.	ELM [29]	VLSI ELM [30]	This work
Diabetes	8x1	77.95%	77.09%	76.25%
Australian Credit	14x1	87.89%	87.89%	88.86%
MNIST	145x10	-	-	91.2%

[a] Software implementation of ELM.
[b] MNIST images are processed with HOG descriptor prior to fetch it to ELM.

Figure 57 illustrates the throughput of various topologies. As expected, the baseline has the highest throughput. This is attributed to resource reusability in the three topologies which leads to 2.16x performance degradation (when k = 4). The throughput can be measured using (25), where #MACs refers to the number of multiply-accumulation operations that take place starting from feeding an inference sample to the network until the final network output is generated. The hardware can run up to 184.18 MHz However, as the throughput and total power of the system are highly affected by the frequency, 100 MHz is chosen.

$$\text{Throughput} = \#\text{MACs} \times \text{CR} \tag{25}$$

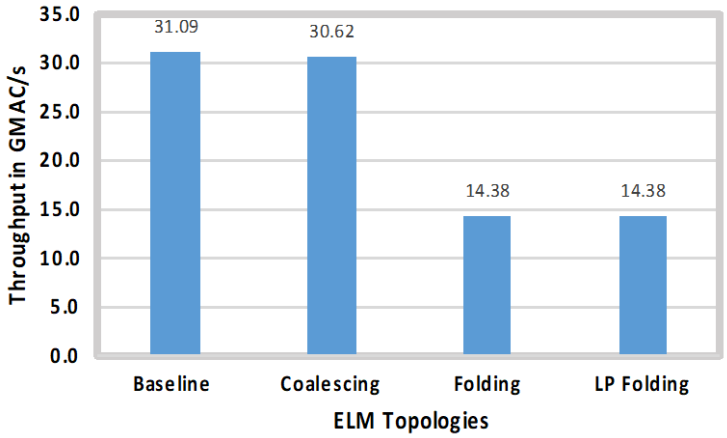

Figure 57. The Throughput of Different ELM Architecture Topologies Represented in GMAC/s

In order to observe the impact of resource reusability and low precision, all the proposed topologies are synthesized on FPGA platform Zynq-7000 xc7z030fbv676-3 and characterized by the number of Look Up Tables (LUTs), First Fits (FFs), and Digital Signal Processors (DSPs) units as shown in Figure 58. For a network size of 145x80x10, LP Folding requires fewest number of resources (DSPs, LUTs, FFs, and memory blocks) as it uses a smaller number of neurons and less resolution.

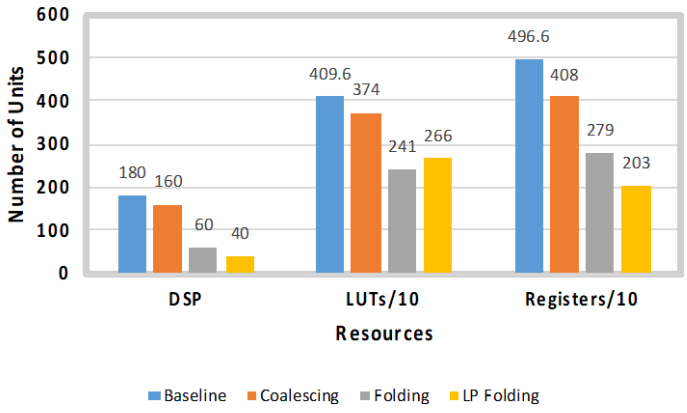

Figure 58. Resource Utilization for the Three Topologies in Terms of LUTs, FFs, and DSPs

Since we identify that LP Folding offers reasonable performance and reduced resource utilization, this topology was synthesized in Synopsys for TSMC 65nm technology node and verified with 2000 feature vectors that are propagated through the network running at 100 MHz. The area and total power of the LP Folding are 0.10 mm^2 and 3.65 mW, respectively. In an attempt to make a fair comparison with the previous ELM implementations, the measurement gap in dynamic power between FPGA and ASIC proposed by Ian Kuon [31] is adopted. This measurement estimates the dynamic power measured on FPGA as approximately 12 times (we understand that current FPGA boards have more power efficiency, and this is a slightly pessimistic approach) on average more than that on ASIC for the same design. According to the power analysis of the real-time ELM implementation proposed in [32], the dynamic power is equivalent to 22.91 mW on ASIC. Although the real-time ELM network has a smaller network size compared to LP Folded ELM, it still consumes more power. This is because the real-time ELM implementation uses more memory units and represents data with higher number of bits. For the proposed design in [33], the authors did not report the dynamic and the static power independently. Also, they did not discuss the specific power measurement approaches which make it hard to compare with the aforementioned design. In case of VLSI ELM [34], the network exhibits very low power consumption but low throughput compared to LP Folding. Running the LP Folding for the same throughput in [34] results in a power consumption of 200 µW.

4.2.2 Digital ESN. The benchmarks used for the ESN were epileptic seizure detection and prosthetic finger control. In the case of seizure detection, the ESN achieved up to 90% accuracy as shown in Figure 59. For prosthetic finger control the ESN achieved around 84% accuracy as shown by the confusion matrices in Figure 60.

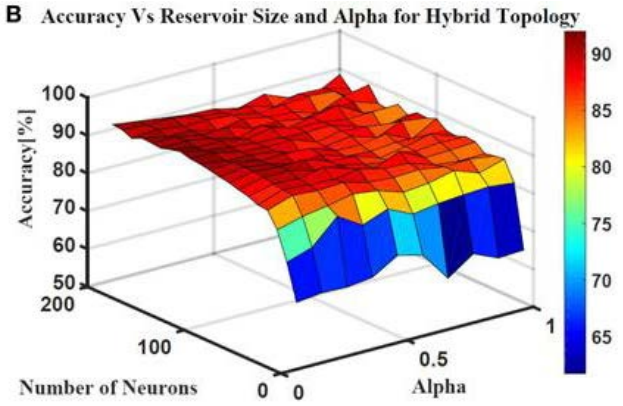

Figure 59. Digital ESN Accuracy for EEG Seizure Detection

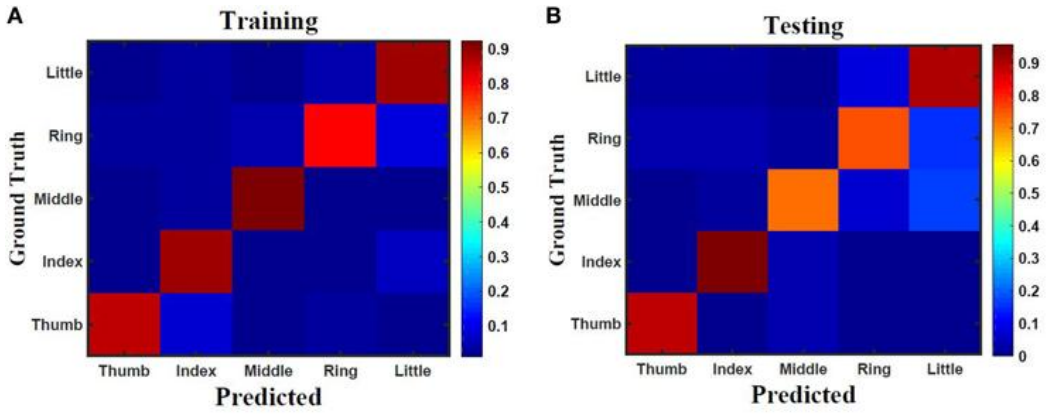

Figure 60. Digital ESN Accuracy for Prosthetic Finger Control

To evaluate the reservoir behavior during the two tasks the kernel quality and the Lyapunov exponent are computed. The kernel quality is shown for the two tasks versus increasing reservoir size in Figure 61. The Lyapunov exponent is shown for both tasks for increasing reservoir sizes in Figure 62.

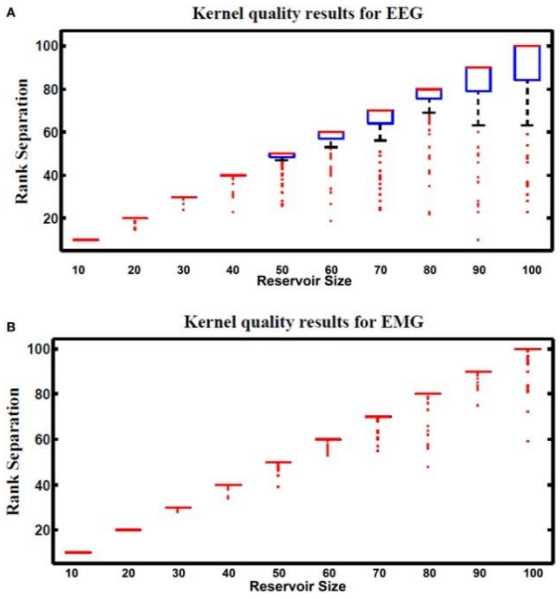

Figure 61. Digital ESN Kernel Quality for EEG and EMG Datasets

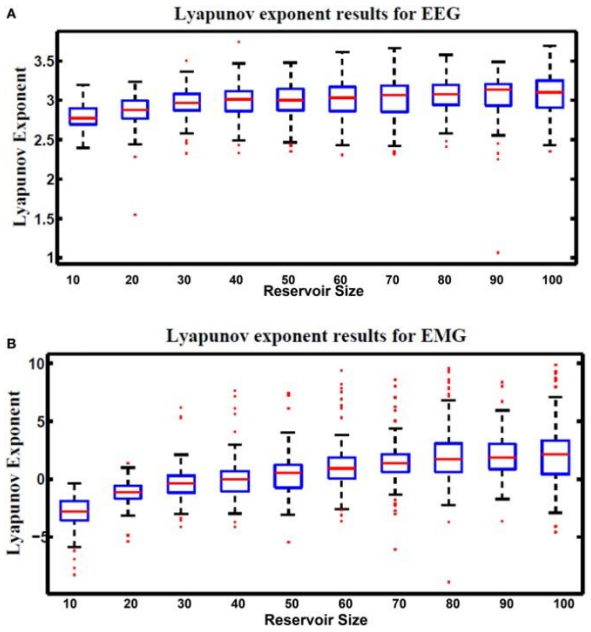

Figure 62. Digital ESN Lyapunov Exponent for EEG and EMG Datasets

In order to reduce the complexity of the ESN even further we explored using a linear reservoir with a non-linear readout as well as sparse connected readout layers. Both networks architectures resulted in significant decrease in performance for the epileptic seizure detection reaching up to 58% accuracy. The FPGA resources for different FPGA boards for a single neuron is given in Table 15. The power for a 30 neuron ESN using the hybrid topology is given for different FPGA boards as well as a mixed-signal approach in Table 16. Lastly, Figure 63 shows how power grows for different network topologies as the reservoir size grows.

