

CMOS-Memristive Neuromorphic Architecture for Nonlinear Signal Processing

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Manu Rathore

May 2024

© by Manu Rathore, 2024
All Rights Reserved.

*Dedicated to my beloved family, whose unwavering love and support have guided me through
this transformative journey.*

Acknowledgments

I would like to express my heartfelt gratitude to my advisor, Dr. Garrett S. Rose, for his invaluable support throughout the completion of my Ph.D. at the University of Tennessee. His guidance and encouragement have been instrumental in shaping my academic journey, providing me with invaluable insights and direction. The discussions and brainstorming sessions with him were not only intellectually stimulating but also served as catalysts for my growth and development as a researcher.

I am also deeply grateful to the members of my doctoral committee, Dr. Catherine Schuman, Dr. Andy Sarles, and Dr. Ahmedullah Aziz, for their invaluable insights and feedback on my work. Their technical expertise and guidance, coupled with their crucial inputs on my writing, were deeply appreciated and have significantly enhanced the quality of my research.

I am also grateful to the faculty members at the Electrical Engineering and Computer Science Department at The University of Tennessee, whose expertise and dedication have enriched my learning experience immensely. Each professor's unique perspective and teaching style have contributed to my growth and understanding in profound ways.

Working on my Ph.D. research during the COVID-19 pandemic has been an experience filled with numerous challenges and uncertainties. Throughout this journey, the unwavering support of my husband, Bhaskar has been instrumental in enhancing my resilience. The support and encouragement from all my family members have been the cornerstone of my strength during the numerous ups and downs on this path. I'm truly appreciative of Bhaskar's invaluable feedback and thorough reviews of my research papers, which have helped enhance the quality of my writing. His insightful perspectives and unwavering belief in my abilities have been a constant source of motivation and inspiration.

I also feel indebted to the entire TENN-lab research team for their camaraderie, collaboration, and support. Their collective efforts and valuable discussions have played a pivotal role in the success of my research endeavors.

Lastly, I would like to extend my heartfelt thanks to each and every individual who has played a role in making this work possible.

Abstract

Neuromorphic computing mimics the functional components and structure of the human brain to achieve highly efficient computing with minimal resources and power consumption. Creating neuromorphic systems in Complementary Metal-Oxide-Semiconductor (CMOS) technology offers an alternative computing paradigm to Von neumann computing. However, implementing these systems on an CMOS Integrated Circuit (IC) poses major challenges. These challenges include implementing synaptic weight multiplication and weight tuning operation that conserves energy and occupies minimal area. Additionally, designing a network-on-chip architecture that is reconfigurable and offers a full-connectivity design space is crucial. Furthermore, implementing a complete architecture for nonlinear data processing and, specifically, online learning on-chip is another significant challenge.

Memristors are emerging nano devices that exhibit behavior similar to synapse in the brain with few of the memristor types offering extremely small footprint and non-volatile resistance on-chip. These properties lead to small area and low-power IC designs. The device models for these emerging devices are crucial for designing circuits and systems that utilize them. In this work, we address the integration of memristors with the circuit simulation environment by modeling crucial features of these devices in a SPICE-compatible model. The presented model offers a very low 2-3% error in LRS tuning when compared to average experimental data while converging well in simulation.

Further, we also present a reconfigurable and scalable network architecture that allows full-connectivity while minimizing resource overhead for spike communication between proximal neurons by ensuring "spikes-as-spikes" communication. For spike communication between neurons farther away on-chip from each other, AER is utilized. Thus, a Hierarchical Circuit-Packet Hybrid approach for scalable neuromorphic systems is proposed. This technique leads to up to 48% savings in area and 90% savings in power for the neurons placed physically close to each other on-chip.

Lastly, we present resource-efficient hardware architecture for online learning using the reservoir computing approach for non-linear signal processing. The application of wireless channel equalization is considered which includes reflection, inter-symbol interference, and

signal attenuation and also has a memoryless non-linearity of a second order. The proposed approach leverages the circuit-switched Network-on-Chip (NoC), SpiCS-Net, as a robust "spikes-as-spikes" routing mechanism for handling input and output spikes within the reservoir. This strategy enables resource optimization by strategically reducing the number of output neurons, while simultaneously preserving the system's flexibility to designate any neuron within the reservoir as an output neuron. The proposed approach offers upto 30% savings in area of the output layer in IBM CMOS10LPE 65 nm technology.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Research Goals and Contributions	3
1.3	Navigating this document	10
2	Background	11
2.1	Neuromorphic Computing	11
2.2	Neuromorphic System: Building Blocks	13
2.2.1	Neuron	13
2.2.2	Memristive Synapse	15
2.2.3	Network-on-chip (NoC)	17
2.2.4	Circuitry for Online Learning	22
3	Compact Device Model for HfO₂ memristors	26
3.1	Background	26
3.2	Capturing Variable Dynamics of HfO ₂ Memristors	28
3.3	<i>RESET</i> Failure in HfO ₂ Memristor	33
3.3.1	Device physics behind <i>RESET</i> failure	33
3.3.2	Experimental Results from 1T1R devices	35
3.3.3	Incorporating <i>RESET</i> Failure into the Verilog-A Model	35
3.4	Simulation Results	39
3.5	Benchmarking Simulation for Convergence	44
3.6	Conclusion	44

4	Circuit-Switched Network-on-Chip Neuromorphic Architecture	47
4.1	SpiCS-Net NoC Architecture	49
4.1.1	Mathematical Analysis	54
4.2	Tuning Concurrent Connectivity of SpiCS-Net for Sparse SRNNs	57
4.3	Implementation	60
4.3.1	Digital Spike Communication	60
4.3.2	Analog Spike Communication	66
4.4	Results	68
4.5	Conclusion	71
5	Hierarchical Network-on-Chip for Scalable Neuromorphic Architecture	74
5.1	Background	74
5.2	Hybrid Circuit-Packet-Switched NoC Architecture	77
5.2.1	Main Contributions	78
5.2.2	R3 Network Layer - Tree structure (Packet-Switching)	80
5.2.3	R2 Network Layer - Mesh Structure (Packet-Switching)	80
5.2.4	R1 Network Layer - Clos Topology (Circuit-Switching)	81
5.2.5	Network Partitioning	83
5.3	Implementation	86
5.4	Results	89
5.5	Discussion	97
6	Architecture for Online Learning	99
6.1	Background	99
6.2	Architecture for Efficient Reservoir Computing using Circuit-Switched NoC .	104
6.2.1	Application: Nonlinear Channel Equalization	108
6.3	Implementation and Verification	111
6.3.1	Mixed-Signal Implementation	111
6.4	Results	115
6.4.1	Mixed-Signal Implementation	118
6.5	Discussion	121

7 Conclusion	125
7.1 Summary of Contributions	126
Bibliography	128
Appendix	143
A Reliability Analysis of Memristive Reservoir Computing Architecture	144
A SRNN based RC Architecture	148
A.1 Hardware Model	151
B Results and Comparisons	154
B.1 Chaotic Time Series Prediction task	154
B.2 Image recognition task	159
B Verilog-A Model for HfO₂ Memristor	161
C System Verilog Code for Reservoir Computing Design	175
A Network-on-Chip: spics_net.sv	175
B Network-on-Chip: switch.sv	179
C Reservoir SRNN: reservoir.vams	181
D Output Readout Layer: Output_layer.sv	185
E Top-level Module: top_reservoir.sv	191
Vita	196

List of Figures

1.1	A Spiking Recurrent Neural Network (SRNN) consists of interconnected neurons and synaptic connections. In an SRNN, neurons are computational units that simulate the behavior of biological neurons, while synaptic connections represent the communication pathways between neurons. Several labeled neurons demonstrate recurrent connectivity, forming loops within the network. Notably, two distinct loops are highlighted: the first loop comprises neurons N1, N2, N3, N4, and loops back to N1, while the second loop involves neurons N4, N5, N6, and returns to N4. These loops illustrate the recurrent interconnections within the network, where signals can propagate in a cyclic manner, allowing for the processing of temporal information and the generation of dynamic responses.	2
1.2	Depicting the architecture of a Spiking Recurrent Neural Network (SRNN)-based Reservoir Computing (RC) system, showcasing the dynamic network structure used for computational tasks.	9

2.1	The Spiking Recurrent Neural Network (SRNN) displays recurrent connectivity among neurons. To implement these recurrent connections on hardware, it becomes imperative to accommodate any-to-any neuron connectivity within the SRNN. Additionally, the process of applying synaptic weights to spikes entails intricate multiplication operations. Both these aspects present significant challenges for hardware realization, as they markedly escalate resource demands, even for modestly sized SRNN configurations. Overcoming these challenges necessitates innovative hardware design strategies capable of efficiently managing the complexities associated with neuron connectivity and synaptic weight computations, thereby enabling the practical deployment of SRNNs in real-world applications.	12
2.2	Neurons communicate through synaptic connections by transmitting voltage spikes. In this abstract representation, spikes originating from Neuron 1 and Neuron 2 are relayed to Neuron 3. Upon reception, Neuron 3 integrates the incoming spikes at its input. When the accumulated potential surpasses a certain threshold, Neuron 3 generates an output spike. This neuron behavior can be modeled as Leaky Integrate-and-Fire model.	14
2.3	Mixed-Signal LIF Neuron circuit used in the system level design of Spiking Recurrent Neural Network (SRNN) takes analog signal at the input and generates digital output spikes [22]. The mixed-signal LIF neuron circuit integrates analog and digital components to replicate the essential characteristics of biological neurons, including membrane potential dynamics, threshold-based spike generation, and refractory behavior.	14
2.4	The memristor 'M' is represented as a fundamental circuit element, embodying the Charge-Flux relationship.	16
2.5	Memristive synapse [117] circuit showing MOSFET(metal-oxide-semiconductor field-effect transistor) controls for resistance programming	18

2.6	The Mesh NoC architecture demonstrates the AER Packet-based spike communication approach. This method involves a recurrent connection loop between neurons N1, N2, and N1. This loop entails spike communication as follows: N1 emits a spike, which is encoded to a multi-bit digital data packet and routed through a series of routers before reaching N2, and vice versa for spike from N2 to N1 [90].	20
3.1	Schematic diagram of a 1T1R (One Transistor One Resistor) configuration, illustrating the integration of an NMOS (N-channel Metal-Oxide-Semiconductor) transistor to regulate the current flow through the memristor device.	29
3.2	Flowchart illustrating the Verilog-A model, depicting unformed and formed states for the memristor. Switching between resistance values, Low Resistance State (LRS) and High Resistance State (HRS), occur during <i>SET</i> and <i>RESET</i> operations, respectively, after forming.	29
3.3	The window function utilized for Low-Resistance State (LRS) and High-Resistance State (HRS) switching is a sigmoid function. The sigmoid functions employed for LRS and HRS switching incorporate two defining values: the midpoint and the slope, which determine the shape and characteristics of the transition. The midpoint is calculated based on the polynomial fit HRS and LRS values from Eq. (3.2) and Eq. (3.3). The slope for LRS is higher than HRS due to faster transitions during <i>SET</i> when compared to <i>RESET</i> operation [92].	32
3.4	I-V plot of the proposed memristor model illustrates multiple switching cycles between the Low-Resistance State (LRS) and High-Resistance State (HRS). This plot captures the dynamic behavior of the memristor as it undergoes switching events, showcasing the characteristic hysteresis loop associated with memristive devices [92].	32

3.5	Cross-sectional schematics depict the impact of various <i>RESET</i> voltages on filament behavior. (a) Demonstrates the impact on a FORMED memristor when the voltage across the branch 1T1R (1 transistor 1 memristor), VRESET, is -1 volt, lower than the <i>RESET</i> threshold V_{tn} in absolute value. Insufficient VRESET results in an intact filament, indicating no <i>RESET</i> operation occurred. (b) Illustrates the <i>RESET</i> operation occurring when VRESET crosses <i>RESET</i> threshold at VRESET = -1.5 volts. The filament experienced breakdown and <i>RESET</i> occurs. (c) Depicts a <i>RESET</i> failure scenario observed when VRESET is -2 volts, much higher than <i>RESET</i> threshold V_{tn} . The elevated VRESET causes defects to migrate from the bottom of the memristor, initiating the formation of a reverse filament from the bottom. Consequently, the operation behaves akin to a <i>SET</i> operation rather than a <i>RESET</i> operation [28]. . . .	34
3.6	The experimental results in this plot showcase testing data from the HfO ₂ memristors for various observed <i>RESET</i> voltages across multiple switching cycles as VRESET is varied. The High-Resistance State (HRS) values at each <i>RESET</i> voltage are depicted in blue. When the <i>RESET</i> voltage isn't sufficiently low, a complete <i>RESET</i> operation fails to occur(Fig. 3.5(a)). This is evident in the range from -1.3V to -1V, where the memory window is insufficiently large. The optimal range appears to be around -1.5V, where a full <i>RESET</i> is achieved(Fig. 3.5(b)). The graph notably indicates the VRESET level where the absolute value of <i>RESET</i> voltage is too high, resulting in <i>RESET</i> failure and a subsequent resistance drop to approximately 1.5 k Ω (Fig. 3.5(c)). Additionally, the value of Low-Resistance State (LRS) is highlighted in orange, demonstrating the shrinking memory window as device operation parameters approach the <i>RESET</i> failure threshold [64].	36

3.7	The Flowchart depicting the modified Verilog-A model for the HfO ₂ memristor device as it transitions between different resistance states. In addition to the traditional Low Resistance State (LRS) and High Resistance State (HRS), a <i>RESET</i> Failure State (RFS) is introduced. Once the device enters the RFS, it remains in this state for the duration of the simulation as it is not recoverable. This model enhancement provides a more comprehensive representation of the device behavior, particularly in scenarios where failure states occur.	38
3.8	Low Resistance State (LRS) with respect to the gate voltage, V_g , of the transistor regulating the current through the memristor device, also known as the current compliance, against experimental data.	40
3.9	HRS vs. reset voltage, V_{reset} , against experimental data showing the impact of V_{reset} on HRS values.	40
3.10	Experimental vs. Simulation data for multiple switching cycles showing LRS samples at different V_g values (LRS Variability)	41
3.11	Experimental vs. Simulation data for multiple switching cycles showing HRS samples at different V_{reset} value (HRS Variability)	42
3.12	The memristive synapse utilized for simulating <i>RESET</i> failure is depicted [117]. The memristor device labeled " R" is shown with different MOSFET devices to control various operations. During the <i>RESET</i> operation, a dedicated V_{DD} for M_{P1} is required. It is crucial to maintain the voltage across the memristor below a certain threshold to prevent <i>RESET</i> failure.	42
3.13	The Cadence simulation results demonstrate the failure modeling of the proposed Verilog-A model. Control signals are employed to initiate the <i>READ</i> , <i>RESET</i> , and <i>SET</i> operations for various values of $RESETB_VDD$. Three distinct test cases are presented: when $RESETB_VDD$ is below the <i>RESET</i> threshold (V_{thn}), within the <i>RESET</i> window (V_{thn} to V_{tRFS}), and above the <i>RESET</i> failure voltage (V_{tRFS}). The bottom waveform plot illustrates the device entering the <i>RESET</i> failure state. Note: A positive $RESETB_VDD$ value corresponds to a negative <i>RESET</i> voltage across the memristor.	43

3.14	The illustrated benchmark circuit acts as a convergence test for the proposed Verilog-A model of the memristor. Comprised solely of memristors, the circuit's ability to converge numerical equations directly relies on the smoothness and continuity of the model equations. This test provides valuable insight into the accuracy and reliability of the proposed Verilog-A model in simulating memristor behavior under various conditions [92].	45
4.1	In a generic Spiking Recurrent Neural Network (SRNN), recurrent connectivity necessitates a larger connectivity space. This requirement means that the Network-on-Chip (NoC) must be capable of supporting a fully-connected SRNN. Unlike traditional neural networks, where connectivity is constrained by feed-forward synaptic communication between adjacent layers only, the SRNN's recurrent connections demand a more extensive network topology. As a result, the hardware resources required for implementing such connectivity increase manifold.	50
4.2	A neuron core, the fundamental unit of SRNN shown in Fig. 4.1, consists of multiple synaptic inputs. These inputs serve as the conduits through which information flows into the neuron, enabling it to process and integrate incoming signals. Each synaptic input represents a connection from other neurons. . .	50
4.3	Crossbar composed of enable/disable switches with one switch per cross-point between an input-output pair. Simple NoC but implementing the entire system using Crossbar Network-on-chip is inefficient since the number of switches scales multiplicatively.	51
4.4	Typical enable/disable controlled switch in a circuit switching crossbar; this switch can be implemented using Transmission gates, MOSFET switches, or Multiplexers	51
4.5	SpiCS-Net demonstrating communication of a spike between a set of neurons within the SRNN.	53

4.6	SpiCS-Net demonstrating one possible way for multicast spike communication within the SRNN. Here, output of one neuron is routed to input synaptic connections of three neurons.	55
4.7	SpiCS-Net architecture shows a lowering of crossbar switches in the network, leading to area saving without restricting connectivity between input and output nodes. If $S_M = p$, the SpiCS-Net would always provide at least one routing path from every input node to any output node. This is the minimum connectivity that needs to be supported by the NoC. By increasing $S_M = p + q - 1$, the network will always allow a path from any input node to any output node without rearranging the connections. $S_M \geq q$ ensures 100% concurrent connectivity if $q > p$. A 100% Concurrent Connectivity implies that all neurons can drive synaptic inputs of all other neurons simultaneously. This is not realistic for a hardware SRNN due to a few reasons: 1. Neuron circuits will have limited fan-out and can not drive unlimited synapses simultaneously; 2. The purpose-built trained SRNNs will never need a fully-connected level of synaptic density. In most applications, SRNNs are sparse than they are dense. Thus, the density of connections that must be established simultaneously (Concurrent Connectivity) can be restricted.	58
4.8	Custom designed LIF (Leaky Integrate-and-Fire) Mixed-Signal memristor-based neuron core layout designed in IBM CMOS10LPE 65nm CMOS-memristor technology with 16 synaptic inputs (size of neuron = 100x190 sq. μm)	61
4.9	Layout of 128x128 SpiCS-Net with 75% concurrent connectivity in IBM CMOS10LPE 65nm process. Area of SpiCS-Net = 0.032 sq. mm	61
4.10	Layout of Spiking Recurrent Neural Network Cluster with SpiCS-Net, 128 Neuron cores, SPI, and scan_config block in IBM10LPE 65nm process. Area = 4.5 sq mm	63

4.11	SpiCS-Net comparison with circuit-switching Crossbar. Area (Total number of switches) vs. number of neurons in SRNN: Crossbar vs. SpiCS-Net. Here, the fan-in and fan-out of the neurons = 1 to arrive at input $\times$ output NoC configuration. It is observed that as the SRNN size goes up, area savings increase.	64
4.12	The total number of switches for various percentages of Concurrent Connectivity in the SpiCS-Net based NoC for reconfigurable SRNN implementation on hardware. The number of switches relates directly to switch circuits and memory requirements on the chip.	64
4.13	Area comparison for various percentages of Concurrent Connectivity in the 128x128 SpiCS-Net in the 65nm process.	65
4.14	Frequency response at Input(1), Intermediate(2), and Output(3) stages of Analog SpiCS-Net Clos topology shown in Fig. 4.5. (a) transmission gate-based Analog SpiCS-Net (b) NMOS switch-based Analog SpiCS-Net (c) PMOS switch-based Analog SpiCS-Net	72
5.1	Drawing inspiration from the intricate synaptic network of the brain: This hierarchical map of the macaque brain network, as detailed in [75], exhibits small-world network characteristics. It boasts dense local clustering alongside sparse global connections, resulting in efficient short-hop paths.	76

5.2	The proposed hierarchical circuit-packet network structure consists of three levels: Mesh (R2) and Tree (R3) represent AER-based packetized spike communication architectures. At the lowest level (R1), a circuit-switched Network-on-Chip (NoC) architecture is implemented, grouping neurons in close proximity in both space and time within the same cluster. This approach combines circuit and packet-switching, facilitating efficient dedicated spike communication (circuit-switched) for closely located neurons on-chip with denser connectivity, while also utilizing the flexibility of packet-switching for globally sparse connections. The proposed NoC is efficient for implementing any SNN, adapting to hardware characteristics for both local and global communication, and is particularly effective for those with a small-world network configuration.	79
5.3	AER packet format at the R2 level* is illustrated. The multiple AER packet types are discernible based on specific header bits.	82
5.4	R2 layer router* is a mesh router employing destination-routing. Routes are determined by combinational logic, triggering a request for one of the cardinal directional outputs. Each directional output features a look-ahead, round-robin arbiter to prevent deadlocks and packet loss.	82
5.5	Clos circuit-switched network at R1 level in the NoC hierarchy with three switch stages, showcasing the number of inputs, outputs, and switches in each stage.	84
5.6	Integration of R1 circuit-switched NoC with mesh R2 level. R1 is used to only establish local connectivity ensuring direct "spikes-as-spikes" communication within the cluster. AER packets converted to spikes from the R2 level are routed to the neurons directly through an R1 router.	85
5.7	Final layout* of implemented design including RISC-V and Reconfigurable Neuromorphic Array (RNA), employing the hierarchical circuit-packet network-on-chip architecture, comprises four R1 layer clusters connected by a mesh AER packetized NoC at the R2 layer.	88

5.8	Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size	91
5.9	Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size	92
5.10	Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size	92
5.11	Area plot against SRNN/cluster size for various NoC approaches reveals significant area overhead with AER packet-based NoC. The Clos network outperforms AER, but the hybrid hierarchical circuit-packet scheme excels above both by scaling much better as the system size increases. These area numbers include placed and routed layout than includes neurons along with NoCs. The hybrid network data corresponds to dividing the SNN into 4 clusters using the hierarchical R1 and R2 levels as shown in Fig. 5.2.	94
6.1	The image illustrates the architecture of a generic Reservoir Computing system, featuring the feed-forward output layer depicted on the right and the reservoir Spiking Recurrent Neural Network (SRNN) with recurrent connections shown on the left. The reservoir is initially configured with random parameters that remain unaltered throughout the training process. In contrast, the output layer typically comprises a dot product layer, with its weights adjusted iteratively by minimizing the discrepancy between the expected output and the actual output at each time step.	101
6.2	Reservoir block diagram of Neuromorphic Reservoir Computing architecture showing array of neuron core in the cluster connected via the circuit switched Noc, SpiCS-Net. The interface input/output signals of the Reservoir SRNN are also routed via SpiCS-Net. This allows for reconfigurability and control over which neurons are selected as input/output neurons, even at run time. .	105

6.3	A neuron core in the Neuron Cluster shown in Fig. 6.2 composing the SRNN reservoir. One neuron core consists of multiple synaptic inputs.	107
6.4	The proposed neuromorphic architecture adopts SpiCS-Net circuit-switched Network-on-Chip (NoC) and implements resource-saving measures by restricting the number of output neurons within the reservoir while allowing any neuron within the system to serve as an output neuron. Leveraging the reconfigurability feature inherent in the NoC, the architecture can dynamically adjust the connectivity of output neurons.	107
6.5	Reservoir Computing based Nonlinear Equalizer system utilizes SRNN reservoir to model nonlinear noise in input data. Analog signals from the Analog Front End are processed by reservoir neurons, generating spiking reservoir states, or spike maps, that capture nonlinear noise patterns. Spike maps operate on output weights to produce a multi-bit digital signal, which is then converted to analog for subtraction of noise from input, enabling the equalization and reconstruction of received distorted input data.	110
6.6	The simulation results from the Output Layer module offer valuable insights into the training dynamics of the neuromorphic architecture, with a specific focus on the evolution of weight coefficients throughout the training period. .	113
6.7	Simulation result from the Verilog-A neuron core reservoir module showing membrane potential of a neuron core as it integrates input potential and generates spike outputs.	113

6.8	Normalized Mean Squared Error for the Mackey Glass prediction task from the reservoir computing system as the number of output neurons changes while keeping SRNN size constant. The graph shows that reducing the number of output neurons in the reservoir to upto 60% leads to similar performance and NMSE does not have a significant impact. This ratio can go as low as 40% for large reservoir sizes with 768 and 1024 neurons. Notice that y-axis is in log scale to show the trend where NMSE is low. This observation supports usage of SpiCS-Net Circuit-Switched NoC for routing output spikes out of the reservoir such that number of output neurons are reduced while preserving the capability of selecting any neuron within the reservoir as an output neuron. .	116
6.9	Similar to previous graph in Fig. 6.8 this graph shows the Symbol Error Rate (SER) for the Channel Equalization task as the number of output neurons are varied. Here the y-axis is not log scale to show overall trend more clearly. It can be observed that reduction of the output neurons from 100% (1) to upto 40% (0.4) does not adversely impact the performance of the system.	117
6.10	To evaluate the effectiveness of the proposed Reservoir Computing (RC) system with a digital output layer against a multiplier-based design of the same size, power consumption is a crucial factor. The bar graph shows that proposed digital output layer design consumes less power compared to the multiplier-based approach due to the absence of power-intensive multiplication operations. These savings are shown for multiple reservoir sizes.	119
6.11	Bar graph comparing area utilization of the output layer efficiency in terms of resource utilization between different designs. The effectiveness of the proposed Reservoir Computing (RC) system with a digital output layer against a multiplier-based design of the same size is demonstrated by the presented results.	120

6.12	Bar graph illustrating power savings achieved through dimensionality reduction in the output layer of the Reservoir Computing system. The graph highlights the significant reduction in power consumption resulting from a 40% decrease in output layer dimensions, demonstrating the efficiency gains afforded by dimensionality reduction techniques.	122
6.13	Bar graph depicting the area savings achieved through dimensionality reduction in the output layer of the Reservoir Computing system. The graph illustrates the substantial decrease in area utilization resulting from a 40% reduction in output layer dimensions, showcasing the efficiency gains achieved by employing dimensionality reduction techniques.	122
A.1	Reservoir Computing (RC) system employing a Spiking Recurrent Neural Network (SRNN) architecture. The SRNN reservoir exhibits dynamic temporal processing capabilities essential for tasks such as time-series prediction and nonlinear data processing [93]	145
A.2	The HfO ₂ memristor-based synapse, designed and fabricated using a 65 nm CMOS process, incorporates various operations through different PMOS branches. These synapses serve as synaptic connections within the SRNN and within the memristor crossbar arrays at cross-points, enabling programming to different weight/conductance values.	149
A.3	MackeyGlass time-series: Spike raster representation depicting the activity within the reservoir network. Each spike corresponds to the firing of a neuron in response to the MackeyGlass time-series input, illustrating the dynamic behavior of the reservoir in processing temporal data [93]	152
A.4	Box plot illustrating the relationship between the standard deviation (SD) of noise in input data, ranging from 0–0.1, and the resulting standard deviation (SD) in the Normalized Root Mean Square Error (NRMSE). The plot demonstrates that as the SD of input data noise increases, there is a corresponding rise in the SD of NRMSE, indicating impact of in variability on performance [93]	156

A.5	The influence of input data noise on Normalized Root Mean Square Error (NRMSE) is significantly greater compared to the impact of synapse variability(Fig. A.4). While variations in synapse properties have minimal effect on NRMSE, changes in input data noise levels have a pronounced impact on the accuracy of the RC system’s predictions [93]	156
A.6	With increasing memristor variability in memristive crossbar-based Dot Product Engine (DPE) weights, the Normalized Root Mean Square Error (NRMSE) also rises, indicating a direct relationship between DPE weight variability and prediction accuracy. This suggests that fluctuations in DPE weights contribute significantly to prediction errors [93]	157
A.7	The Normalized Root Mean Square Error (NRMSE) per Epoch for the Mackey-Glass (MG) dataset reveals the impact of different sources of noise, stochastic effects, and plasticity mechanisms, internal and external to the Reservoir Computing (RC) system. Variations in NRMSE across epochs illustrate how factors such as external sources of noise, synaptic plasticity mechanisms, and stochastic behaviors affect the predictive performance of the RC system [93]	157
A.8	NRMSE plotted against Memristor Conductance Drift with and without Spike-Timing-Dependent Plasticity (STDP). This comparison highlights the impact of memristor conductance drift, with and without the application of STDP, on the predictive accuracy of the system. Understanding these relationships sheds light on the efficacy of STDP in mitigating the effects of memristor conductance drift on prediction performance [93]	158
A.9	The accuracy per epoch for the Iris dataset is compared across various scenarios, including synapse variability, Memristor Conductance Drift (MCD), and the application of Spike-Timing-Dependent Plasticity (STDP). This analysis provides insights into how these factors influence the classification performance of the system over the training epochs, offering valuable information for optimizing the performance of the Reservoir Computing system on image recognition tasks [93]	160

Chapter 1

Introduction

1.1 Motivation

Neuromorphic computing is a groundbreaking paradigm in the post Moore’s Law era that finds its inspiration in the architecture of the human brain for targeted task training [72]. Emulating the brain’s intricate architecture through abstract representations such as Spiking Recurrent Neural Networks (SRNNs), it uses specialized hardware and emerging devices to process information in a manner that mimics cognitive processes in the brain [67]. This innovative approach offers tremendous potential for applications like artificial intelligence, robotics, and cognitive computing, enabling the development of energy-efficient, highly parallel, and adaptable computing systems. As the need for more efficient computing solutions continues to escalate, the field of neuromorphic computing emerges as a promising frontier driving transformative technological advancements.

In Fig. 1.1, we illustrate a Spiking Recurrent Neural Network (SRNN), which comprises interconnected neurons. Each neuron in the SRNN processes incoming signals and generates output spikes based on its internal state and the strength of its connections to other neurons. The recurrent connections emphasized in Fig. 1.1 play a pivotal role enabling processing of information from past inputs, enabling SRNNs to exhibit memory property [110]. As a result, SRNNs prove to be highly efficient for the processing of temporal information. Further, due to its utilization of spike data, the SRNNs inherently operate with low power consumption

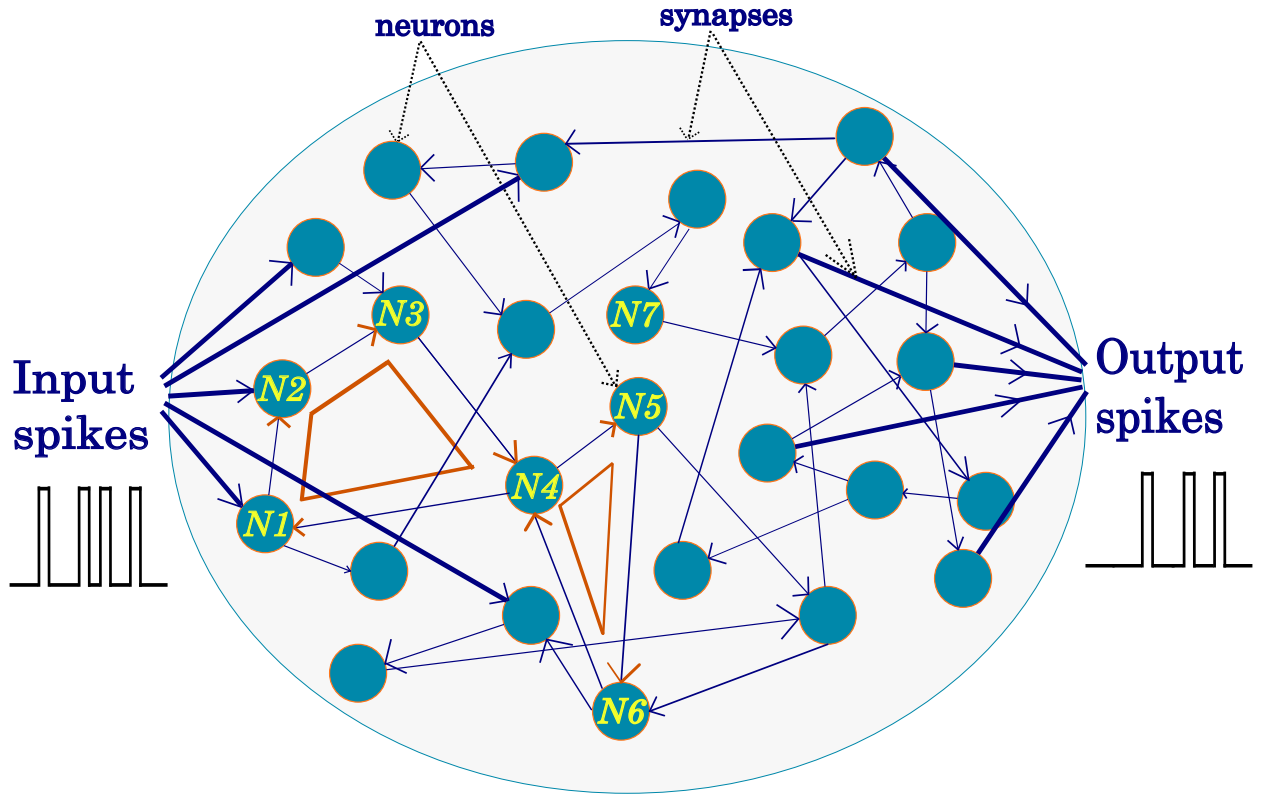


Figure 1.1: A Spiking Recurrent Neural Network (SRNN) consists of interconnected neurons and synaptic connections. In an SRNN, neurons are computational units that simulate the behavior of biological neurons, while synaptic connections represent the communication pathways between neurons. Several labeled neurons demonstrate recurrent connectivity, forming loops within the network. Notably, two distinct loops are highlighted: the first loop comprises neurons N1, N2, N3, N4, and loops back to N1, while the second loop involves neurons N4, N5, N6, and returns to N4. These loops illustrate the recurrent interconnections within the network, where signals can propagate in a cyclic manner, allowing for the processing of temporal information and the generation of dynamic responses.

[121]. Additionally, the use of emerging memristor devices, which feature compact footprints and support low-power analog computations, enhances the practicality of implementing SRNNs on hardware [93]. The implementation of a hardware-based SRNN system with online learning holds the promise of delivering more energy-efficient and highly accurate solutions for trainable data processing systems at the edge. However, the implementation of these SRNNs on hardware, especially in a reconfigurable manner, presents numerous challenges. These challenges are discussed in Section 1.2.

1.2 Research Goals and Contributions

Challenge 1: Reliability of Memristor-based Synaptic Weights in SRNNs

Implementing synapse-like properties in neuromorphic systems often requires hardware support for resource-intensive multiplication operations. This necessity arises from the requirement to apply synaptic strengths to the spikes communicated between a set of neurons through these synapses. Thus, for Spiking Recurrent Neural Networks (SRNNs) with numerous synaptic connections, multipliers can demand significant resources and pose challenges in implementation as the system scales. Memristor devices offer a promising alternative by enabling analog physics-based computations. As stated earlier, memristors boast advantages such as a compact footprint and low-power computational capabilities, making them ideal candidates for future computing systems [25].

Nevertheless, the integration of memristors introduces its own set of challenges, particularly concerning reliability and device variability [93]. To address these challenges, a crucial step is the development of a SPICE-compatible circuit simulation model. This model must accurately capture and account for the variability and potential failures exhibited by memristor devices. By incorporating variability and failure modes into the simulation model, researchers can gain insights into the behavior of memristors under different operating conditions and design more robust and reliable neuromorphic systems [2]. Additionally, such simulation models facilitate the exploration of novel memristor-based architectures and algorithms, paving the way for advancements in neuromorphic computing.

- In this work, a SPICE-compatible empirical circuit simulation model for HfO_2 memristors has been developed to incorporate critical aspects such as device variability and Reset failure. The primary aim of this model is to facilitate the reliable system-level design using HfO_2 memristors, ensuring robust performance unhindered by device variability and creating failure-aware designs.
- Further, a system level reliability analysis is performed on the SRNN in the context of a Reservoir Computing based neuromorphic system. This analysis considers read/write switching variability, memristor conductance drift and impact of STDP on the memristor-based synapses within the SRNN.

The Verilog-A model plays a fundamental role in facilitating the reliable design of systems that rely on these devices by allowing for the assessment and prediction of their behavior under various conditions using SPICE simulation tools.

Challenge 2: Network-on-Chip Implementation for SRNNs

While the design and development of various neuron and synapse implementations has garnered much interest within the domain of hardware-based neural networks, the network-on-chip (NoC) design for supporting reconfigurable connectivity on hardware has not been explored enough. Addressing this reconfigurable connectivity aspect is particularly crucial because the connections between neurons in the Spiking Recurrent Neural Network (SRNN) essentially define the entire system, and when it comes to physical hardware, establishing these connections presents a notable challenge [91].

Among the most efficient methods that have been proposed in the literature for implementing reconfigurable Spiking Recurrent Neural Networks (SRNNs) on hardware, Address Event Representation (AER) is the most prominent. AER involves representation of spike events in the form of packets representing address of source/destination neurons, facilitating spike communication across the network using various Network-on-Chip (NoC) architectures, such as mesh or tree structures [82, 20, 41, 111, 121]. This approach draws inspiration from silicon retina research [68] contributing to the development of large scale hardware implementations for Spiking Neuromorphic Architectures. While the Address Event

Representation (AER) approach excels in handling large reconfigurable Spiking Recurrent Neural Networks (SRNNs), it introduces an asymmetric packet-handling overhead for smaller networks consisting of only a few hundred neurons. In such cases, the additional circuitry needed to implement AER presents unnecessary resource overhead and poses challenges in terms of communication efficiency [87].

When considering the necessity for reconfigurable connectivity in Spiking Recurrent Neural Networks (SRNNs) of varying sizes, it becomes evident that the requirements for small networks differ significantly from those of larger networks. Additionally, scalability is a crucial factor, ensuring that both proximal neurons and distant neurons can be interconnected while optimizing resources and power consumption to balance the varying demands of their respective communication pathway. In small networks, the emphasis may be on efficient local connectivity and resource utilization, while in larger networks, the focus shifts towards establishing connections between distant neurons to facilitate global spike synchronization and minimal signal degradation [91]. These differing requirements are not addressed in the current research landscape, emphasizing the need for more focused investigation and innovation in NoC design, given its pivotal role in designing hardware SRNNs.

- To address this, we introduce a circuit switching based approach called Spike-routing Circuit Switched Network-on-chip (SpiCS-Net) for "spikes-as-spikes" communication within the Spiking Recurrent Neural Network(SRNN) [91]. What sets this approach apart is its versatility of implementation irrespective of the nature of spikes, whether digital or analog. This adaptability is achieved by flexibility of the switch design according to the specific characteristics of the spikes, ensuring that spike degradation is prevented [89]. Additionally, this proposed approach is built upon a multistage interconnect Clos network, which is delay-agnostic by design. As a result, this network does not require spike synchronization, unlike packet-based spike communication systems.
- We also provide a detailed analysis of the SpiCS-Net architecture while employing various switch implementations. The analysis encompasses area, power consumption, and throughput data for SpiCS-Net implementations with different switch types,

including multiplexer-based switches, transmission gates, as well as NMOS and PMOS-based switches. This evaluation aims to offer a comprehensive understanding of the performance and resource utilization of SpiCS-Net under various switch configurations.

Further, it’s evident that reconfigurable Spiking Recurrent Neural Networks (SRNNs) demand the ability for full on-chip connectivity driven by the presence of recurrent connections, which can lead to notably resource-intensive implementation. Nevertheless, not all of these connections need to be active simultaneously [69]. There are several reasons for this. Firstly, neurons have inherent limitations in terms of their fan-out and fan-in, which naturally limits the number of connections in SRNNs. Secondly, for various applications, SRNNs exhibit sparse recurrent connectivity [96, 49], which further supports the notion that not all connections need to be established concurrently. Thus, when it comes to network-on-chip architectures for SRNNs, there is a need to tackle the issue of allocating disproportionate resources, area, and power for implementing a fully-connected network, even when only a small percentage of those connections are used simultaneously.

- The proposed SpiCS-Net leverages the sparseness inherent in Spiking Recurrent Neural Networks (SRNNs) to achieve a resource-efficient Network-on-Chip (NoC) implementation. It accomplishes this by fine-tuning the on-chip concurrent connectivity requirement, resulting in area-efficient SRNNs while still enabling a fully connected network through the NoC.

Furthermore, as pointed earlier, it is imperative to address the distinct requirements for proximal and distant neurons on-chip. While the proposed circuit-switched scheme proves highly effective for facilitating connections among proximal neurons, it encounters limitations when attempting to scale the Network-on-chip (NoC) to connections between distant neurons. The metal interconnects associated with long-distance communication introduce RC-delay and vulnerable to Crosstalk leading to challenges such as signal degradation and difficulties with spike synchronization. To mitigate these issues, a hierarchical approach utilizing a packetized communication scheme between clusters of proximal neurons becomes essential. This approach involves packetizing spikes to preserve spike timing information, thereby enabling efficient spike transmission over longer distances. At the receiving neuron cluster, the packetized

spike can be reconstructed, ensuring the integrity of the transmitted information while facilitating effective communication between distant regions of the neuromorphic system. While hierarchical approaches have been previously explored in the design of neuromorphic architectures, such as those demonstrated by [78, 82], these prior works predominantly focused on utilizing packetized spike communication.

- We further propose a novel hierarchical hybrid Network-on-Chip (NoC) architecture designed to seamlessly integrate circuit-switched mechanisms, optimized for handling dense local connectivity, with packetized schemes ideal for globally sparse spike event communication. In contrast, our approach distinguishes itself by combining both circuit-switched and packetized strategies within the same architecture, thereby offering a versatile solution tailored to the unique demands of hardware implementations in neural networks. To the best of our knowledge, the utilization of a hybrid packet and circuit-switched architecture in the connection fabric of neuromorphic architectures has not been previously explored. This innovative approach opens up new avenues for optimizing communication efficiency, resource utilization, and scalability in neuromorphic computing systems.

Challenge 3: Online Learning technique for SRNNs on hardware

Online learning on hardware, particularly in the case of Spiking Recurrent Neural Networks (SRNNs), poses a formidable challenge for two primary reasons. First, the presence of recurrent connections poses significant challenges not only for the implementation of Network-on-Chip (as discussed in previous subsection) architectures but also for the training process, as it requires time-dependent weight updates that track previous sets of inputs for each neuron [108, 84]. This complexity can escalate rapidly, even with relatively small network sizes. Second, the nature of spiking data presents its own intricacies. Calculating errors in regression tasks can be challenging, given that most derivative errors tend to approach zero in the context of spiking data [83, 84].

Reservoir computing (RC) is an innovative bio-inspired learning technique that has emerged independently from two distinct perspectives: machine learning, exemplified by Echo State Networks (ESN), and brain-inspired Liquid State Machines (LSM) [67, 47]. At its core,

RC employs a fixed Recurrent Neural Network (RNN), mirroring the neural architecture of the brain, to facilitate the efficient processing of intricate temporal information by training only the feedforward connections in the Output Layer as shown in Fig. 1.2. This addresses a major challenge by obviating the need for training recurrent connections.

However, even after simplifying the learning process to a very large extent, several challenges still remain to be tackled for hardware implementation of online learning using Reservoir Computing. First, the nature of spike data, as mentioned earlier, renders computed derivatives in regression algorithms unusable [84]. Second, the implementation of an output readout layer necessitates a Multiply-and-accumulate (MAC) unit making hardware implementation extremely challenging. Most Reservoir Computing architectures proposed in the literature employ crossbars constructed using memristor devices for MAC implementation in the analog domain [93]. However, our work highlights that the variability of memristor devices is too high for the Output Layer in Reservoir Computers. This is due to the fact that the entire performance of the RC system depends on the accuracy of the Output Layer. Thus, while variability of memristors is useful when employed within the reservoir [30, 77], it becomes a problem for accuracy in the Output Layer [93].

- In this work, we have introduced an online hardware-based digital implementation of the Output Layer of Reservoir Computing system for spiking hardware. This approach omits the use of multipliers completely by using digital spikes that remain high when spike occurs and low when there is no spike making them not as momentary spike-like. This not only reduces power consumption on the chip by minimizing switching activity but also helps avoid the zero derivative when the switching activity is absent in some regions of SRNN reservoir. Additionally, the weight update factor is also implemented completely using digital bit-wise operations and the system employs just adders for a complete digital design of learning algorithm as well. This results in a digital output from the Output Layer of RC system.
- The proposed system further reduces the Output Layer circuitry by using the proposed network-on-chip (NoC) architecture in conjunction with dimensionality reduction.