Table 15. FPGA Resources for the Digital ESN

FPGA	Clock (nS)	LUT's	FF's	Mult (mW)	Tanh (mW)	Sum (mW)	Total (mW)
Virtex5-LX110T	2.4	199	16	6.7	0.14	6.31	28.19
Virtex6-LX550TL	2.28	247	11	7.68	2.61	4.58	27.15
Spartan6-LX150T	5.30	248	12	3.34	1.47	2.17	15.63

Table 16. Total Power for Digital ESN on Different FPGA Boards and Mixed Signal Design using Device Mismatch

Design	Power (mW)
Virtex5-LX110T	135.36
Virtex6-LX550TL	47.76
Spartan6-LX150T	26.7
Analog ESN with synapses using device mismatch	0.20E-3

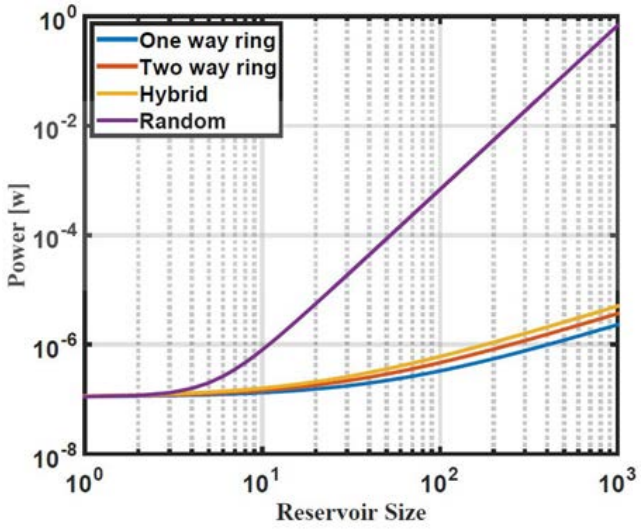

Figure 63. Power versus Scaling Reservoir Size for Different Digital ESN Topologies

4.2.3 Digital LSM. The reconfigurable digital LSM implementation was tested on epileptic seizure detection and user identification. In the epileptic seizure detection, an EEG signal was filtered into 4 distinct frequency ranges and then passed into the LSM. The LSM was able to achieve around 85% accuracy as shown in the confusion plots in Figure 64. The second task was user identification from walking patterns. In this task the data was information from an accelerometer kept in an individual breast pocket which recorded their movement while walking. For this case the LSM was able to achieve 98% accuracy shown in the confusion plots in Figure 65.

Figure 64. Digital LSM Accuracy for EEG Seizure Detection

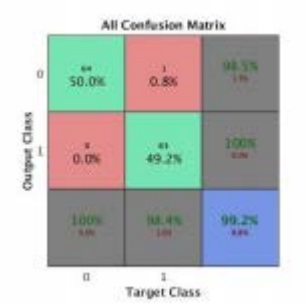

Figure 65. Digital LSM Accuracy for User Identification

Another important analysis of the digital LSM was using the separation property and Lyapunov exponent to evaluate the LSM's behavior. For the seizure detection task, the LSM had a separation of 0.1028 and for the user identification task the separation was 0.2945. The 3x larger separation value for the user identification task corresponds to the increased performance. Figure 66 shows a boxplot of the Lyapunov exponent for the LSM for both tasks. It can be observed that both tasks had a positive Lyapunov exponent which meant the liquid is operating in the chaotic region. This means a small change in the input pattern results in a significant change in the reservoir state.

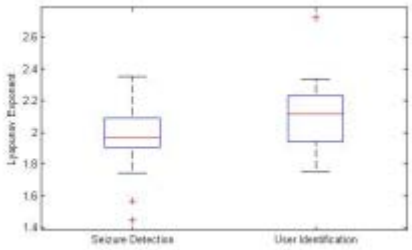

Figure 66. Lyapunov Exponent for Seizure Detection and User Identification

The power and area for the digital LSM were measured to compare to other designs and network topologies. The measurements are for a 60-neuron reservoir and exclude the output layer. For the LSM design the total FPGA resource utilization on a Xilinx Virtex7 XC7VX980T FPGA Board and is given in Table 17. The power and area of the liquid layer in the design for TSMC 65nm technology is given in Table 18.

Table 17. Digital LSM FPGA Resource Utilization for Xilinx Virtex7 XC7VX980T FPGA Board (60 Neurons)

	Used Resources	Available Resources	Utilization %
Number of Slice Registers	34,586	1,224,000	2
Number of Slice LUT's	32,455	612,000	5
Number of Occupied Slices	11.901	153,000	7
Number of LUT FF pairs used	39,825	-	-

Table 18. Area and Power of LSM Architecture for 65nm Technology

	TSMC 65nm
Total Area (mm^2)	2
Dynamic Power (mW)	55.61
Static Power μW	69.61
Total Power (mW)	55.68

4.3 Stochastic Architectures

4.3.1 Stochastic ELM. The proposed stochastic no-prop MLP using a threshold activation function was tested on binary signal regeneration and the Salinas Valley dataset. The goal of the binary signal restoration was to restore the integrity of a noisy signal and the network was able to achieve 92% accuracy with 32 nodes and 100% accuracy with 64 nodes as shown in Figure 67. The Salinas Valley dataset is a hyperspectral imaging dataset and for a 5-class problem the network was able to achieve around 62% accuracy as seen in Figure 68.

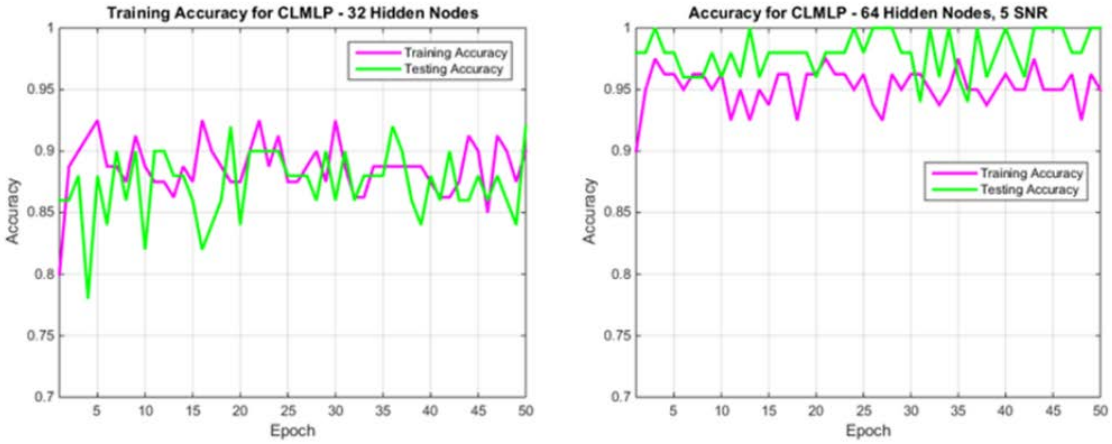

Figure 67. Stochastic ELM Accuracy for Signal Integrity

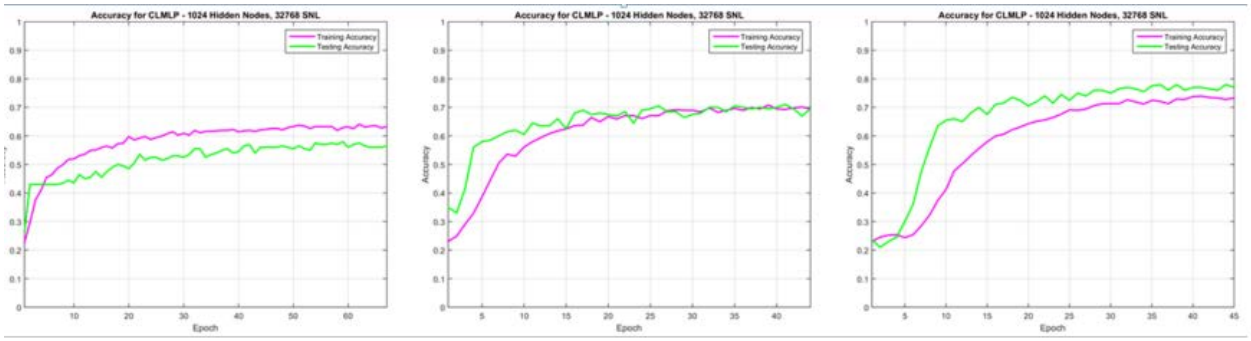

Figure 68. Stochastic ELM Accuracy for Salinas's Valley Dataset for an ELM with 1024 Hidden Nodes and a Stochastic Bit Length of 32768 Tested with Increasing Learning Rates (In Each run the Learning Rate was Increased by a Factor of Two)

The stochastic ELM implementation showed a significant speed up on a GPU compared to a Central Processing Unit (CPU) based implementation. For the adder and multiplier, there was a linear increase in speedup as the stochastic bit length increased up to a speedup of around 225x for an 8000 bit stochastic vector. The stochastic number generator showed a significant speedup, however the improvement in performance shows a logarithmic scaling with the stochastic bit length reaching a speedup of around 75x for an 8000 bit stochastic vector. The speedup of these three components is shown in Figure 69.

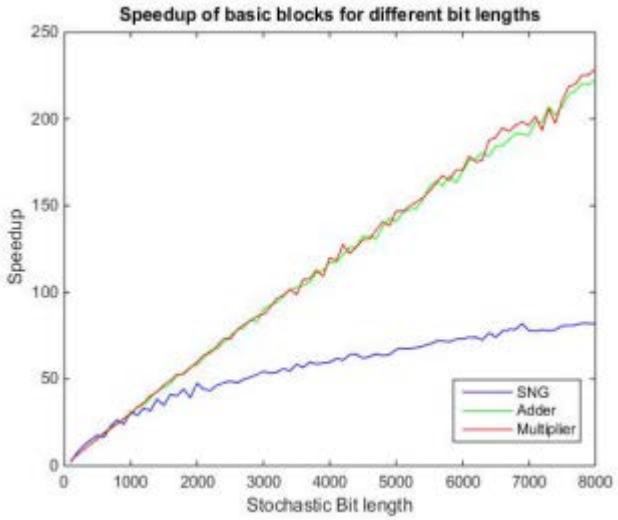

Figure 69. Speedup of Different Stochastic Components Between a GPU and a CPU

The stochastic ELM achieved an accuracy of 92% on the MNIST dataset and 87.5% on the orthopedic dataset (Figure 70 and Figure 71 respectively). The GPU implementation of the network achieved up to a speedup of 18.86x compared to a CPU as listed in Table 19 for MNIST [28] and up to 105.57x for the orthopedic dataset listed in Table 20.