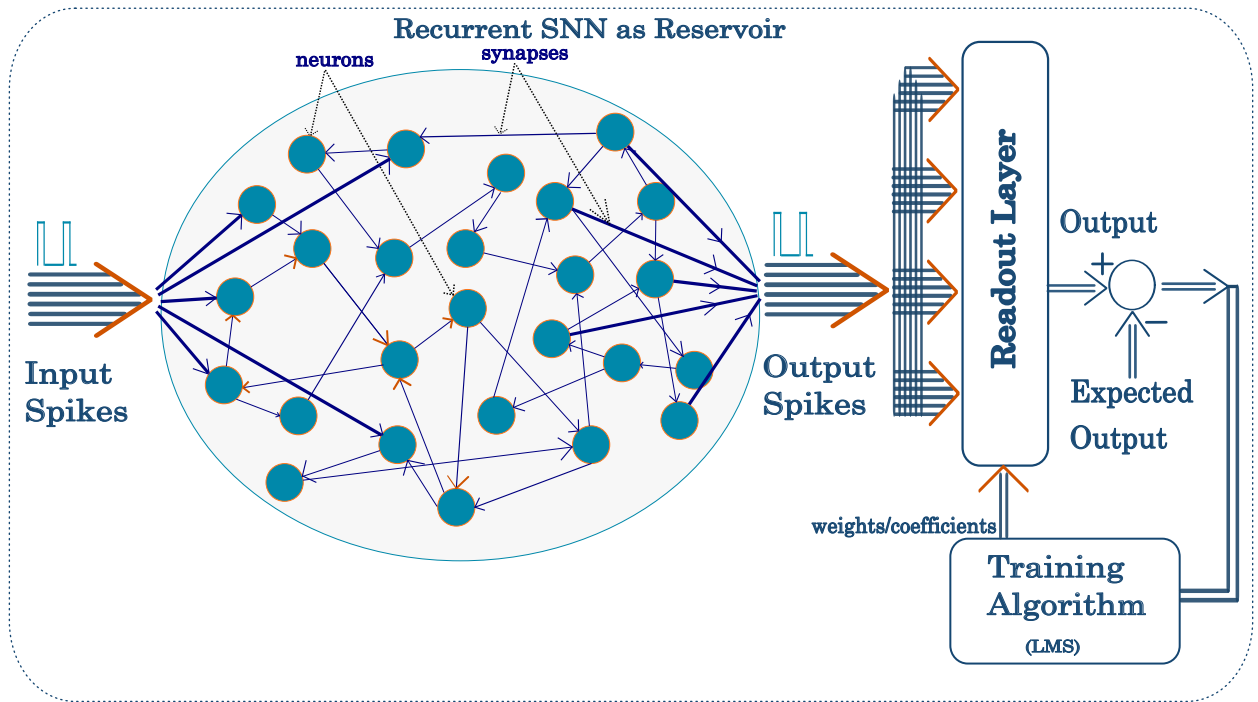


Figure 1.2: Depicting the architecture of a Spiking Recurrent Neural Network (SRNN)-based Reservoir Computing (RC) system, showcasing the dynamic network structure used for computational tasks.

- The system architecture is designed to implement online learning for the nonlinear channel equalization task which has an analog input at the receiver and a digital output of two bit symbols.

This approach is distinctive in its focus on area efficiency and its utilization of a simple gradient-descent based Least Mean Squares (LMS) algorithm designed to tackle the specific challenges posed by spike-based learning.

1.3 Navigating this document

Chapter 2 covers background and prior research on hardware implementations of memristive neuromorphic architectures with regard to memristor device modeling, network-on-chip design and online learning for Spiking Recurrent Neural Network architectures.

Chapter 3 covers device modeling of hafnium-dioxide (HfO_2) memristors specifically focused on variable switching dynamics of the device in the Low and High Resistance States (LRS and HRS) and *RESET* failure. Appendix A delves into the impact of memristor device variability on the overall efficiency, functionality, and reliability of SRNNs in the context of reservoir computing.

Chapter 5 and Chapter 4 cover Network-on-chip implementation of the circuit switching based network for spike routing within the SRNN. Chapter 4 discusses how sparseness of connectivity in the SRNN can be leveraged for resource-efficient NoC. Chapter 5 discusses hierarchical hybrid approach for routing spikes in a large-scale SRNN maintaining the integrity of "spikes-as-spikes" during local communication between neurons.

Chapter 6 outlines the resource-saving strategy for implementing memristive SRNN based Reservoir Computing (RC) on hardware for the non-linear channel equalization task. The system and circuit level simulations are completed using the model covered in Chapter 3 with the SRNN network-on-chip described in Chapter 5 and Chapter 4.

Finally, Chapter 7 very briefly summarizes the overall ideas with Conclusion followed by Future Work.

Chapter 2

Background

2.1 Neuromorphic Computing

The pursuit of brain-inspired computation, commonly referred to as neuromorphic computing, presents a compelling avenue for revolutionizing computational systems. Implementation on dedicated hardware that is trainable offers the exciting prospect of realizing low-power, resource-efficient, and remarkably effective computing paradigms that adapt to changing patterns in input data. In contrast, software-based neuromorphic computing approaches on von Neumann architecture struggle to fully capitalize on the extraordinary efficiency gains achievable through hardware-centric solutions. Neuromorphic computing specialized hardware not only promises to unlock unprecedented levels of computational efficiency but also opens doors to novel applications across diverse domains, from artificial intelligence to embedded systems, ushering in a new era of computing prowess.

Spiking Recurrent Neural Networks (SRNNs) represent a class of neuromorphic systems inspired by the brain’s mechanisms drawing upon the principles of data encoding, utilizing spikes as a fundamental unit of information representation, and incorporate recurrent connections to emulate the brain’s dynamic behavior. By mimicking the brain’s intricate interplay of spike communication between neurons, SRNNs offer a promising avenue for advancing the frontier of neural computation, paving the way for biologically plausible artificial intelligence systems. With their recurrent connections, as depicted in Fig. [2.1](#), SRNNs exhibit

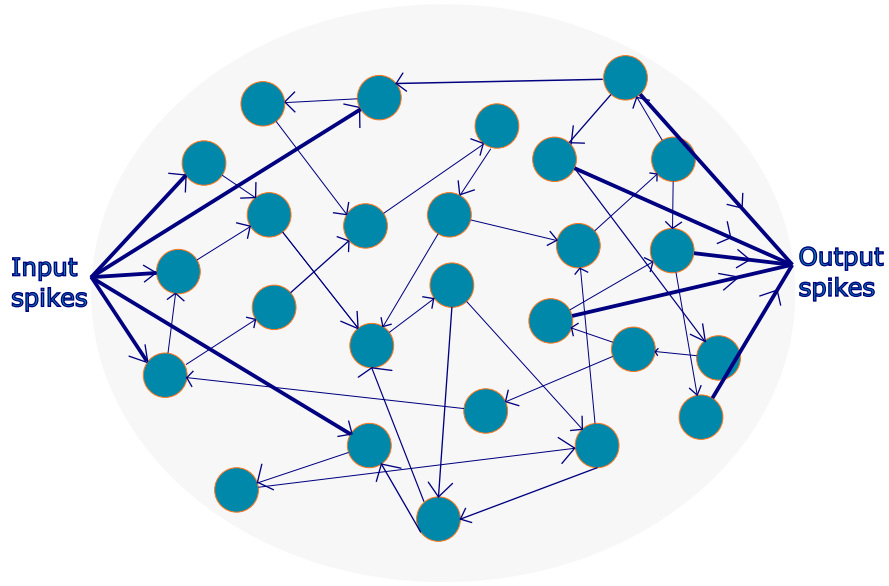


Figure 2.1: The Spiking Recurrent Neural Network (SRNN) displays recurrent connectivity among neurons. To implement these recurrent connections on hardware, it becomes imperative to accommodate any-to-any neuron connectivity within the SRNN. Additionally, the process of applying synaptic weights to spikes entails intricate multiplication operations. Both these aspects present significant challenges for hardware realization, as they markedly escalate resource demands, even for modestly sized SRNN configurations. Overcoming these challenges necessitates innovative hardware design strategies capable of efficiently managing the complexities associated with neuron connectivity and synaptic weight computations, thereby enabling the practical deployment of SRNNs in real-world applications.

a distinctive capability to process spatio-temporal data [47, 67, 93], distinguishing them from conventional artificial neural networks which lack temporal feature processing capabilities. Additionally, the utilization of spike data inherently facilitates low-power computations, as information is encoded through the precise timing and characteristics of "single-bit" spikes rather than multi-bit data representations [121]. This intrinsic efficiency further underscores the appeal and potential of SRNNs in diverse computational tasks for applications where power efficiency and temporal processing are of significance.

2.2 Neuromorphic System: Building Blocks

2.2.1 Neuron

The Leaky Integrate-and-Fire (LIF) neuron model, widely used in computational neuroscience and neuromorphic system design, is a fundamental block designed to emulate the behavior of biological neurons [32, 46, 67]. Fig. 2.2 shows an abstract image of neurons computing with voltage spikes in the brain. The LIF model attempts to model this behavior by representing crucial neuron parameters like membrane potential, firing threshold, and leak factor corresponding to each neuron as parameters in the model. When the accumulated voltage at the input of the neuron raises the membrane potential V above a certain threshold value V_{th} , the neuron generates an output spike, and its membrane potential is reset to a resting value V_{mem} . Consequently, the neuron may enter a refractory period during which it is unresponsive to incoming stimuli.

This work focuses on hardware implementation of neuromorphic systems and thus utilizes a Mixed-Signal LIF neuron circuit. This Mixed-Signal LIF neuron comprises of both analog and digital circuitry to accurately model the dynamics of neuronal membrane potential and spike generation as represented in Fig. 2.3 [22]. The circuit consists of an analog integrator that accumulates incoming synaptic currents and a threshold comparator that generates digital output spikes when the membrane potential exceeds a certain threshold. The analog integrator is implemented using operational amplifiers (opamps) and capacitors and it simulates the leaky integration of synaptic inputs over time. This integration process

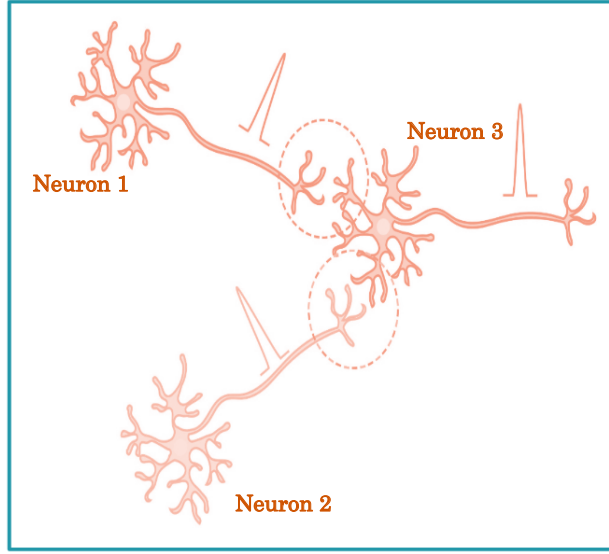


Figure 2.2: Neurons communicate through synaptic connections by transmitting voltage spikes. In this abstract representation, spikes originating from Neuron 1 and Neuron 2 are relayed to Neuron 3. Upon reception, Neuron 3 integrates the incoming spikes at its input. When the accumulated potential surpasses a certain threshold, Neuron 3 generates an output spike. This neuron behavior can be modeled as Leaky Integrate-and-Fire model.

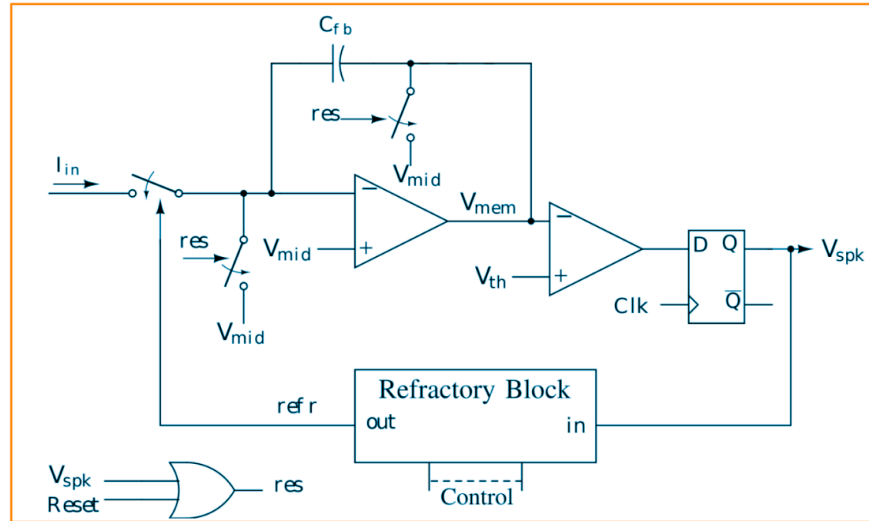


Figure 2.3: Mixed-Signal LIF Neuron circuit used in the system level design of Spiking Recurrent Neural Network (SRNN) takes analog signal at the input and generates digital output spikes [22]. The mixed-signal LIF neuron circuit integrates analog and digital components to replicate the essential characteristics of biological neurons, including membrane potential dynamics, threshold-based spike generation, and refractory behavior.

is governed by the neuron’s membrane potential and time constant, which determines the rate at which the membrane potential decays in the absence of input.

In addition to the analog integrator, the circuit includes digital logic for spike generation and reset functionalities. When the membrane potential surpasses the predefined threshold voltage, indicating the arrival of a sufficient synaptic input, the threshold comparator triggers a digital spike generation process. The threshold of the neuron is defined using an external analog signal to the second comparator, V_{th} . This is achieved by the second opamp in conjunction with a flip-flop circuit that generates a digital spike signal, representing the neuron’s output spike.

Following spike generation, the circuit incorporates a programmable refractory period mechanism to enforce a brief period during which the neuron is unresponsive to incoming synaptic inputs. This refractory period mimics the biological neuron’s post-spike recovery phase and helps regulate the neuron’s firing rate.

2.2.2 Memristive Synapse

Incorporating synapse-like functionalities into Spiking Recurrent Neural Networks (SRNNs) requires hardware support for multiplication operations. In the realm of SRNNs, the magnitude and sheer volume of synaptic connections can render digital multiplier circuits cumbersome. Memristors are emerging devices touted as the missing fundamental circuit element [25] as depicted in the Fig. 2.4. These devices offer an low power analog computing solution to this multiplier challenge by employing computations based on Ohm’s law [105, 61]. A memristor is a two-terminal electrical device that demonstrates non-linear characteristics in its connection between electric charge and magnetic flux linkage (Fig. 2.4). The theory behind this innovative device was introduced by Leon Chua in 1971 [24], and a functional device was successfully demonstrated in 2008 [105]. Thanks to their compact size, seamless integration with Complementary Metal Oxide Semiconductor (CMOS) technology [52], and the ability to program resistance states [27], memristors open up possibilities for resource-efficient analog computations. By harnessing these inherent properties of memristors, SRNNs can achieve

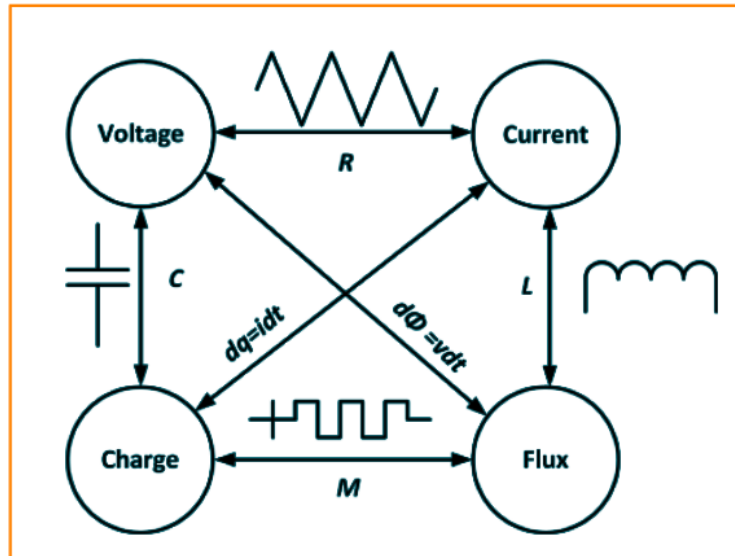


Figure 2.4: The memristor 'M' is represented as a fundamental circuit element, embodying the Charge-Flux relationship.

efficient and scalable synaptic functionality, paving the way for more streamlined hardware implementations capable of handling complex neural computations effectively.

The memristive synapse utilized in this work is shown in Fig. 2.5. the circuit utilizes CMOS (Complementary Metal-Oxide-Semiconductor) transistors to implement four fundamental operations on the memristor: *FORM*, *SET*, *RESET*, and *READ*. The initial state of a fabricated memristor is an *UNFORMED* state, characterized by a high resistance in the range of tens of $M\Omega$. Upon *FORMING*, the resistance level transitions to tens of $K\Omega$ as a conduction filament is established. The *FORMING* process is a one-time initialization step for the memristor device. To program the memristor to different resistance levels in the Low Resistance State (LRS), the compliance current is controlled using the *V_SET_READ* signal. This involves the *RESET* operation, which transitions the memristor back to a High Resistance State (HRS) before programming the resistance in the LRS to the desired values.

The Hafnium Oxide (HfO_2) memristors utilized in this work offer non-volatility and high switching endurance [62, 61] along with incremental resistance programming with nearly 8-10 separable resistance levels in the Low Resistance State (LRS) [63]. This makes them a suitable choice for designing low-power neuromorphic architectures. However, addressing challenges related to memristor reliability and variability is essential to fully leverage these advantages in SRNNs. A crucial initial step is integrating this variability and device failure into a SPICE-compatible circuit simulation model. This model facilitates simulating and analyzing the behavior of memristor devices under diverse conditions, enabling a better understanding and mitigation of the impact of variability and failure on system performance. Such modeling is indispensable for developing reliable system-level designs capable of accommodating the real-world behavior of these devices.

2.2.3 Network-on-chip (NoC)

The Network-on-Chip (NoC) in Spiking Recurrent Neural Network (SRNN) hardware serves the critical function of facilitating efficient communication among neurons, enabling the exchange of spikes (neuronal activation signals) within the network. Given that the core functionality of the spiking neural network is centered around the network structure between neurons, establishing an effective communication infrastructure is paramount for spike

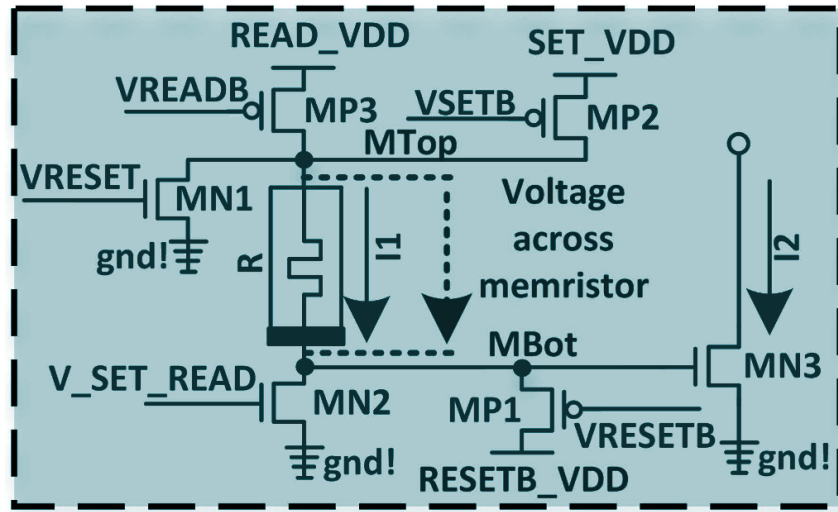


Figure 2.5: Memristive synapse [117] circuit showing MOSFET(metal-oxide-semiconductor field-effect transistor) controls for resistance programming

communication, synchronization and the processing of spatio-temporal information. Recurrent connections, where neurons feed their output spikes back into the network, are fundamental to SRNN operation. NoCs play a pivotal role in implementing all these connections by enabling bidirectional communication paths between neurons, thus facilitating effectively duplex connections. Additionally, preserving spike timing information throughout communication is crucial for maintaining the temporal dynamics and computational properties of SRNNs. Therefore, NoCs must ensure accurate preservation of spike timing during the communication process to uphold the integrity of the network’s functionality.

To meet the necessary criteria, the Network-on-Chip (NoC) must possess scalability and offer flexible communication pathways to accommodate various network densities and configurations. This scalability is crucial for adapting SRNN hardware to different quantities of neurons and synaptic connections. Additionally, optimizing hardware resources becomes imperative due to the exponential rise in wiring complexity associated with direct point-to-point connections between neurons as the network size increases.

Consequently, a significant challenge in designing a Spiking Recurrent Neural Network (SRNN) that is hardware-reconfigurable lies in the constraints imposed by connectivity requirements [91]. Hardware designs require establishing physical wired connections between nodes or neurons. Therefore, it’s crucial to address how this reconfigurable connectivity can be effectively achieved while ensuring any-to-any neuron connection.

Numerous efficient methods have been proposed in the literature for implementing network-on-chip for reconfigurable Spiking Recurrent Neural Networks (SRNNs). Among these, the Address Event Representation (AER) approach stands out as a prominent solution recognized for its scalability, efficiency, and reconfigurability [14, 82, 111, 121]. The AER approach utilizes address events to represent spikes in the form of data packets, where these packets contain information about the source and destination neurons’ addresses [82, 20, 41, 111, 121]. A depiction of AER in a mesh network architecture is presented in Fig. 2.6. This figure shows the spike path for communication between neurons N1 and N2.

While the AER approach proves highly efficient for large-scale neuromorphic systems, it presents an asymmetric resource overhead when for small-sized networks consisting of only a few hundred neurons physically proximate to each other on-chip [87]. This overhead arises

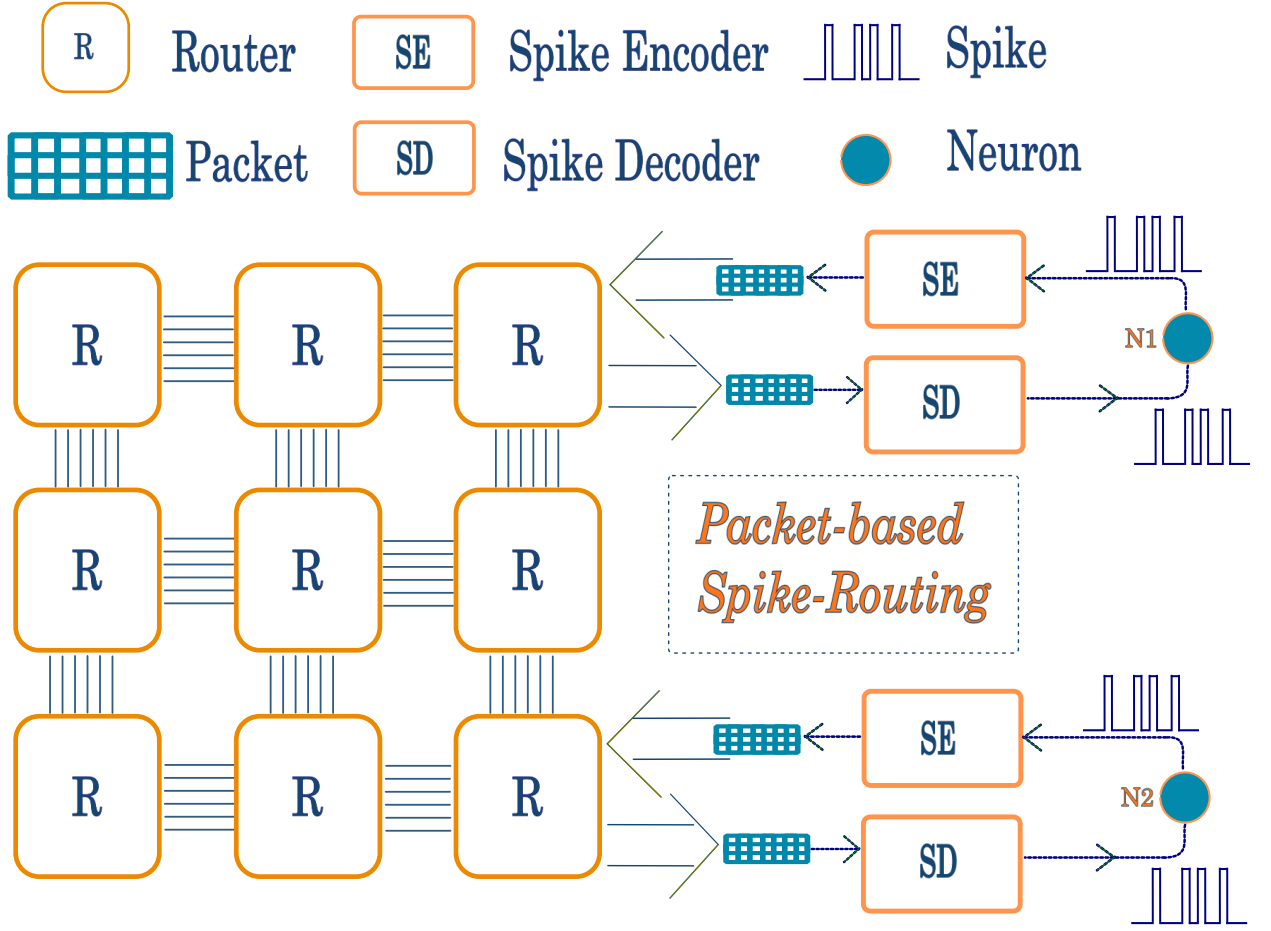


Figure 2.6: The Mesh NoC architecture demonstrates the AER Packet-based spike communication approach. This method involves a recurrent connection loop between neurons N1, N2, and N1. This loop entails spike communication as follows: N1 emits a spike, which is encoded to a multi-bit digital data packet and routed through a series of routers before reaching N2, and vice versa for spike from N2 to N1 [90].

from additional circuit components serving various functions. These include sensing and sampling spikes from individual neurons, converting sampled spikes into multi-bit address event packets, and using multiple routers to manage packet transmission via a shared bus within the NoC [113, 114]. Even basic routers must incorporate features like arbitration, collision detection, and avoidance to handle scenarios of simultaneous spike arrivals. On-chip memory is also essential to store routing tables for source or destination routing, with memory size depending on the number of bits in the address event packets and the required network connectivity. Moreover, each neuron requires a spike decoder to convert packets back into spikes, and in certain cases, a pulse shaping circuit may be necessary depending on the neuron type.

Consequently, the energy efficiency inherent in spiking architectures, stemming from single-bit voltage spikes, is significantly reduced when binary data is employed [90]. As a result, for applications where area and power efficiency is paramount, the AER-based packet communication NoC is a non-optimal solution because it entails significantly higher area and power consumption.

Other significant instances of neuromorphic hardware NoCs encompass techniques like point-to-point synapse switching with multiple-input multiple-output (MIMO) orthogonal frequency division multiplexing (OFDM) [103] and 3D IC interconnects [11, 44]. To mitigate the challenges of implementing high-density NoCs, certain neuromorphic systems opt to restrict connectivity [74]. However, many NoC designs fail to fully consider the requirements imposed by the structure and properties of neuromorphic networks. Consequently, there is still a lack of consensus regarding the preferred properties of the NoC in terms of the expected characteristics of the SRNNs [121, 36, 45, 50, 95]. Hence, it is imperative to explore an alternative approach that considers the brain’s structure and connectivity requirements, aiming to alleviate resource-intensive aspects and provide a more efficient method for implementing SRNNs on hardware. This exploration involves circuit-switching-based strategies, more streamlined data representation, and other innovative methods to enhance the efficiency of SRNN implementations.

2.2.4 Circuitry for Online Learning

Systems equipped with online learning capabilities can respond to real-time changes, such as shifting weather conditions in autonomous vehicles, evolving chaotic patterns for a robot in disaster struck environment, or adjusting to different wireless channel characteristics in RF receiver systems [5]. This adaptability is crucial for ensuring optimal performance and maintaining safety and efficiency in such contexts. Consequently, online learning on hardware offers a dynamic and adaptive approach to machine learning that is well-suited to applications with changing conditions and those that require continual improvement and versatility. It empowers systems to evolve, learn from experience, and perform at their best in complex and evolving environments.

Spiking Recurrent Neural Networks (SRNNs) embody dynamical properties owing to their structure. These properties arise from the nature of connectivity, resulting in what is known as an "echo" property [47]. This echo signifies a time-extended response of SRNN to an input, thus characterizing SRNN as a dynamical system. Due to these properties, training an SRNN on hardware is a highly intricate task due to the emergence of infinite states for a specific node or neuron when the network is unfolded in time [83, 84]. Furthermore, the intricate web of inter-dependencies resulting from the recurrent connections exacerbates the mathematical optimization challenge, making the process complex and cumbersome. One of the primary advantages of online learning in hardware is its applicability to dynamic and evolving environments [99].

There have been many approaches for training Recurrent Neural Networks (RNNs) in the literature. One such approach is Long Short Term Memory (LSTM), which does reduce the training effort compared to traditional RNNs [102]. Other training approaches for RNNs like Back Propagation Through Time (BPTT) [15] and eProp [84] are able to provide high accuracy in data processing but still suffer from high training complexity from hardware implementation perspective since they involve intricate operations and demand substantial memory resources. Nevertheless, when compared to the Reservoir Computing (RC) approach, all other training approaches still retain a significantly high level of complexity [76]. Reservoir Computing (RC) as a computing paradigm has emerged from two distinct perspectives, as

outlined in the seminal works of Jaeger (2001) and Maass (2002) [47, 67]. In 2001, Jaeger introduced "Echo State Networks (ESN)" from an machine learning engineering perspective, offering an alternative approach for training computational systems based on Recurrent Neural Networks (RNN) [47]. In parallel, Maass, in 2002, explored a task-agnostic, biologically inspired model of computation termed "Liquid State Machines (LSM)." LSMs are brain-inspired dynamical systems characterized by nodes that continuously interact and possess recurrent connections [67].

Jaeger's work [47] approached echo state networks from a machine learning perspective and devised novel algorithmic methods for training recurrent neural networks. Training Spiking Recurrent Neural Networks (SRNNs) poses challenges due to the intricate computations within networks of interconnected neurons. The difficulty is magnified when networks confront rapidly changing inputs, as their dynamics become hard to predict. Jaeger introduced the memory aspect of recurrent neural networks, termed "echo," arising from inputs stored in reservoir states. The echo's strength relates to the input's recency; older inputs have a diminishing effect. Jaeger's method entails training network-to-output weights while maintaining fixed echo states, achieving the desired output via supervised learning. It's essential to note that Jaeger's recurrent neural networks function as discrete-time computational devices, conducting computations each clock cycle and retaining state for a single cycle, rendering them discrete-time dynamical reservoir systems. This approach provides insights into addressing challenges posed by SRNNs' complex computations and memory dynamics when handling rapidly changing inputs.

In contrast to Jaeger's perspective, Maass was primarily motivated by the pursuit of designing biologically inspired computational architectures [67]. His focus was on spiking recurrent neural networks that operated as a continuous evolution of reservoir states in an analog manner. This continuous operation meant that the reservoir states, or neuron/node states, evolved from one to the next without being static at any point in time. This characteristic gave rise to the name "liquid state machine (LSM)." These systems exhibit inherent asynchrony and are often characterized by these "liquid" states, implying transience. This operation is akin to analog processes in electronic circuits and reflects real-time computing. Maass argued that stable internal states were not a prerequisite for generating a stable output.

Transient internal states, owing to the high dimensionality of the dynamical system, could be effectively transformed by readout neurons into stable target outputs. This reinforces the idea that the readout layer, which interprets the reservoir’s internal state, can be trained for specific operations, highlighting the adaptability and flexibility of LSMs.

Reservoir computing is indeed regarded as a biologically inspired framework, owing to its incorporation of recurrent neural connections [97, 110]. Considering that the mechanisms governing learning in the brain remain somewhat unclear, the readout layer presents a plausible alternative for achieving learning within this framework. The inherent simplicity of Reservoir Computing’s training approach can be seamlessly integrated into neuromorphic systems, fostering the creation of adaptable, energy-efficient, and highly parallel computing structures. These advantages are further magnified when contemplating hardware-based online learning. Clearly, Reservoir Computing has emerged as a highly practical and compelling solution for implementing online learning on neuromorphic hardware [60, 55, 73, 34]. There are multiple reasons for this including,

- Reservoir computing does not involve training recurrent connections.
- The training of the output layer in reservoir computing follows a linear transformation.
- In a reservoir computing system, it is possible to achieve robust and reliable results even with a relatively small reservoir size.
- "Sparseness" in connectivity of the Recurrent Neural Networks (RNNs) in reservoirs are essential for ensuring the generalizability of solutions to previously unseen inputs. The introduction of sparseness also paves the way for reduced resource consumption on hardware, making the system more efficient.
- Using the same reservoir for training multiple output layers separately, each serving different functions, offers resource and area savings, enhancing overall efficiency.
- While memristors introduce challenges in system design due to their variable switching behavior, their integration into the reservoir enhances its rich dynamical states. Reservoir computing is uniquely equipped to not only accommodate but also capitalize

on the inherent variability in memristor-based SRNNs, enabling effective processing of non-linear temporal data with the added benefit of low area and power consumption.

The training approach in RC is notably less resource-intensive, making it a strong candidate for hardware-based implementations. Approaches such as the Moore-Penrose inverse offer a promising solution for analog systems by providing an approximate solution to linear equations, facilitating the implementation of linear regression on hardware [106]. This method leverages the pseudoinverse technique to address the least squares problem. Additionally, the least mean square-based gradient descent method provides a simple yet resource-efficient approach to training the output layer of a digital RC system. This efficiency is achieved through the reservoir’s transformation of inputs into a higher-dimensional space, rendering the inputs linearly separable.

However, despite its potential, reservoir computing primarily remains a software-based approach due to the computational demands of multiplication operations involved in the training algorithm [76]. Consequently, implementing it in hardware is still costly, as highlighted in [58].

Chapter 3

Compact Device Model for HfO₂ memristors

3.1 Background

A device model mimics the physical device behavior and is used in simulation environments to facilitate the integration of a device into a circuit or system, enabling the analysis of the overall system behavior. Undoubtedly, the necessity for developing a comprehensive physics-based SPICE-compatible model with tunable physical parameters is paramount for any electrical device. Such a model should encompass realistic bounds on its operating range and accurately replicate experimental behavior without compromising its utility in large-scale circuit simulations [115]. However, when working with emerging devices, especially those that are still evolving, it becomes crucial to use empirical models [92]. Empirical models offer several advantages, including rapid development and simplicity in usage. This holds particularly true for empirical models that derive their parameters directly from observable behavior in experiments. This means that empirical models can be quickly adapted to capture the evolving characteristics of emerging devices, making them highly practical for designing circuits based on such devices.

The growing popularity of memristors for neuromorphic and other applications has spurred the development of numerous SPICE-compatible memristor models. These models encompass

a range from straightforward behavioral representations to intricate device physics-inspired models [13, 85, 104, 123, 122]. Most memristor models in literature assume specific conduction and switching mechanisms that make them applicable for specific device characteristics [85, 104]. Moreover, models in the literature often lack parameters observable in experiments, resulting in non-translatable models [3]. However, the same device may exhibit multiple such switching mechanisms. Moreover, there is ongoing debate regarding the maturity and contents of these switching models [122, 123], which often involve numerous empirical parameters. Further, it's crucial to acknowledge that many of these models suffer from inaccuracies when describing device behavior, and they often prove to be inefficient in simulating circuits due to convergence problems [115]. This issue is particularly concerning, as a significant portion of both empirical and physics-based models will typically operate within specific, realistic operational boundaries. These boundaries are essential to ensure that simulators can reliably converge to a solution [115].

In the context of HfO_2 memristors, which offer high endurance and non-volatile resistance states [61], the devices exhibit stochasticity and resistance variability in both their Low Resistance State (LRS) and High Resistance State (HRS). This inherent variability places limitations on their practical application and overall performance [63]. Moreover, it's crucial to note that the variability observed in memristors is not purely stochastic; rather, it fluctuates with the changing compliance current flowing through the device and the applied voltage across the memristor [63]. This recently documented behavior suggests the possibility of analog resistance programming in memristors, a phenomenon that has not yet been modeled.

Moreover, modeling device failure is also crucial for design of robust neuromorphic systems. It has been observed that the extensive utilization of parameters can result in the emergence of non-ideal device regions, which may lead to potential short- and long-term degradation of the Memristor Conduction Filament (CF). This degradation eventually culminates in failure of the memristor or a phenomenon known as "memristive sticking," wherein a memristive device becomes incapable of transitioning from one resistance state to another [17]. Among various types of failures that can occur in memristive synapses, *RESET* failure takes on particular significance due to its severity, rendering the device unswitchable. Various research groups have delved into the analysis of *RESET* failure [64, 54, 23, 21, 8, 92] underscoring the

importance of modeling this behavior for circuit simulations. Moreover, it's worth noting that the memristive synapse utilized here [117] will be rendered ineffective if a successful *RESET* operation is unattainable.

This emphasizes the pressing requirement for a comprehensive Verilog-A model for HfO₂ memristors that incorporates resistance state variability along with *RESET* failure, providing designers with the capability to validate and explore corner cases that might otherwise lead to design and device failures. It is imperative to develop a versatile and comprehensive compact model that not only aligns with experimental data but also seamlessly works with SPICE-level circuit simulation tools. By doing so, we can fully unlock the versatile potential of these remarkable devices.

3.2 Capturing Variable Dynamics of HfO₂ Memristors

The proposed model is based on the following equation [3] that characterizes the rate of change in memristance when the device crosses its positive and negative switching thresholds. The determination of Low Resistance State (LRS) and High Resistance State (HRS) values relies on the instantaneous current and voltage across the memristor based on mathematical relationships that are derived from experimental observations of HfO₂ memristors fabricated at SUNY Polytechnic Institute [63]. It should be noted that the instantaneous current $I(t)$ through the memristor is a function of V_g , represented in Fig. 3.1 as the gate voltage applied at the MOSFET device. To visually illustrate the functionality of the proposed model, the state transition depiction is presented in Fig. 3.2.

And, the mathematical description for the proposed model is given by the following equations,

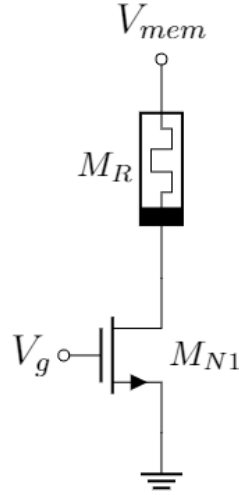


Figure 3.1: Schematic diagram of a 1T1R (One Transistor One Resistor) configuration, illustrating the integration of an NMOS (N-channel Metal-Oxide-Semiconductor) transistor to regulate the current flow through the memristor device.

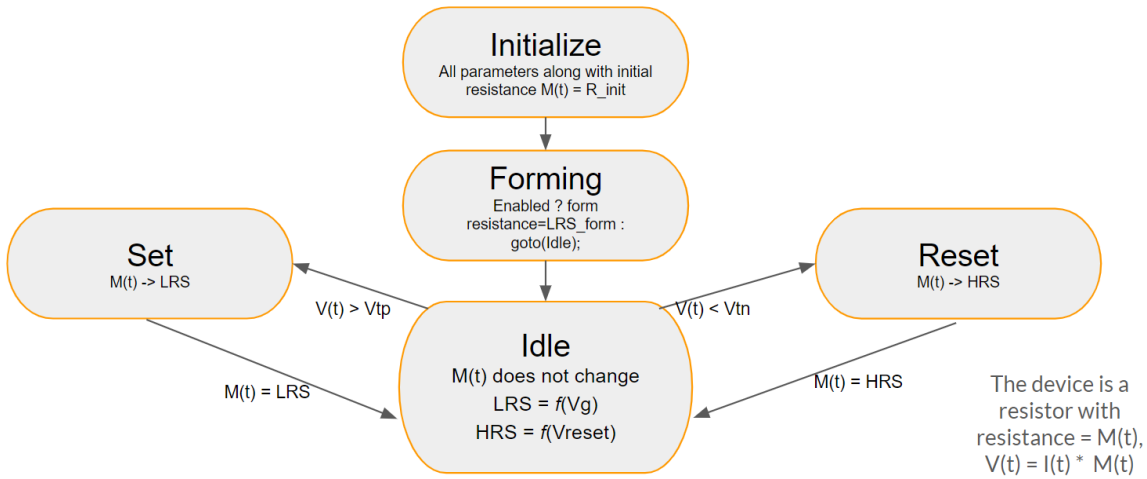


Figure 3.2: Flowchart illustrating the Verilog-A model, depicting unformed and formed states for the memristor. Switching between resistance values, Low Resistance State (LRS) and High Resistance State (HRS), occur during *SET* and *RESET* operations, respectively, after forming.

$$\frac{dM}{dt} = \begin{cases} C_{LRS} \frac{\Delta r}{t_{swp}} \left(\frac{(V(t)-V_{tp})}{V_{tp}} \right)^{\alpha_{LRS}} W_{LRS} M(t), & V(t) > V_{tp} \\ C_{HRS} \frac{\Delta r}{t_{swn}} \left(\frac{(V(t)-V_{tn})}{V_{tn}} \right)^{\alpha_{HRS}} W_{HRS} M(t), & V(t) \leq V_{tn} \end{cases} \quad (3.1)$$

$$LRS = f(V_g) = f(I(t)) = 2.44e11I^2(t) - 1.32e08I(t) + 1.95e04 \quad (3.2)$$

$$HRS = f(V_{reset}) = 1.5e05V^3(t) + 3.83e05V^2(t) - 1.22e08V(t) + 8.17e04 \quad (3.3)$$

In the proposed model, $M(t)$ represents the instantaneous memristance or resistance value of the device, while $I(t)$ and $V(t)$ denote the instantaneous current and voltage across the device, respectively. During the transition to the Low Resistance State (LRS) in the *SET* operation, the variable resistance (LRS) is determined as a function of $I(t)$, and the memristor's instantaneous resistance switches to this calculated LRS using Eq. (3.2) each time the device transitions to LRS. Similarly, in the *RESET* operation, the High Resistance State (HRS) is determined as a function of $V(t)$ using Eq. (3.3). These equations represent fitted data obtained from experimental observations. In Eq. (3.1), V_{tp} and V_{tn} represent the positive and negative switching thresholds, respectively, which can be determined from experimental data. Parameters t_{swp} and t_{swn} denote switching times, indicating the transition time from HRS to LRS and vice versa. The parameters C_{LRS}/C_{HRS} and $\alpha_{LRS}/\alpha_{HRS}$ are fitting parameters derived from experimental data for $\frac{dM}{dt}$ versus applied voltage [2]. Δr represents the instantaneous change in resistance from the previous time step. W_{LRS} and W_{HRS} are window functions utilized to smooth the $\frac{dM}{dt}$ curve near the memristor state transitions from LRS to HRS and vice versa. The window functions utilized in the model are represented by W_{LRS}/W_{HRS} and are defined as,

$$W_{LRS} = \left(\frac{1}{1 + e^{-\frac{1}{\beta_{LRS} \Delta r} (-M(t) + \theta_{LRS} LRS)}} \right) \quad (3.4)$$

$$W_{HRS} = \left(\frac{1}{1 + e^{-\frac{1}{\beta_{HRS/LRS}\Delta r}(-M(t) + \theta_{HRS/LRS}HRS)}} \right) \quad (3.5)$$

where the transition occurs when $M(t)$ equals the $\theta_{HRS/LRS}$ value, resulting in absolute magnitude value equal to 0.5 for the window function $W_{HRS/LRS}$. The term $\beta_{HRS/LRS}\Delta r$ specifies the gradient of the sigmoid at transition. A small $\beta_{HRS/LRS}\Delta r$ results in a gradual transition, while a large value leads to a steeper transition. Since the rate of change of memristance and the switching dynamics for LRS and HRS switching differ significantly, distinct window functions are used for each. The parameters $\beta_{HRS/LRS}$ and $\theta_{HRS/LRS}$ are fitting parameters for the window functions. Figure 3.3 illustrates the sigmoid window functions plotted as a function of the instantaneous resistance value $M(t)$. Additionally, Figure 3.4 presents the I-V plot for the proposed model.

As previously mentioned, for HfO₂ memristors, the resistance values in the Low Resistance State (LRS) are influenced by the applied gate voltage V_g in Fig. 3.1 during the transition from the High Resistance State (HRS) to LRS [63]. Moreover, this mechanism allows for better control over the memristor resistance near LRS compared to HRS, resulting in much more reproducible behavior. To capture the behavior described in Eq. (3.3) and Eq. (3.2), polynomial fitting was employed to derive a continuous function representing the relationship between the Low Resistance State (LRS) and the applied gate voltage V_g (Eq. (3.2)). The order of the polynomial was kept as low as possible to maintain model simplicity. Numpy [38] was utilized for polynomial fitting of the average experimental data, utilizing a second-order polynomial function for both LRS and its corresponding gate voltage (V_g), as well as for the High Resistance State (HRS) and its associated reset voltage (V_{reset}). Ensuring the continuity of the fitted function is crucial to prevent convergence issues. Simulation results indicate that the differentiability of this function does not significantly affect convergence. Additionally, the relationship between V_{reset} and HRS, as described by Eq. (3.3), is established using polynomial fitting of experimental data.

This model effectively captures the variability in the Low Resistance State (LRS) and High Resistance State (HRS) memristance values, partly attributed to the dependence of

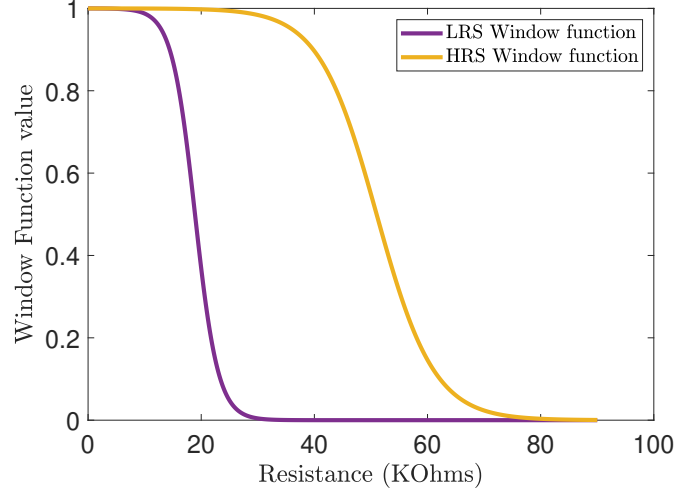


Figure 3.3: The window function utilized for Low-Resistance State (LRS) and High-Resistance State (HRS) switching is a sigmoid function. The sigmoid functions employed for LRS and HRS switching incorporate two defining values: the midpoint and the slope, which determine the shape and characteristics of the transition. The midpoint is calculated based on the polynomial fit HRS and LRS values from Eq. (3.2) and Eq. (3.3). The slope for LRS is higher than HRS due to faster transitions during *SET* when compared to *RESET* operation [92].