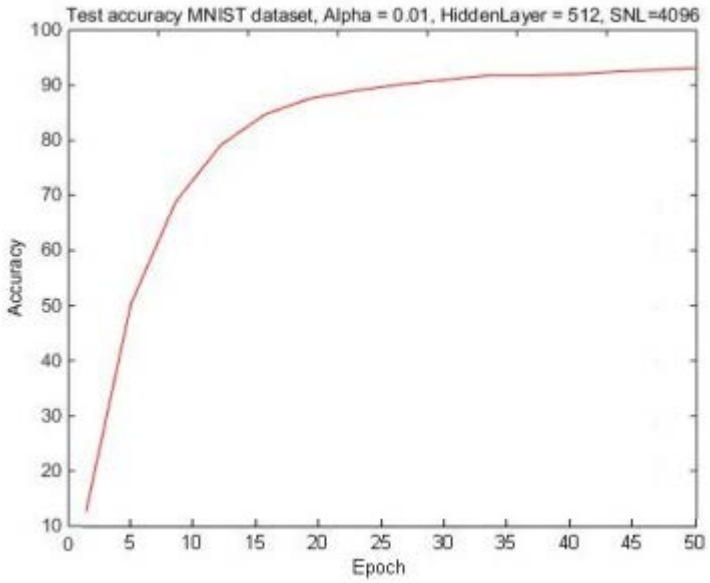

Figure 70. Accuracy of Stochastic ELM on MNIST Dataset

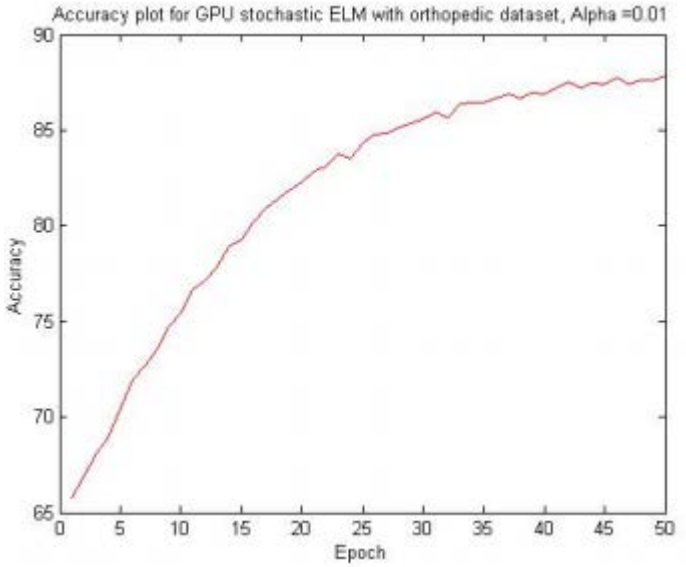

Figure 71. Accuracy of Stochastic ELM on Orthopedic Dataset

Table 19. Speedup of GPU-based Stochastic ELM vs. a CPU Implementation on the MNIST Dataset

SNL	CPU (mins)	GeForce GTX 480s (mins)	Speedup
2^{11}	1842.34	132.3	13.93
2^{12}	3661.42	204.2	17.93
2^{13}	7242.76	384	18.86

Table 20. Speedup of GPU-based Stochastic ELM vs. a CPU Implementation on the Orthopedic Dataset

SNL	CPU (mins)	GeForce GTX 1050 Ti (mins)	Speedup
2^{11}	1842.34	51.2	35.99
2^{12}	3661.42	60.4	60.61
2^{13}	7242.76	68.8	105.57

4.3.2 Stochastic ESN. The stochastic ESN was implemented on an EEG seizure detection dataset [15] and achieved an accuracy of 72.5% (Figure 72) and a speedup of up to 45.46x overs a CPU (Table 21).

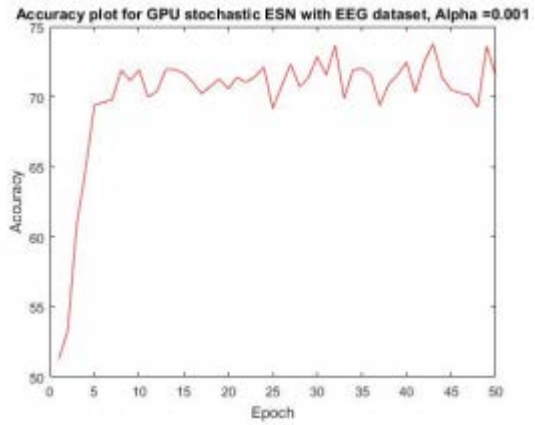

Figure 72. Accuracy of Stochastic ESN on EEG Seizure Detection Dataset

Table 21. Speedup of a GPU-based Stochastic ESN vs. a CPU Implementation on the EEG Seizure Detection Dataset

SNL	CPU (mins)	GeForce GTX 1050 Ti (mins)	Speedup
2^{11}	4211.486	177.55	23.72
2^{12}	9898.15	287.93	34.37
2^{13}	18560.23	418.2	45.46

4.4 Neuromemristive Architectures

4.4.1 Ziksa. Ziksa was verified behaviorally in cadence using GDPK 45nm technology node. Figure xx shows the state change of 4 memristors in a 2x2 crossbar and the potential applied across them by Ziksa. Ziksa was then used to train a three-layer feed-forward neural network trained to identify anomalies in patient data to screen for breast cancer. The dataset used in this application was the Wisconsin breast cancer dataset. In order to understand the reliability of Ziksa with the device variability of memristors, data of writing to a memristor was obtained from TiN-TiOx-TaTiN devices developed in the Microsystems and Engineering Sciences Application at Sandia National Labs. From the data our device resistance range was constrained from 600Ω to 9000Ω and the variance in the resistance change could be modeled with a Gaussian distribution with zero mean and a standard deviation of 54%. Under these constraints Ziksa was able to train the network up to 92% accuracy compared to the 94% using stochastic gradient descent and 98% using back-propagation as shown in Figure 73.

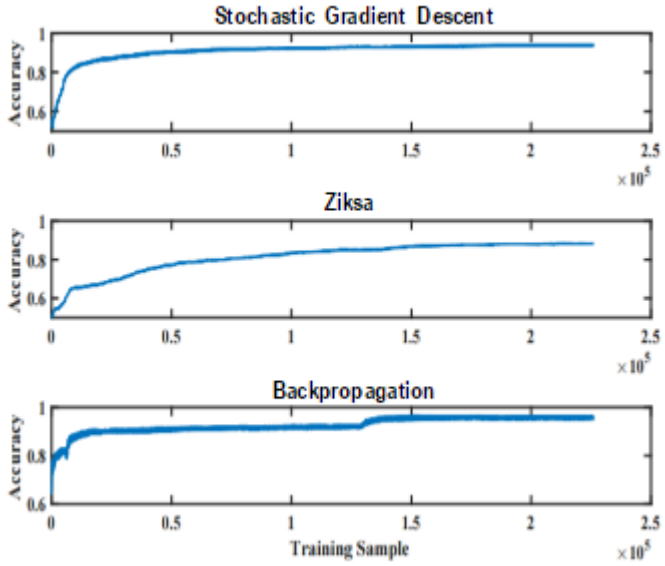

Figure 73. Ziksa Training Accuracy for Wisconsin Breast Cancer Dataset Compared to Stochastic Gradient Descent and Backpropagation

Ziksa was demonstrated to implement unsupervised STDP using the simplified STDP rule where no loss in performance was observed between the original rule and the simplified rule. The best performance of the network was 65% on MNIST.

Figure ***74*** shows a comparison between the learned weights of the original STDP rule which did not use the sign function and the proposed simplification.

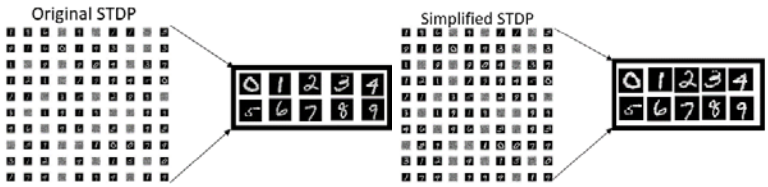

Figure 74. Comparison Between Learned Weights of Original STDP Rule and Simplified STDP Rule

4.4.2 Figure 75 shows the initialization of the memristor crossbar to implement a random weight matrix. Initially, all the memristors are written to their highest resistance state, then memristors are randomly decreased to cause an increase in the weight. Figure 76 demonstrates a 1x4 row of memristors with four different inputs being sent into the current amplifier and from there to spiking neuron. As can be observed, the inputs cause the potential across the capacitance to charge to -0.3V and the spiking neuron emits a spike.

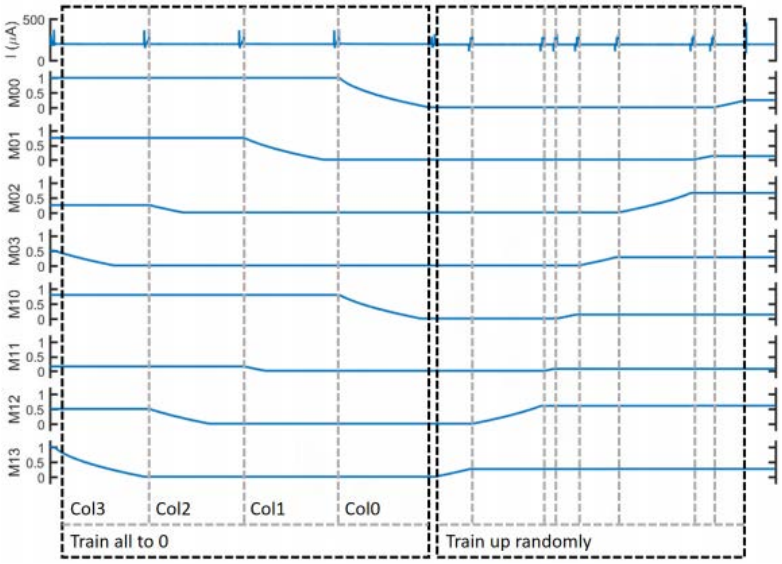

Figure 75. Random Initialization of Memristor Crossbar

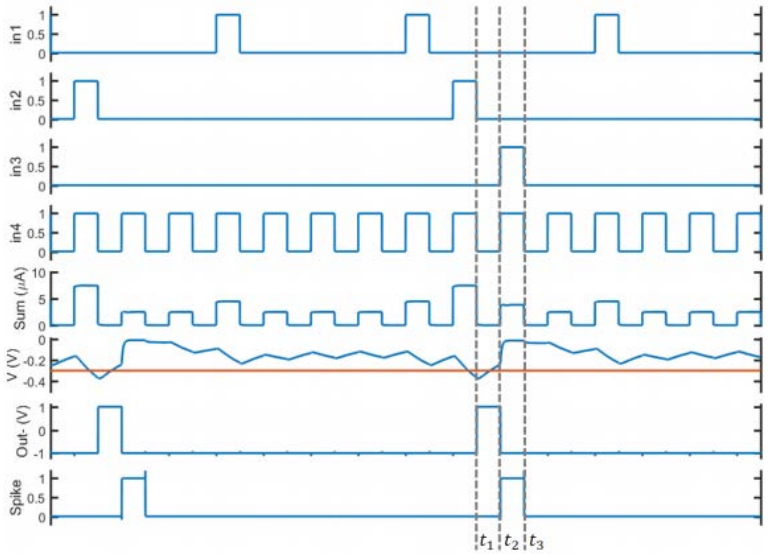

Figure 76. Verification of Spiking Behavior of Neuromemristive LSM Reservoir Layer

The current amplifier was tested with a load resistance of 100 kΩ and a capacitance of 500 fF to match the spiking neuron parameters. The resulting nominal, 45 corners, and Monte Carlo results are shown in Table 22. The transistor, resistor, and capacitor sizing's are listed in Table 23.