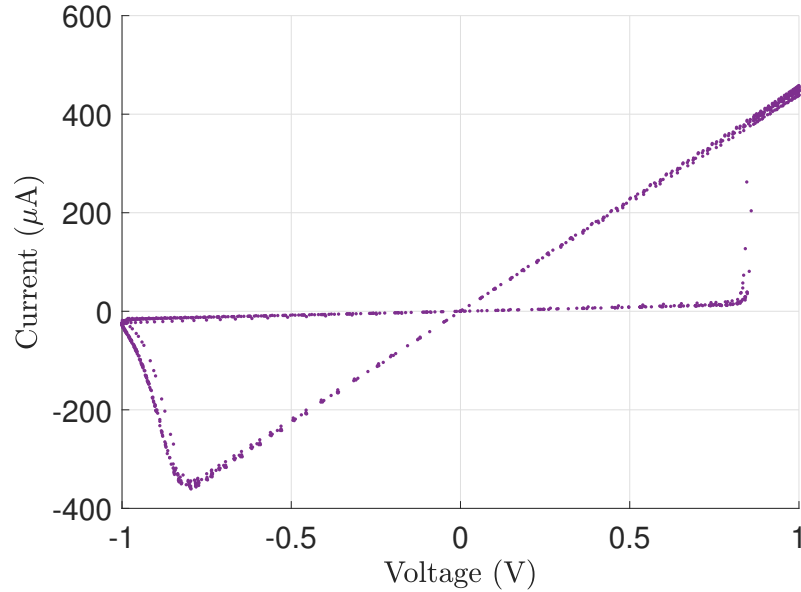


Figure 3.4: I-V plot of the proposed memristor model illustrates multiple switching cycles between the Low-Resistance State (LRS) and High-Resistance State (HRS). This plot captures the dynamic behavior of the memristor as it undergoes switching events, showcasing the characteristic hysteresis loop associated with memristive devices [92].

LRS/HRS on different compliance currents V_g/V_{reset} , respectively, sampled at specific time steps during device switching in the simulation.

Stochastic behavior is introduced into the model through various parameters in Eq. (3.1), which are defined as a range of values with a mean and standard deviation based on experimental data. These parameters follow a Gaussian distribution, with the LRS/HRS value calculated as a distribution around the resistance value evaluated from the polynomial-fit function.

3.3 *RESET* Failure in HfO₂ Memristor

3.3.1 Device physics behind *RESET* failure

The HfO₂ memristor functions as a filamentary device, with the movement of oxygen anions being triggered by an applied potential. The movement of oxygen anions predominantly occurs during device switching events but also takes place when shorter impulse signals are applied. Operation parameter adjustments to the movement of oxygen anions can alter the structure of the device. Material interface effects also influence the switching performance when the device is under higher stress. This stress can arise from various factors such as increased voltage, current, or temperature. Specifically, in the context of this discussion, applying voltage stress above a critical threshold can result in *RESET* failure. According to the hypothesis outlined in [28], the rise in electric field above a threshold across the device during *RESET* operation causes defects to be injected from the Bottom Electrode-Switching Layer (BE-SL) interface into the Conduction Filament (CF) and Switching Layer (SL) [31, 8]. This phenomenon results in the convergence of Low-Resistance State (LRS) and High-Resistance State (HRS), as depicted in Fig. 3.5. Additionally, during device fabrication, other impurities, and defects such as Ti or N may also infiltrate the SL. Ultimately, the rate of defect movement from the BE-SL interface into the conduction filament surpasses the rate of defect movement from the filament to the Top Electrode (TE) under the application of large electric field or prolonged switching activity. This leads to a considerable enlargement of the CF cross-section and consequent switching failure [31, 8]. An alternative hypothesis is the interaction between

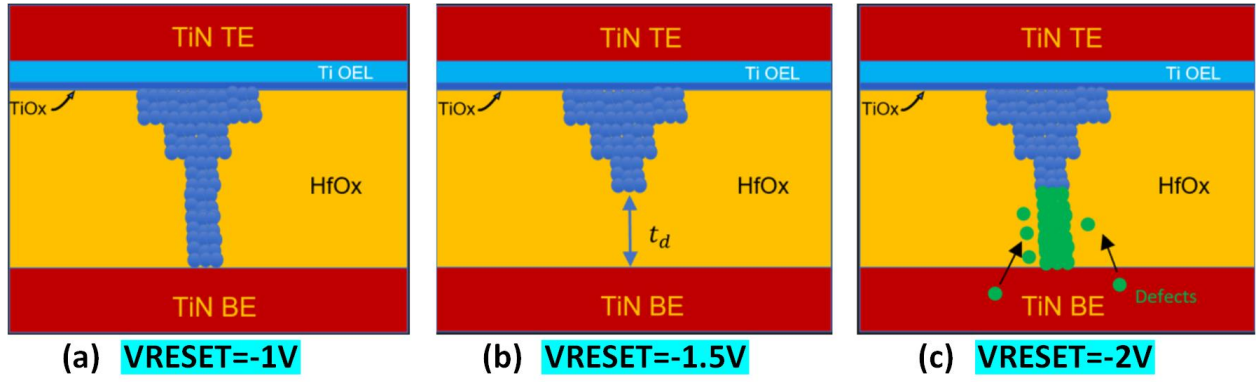


Figure 3.5: Cross-sectional schematics depict the impact of various *RESET* voltages on filament behavior. (a) Demonstrates the impact on a FORMED memristor when the voltage across the branch 1T1R (1 transistor 1 memristor), V_{RESET} , is -1 volt, lower than the *RESET* threshold V_{tn} in absolute value. Insufficient V_{RESET} results in an intact filament, indicating no *RESET* operation occurred. (b) Illustrates the *RESET* operation occurring when V_{RESET} crosses *RESET* threshold at $V_{RESET} = -1.5$ volts. The filament experienced breakdown and *RESET* occurs. (c) Depicts a *RESET* failure scenario observed when V_{RESET} is -2 volts, much higher than *RESET* threshold V_{tn} . The elevated V_{RESET} causes defects to migrate from the bottom of the memristor, initiating the formation of a reverse filament from the bottom. Consequently, the operation behaves akin to a *SET* operation rather than a *RESET* operation [28].

the BE-SL interface, where the creation of a TiOxNy layer between the two layers could lead to device failure [120, 107]. This additional layer could enable migration of additional oxygen anions to the SL during *SET* leading to change in LRS resistance levels. Based on the increase in stress on the device, the resulting oxygen bond with the TiN BE could be strong enough to cause device failure.

3.3.2 Experimental Results from 1T1R devices

The experimental results illustrated in Fig. 3.6 show the effect of varying *RESET* voltages over multiple switching cycles [64]. The median High Resistance State (HRS) at each *RESET* voltage is depicted in blue. When the *RESET* voltage is insufficiently low, a complete *RESET* is not achieved, evident in the range from -1.3V to -1V where the memory window is inadequate. The optimal range, approximately -1.5V, ensures a thorough *RESET*. Notably, the graph highlights the threshold at which excessively low *RESET* voltages lead to *RESET* failures, resulting in a drastic resistance drop to around 1.5k Ω . Additionally, the Low Resistance State (LRS) value, illustrated in orange, demonstrates the diminishing memory window as device operation parameters approach the threshold of *RESET* failure.

3.3.3 Incorporating *RESET* Failure into the Verilog-A Model

The memristor model described in Eq. (3.1) provides the mathematical relationship for the change in the memristor resistance or memristance, M with respect to time dM/dt [92, 3]. During periods of non-switching, the device maintains a particular resistance state, typically either Low-Resistance State (LRS) or High-Resistance State (HRS). To incorporate *RESET* failure, we propose that the model characterizes another state akin to LRS and HRS. As elucidated earlier in Section 3.3.1, when the applied voltage across the memristor surpasses the *RESET* threshold voltage V_{tRFS} , the device's resistance transitions to R_{RFS} , which is considerably lower than the device resistance at LRS. Hence, the proposed model is designed to encompass three states: the Low Resistance State (LRS), the High Resistance State (HRS), and a *RESET* Failure State (RFS). As discussed in (Section 3.3.1), once the device enters *RESET* failure state, it becomes irrecoverable. This means that the device loses its switching

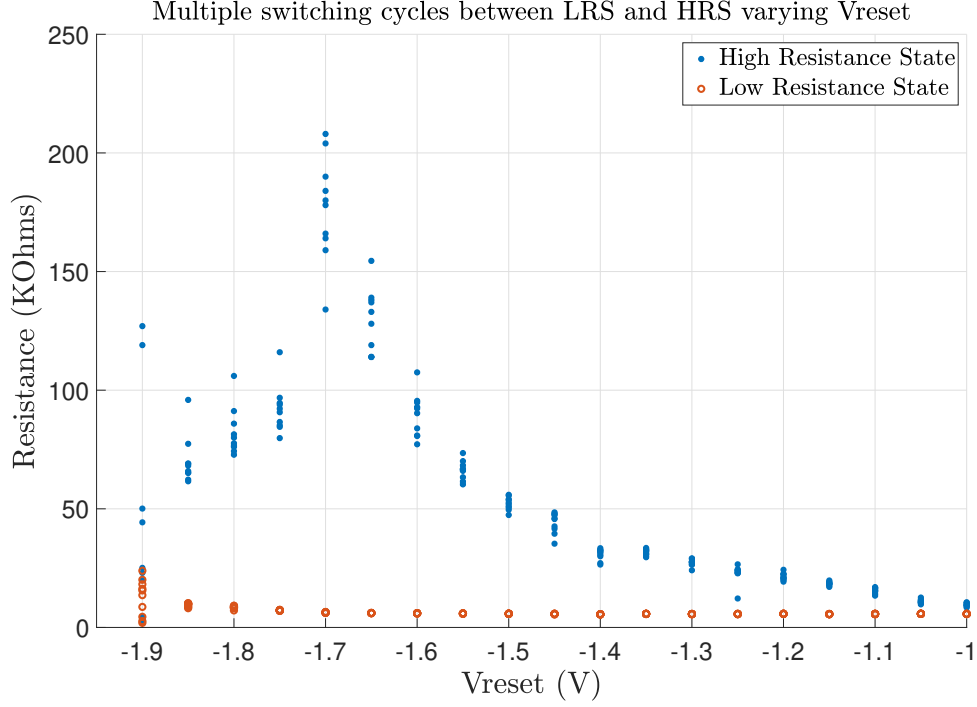


Figure 3.6: The experimental results in this plot showcase testing data from the HfO_2 memristors for various observed *RESET* voltages across multiple switching cycles as V_{RESET} is varied. The High-Resistance State (HRS) values at each *RESET* voltage are depicted in blue. When the *RESET* voltage isn't sufficiently low, a complete *RESET* operation fails to occur (Fig. 3.5(a)). This is evident in the range from -1.3V to -1V, where the memory window is insufficiently large. The optimal range appears to be around -1.5V, where a full *RESET* is achieved (Fig. 3.5(b)). The graph notably indicates the V_{RESET} level where the absolute value of *RESET* voltage is too high, resulting in *RESET* failure and a subsequent resistance drop to approximately $1.5 \text{ k}\Omega$ (Fig. 3.5(c)). Additionally, the value of Low-Resistance State (LRS) is highlighted in orange, demonstrating the shrinking memory window as device operation parameters approach the *RESET* failure threshold [64].

behavior and turns effectively into a constant resistance. Moreover, this resistance remains fixed at $R_{RFS} < LRS$. This is illustrated in the flowchart depicted in Fig. 3.7.

The updated device model is now represented by the modified mathematical equation defined as,

$$\frac{dM}{dt} = \begin{cases} C_{LRS} \frac{\Delta r}{t_{swp}} \left(\frac{V(t) - V_{thp}}{V_{thp}} \right)^{\alpha_{LRS}} Win_{LRS} M(t), & V(t) > V_{thp} \\ C_{HRS} \frac{\Delta r}{t_{swn}} \left(\frac{V(t) - V_{thn}}{V_{thn}} \right)^{\alpha_{HRS}} Win_{HRS} M(t), & V(t) < V_{thn} \\ C_{RFS} \frac{\Delta r}{t_{swRFS}} \left(\frac{V(t) - V_{tRFS}}{V_{tRFS}} \right)^{\alpha_{RFS}} Win_{RFS} M(t), & V(t) < V_{tRFS} \end{cases} \quad (3.6)$$

here, $M(t)$ represents the instantaneous device resistance. V_{thp} and V_{thn} denote the positive and negative switching thresholds for the applied voltage, respectively. V_{tRFS} signifies the voltage threshold where the device is pushed into reset failure state. t_{swp} , t_{swn} , and t_{swRFS} refer to the switching time taken by the device to transition to a specific state. The constants C_{LRS} , C_{HRS} , and C_{RFS} , along with α_{LRS} , α_{HRS} , and α_{RFS} , serve as fitting parameters that align the switching activity of the device model to the I-V characteristics observed in experimental data. These parameters can be found by utilizing methods outlined in [2].

Win_{LRS} and Win_{HRS} are window functions that aid in convergence of the model in simulations. This is done by smoothing the $\frac{dM}{dt}$ transition during the switching activity.

$$Win_{(LRS/RFS)/HRS} = \left(\frac{1}{1 + e^{\frac{1}{\Delta r \beta_{LRS/HRS}} (M(t) - \theta_{LRS/HRS} (LRS/R_{RFS})/HRS)}} \right) \quad (3.7)$$

It's important to note that V_{tRFS} falls within the range of -1.7 V to -2.2 V, and V_{thn} represents the negative switching threshold within the range of -0.7 V to -1.5 V [64]. Hence,

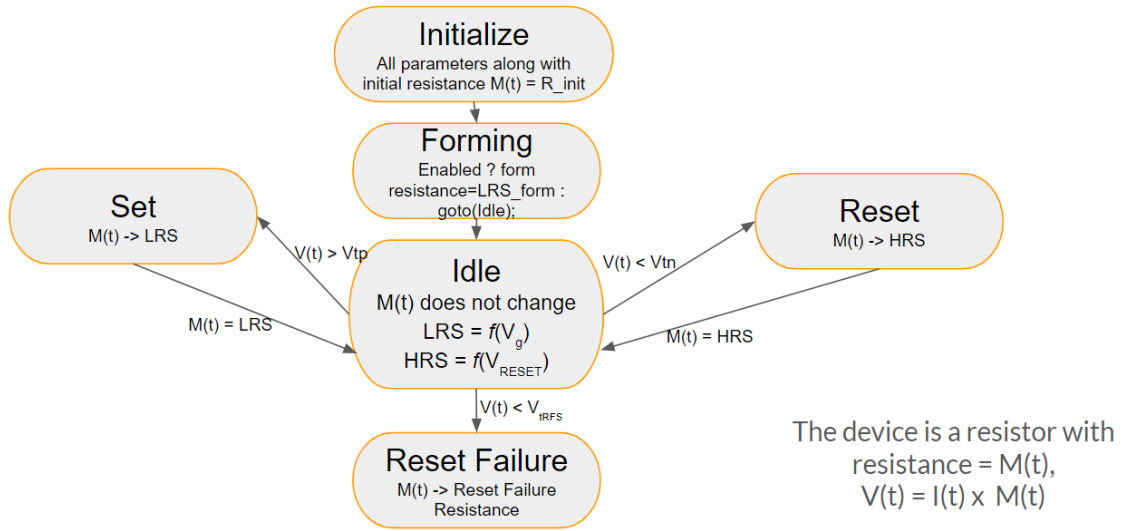


Figure 3.7: The Flowchart depicting the modified Verilog-A model for the HfO_2 memristor device as it transitions between different resistance states. In addition to the traditional Low Resistance State (LRS) and High Resistance State (HRS), a *RESET* Failure State (RFS) is introduced. Once the device enters the RFS, it remains in this state for the duration of the simulation as it is not recoverable. This model enhancement provides a more comprehensive representation of the device behavior, particularly in scenarios where failure states occur.

during circuit operation, it's highly probable that the device will transition from the HRS state to the RFS state upon crossing the *RESET* failure threshold.

3.4 Simulation Results

The Current-Voltage (I-V) characteristics acquired from the device model, featuring the distinctive hysteresis loop, are depicted in Fig. 3.4. Furthermore, the plot illustrating model accuracy for LRS values, showcasing experimental data versus simulated values from the model for various gate voltages V_g (Compliance current), is presented in Fig. 3.8. The error between experimental and simulated values is approximately 2%. This discrepancy is deemed insignificant when considering device variability. Additionally, the accuracy for the High-Resistance State (HRS) fitted data is depicted in Fig. 3.9. This plot illustrates simulated values versus experimental data for the HRS values at different V_{reset} settings. The error between experimental and simulated values in this context is approximately 3%, which remains negligible.

Fig. 3.10 and Fig. 3.11 present scatter plots depicting experimental and simulated resistance values observed at multiple switching instances. These plots vividly illustrate the stochastic nature and variable switching dynamics of the model across numerous switching cycles. The distinct spreads evident in both plots underscore the considerable variability in the observed High Resistance State (HRS) when compared to Low Resistance State (LRS) values, aligning closely with experimental findings [63]. The variability effect from the mathematical model can be attributed in part to the introduced dependency of HRS on V_{reset} , and LRS on V_g , along with the inherent stochasticity of the switching parameters.

The proposed model undergoes simulation using Cadence Spectre to investigate *RESET* failure, as illustrated in Fig. 3.12. Fig. 3.13 presents the control signals enabling various operations such as *READ*, *RESET*, and *SET*. Three test scenarios are analyzed with $RESETB_VDD$ set at 1 V, 1.5 V, and 2 V. The voltage across the memristor and the resistance level during *RESET* and *SET* are observed to examine *RESET* failure.

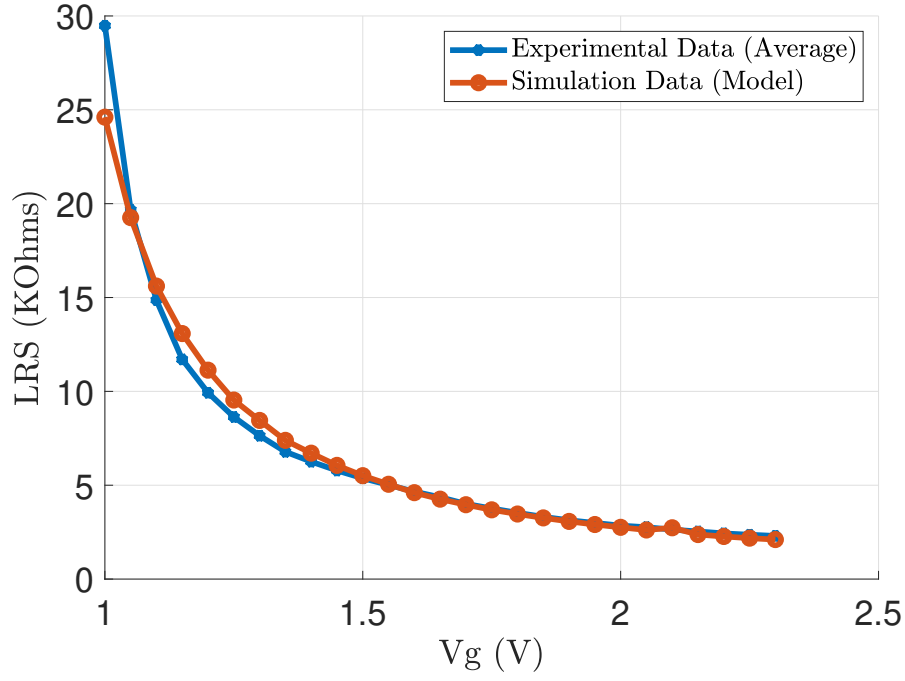


Figure 3.8: Low Resistance State (LRS) with respect to the gate voltage, V_g , of the transistor regulating the current through the memristor device, also known as the current compliance, against experimental data.

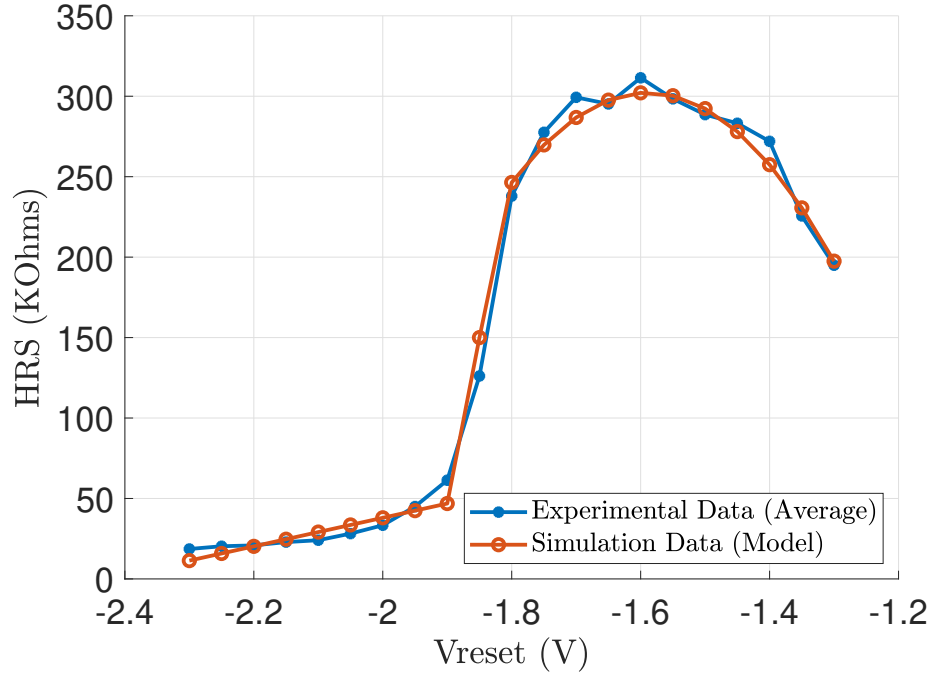


Figure 3.9: HRS vs. reset voltage, V_{reset} , against experimental data showing the impact of V_{reset} on HRS values.

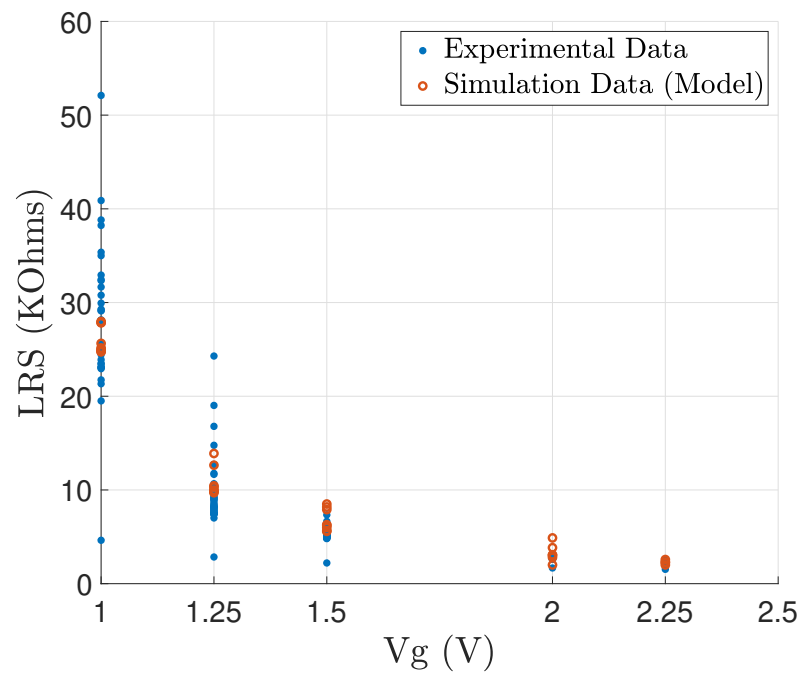


Figure 3.10: Experimental vs. Simulation data for multiple switching cycles showing LRS samples at different V_g values (LRS Variability)

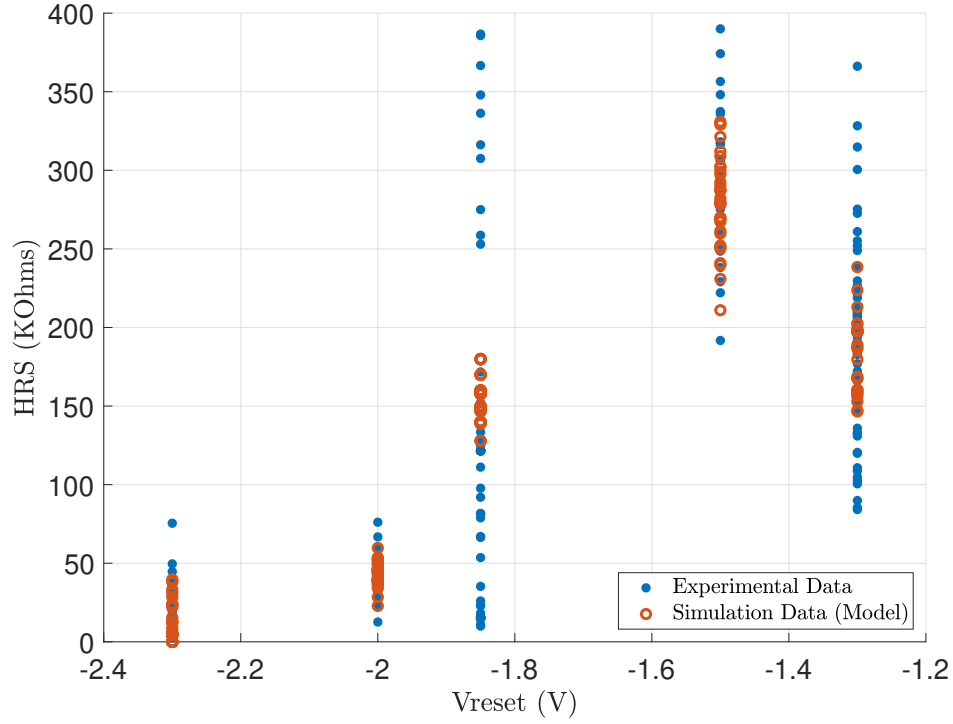


Figure 3.11: Experimental vs. Simulation data for multiple switching cycles showing HRS samples at different V_{reset} value (HRS Variability)

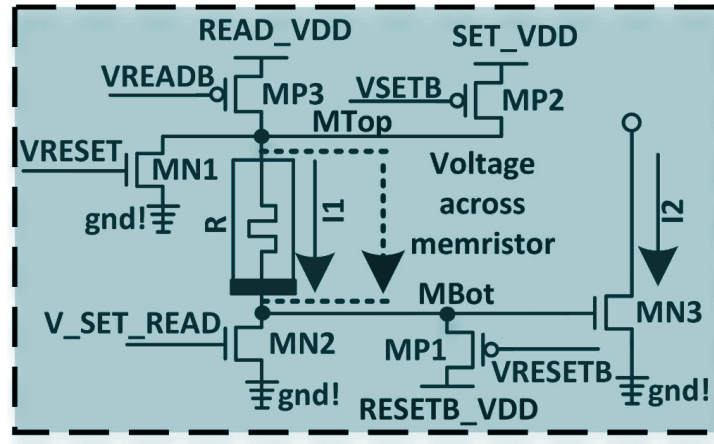


Figure 3.12: The memristive synapse utilized for simulating *RESET* failure is depicted [117]. The memristor device labeled "R" is shown with different MOSFET devices to control various operations. During the *RESET* operation, a dedicated V_{DD} for $MP1$ is required. It is crucial to maintain the voltage across the memristor below a certain threshold to prevent *RESET* failure.

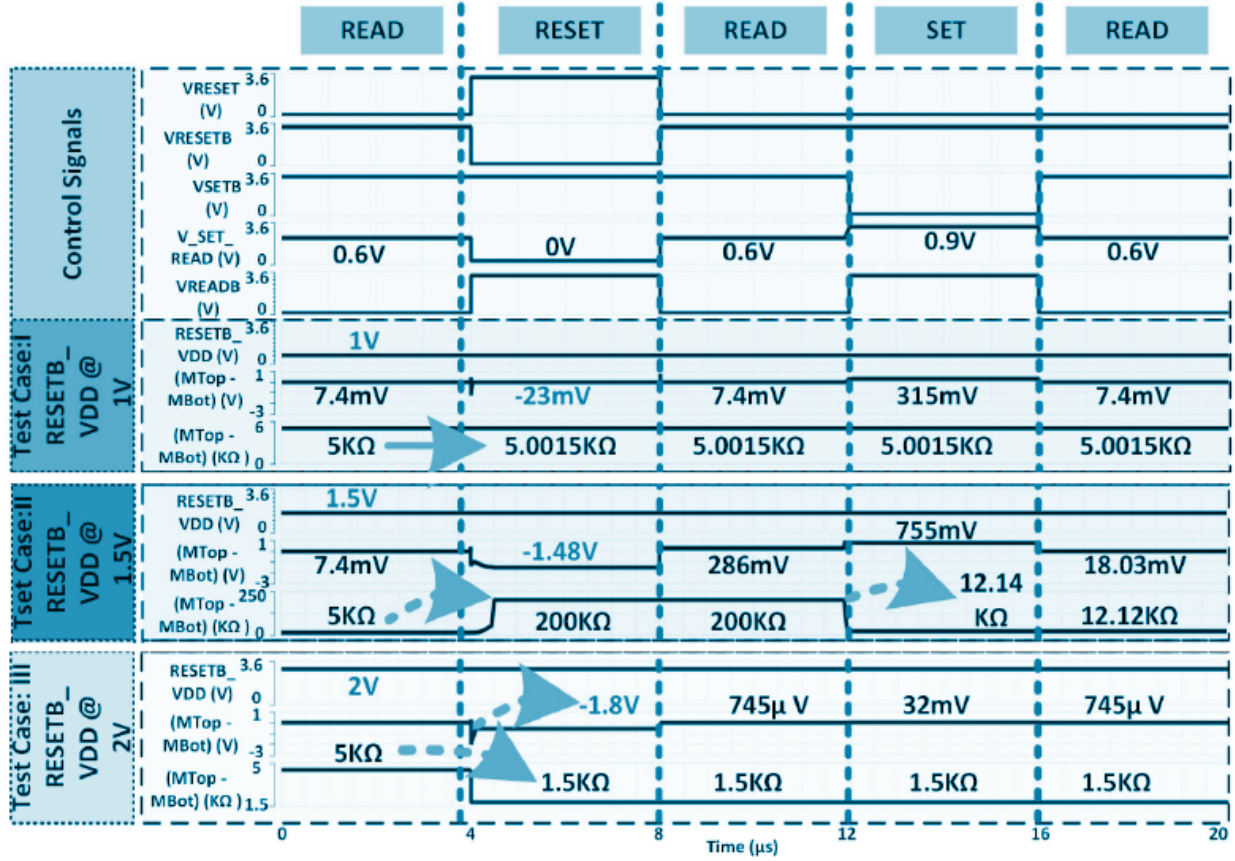


Figure 3.13: The Cadence simulation results demonstrate the failure modeling of the proposed Verilog-A model. Control signals are employed to initiate the *READ*, *RESET*, and *SET* operations for various values of *RESETB_VDD*. Three distinct test cases are presented: when *RESETB_VDD* is below the *RESET* threshold (V_{thn}), within the *RESET* window (V_{thn} to V_{tRFS}), and above the *RESET* failure voltage (V_{tRFS}). The bottom waveform plot illustrates the device entering the *RESET* failure state. Note: A positive *RESETB_VDD* value corresponds to a negative *RESET* voltage across the memristor.

3.5 Benchmarking Simulation for Convergence

Addressing the importance of the achieving convergence in circuit simulations, particularly when dealing with device models documented in the literature, is widely recognized [115]. To confront this challenge, we conduct a convergence analysis for the proposed model, utilizing a benchmarking circuit as a reference point to evaluate the stability of numerical solutions. To showcase the effective convergence of the model in circuit simulation, a benchmark circuit comprising 840 memristors [12] was simulated in Cadence Virtuoso. This benchmarking circuit, shown in Fig. 3.14, serves as a valuable tool for assessing the convergence of the model in simulation, given its large size and exclusive utilization of memristor devices interconnected in a mesh. The duration taken by different models for the same simulation run serves as a metric for the model’s convergence. The circuit receives a sinusoidal input signal waveform oscillating at a frequency of 10kHz.

Using an identical simulation environment in Cadence Spectre, the simulation time is compared for various Verilog-A models documented in the literature and results are presented in Table 3.1. The Amer model, known for its relatively shorter simulation time, is also an empirical model that treats LRS and HRS values as constants [3]. It is anticipated that the proposed model, which integrates experimentally observed HRS and LRS behavior along with stochasticity, will exhibit slower simulation times in comparison.

3.6 Conclusion

A concise empirical circuit simulation model has been developed to accurately represent the behavior of hafnium-dioxide (HfO_2) memristor devices. This model, designed for system-level simulations, has undergone rigorous convergence testing using Cadence Spectre. Its calibration is grounded in experimental data, ensuring its fidelity in simulating real-world scenarios. One notable aspect of this model is its incorporation of a sigmoid function as a window function, tailored to smooth resistance/memristance transitions during device switching, thus enhancing simulation convergence.

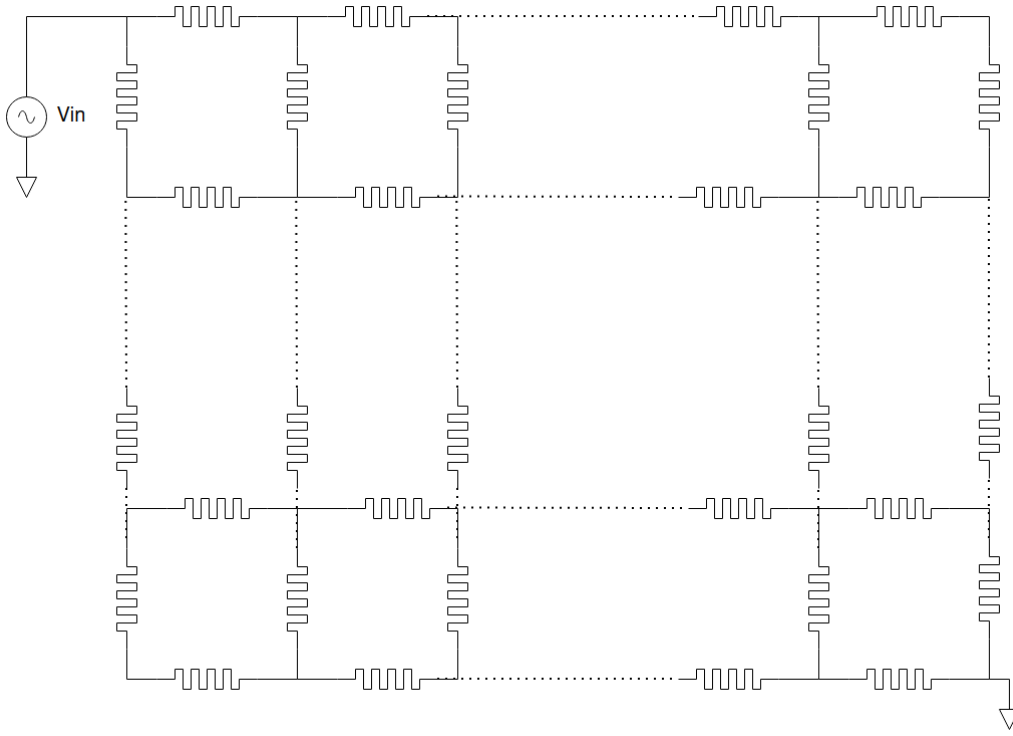


Figure 3.14: The illustrated benchmark circuit acts as a convergence test for the proposed Verilog-A model of the memristor. Comprised solely of memristors, the circuit’s ability to converge numerical equations directly relies on the smoothness and continuity of the model equations. This test provides valuable insight into the accuracy and reliability of the proposed Verilog-A model in simulating memristor behavior under various conditions [92].

Table 3.1: Simulation times (indicating convergence) for test circuit (Fig. 3.14) with different device models [93]

Model	Simulation Time	Incorporates Variability	Temperature Effects
PWL Model	4 min	No	No
Pino Model [86]	4 min 24 sec	No	No
Modified Pino Model [71]	1 min 24 sec	No	No
Amer Model [3]	10 sec	No	No
Unimore Model [88]	32 min 15 sec	Yes	Yes
Peking Model [37]	<i>No convergence achieved</i>	No	Yes
Proposed Model	35 sec	Yes	Yes

An inherent capability of this model is its ability to accurately capture the tunability of both the Low-Resistance State (LRS) and High-Resistance State (HRS) values, as demonstrated in previous research by Liehr et al. (2020) [63]. Moreover, it effectively accounts for the intrinsic variability and stochasticity present in LRS and HRS characteristics.

Another noteworthy feature of the proposed model is its ability to simulate *RESET* failure behavior. This is achieved by treating the failure state as an additional state that the memristor can transition to, alongside LRS and HRS states. Importantly, the model's adaptability to different bipolar memristors is facilitated by its reliance on experimental data specific to the device under consideration. This versatility empowers designers to deploy the model across a wide array of memristor types, eliminating the necessity for laborious development of physics-based models for every device variant with iterative refinement.

Chapter 4

Circuit-Switched Network-on-Chip Neuromorphic Architecture

The design of reconfigurable Spiking Recurrent Neural Networks (SRNNs) requires careful consideration of on-chip connectivity architecture, which is governed by the constraints of physical wiring.

The capability of full connectivity is desirable for a reconfigurable Network-on-Chip (NoC) architecture. Full connectivity refers to the scenario where every neuron on the chip can receive input from all other neurons simultaneously. However, not all potential synaptic connections within a neural network need to be active simultaneously. This can be inferred from various inherent factors associated with the functionality of SRNNs, including the characteristics of individual neurons and the structure of synaptic connections.

Primarily, the restricted fan-out and fan-in of neurons impose inherent limitations on the number of connections within SRNNs. Moreover, research shows that SRNNs often exhibit sparse recurrent connectivity patterns [96, 49]. This means that only a fraction of potential connections between neurons are utilized at any given time. Such sparsity not only conserves computational resources but also facilitates efficient information processing

within the network. Furthermore, investigations into Reservoir Computing (RC) systems, as discussed in [97], emphasize the advantages of sparse connectivity in SRNNs. Sparse networks demonstrate enhanced generalization capabilities, enabling them to effectively process unseen input patterns. This adaptability and extrapolation ability are crucial for real-world applications where the network needs to operate in varied and dynamic environments.

Further, addressing the substantial resource, area, and power requirements for implementing a fully-connected network in Network-on-Chip (NoC) architecture for Spiking Recurrent Neural Networks (SRNNs) can be challenging, especially when only a small percentage of those connections are active simultaneously. Hence, the objective is to develop resource-efficient NoC for sparse SRNNs on hardware, that offers the capacity for a fully connected network. Below are the key characteristics to be considered for designing the Network-on-Chip (NoC) architecture for a neuromorphic system:

- **Routing and Interconnection:** The NoC must establish connections between any two arbitrary sets of neurons within the system to robustly support recurrent connections.
- **Resource Efficiency:** It is imperative for the NoC to operate with utmost energy and area efficiency, thus, optimizing resource utilization.
- **Reliability:** Ensuring high fidelity spike transmission with minimal degradation and information loss is paramount for the NoC’s functionality.
- **Synchronization:** The NoC must prioritize preserving spike timing information during data transmission to maintain temporal accuracy.
- **Scalability:** The NoC should seamlessly accommodate the expansion and addition of neurons, facilitating system growth.
- **Flexibility:** The design space of the NoC should encompass diverse routing solutions, providing adaptability to varying system requirements.

While most NoC architectures in literature focus on large-scale implementations of spiking hardware, in many realistic applications, spiking neural network sizes are quite small with just a few hundred neurons [1]. The challenges in designing reconfigurable small-scale NoCs are

not explored to a large extent. These challenges primarily revolve around proximal neurons, referring to neurons placed closer together on-chip. Further, designing even a relatively smaller SRNN with hundreds of neurons on hardware presents a significant challenge, particularly due to the presence of recurrent connections. These connections, as shown in Fig. 4.1, facilitate any-to-any neuron connectivity, allowing loops to form within the neural network architecture. Such recurrent connectivity enables the system to effectively process spatio-temporal data, granting the SRNN a fading memory property that allows the network to retain information from past inputs. The neurons depicted in Fig. 4.1 are abstracted as neuron cores shown in Fig. 4.2 where each neuron has multiple synapses at its inputs. The fan-in of a neuron is determined by the number of synapses at its input.

To tackle the challenge of small-scale reconfigurable Network-on-Chip (NoC) designs for SRNNs, we proposed a Circuit Switching-based approach. The proposed NoC approach offers several advantages, including being area-efficient, scalable, flexible, and delay-agnostic. This circuit-switched approach can be utilized independently for relatively small network sizes, particularly when the maximum distance between two neurons on the chip is not significant. We introduce the NoC architecture as SpiCS-Net, pronounced "spikes-net," standing for Spike-routing Clos Circuit-Switched Network-on-Chip. SpiCS-Net represents a fully circuit-switched Multistage-Interconnect-Network, specifically designed to cater to the needs of resource-efficient Spiking Recurrent Neural Networks (SRNNs). Its architecture prioritizes resource efficiency, reliability, synchronization, and scalability, making it an ideal solution for implementing SRNNs. SpiCS-Net seamlessly supports unicast, multicast, and broadcast features required by SRNNs, achieving this with significantly lower resource requirements. Furthermore, SpiCS-Net facilitates communication of both digital and analog spikes, thus enabling its use in both digital and analog SRNN implementations.

4.1 SpiCS-Net NoC Architecture

SpiCS-Net utilizes the Multistage-Interconnect-Network Clos switch-based topology, which adopts a multi-stage approach to minimize the number of switches compared to a crossbar-based connectivity matrix. A crossbar NoC is illustrated in Fig. 4.3, where each input is

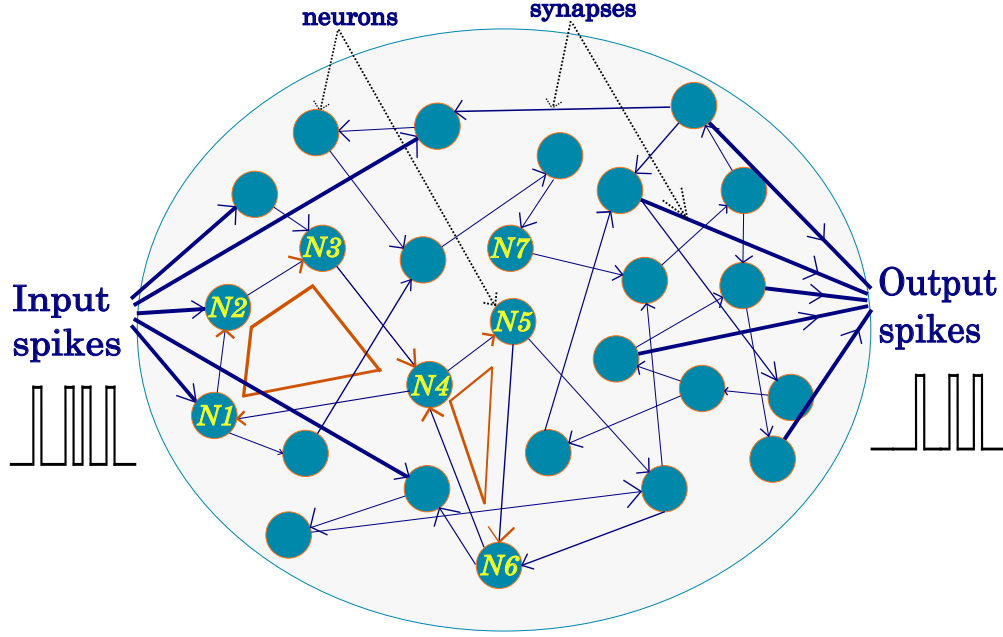


Figure 4.1: In a generic Spiking Recurrent Neural Network (SRNN), recurrent connectivity necessitates a larger connectivity space. This requirement means that the Network-on-Chip (NoC) must be capable of supporting a fully-connected SRNN. Unlike traditional neural networks, where connectivity is constrained by feed-forward synaptic communication between adjacent layers only, the SRNN's recurrent connections demand a more extensive network topology. As a result, the hardware resources required for implementing such connectivity increase manifold.

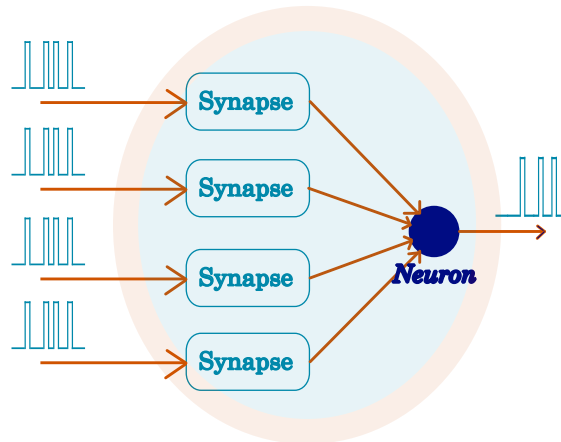


Figure 4.2: A neuron core, the fundamental unit of SRNN shown in Fig. 4.1, consists of multiple synaptic inputs. These inputs serve as the conduits through which information flows into the neuron, enabling it to process and integrate incoming signals. Each synaptic input represents a connection from other neurons.