Table 22. Device Sizing used in Current Amplifier Design

Device	W (m)	L (m)	Multiplier	Area (m^2)	Percentage (%)
M1	8u	1u	1	10.2p	2.15
M2	8u	1u	1	10.2p	2.15
M3	12u	1u	1	15.4p	3.22
M4	12u	1u	1	15.4p	3.22
M5	15u	2u	1	34.2p	7.18
M6	15u	2u	1	34.2p	7.18
M7	10u	750n	1	1.03p	2.16
M8	10u	2u	1	2.28	4.792
M9	10u	300n	1	5.8p	1.22
M10	10u	300n	1	5.8p	1.22
M11	10u	200n	1	4.8p	1.01
M12	10u	200n	1	4.8p	1.01
M13	5u	2.5u	1	13.9p	2.92
M14	5u	2.5u	1	13.9p	2.92
M15	10u	2u	1	22.8p	4.79
M16	10u	2u	1	22.8p	4.79
Rz	500n	8.13u	1	4.6p	0.853
Cc	5u	5u	9	225p	47.2

Table 23. The Resulting PVT and Monte Carlo Simulation Results for the Current Amplifier

Metric	Specs	Nom	45 Corners	Monte Carlo
Power (μW)	Minimize	78.47	[63.07, 96.37]	[74.9, 83.75]
Gain (A/A)	-[0.95, 1.05]	-1.01	-[1.004, 1.049]	-[1.003, 1.022]
Phase Margin (˚)	≥ 60	75.66	[61.53, 88.65]	[64.79, 96.62]
Bandwidth (MHz)	≥ 20	87.5	[43.18, 100.2]	[56.03, 72.77]
ICMR max (mV)	≥ 200	364.2	[249.2, 476.2]	[329.4, 409.2]
ICMR min (mV)	≤ -200	-355.1	-[510.1, 205.1]	-[385.1, 320.1]
Input Offset (mV)	≤ 10	514.9u	[372.9u, 880.9u]	[2.087u, 4.711m]
Pos Slewrate (uA/ns)	≥ 2.5	2.849	[2.759, 2.895]	[2.756, 2.891]
Neg Slewrate (uA/ns)	≤ -2.5	-2.654	-[2.737, 2.568]	-[2.62, 2.528]
Intersect (nA)	[-1uA, 1uA]	-231.1	[-388.1, -127.8]	[-423.1, 13.15]
Curr In Offset (nA)	≤ 100	7.952	[4.297, 41.76]	[0.2463, 88.65]
Curr Out Diff (uA)	≤ 1.5	1.101	[0.891, 1.289]	[1.057, 1.209]

The sizing for the devices in the comparator are given in Table 24 and the PVT and Monte Carlo analysis is listed in Table 25.

Table 24. Device Sizing used to Implement Double Tail Latched Comparator

Device	W (m)	L (m)	Multiplier	Area (m^2)	Percentage (%)
M1	10u	150n	1	2.3p	32
M2	10u	150n	1	2.3p	32
M3	360n	180n	1	0.166p	1.23
M4	360n	180n	1	0.166p	1.23
M5	2u	150n	1	0.86p	6.4
M6	360n	180n	2	0.331p	2.47
M7	360n	180n	2	0.331p	2.47
M8	360n	180n	2	0.331p	2.47
M9	360n	180n	2	0.331p	2.47
M10	360n	180n	2	0.331p	2.47
M11	720n	180n	2	0.662p	4.93
M12	720n	180n	2	0.662p	4.93
M13	360n	180n	2	0.331p	2.47
M14	360n	180n	1	0.166p	1.23
M15	360n	180n	1	0.166p	1.23

Table 25. PVT and Monte Carlo Analysis of Double Tail Latched Comparator

Metric	Spec	Nominal	45 Corners	Monte Carlo
Power (μW)	Minimize	145.6	[105.6, 196.9]	[145.9, 167.7]
Clock Rate (GHz)	1.2	---	---	---
Input Drive (mV)	≤ 20	---	---	---
Offset Voltage (mV)	≤ 10	47.86n	[-47.68n, 47.68n]	[-9.006, 6.065]
Kickback (mV)	≤ 10	7.481	[6.696, 8.128]	[7.703, 7.611]
Hysteresis (mV)	≤ 5	1.504	[0.366, 3.396]	[0.418, 2.539]

For both the voltage and current reference the primary design concern was the accuracy of the circuit which was measured by the parts per million (PPM) which was measured by sweeping the temperature of the system. The device sizing for the Banba voltage reference are listed in Table 26 and the PVT and Monte Carlo simulations are listed in

Table **27**. The device sizing for the current reference is given in Table 28 while the PVT and Monte Carlo analysis is given in Table 29.

Table 26. Device Sizing for Banba Bandgap Voltage Reference

Device	W (m)	L (m)	Multiplier	Area (m^2)	Percentage (%)
M1	10u	10u	1	103p	0.129
M2	10u	10u	1	103p	0.129
M3	10u	10u	1	103p	0.129
M4	8u	800n	1	8.65p	0.011
M5	8u	800n	1	8.65p	0.011
M6	10u	10u	1	103p	0.129
M7	10u	10u	1	103p	0.129
M8	10u	5u	1	52.8p	0.067
M9	10u	10u	1	103p	0.129
M10	10u	5u	1	52.8p	0.067
M11	720n	360n	1	0.461p	0.006
M12	720n	360n	1	0.461p	0.006
M13	720n	360n	1	0.461p	0.006
M14	720n	360n	1	0.461p	0.006
M15	720n	360n	1	0.461p	0.006
Q	17u	17u	1	289p	0.364
Qn	17u	17u	1	2.31n	2.91
R1	8.4u	52.1u	67	29.3n	36.9
R2	8.4u	52.1u	67	29.3n	36.9
R3	8.4u	2.56u	67	1.44n	1.81
R4	8.4u	28.6u	67	16.1n	20.2

Table 27. PVT and Monte Carlo Analysis for Banba Bandgap Voltage Reference

Metric	Spec	Nominal	15 Corners	Monte Carlo
Power (μW)	Minimize	57.85	[43.63, 84.22]	[51.2,65.81]
Vref (mV)	700	700	[692.2, 712.5]	[660, 746.1]
PPM (PPM/˚C)	≤ 200	9.344	[9.081, 50.68]	[6.181, 102.3]
Range (mV)	---	0.654	[0.636, 3.611]	[0.448, 6.869]

Table 28. Device Sizing for Current Reference

Device	W (m)	L (m)	Multiplier	Area (m^2)	Percentage (%)
M1	8u	1u	1	10.2p	0.123
M2	8u	1u	1	10.2p	0.123
M3	3.6u	1.8u	1	7.49p	0.09
M4	3.6u	1.8u	1	7.49p	0.09
M5	10u	2u	2	45.6p	0.549
M6	3.6u	1.8u	1	7.49p	0.09
M7	3.6u	1.8u	1	7.49p	0.09
M8	3.6u	1.8u	1	7.49p	0.09
M9	10u	2u	1	22.8p	0.275
R1	8.4u	9.73u	100	8.17n	98.5

Table 29. PVT and Monte Carlo Analysis for Current Reference

Metric	Spec	Nominal	15 Corners	Monte Carlo
Power (μW)	Minimize	79.08	[58.81, 109]	[72.25, 92.03]
Iref (μA)	10	10	[8.295, 12.5]	[9.134, 11.74]
PPM (PPM/˚C)	≤ 200	37.67	[5.891, 95.1]	[6.531, 130.9]
Range (nA)	---	37.68	[5.312, 78.89]	[6.769, 127.3]

The total area of the proposed design is 91.9nm^2 shown in Table 30.

Table 30. Total Area Consumption of a 2x4 Reservoir Layer

Component	Area (m^2)	Percentage (%)
ZTC	15.6E-12	0.00169
Current Amplifier	953E-12	1.04
Comparator	26.9E-12	0.00292
Spiking Neuron Circuitry	3.12E-9	3.39
Voltage Reference	79.5E-9	86.5
Current Reference	8.3E-9	9.03
Total Area	91.9E-9	---

4.4.3 Robustness of neuromemristive LSM. Due to device characteristics memristor devices in the analog domain have a substantial amount of read and write noise associated with their operation. In order to implement RC architectures in neuromemristive systems it is important for them to be robust to sources of internal or external noise. We studied the impact of write and read noise on an LSM in order to verify the robustness of the algorithm using a synthetic dataset of spike trains, as well as the TIMIT speech corpus, and Spoken Arabic Digit Recognition dataset [27].

To simulate write noise in the network we considered three cases; the first was where each synapse could only be trained to a maximum or minimum resistance state that varied from a set value by a Gaussian

distribution with zero mean and a standard deviation of up to 10%. The second case was done by adding some Gaussian noise to the resistance update with a zero mean and a standard deviation up to 30%. Lastly memristors were stuck at their initial resistance state, the number of memristors fixed in each simulation ranged from 1% of the total memristors in the crossbar to 30%. In all three simulations we observed no impact from the noise added to the network on the LSMs performance on the synthetic dataset.

In order to simulate noise when reading we considered two situations in the memristor which could cause an incorrect read. The first case was when the resistance of the memristor was not at the exact resistance that is expected. To simulate this a Gaussian distribution of noise with zero mean and a standard deviation of up to 30% was added to the resistance of each device. This had no impact on performance of the LSM. The second case was when Gaussian noise was added to every read of a memristor with a zero mean and a standard deviation of up to 50%. These results are shown in Figure 77 where it can be observed that until the standard deviation reaches 25%, the LSMs performance suffers negligible impacts.

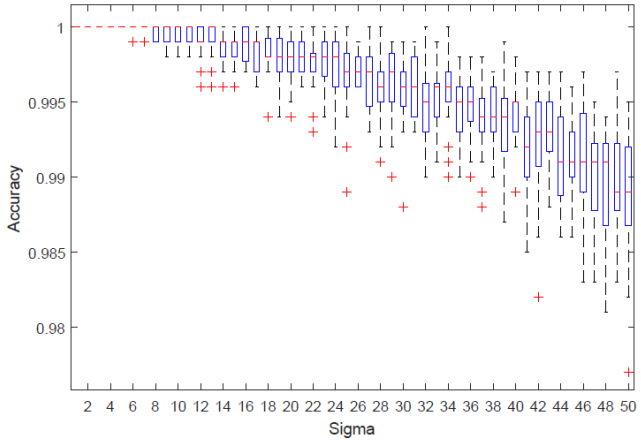

Figure 77. Increasing Synaptic Noise vs Accuracy

The last part of this study was a combined effect of all sources of noise in the liquid. In a spiking neural network, the impact of noise will manifest itself in the firing patterns of the neurons either causing the neurons to spike early by increasing the amount of synaptic current they integrate over time or in the opposite case delaying the firing of the neuron. In order to simulate this at every state random neurons were chosen to be impacted by noise where they would give the incorrect output so neurons that were supposed to spike were delayed and set back under the threshold while neurons that weren't spiking would have their membrane potentials boosted above the threshold. The number of neurons subjected to this noise was tested from 1% of neurons in the liquid to 30%. The results for the synthetic dataset are shown in Figure 78 which shows roughly a 2% decrease in accuracy when 14% of the neurons in the liquid are failing when the input space consisted of five features. In order to understand the impact of the size of the feature space, the same experiment was repeated for inputs with 1 feature to 13 features and the results are shown in Figure 79. The key observation from this plot is the increasing robustness to the number of features and the qualitative leap in performance after the input space has more than 4 features. Finally, the cumulative noise scenario was implemented on the LSM on two real-world datasets. The first was the TIMIT Speech Corpus where the task was to recognize four different phonemes: "e", "eh", "er", and "u" while the second task was to recognize the spoken Arabic digit which had 10 classes for 0 through 9. The results are shown in Figure 80 and Figure 81 respectively. An interesting observation is the trend in these simulations showed no

correlation between the noise injected into the network and the performance. One possible explanation for this is that there is 13 features in these two datasets and when observing Figure 79, the 13 feature scenario with the synthetic dataset shows very high robustness however there is still a noticeable gradual decline in performance after around 25% of the neurons are subjected to noise. Another potential cause of this behavior for the real-world tasks is that the noise in the data is causes such a large variance in similar inputs that the changes induced by the failing neurons is not having an impact on performance.