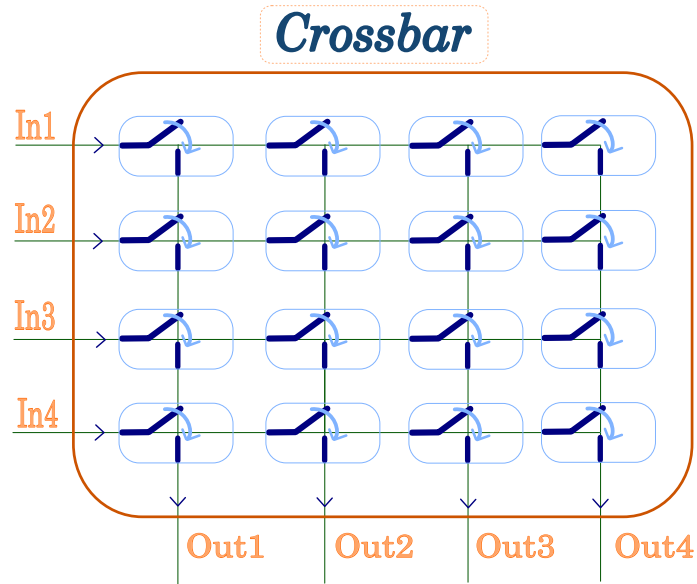


Figure 4.3: Crossbar composed of enable/disable switches with one switch per cross-point between an input-output pair. Simple NoC but implementing the entire system using Crossbar Network-on-chip is inefficient since the number of switches scales multiplicatively.

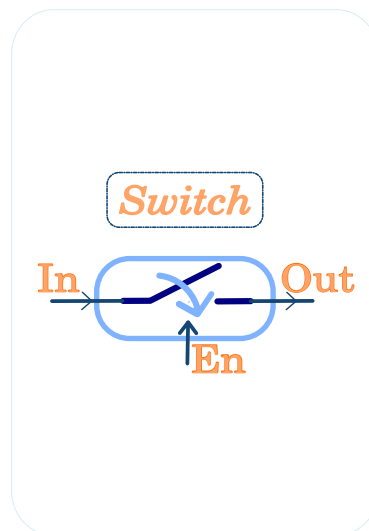


Figure 4.4: Typical enable/disable controlled switch in a circuit switching crossbar; this switch can be implemented using Transmission gates, MOSFET switches, or Multiplexers

connected to all outputs using a single enable/disable switch shown in Chapter 4. This switch controls the establishment or disconnection of a wired connection between them. The proposed network architecture adopts a Clos topology, leveraging small crossbar modules to construct a Network-on-Chip (NoC) with multiple stages as shown in Fig. 4.5. Consequently, an input signal must traverse multiple stages of crossbars to establish a connection with a specific output. This hierarchical structure of crossbars results in a significant reduction in the total number of switches, by more than threefold in some instances.

SpiCS-Net offers reconfigurability and full connectivity akin to a crossbar while mitigating the total number of switches through the incorporation of multiple switching stages [26]. This architecture can be designed to possess a non-blocking property, ensuring the network's ability to establish connections from any input terminal to any output terminal without disrupting existing connections within the network at run time. By segmenting communication switches into stages, as depicted in Fig. 4.5, the overall number of switches required for communication is significantly reduced [26]. In such a network, it is evident that both latency and throughput remain uniform across all connections. In the case of the three-switch SpiCS-Net depicted in Fig. 4.5, each spike traverses three switches. It is important to note that this concept is independent of the number of stages utilized. Indeed, considering the feasibility of reducing the number of switches through the use of additional stages, it is worthwhile to explore configurations with five or seven stages.

The SpiCS-Net architecture offers a Network-on-Chip (NoC) solution that facilitates communication of "*spikes-as-spikes*," thereby achieving highly efficient spike communication without the need for additional functionality required by packet-based approaches. Furthermore, the proposed network operates entirely asynchronously, enhancing the biological plausibility of the Spiking Recurrent Neural Network (SRNN) design while preserving the analog state of the spike in the time domain [57, 7]. The streamlined spike data path, comprising just three switches in the crossbars, accelerates communication, resulting in significantly faster computations.

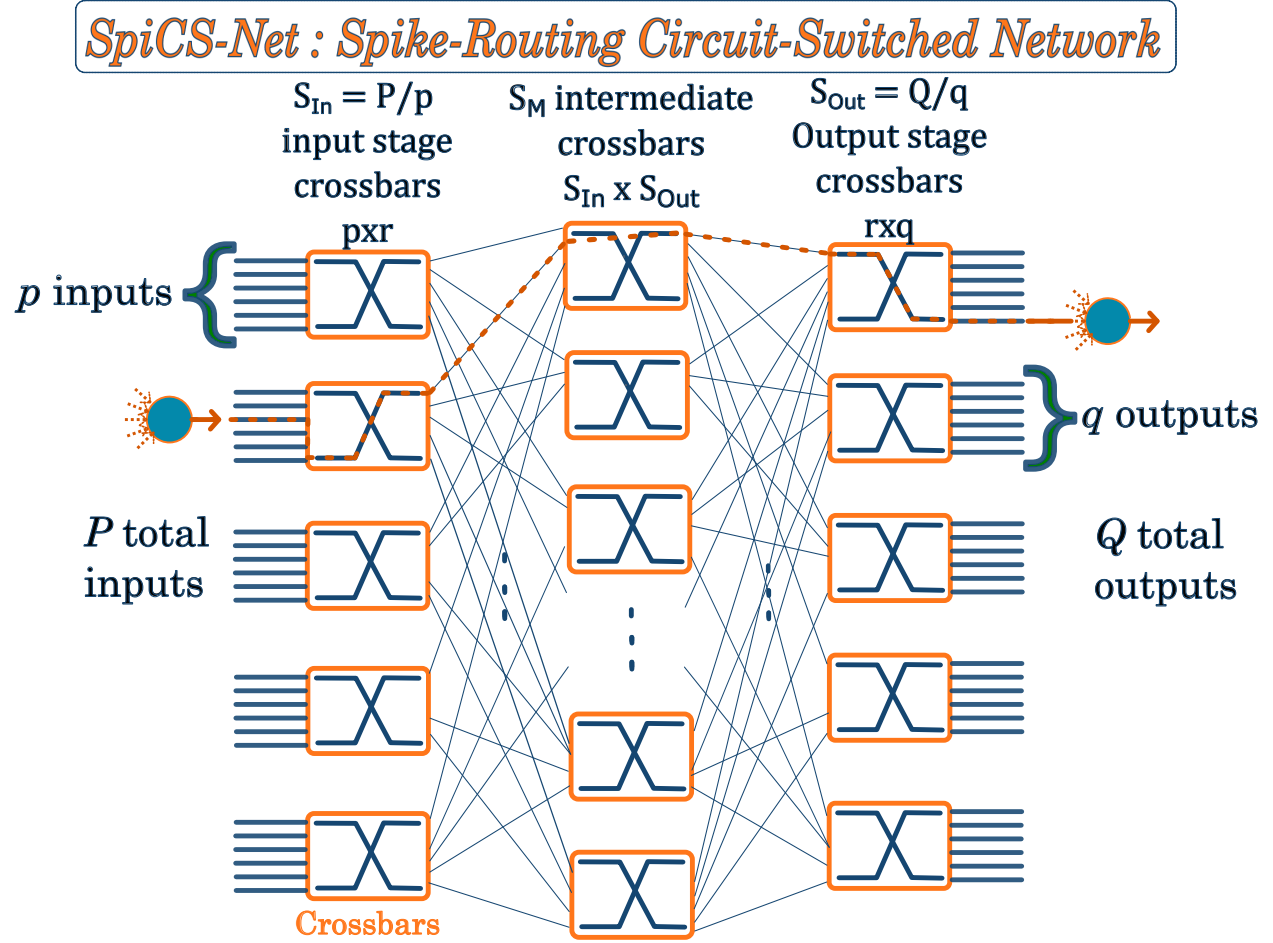


Figure 4.5: SpiCS-Net demonstrating communication of a spike between a set of neurons within the SRNN.

4.1.1 Mathematical Analysis

The expression for total number of switches [26] in the SpiCS-Net shown in Fig. 4.5 is given by,

$$X_{clos} = (p + q - 1)(P + Q + \frac{PQ}{pq}) \quad (4.1)$$

To achieve optimal network parameters and minimize the number of switches, it is imperative to minimize the X_{clos} value. This reduction ensures that the switch numbers required for implementing the network in Fig. 4.5 are as small as possible, consequently reducing the area of SpiCS-Net on-chip. P and Q can be calculated using the SRNN size and other network parameters (see Table 4.1). In the scenario when $P = Q$ and $p = q$, the Eq. (4.1) simplifies to,

$$X_{clos} = (2p - 1)(2P + \frac{P^2}{p^2}) \quad (4.2)$$

For minimum value of p ,

$$\frac{dX_{clos}(3)}{dp} = 2p^3 - pP + P = 0 \quad (4.3)$$

It's essential to note that if p and q are incorrectly sized, there's a risk of increasing the area overhead instead of reducing it for a specific network size.

In a unicast SpiCS-Net, where one input drives only one output, the maximum number of connections in the SpiCS-Net is given by $\min(P, Q)$. However, for multicast network implementations, the maximum number can be higher if $P < Q$. Fig. 4.6 illustrates the SpiCS-Net for multicast spike communication. To ensure the multicast property, the connections between stage 1 and stage 2, $Conn_{12} = (P/p) \times r$, should exceed P ; and connections between stage 2 and stage 3, $Conn_{23} = r \times (Q/q)$, should surpass $Conn_{12}$, and so forth. This implies that the value of Q/q should be greater than P/p . Combining this requirement with the non-blocking condition in Eq. (4.4), we can determine the necessary conditions on design parameters for implementing a multicast network. Therefore, for a given P and Q of the

Table 4.1: Design variables, descriptions, and their correspondence to SpiCS-Net.

Variables	Description
N	Number of neurons in SRNN
syn_out_{neuron}	Fan-out per neuron
syn_in_{neuron}	Fan-in per neuron
$P = syn_out_{neuron} * N$	Number of inputs (SpiCS-Net)
$Q = syn_in_{neuron} * N$	Number of outputs (SpiCS-Net)
p	P inputs divided in groups of p ; inputs to smaller crossbars (Fig. 4.5)
q	Q outputs divided in groups of q ; outputs from smaller crossbars (Fig. 4.5)
$S_{In} = \frac{P}{p}$	Number of crossbars in input stage of SpiCS-Net
$S_{Out} = \frac{Q}{q}$	Number of crossbars in output stage of SpiCS-Net
S_M	Number of crossbars in intermediate stage of SpiCS-Net.

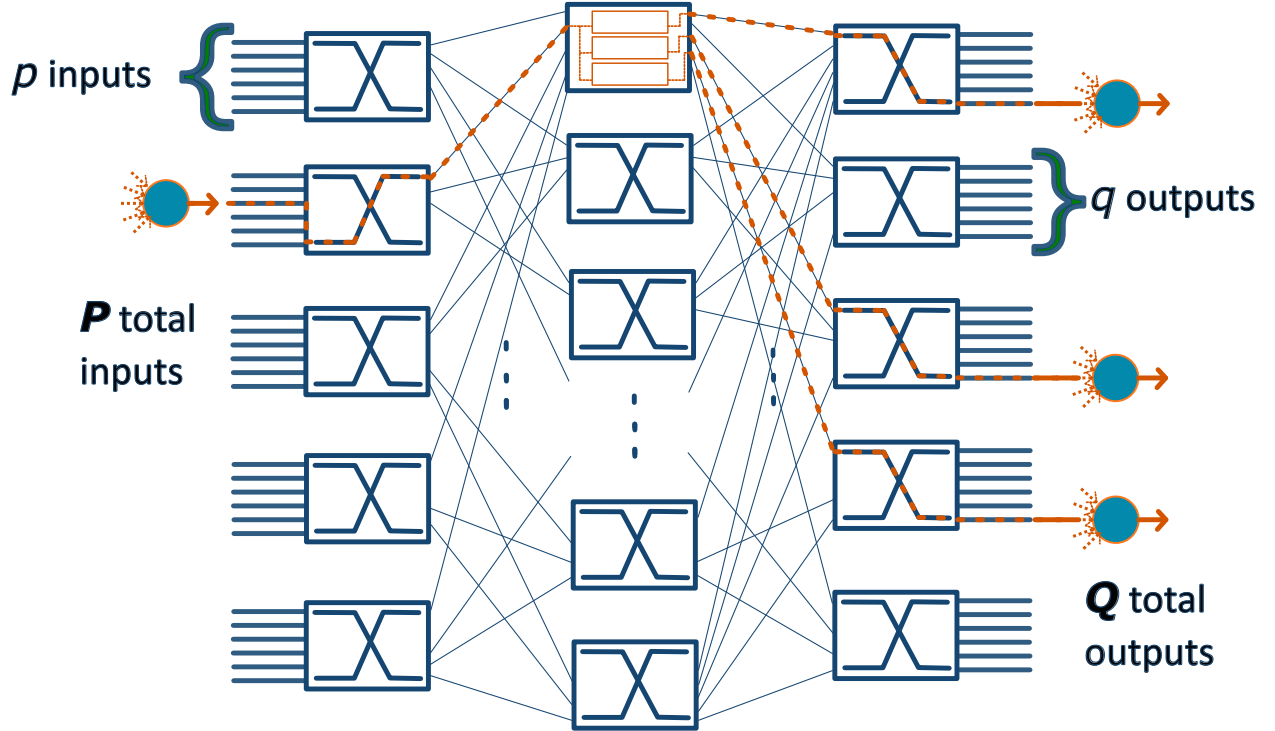


Figure 4.6: SpiCS-Net demonstrating one possible way for multicast spike communication within the SRNN. Here, output of one neuron is routed to input synaptic connections of three neurons.

NoC, the conditions on p , q , and r are governed by Eq. (4.1), minimization of Eq. (4.3) and Eq. (4.4).

$$r \leq p + q - 1, \quad P/p < Q/q, \quad r > p, \quad r > q \quad (4.4)$$

When designing the SpiCS-Net architecture for the SRNN comprising N_n neurons, it's crucial to account for the number of connections required by the SRNN. Between any two neurons, a maximum of 2 unidirectional connections can exist. Therefore, the total possible connections for N_n neurons is given by,

$$C_{max} = 2 * \binom{N_n}{2} = N_n(N_n - 1) \quad (4.5)$$

In Eq. (4.5), we define the maximum possible connections for the ideal SRNN, assuming neuron circuits with infinite fan-in and fan-out.

We define fan-in, denoted as syn_{in} , of a neuron as the maximum number of synapses that can drive input spikes into the neuron, and fan-out, denoted as syn_{out} , as the total number of neurons a spike can be routed to from the output of a single neuron. Given that all connections are formed within the SRNN, the maximum number of possible connections is given by,

$$C_{max} = N_n * \max(syn_{in}, syn_{out}) \quad (4.6)$$

For a multicast SpiCS-Net implementation, Eq. (4.6) establishes the values of P and Q for a specific-sized SRNN and the fan-out and fan-in value, respectively. Subsequently, Eq. (4.1) can be utilized to determine the optimal values of p and q . Moreover, the value of r can be determined such that the conditions outlined in Eq. (4.4) are satisfied. Other parameters of the network, such as switch sizes and the number of switches, can be expressed in terms of P , Q , p , q , and r , as depicted in Fig. 4.5 and Table 4.1.

4.2 Tuning Concurrent Connectivity of SpiCS-Net for Sparse SRNNs

As discussed earlier, while the design of reconfigurable Spiking Recurrent Neural Networks (SRNNs) demands the capability of full on-chip connectivity, but not all of these connections must be active simultaneously. Several factors contribute to this, including the limited fan-out and fan-in of neurons and sparse recurrent connectivity [96, 49].

Addressing the substantial resource, area, and power requirements for implementing a fully-connected network in Network-on-Chip (NoC) architectures for Spiking Recurrent Neural Networks (SRNNs) is also necessary, especially when only a small percentage of those connections are active simultaneously. We propose a novel solution through utilization of the proposed circuit-switched NoC architecture, SpiCS-Net, which leverages the concurrent connectivity requirement on-chip. This approach aims to create resource-efficient, sparse SRNNs on hardware while still offering the capacity for a fully connected network.

In this context, Concurrent Connectivity is a measure representing the maximum ratio of connections that can be established concurrently through the Network-on-Chip (NoC) to the total number of connections required for full connectivity. The SpiCS-Net architecture can be purposefully modified to enable the adjustment of Concurrent Connectivity. This adaptability offers a practical solution for implementing efficient and resource-conscious sparse Spiking Recurrent Neural Networks (SRNNs) in hardware. It optimizes resource usage while accommodating the specific network requirements. Consequently, we fine-tune the Concurrent Connectivity of SpiCS-Net to harness the sparseness inherent in Spiking Recurrent Neural Networks (SRNNs).

The central concept here revolves around harnessing the blocking property inherent in SpiCS-Net and establishing its relationship with the sparsity of Spiking Neural Networks (SNNs) to achieve reductions in both area and power consumption. The implementation of a blocking Clos topology, as depicted in Fig. 4.7, imposes restrictions on the number of connections permissible through the network. This results in a scenario where a few input and output nodes will remain unconnected. The degree of blocking can be managed by adjusting the number of crossbars in the intermediate stage, as illustrated in the figure. The objective

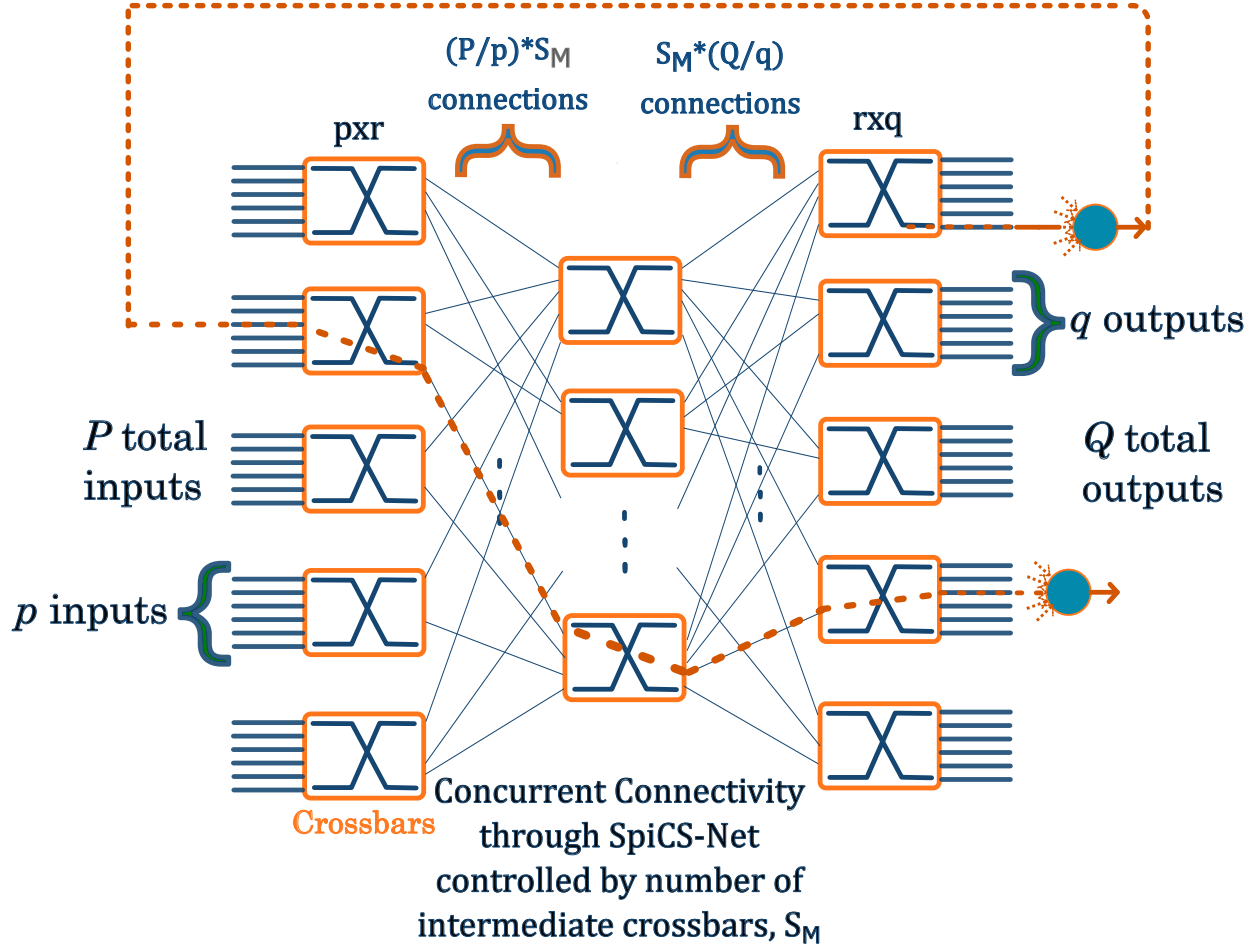


Figure 4.7: SpiCS-Net architecture shows a lowering of crossbar switches in the network, leading to area saving without restricting connectivity between input and output nodes. If $S_M = p$, the SpiCS-Net would always provide at least one routing path from every input node to any output node. This is the minimum connectivity that needs to be supported by the NoC. By increasing $S_M = p + q - 1$, the network will always allow a path from any input node to any output node without rearranging the connections. $S_M \geq q$ ensures 100% concurrent connectivity if $q > p$. A 100% Concurrent Connectivity implies that all neurons can drive synaptic inputs of all other neurons simultaneously. This is not realistic for a hardware SRNN due to a few reasons: 1. Neuron circuits will have limited fan-out and can not drive unlimited synapses simultaneously; 2. The purpose-built trained SRNNs will never need a fully-connected level of synaptic density. In most applications, SRNNs are sparse than they are dense. Thus, the density of connections that must be established simultaneously (Concurrent Connectivity) can be restricted.

is to minimize the count of cross-points or switches in the network by decreasing the number of concurrent connections. Each cross-point in the network corresponds to a switch, and by reducing their numbers, we directly reduce the hardware resources and memory required for implementation. This further leads to lower energy consumption as a result of reduced circuitry.

To leverage the sparseness of SRNNs, we are interested in the upper limit on concurrent connections that can be established in the NoC. Conversely, the ratio of connections in the SpiCS-Net that are blocked to total connections in a fully connected SRNN. The total number of neurons N with fan-out per neuron (synaptic output) syn_out_{neuron} in the network defines the number of input nodes of SpiCS-Net, $P = N * syn_out_{neuron}$. And N combined with the fan-in per neuron (synaptic input) syn_in_{neuron} defines number of output nodes, $Q = syn_in_{neuron} * N$. The maximum possible unicast connections in a fully-connected SRNN are given by,

$$C_{max} = \max(P, Q) \quad (4.7)$$

Further, Fig. 4.7 shows the total number of connections between the input-middle and middle-output stages. The Clos topology transforms the problem of connecting P inputs to Q outputs into a problem of connecting p inputs and q outputs. Thus, for establishing full-connectivity concurrently, the number of crossbars in the intermediate stage is limited by values of p and q ,

$$S_M \geq \max(p, q) \quad (4.8)$$

This condition describes the limitation on the number of intermediate stage crossbars for a Rearrangeably Non-blocking Clos network, in contrast to the condition for a Strictly Non-blocking Clos network $S_M \geq p + q - 1$ [26]. We use the condition for Rearrangeably Non-Blocking Clos Network with the assumption that the SpiCS-Net configuration will be programmed on the hardware all at once instead of establishing connections one at a time. This also differentiates SpiCS-Net from the traditional telecommunication use case of Clos networks. So, in Eq. (4.8), as we violate this condition and reduce the value of S_M , the

Clos network in Fig. 4.7 starts blocking full-connectivity between p and q . However, the topology still allows connections from any node at input P to any node at the output Q . This is demonstrated in Fig. 4.7. The compromise is that all possible connections in a fully-connected network can not be established concurrently. As stated earlier, the SRNNs of interest are sparse networks. Thus, we can restrict the concurrent connectivity by reducing the number of crossbars in the intermediate stage, S_M for an area-efficient circuit-switched NoC implementation. As we start reducing the number of switches in the intermediate stage from S_M to $\bar{S}_M$, the total number of concurrent connections in terms of S_M is given by,

$$CC_{\bar{S}_M} = C_{max} - \max\left(\frac{P}{p}(S_M - \bar{S}_M), \frac{Q}{q}(S_M - \bar{S}_M)\right) \quad (4.9)$$

concurrent connectivity in terms of percent is given by,

$$CC_{percent} = \left(\frac{CC_{\bar{S}_M}}{C_{max}}\right) * 100 \quad (4.10)$$

This $CC_{percent}$ value is correlated with the degree of sparseness and synaptic density in the SRNN. Thus, provided a maximum synaptic density for the reconfigurable SRNN, the total number of crossbars in the intermediate stage of SpiCS-Net can be determined.

4.3 Implementation

4.3.1 Digital Spike Communication

The SRNN connected using SpiCS-Net architecture is designed in Verilog HDL and synthesized to the 65nm IBM CMOS10LPE process technology. The SPiCS-Net architecture utilizes multiplexer based crossbars in the digital implementation. The neurons are Leaky Integrate and Fire (LIF) Neurons with fan-in = 16 synaptic inputs. The layout for the LIF neuron is shown in Fig. 4.8 and layout for the multiplexer based SpiCS-Net with 75% concurrent connectivity is shown in Fig. 4.9. A SpiCS-Net controller is further implemented to drive the control bits to switches in SpiCS-Net. These SpiCS-Net control bits define the connectivity between neurons to route spikes through the NoC. Since these control bits do not affect the datapath of spikes, the timing for these interconnects is not a part of the critical signal

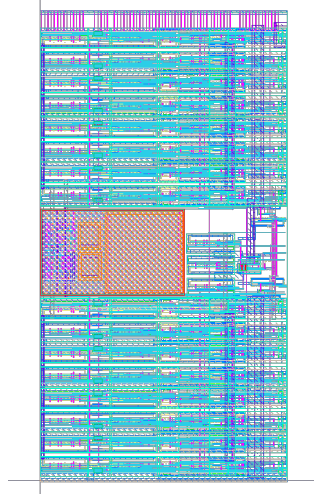


Figure 4.8: Custom designed LIF (Leaky Integrate-and-Fire) Mixed-Signal memristor-based neuron core layout designed in IBM CMOS10LPE 65nm CMOS-memristor technology with 16 synaptic inputs (size of neuron = 100x190 sq. μm)

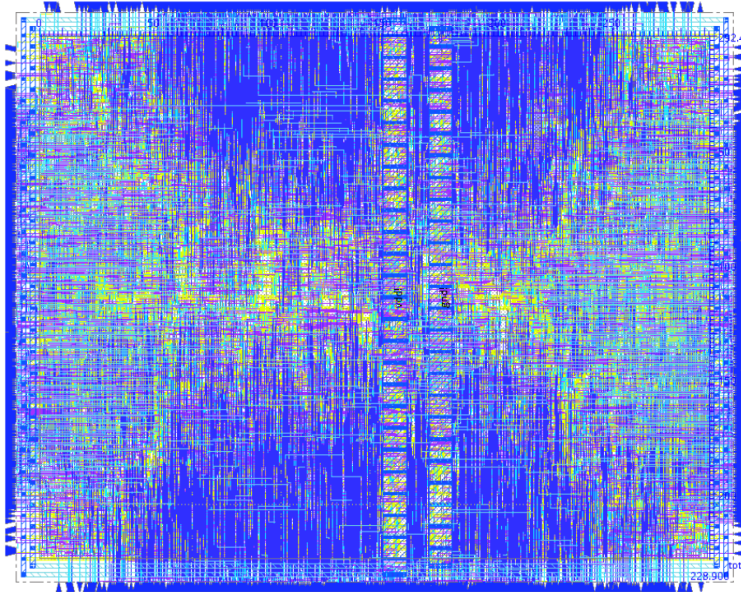


Figure 4.9: Layout of 128x128 SpiCS-Net with 75% concurrent connectivity in IBM CMOS10LPE 65nm process. Area of SpiCS-Net = 0.032 sq. mm

paths. The entire design is placed and routed using Innovus as shown in Fig. 4.10 with a total area of 4.5 sq.mm. This SRNN design includes 128 neurons, switches for SpiCS-Net, and a SpiCS-Net controller that loads the network configuration for controlling individual enable bits for the switches.

Cadence Spectre SPICE simulations from the parasitic extracted netlist of SpiCS-Net are used to get the latency, energy, and throughput. This is done for different-sized SRNNs and results are presented in Table 4.2. The fan-in and fan-out of neurons were considered as 1 for uniformity to obtain comparable results for different SRNN sizes. The dynamic power, propagation delay, and throughput are calculated from Cadence Spectre simulations with spike pulse width = $1\mu\text{sec}$, rise/fall time = $10p\text{sec}$. The dynamic energy consumption per spike from post-layout simulations for different neural network sizes is also presented Table 4.2.

We also compare SpiCS-Net with limited with a circuit-switched Crossbar. The plot in Fig. 4.11 shows a comparison between SpiCS-Net and Crossbar for the area (number of switches) required on-chip for different SRNN sizes (number of neurons). It is observed that when compared to a Crossbar, SpiCS-Net area savings increase as the size of SRNN increases. The SpiCS-Net area savings when compared to crossbar are also contributed by the memory requirement on-chip to store the network configuration; the memory is reduced due to reduction in number of switches. The trade-off is a lower throughput because of only one switch in the signal path is Crossbar as opposed to three switches for SpiCS-Net implementation.

To show how varying the Concurrent Connectivity affects the NoC area and memory on-chip, the number of switches vs the percentage of concurrent connectivity in SpiCS-Net is shown in Fig. 4.12. Further, the SpiCS-Net area in IBM65nm process as Concurrent Connectivity is varied in a 128x128 SpiCS-Net is also presented in Fig. 4.13. This data is the layout data, extracted from the synthesized, placed, and routed netlist and exhibits the same trend as illustrated in Fig. 4.12. The data in tabular format is also shown in Table 4.3 to show a comparison of how area savings scale as concurrent connectivity is varied. The data presented in Fig. 4.13 and Table 4.3 corresponds to SpiCS-Net area and does not include neuron area on-chip.

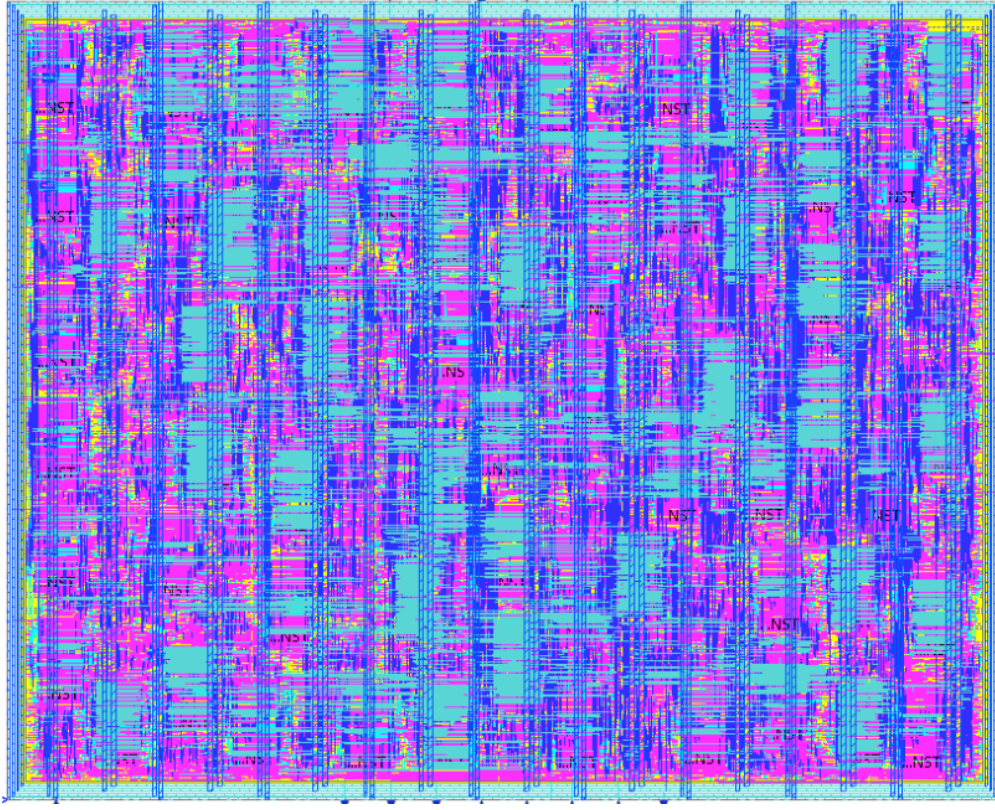


Figure 4.10: Layout of Spiking Recurrent Neural Network Cluster with SpiCS-Net, 128 Neuron cores, SPI, and scan_config block in IBM10LPE 65nm process. Area = 4.5 sq mm

Table 4.2: Latency, throughput and Energy Consumption for different sized SpiCS-Net implemented in IBM 65nm technology.

SRNN size (no. of neurons)	SpiCS-Net		
	Average latency (ps)	Throughput (spikes/s)	Energy per spike (pJ)
4	104 ps	9.6×10^9	0.296 pJ
16	159 ps	6.2×10^9	1.4 pJ
64	223 ps	4.4×10^9	3.76 pJ
128	281 ps	3.6×10^9	8.51 pJ
256	367 ps	2.7×10^9	36.1 pJ

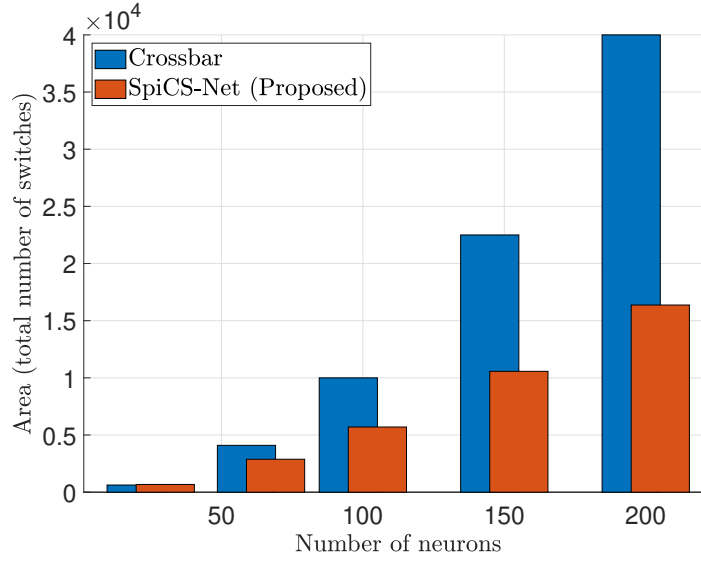


Figure 4.11: SpiCS-Net comparison with circuit-switching Crossbar. Area (Total number of switches) vs. number of neurons in SRNN: Crossbar vs. SpiCS-Net. Here, the fan-in and fan-out of the neurons = 1 to arrive at input $\times$ output NoC configuration. It is observed that as the SRNN size goes up, area savings increase.

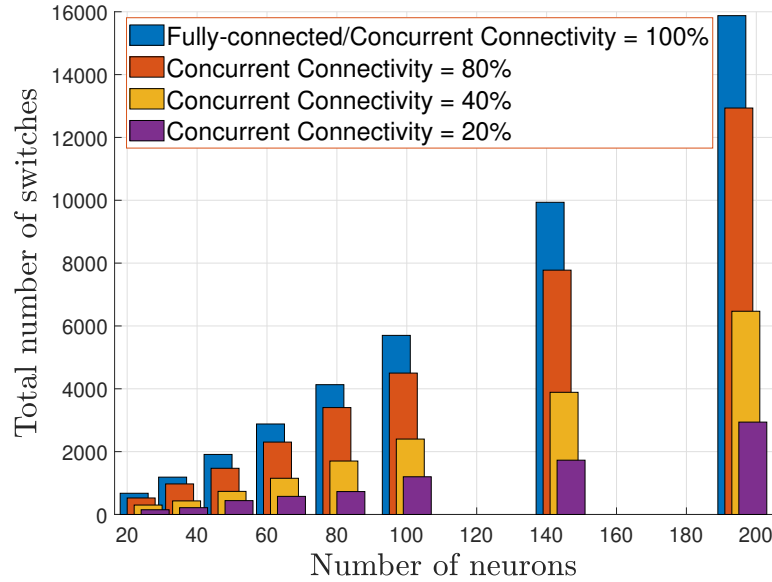


Figure 4.12: The total number of switches for various percentages of Concurrent Connectivity in the SpiCS-Net based NoC for reconfigurable SRNN implementation on hardware. The number of switches relates directly to switch circuits and memory requirements on the chip.

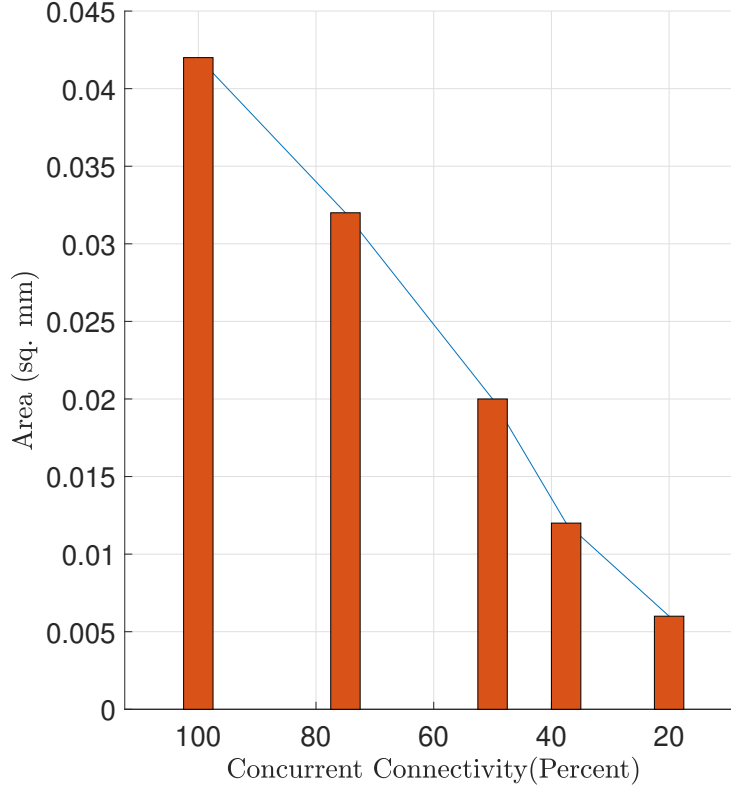


Figure 4.13: Area comparison for various percentages of Concurrent Connectivity in the 128x128 SpiCS-Net in the 65nm process.

Table 4.3: Comparison of area on-chip for varying Concurrent Connectivity and with packet-based Network-on-Chip Architectures

Design	Area(mm^2)	Relative Area
SpiCS-Net* 128x128 $CC_{percent}=37.5\%$	0.012	1x
SpiCS-Net* 128x128 $CC_{percent}=75\%$	0.032	2.6x
SpiCS-Net* 128x128 $CC_{percent}=100\%$ (non-blocking)	0.042	3.5x
SpiCS-Net* 128x128 (strictly non-blocking)	0.08	6.6x
ClosNN 128x128 [41]	0.904	75x
H-NoC 400 neurons [20]	0.587	49x

* SpiCS-Net (this work)

Finally, a comparison with other approaches in the literature is presented in Table 4.4. The power consumption is shown for the Mackey-Glass prediction task where the SRNN was used as a Reservoir in a Reservoir Computing system. The Mackey-Glass time-series prediction task was trained on a Python model, and the power consumption is estimated based on the simulation data for energy per spike from Cadence Spectre for post-layout simulations of SpiCS-Net (Table 4.2).

4.3.2 Analog Spike Communication

When implemented using analog circuits, Spiking Recurrent Neural Networks (SRNNs) can achieve rapid computations by harnessing analog operations, which rely on the flow of electrical charge through specialized devices and circuit structures [72]. Research has also demonstrated that preserving analog information within the spike data leads to increased system efficiency while reducing overall resource consumption [53]. And even though analog computing faces scalability challenges for designing large-scale systems, it excels in small-scale computing applications due to its exceptional computing efficiency in real-time, low-power, and swift data processing tasks, thanks to its reliance on continuous values and minimal energy consumption [93]. This efficiency, combined with rapid computation capabilities, aligns seamlessly with the power constraints of edge computing devices. These devices often require real-time decision-making, and sensor data processing on battery or energy-harvesting resource usage, making power efficiency a critical consideration.

The SpiCS-Net architecture offers a circuit-switched solution that enables "spikes-as-spikes" communication preserving the analog nature of spiking data. This can be achieved by designing switches specifically tailored to facilitate analog spike communication. SpiCS-Net with multiple switch styles is well-suited for analog communication of spiking data. This not only enhances the energy efficiency but also contributes to the biological plausibility of the Spiking Recurrent Neural Network (SRNN) by preserving the analog state of the spike in the time domain [57, 7, 70]. The spike data path, comprising only three switches, ensures efficient communication, resulting in significantly faster computation.

The proposed Network-on-Chip (NoC) architecture for analog spike communication is realized by implementing three distinct types of switches: the transmission gate (TG) switch,

Table 4.4: Comparison with previous Network-on-Chip Architectures for SNNs

	3DNoC-SNN [114]	ClosNN [42]	H-NoC for SNN [20]	Digital SpiCS-Net (This work)
Technology	45nm	45nm	65nm	65nm
Size	3x3x4	128x128	400 neurons	128x128
Switching Technique	Packet-switching	Packet-switching	Packet-switching	Circuit-switching
Packet Size	31-bit	32-bit	48-bit	1-bit
Structure	3D layering	Clos Multistage Network	Hierarchical-Star-Mesh	Clos Multistage Network
Sparseness-aware	N	N	N	Y
Recurrent Connectivity Support	N	N	Y	Y
Area (mm^2)	0.031 (per Router)	0.904	0.587	0.042
Power consumption	13.16mW	0.85mW (ECG)	10.13mW (Wisconsin)	7.5uW (Mackey-Glass prediction; synaptic density = 10%)
Throughput	0.0313 spike/node/-cycle	—	3.33X10 ⁹ spikes per second	3.6X10 ⁹ spikes per second

NMOS single transistor switch, and PMOS single transistor switch. Each switch is designed and laid out individually using Cadence Spectre in TSMC 65nm technology. Abstract views are generated to produce LEF files for these switches. Subsequently, these switches are integrated as black-box modules to synthesize the SpiCS-Net network, designed in Verilog. The Verilog design of SpiCS-Net is entirely parameterized, allowing for the creation of SpiCS-Net of varying sizes. The design parameters remain the same as shown in Table 4.1. An SRNN size of 36 is chosen as realistic for practical applications [93] while being small enough to run post-layout simulations. Notably, for an SRNN size = 36 neurons with fan-in = 2, the number of switches in a crossbar (Fig. 4.3) would have been 2592, while the number of switches in SpiCS-Net = 1080, which is 2.4 times reduction in number of switches. Synthesis for SpiCS-Net is carried out using Synopsys Design Compiler, and the entire design is placed and routed using Cadence Innovus, resulting in the layout of SpiCS-Net. This placed and routed design is finally imported into Cadence Spectre environment to run post-layout simulations of the designed NoC to get the latency, energy, and throughput values. The performance of various switch types and their impact on the analog spike data is presented in Table 4.5. The triangular input spike has a rise/fall time = 500 ns and width = 1us. The frequency response of designed SpiCS-Net with different switches is also presented to gauge the impact on frequency components in the analog spike. A comparison with the various approaches in the literature is also shown in Table 4.6.

4.4 Results

Table 4.4 and Table 4.6 show the comparison of both digital and analog implementations of SpiCS-Net with few packet-based spike communication approaches in the literature. Compared to packet-based approaches, a spike transmitted through SpiCS-Net encounters just three switches in the signal path. Due to this, SpiCS-Net is faster at routing spikes between input and output and thus offers a 9% higher throughput compared to [20]. Throughput refers to the rate at which data can be transmitted across the NoC architecture. In packetized architectures, throughput is evaluated based on the maximum delay between neighboring

Table 4.5: Latency, Throughput, and Energy Consumption for different switches used for Analog SpiCS-Net design implemented in TSMC 65nm technology. Implementation details are provided in Table 4.1. Input spike is triangular signal with width of 1us and rise/fall time 500ns; VDD = 1.2V

	Analog SpiCS-Net				
Switch type	Average latency (ns)	Max peak-to-peak swing (V)	Throughput (spikes/s)	Energy per spike (pJ)	Area (sq. mm)
Transmission Gate (TG)	180 ns	1.19	9.6×10^6	70.8 pJ	0.0123
NMOS switch	153 ns	0.644	6.2×10^6	38.4 pJ	0.008
PMOS switch	221 ns	0.615	4.4×10^6	55.2 pJ	0.0112

Table 4.6: Comparison with previous works

	3DNoC-SNN [114]	ClosRNN [41]	H-NoC for SRNN [20]	TG-based Analog SpiCS-Net (This work)
Technology	45nm	45nm	65nm	65nm
Size	3x3x4	128X128	400 neurons	36x72
Switching Technique	Packet-switching	Packet-switching	Packet-switching	Circuit-switching
Packet Size	31-bit	32-bit	48-bit	1-bit
Structure	3D layering	Clos network topology	Hierarchical-Star-Mesh	Clos network topology
Analog spiking data	No	No	No	Yes
Area (mm^2)	0.031 (per Router)	0.904	0.587	0.012
Power consumption	13.16mW	0.85mW (ECG)	10.13mW (Wisconsin)	62.39uW (Mackey-Glass prediction task)
Throughput	0.0313 spike/node/-cycle	–	3.33×10^9 spikes per second	5.97×10^6 spikes per second

routers. In contrast, circuit-switched schemes measure throughput considering the actual delay between input and output terminals.