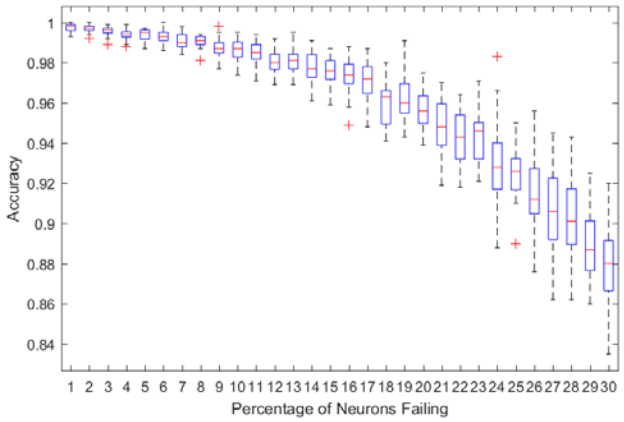

Figure 78. Increasing Percentage of Failing Neurons for Synthetic Dataset

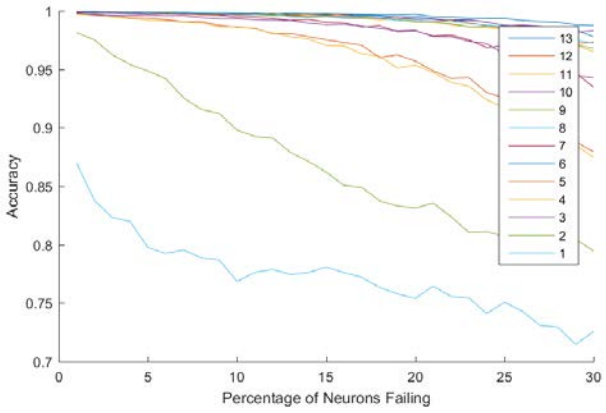

Figure 79. Increasing Percentage of Failing Neurons for Synthetic Dataset for Different Input Feature Sizes

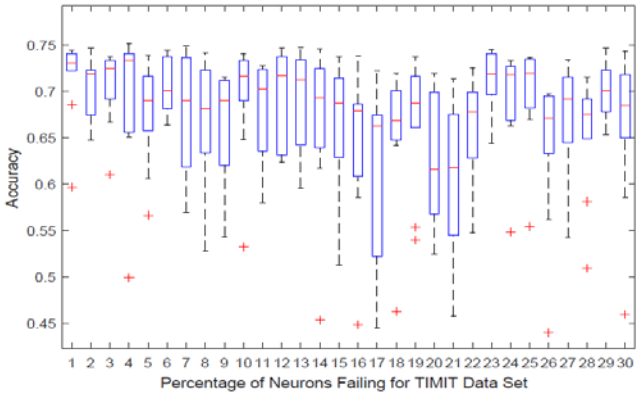

Figure 80. Increasing Failing Neurons for TIMIT Dataset

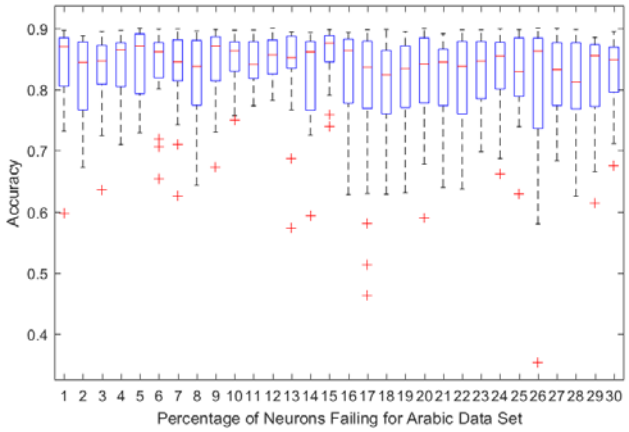

Figure 81. Increasing Failing Neurons for Arabic Digit Dataset

Lastly, to consider the performance of a neuromemristive based deep-LSM we study the robustness by modelling read and write noise as a Gaussian process, the memory and energy consumption of the deep-LSM with respect to a vanilla LSM and LSTM, and the number of weight updates and operations needed by each network reported in Table ***31***.

Table 31. Performance of deep-LSM with 3 Hidden Layers of 500 neurons and 2 WTA Layers with 50 Neurons when Gaussian Read and Write Noise is Added to Model Memristive Device Noise.

Model (3 Layers)	Accuracy	Standard Deviation
deep-LSM	82.9	6.78
deep-LSM (with noise)	81.92	10.08

The deep-LSM showed similar robustness to speech recognition [15] with a 1% degradation in performance.

4.4.4 Semi-trained crossbars. The proposed semi-trained crossbar approach was tested for an ELM on a set of spatial datasets listed in Table 32 using a signed delta rue for training.

Table 32. Performance of ELM using Semi-trained Memristive Crossbar Compared to Baseline Software and VLSI Models. The Proposed ELM used a Much Lower Number of Neurons to Achieve Comparable Performance on all Benchmark Applications

DataSet	ELM, η = 1000 [Huang et al. 2012][a]	VLSI ELM, η = 120 [Yao and Basu 2017][b]	This work, η = variable
Diabetes	77.95%	77.09%	72.73% (η=65)
Australian Credit	87.89%	87.89%	82.16% (η=40)
Iris	96.04%	-	84.66% (η=20)
MNIST	-	-	93.53% (η=180)[c]

[a] Software implementation of ELM.
[b] Non-memristive mixed signal implementation of ELM.
[c] All MNIST images are preprocessed with HOG feature descriptor, described in [Zyarah and Kudithipudi 2017].

We also studied the scalability in terms of transistor count for various crossbar sizes shown in Figure 82 and the power consumption to perform on-device learning reported in Table 33.

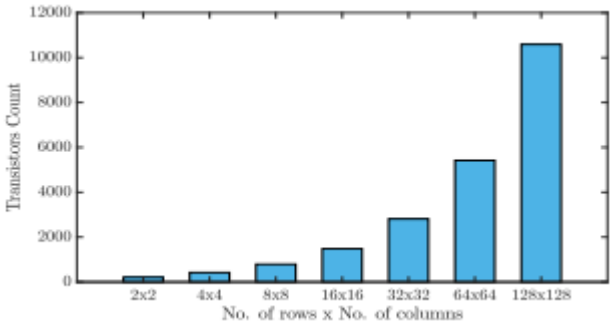

Figure 82. Scalability in Terms of Transistor Count for Different Crossbar Sizes Using the Semi-Trained On-device Learning Approach with Ziksa

Table 33. Power Consumption of On-device Learning with Ziksa for Implementing the Signed Delta Update Rule.

Architecture	Digital logic circuit	Crossbar	Total
Fully-trained two-crossbar	1.45 µW	39.53 µW	40.98 µW
Semi-trained two-crossbar	1.22 µW	20.85 µW	22.07 µW
Fully-trained one-crossbar	1.52 µW	41.01 µW	42.53 µW
Semi-trained one-crossbar	1.15 µW	41.01 µW	42.16 µW

5.0 Semi-Random Deep Neural Networks

Semi-random Deep Neural Networks (DNNs) are feedforward architectures that retain random weights in the hidden layers while training on only the output layer. Unlike the commonly used gradient descent approach to training neural networks, semi- random DNNs make use of solving the least squares approximation for the output weights in a single training step. The least squares approximation is defined by (26)

$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{W}_{\mathbf{out}} \tag{26}$$

where H is the output of the final hidden layer, $\mathbf{W}_{\mathbf{out}}$ is the hidden layer to output weight matrix, and $\hat{\mathbf{y}}$ is the predicted output. To solve for the weights, $\mathbf{W}_{\mathbf{out}}$, the inverse of $\mathbf{H}$ needs to be computed. To take the inverse of a matrix the matrix must be square. however, it is rare for this to occur given most data. Therefore, the Moore-Penrose pseudoinverse is used to generalize the inverse of a matrix as shown in (27).

$$\mathbf{H}^{\dagger} = (\mathbf{H}^{T}\mathbf{H} + \lambda \mathrm{I})^{-1}\mathbf{H}^{T\ldots} \tag{27}$$

In (26) I is an identity matrix that is multiplied by a λ variable which acts as a regularization term. By using (26) and (27) the weight matrix $\mathbf{W}_{\mathbf{out}}$ in semi-random DNNs can be calculated using (28), where $\mathbf{y}$ is the ground truth labels.

$$\mathbf{W}_{\mathbf{out}} = \mathbf{H}^{\dagger}\mathbf{y} \tag{28}$$

Typically, a vanilla ELM is a shallow network consisting of 1 hidden fully connected layer that contains a large number of neurons and an output fully connected layer. As the number of hidden neurons increases, the least squares approximation advances closer to the ideal solution yielding a high accuracy model. The first architecture we employ is the Convolutional ELM (CELM) feature extractor for the input and then processes the output through the fully connected layers of the ELM.

In addition to the convolutional layer we adopt an input-to-output connection. This connection was taken as inspiration from both the ResNet [37] and the RVFL[36] networks. In the ResNet architecture, residual blocks allowed for deep networks to avoid the problem of vanishing gradients through the use of skipped connections that retain residual mappings of previous layers. This skipped connection contains important information that is lost across deeper layers in DNNs. This technique, however, is utilized for DNNs that contained tens to hundreds of layers. In this work we experiment with semi-random DNNs that contain fewer layers than the ResNet architectures, but we find that skipped connections still benefit the performance of our networks.

We additionally look to the Random Vector Functional Link (RVFL) network, which is a vanilla ELM that utilizes a skipped input-to-output connection as this empirically increased performance on shallow fully connected networks. As this connection is added to the network, the $\mathbf{H}$ matrix in (25) is modified by concatenating the input onto the matrix. The Moore-Penrose pseudoinverse is then performed on the new $\mathbf{H}$ matrix using (26). We test the connection on the

output layer, also known as the second fully connected layer in this instance, and the first fully connected layer for the CELM architecture. An illustration of the CELM architecture can be seen in Figure 83.

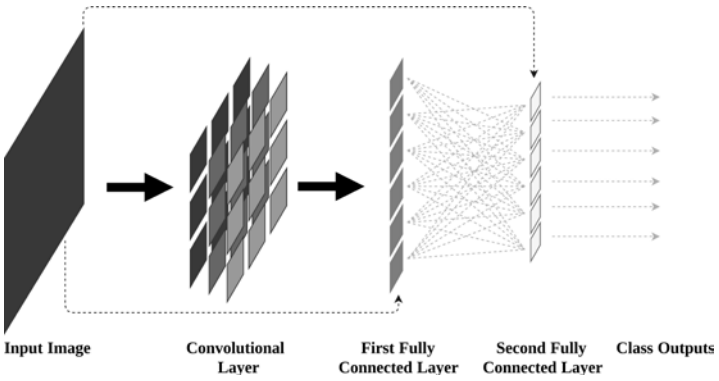

Figure 83. Illustration of the CELM Architecture. A Convolutional Layer is used as a Feature Extractor that is Passed to Fully Connected Layers. Skipped Connections from the Input to Intermediate and/or Output Layers in the Network are used to Observe How this Connection Influences Performance.

Figure 83 depicts the CELM architecture with the single convolutional layer and ELM's fully connected layers. Dashed lines to the first and second fully connected layers are shown here as we experiment with the usage of skipped connections to either an intermediate fully connected layer, the last fully connected layer, or to both layers.

Another architecture that utilizes the ideas of the RVFL and ELM is the Convolutional RVFL (CRVFL) [38]. The CRVFL consists of a convolutional layer, pooling layer, normalization layer, and a single fully connected layer. An additional direct connection from the input to the last fully connected layer was also used in this architecture. The authors in [38] showed that the CRVFL was able to perform effectively for visual tracking while benefiting from fast training times. Therefore, we use this architecture to evaluate its performance on real-time image classification on MSTAR. In this work we modify the CRVFL slightly by adding in an additional fully connected layer. With an additional fully connected layer, we refer to this architecture as CRVFL-FC, shown in Figure 84.

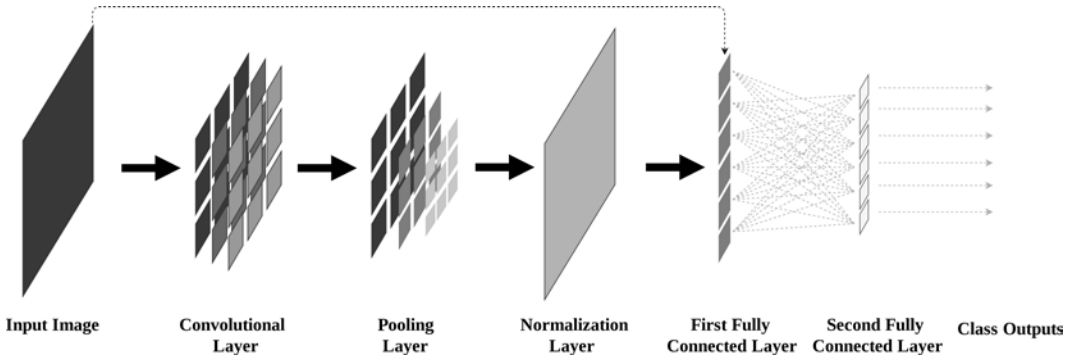

Figure 84. Illustration of the CRVFL-FC Architecture.

A convolutional layer is used to extract features that are then sent to a pooling layer to reduce the spatial size of the feature maps. A normalization layer is used to decrease covariance shift and then the output is fed to the last two fully connected layers. A skipped connection is used from the input to the first fully connected layer as this was used in the original architecture.

We leave the input to the first fully connected layer on for our experiments to assess the network's performance. Training with the additional fully connected layers uses 16GB RAM.

To compare against the semi-random DNNs, we use conventional DNN architectures. The first architecture is a simple shallow convolutional neural network (CNN) consisting of a single convolutional layer and a single fully connected layer. Although this architecture isn't deep, it acts as baseline to evaluate whether the DNN architectures are overfitting to the data and if a deep architecture is needed for this task. The reason why overfitting is suspected to occur is due to the MSTAR dataset containing fewer than 10,000 samples.

The network takes in 224x224 RGB images that are reduced to 1,000 weights at the last fully connected layer. The first architecture, AlexNet, is an architecture originally introduced in 2012 by Alex Krizhevsky [39]. It consists of 5 convolutional layers, 3 max pooling layers, and 3 fully connected layers. Similarly, to other deep architectures the network is trained using gradient descent along with backpropagation. This conventional DNN architecture is used for our experiments, as it was one of the first DNNs to show that deep architecture achieves greater performance on vision tasks when compared to shallow networks.

The second architecture used in our experiments is ResNet. ResNet was created at a time when DNNs had diminishing returns when using more layers. The introduction of residual blocks in ResNet allowed for DNNs to increase their number of layers and in turn increase their performance. These residual blocks contained important input-to-intermediate layer connections that allowed for networks to extract more features from the input. Therefore, in this work we use a ResNet with 101 layers.

The last architecture that is used in our experiments is PyramidNet [xx], more specifically the additive PyramidNet with a depth of 110 and a widening factor of 64. PyramidNet consists of multiple residual blocks and convolutional layers with 3 3 kernel sizes. Residual blocks are shown to decrease classification accuracy if they were down sampled as this led to a sharp increase in feature map dimensions. This problem is avoided in the PyramidNet architecture as the number of features instead gradually increases. The number of output filters are accumulated iteratively with each successive convolutional layer. This leads to a pyramid structure, where the network takes the input and incrementally increases the number of output filters until the last 2 layers. The last 2 layers consist of an average pooling layer and a fully connected layer respectively. We summarize which conventional DNNs are chosen in Table 34. In the table, number is denoted as 'No.' and in the total number of parameters million is denoted with an 'M'.

Table 34. Summary of Chosen Conventional DNNs

	Shallow CNN	AlexNet	ResNet	PyramidNet
No. of Conv. Layers	1	5	99	108
No. of Fully Connected Layer	1	3	1	1
Filter Sizes	3	3, 5, 11	1, 3, 7	1, 3
Total No. of Parameters	125,898/251,786*	60M	44.5M	9M**

*If there are 16 filters from the convolutional layer then there are 125,610 parameters. If there are 32 filter then there are 251,210 parameters.
**For 200 layers and a widening factor of 300 the parameters would be closer to 62M but we use 108 layers for our experiments as it's closer to the ResNet model's layer count

In our experiments the CELM and CRVFL-FC architectures contain a convolutional layer with either 16 or 32 output filters at a kernel size of 3x3. Additionally, we vary the λ regularization value at 0.1, 0.01, and 0.001 to assess the network at different regularization rates. In the CRVFL-FC architecture the pooling layer is an average pooling layer with a kernel size of 2 × 2 and a stride of 2. A normalization layer with a size of 2, an $\alpha = 0.0625$, and a $\beta = 0.75$ is used before the first fully connected layer. Lastly, the first fully connected layer contains 13,328 neurons from the output of the batch normalization layer. As this number of inputs is too many to train, we train on the last output layer. For all semi-random DNNs we use 1000 hidden neurons for the last fully connected layer. In the CELM 3 fully connected layer case we use 10,000 hidden neurons for the 2nd fully connected layer. All weights are initialized with a Gaussian distribution of mean 0 and standard deviation of 0.05. The semi-random DNNs are averaged over 100 randomly shuffled runs of the data.

The performance of the semi-random DNNs and the conventional DNN architectures are reported in Table 35. We record the regularization rate (Reg. Rate), number of output filters (No. of Filters) in the first convolutional layer that achieved the highest test accuracy, with standard deviations, and number of trained parameters (No. of Params) relative to the MSTAR dataset. The memory usage, depicted in the table, is more specifically the average GPU memory usage during training. The input to fully connected layer skipped connections are denoted by 'Skip' to their respective fully connected layer. For example, the 'CELM Skip 1' architecture feeds the input image to the convolutional layer and additionally utilizes a connection from the input image to the first fully connected layer. The table shows that the semi-random DNNs achieve comparable accuracies to the conventional DNNs while decreasing training time by orders of magnitude. We also see that the additional use of a skipped connection from the input image to any of the fully connected layers shows an increase in performance. However, this skipped connection along with computing the least squares approximation leads to high memory usage. This is due to the semi-random DNNs requiring the entire dataset to be loaded in as a single batch and computed on over a single training step. As MSTAR is a small dataset this was able to be computed with our systems. Unfortunately, increasing the number of filters or

increasing the last fully connected layer's number of neurons to a count of over 1,000 led to memory errors. Therefore, the use of these semi-random DNNs over the conventional DNNs has a trade-off of heavy memory usage for rapid training times.

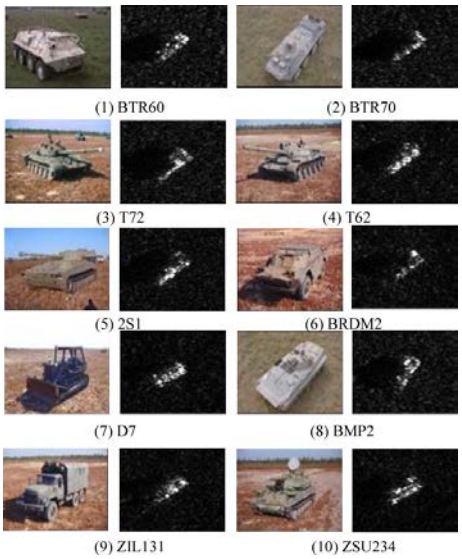

Figure 85. The MSTAR Dataset Containing 10 Unique Vehicle Classes

Table 35. MSTAR Results on the Semi-Random and Conventional DNN Architectures *

Architecture	Reg. Rate	No. of Filters	Accuracy (%) Train	Accuracy (%) Test	Training Time (Minutes)	Memory Usage (MiB)	No. of Params**
ELM	0.1	-	96.78 ± 0.25	94.82 ± 0.49	0.18	731	10,000
RVFL	0.1	-	98.62 ± 0.14	96.29 ± 0.36	0.18	837	17,840
CELM No Skip	0.01	32	95.98 ± 0.49	94.93 ± 0.67	0.18	3455	10,000
CELM Skip 1	0.01	32	96.77 ± 0.29	95.43 ± 0.59	0.19	3909	17,840
CELM Skip 2	0.01	32	98.32 ± 0.29	96.72 ± 0.40	0.15	3561	17,840
CELM Skip 1 & 2	0.1	16	98.31 ± 0.17	**97.02 ± 0.32**	0.19	3909	17,840
CELM Skip 3	0.01	16	97.45 ± 0.27	96.36 ± 0.38	0.19	5735	17,840
CRVFL-FC	0.01	32	96.28 ± 0.29	95.24 ± 0.42	0.18	4641	17,840
Shallow CNN	-	16	99.07 ± 0.05	**98.59 ± 0.39**	30.22	609	125,610
AlexNet	-	-	97.51 ± 0.20	98.12 ± 1.04	32.08	1359	36,051,658
ResNet	-	-	97.57 ± 0.58	98.24 ± 0.69	63.46	3383	42,514,378
PyramidNet	-	-	96.23 ± 0.58	98.17 ± 0.85	52.65	881	2,431,506

*We conduct all of our experiments on an Intel Core i5-6500 CPU @ 3.20 GHz, 16 GB RAM, with an NVIDIA Titan X GPU. The PyTorch[35] framework is used for image processing and to run all experiments on the GPU. We use Pytorch version 0.4.0.

**The number of parameters is reported relative to the 28 × 28 images and slightly modified architectures.

The results additionally show that the training accuracies for the conventional DNNs are marginally lower than the test accuracies. This can be attributed to a few reasons. When the MSTAR dataset is downsized to 28x28 images, we lose features but are kept with distinct features that

differentiate the classes. The dataset can also have small intrinsic variance where the training variance could be greater than the testing variance.