Next, we look at the power consumption of the NoC. The entirely combinational logic of SpiCS-Net implies that there is no clock-related dynamic power consumption. Thus, SpiCS-Net has a significantly lower power consumption of 7.5 uW for the Mackey-Glass prediction task using SRNN in a Reservoir Computing system.

Further, SpiCS-Net also omits the extra spike-packet-handling circuits of packetized NoC architectures, which leads to nearly 13x savings in area when compared to [20]. When compared to packet-based approaches, a much lower memory requirement also contributes to the area saving by SpiCS-Net.

Overall, the digital SpiCS-Net architecture reduces the overall power consumption, latency, and area requirement within the SRNN when compared to traditional approaches. It is clear from Table 4.3 and Table 4.4 that SpiCS-Net is highly efficient in terms of area, power, and throughput compared to other approaches in the literature. These savings can be attributed to:

- **Throughput:** SpiCS-Net uses just three combinational switches to establish a connection between two neurons. This implies that the time required for a spike encoder to generate a spike packet in packet-based NoCs would be higher than the spike communication time on SpiCS-Net.
- **Power:** No sequential clock activity in SpiCS-Net leads to low dynamic power.
- **Area:** The hardware needed for switches of SpiCS-Net is orders of magnitude less than routers in packet-based NoCs. This area can be further reduced by leveraging concurrent connectivity. The reduction is almost 3.5x for concurrent connectivity = 37.5% when compared to 100% concurrent connectivity SpiCS-Net (Table 4.3).

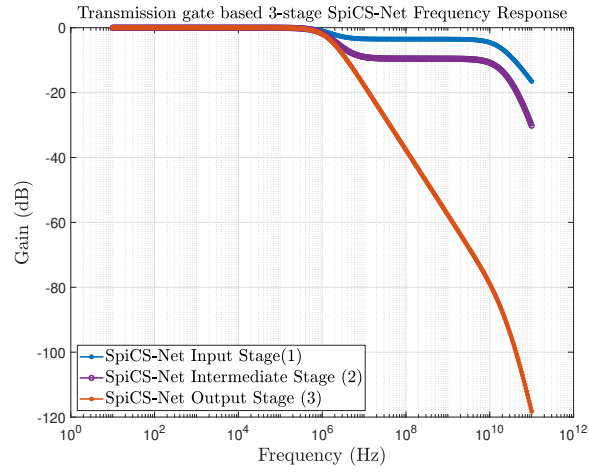
For the Analog SpiCS-Net NoC architecture, Table 4.5 provides comprehensive data on latency, throughput, power consumption, and area requirements, incorporating three different types of switches. Additionally, Table 4.6 presents a comparison of the TG-based Analog SpiCS-Net architecture with spike communication approaches found in the existing literature.

It's important to note that, due to the absence of directly comparable circuit-switched spike-routing architectures in the literature, the comparison is made against packetized communication methods for spike data communication. The comparison with the literature shows that except for the throughput, Analog SpiCS-Net offers much higher efficiency in all other metrics including area and power consumption. This is due to the RC delay of the switches which add to the parasitics of the driven node. The frequency response for all switch types shown in Fig. 4.14 also corroborates this observation. The comparatively low throughput of Analog SpiCS-Net can possibly be justified as being substantially higher than the spike frequency observed on biological timescales [33].

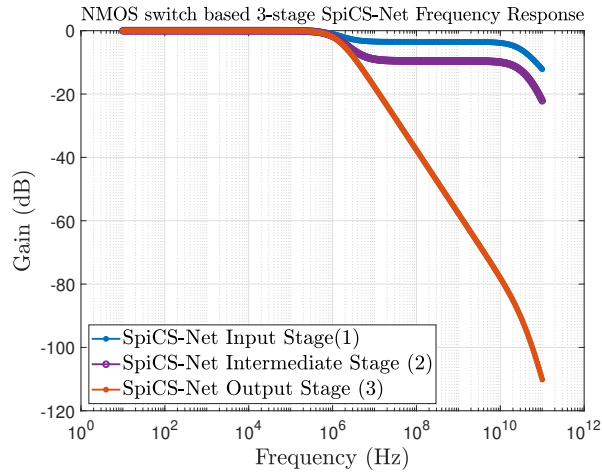
4.5 Conclusion

Spiking Recurrent Neural Network (SRNN) are a potential solution for resource-efficient low-power nonlinear signal processing and particularly excel when utilizing analog timing information of spike data. The spike itself can be digital or analog, the timing of the spike can encode a significantly high density of information when compared to binary data. Implementing these SRNNs on hardware in a reconfigurable manner will not only facilitate continual learning but also enable various application-trained neural networks on a chip for dynamic and diverse computing tasks. However, NoC architecture for routing spikes become progressively challenging on hardware as the size of SRNNs increases. This is true for both analog and digital SRNNs. Traditional packet-based approaches, while scalable, fail to preserve analog spike information.

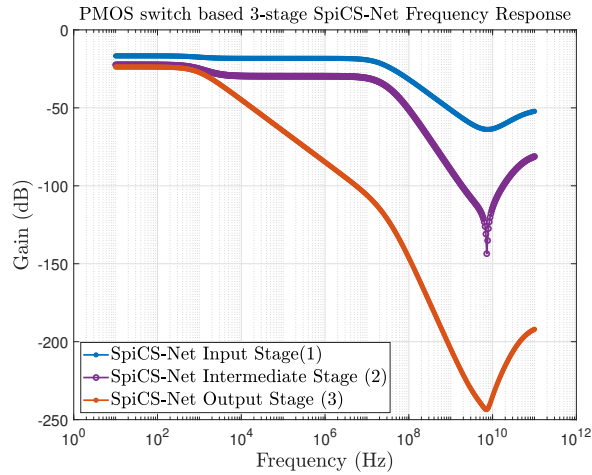
The proposed SpiCS-Net architecture for efficient Network-on-Chip (NoC) implementations can be tailored for both analog and digital SRNNs. This NoC leverages a Circuit-Switching based Multi-stage Interconnect Network (MIN) Clos topology for efficient communication of "spikes-as-spikes". Our findings demonstrate SpiCS-Net offers high area and power efficiency, especially in the case of small SRNNs where neurons on the hardware are closely packed on-chip. One key advantage of SpiCS-Net is its versatility, accommodating both digital and analog spikes. It eliminates clock-associated dynamic power consumption and ensures uniform delay owing to its architectural design. The network also eliminates spike collision or



(a)



(b)



(c)

Figure 4.14: Frequency response at Input(1), Intermediate(2), and Output(3) stages of Analog SpiCS-Net Clos topology shown in Fig. 4.5. (a) transmission gate-based Analog SpiCS-Net (b) NMOS switch-based Analog SpiCS-Net (c) PMOS switch-based Analog SpiCS-Net

loss concerns by offering dedicated wired connections. Results demonstrate that SpiCS-Net achieved remarkable area savings, with up to 13 times less area compared to packetized NoCs. SpiCS-Net promises to open new horizons for efficient and adaptable hardware implementations of spiking neuromorphic hardware.

Chapter 5

Hierarchical Network-on-Chip for Scalable Neuromorphic Architecture

5.1 Background

Neuromorphic computing derives inspiration from the computational primitives and learning mechanisms in the brain [72] and showcases exceptional efficiency and significantly low energy consumption [72] resulting, in part, from the unique event based information processing format that uses voltage spikes [98, 121, 36]. These spikes can manifest as either digital or analog signals, yet they retain information through the analog timing of spike events and occasionally through their waveform shapes.

Large-scale neuromorphic computing systems typically comprise hundreds of thousands of neurons, accompanied by at least an order of magnitude higher synaptic connections [29, 78] to form networks for spike communication across the system. As stated earlier, a substantial challenge for implementation of these neuromorphic systems on hardware is accommodating the extent of synaptic connections required to establish a versatile general-purpose computing paradigm [10, 18, 100]. The desired Network on Chip (NoC) architecture

must handle extensive connectivity space, ensuring robust communication, reconfigurability for general-purpose computing, while balancing performance, resource utilization, and energy consumption. These challenges are particularly daunting because the required number of synaptic connections within the network of neurons increases exponentially with each additional neuron added to the network.

When designing neuromorphic Network-on-Chip (NoC) architectures, another crucial aspect to consider is drawing inspiration from the network structure found in the brain, which aligns with the very essence of "neuromorphic" design philosophy. In the brain, spike communication networks often take the form of cortical columns, characterized by dense connectivity among neurons in close physical proximity and sparse connections to neurons situated farther away in other cortical columns [111]. Prior studies have elucidated that these neural networks in the brain demonstrate a characteristic "small-world network structure" [9, 75]. This can be observed, for instance, in the macaque brain structure shown in Fig. 5.1. The small-world structure in essence corresponds to a locally dense and globally sparse connectivity.

Studies exploring network configurations in the brain have unveiled the remarkable efficacy owing to the small-world architecture [56]. This intricate organization, characterized by dense local connectivity within cortical columns and sparse long-range connections, has proven to outperform both randomly structured and regularly arranged networks. The small-world architecture not only fosters rapid response times but also gives rise to emergent behaviors, showcasing its unparalleled efficiency in facilitating complex computations within the brain's neural networks. As a result, the design of neuromorphic Network-on-Chip (NoC) architectures must carefully account for the dual requirements of spike communication: locally dense and globally sparse connections spanning across distant regions [69]. Both of these communication patterns presents unique requirements and challenges, necessitating a nuanced approach to ensure the efficient transmission of spikes throughout the network while maintaining the integrity of neural information processing.

This aligns with the perspective of hardware implementation, where the physical placement of neurons plays a critical role. This consideration stems from the need to optimize communication pathways and minimize latency between neurons that are not abstract

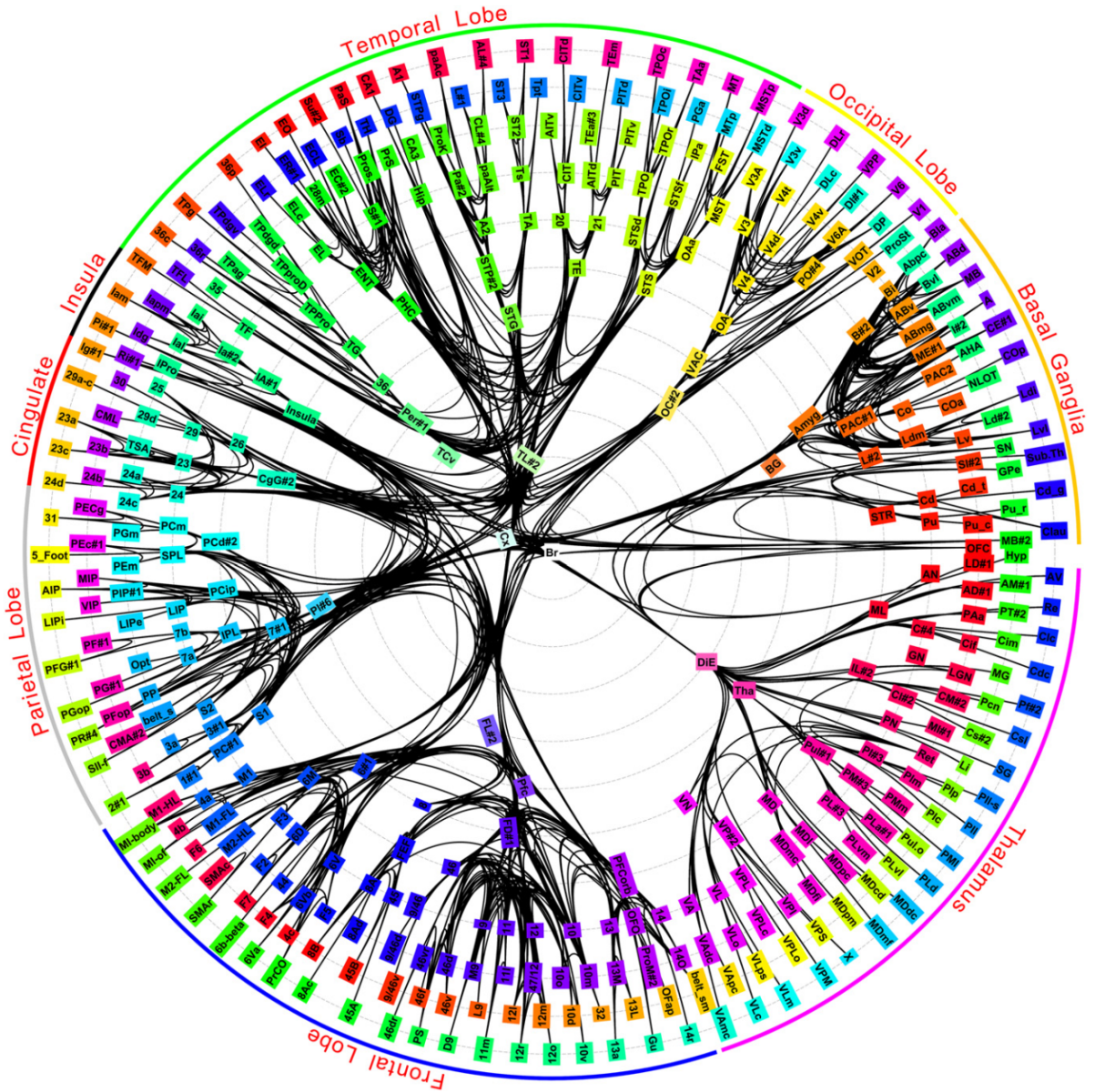


Figure 5.1: Drawing inspiration from the intricate synaptic network of the brain: This hierarchical map of the macaque brain network, as detailed in [75], exhibits small-world network characteristics. It boasts dense local clustering alongside sparse global connections, resulting in efficient short-hop paths.

* The image is from the cited work by Modha [75] and is utilized here with the author's permission.

functions but are physical circuits placed at varying distances from each other on chip. The Network-on-Chip (NoC) should address communication efficiency for neurons located in close proximity on the chip, while also accommodating neurons positioned farther away where wired connections between neurons may be impractical. Notably, this consideration directly aligns with the locally dense and globally sparse connectivity requirements observed in small-world brain network configurations. As a result, orchestrating the mapping of neural networks to hardware can ensure that densely connected neurons are situated in close proximity to each other on the chip, while sparsely connected ones are positioned at greater distances from each other.

Indeed, achieving an optimal neuromorphic NoC architecture involves striking a balance between power efficiency and signal integrity. Local communication benefits from the efficiency of wired connections, allowing for rapid and direct interactions between proximal neurons. In contrast, global communication thrives on packetized communication, which offers robustness particularly for long-range transmissions both on and off the chip. Balancing these two communication modes entails leveraging the strengths of each approach to ensure seamless and efficient information exchange across varying spatial scales within the neural network.

5.2 Hybrid Circuit-Packet-Switched NoC Architecture

Drawing upon the aforementioned considerations, we propose a hierarchical hybrid Network-on-Chip (NoC) architecture that seamlessly integrates circuit-switched mechanisms, ideal for handling dense local connectivity, with packetized schemes optimized for globally sparse spike event communication. While hierarchical approaches have been previously explored in the design of neuromorphic architectures, as demonstrated by [78, 82], but they utilized packetized spike communication. The presented approach distinguishes itself by combining both circuit-switched and packetized strategies, offering a versatile solution tailored to the unique demands of neural network hardware implementations. To the best of our knowledge, the utilization of a hybrid packet and circuit-switched architecture in the connection fabric of neuromorphic architectures has not been explored. The proposed Network-on-Chip (NoC) architecture optimizes resource utilization to facilitate efficient spike communication tailored

to each hierarchical level, as illustrated in Fig. 5.2. At the lowest layer, labeled "R1," a densely connected neuron "cluster" is implemented using a circuit-switched multicast Clos topology. Meanwhile, higher levels of the hierarchy employ packetized Address-Event Representation (AER) communication.

This hierarchical strategy allows for targeted optimization of network performance across different hierarchical levels, accommodating the diverse requirements of various network configurations and enhancing overall efficiency and scalability.

5.2.1 Main Contributions

The main contributions are:

- The innovative hierarchical NoC design involves dividing the Spiking Neural Network (SNN) into a hierarchy of neuron clusters, where the proximate neurons have relatively denser connectivity (R1) and the neurons situated at a greater distance have sparse connectivity thus utilizing higher levels of NoC hierarchy(R2 and R3).
- This innovative hybrid architecture combines circuit and packet-switching for reconfigurable spiking hardware connectivity. Notably, at the R1 level, circuit-switching eliminates the need to convert local spikes to packets, reducing overall resource requirements. Simultaneously, for global spike communication to distant neurons, spikes are converted into packets and transmitted via an Address Event Representation (AER) bus architecture. (Fig. 5.2).
- The proposed NoC is well-suited for the small-world network configuration, leveraging direct "spikes-as-spikes" communication for local dense connections and resorting to spike-to-packet conversion only for sparse connections.
- The lowest hierarchical level utilizes a circuit-switched network with a Clos topology, demonstrating resource efficiency and compatibility with both analog and digital spikes. This approach significantly enhances power efficiency, latency, and throughput performance compared to packet-based communication schemes like Address Event Representation (AER) for locally dense spike communication.

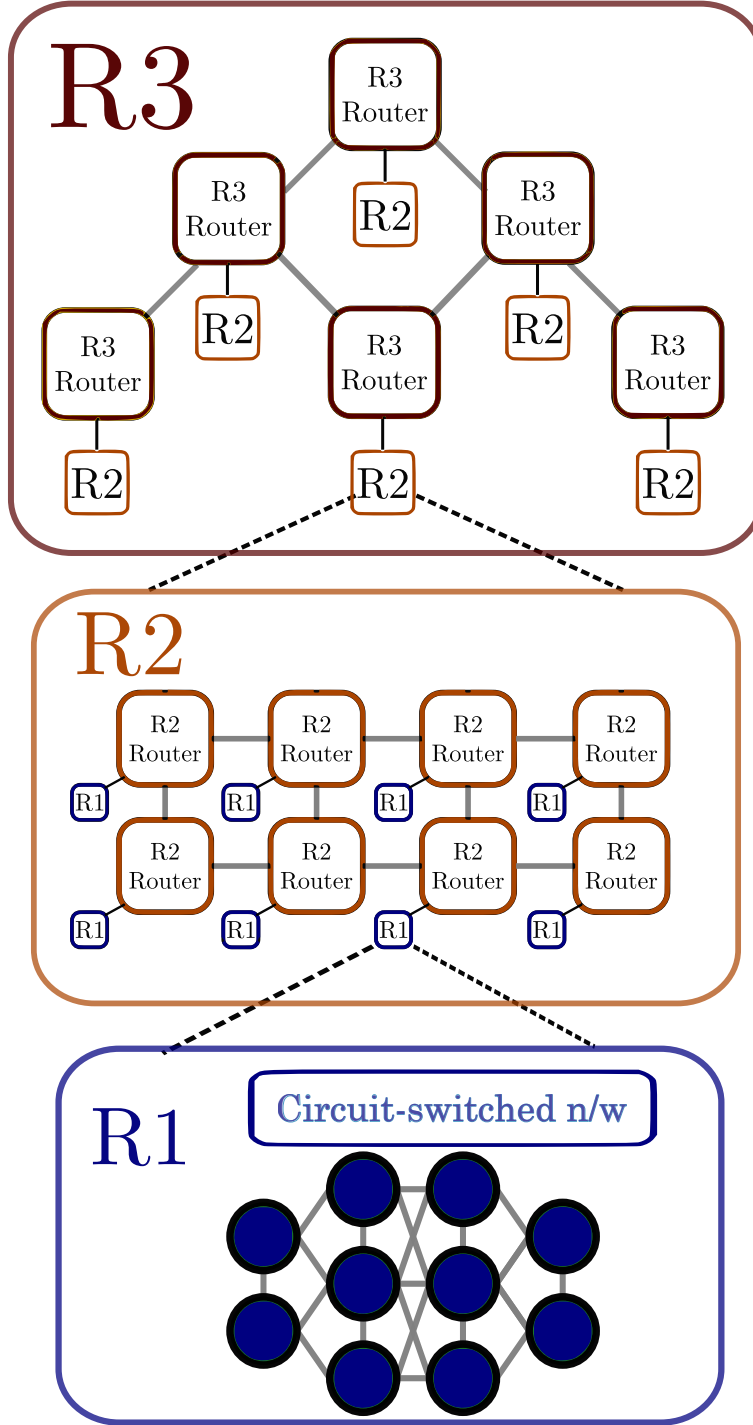


Figure 5.2: The proposed hierarchical circuit-packet network structure consists of three levels: Mesh (R2) and Tree (R3) represent AER-based packetized spike communication architectures. At the lowest level (R1), a circuit-switched Network-on-Chip (NoC) architecture is implemented, grouping neurons in close proximity in both space and time within the same cluster. This approach combines circuit and packet-switching, facilitating efficient dedicated spike communication (circuit-switched) for closely located neurons on-chip with denser connectivity, while also utilizing the flexibility of packet-switching for globally sparse connections. The proposed NoC is efficient for implementing any SNN, adapting to hardware characteristics for both local and global communication, and is particularly effective for those with a small-world network configuration.

- The proposed NoC reduces energy per spike by 100x and gains 1000x in delay and throughput for local communication to proximate neurons in the cluster. The proposed NoC scales better with SNN size and has 1.7x less area compared to AER mesh for an SNN with 256 neurons. The proposed network consumes 6.4x less power compared to existing NoC approaches in literature.

5.2.2 R3 Network Layer - Tree structure (Packet-Switching)

The R3 routing layer represents the highest level designed for larger implementations involving the integration of multiple neural arrays across separate chips. Within this framework, each neural array interfaces with an R3 router, establishing interconnections with other R3 routers in a structured tree arrangement (as shown in Fig. 5.2). The decision to adopt a tree-based routing structure at this level stems from the strategic emphasis on minimizing latency, albeit at the expense of communication bandwidth. Even though this may lead to increase in traffic congestion along designated pathways owing to the intrinsic properties of the tree structure, it's crucial to note that, as discussed earlier, long-distance global spike communication tends to be sparse. This inherent characteristic mitigates the aforementioned traffic congestion issue, as global pathways experience inherently low traffic levels. Moreover, the paramount importance of latency reduction is addressed effectively by the minimized router hops in tree routing structure, thereby optimizing overall latency characteristics of the system.

5.2.3 R2 Network Layer - Mesh Structure (Packet-Switching)

In the NoC architecture, every R2 router is paired with an R1 cluster, creating a structured 2D-mesh setup in the R2 layer (as shown in Fig. 5.2). The second-level hierarchy adopts a packetized approach, ensuring scalability for transmitting spike signals over longer distances. The adoption of a mesh structure at the R2 level is driven by the priority placed on achieving higher bandwidth and fostering robust communication capabilities. Despite the trade-off resulting in a modest increase in latency, this approach strategically addresses the latency challenge by facilitating multiple pathways to the same destination. By offering redundant routes, the network enhances overall bandwidth capacity, thereby effectively mitigating the

impact of increased latency. This design choice underscores a balanced approach aimed at optimizing both latency and bandwidth performance within the network architecture at this level.

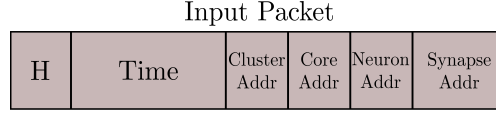
In the R2 layer router uses destination-routing. Routes are determined using combinational logic, triggering a request for one of the cardinal directional outputs. The packet structure shown in Fig. 5.3 encompasses various headers, each defining distinct functions that the packet can serve. The unused headers offer the chance to assign extra functions to the packets as required. The router receives these packets from four directions as they are connected in a mesh topology. This is depicted in Fig. 5.4 as four cardinal directions, East, North, West and South. To prevent deadlocks and packet loss, each directional output includes a look-ahead, round-robin arbiter mechanism.

5.2.4 R1 Network Layer - Clos Topology (Circuit-Switching)

To establish connections within the R1 cluster, we utilize a non-blocking circuit-switching Clos topology. This particular topology is selected for its reconfigurability and ability to provide full connectivity similar to a crossbar. This topology not only achieves this with fewer switches by incorporating multiple crossbar stages, but also ensures efficient spike routing within the cluster while optimizing resource utilization, striking a balance between reconfigurability and scalability [26].

The Clos topology inherently ensures uniform delay and throughput across all links or channels within the network [51]. This characteristic makes it an ideal choice for multicast operations and, when necessary, the broadcast of spikes throughout the cluster. In scenarios involving densely packed neuron clusters, both spatially and temporally, the Clos network architecture facilitates digital/analog agnostic spike communication. This versatility is achieved through the selective deployment of switch architectures that can be tailored to the nature of the spikes being transmitted. For instance, a MUX-based switch architecture is suitable for handling digital spikes, while a transmission gate-based switch is preferable for analog spikes.

Moreover, the asynchronous nature of the network design contributes to its biological plausibility by preserving analog spike states and information in the time domain [57]. This



H = 000 (Apply Spike)
 H = 001 (Run)
 H = 010 (Reset)
 H = 011 (Program)
 H = Others (Unused)

Figure 5.3: AER packet format at the R2 level* is illustrated. The multiple AER packet types are discernible based on specific header bits.

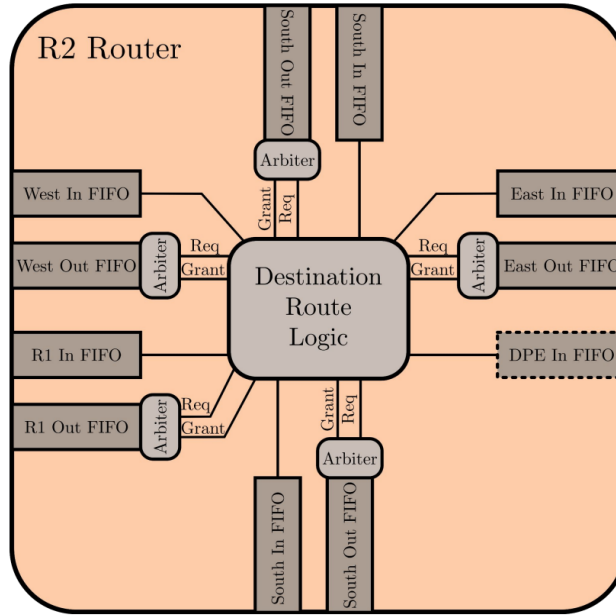


Figure 5.4: R2 layer router* is a mesh router employing destination-routing. Routes are determined by combinational logic, triggering a request for one of the cardinal directional outputs. Each directional output features a look-ahead, round-robin arbiter to prevent deadlocks and packet loss.

* Acknowledgment: The R2 router design is done by Adam Foshie at TENNLab Research group.

asynchronous operation not only aligns with the natural dynamics of neural systems but also results in faster computations, enhancing the overall efficiency of the network.

In Fig. 5.5, we present the adopted Clos topology, showcasing its intricate configuration. Additionally, Fig. 5.6 illustrates the seamless integration of circuit-packet communication. This integration approach is chosen for its simplicity in implementation and spike synchronization, ensuring non-interference between both spike communication modes.

The R1 router within the integrated system, depicted in Fig. 5.6, is responsible for converting data packets from the R2 level into spikes, subsequently routing them to the respective neurons. These Address-Event Representation (AER) packets received by the R1 router contain crucial information, including the address of the neuron and its specific synapse port.

It’s important to note that internal connections between neurons within the same cluster are exclusively established through the R1 circuit-switched Clos NoC. This ensures efficient and reliable communication within the densely packed clusters, contributing to the overall performance and functionality of the neural network architecture.

5.2.5 Network Partitioning

In the proposed hierarchical NoC design for a spiking neural network (SNN), the network is partitioned into smaller clusters. The size of these clusters can be intuitively correlated with the local and global clustering coefficients of the SNN [116, 109]. These coefficients are indicative of the network’s propensity for neurons to form clusters or tightly interconnected groups. By aligning the cluster size with these coefficients, we aim to optimize the network architecture to better emulate the natural clustering behavior observed in biological neural networks. This approach allows for more efficient communication and computation within the network, enhancing its overall performance and biological plausibility. For a given neuron n with s_n neighboring neurons in a network, there could be a maximum of $\frac{s_n \cdot (s_n - 1)}{2}$ connections between them. C_s is the fraction of these maximum allowable connections that actually exist between the neighbors of neuron n . The local clustering coefficient is the average of the fraction of maximum allowable connections that actually exist between the neighbors of neuron over all neurons in the network. And, the global clustering coefficient is the ratio of closed

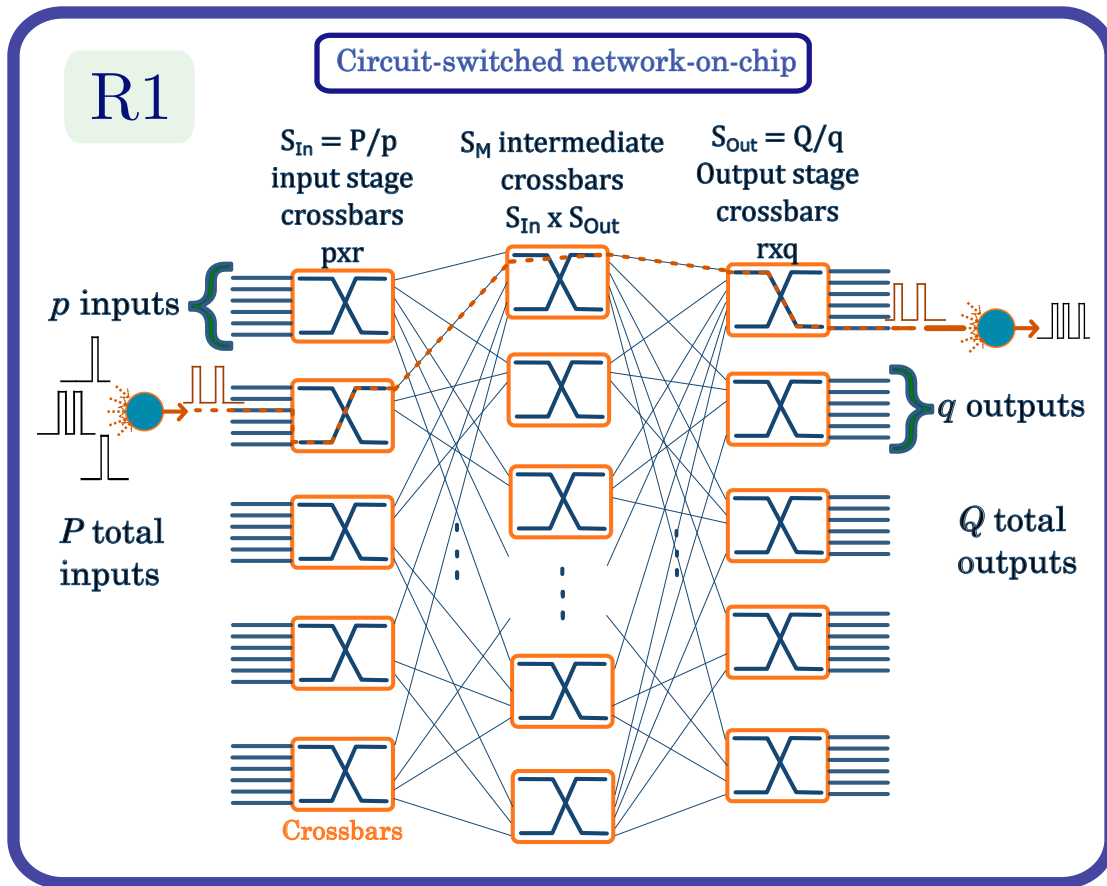


Figure 5.5: Clos circuit-switched network at R1 level in the NoC hierarchy with three switch stages, showcasing the number of inputs, outputs, and switches in each stage.

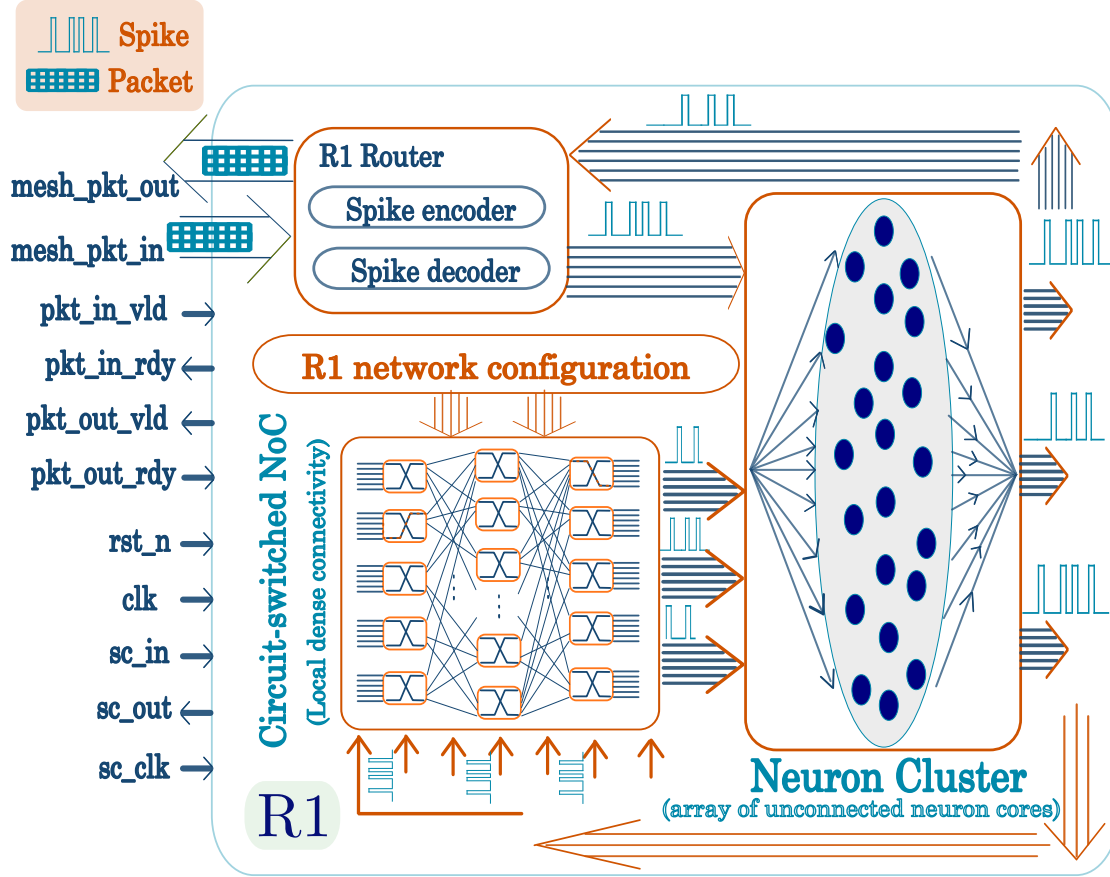


Figure 5.6: Integration of R1 circuit-switched NoC with mesh R2 level. R1 is used to only establish local connectivity ensuring direct "spikes-as-spikes" communication within the cluster. AER packets converted to spikes from the R2 level are routed to the neurons directly through an R1 router.

triplets to all triplets in a network, offering a comprehensive measure of clustering. A high global coefficient indicates a network where nodes form many interconnected clusters. The cluster size N_c can be determined by combining N and the clustering coefficient. Alternatively, partitioning algorithms can also be employed to effectively divide a neural network into clusters with the aim to maximize connection density within clusters while minimizing inter-cluster connectivity. By leveraging such algorithms, we can strategically organize the network structure to enhance local processing within clusters while minimizing long-range communication requirements.

5.3 Implementation

The final design encompasses the Reconfigurable Neuromorphic Array (RNA), which integrates two levels of the proposed NoC hierarchy (R1 and R2 layers), alongside a RISC-V core. Communication between the RISC-V core and RNA is facilitated through an AXI-Lite interface, while direct interfacing between the RNA and off-chip components is achieved via the Serial Peripheral Interface (SPI).

The digital implementation of the RNA is developed in Verilog Hardware Description Language (HDL). Each neuron within the Spiking Neural Network (SNN) features 16 synaptic inputs and functions as a time-division multiplexed neuron with four time slices. Although spikes are digital in nature, the timing of spikes captures analog information, which is preserved within the delay-agnostic R1 layer.

The realization of the R1 layer’s circuit-switched Clos topology is achieved through multiplexer-based crossbars, ensuring efficient routing and communication within the neural network architecture. This comprehensive design framework embodies a balanced integration of digital and analog functionalities, enabling efficient and robust neural computation while leveraging the benefits of hardware acceleration.

The entire design is synthesized to the 65nm IBM technology utilizing Synopsys Design Compiler. For this synthesis process, a custom standard cell library developed in-house is employed. Following synthesis, placement, and routing tasks are performed using Cadence Innovus, providing valuable area estimates.

To assess performance metrics such as latency, power consumption, and throughput for varying cluster sizes, simulations are conducted using the parasitic extracted netlist in Cadence Virtuoso. These simulations offer crucial insights into the behavior of the system under different configurations. Furthermore, the layout of the entire RNA system is shown in Fig. 5.7, providing a comprehensive depiction of the physical arrangement of components within the design.

In the R1 network, each neuron is connected to a spike decoder at its output, responsible for transforming spikes into Address-Event Representation (AER) packets. These spike decoders interface with an associated AER router within the R2 level mesh structure, establishing connections with other routers in each of the four cardinal directions. To manage multiple packets arriving simultaneously, the router integrates a round-robin style arbiter.

Furthermore, a spike encoder is situated at the inputs of each neuron to convert AER packets intended for that neuron back into spikes. It's worth noting that the implementation presented here is for a fully connected, non-blocking cluster R1 network which ensures efficient communication and full-connectivity within the cluster, facilitating seamless information exchange and processing among neurons.

The AER-based network, employed for comparison, adopts a straightforward mesh topology derived from the R2 network without incorporating additional complexities such as error detection, error correction, or flow control mechanisms. In this setup, each neuron is connected to a spike decoder at its output, which transforms spikes into Address-Event Representation (AER) packets. These spike decoders interface with an associated AER router within the mesh, establishing connections with other routers in each of the four cardinal directions. To manage multiple packets routed in the same direction, the router integrates a round-robin style arbiter. Moreover, a spike encoder is positioned at the inputs of each neuron to convert AER packets intended for that neuron back into spikes. The AER-based network is implemented in Verilog HDL and synthesized to identical 65nm IBM technology. The neurons utilized in both networks remain consistent throughout the design process. The design is placed and routed using Innovus to derive area estimates, and the gate-level netlist is employed for simulating network latency and throughput. Furthermore, the parasitic extracted netlist is utilized to estimate power consumption for each network component

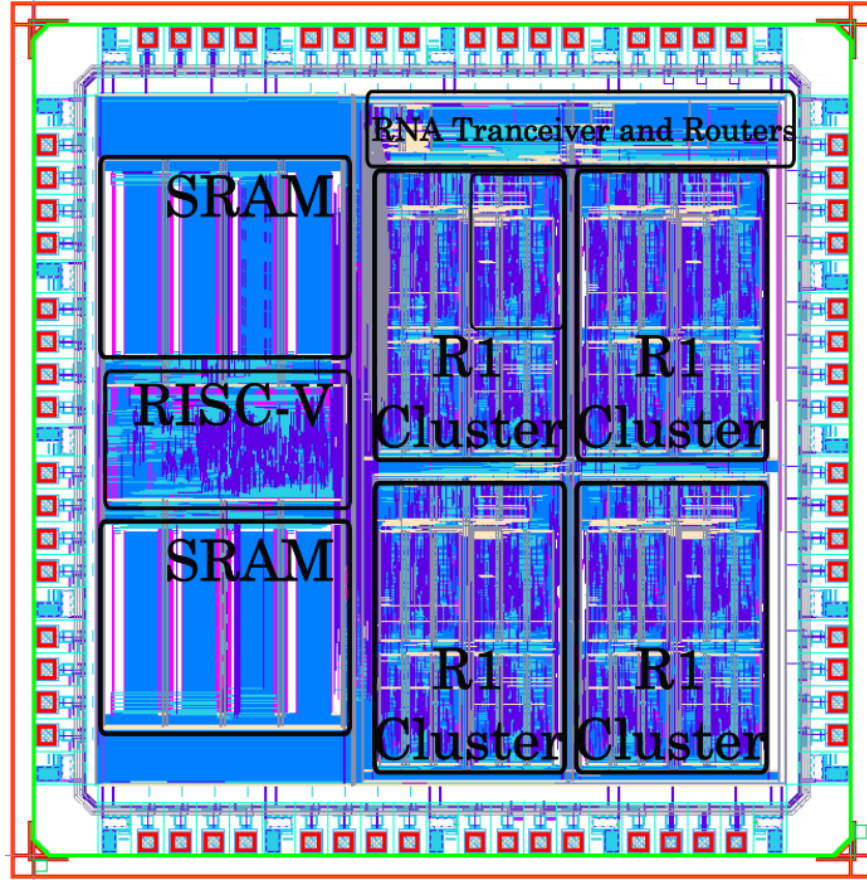


Figure 5.7: Final layout* of implemented design including RISC-V and Reconfigurable Neuromorphic Array (RNA), employing the hierarchical circuit-packet network-on-chip architecture, comprises four R1 layer clusters connected by a mesh AER packetized NoC at the R2 layer.

* Acknowledgment: This is a product of collaborative effort of multiple graduate students at TENNLab Research group.

individually. This comprehensive approach ensures accurate comparison and evaluation of both network architectures.

5.4 Results

As depicted in Fig. 5.8, the energy comparison between the Clos network and the AER network reveals significant differences. The AER network employs a sequential implementation, requiring multiple clock cycles for spike transmission. In contrast, the Clos network operates entirely on a combinational basis, devoid of clock-related dynamic power. This leads to energy consumption per spike for the Clos NoC being nearly 100X lower than that of the AER network when routing a spike between neighboring neurons. Furthermore, this energy advantage extends to around 140X when routing a spike between opposite corners of the cluster. Additionally, the leakage power values presented in Table 5.3 further underscore the low energy consumption advantage of the Clos circuit-switched NoC when idle. This substantial energy efficiency of the Clos network, attributed to its combinational operation and optimized routing, underscores its potential for power-sensitive applications, offering a compelling alternative to conventional AER-based architectures.

In Fig. 5.9, the latency comparison between the Clos and AER networks across various-sized clusters highlights the superiority of circuit-switched dedicated paths in quickly routing spikes. In contrast, the latency in the AER network is contingent on the multi-bit data path that the packet needs to traverse. Specifically, AER's "Best Case" scenario involves communication with neighboring neurons, whereas the "Worst Case" scenario entails communication with diagonally opposite neurons in the mesh. Notably, the worst-case latency scales as $2n - 1$ with the number of nodes, n , in the AER mesh.

Furthermore, in Fig. 5.10, it is observed that the throughput of the AER network remains relatively constant even as the network size grows. Despite AER's best-case latency and throughput remaining stable regardless of network size, the circuit-switched NoC consistently exhibits lower latency and higher throughput for proximal neurons. This comprehensive analysis underscores the advantages of the circuit-switched NoC in terms of both latency

Table 5.1: Latency, Throughput comparison for different sized SRNN implemented using AER packetized mesh-based NoC vs. a fully-connected R1 layer Clos circuit-switched NoC. Digital spike signal with voltage swing = 1.2V and pulse width = 50 ns with $t_{rise/fall} = 50$ ps is utilized.

Cluster size (no. of neurons)	Clos network		Traditional AER			
	Average latency (ps)	Throughput (spikes/s)	Worst Case Latency (μ s)	Worst Case Throughput (spikes/s)	Best Case Latency (μ s)	Best Case Throughput (spikes/s)
4	104 ps	9.6×10^9	16	1.43×10^5	9	1.43×10^5
16	159 ps	6.2×10^9	44	1.43×10^5	9	1.43×10^5
64	223 ps	4.4×10^9	100	1.43×10^5	9	1.43×10^5
256	367 ps	2.7×10^9	212	1.43×10^5	9	1.43×10^5

Table 5.2: Energy/Power consumption per spike transmission

Cluster size	Clos	AER (Worst Case)	AER (Best Case)
4	0.296 pJ	107.1 pJ	60.2 pJ
16	1.4 pJ	295.4 pJ	60.2 pJ
64	8.51 pJ	670.6 pJ	60.2 pJ
256	102.1 pJ	1423.8 pJ	60.2 pJ

Table 5.3: Area and Leakage power for NoC router modules at R1 and R2 levels in proposed hierarchical NoC design

Module	Area	Leakage Power
R1 circuit-switched NoC	0.047 mm ²	107.604 nW
R1 router	0.102 mm ²	891.524 nW
R2 router	0.175 mm ²	4.6 μ W

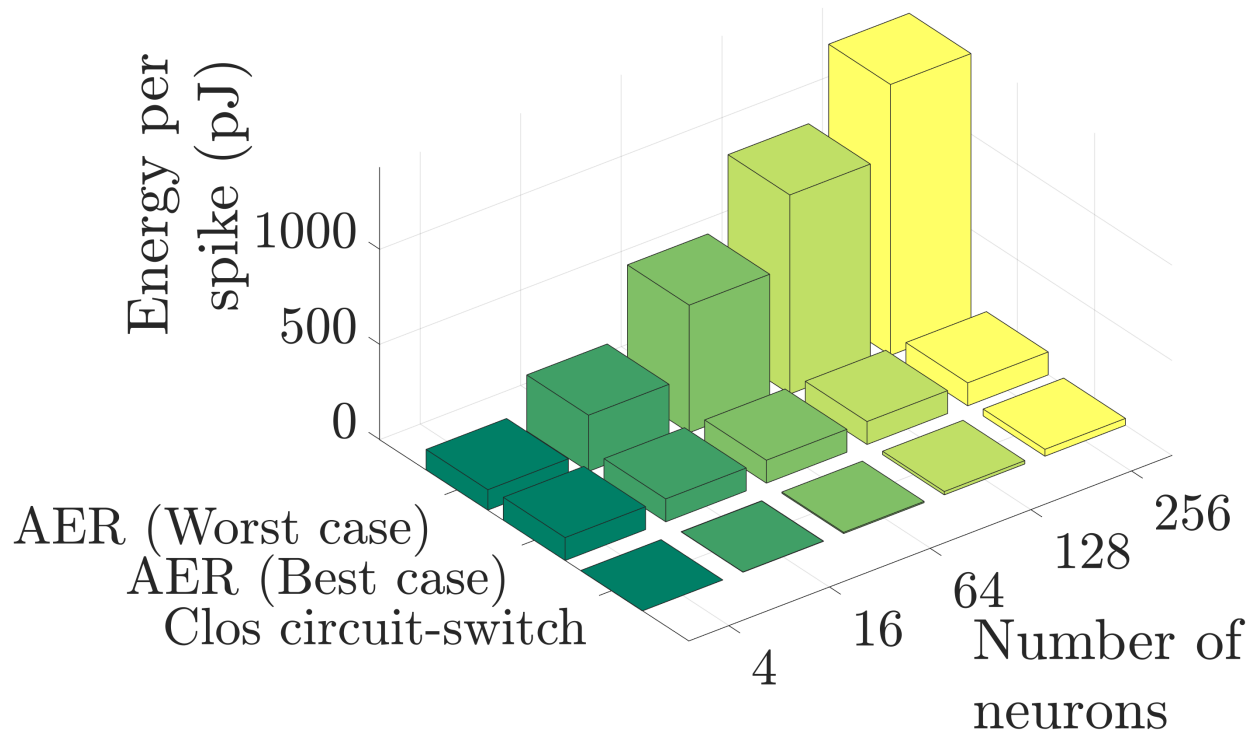


Figure 5.8: Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size

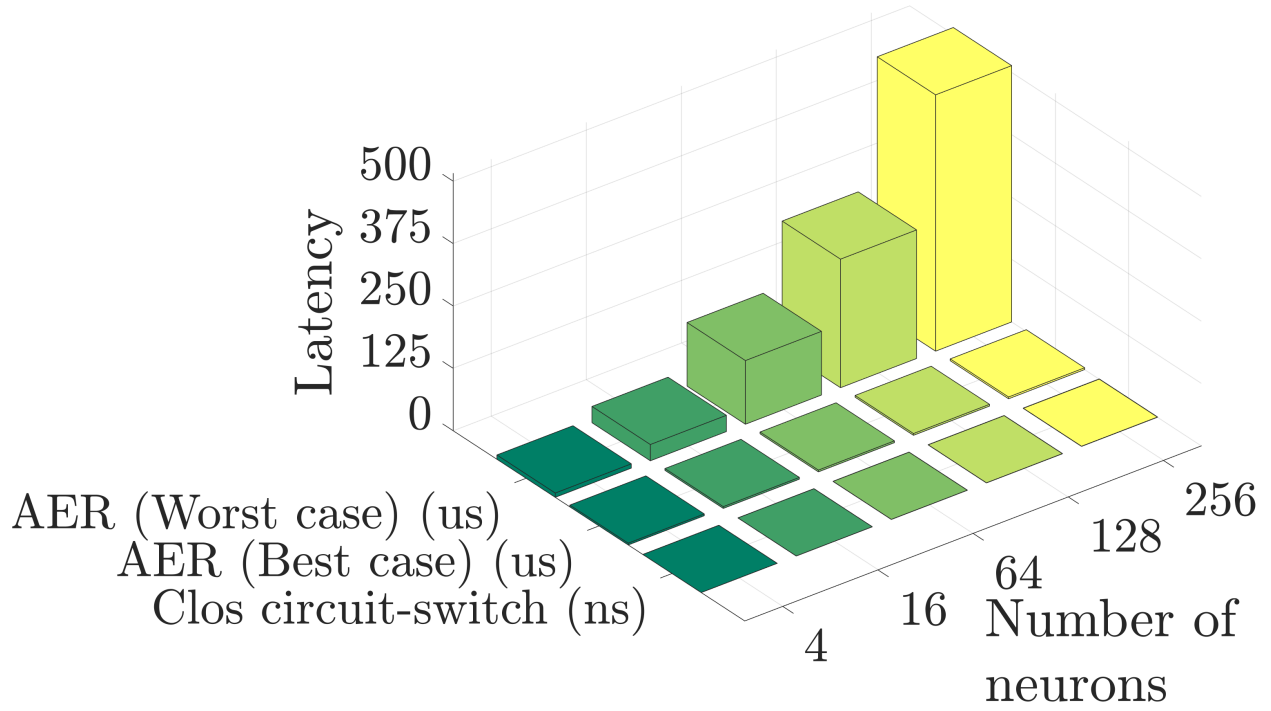


Figure 5.9: Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size

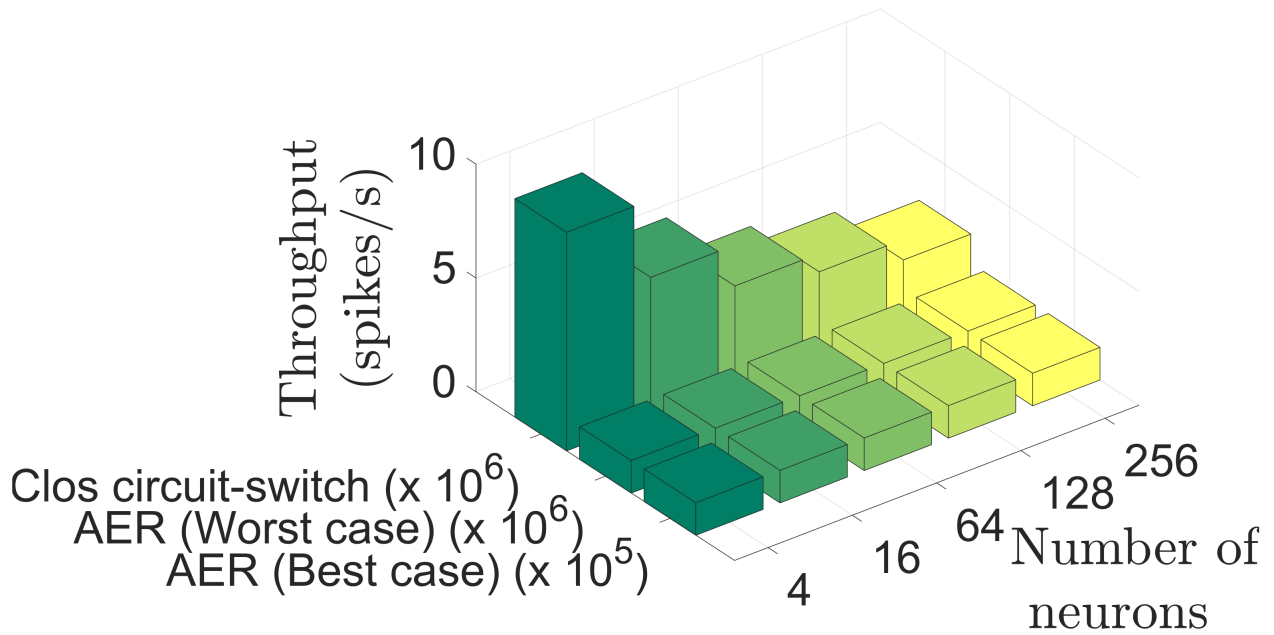


Figure 5.10: Comparing circuit-switched clos topology and AER mesh packetized networks across different neural network sizes. (a) Energy per spike vs. cluster size (b) Latency vs. cluster size (c) Throughput vs. cluster size

and throughput, demonstrating its suitability for efficiently handling spike communication in clustered neural network architectures.

In Fig. 5.11, the comparison of area utilization between AER packet-switched, Clos circuit-switched, and the proposed hybrid hierarchical network with one R2 router and four R1 clusters is presented. Remarkably, the proposed hierarchical hybrid circuit-packet NoC demonstrates superior scalability as the size of the Spiking Neural Network (SNN) increases, outperforming both AER packet-based and purely circuit-switched Clos networks. This comparative analysis underscores the efficiency and effectiveness of the proposed hybrid hierarchical network architecture, highlighting its potential to accommodate larger SNNs while minimizing area overhead. This scalability advantage positions the hybrid network as a promising solution for future neural network implementations, offering a balance between area utilization and network performance.

In Table 5.3, the area and leakage power of various routing modules utilized by the proposed circuit-packet hierarchical NoC are detailed. Notably, the circuit-switched network demonstrates high efficiency in terms of area and leakage power, with only 107.6 nW leakage, significantly lower than the R1 and R2 router modules, which exhibit 8x to 42x higher leakage. This efficiency is attributed to the utilization of small combinational switches within the circuit-switched Clos NoC.

As a result, employing the circuit-switched network for local dense communication yields substantial savings in both area and power, with only sparse communication allocated to the packetized R2 layer. This strategic allocation of communication resources optimizes energy utilization and enhances overall system efficiency.

Furthermore, Table 5.4 highlights this effectiveness by considering the energy per spike metric. Within-cluster communication requires 100 times less energy per spike compared to the best-case scenario for cluster-to-cluster communication. This disparity underscores the efficiency gains achieved by leveraging the hierarchical structure of the proposed circuit-packet hybrid NoC architecture. It is important to acknowledge the scalability limitation of purely circuit-switching NoCs. This challenge is effectively addressed here by incorporating packetized higher communication layers in the proposed hierarchical circuit-packet hybrid NoC, ensuring scalability while preserving efficiency and performance.

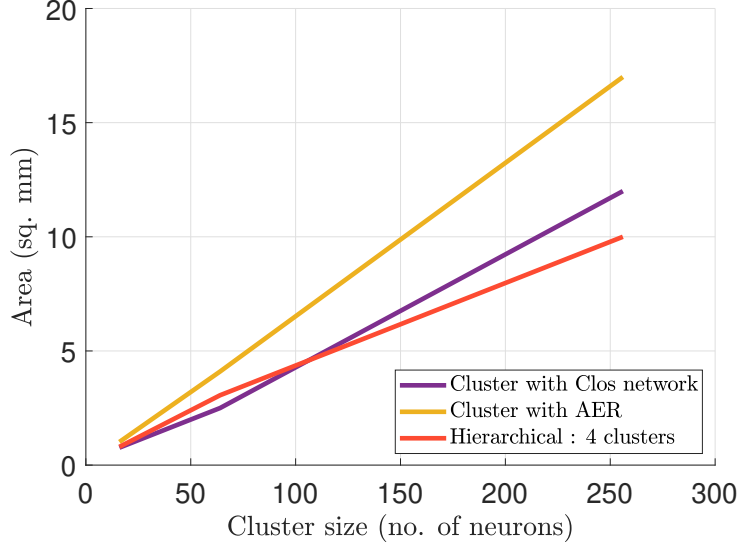


Figure 5.11: Area plot against SRNN/cluster size for various NoC approaches reveals significant area overhead with AER packet-based NoC. The Clos network outperforms AER, but the hybrid hierarchical circuit-packet scheme excels above both by scaling much better as the system size increases. These area numbers include placed and routed layout than includes neurons along with NoCs. The hybrid network data corresponds to dividing the SNN into 4 clusters using the hierarchical R1 and R2 levels as shown in Fig. 5.2.

Table 5.4: Cluster-to-cluster and Intra-cluster spike communication in proposed hierarchical NoC with R2_mesh_size = 4 and R1_network_size = 16X64

Spike communication	Energy per spike (pJ)	Latency	Throughput (spikes/s)
Cluster-to-Cluster (Best case)	756.56 pJ	3.08 μ s	19.51 x 10 ⁶
Cluster-to-Cluster (Worst case)	1056.11 pJ	3.91 μ s	19.51 x 10 ⁶
Within Cluster (circuit-switched)	7.44 pJ	223 ps	4.4 x 10 ⁹

All results presented are derived from post-layout simulations. It is evident from these results that the Clos network architecture effectively reduces overall power consumption and communication delay during spike events within the cluster. However, as the cluster size increases, the power and latency of the Clos network escalate significantly compared to the AER implementation. Moreover, larger Clos networks may incur substantial interconnect parasitics, leading to heightened delay and potential degradation of spike signals.

Furthermore, considering the sparse global connections requirement, employing a large Clos network may not be resource-efficient. This observation is underscored by Fig. 5.11, which highlights that the optimal approach involves dividing the overall design into smaller-sized, tightly-packed neuron clusters. This strategy not only mitigates the scalability challenges associated with large Clos networks but also enhances resource utilization and network efficiency.

Finally, Table 5.5 provides a comprehensive comparison of the proposed NoC with existing literature. In this comparison, the estimated power consumption per spike obtained from simulations in Cadence Spectre is utilized to calculate the estimated power for the Mackey-Glass chaotic time-series prediction task, employing Reservoir Computing as a learning technique.

Although the throughput of the proposed hierarchical NoC may be lower, limited by the R2 routing layer, the power consumption of the proposed NoC is significantly lower compared to other approaches. Specifically, the power consumption is found to be 6.4 times lower than alternative methods. This noteworthy reduction in power consumption underscores the efficiency and effectiveness of the proposed NoC architecture, positioning it as a promising solution for energy-efficient neural network implementations.

Overall, Table 5.5 highlights the competitive performance and efficiency gains achieved by the proposed NoC architecture, further validating its potential for diverse applications in neural network computing tasks.

Table 5.5: Comparison with previous Network-on-Chip Architectures for SNNs

	3DNoC-SNN [114]	ClosNN [41]	H-NoC for SNN [20]	Circuit-Packet Hier NoC (* This work)
Technology	45nm	45nm	65nm	65nm
Size	3x3x4	128x128	400 neurons	128 Neurons
Switching Technique	Packet-switching	Packet-switching	Packet-switching	Circuit-Packet-hybrid
Packet Size	31-bit	32-bit	48-bit	1-bit/16-bit
Structure	3D layering	Clos packetized Network	Hierarchical-Star-Mesh	Hierarchical Clos and AER Mesh Circuit-Packet NoC
Area (mm^2)	0.031 (per Router)	0.904	0.587	0.601
Power consumption	13.16mW	0.85mW (ECG)	10.13mW (Wisconsin)	132.29uW (Mackey-Glass prediction; 2 clusters with 64 neurons each)
Throughput	0.0313 spike/node/-cycle	—	3.33X10 ⁹ spikes per second	19.51X10 ⁶ spikes per second

5.5 Discussion

The presented novel approach to designing neuromorphic network architectures combines packet- and circuit-switched approaches for spike communication hierarchically. We further propose using Clos topology at the lowest hierarchy level for spike communication among tightly packed neuron clusters. It is clear from the results presented that using the proposed circuit-packet hierarchical Network on Chip (NoC) is highly resource-efficient while providing higher performance overall when compared to either a purely circuit-switched or purely packet-switched network. The proposed NoC is highly efficient for implementing any Spiking Neural Network (SNN), but is particularly effective for SNNs with small-world network properties. The results demonstrate that the presented NoC achieves the best performance with minimal trade-off by using circuit- and packet-switching techniques where they perform best. Thus, dividing the neurons in the SNN into smaller tightly packed clusters with a high degree of connectivity and using AER for more global sparse communication provides benefits of both approaches and the results make a strong case for dividing the neuron into a hierarchy of clusters with the lowest level using Clos for intra-cluster communication. The NoC is extremely efficient due to more dense communications keeping "spikes-as-spikes"; and only the sparse communication between clusters would be converted to AER packets. The results show that, for local spike communication within the cluster, the circuit-switched R1 layer NoC consumes nearly 100x less energy per spike, has a 1000x lower delay, and has higher throughput compared to a spike being communicated from one cluster to another using AER mesh packetized approach.

This network design approach is promising for the implementation of neuromorphic systems, and is biologically plausible and efficient for both local and global spike communication. It adapts to hardware limitations, using dedicated circuits for nearby neurons and a packetized approach for distant ones. An analogy can be drawn to handing an item to a neighbor (circuit-switched) versus sending a package to someone 500 miles away (packet-switched requiring a delivery system and robust packaging to prevent loss or damage). Compared to other NoC approaches in the literature, the power consumed by the proposed NoC is up to 6.4 times lower.

Clustering coefficient metrics and existing partitioning algorithms could be employed to map the neural network design to tightly packed dense clusters and sparse connectivity between clusters. The circuit-packet hierarchical network is also well suited for approaches that employ deep and split SNN architectures that limit the recurrent connectivity to intra-cluster communication and only have feed-forward connections to other clusters. The proposed hybrid NoC approach directly addresses the connectivity challenges of neuromorphic architectures to make them more scalable, reconfigurable, and efficient at the same time.

Chapter 6

Architecture for Online Learning

6.1 Background

Online learning on neuro-inspired hardware introduces the potential for continual learning within the deployed system, a highly valuable capability for applications operating in dynamic environmental conditions [5]. It also proves indispensable for systems necessitating re-training functionality to adapt to different tasks and scenarios. This adaptive approach ensures that the system can evolve and improve its performance over time, making it a vital component in versatile and ever-changing computing applications [81, 99]. Traditional learning models often involve static, pre-trained algorithms that do not adapt to changes in their operating environment [94]. In contrast, online learning allows a system to continually update and refine its knowledge and performance as it encounters new data and experiences [6]. This is particularly valuable in scenarios where the system must handle a wide range of tasks or adapt to different usage needs. For example, in robotics, a robot equipped with online learning can learn new tasks as it encounters them, allowing it to continually expand its capabilities. The ability to adapt and refine the learned policies in real-time is essential for optimizing performance and achieving objectives efficiently.

Online learning in neuromorphic systems poses a formidable challenge due to its significant computational demands [84, 5, 112]. The process typically involves iterative machine learning operations to establish and adjust the neural network’s parameters, which can be resource-intensive and time-consuming. This challenge is exacerbated by the need for real-time processing and adaptation to dynamic environments. Additionally, in the case of Spiking Recurrent Neural Networks (SRNNs)-based neuromorphic systems, the training of recurrent connections further presents significant computational challenges for hardware implementation [66].

Reservoir Computing (RC) emerges as a promising solution to alleviate the computational demands imposed by recurrent connections. Unlike traditional approaches, RC focuses training efforts primarily on a subset of connections, notably the feed-forward output layer of the network [66, 67, 47]. Fig. 6.1 depicts the architecture of a traditional Reservoir Computing system, illustrating the feed-forward output layer with the reservoir Spiking Recurrent Neural Network (SRNN). The reservoir is initially configured with random initialization and maintains this configuration throughout runtime. Meanwhile, the output layer, responsible for implementing a dot product between reservoir inputs and output layer weights, undergoes iterative adjustments to minimize the discrepancy between the expected output and the instantaneous output.

Despite alleviating the computational load associated with training recurrent connections, Reservoir Computing (RC) implementations still encounter obstacles due to the resource-intensive training process and the implementation of output layer operations, specifically the multiply and accumulate circuitry [65]. Both of these modules are shown in the Fig. 6.1 as the "Training Algorithm" and the "Readout Layer" blocks. These challenges become more pronounced as the reservoir SRNN sizes grow, thus, exacerbating the computational demands on the hardware implementation. Hence, two primary challenges arise for reconfigurable and scalable implementations: the first pertains to the hardware responsible for implementing the learning algorithm to tune weights, while the second concerns the multiply-and-accumulate circuitry. Thus, hardware implementations of RC must confront these challenges while ensuring characteristics such as real-time processing, low power consumption, and adaptability to dynamic environments.

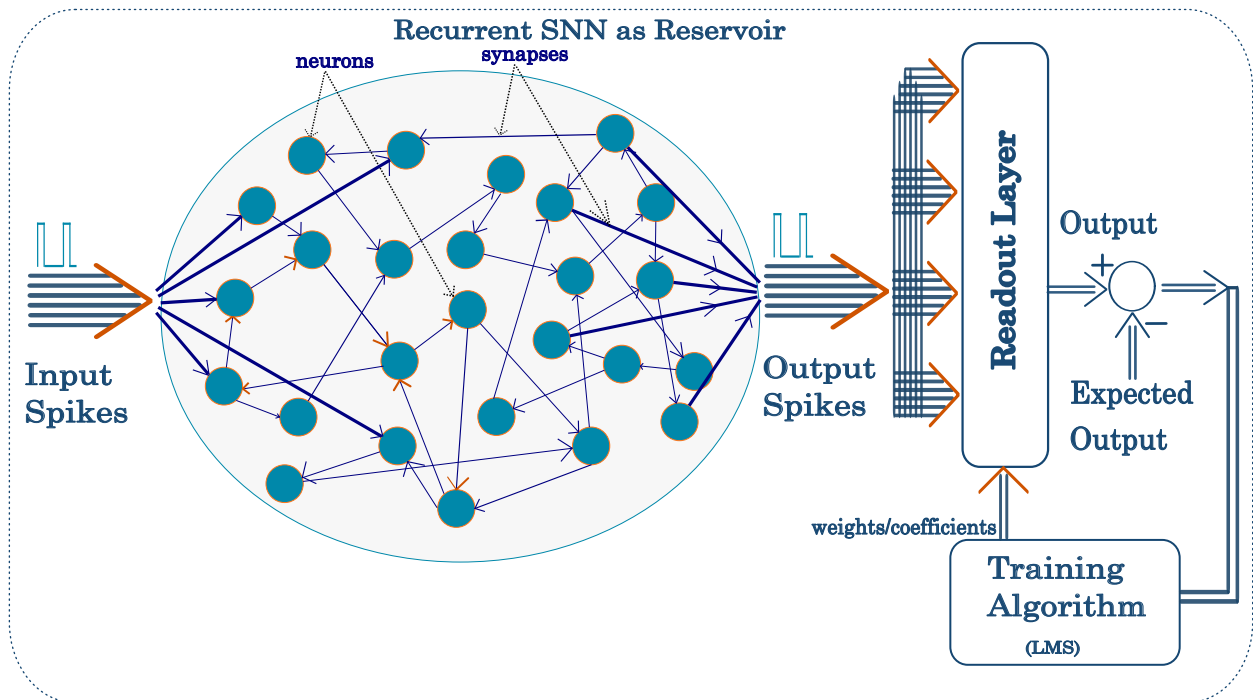


Figure 6.1: The image illustrates the architecture of a generic Reservoir Computing system, featuring the feed-forward output layer depicted on the right and the reservoir Spiking Recurrent Neural Network (SRNN) with recurrent connections shown on the left. The reservoir is initially configured with random parameters that remain unaltered throughout the training process. In contrast, the output layer typically comprises a dot product layer, with its weights adjusted iteratively by minimizing the discrepancy between the expected output and the actual output at each time step.

Many works in the literature address the resource efficiency challenge of the multiply and accumulate circuitry in the Readout layer by utilizing emerging devices like memristors as an analog computing array [65, 73, 101]. Nevertheless, many works also acknowledge that the impact of variability can be a significant concern [65].

Further, as outlined in the memristive reservoir computing study described in Appendix A, it has been observed that these devices suffer from significant variability issues. While the device variability does not significantly impede the behavior of the SRNN reservoir, it can severely impact system performance if these devices are used in the output layer of the reservoir (see Appendix A). This is because the performance of a Reservoir Computing system hinges significantly on the efficiency and reliability of its output layer [93].

In RC systems, while the reservoir serves as a dynamic memory element that processes input data, it's the output layer that translates the rich dynamical states of the reservoir into desired output signals. Since the output readout layer plays a crucial role in mapping the reservoir dynamics to the desired output, any inefficiencies or instabilities in this layer can directly impact the overall performance of the RC system. Thus, in the context of hardware RC implementations, emerging devices employed in the output layer, raise significant issues of variability in device characteristics that introduce uncertainties and errors into the output mapping process. Therefore, in order to achieve robust and reliable performance in hardware-based RC systems, careful attention must be paid to the resource requirement for implementation of the output readout layer and the training algorithm module.

The proposed architecture for the neuromorphic system thus utilizes Reservoir Computing based training approach while aiming to avoid the use of memristors in its output layer. Further, we notice that traditionally, output spikes from all neurons in the reservoir are utilized for training the output layer. This leads to increased computational complexity due to the greater number of variables involved and increased resources related to learning hardware implementation and the multiply-accumulate circuits for output layer. Here, we utilize the concept of dimensionality reduction to reduce the number of outputs from the reservoir such that the system performance is not impacted adversely. To show this, we also illustrate that attaining optimal task accuracy doesn't always demand the utilization of all reservoir neurons, and remarkably, in some cases, even when employing less than half of the

total reservoir neurons to train the output layer, the mean squared error (MSE) remains comparable to scenarios where all neurons are utilized. This finding underscores the potential for significant resource savings in hardware implementations of Reservoir Computing system by selectively leveraging a subset of reservoir neurons for training purposes. By shedding light on the effectiveness of utilizing a fraction of reservoir neurons, proposed architecture offers insights that can inform the design of more resource-efficient hardware architectures with enhanced performance and scalability in neuromorphic computing systems.

The Network-on-Chip (NoC) architecture defined in Chapter 4 integrates reconfigurability and enables the dynamic adjustment of output neuron connectivity when needed. This functionality ensures that only relevant neurons contribute to output computations, enhancing the efficiency and accuracy of the system. Remarkably, the overall resource overhead remains minimal due to the output layer implementation, which utilizes AND operations instead of multipliers for reduced complexity, achieves significant area savings.

Overall, the proposed architecture represents a sophisticated yet practical approach to resource optimization in neuromorphic computing systems. By judiciously managing the allocation of input and output neurons and leveraging the flexibility of the circuit-switched NoC, along with omitting the multiplier circuits in the output layer, the architecture offers a scalable and efficient solution for a wide range of applications, from time-series prediction to pattern recognition and beyond. Overcoming these challenges, the proposed architecture could unlock the full potential of neuromorphic computing, facilitating the development of intelligent systems capable of real-time processing, adaptive learning, and efficient operation across diverse application domains.

The efficiency of the proposed architecture is assessed through its application to the nonlinear wireless channel equalization task, a widely recognized problem with significant practical implications [48]. This challenge is pervasive across various communication systems and is characterized by its complexity and relevance in real-world scenarios. The proposed architecture offers a low-power and low-area online learning hardware solution to address this problem, thereby demonstrating its potential for enhancing the performance and scalability of wireless communication systems.

6.2 Architecture for Efficient Reservoir Computing using Circuit-Switched NoC

The proposed neuromorphic architecture is based on Reservoir Computing (RC) approach. The system is composed of a Spiking Recurrent Neural Network (SRNN) serving as a reservoir followed by a trainable readout layer. This architecture offers an innovative solution to minimize resource requirement on hardware by employing a circuit-switched Network-on-Chip (NoC) with dimensionality reduction. Here, dimensionality reduction means utilizing less than total neurons in the reservoir as output neurons. The circuit-switched NoC is used as a wrapper around the SRNN neurons thus providing a tool to optimize the selection of both input and output neurons within the reservoir. This is demonstrated in Fig. 6.2. By strategically limiting the number of neurons dedicated to input and output functions, the architecture effectively conserves resources while maintaining performance integrity. The resource savings are a result of two contributing reasons. First reason is the area and power resource savings related to spike routing by using this proposed NoC as reported in Chapter 4. Second reason is the reduction in the size of output layer and training algorithm implementation hardware due to the introduced possibility of utilizing less than total number of neurons in the SRNN reservoir as output neurons. Thus, the circuit-switched NoC, SpiCS-Net, plays a pivotal role in the proposed architecture by providing the necessary infrastructure for dynamic neuron connectivity and activity-based reconfiguration.

The block diagram demonstrating the proposed SRNN reservoir architecture utilizing the Circuit switched SpiCS-Net NoC is shown in Fig. 6.2. As discussed in the Section 6.1, by limiting the input and output neurons to less than total number of neurons, we save area and power while not affecting the training task efficiency adversely. In Fig. 6.2, SRNN reservoir consists of a "Neuron Cluster/Array" that utilizes the Circuit-switched NoC (SpiCS-Net) to route all the spikes in the system. This includes input spikes from the external input ports, internal recurrent connections to other neurons in the "Neuron Cluster/Array", and output spikes to the output readout layer.

The reservoir architecture is initialized with random configuration and parameters. As portrayed in Fig. 6.2, the neurons depicted individually represent neuron core modules, each

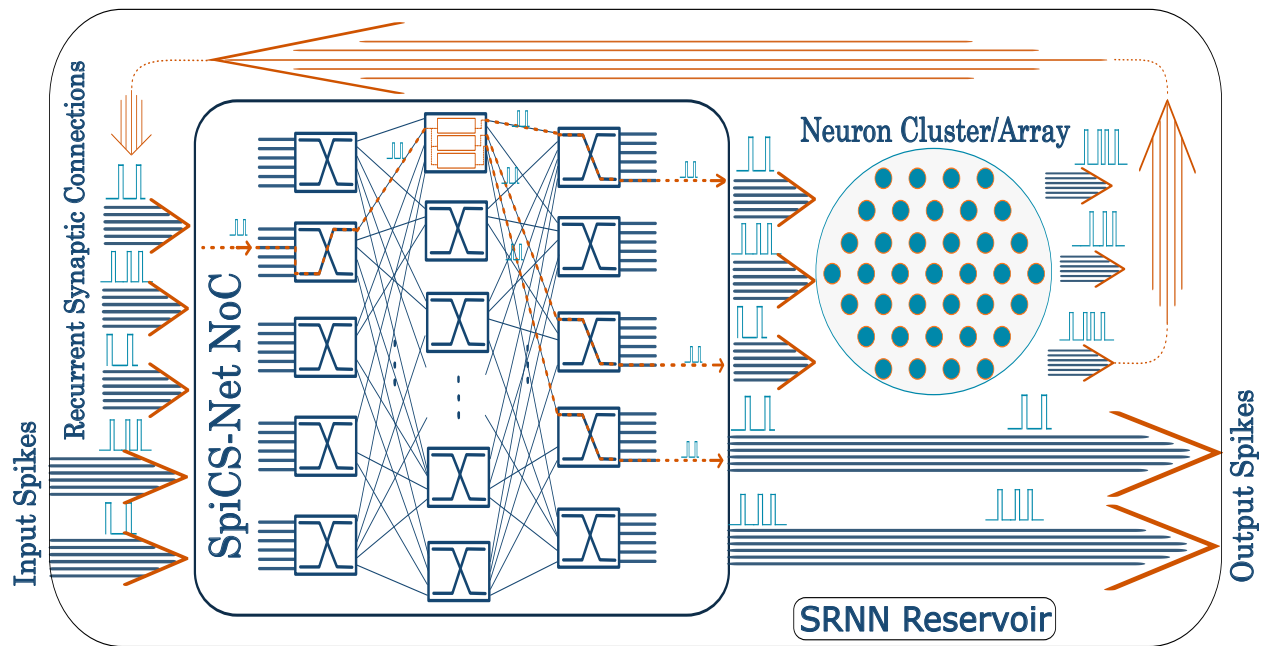


Figure 6.2: Reservoir block diagram of Neuromorphic Reservoir Computing architecture showing array of neuron core in the cluster connected via the circuit switched Noc, SpiCS-Net. The interface input/output signals of the Reservoir SRNN are also routed via SpiCS-Net. This allows for reconfigurability and control over which neurons are selected as input/output neurons, even at run time.

comprising a singular neuron with multiple synaptic connections at the input, elucidated in Fig. 6.3. These modules are integral components of the reservoir’s structure. The neuron parameters encompass threshold voltage and programmable leak, contributing to the dynamic behavior of the neurons within the reservoir. Moreover, the reservoir incorporates memristive synapses, characterized by associated resistance values. These values play a pivotal role in shaping the synaptic strength and plasticity of the connections within the reservoir network. Importantly, the initialization of memristive synapses is facilitated by the memristor control circuit depicted in Chapter 2 Fig. 2.5. Through randomized configuration of these parameters and synaptic properties, the reservoir architecture orchestrates a dynamic computational substrate capable of facilitating efficient nonlinear information processing tasks.

Similar to the neuron core parameters, the connectivity of the SRNN reservoir is also randomized. The configuration of SpiCS-Net, which defines SRNN connectivity, is regulated by the enable bits of the individual switches within the smaller crossbars(Chapter 4 Fig. 4.3). This connectivity information from the external ports is directed to a connectivity configuration block, as depicted in Fig. 6.4. Within this block, digital logic bits are generated to define the configuration of SpiCS-Net. Consequently, SpiCS-Net is configured to a randomized network organization through the utilization of the connectivity configuration block. This randomized connectivity scheme plays a crucial role in the dynamical behavior and information processing capabilities of the SRNN, contributing to its effectiveness in handling nonlinear computational tasks. Despite the seemingly arbitrary method of randomizing the SRNN, it is intriguing to observe that these randomly initialized reservoirs exhibit remarkable efficiency in addressing nonlinear data processing tasks [16].

The output layer of the reservoir implements a novel approach that eliminates the need for traditional multipliers, opting instead for digital operations within the reservoir’s output spikes. This innovative design significantly reduces computational complexity and hardware requirements. Furthermore, the output layer employs an Least Mean Squares (LMS)-based gradient descent algorithm for training the output layer weights. This technique enables efficient weight adjustments based on the discrepancy between the desired output and the actual output generated by the reservoir. The reservoir’s output consists of digital spike outputs from multiple neurons in the SRNN, providing a robust and reliable representation

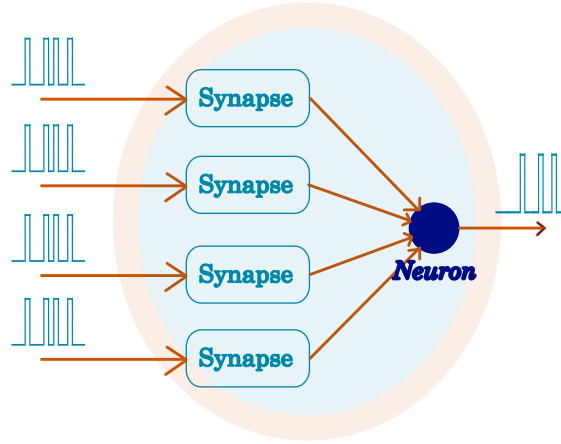


Figure 6.3: A neuron core in the Neuron Cluster shown in Fig. 6.2 composing the SRNN reservoir. One neuron core consists of multiple synaptic inputs.

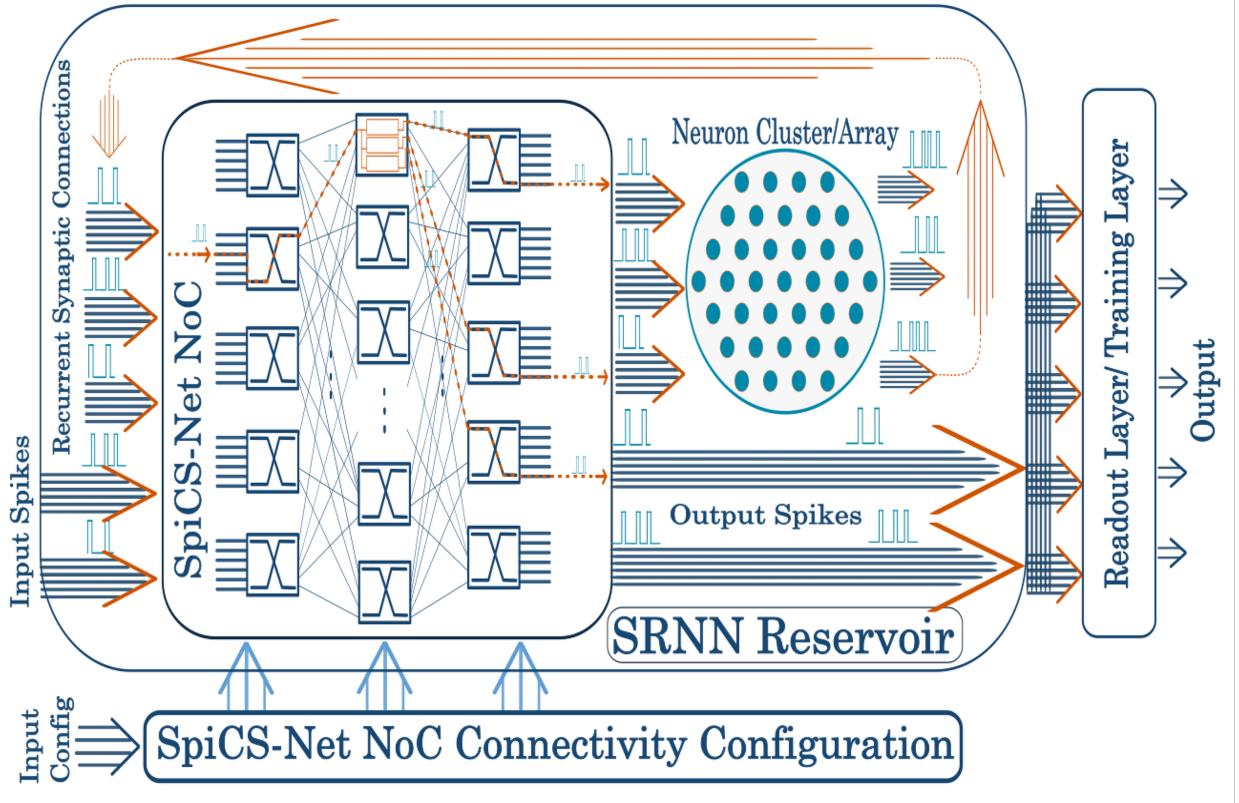


Figure 6.4: The proposed neuromorphic architecture adopts SpiCS-Net circuit-switched Network-on-Chip (NoC) and implements resource-saving measures by restricting the number of output neurons within the reservoir while allowing any neuron within the system to serve as an output neuron. Leveraging the reconfigurability feature inherent in the NoC, the architecture can dynamically adjust the connectivity of output neurons.

of the input in the form of network’s activity. This integrated approach enhances the performance and scalability of the reservoir computing system, particularly in tasks such as nonlinear data processing and wireless channel equalization.

6.2.1 Application: Nonlinear Channel Equalization

The Nonlinear Channel Equalization task is a crucial problem in communication systems where the transmitted signal undergoes distortion due to the channel’s nonlinear characteristics. The objective of this task is to restore the original transmitted signal by mitigating the distortion introduced during transmission. Such distortions encompass various factors such as signal attenuation, noise, crosstalk, and inter-symbol interference, among others, all of which have the potential to degrade the quality of the transmitted signal. Nonlinear equalization algorithms play a critical role in addressing these channel effects, thereby enhancing the accuracy of data transmission and reception in wireless systems. By effectively compensating for these distortions, nonlinear equalization algorithms contribute to improving the reliability and efficiency of wireless communication networks.

Reservoir Computing (RC) offers a promising approach to address this challenge by leveraging the nonlinear dynamics of the Spiking Recurrent Neural Network(SRNN). Utilizing the nonlinear dynamics of the SRNN, or the reservoir, a RC system models and effectively captures the complex temporal dependencies present in the distorted input signal. The reservoir acts as a dynamic fading memory, representing information from past inputs in the evolving reservoir states, or the reservoir output. These outputs are used to train a simple linear readout layer to efficiently model the nonlinear noise components in the distorted signal and reconstruct the original transmitted signal. The Nonlinear Channel Equalization Task serves as a benchmark for evaluating the performance of RC systems in handling nonlinear distortions in wireless communication channels, highlighting their efficacy in real-world signal processing applications.

Indeed, the Nonlinear Channel Equalization task represents a practically relevant and valuable application for Reservoir Computing (RC) systems. RC systems excel in handling nonlinear temporal data processing tasks, making them well-suited for addressing the complexities inherent in equalizing distorted signals within communication systems [5, 48,

6, 101]. The task involves transmitting a sequence of 2-bit symbols (from the set $\{-3, -1, 1, 3\}$) through a wireless channel. At the receiver, this transmitted sequence $d(n)$ is received as a corrupted version $q(n)$ due to various factors, including noise, multi-path propagation (causing inter-symbol interference), and nonlinear distortion from power amplifiers. The objective is to recover the original transmitted symbols $d(n)$ from the received, corrupted sequence, $q(n)$. The channel is modeled by a tapped delay line linear system with length of 10. This task simulates real-world scenarios in wireless communication, where signal degradation occurs during transmission and needs to be corrected for reliable data recovery.

$$q(n) = 0.08d(n+2) - 0.12d(n+1) + d(n) + 0.18d(n-1) - 0.1d(n-2) + 0.091d(n-3) - 0.05d(n-4) + 0.04d(n-5) + 0.03d(n-6) + 0.01d(n-7) \quad (6.1)$$

$$r(n) = q(n) + 0.036q^2(n) - 0.011q^3(n) \quad (6.2)$$

The Reservoir Computing based Equalizer system block diagram for this task is depicted in Fig. 6.5. In this system, the reservoir serves to model the nonlinear noise present in the input data. The input signal comprises the analog signal received at the Analog Front End (AFE) of the wireless receiver. This signal is conveyed to the reservoir neurons, which integrate the analog signal at their inputs and generate digital spikes when the accumulated voltage surpasses the neuron threshold. The recurrent connectivity within the reservoir induces dynamic system behavior, effectively capturing the noise patterns inherent in the data. The output from the reservoir is represented as a spike map, where each neuron output remains high for a spike event or stays low in the absence of a spike event during that clock period. The output layer weights, resembling tap coefficients in a traditional equalizer circuit, undergo operations with the spike map to produce a multi-bit digital output signal. This digital output corresponds to the nonlinear noise embedded within the input data. Subsequently, this multi-bit output undergoes conversion via a Digital-to-Analog Converter (DAC) to obtain the analog equivalent of the noise. This analog noise signal is subsequently

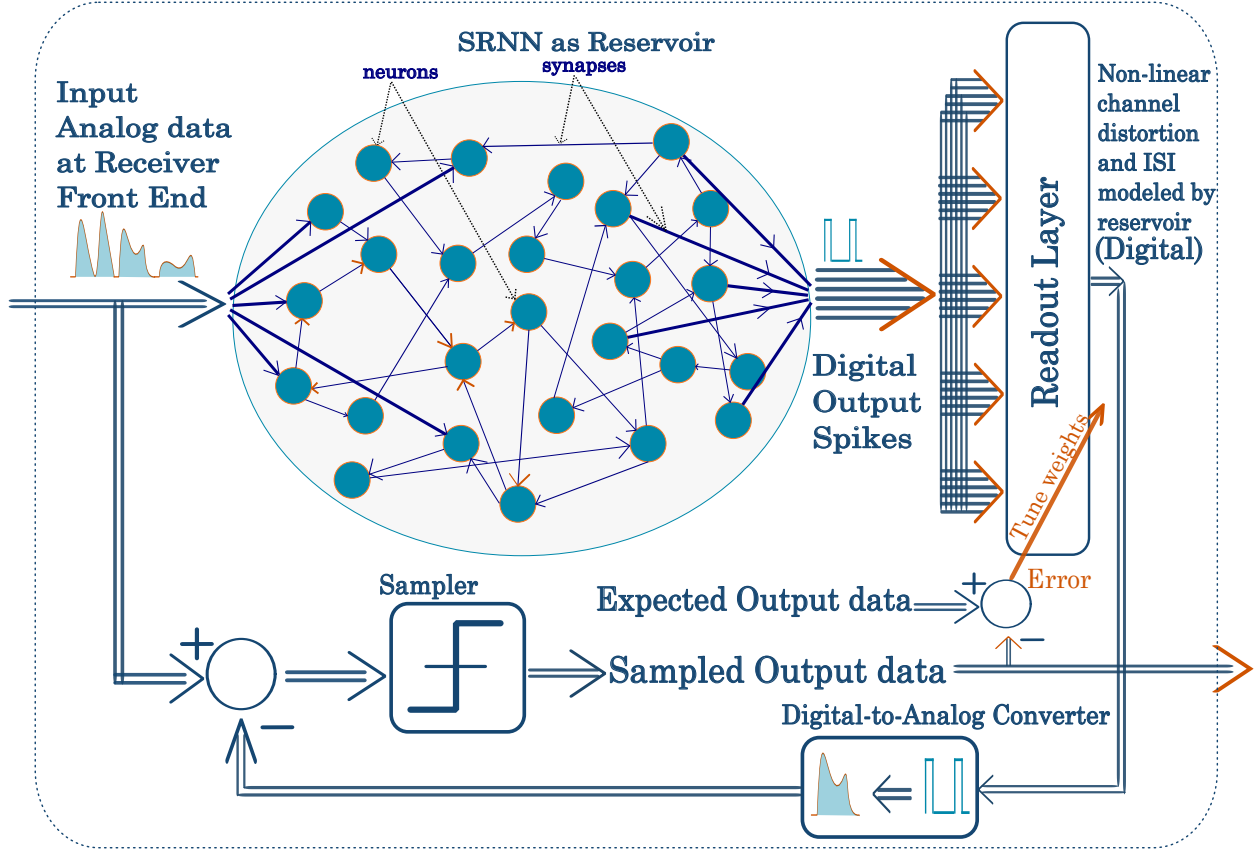


Figure 6.5: Reservoir Computing based Nonlinear Equalizer system utilizes SRNN reservoir to model nonlinear noise in input data. Analog signals from the Analog Front End are processed by reservoir neurons, generating spiking reservoir states, or spike maps, that capture nonlinear noise patterns. Spike maps operate on output weights to produce a multi-bit digital signal, which is then converted to analog for subtraction of noise from input, enabling the equalization and reconstruction of received distorted input data.

subtracted from the received signal, facilitating the reconstruction of the desired undistorted signal from the received data.