We see that the conventional DNNs achieve higher test accuracies than the semi-random DNNs, but the shallow CNN had the highest test accuracy. We evaluate if training the deep architectures for 150 epochs led to overfitting by plotting the models at training epochs of 50, 100, and 150 as seen in Figure. From the plot we can see that high-test accuracy is attained by these deep models at lower training epochs. Therefore, the model may be overfitting as the accuracy varies by a minimal amount at higher training epochs. To decrease the effects of overfitting in these models we use augmented images and evaluate on only the conventional DNNs as the semi-random DNNs have a memory bottleneck relative to the dataset size. We train for 50 epochs for these runs. The performance of the conventional DNNs on the augmented images can be seen in Table 36.

Table 36. Augmented MSTAR Results on the Conventional DNN Architectures

Architecture	No. of Filters	Accuracy (%) Train	Test	Training Time (Minutes)	Memory Usage (MiB)	No. of Params**
Shallow CNN	32	98.12 ± 0.11	99.14 ± 0.23	71.62	641	251,210
AlexNet	-	98.19 ± 0.10	99.52 ± 0.49	81.17	1757	36,051,658
ResNet	-	96.70 ± 1.41	99.31 ± 0.15	129.64	2685	42,514,378
PyramidNet	-	94.40 ± 1.00	98.86 ± 0.59	119.19	2199	2,431,506

*We conduct all of our experiments on an Intel Core i5-6500 CPU @ 3.20 GHz, 16 GB RAM, with an NVIDIA Titan X GPU. The PyTorch framework is used for image processing and to run all experiments on the GPU. We use Pytorch version 0.4.0.
**The number of parameters is reported relative to the 28 × 28 images and slightly modified architectures.

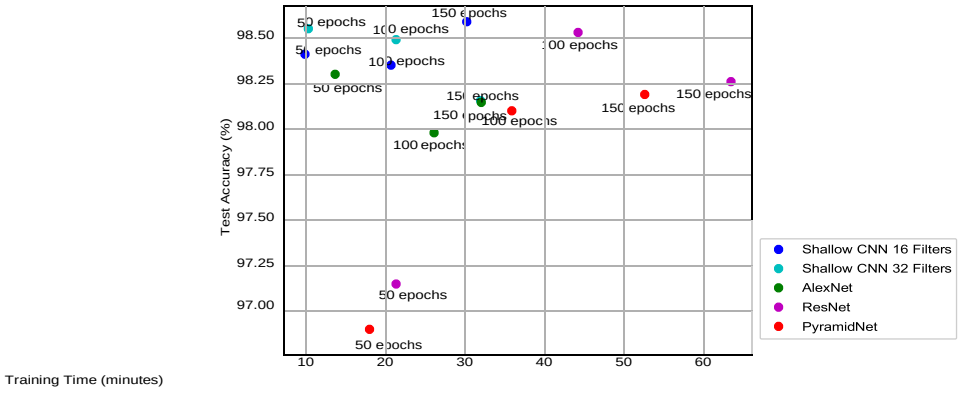

Figure 86. Plot of Accuracy vs Time of Conventional DNN Architectures at Various Training Epochs

Using the augmented images, it can be seen that the accuracy is still high but the models are concluded to not overfit this data in comparison to the original MSTAR dataset. The augmented images contain various degree shifts so the conventional DNNs are also not generalizing to the specific 15 and 17-degrees from the original dataset but rather the rotation has little effect in hindering the network's performance. We hypothesize that the test accuracy being lower is due to the intrinsic small variance between images, where the training variance seems larger than the testing variance. Additional training time, such as with 150 epochs, showed that the training accuracies were closer to the testing accuracy. We illustrate a similar plot to Figure 86 for the semi-random DNNs, as shown in Figure 87, for each of the semi-random DNN models' training time and accuracy.

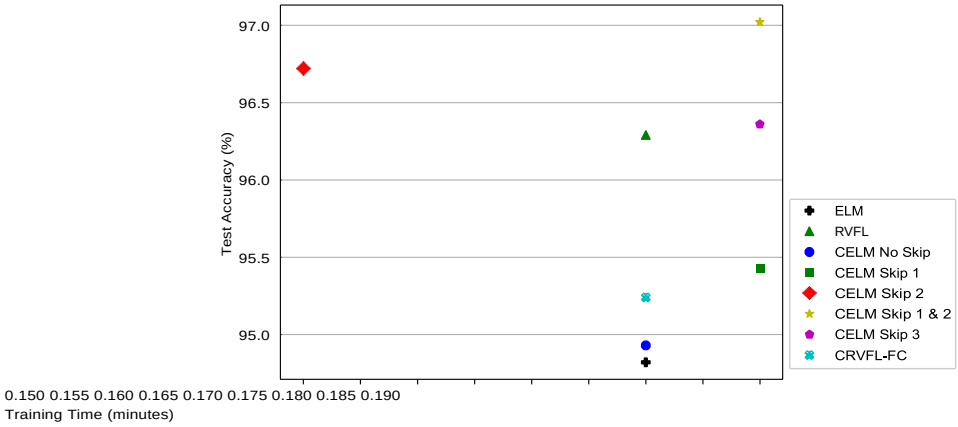

Figure 87. Plot Depicting Accuracy vs Training Time for the Semi-random DNNs with Varying Topologies

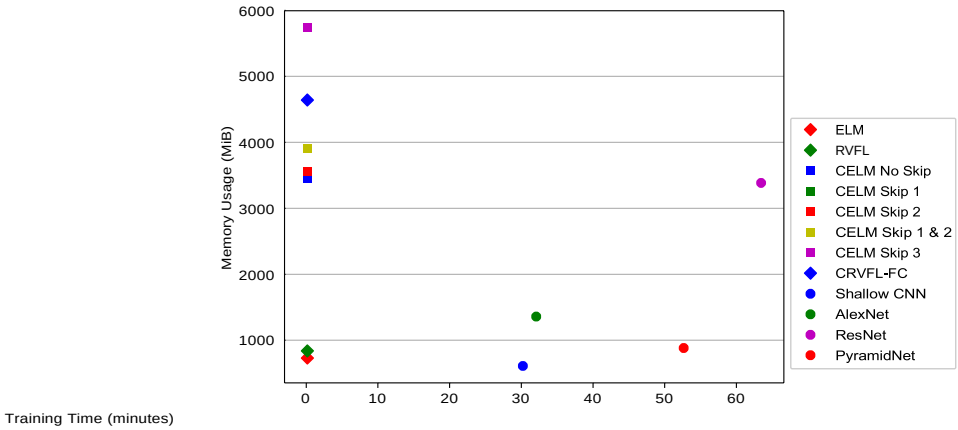

Figure 88. Plot Depicting GPU Memory Usage vs Training Time for all the Networks in this Study. A Clear Delineation can be Seen Between the Semi-random vs. DNN Models

We can see that the training time is unchanged or is minimal between each semi-random DNN model. The accuracies vary across each model depending on the connectivity and overall layers in the model. The CELM with two skipped connections was found to perform the best, only by a marginal percentage and at the cost of high memory usage as highlighted in Table 36. A plot of the GPU

memory usage and training times for each model is shown in Figure 88. This plot shows how the semi-random DNNs train within seconds at the cost of higher memory usage. However, even amongst the semi-random DNNs, the ELM and RVFL networks achieve fast training times with minimal memory utilization. The conventional DNNs, with a batch size of 16, are able to minimize their memory usage at the cost of lengthy training times. Although, with a larger batch size, the conventional DNNs will utilize more memory along with increased training times.

The skipped connections hold the highest value in semi-random DNNs as they improve the networks performance empirically. Both the ELM and the CELM with no skipped connections were shown to perform poorly compared to other architectures. The skipped connections to both the first and second fully connected layers for the CELM performed the best. Intuitively this is because the skipped connections keep the original features from the inputs along with processed features through the layers of the network. Therefore, if more layers are added with additional skipped connections, we may see improvement in the network. We tested this scenario with the CELM with three fully connected layers and one skipped connection to the output layer. It performed worse than a CELM with only two fully connected layers and a skipped connection to the output layer. However, with skipped connections to intermediate layers for three or more fully connected layers the accuracy of the network seems to improve.

6.0 CONCLUSION

In conclusion, this research report has summarized the results of investigation into both algorithmic advancements and hardware implementations of random projection networks. At an algorithmic level, we have proposed hardware simplifications of plasticity rules (such as short-term plasticity and the signed delta rule), accelerating stochastic simulations through GPUs, and the extension of basic random projection networks to deep architectures greatly improving their performance. The end goal of this work is the development of suitable neuromorphic architectures for on-device intelligence in embedded platforms. Towards this goal, we proposed several basic designs for digital and memristor-based architectures, which have been refined over the course of three years. To demonstrate the value of the research presented in this report we have studied the algorithmic performance on simple benchmarks, the hardware resource, area, and energy consumption, the scalability of the proposed models, the robustness to device noise, and extended the results to estimations on complex real-world applications such as speech recognition and video-activity recognition. The results demonstrate that the proposed designs are capable of surpassing state-of-the-art algorithms with a fraction of the resource and energy consumption.

7.0 REFERENCES

[1] Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: a new learning scheme of feedforward neural networks." *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. Vol. 2. IEEE, 2004.

[2] Maass, Wolfgang, Thomas Natschläger, and Henry Markram. "Real-time computing without stable states: A new framework for neural computation based on perturbations." *Neural computation* 14.11 (2002): 2531-2560.

[3] Jaeger, Herbert. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note." *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001): 13.

[4] Zyarah, Abdullah M., and Dhireesha Kudithipudi. "Semi-Trained Memristive Crossbar Computing Engine with In Situ Learning Accelerator." *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14.4 (2018): 43.

[5] Ma, Qianli, Lifeng Shen, and Garrison W. Cottrell. "Deep-esn: A multiple projection-encoding hierarchical reservoir computing framework." *arXiv preprint arXiv:1711.05255*(2017).

[6] Gallicchio, Claudio, Alessio Micheli, and Luca Pedrelli. "Comparison between DeepESNs and gated RNNs on multivariate time-series prediction." *arXiv preprint arXiv:1812.11527* (2018).

[7] Graham, Dillon, et al. "Convolutional Drift Networks for Video Classification." *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017.

[8] Carmichael, Zachariah, et al. "Mod-DeepESN: Modular Deep Echo State Network." *arXiv preprint arXiv:1808.00523* (2018).

[9] Soures, Nicholas and Dhireesha Kudithipudi. "Spiking Reservoir Networks." *IEEE signal processing magazine* (2019) In Review.

[10] A. Litwin-Kumar, K. D. Harris, R. Axel, H. Sompolinsky, and L. Abbott, "Optimal degrees of synaptic connectivity," Neuron, vol. 93, no. 5, pp. 1153–1164, 2017.

[11] Barak, Omri, Mattia Rigotti, and Stefano Fusi. "The sparseness of mixed selectivity neurons controls the generalization–discrimination trade-off." *Journal of Neuroscience* 33.9 (2013): 3844-3856.

[12] M. Lukosevi ˇ cius, "A practical guide to applying echo state networks," in ˇ Neural networks: tricks of the trade. Springer, 2012, pp. 659–686.

[13] Polepalli, Anvesh, Nicholas Soures, and Dhireesha Kudithipudi. "Digital neuromorphic design of a Liquid State Machine for real-time processing." *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016.

[14] Soures, Nicholas M. *Deep Liquid State Machines with Neural Plasticity and On-Device Learning*. Diss. Rochester Institute of Technology, 2017.

[15] Soures, Nicholas, et al. "Enabling On-Device Learning with Deep Spiking Neural Networks for Speech Recognition." *ECS Transactions* 85.6 (2018): 127-137.

[16] Soures, Nicholas and Dhireesha Kudithipudi. "Spiking Reservoir Networks." *IEEE signal processing magazine* (2019) In Review.

[17] Zyarah, Abdullah M., and Dhireesha Kudithipudi. "Resource sharing in feed forward neural networks for energy efficiency." *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2017.

[18] Kudithipudi, Dhireesha, et al. "Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing." *Frontiers in neuroscience* 9 (2016): 502.

[19] D. Christiani, C. Merkel and D. Kudithipudi, "Invited: Towards a scalable neuromorphic hardware for classification and prediction with stochastic No-Prop algorithms," *2016 17th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, 2016, pp. 124-128.
doi: 10.1109/ISQED.2016.7479187

[20] Ramakrishnan, Swathika. "Accelerating Stochastic Random Projection Neural Networks." MS Thesis, 2017 (https://scholarworks.rit.edu/theses/9701/)

[21] C. Merkel *et al*., "Neuromemristive Systems: Boosting Efficiency through Brain-Inspired Computing," in *Computer*, vol. 49, no. 10, pp. 56-64, Oct. 2016.
doi: 10.1109/MC.2016.312

[22] Zyarah, Abdullah M., et al. "Ziksa: On-chip learning accelerator with memristor crossbars for multilevel neural networks." *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017.

[23] Zyarah, Abdullah M., Nicholas Soures, and Dhireesha Kudithipudi. "On-Device Learning in Memristor Spiking Neural Networks." In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5. IEEE, 2018.

[24] Norton, David, and Dan Ventura. "Improving liquid state machines through iterative refinement of the reservoir." *Neurocomputing* 73.16-18 (2010): 2893-2904.

[25] Garofolo, John S., et al. "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1." *NASA STI/Recon technical report n* 93 (1993).

[26] Soures, Nicholas, et al. "Reservoir Computing in Embedded Systems: Three variants of the reservoir algorithm." *IEEE Consumer Electronics Magazine* 6.3 (2017): 67-73.

[27] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11):2278-2324, November 1998.

[29] A. M. Zyarah and D. Kudithipudi, “Extreme learning machine as a generalizable classification engine,” in Neural Networks (IJCNN), 2017 International Joint Conference on. IEEE, 2017.

[30] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, “Extreme learning machine for regression and multiclass classification,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 42, no. 2, pp. 513–529, 2012

[31] E. Yao and A. Basu, “VLSI extreme learning machine: A design space exploration,” CoRR, vol. abs/1605.00740, 2016. [Online]. Available: http://arxiv.org/abs/1605.00740

[32] I. Kuon and J. Rose, “Measuring the gap between fpgas and asics,” IEEE transactions on computer-aided design of integrated circuits and systems, vol. 26, no. 2, pp. 203–215, 2007.

[33] J. V. Frances-Villora, A. Rosado-Munoz, J. M. Mart ˜ ´ınez-Villena, M. Bataller-Mompean, J. F. Guerrero, and M. Wegrzyn, “Hardware implementation of real-time extreme learning machine in fpga: Analysis of precision, resource occupation and performance,” Computers & Electrical Engineering, 2016.

[34] S. Decherchi, P. Gastaldo, A. Leoncini, and R. Zunino, “Efficient digital implementation of extreme learning machines for classification,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 59, no. 8, pp. 496–500, 2012.

[35] Andrzejak, Ralph G., et al. "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state." *Physical Review E* 64.6 (2001): 061907.

[36] Husmeier, Dirk. "Random vector functional link (RVFL) networks." *Neural Networks for Conditional Probability Estimation*. Springer, London, 1999. 87-97.

[37] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[38] Zhang, Le, and Ponnuthurai Nagaratnam Suganthan. "Visual tracking with convolutional random vector functional link network." *IEEE transactions on cybernetics* 47.10 (2016): 3243-3253.

[39] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[40] Han, Dongyoon, Jiwhan Kim, and Junmo Kim. "Deep pyramidal residual networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

LIST OF PUBLICATIONS

Title	Reconfigurable Digital Liquid State Machine for Real-Time Processing
Author(s)	Anvesh Polepalli, Dhireesha Kudithipudi
Publication Date	May 1st, 2016
Publication venue	ACM Conference- Nanocom
Publication keywords	LSM, Reservoir, Digital

Title	Robustness of a memristor based liquid state machine.
Author(s)	Nicholas Soures, Lydia Hays, and Dhireesha Kudithipudi
Publication Date	2017
Publication venue	Neural Networks (IJCNN), 2017 International Joint Conference on
Publication keywords	Liquid State Machine, Memristor, Robustness, Reliability, Reservoir

Title	Ziksa: On-chip learning accelerator with memristor crossbars for multilevel neural networks
Author(s)	Abdullah M. Zyarah, Nicholas Soures, Lydia Hays, Robin B. Jacobs-Gedrim, Sapan Agarwal, Mathew Marinella, and Dhireesha Kudithipudi
Publication Date	28-31 May 2017
Publication venue	Circuits and Systems (ISCAS), 2017 IEEE International Symposium on
Publication keywords	

Title	Reservoir Computing in Embedded Systems: Three variants of the reservoir algorithm
Author(s)	Nicholas Soures, Cory Merkel, and Dhireesha Kudithipudi
Publication Date	14 June 2017
Publication venue	IEEE Consumer Electronics Magazine 6.3
Publication keywords	

Title	On accelerating stochastic neural networks
Author(s)	Swathika Ramakrishnan and Dhireesha Kudithipudi
Publication Date	September 27 - 29, 2017
Publication venue	Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication
Publication keywords	Stochastic Computing, Neural Networks, Neuromorphic, Extreme Learning Machine

Title	On-Device STDP and Synaptic Normalization for Neuromemristive Spiking Neural Network
Author(s)	Nicholas Soures, Lydia Hays, Eric Bohannon, Abdullah M. Zyarah, and Dhireesha Kudithipudi
Publication Date	6-9 Aug 2017
Publication venue	Circuits and Systems (MWSCAS), 2017 IEEE 60th International Midwest Symposium on
Publication keywords	Synaptic Normalization, Memristor, Neuromemristive Systems, Spiking Neural Network, STDP

Title	Resource Sharing in Feed Forward Neural Networks for Energy Efficiency
Author(s)	Abdullah M. Zyarah and Dhireesha Kudithipudi
Publication Date	6-9 Aug 2017
Publication venue	Circuits and Systems (MWSCAS), 2017 IEEE 60th International Midwest Symposium on
Publication keywords	Extreme Learning Machine, Folding, Coalescing

Title	Enabling On-Device Learning with Deep Spiking Neural Networks for Speech Recognition
Author(s)	Nicholas Soures, Dhireesha Kudithipudi, Robin Jacobs-Gedrim, Sapan Agarwal, Matthew Marinella
Publication Date	May 15 2018
Publication venue	ECS Transactions
Publication keywords	NA

Title	Spiking Reservoir Networks
Author(s)	Nicholas Soures and Dhireesha Kudithipudi
Publication Date	In Review
Publication venue	IEEE Signal Processing Magazine
Publication keywords	Spiking Neural Network, Reservoir Computing, Neuromorphic Computing, Liquid State Machine

Title	Convolutional Drift Networks for Video Classification
Author(s)	Dillon Graham, Seyed Hamed Fatemi Langroudi, Christopher Kanan, and Dhireesha Kudithipudi
Publication Date	Nov, 2017
Publication venue	IEEE International Conference on Rebooting Computing
Publication keywords	Deep Learning, Reservoir Computing, Video Activity Classification

Title	Mod-DeepESN: Modular Deep Echo State Network
Author(s)	Zach Carmichael, Seyed Hamed Fatemi Langroudi, Stuart Burtner, and Dhireesha Kudithipudi
Publication Date	Aug 1, 2018
Publication venue	CCNeuro and Arxiv
Publication keywords	Echo State Network, Intrinsic Plasticity, Time series prediction, Reservoir Computing

Title	Semi-Trained Memristive Crossbar Computing Engine with In Situ Learning Accelerator
Author(s)	Abdullah M. Zyarah and Dhireesha Kudithipudi
Publication Date	December 2018
Publication venue	AMC Journal on Emerging Technologies in Computing Systems (JETC)
Publication keywords	On-Device Learning, Semi-trained Neural Network, Memristive Crossbar, Extreme Learning Machine

Title	An Ensemble Learning Approach to the Predictive Stability of Echo State Networks
Author(s)	Q Wu, E Fokoue, D Kudithipudi
Publication Date	August 2018
Publication venue	Journal of Informatics and Mathematical Sciences
Publication keywords	Echo state networks

Title	On the statistical challenges of echo state networks and some potential remedies
Author(s)	Q Wu, E Fokoue, D Kudithipudi
Publication Date	Feb 2018
Publication venue	Arxiv
Publication keywords	Echo state networks

Title	On-Device Learning in Memristor Spiking Neural Networks
Author(s)	Abdullah M. Zyarah, Nicholas Soures, and Dhireesha Kudithipudi
Publication Date	27-30 May 2018
Publication venue	IEEE International Symposium on Circuits and Systems (ISCAS)
Publication keywords	NA

Title	Deep Liquid State Machine with Neural Plasticity for Video Activity Recognition
Author(s)	Nicholas Soures and Dhireesha Kudithipudi
Publication Date	In Review
Publication venue	Frontiers in Neuroscience
Publication keywords	Spiking Neural Network, LSM, Local learning, Deep

Title	Semi-random deep neural networks for near real-time target classification
Author(s)	Humza Syed, Ryan Bryla, Cory Merkel, Uttam Mazumder, Dhireesha Kudithipudi
Publication Date	April 18, 2019
Publication venue	SPIE
Publication keywords	Random Projection Networks

LIST OF ACRONYMS

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
CDNs	Convolutional Drift Networks
CELM	Convolutional Extreme Learning Machine
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CR	Classification Rate
CRVFL	Convolutional Random Vector Functional Link
DAC	Digital Analog Converter
DNNs	Deep Neural Networks
ELM	Extreme Learning Machine
ESN	Echo State Network
FF	First Fit
FPGA	Floating Point Grid Array
GPU	Graphics Processing Unit
HRS	High-Resistive State
ICMR	Input Common Mode Range
LIF	Leaky Integrate-and-Fire
LRS	Low-Resistive State
LSM	Liquid State Machine
MLP	Multilayer Perceptron
PCA	Principal Component Analysis
PPM	Parts Per Million
PVT	Process, Voltage and Temperature
RC	Reservoir Computing
RPNs	Random Projection Networks
RTL	Register-Transfer Level
RVFL	Random Vector Functional Link
SNN	Spiking Neural Network
SOTA	State-of-the-Art
STP	Short-Term Plasticity
STDP	Spike-Timing-Dependent Plasticity
SWaP	Size, Weight, and Power
VLSI	Very-Large-Scale Integration
WTA	Winner-Take-All