6.3 Implementation and Verification

MATLAB Characterization

Initially, a dimensionality reduction analysis is conducted at the system level to quantify the extent of reduction that maintains system performance without adverse effects. This analysis, carried out using MATLAB, focuses on two specific tasks: Mackey-Glass prediction and Nonlinear Channel Equalization. The specifics of the Mackey-Glass prediction task are elaborated in Appendix A. Subsequently, the comprehensive system design is implemented entirely in MATLAB, facilitating both the characterization and training phases of the output layer. In the Mackey-Glass prediction task, the MATLAB model is meticulously fine-tuned and optimized to accurately predict the behavior of chaotic time-series data. This entails thorough training of the output layer to forecast future values based on historical data. Similarly, in the context of the Nonlinear Channel Equalization task, MATLAB provides the requisite framework for training and evaluating the system's performance in mitigating distortions and noise within communication channels. The model undergoes extensive training to effectively equalize the channel response and enhance the reliability of data transmission, a critical requirement in communication systems.

6.3.1 Mixed-Signal Implementation

Following the MATLAB characterization, the RC-based equalizer, which encompasses SpiCS-Net NoC alongside the output layer and LMS training algorithm, undergoes meticulous design and implementation in Verilog Hardware Description Language(HDL). This process involves the translation of the system's architecture and functionality, established in MATLAB, into the Verilog hardware description language (HDL). Through this translation, the intricate mechanisms and algorithms developed in MATLAB are encapsulated into a hardware-oriented design, enabling the realization of the RC-based equalizer within a hardware environment.

Upon completion of the Verilog design, extensive mixed-signal simulations are conducted using a Verilog-A model of the neuron core. This Verilog-A model closely mirrors the interface and behavior of the neuron core schematic, capturing its functionality through mathematical model equations that incorporate integrators and comparators. The mixed-signal verification process entails validating the functionality and performance of the design across both digital and analog domains. To achieve this, Cadence tools such as NC-Verilog and Spectre are employed for comprehensive simulation and verification. These tools ensure the correctness and reliability of the hardware implementation by rigorously analyzing the system's behavior under various operating conditions and stimuli. Through the verification process, any potential issues or discrepancies in the design are identified and rectified, ensuring the integrity and robustness of the final hardware implementation. During the simulation process, diverse test cases and scenarios are executed to evaluate the system's behavior across different conditions and inputs. This rigorous verification process aims to uncover and address any potential issues or inconsistencies in the design, thereby guaranteeing that the final hardware implementation aligns with the specified requirements and performance criteria.

The simulation results from the Output Layer module in Fig. 6.6 shows the training process of the evolution of weight coefficients over the training period. These weight coefficients play a critical role in determining the influence of input spikes on the output neuron responses, thereby shaping the network's overall predictive capabilities. During the training phase, the weight coefficients are adjusted iteratively based on the error between the predicted and actual outputs. The simulation depicts how these weights are dynamically updated to minimize the prediction error and enhance the accuracy of the model. As the training progresses, the weight coefficients converge towards optimal values that enable the network to accurately predict the desired output patterns. Furthermore, the simulation results provide insight into the convergence behavior of the weight coefficients, ensuring that the network achieves satisfactory performance levels within a reasonable training period.

Fig. 6.7 shows the Verilog-A simulation results for the behavior of the reservoir module with the neuron core. This simulation results show the process of integrating input potentials over time and the subsequent generation of spike outputs by the neuron core. The membrane potential of the neuron core is visualized over a simulated time period, highlighting its



Figure 6.6: The simulation results from the Output Layer module offer valuable insights into the training dynamics of the neuromorphic architecture, with a specific focus on the evolution of weight coefficients throughout the training period.

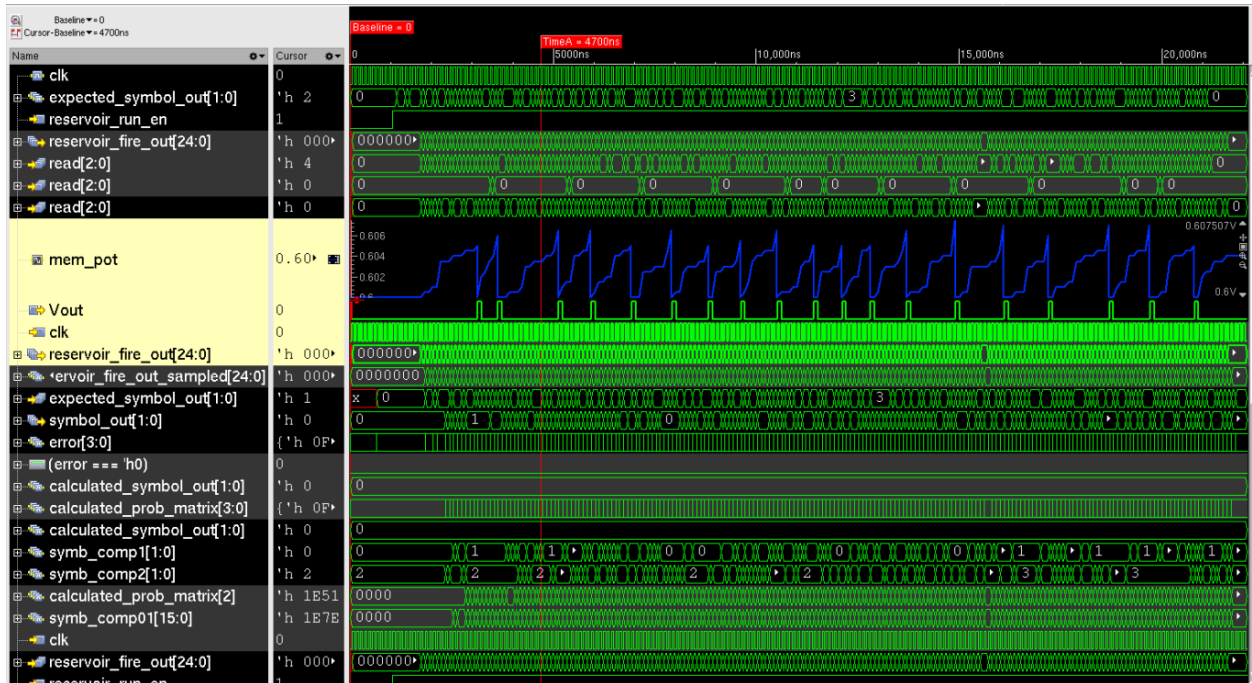


Figure 6.7: Simulation result from the Verilog-A neuron core reservoir module showing membrane potential of a neuron core as it integrates input potential and generates spike outputs.

fluctuations in response to incoming inputs. Initially, the membrane potential remains at a resting state level. As input potentials are applied, the membrane potential gradually increases, reflecting the integration of these inputs by the neuron core. Upon reaching a certain threshold level, indicative of the neuron's excitability, the membrane potential abruptly spikes, representing the generation of an output spike. This spike event signifies the neuron's response to the integrated inputs, conveying information encoded in the form of spikes. The simulation results provide valuable insights into the temporal dynamics of the neuron core's operation, illustrating how it processes incoming signals and generates output spikes based on predefined thresholds. The results also show the `reservoir_fire_out` signal which collectively represent spikes from all neurons in the reservoir.

After the Verilog design is completed and thoroughly simulated, the next step is synthesis to target specific hardware technology. In this case, the design is synthesized to IBM CMOS10LPE 65 nm technology using Synopsis Design Compiler. This process involves mapping the Verilog code to the target technology library, optimizing the design for area, timing, and power considerations. Once synthesis is completed, the design undergoes placement and routing using Cadence Innovus. Placement involves determining the physical locations of the various logic elements on the chip, while routing involves establishing the interconnections between these elements. The goal of this process is to minimize signal delays, optimize power consumption, and ensure proper functionality of the design. During placement and routing, area estimates for the different modules of the design are obtained. This provides valuable insights into the chip area required for implementing the neuromorphic architecture. Additionally, timing constraints are taken into account to ensure that the design meets the required performance specifications.

Overall, the combination of MATLAB characterization and Verilog hardware design, coupled with comprehensive simulation and verification using industry-standard tools, ensures the robustness and reliability of the neuromorphic architecture. Further, synthesis and placement/routing are carried out as critical steps in the hardware implementation process, enabling the translation of the Verilog design into a physical layout that can be fabricated using the target semiconductor technology.

6.4 Results

MATLAB Characterization

The characterization in MATLAB explores the Normalized Mean Squared Error (NMSE) for the nonlinear data processing tasks within a reservoir computing system, specifically investigating the impact of varying the number of output neurons while maintaining a constant size for the Spiking Recurrent Neural Network (SRNN). As depicted in the graph Fig. 6.8, there is an intriguing trend observed for the NMSE of Mackey-Glass prediction task: reducing the number of output neurons in the reservoir to up to 60% exhibits similar performance, with no significant impact on NMSE. Notably, this ratio can be as low as 40% for larger reservoir sizes, such as those with 768 and 1024 neurons. It's worth noting that the y-axis is presented in a logarithmic scale to effectively demonstrate the trend where NMSE remains low across different configurations. This observation underlines the potential efficacy of leveraging SpiCS-Net Circuit-Switched NoC for routing output spikes out of the reservoir. Such an approach enables the reduction of output neurons while retaining the capability to select any neuron within the reservoir as an output neuron, thus offering a streamlined and resource-efficient solution for reservoir computing tasks.

Further, The graph in Fig. 6.9 illustrates the Symbol Error Rate (SER) for the Channel Equalization task, focusing on how the SER changes with varying numbers of output neurons. Unlike the previous analysis, the y-axis in this graph is not presented in a logarithmic scale to provide a clearer visualization of the overall trend. Notably, it can be observed that reducing the number of output neurons from 100% (1) to as low as 40% (0.4) does not significantly impact the performance of the system. Even with a substantial reduction in the number of output neurons, the SER remains relatively stable, indicating robustness in the system's ability to handle the channel equalization task effectively. This observation suggests that optimizing the number of output neurons can potentially lead to resource savings without compromising system performance, thereby offering valuable insights for designing efficient reservoir computing systems for channel equalization applications.

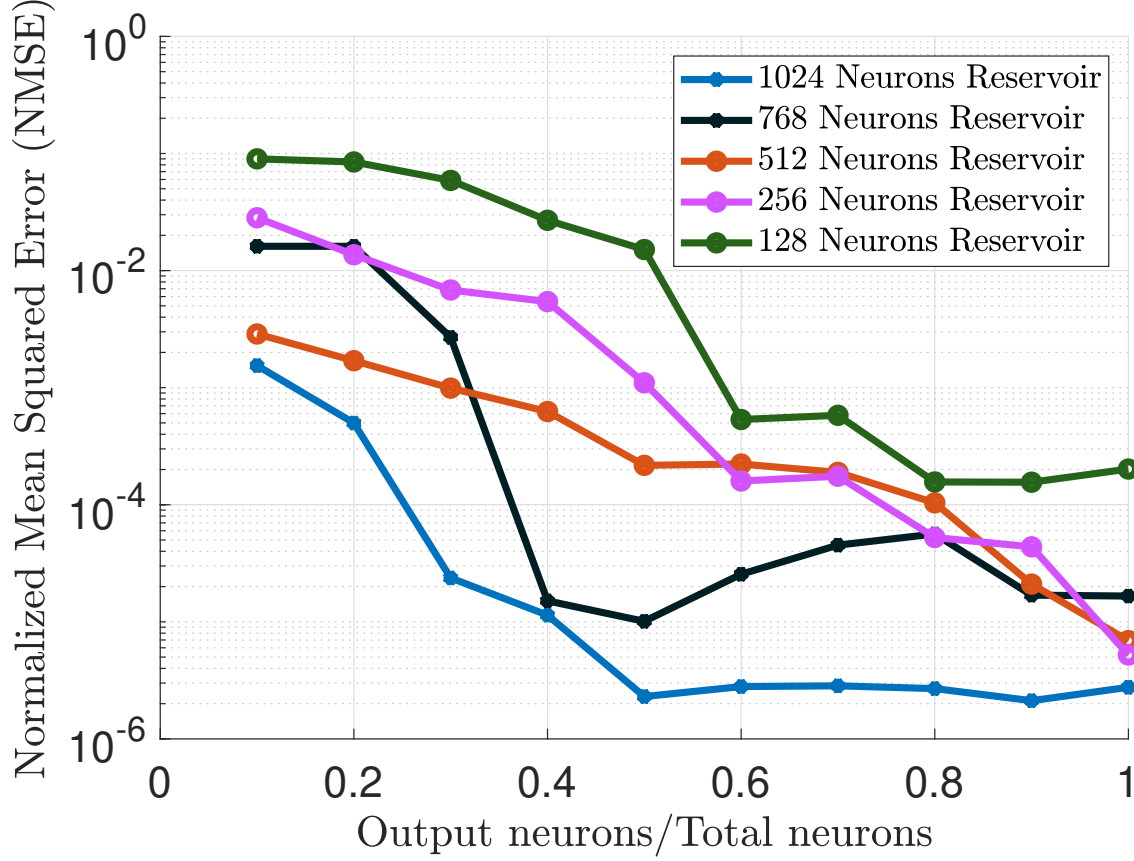


Figure 6.8: Normalized Mean Squared Error for the Mackey Glass prediction task from the reservoir computing system as the number of output neurons changes while keeping SRNN size constant. The graph shows that reducing the number of output neurons in the reservoir to upto 60% leads to similar performance and NMSE does not have a significant impact. This ratio can go as low as 40% for large reservoir sizes with 768 and 1024 neurons. Notice that y-axis is in log scale to show the trend where NMSE is low. This observation supports usage of SpiCS-Net Circuit-Switched NoC for routing output spikes out of the reservoir such that number of output neurons are reduced while preserving the capability of selecting any neuron within the reservoir as an output neuron.

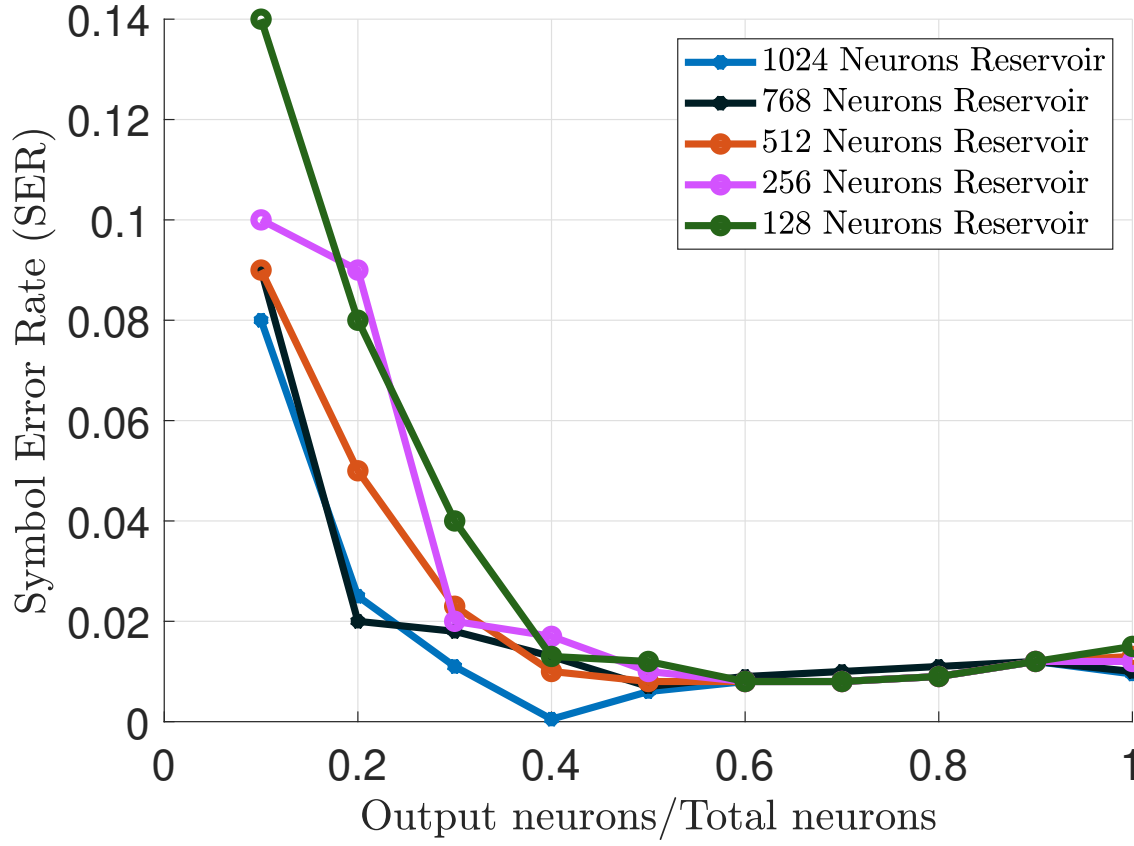


Figure 6.9: Similar to previous graph in Fig. 6.8 this graph shows the Symbol Error Rate (SER) for the Channel Equalization task as the number of output neurons are varied. Here the y-axis is not log scale to show overall trend more clearly. It can be observed that reduction of the output neurons from 100% (1) to upto 40% (0.4) does not adversely impact the performance of the system.

6.4.1 Mixed-Signal Implementation

Table 6.1 shows the post-layout simulation and layout data for the output layer of the proposed architecture. The area estimate for the layout is determined from the synthesized, placed and routed design in IBM CMOS10LPE 65nm technology. The layout generated in Cadence Innovus is imported to Cadence Specter to run post-layout simulations for getting the static and dynamic power consumption per clock cycle. The simulation utilizes a clock with frequency 10 MHz. The input, output and internal signals from mixed signal simulations are shown in waveforms Fig. 6.6 and Fig. 6.7. The power and area savings due to usage of circuit-switched SpiCS-Net are discussed in Chapter 4. In this section we further discuss resource savings due to dimensional reduction for saving resource consumption of the output layer and training algorithm circuitry.

To evaluate the effectiveness of the proposed Reservoir Computing (RC) system with a digital output layer against a multiplier-based design of the same size, several performance metrics can be considered. These metrics may include computational complexity, power consumption, and area utilization. By comparing these metrics between the implemented design with a digital output layer and the multiplier-based approach of the same size, insights can be gained into the relative advantages and disadvantages of each approach in the context of Reservoir Computing for Nonlinear Channel Equalization.

Fig. 6.10 shows that the proposed digital output layer design consumes less power compared to the multiplier-based approach due to the absence of power-intensive multiplication operations. Similarly, Fig. 6.11 compares the area utilization of the proposed digital output layer design with that of the multiplier-based approach, showing higher efficiency of the proposed architecture output layer.

Further, reducing the dimensions of the output layer by 40% leads to substantial area savings for the RC system. By shrinking the size of the output layer by reducing the number of output neurons, the overall area occupied by this component decreases significantly. As a result, fewer hardware resources are required to implement the output layer functionality, leading to more efficient area utilization. this is shown in Fig. 6.13.

Table 6.1: Area of the Output Readout Layer as number of neuron inputs to the Readout Layer are varied.
Weight coefficient bits = 16

No. of neurons	Area (sq. μm)	Static Power Consumption (pJ)	Power Consumption per cycle (nW)
32	1132.98	45.9	0.3498
64	1608.3	79.23	0.80988
128	2278.2	134.8	1.8627
256	3237.65	217.87	4.371
512	4393.9	347.39	10.066
768	5631.28	476.2	23.25
1024	6811.87	629.86	54.87

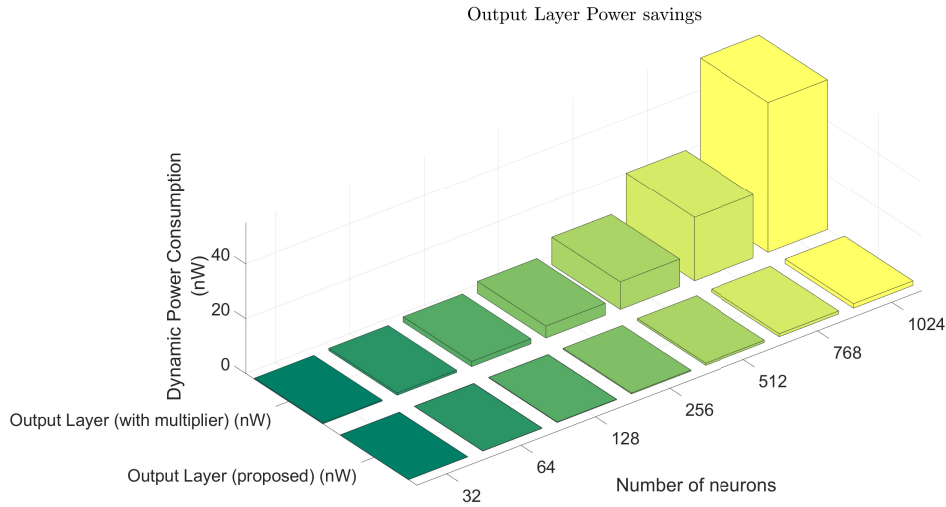


Figure 6.10: To evaluate the effectiveness of the proposed Reservoir Computing (RC) system with a digital output layer against a multiplier-based design of the same size, power consumption is a crucial factor. The bar graph shows that proposed digital output layer design consumes less power compared to the multiplier-based approach due to the absence of power-intensive multiplication operations. These savings are shown for multiple reservoir sizes.

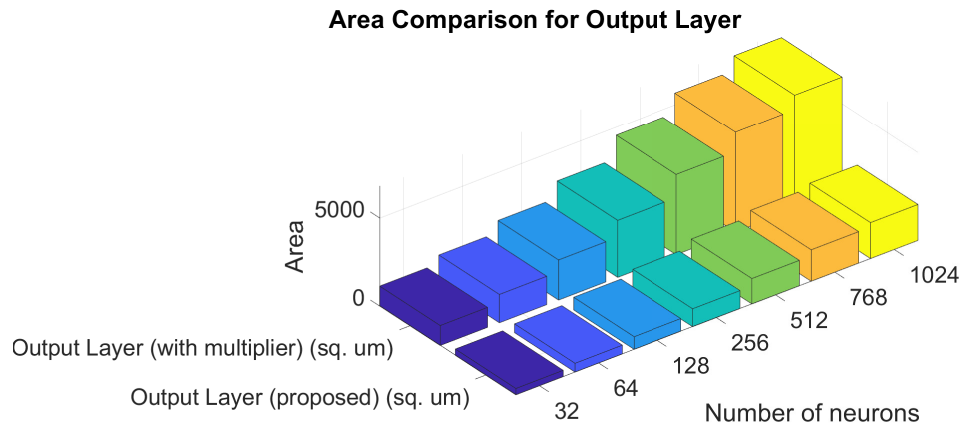


Figure 6.11: Bar graph comparing area utilization of the output layer efficiency in terms of resource utilization between different designs. The effectiveness of the proposed Reservoir Computing (RC) system with a digital output layer against a multiplier-based design of the same size is demonstrated by the presented results.

Moreover, these area savings are often accompanied by reductions in power consumption as depicted in the bar plot in Fig. 6.12. With a smaller output layer, the associated circuitry consumes less power during operation, contributing to overall energy efficiency. This dual benefit of area and power savings underscores the advantages of dimensionality reduction technique in optimizing the hardware implementation of RC systems.

In practical terms, these area and power results demonstrate the potential for cost savings and improved scalability of RC systems. By minimizing resource requirements without compromising performance, dimensionality reduction technique implemented by using Spi-CS-Net enhances the feasibility of deploying RC systems in various applications, particularly those with stringent constraints on area and power consumption.

6.5 Discussion

Our findings highlight the significant resilience of the reservoir system’s performance, demonstrating that even with just half the number of neuron outputs directed to the output layer instead of utilizing all neurons within the reservoir, the impact on the Normalized Mean Squared Error (NMSE) remains relatively minimal. This resilience is particularly pronounced in larger reservoirs, where the effect becomes increasingly consistent and even leads to improved NMSE performance in some cases. By leveraging the Network-on-Chip (NoC) to selectively designate output neurons, our proposed approach achieves not only enhanced performance across varying reservoir sizes but also substantial savings in hardware resources, area, and power consumption. This strategic utilization of the NoC underscores its pivotal role in optimizing resource allocation while upholding superior performance standards in neuromorphic computing applications. Ultimately, our research underscores the robustness and effectiveness of this approach in addressing the challenges of resource-constrained hardware implementations, paving the way for more efficient and scalable neuromorphic computing systems.

A suggested addition to the architecture is the incorporation of an activity detector, which serves to identify and categorize neurons based on their spiking behavior. This functionality enables the detection and filtering of neurons that consistently spike (always spiking) or

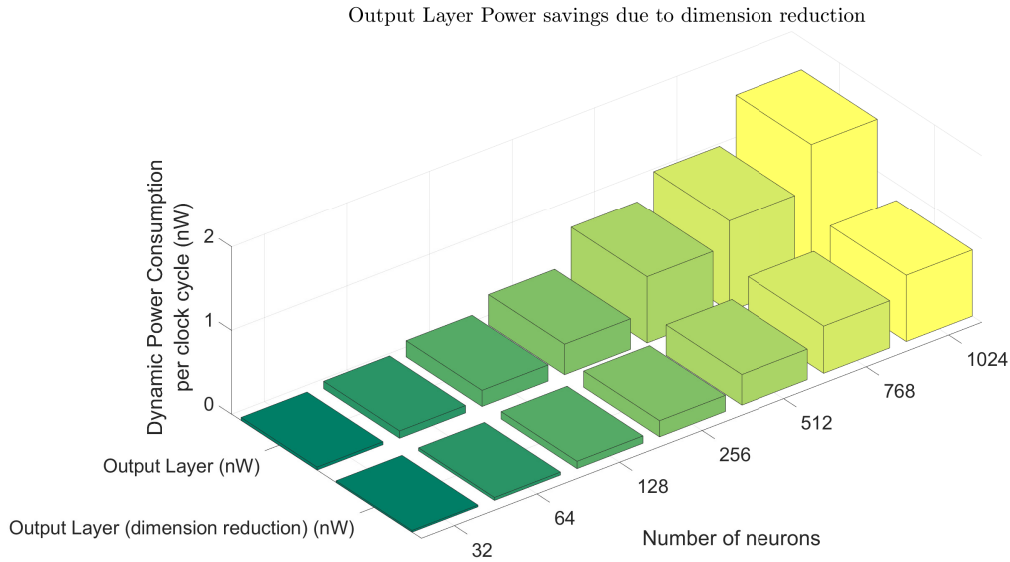


Figure 6.12: Bar graph illustrating power savings achieved through dimensionality reduction in the output layer of the Reservoir Computing system. The graph highlights the significant reduction in power consumption resulting from a 40% decrease in output layer dimensions, demonstrating the efficiency gains afforded by dimensionality reduction techniques.

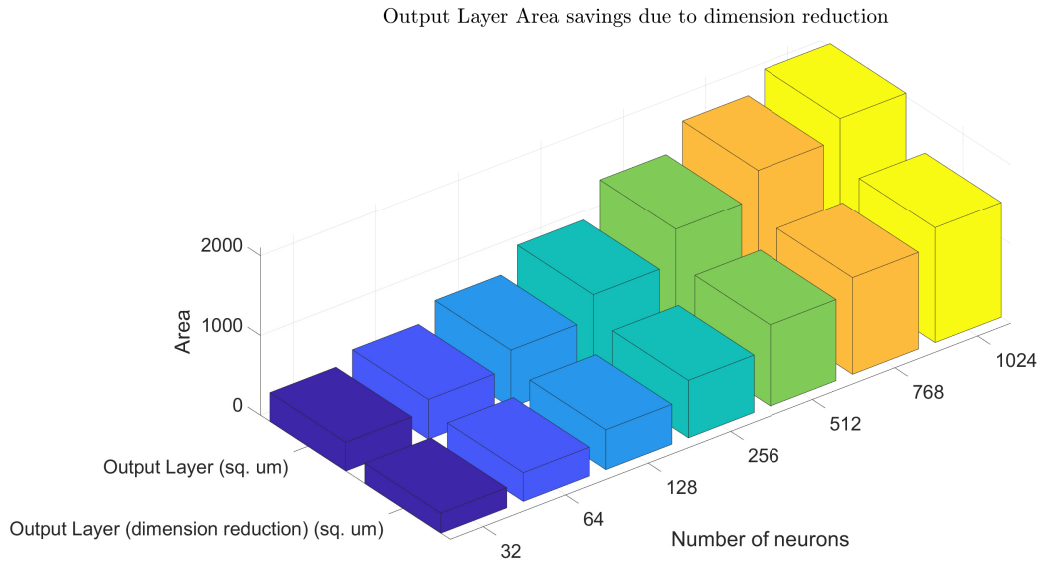


Figure 6.13: Bar graph depicting the area savings achieved through dimensionality reduction in the output layer of the Reservoir Computing system. The graph illustrates the substantial decrease in area utilization resulting from a 40% reduction in output layer dimensions, showcasing the efficiency gains achieved by employing dimensionality reduction techniques.

Table 6.2: Area of the Output Readout Layer as number of neuron inputs to the Readout Layer are varied.
Weight coefficient bits = 16

Reservoir Size	Area (output neurons = reservoir size) (sq. μm)	Area (output neurons = $0.6 \times$ reservoir size) (sq. μm)
25	1008.36	745.2
32	1132.98	893.3
64	1608.3	1215.02
128	2298.2	1781.3
256	3237.65	2496.2
512	4393.9	3399.07
768	5631.28	4205.28
1024	6811.87	4997.21

Table 6.3: Performance of the proposed Reservoir Computing (RC) system applied to Nonlinear Channel Equalization compared with previously reported RC systems

	Jaeger 2004 [48]	Antonik 2017 [5]	Skibinsky 2018 [101]	* Proposed RC system leveraging SpiCS-Net
Technology	Software Echo State Network	Photonic Reservoir	FPGA-based Cyclic Reservoir Computer	CMOS-Memristor 65nm
Size	46 neurons	-	27 neurons	32 Neurons
Symbol Error Rate(SER)	0.0001	0.0013 +- 0.0009	0.007	0.0027
Application	Non-linear Chan Eq	Non-linear Chan Eq	Non-linear Chan Eq	Non-linear Chan Eq

remain inactive (never spiking), thereby facilitating efficient resource allocation within the reservoir.

Chapter 7

Conclusion

In conclusion, neuromorphic computing represents a promising frontier in efficient computing, emulating the structure and functionality of the human brain while minimizing resource usage and power consumption. The integration of neuromorphic systems within CMOS technology provides an alternative to traditional Von Neumann computing paradigms, offering unique advantages and challenges. The promise of neuromorphic computing lies in its emulation of the human brain's structure and function, which offers unprecedented efficiency in computation compared to traditional Von Neumann architectures. This efficiency stems from the brain's ability to perform complex tasks with minimal power consumption, a feat that traditional computing struggles to replicate.

However, transitioning neuromorphic concepts into practical integrated circuit implementations presents several challenges. Among these challenges, one crucial aspect is the analog implementation of synaptic weight multiplication and tuning. Mimicking the brain's synapses in hardware requires precise control over analog parameters while minimizing energy consumption and physical footprint. The integration of memristors, nanoscale devices exhibiting synaptic behavior, offers promising opportunities for efficient computation and learning. By accurately modeling memristor behavior in circuit simulations, we pave the way for their seamless integration into neuromorphic hardware designs.

Another significant hurdle is designing network-on-chip architectures tailored to the unique requirements of neuromorphic systems. These architectures must offer full connectivity while

minimizing resource overhead and accommodating dynamic reconfiguration. Achieving these objectives demands novel approaches to routing, communication protocols, and network topology design, ensuring efficient spike communication and information flow within the system. Our proposed SpiCS-Net architecture revolutionizes spike communication in neuromorphic systems, enabling efficient routing and reducing area and power consumption. Leveraging SpiCS-Net as a robust routing mechanism for online learning tasks enhances the scalability and adaptability of neuromorphic architectures, facilitating complex computations with minimal resource utilization.

Moreover, enabling online learning directly on-chip introduces additional complexities, including real-time adaptation, synaptic plasticity, and training algorithms. Integrating these capabilities within the constraints of hardware imposes stringent requirements on computational efficiency, memory management, and algorithmic implementation.

In conclusion, this advances the field of neuromorphic computing by addressing critical challenges and proposing innovative solutions. By combining cutting-edge hardware design, novel circuit architectures, and advanced modeling techniques, we pave the way for the development of efficient, scalable, and adaptive neuromorphic systems with broad applications in artificial intelligence, robotics, and beyond.

7.1 Summary of Contributions

- Verilog-A SPICE compatible model for HfO_2 that offers Resistance tunability as the compliance current is varied. The model also includes Reset failure mechanism to improve reliability of neuromorphic circuit designs.
- Circuit-Switched Network-on-chip for Spiking data (SpiCS-Net) transfer across chip which is a novel utilization of well-established telecommunication network Clos topology.
- Mathematical basis for designing SpiCS-Net
- SpiCS-Net provides that ability to leverage sparsity of the Spiking Neural Network (SNN) to save resources, area and power.

- Hierarchical NoC to scale the Circuit-Switched NoC for large scale neuromorphic systems. This hierarchical NoC is a novel circuit-packed hybrid scheme which ensures spikes-as-spikes at the lowest level of hierarchy while using packetized AER communication at the higher levels for scalability and communication bandwidth.
- Usage of the proposed NoC in Reservoir Computing Systems where it will lead to resource efficiency due to offered reconfigurability and reduction in the size of output layer.
- Our research reveals that in reservoirs initialized with random weights, certain neurons may exhibit consistent spiking behavior (always spiking) or remain inactive throughout (never spiking). This issue can be addressed by implementing a simple activity detection circuit capable of identifying these distinct neuron categories. This activity detection together with the connectivity configuration block for the circuit-switched NoC can be used for implementing dynamic neuron connectivity, discarding any output neurons which have repetitive or absent spiking information.

Bibliography

- [1] Jonathan D Ambrose et al. “Grant: Ground-roaming autonomous neuromorphic targeter”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8 (cit. on p. 48).
- [2] Sherif Amer. “Device Modeling and Circuit Design of Neuromorphic Memory Structures”. In: (2019) (cit. on pp. 3, 30, 37).
- [3] Sherif Amer, Sagarvarma Sayyaparaju, et al. “A practical hafnium-oxide memristor model suitable for circuit design and simulation”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017 (cit. on pp. 27, 28, 35, 44, 45).
- [4] Hongyu An et al. “Robust deep reservoir computing through reliable memristor with improved heat dissipation capability”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.3 (2020), pp. 574–583 (cit. on p. 146).
- [5] Piotr Antonik, Marc Haelterman, and Serge Massar. “Online training for high-performance analogue readout layers in photonic reservoir computers”. In: *Cognitive Computation* 9.3 (2017), pp. 297–306 (cit. on pp. 22, 99, 100, 108, 123).
- [6] Piotr Antonik et al. “FPGA implementation of reservoir computing with online learning”. In: *24th Belgian-Dutch Conference on Machine Learning*. 2015 (cit. on pp. 99, 109).
- [7] Daniel Auge et al. “A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks”. In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710 (cit. on pp. 52, 66).
- [8] Simone Balatti et al. “Voltage-controlled cycling endurance of HfO_x-based resistive-switching memory”. In: *IEEE Transactions on Electron Devices* 62.10 (2015), pp. 3365–3372 (cit. on pp. 27, 33).
- [9] Danielle S Bassett and Edward T Bullmore. “Small-world Brain Networks Revisited”. In: *The Neuroscientist* 23.5 (2017), pp. 499–516 (cit. on p. 75).
- [10] Arindam Basu, Lei Deng, et al. “Spiking Neural Network Integrated Circuits: A Review of Trends and Future Directions”. In: *2022 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2022, pp. 1–8 (cit. on p. 74).

- [11] Abderazek Ben Abdallah and Khanh N Dang. “Toward robust cognitive 3d brain-inspired cross-paradigm system”. In: *Frontiers in Neuroscience* (2021), p. 795 (cit. on p. 21).
- [12] Dalibor Biolek, Zdenek Kolka, et al. “Memristor models for spice simulation of extremely large memristive networks”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016 (cit. on p. 44).
- [13] Zdeněk Biolek, Dalibor Biolek, and Viera Biolkova. “SPICE Model of Memristor with Nonlinear Dopant Drift.” In: *Radioengineering* (2009) (cit. on p. 27).
- [14] Kwabena A Boahen. “Point-to-point connectivity between neuromorphic chips using address events”. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5 (2000), pp. 416–434 (cit. on p. 19).
- [15] Thomas Bohnstingl et al. “Biologically-inspired training of spiking recurrent neural networks with neuromorphic hardware”. In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2022, pp. 218–221 (cit. on p. 22).
- [16] Erik Bollt. “On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.1 (2021) (cit. on p. 106).
- [17] TS Böske et al. *Int. Electron Devices Meeting*. 2011 (cit. on p. 27).
- [18] Maxence Bouvier et al. “Spiking neural networks hardware implementations and challenges: A survey”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.2 (2019), pp. 1–35 (cit. on p. 74).
- [19] Jens Bürger and Christof Teuscher. “Variation-tolerant computing with memristive reservoirs”. In: *International symposium on nanoscale architectures (NANOARCH 2013)*. IEEE/ACM. 2013, pp. 1–6 (cit. on p. 146).

- [20] Snaider Carrillo et al. “Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations”. In: *IEEE Transactions on Parallel and Distributed Systems* 24.12 (2012), pp. 2451–2461 (cit. on pp. 4, 19, 65, 67–70, 96).
- [21] Z. Chai et al. “RTN-based defect tracking technique: Experimentally probing the spatial and energy profile of the critical filament region and its correlation with HfO₂RRAM switching operation and failure mechanism”. In: *2016 IEEE Symposium on VLSI Technology*. 2016, pp. 1–2. DOI: 10.1109/VLSIT.2016.7573402 (cit. on p. 27).
- [22] Nishith N Chakraborty and Garrett S Rose. “Programmable Refractory Period Implementations in a Mixed-Signal Integrate-And-Fire Neuron”. In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2022, pp. 770–774 (cit. on pp. 13, 14).
- [23] Yang Yin Chen et al. “Understanding of the endurance failure in scaled HfO₂-based 1T1R RRAM through vacancy mobility degradation”. In: *2012 International Electron Devices Meeting*. 2012, pp. 20.3.1–20.3.4. DOI: 10.1109/IEDM.2012.6479079 (cit. on p. 27).
- [24] L. Chua. “Memristor-The missing circuit element”. In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519. DOI: 10.1109/TCT.1971.1083337 (cit. on p. 15).
- [25] Leon Ong Chua. “Everything You Wish to Know About Memristors but Are Afraid to Ask”. In: *Handbook of Memristor Networks(2015)*. 2015 (cit. on pp. 3, 15).
- [26] Charles Clos. “A study of non-blocking switching networks”. In: *Bell System Technical Journal* 32.2 (1953), pp. 406–424 (cit. on pp. 52, 54, 59, 81).
- [27] Erika Covi et al. “Analog memristive synapse in spiking networks implementing unsupervised learning”. In: *Frontiers in neuroscience* 10 (2016), p. 482 (cit. on p. 15).
- [28] Hritom Das et al. “Rfam: Reset-failure-aware-model for hfo₂-based memristor to enhance the reliability of neuromorphic design”. In: *Proceedings of the Great Lakes Symposium on VLSI 2023*. 2023, pp. 281–286 (cit. on pp. 33, 34).

- [29] Mike Davies, Narayan Srinivasa, et al. “Loihi: A Neuromorphic Manycore Processor With On-Chip Learning”. In: *Ieee Micro* 38.1 (2018), pp. 82–99 (cit. on p. 74).
- [30] Chao Du et al. “Reservoir computing using dynamic memristors for temporal information processing”. In: *Nature communications* 8.1 (2017), p. 2204 (cit. on pp. 8, 146).
- [31] AV Fadeev and KV Rudenko. “To the issue of the memristor’s hrs and lrs states degradation and data retention time”. In: *Russian Microelectronics* 50 (2021), pp. 311–325 (cit. on p. 33).
- [32] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002 (cit. on p. 13).
- [33] Mehrshad Golesorkhi et al. “The Brain and its Time: Intrinsic Neural Timescales are Key for Input Processing”. In: *Communications biology* 4.1 (2021), p. 970 (cit. on p. 71).
- [34] Sarthak Gupta, Satrajit Chakraborty, and Chetan Singh Thakur. “Neuromorphic time-multiplexed reservoir computing with on-the-fly weight generation for edge devices”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 24).
- [35] Geonhui Han et al. “Alternative negative weight for simpler hardware implementation of synapse device based neuromorphic system”. In: *Scientific reports* (2021) (cit. on p. 150).
- [36] Joon-Kyu Han, Seong-Yun Yun, et al. “A review of artificial spiking neuron devices for neural processing and sensing”. In: *Advanced Functional Materials* 32.33 (2022), p. 2204102 (cit. on pp. 21, 74).
- [37] Lixia Han, Linlin Cai, et al. “Peking University Analog-Switching Resistive Random Access Memory (RRAM) Verilog-A model Version 3.1.0”. In: (2021. doi:10.21981/MRFT-C373). DOI: 10.21981/MRFT-C373 (cit. on p. 45).
- [38] Charles R Harris, K Jarrod Millman, et al. “Array programming with NumPy”. In: *Nature* (2020) (cit. on p. 31).

- [39] Alex Henderson et al. “Memristor Based Circuit Design for Liquid State Machine Verified with Temporal Classification”. In: *International Joint Conference on Neural Networks (IJCNN 2022)*. IEEE. 2022, pp. 1–9 (cit. on p. [144](#)).
- [40] Stephen Alexander Henderson Jr. “A Memristor-Based Liquid State Machine for Auditory Signal Recognition”. PhD thesis. University of Dayton, 2021 (cit. on p. [144](#)).
- [41] Reza Hojabr, Mehdi Modarressi, et al. “Customizing Clos Network-on-chip for Neural Networks”. In: *IEEE Transactions on Computers* 66.11 (2017), pp. 1865–1877 (cit. on pp. [4](#), [19](#), [65](#), [69](#), [96](#)).
- [42] Reza Hojabr, Mehdi Modarressi, et al. “Customizing clos network-on-chip for neural networks”. In: *IEEE Transactions on Computers* 66.11 (2017), pp. 1865–1877 (cit. on p. [67](#)).
- [43] Md Razuan Hossain et al. “Memristor based Reservoir Network for Chaotic Time Series Prediction”. In: *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME 2022)*. IEEE. 2022, pp. 1–6 (cit. on p. [147](#)).
- [44] Ogbodo Mark Ikechukwu, Khanh N Dang, et al. “On the design of a fault-tolerant scalable three dimensional NoC-based digital neuromorphic system with on-chip learning”. In: *IEEE Access* 9 (2021), pp. 64331–64345 (cit. on p. [21](#)).
- [45] Giacomo Indiveri, Bernabé Linares-Barranco, et al. “Neuromorphic silicon neuron circuits”. In: *Frontiers in neuroscience* 5 (2011), p. 73 (cit. on p. [21](#)).
- [46] Eugene M Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572 (cit. on p. [13](#)).
- [47] Herbert Jaeger. “The “echo state” approach to analysing and training recurrent neural networks-with an erratum note”. In: *German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13 (cit. on pp. [7](#), [13](#), [22](#), [23](#), [100](#)).
- [48] Herbert Jaeger and Harald Haas. “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication”. In: *science* 304.5667 (2004), pp. 78–80 (cit. on pp. [103](#), [108](#), [123](#)).

- [49] Herbert Jaeger, Wolfgang Maass, and Jose Principe. “Special issue on echo state networks and liquid state machines.” In: (2007) (cit. on pp. 6, 47, 57).
- [50] Amirhossein Javanshir, Thanh Thi Nguyen, et al. “Advancements in Algorithms and Neuromorphic Hardware for Spiking Neural Networks”. In: *Neural Computation* 34.6 (2022), pp. 1289–1328 (cit. on p. 21).
- [51] Yikun Jiang and Mei Yang. “Circuit design of Clos-based on-chip interconnection networks”. In: *Microprocessors and Microsystems* 46 (2016), pp. 184–192 (cit. on p. 81).
- [52] Sung Hyun Jo, Kuk-Hwan Kim, and Wei Lu. “High-density crossbar arrays based on a Si memristive system”. In: *Nano letters* 9.2 (2009), pp. 870–874 (cit. on p. 15).
- [53] A. Joubert et al. “Hardware spiking neurons design: Analog or digital?” In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 2012, pp. 1–5. DOI: 10.1109/IJCNN.2012.6252600 (cit. on p. 66).
- [54] Wonjoo Kim et al. “Forming-free metal-oxide ReRAM by oxygen ion implantation process”. In: *2016 IEEE International Electron Devices Meeting (IEDM)*. 2016, pp. 4.4.1–4.4.4. DOI: 10.1109/IEDM.2016.7838345 (cit. on p. 27).
- [55] Dhireesha Kudithipudi et al. “Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing”. In: *Frontiers in neuroscience* 9 (2016), p. 502 (cit. on pp. 24, 144).
- [56] Luis F. Lago-Fernández et al. “Fast Response and Temporal Coherent Oscillations in Small-World Networks”. In: *Phys. Rev. Lett.* 84 (Mar. 2000), pp. 2758–2761 (cit. on p. 75).
- [57] Tor Sverre Lande. *Neuromorphic Systems Engineering Neural Networks in Silicon*. eng. The Springer International Series in Engineering and Computer Science, Analog Circuits and Signal Processing, 447. Boston, MA: Springer US, 1998. ISBN: 9780585280011 (cit. on pp. 52, 66, 81).

- [58] Kundo Lee and Tomoki Hamagami. “Reservoir Computing for Scalable Hardware with Block-Based Neural Network”. In: *IEEEJ Transactions on Electrical and Electronic Engineering* 16 (Aug. 2021). DOI: 10.1002/tee.23473 (cit. on p. 25).
- [59] Boxun Li et al. “ICE: Inline calibration for memristor crossbar-based computing engine”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE 2014)*. IEEE. 2014 (cit. on p. 154).
- [60] Xiangpeng Liang et al. “Rotating neurons for all-analog implementation of cyclic reservoir computing”. In: *Nature Communications* 13 (Mar. 2022) (cit. on p. 24).
- [61] Maximilian Liehr. “Reliability and Performance Optimization of Resistive Random-Access Memory Devices for Advanced Computing and Memory Applications”. In: (2022) (cit. on pp. 15, 17, 27, 147).
- [62] Maximilian Liehr, Jubin Hazra, et al. “Fabrication and performance of hybrid RERAM-CMOS circuit elements for dynamic neural networks”. In: *Proceedings of the International Conference on Neuromorphic Systems*. 2019 (cit. on p. 17).
- [63] Maximilian Liehr, Jubin Hazra, et al. “Impact of Switching Variability of 65nm CMOS Integrated Hafnium Dioxide-based ReRAM Devices on Distinct Level Operations”. In: *International Integrated Reliability Workshop*. 2020 (cit. on pp. 17, 27, 28, 31, 39, 46, 147).
- [64] Maximilian Liehr et al. “Failure Analysis of 65nm CMOS Integrated Nanoscale ReRAM Devices on a 300mm Wafer Platform”. In: *2022 IEEE International Integrated Reliability Workshop (IIRW)*. IEEE. 2022, pp. 1–7 (cit. on pp. 27, 35–37).
- [65] Dingbang Liu, Hao Yu, and Yang Chai. “Low-power computing with neuromorphic engineering”. In: *Advanced Intelligent Systems* 3.2 (2021), p. 2000150 (cit. on pp. 100, 102).
- [66] Mantas Lukoševičius and Herbert Jaeger. “Reservoir computing approaches to recurrent neural network training”. In: *Computer science review* 3.3 (2009), pp. 127–149 (cit. on p. 100).

- [67] Wolfgang Maass, Thomas Natschläger, and Henry Markram. “Real-time computing without stable states: A new framework for neural computation based on perturbations”. In: *Neural computation* 14.11 (2002), pp. 2531–2560 (cit. on pp. [1](#), [7](#), [13](#), [23](#), [100](#)).
- [68] Misha Mahowald. “VLSI analogs of neuronal visual processing: a synthesis of form and function”. In: (1992) (cit. on p. [4](#)).
- [69] Luca Manneschi, Andrew C Lin, and Eleni Vasilaki. “SpaRCe: Improved learning of reservoir computing systems through sparse representations”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on pp. [6](#), [75](#)).
- [70] A.J. Martin and M. Nystrom. “Asynchronous Techniques for System-on-Chip Design”. In: *Proceedings of the IEEE* 94.6 (2006), pp. 1089–1120. DOI: 10.1109/JPROC.2006.875789 (cit. on p. [66](#)).
- [71] Nathan R. McDonald, Robinson E. Pino, et al. “Analysis of dynamic linear and non-linear memristor device models for emerging neuromorphic computing hardware design”. In: *International Joint Conference on Neural Networks* (2010) (cit. on p. [45](#)).
- [72] Carver Mead. “Neuromorphic electronic systems”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636 (cit. on pp. [1](#), [66](#), [74](#)).
- [73] Gianluca Milano et al. “In materia reservoir computing with a fully memristive architecture based on self-organizing nanowire networks”. In: *Nature Materials* 21.2 (2022), pp. 195–202 (cit. on pp. [24](#), [102](#), [144](#)).
- [74] J. Parker Mitchell, Mark E. Dean, et al. “DANNA 2: Dynamic Adaptive Neural Network Arrays”. In: *ICONS* (2018). 2018. ISBN: 9781450365444 (cit. on p. [21](#)).
- [75] Dharmendra S Modha and Raghavendra Singh. “Network Architecture of the Long-Distance Pathways in the Macaque Brain”. In: *Proceedings of the National Academy of Sciences* 107.30 (2010), pp. 13485–13490 (cit. on pp. [75](#), [76](#)).
- [76] John Moon, Yuting Wu, and Wei D Lu. “Hierarchical architectures in reservoir computing systems”. In: *Neuromorphic Computing and Engineering* 1.1 (2021), p. 014006 (cit. on pp. [22](#), [25](#)).

- [77] John Moon et al. “Temporal data classification and forecasting using a memristor-based reservoir computing system”. In: *Nature Electronics* 2.10 (2019), pp. 480–487 (cit. on p. 8).
- [78] Saber Moradi et al. “A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)”. In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (2018), pp. 106–122. DOI: 10.1109/TBCAS.2017.2759700 (cit. on pp. 7, 74, 77).
- [79] Fabiha Nowshin et al. “Recent Advances in Reservoir Computing With A Focus on Electronic Reservoirs”. In: *International Green and Sustainable Computing Workshops (2020)*. 2020 (cit. on pp. 144, 146).
- [80] Fabiha Nowshin. “Spiking neural network with memristive based computing-in-memory circuits and architecture”. PhD thesis. Virginia Tech, 2021 (cit. on p. 144).
- [81] Andrés Otero et al. “Evolutionary FPGA-Based Spiking Neural Networks for Continual Learning”. In: *International Symposium on Applied Reconfigurable Computing*. Springer. 2023, pp. 260–274 (cit. on p. 99).
- [82] Jongkil Park et al. “Hierarchical address event routing for reconfigurable large-scale neuromorphic systems”. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2408–2422 (cit. on pp. 4, 7, 19, 77).
- [83] Hélene Paugam-Moisy and Sander M Bohte. “Computing with spiking neuron networks.” In: *Handbook of natural computing* 1 (2012), pp. 1–47 (cit. on pp. 7, 22).
- [84] Adam Perrett et al. “Online learning in SNNs with e-prop and Neuromorphic Hardware”. In: *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*. 2022, pp. 32–39 (cit. on pp. 7, 8, 22, 100).
- [85] Matthew D Pickett, Dmitri B Strukov, et al. “Switching dynamics in titanium dioxide memristive devices”. In: *Journal of Applied Physics* (2009) (cit. on p. 27).

- [86] Robinson E Pino, James W Bohl, et al. “Compact method for modeling and simulation of memristor devices: Ion conductor chalcogenide-based memristor devices”. In: *IEEE/ACM International Symposium on Nanoscale Architectures*. 2010 (cit. on p. [45](#)).
- [87] Junran Pu et al. “A low power and low area router with congestion-aware routing algorithm for spiking neural network hardware implementations”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.1 (2020), pp. 471–475 (cit. on pp. [5](#), [19](#)).
- [88] Francesco Maria Puglisi, Tommaso Zanotti, and Paolo Pavan. “Unimore Resistive Random Access Memory (RRAM) Verilog-A Model”. In: (2019. doi:10.21981/15GF-KX29). DOI: doi:10.21981/15GF-KX29 (cit. on p. [45](#)).
- [89] Manu Rathore and Garrett S Rose. “AnSpiCS-Net: Reconfigurable Network-on-Chip for Analog Spiking Recurrent Neural Networks”. In: *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2024 (cit. on p. [5](#)).
- [90] Manu Rathore and Garrett S. Rose. “Leveraging Sparsity of SRNNs for Reconfigurable and Resource-Efficient Network-on-Chip”. In: *Neuro-Inspired Computational Elements (NICE)*. IEEE. 2024 (cit. on pp. [20](#), [21](#)).
- [91] Manu Rathore and Garrett S. Rose. “SpiCS-Net : Circuit Switched Network on Chip for Area-Efficient Spiking Recurrent Neural Networks”. In: *International Conference on VLSI Design*. IEEE. 2024 (cit. on pp. [4](#), [5](#), [19](#)).
- [92] Manu Rathore et al. “A Compact Model for the Variable Switching Dynamics of HfO₂ Memristors”. In: *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2022, pp. 1–4 (cit. on pp. [26](#), [27](#), [32](#), [35](#), [45](#)).
- [93] Manu Rathore et al. “Reliability Analysis of Memristive Reservoir Computing Architecture”. In: *Proceedings of the Great Lakes Symposium on VLSI 2023*. 2023, pp. 131–136 (cit. on pp. [3](#), [8](#), [13](#), [45](#), [66](#), [68](#), [102](#), [145](#), [152](#), [156–158](#), [160](#)).
- [94] Catherine D Schuman, J Parker Mitchell, et al. “Evolutionary optimization for neuromorphic systems”. In: *Proceedings of the Neuro-inspired Computational Elements Workshop*. 2020, pp. 1–9 (cit. on p. [99](#)).

- [95] Catherine D Schuman et al. “A survey of neuromorphic computing and neural networks in hardware”. In: *arXiv preprint arXiv:1705.06963* (2017) (cit. on p. 21).
- [96] Catherine D Schuman et al. “Resilience and robustness of spiking neural networks for neuromorphic systems”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–10 (cit. on pp. 6, 47, 57).
- [97] Luís F Seoane. “Evolutionary aspects of reservoir computing”. In: *Philosophical Transactions of the Royal Society B* 374.1774 (2019), p. 20180377 (cit. on pp. 24, 48).
- [98] Amar Shrestha, Haowen Fang, et al. “A Survey on Neuromorphic Computing: Models and Hardware”. In: *IEEE Circuits and Systems Magazine* 22.2 (2022), pp. 6–35 (cit. on p. 74).
- [99] Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. “Bayesian continual learning via spiking neural networks”. In: *Frontiers in Computational Neuroscience* 16 (2022), p. 1037976 (cit. on pp. 22, 99).
- [100] Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. “Spiking neural networks—Part III: Neuromorphic communications”. In: *IEEE Communications Letters* 25.6 (2021), pp. 1746–1750 (cit. on p. 74).
- [101] Erik S Skibinsky-Gitlin et al. “Cyclic reservoir computing with FPGA devices for efficient channel equalization”. In: *Artificial Intelligence and Soft Computing: 17th International Conference, ICAISC 2018, Zakopane, Poland, June 3-7, 2018, Proceedings, Part I* 17. Springer. 2018, pp. 226–234 (cit. on pp. 102, 109, 123).
- [102] Kamilya Smagulova and Alex Pappachen James. “A survey on LSTM memristive neural network architectures and applications”. In: *The European Physical Journal Special Topics* 228.10 (2019), pp. 2313–2324 (cit. on p. 22).
- [103] Narayan Srinivasa, Deying Zhang, et al. “A robust and scalable neuromorphic communication system by combining synaptic time multiplexing and MIMO-OFDM”. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.3 (2014), pp. 585–608 (cit. on p. 21).

- [104] Dmitri B Strukov and R Stanley Williams. “Exponential ionic drift: fast switching and low volatility of thin-film memristors”. In: *Applied Physics A* (2009) (cit. on p. [27](#)).
- [105] Dmitri B Strukov et al. “The missing memristor found”. In: *nature* 453.7191 (2008), pp. 80–83 (cit. on p. [15](#)).
- [106] Zhong Sun et al. “One-step regression and classification with cross-point resistive memory arrays”. In: *Science advances* 6.5 (2020), eaay2378 (cit. on p. [25](#)).
- [107] Elena I Suvorova et al. “Structure, Oxygen Content and Electric Properties of Titanium Nitride Electrodes in TiNx/La: HfO2/TiNx Stacks Grown by PEALD on SiO2/Si”. In: *Nanomaterials* 12.20 (2022), p. 3608 (cit. on p. [35](#)).
- [108] Aboozar Taherkhani et al. “A review of learning in biologically plausible spiking neural networks”. In: *Neural Networks* 122 (2020), pp. 253–272 (cit. on p. [7](#)).
- [109] Norikazu Takahashi. “On clustering coefficients of graphs with the fixed numbers of vertices and edges”. In: *2009 European Conference on Circuit Theory and Design*. IEEE. 2009, pp. 814–817 (cit. on p. [83](#)).
- [110] Gouhei Tanaka et al. “Recent advances in physical reservoir computing: A review”. In: *Neural Networks* 115 (Mar. 2019). DOI: 10.1016/j.neunet.2019.03.005 (cit. on pp. [1](#), [24](#), [146](#)).
- [111] Chetan Singh Thakur et al. “Large-scale neuromorphic spiking array processors: A quest to mimic the brain”. In: *Frontiers in neuroscience* 12 (2018), p. 891 (cit. on pp. [4](#), [19](#), [75](#)).
- [112] Chetan Singh Thakur et al. “An online learning algorithm for neuromorphic hardware implementation”. In: *arXiv preprint arXiv:1505.02495* (2015) (cit. on p. [100](#)).
- [113] Huy-The Vu, Yuichi Okuyama, and Abderazek Ben Abdallah. “Analytical performance assessment and high-throughput low-latency spike routing algorithm for spiking neural network systems”. In: *The Journal of Supercomputing* 75 (2019), pp. 5367–5397 (cit. on p. [21](#)).

- [114] Th H Vu, Yuichi Okuyama, and Abderazek Ben Abdallah. “Comprehensive analytic performance assessment and K-means based multicast routing algorithm and architecture for 3D-NoC of spiking neurons”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.4 (2019), pp. 1–28 (cit. on pp. [21](#), [67](#), [69](#), [96](#)).
- [115] Tianshi Wang and Jaijeet Roychowdhury. “Well-posed models of memristive devices”. In: *arXiv preprint arXiv:1605.04897* (2016) (cit. on pp. [26](#), [27](#), [44](#)).
- [116] Duncan J Watts and Steven H Strogatz. “Collective dynamics of ‘small-world’ networks”. In: *nature* 393.6684 (1998), pp. 440–442 (cit. on p. [83](#)).
- [117] Ryan Weiss et al. “STDP Based Online Learning for a Current-Controlled Memristive Synapse”. In: *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2022, pp. 1–4. DOI: 10.1109/MWSCAS54063.2022.9859294 (cit. on pp. [18](#), [28](#), [42](#)).
- [118] H-S Philip Wong et al. “Metal–oxide RRAM”. In: *Proceedings of the IEEE* 100.6 (2012), pp. 1951–1970 (cit. on p. [147](#)).
- [119] T Patrick Xiao et al. “Analog architectures for neural network acceleration based on non-volatile memory”. In: *Applied Physics Reviews* 7.3 (2020), p. 031301 (cit. on pp. [144](#), [147](#)).
- [120] Zhihua Yong et al. “Tuning oxygen vacancies and resistive switching properties in ultra-thin HfO₂ RRAM via TiN bottom electrode and interface engineering”. In: *Applied Surface Science* 551 (2021), p. 149386 (cit. on p. [35](#)).
- [121] Aaron R Young et al. “A review of spiking neuromorphic hardware communication systems”. In: *IEEE Access* 7 (2019), pp. 135606–135620 (cit. on pp. [3](#), [4](#), [13](#), [19](#), [21](#), [74](#)).
- [122] Lu Zhang, Zhijie Chen, et al. “A compact modeling of TiO₂-TiO_{2-x} memristor”. In: *Applied Physics Letters* (2013) (cit. on p. [27](#)).
- [123] Lu Zhang, Ning Ge, et al. “Low voltage two-state-variable memristor model of vacancy-drift resistive switches”. In: *Applied Physics A* (2015) (cit. on p. [27](#)).

- [124] Yanan Zhong et al. “Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing”. In: *Nature communications* 12.1 (2021) (cit. on p. [146](#)).

Appendix

Appendix A

Reliability Analysis of Memristive Reservoir Computing Architecture

The integration of memristor-based Spiking Recurrent Neural Networks (SRNNs) within the framework of Reservoir Computing (RC), as depicted in Fig. A.1, offers a compelling avenue for the development of neuromorphic systems tailored for spatio-temporal data processing. This approach capitalizes on the advantages afforded by memristor technology alongside the RC training methodology, resulting in reduced area, power, and computational demands. Notably, SRNNs featuring memristive synapses have garnered significant attention as reservoirs owing to the intricate dynamics engendered by recurrent connections [73, 80, 55, 79, 40, 39].

These systems have the potential to offer efficient edge computing solutions. However, as mentioned in Chapter 3, a notable challenge in designing memristive circuits stems from the inherent device variability linked with these devices. Issues such as noise in read/write resistance states, memristor conductance drift (MCD) under electric fields, diminishing memory windows, and endurance problems are prominent concerns. These factors significantly influence the signal-to-noise ratio and accuracy in memristor-based computations [119].

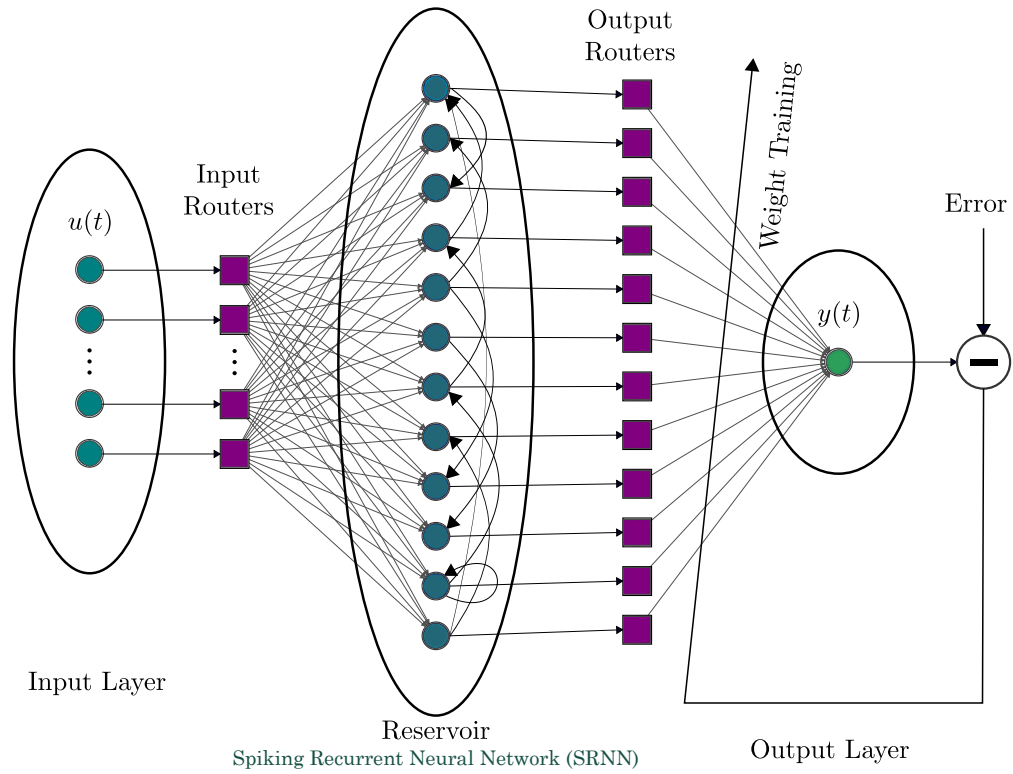


Figure A.1: Reservoir Computing (RC) system employing a Spiking Recurrent Neural Network (SRNN) architecture. The SRNN reservoir exhibits dynamic temporal processing capabilities essential for tasks such as time-series prediction and nonlinear data processing [93]

Consequently, these memristor devices find potential applications in scenarios where device variability does not significantly affect overall system accuracy. In fact, in some cases such variability can be advantageous, contributing to the rich dynamics of the memristor-based reservoirs [110]. For example, [19] leveraged a memristor crossbar-based reservoir, exploiting the diverse dynamical properties contributed by memristor device variability. [4] proposed a novel device configuration for memristor which is better suited for the Reservoir Computing (RC) approach, achieving high performance accuracy for classification tasks based on experimental findings. Additionally, [30] introduced a compact reservoir architecture utilizing a dynamic memristor array, comprising just 88 memristors. [124] utilized a controllable mask process to deliberately introduce specific variability characteristics into dynamic memristors. They demonstrated the use of a single memristor as a reservoir with a small footprint, showcasing how the variability of a single device can be harnessed to achieve complex dynamics. Additionally, [79] also explores various examples of memristive reservoir architectures that offer advantages for classification and prediction tasks.

Some of these designs utilize memristors to build dynamic reservoir systems, where variability is not considered a drawback but rather a feature that enhances the system’s complex dynamics. Hence, while existing studies recognize the potential benefits of device variability of memristors for improving reservoir dynamics, they do not extensively examine its detrimental impact on both the Spiking Recurrent Neural Network (SRNN) reservoir and the memristive crossbar output readout layer. Nevertheless, these studies consistently emphasize several notable advantages, such as power efficiency, compact designs with fewer components, reduced data and memory requirements for training, and high computational accuracy. Additionally, SRNN-based systems demonstrate outstanding performance in processing temporal data, tackling nonlinear tasks, and even predicting chaotic time-series, thus establishing their suitability for a diverse range of applications in this novel computing paradigm.

Considering their promising potential and widespread adoption, memristor-based systems require a comprehensive high-level study to comprehend the effects of memristor variability on overall performance.

HfO₂ memristors possess advantageous properties that render them promising for various applications. These memristors, utilizing hafnium oxide as their base material, demonstrate non-volatile memory characteristics, ensuring data retention even in the absence of power [118]. Moreover, they exhibit low power consumption, making them suitable for energy-efficient operations. The high endurance and reliability of HfO₂ memristors enable repeated read and write cycles without significant degradation, enhancing their durability and longevity. Additionally, their scalability aligns well with advanced nanofabrication techniques, allowing for integration into next-generation computing architectures with ease [61]. In summary, these combined attributes establish HfO₂ memristors as crucial elements in emerging technologies, including neuromorphic computing, artificial intelligence, and non-volatile memory storage systems.

Thus, A HfO₂ memristor-based Reservoir Computing (RC) system holds the potential to deliver efficient edge computing solutions. However, in the context of HfO₂ memristors, the devices display stochastic behavior and resistance variability in their Low Resistance State (LRS) and High Resistance State (HRS). This inherent variability imposes limitations on their practical application and overall performance [63]. Issues such as read/write noise for both Low Resistance State (LRS) and High Resistance State (HRS), conductance drift induced by electric field across the device (MCD), and the diminishing memory window are primary concerns [119].

Here, we delve into how device variability affects the performance of a memristor-based reservoir computer. We evaluate its performance on chaotic time-series prediction tasks, a domain where the reservoir computing approach excels [43]. Additionally, we investigate the impact of different kinds of noise in an image classification task. Remarkably, we unveil a distinct correlation between the impact of MCD and the synaptic plasticity in the Spiking Recurrent Neural Network (SRNN). Our analysis includes investigating the read/write noise in memristive synapses and the conductance weights in the memristor crossbar Dot Product Engine (DPE) Output Layer. Moreover, we examine the consequences of input data noise and memristor variability in SRNN synapse, aiming to offer a thorough assessment of the overall system’s performance.

This analysis focuses on a Spiking Recurrent Neural Network (SRNN) reservoir based RC system as depicted in Fig. A.1. Our study endeavors to quantify the level of sensitivity within a Reservoir Computing (RC) system. Through a thorough analysis, we aim to offer insights and recommendations for making informed design choices towards achieving a practical hardware-based RC implementation.

A SRNN based RC Architecture

The RC system we’ve designed incorporates SRNN for the reservoir layer (depicted in Fig. A.1). Each circle in the figure symbolizes nodes or neurons within the SRNN. These nodes utilize the adaptive Leaky Integrate and Fire (LIF) neuron model. In the LIF neuron structure, as discussed in previous chapters, an analog integrator accumulates read currents from memristive synapses followed by a comparator circuit that compares the integrated input currents with a tunable threshold to determine occurrence of a spike event. If the accumulated potential surpasses the threshold value, a fire event generates a spike at neuron output and the neuron enters a refractory period. After spiking, the accumulated potential of the neuron is reset to a resting membrane potential. If not in the refractory period, the integrator continues to accumulate input currents from memristive synapses with a programmed leak factor. The connections between nodes represent memristor-based synapses (see Fig. A.2). Additionally, the architecture’s output layer is comprised of a memristive crossbar array, which acts as the output readout layer or Dot Product Engine (DPE).

Memristor-based Synapse

The memristive synapse illustrated in Fig. A.2 operates as a current-controlled synapse utilizing HfO_2 memristor devices in a 65 nm CMOS process. The different branches of the synapse circuit facilitate four key operations on the memristor: Form, Reset, Set, and Read. The Forming operation in Fig. A.2(a) is an initial, one-time process aimed at transitioning the very high resistance of the newly fabricated device (approximately $10\text{ M}\Omega$) to a Low Resistance State (LRS) of around $10\text{ k}\Omega$. This lowering of resistance is the result of formation of the conduction filament (CF). Subsequently, the memristor switching to a High Resistance

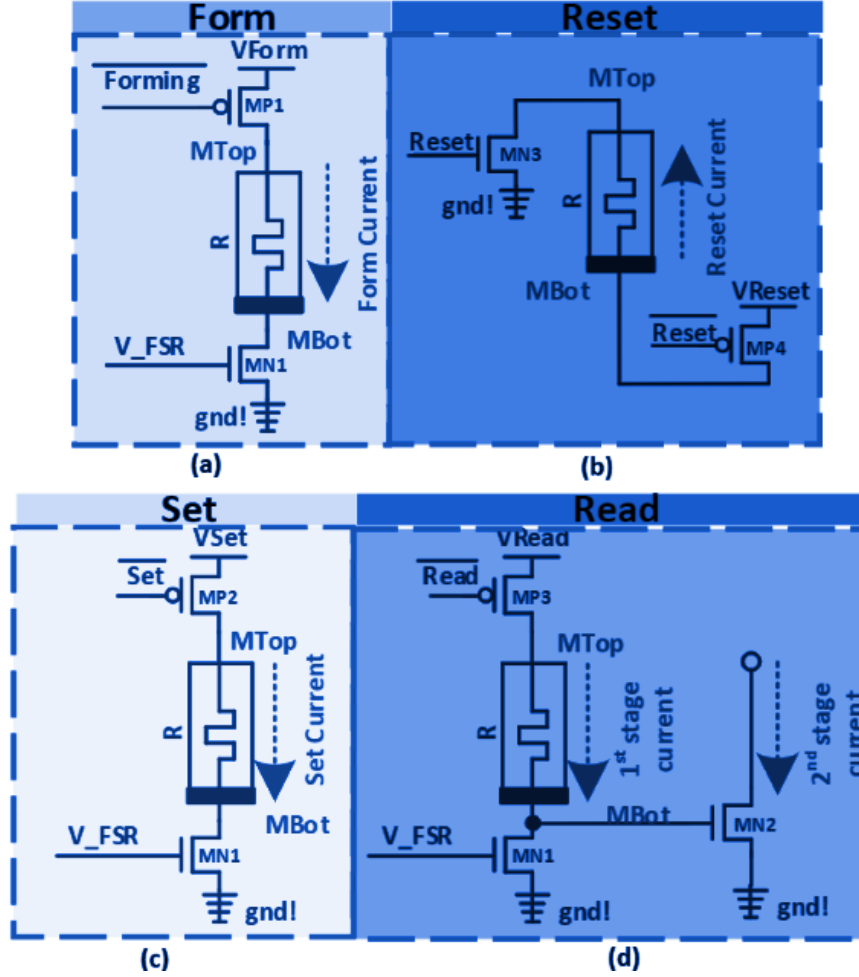


Figure A.2: The HfO₂ memristor-based synapse, designed and fabricated using a 65 nm CMOS process, incorporates various operations through different PMOS branches. These synapses serve as synaptic connections within the SRNN and within the memristor crossbar arrays at cross-points, enabling programming to different weight/conductance values.

State (HRS) of approximately $100\text{ k}\Omega$ is done through the Reset operation in Fig. A.2(b). This operation involves current flow from MBot to Mtop, partially breaking the CF with the assistance of $MP4$ and $MN3$. The SET operation in Fig. A.2(c) allows the memristor device to be tuned to multiple resistance levels near the LRS. During the SET operation, $MP2$ and $MN1$ are digitally controlled, and V_FSR is adjusted between 0.8 mV to 1.2 V . This V_FSR parameter sets the device at LRS to a specified resistance value. Notably, the V_FSR signal and $MN1$ device are utilized for three operations: Forming, Set, and Read. In memristor circuit, the read operation refers to the usage of the memristor for setting different weights on the synapses and DPE branches. After a successful SET operation, the read operation is done using $MP3$, $MN1$, and $MN2$ as depicted in Fig. A.2(d). This entails a two-stage operation designed to minimize power consumption and increase synapse control capability. The gate voltage of $MP3$ and $MN1$ is set to 0 V and 0.6 V , respectively, in the first stage. The device $MN1$ operates in the linear region of operation and acts as a current limiter. This generates a first-stage current, which is used to derive $MN2$. Lastly, the Read current, accumulated from the synapse, is sensed at the drain of $MN2$ as the second-stage current. Subsequently, this Read current is forwarded to the neuron. In Cadence Spectre simulation, the energy consumption of the synapse for each spike event is approximately $5pJ$ at $VDD = 1.2V$.

Memristor Crossbar: Output Readout DPE layer

The memristive crossbar-based Dot Product Engine (DPE) presents an energy-efficient approach for achieving vector-matrix multiplication operation in analog manner. This setup allows the programming of memristor device to multiple resistance values in the LRS, facilitating the multiplication operation through the input voltage and device conductance resulting in the output current calculation ($I = V * G$). Constructed using the same synapse as a functional unit, the memristor crossbar array read operation for one column is depicted in Fig. A.2(d). In this context, an abstract system-level RC model is trained with both positive and negative weights, which can be correlated with the analog conductance value of the memristor using the methodology proposed by [35].

A.1 Hardware Model

The system's performance is evaluated using system modeling in Python. Within the reservoir, the SRNN comprises n neurons interconnected via memristor-based synapses. The spike raster depicted in Fig. A.3 visually represents the spiking activity of neurons in the SRNN. This spike raster aids in extracting the spike frequency from output of each neuron, which is then fed to the readout layer as a vector. In the readout layer, a multiply-accumulate dot product operation projects these input spike frequencies from an n dimensional to an m dimensional space. Training is focused on the output layer weights using gradient descent technique.

Synapse and Neuron

The neurons maintain a non-negative membrane potential, which is influenced by the analog valued inputs received via multiple input synapses. Each neuron also contains a list of its output synaptic connections. A programmable leak within each neuron determines the constant rate at which the membrane potential keeps leaking or decreasing, regardless of spiking activity. Each synapse can connect to two neurons, one at its input and one at its output, and is associated with a parameterized weight. Additionally, the synapse models Memristor Conductance Drift (MCD), Spike-Timing-Dependent Plasticity (STDP), and read/write noise implemented as a Gaussian distribution.

Reservoir Creation and Initialization

The "create reservoir" function takes the number of neurons and synapse density as inputs to generate a randomized network. In our study, typical system parameters of the SRNN are 30 neurons and a 10% synaptic density. Other parameters of the "create reservoir" function include the range of allowed neuron thresholds, range of leak values, and average synapse weight. These parameters significantly impact the SRNN spiking activity and may require tuning for each application. If the activity on any neuron consistently involves spiking at a constant frequency, valuable information may not be effectively conveyed to the output layer. The same applies to highly infrequent spiking activity.

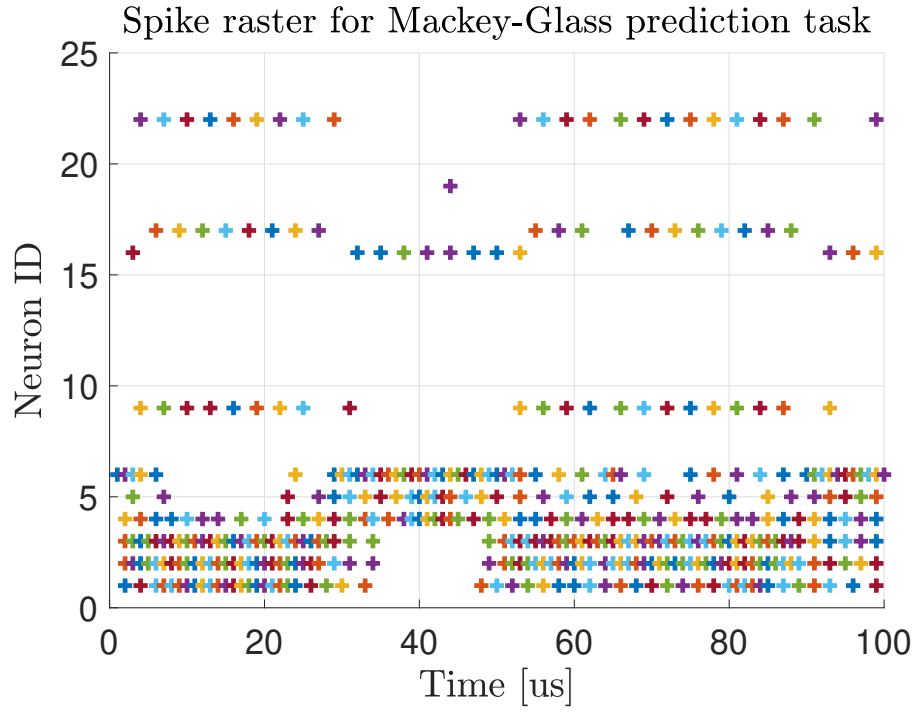


Figure A.3: MackeyGlass time-series: Spike raster representation depicting the activity within the reservoir network. Each spike corresponds to the firing of a neuron in response to the MackeyGlass time-series input, illustrating the dynamic behavior of the reservoir in processing temporal data [93]

Inputs to the RC System

In the time-series prediction task, the neuron input ports in the reservoir are sequentially fed by the real-valued input over time, resulting in evolving SRNN state. The LIF neurons convert a time-based real-valued input signal to output spikes. For the image recognition task, the inputs are fed after encoding the image to spike data. This is done using a rate-based spike encoder which converts amplitude of image pixels into frequency of the spike. This input remains constant throughout the entire simulation. Upon completion of the simulation, a fire matrix is created using the spike activity data and this matrix is inputted to the DPE output readout layer.

Training the Output Layer

The weight matrix, denoted as W , and bias vector, denoted as b , in the DPE, are initialized as uniformly and randomly distributed. The size of the weight matrix is $n \times m$, where n represents the number of neurons and m denotes the number of outputs. In classification tasks, m corresponds to the number of classes that correspond to the target dataset, while in time-series prediction, m equals the total number of dimensions present in the target data.

Using the vector generated using the reservoir spike frequencies as X , in the equation $WX + b = \hat{y}$, a vector of predictions, denoted as $\hat{y}$, is obtained. In classification tasks, the predicted class is determined by selecting the largest value in $\hat{y}$. Conversely, the vector $\hat{y}$ for the time-series prediction task is the relative change required from the final input to form the next input.

To train the weights and biases, a series of input and output pairs are fed into the entire reservoir. The input is processed by the SRNN, and the output is used to calculate the root mean squared error (RMSE) for $\hat{y}$. The derivative of the RMSE with respect to the weights and biases is utilized to compute the updated value for each weight and bias. This derivative, or slope, of the error is initially multiplied by a small value known as the learning rate. This step is essential for reducing the likelihood of overshooting any local minima.

Memristor Variability Modeling

The memristive synapses within the SRNN and the weight matrix of the output layer incorporate memristor variability. Read/write memristor variability is simulated by adjusting the synapse weight used in potentiation through the addition of a random value drawn from a normal distribution at each read/write operation.

Memristor Conductance Drift (MCD) occurs at each spike event in a synapse, leading to a shift in the weight value’s magnitude by a predetermined amount. It is assumed that a voltage of 0.1 V results in a 2 percent deviation from the original state over a period of 1 sec [59].

B Results and Comparisons

The reservoir computer is trained for two distinct tasks: chaotic time series prediction using the Mackey-Glass (MG) and Lorenz datasets, and pattern recognition using the Iris image dataset. Each task offers unique perspectives and application scopes for evaluating a memristive reservoir computer.

B.1 Chaotic Time Series Prediction task

The Mackey-Glass time-series equation is defined as,

$$\frac{dP(t)}{dt} = \frac{aP(t - \tau)}{(1 + P(t - \tau)^n)} - bP(t) \quad (\text{A.1})$$

And Lorenz equation is given by,

$$\frac{dx}{dt} = \sigma(y - x), \frac{dy}{dt} = x(\rho - z) - y, \frac{dz}{dt} = xy - \beta z \quad (\text{A.2})$$

While the MG and Lorenz datasets exhibit variations in the number of variables and their dynamical interactions within the reservoir, their implementation maintains consistency with a SRNN size of 30 neurons and 90 synapses for the reservoir layer. In the time-series prediction task, as the standard deviation (SD) of random Gaussian noise introduced to the normalized

input data increases, the spread of Normalized Root Mean Squared Error (NRMSE) also escalates. This is depicted in the plot shown in Fig. A.4. However, it is noteworthy that the mean of the NRMSE remains relatively unaffected and remains consistent, underscoring the noise resilience of the RC system.

The analysis of memristor variability can be segmented into two stages. The first stage examines the variability associated with memristor read/write operations in both the reservoir and the DPE weights. Second, we consider the effect of MCD and STDP on memristive synapses.

As illustrated in Fig. A.5, it is observed that even as the synapse read/write noise increases, the NRMSE exhibits minimal escalation, particularly when contrasted with the effect of input data noise depicted in the same graph. Consequently, the synaptic variability does not significantly impair the system’s performance. However, there is no conclusive evidence to suggest that synaptic memristive variability confers benefits to SRNN reservoirs. The maximum standard deviation (SD) for synapse and data noise in the graph is 0.1, which is comparable to the synaptic weight factor of 1. Additionally, the input data is normalized to the range of (0, 1).

In Fig. A.6, the impact of memristor variability in DPE weights is illustrated. It is evident that the achievable Normalized Root Mean Squared Error (NRMSE) is highly sensitive to variability in the DPE layer. In fact, the curves representing DPE memristor variability and the combined effect of all noise factors almost overlap for the MG time-series. To delve deeper into this observation, we examine the NRMSE versus Epoch curves depicted in Fig. A.7. During training, for all cases except DPE memristor variability, the NRMSE eventually converges to a small value. However, even with minimal memristor conductance variability in the DPE, the NRMSE diverges and continues to increase with each training epoch. Moreover, it was noted in several instances that if a system was trained with DPE weight variability present in the initial training epochs, even after reducing the variability to lower level in subsequent epochs, the system failed to converge and reduce the NRMSE.

In Fig. A.8, the plot of NRMSE versus Memristor Conductance Drift (MCD) for the MG and Lorenz prediction tasks is presented. For the MG time-series, an increase in MCD results in a corresponding increase in NRMSE. However, this adverse impact on NRMSE is mitigated

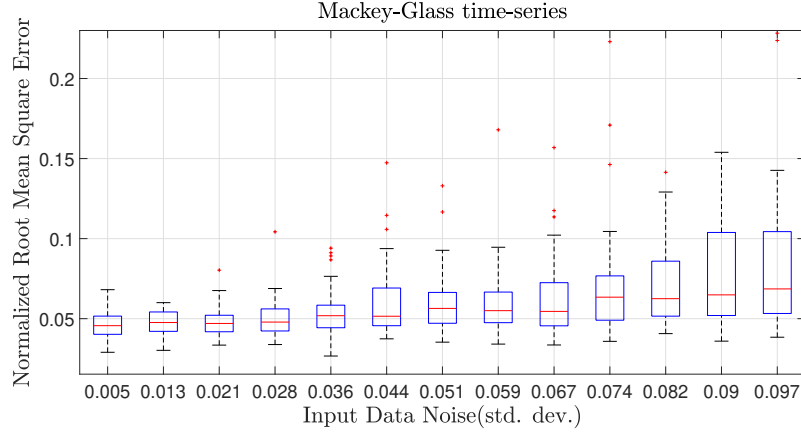


Figure A.4: Box plot illustrating the relationship between the standard deviation (SD) of noise in input data, ranging from 0 0.1, and the resulting standard deviation (SD) in the Normalized Root Mean Square Error (NRMSE). The plot demonstrates that as the SD of input data noise increases, there is a corresponding rise in the SD of NRMSE, indicating impact of in variability on performance [93]

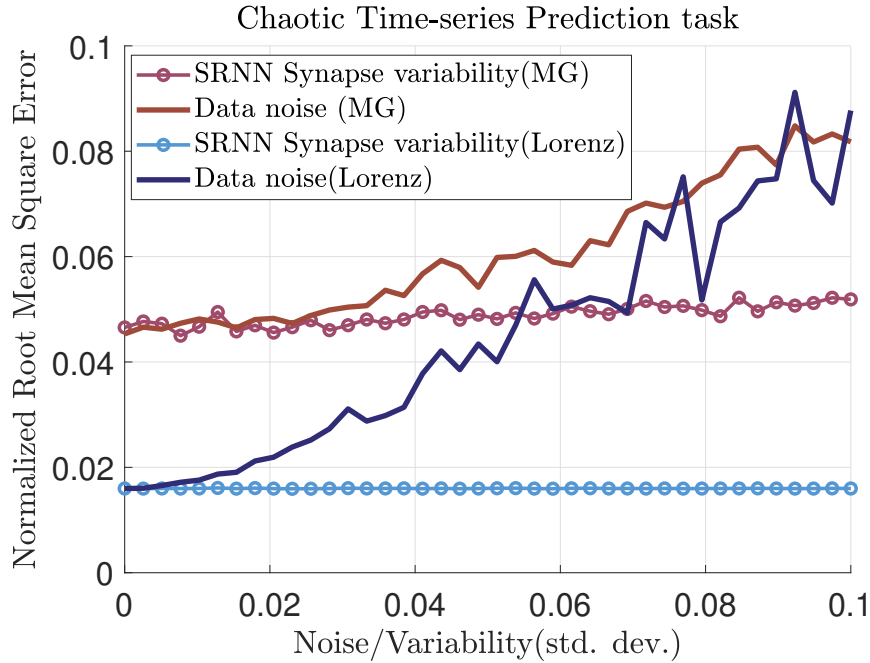


Figure A.5: The influence of input data noise on Normalized Root Mean Square Error (NRMSE) is significantly greater compared to the impact of synapse variability(Fig. A.4). While variations in synapse properties have minimal effect on NRMSE, changes in input data noise levels have a pronounced impact on the accuracy of the RC system's predictions [93]

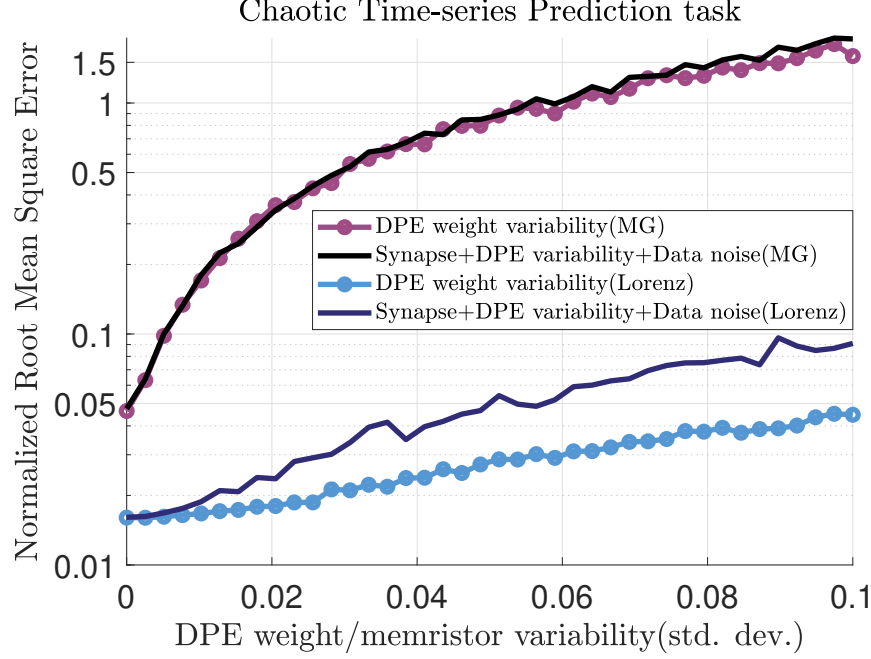


Figure A.6: With increasing memristor variability in memristive crossbar-based Dot Product Engine (DPE) weights, the Normalized Root Mean Square Error (NRMSE) also rises, indicating a direct relationship between DPE weight variability and prediction accuracy. This suggests that fluctuations in DPE weights contribute significantly to prediction errors [93]

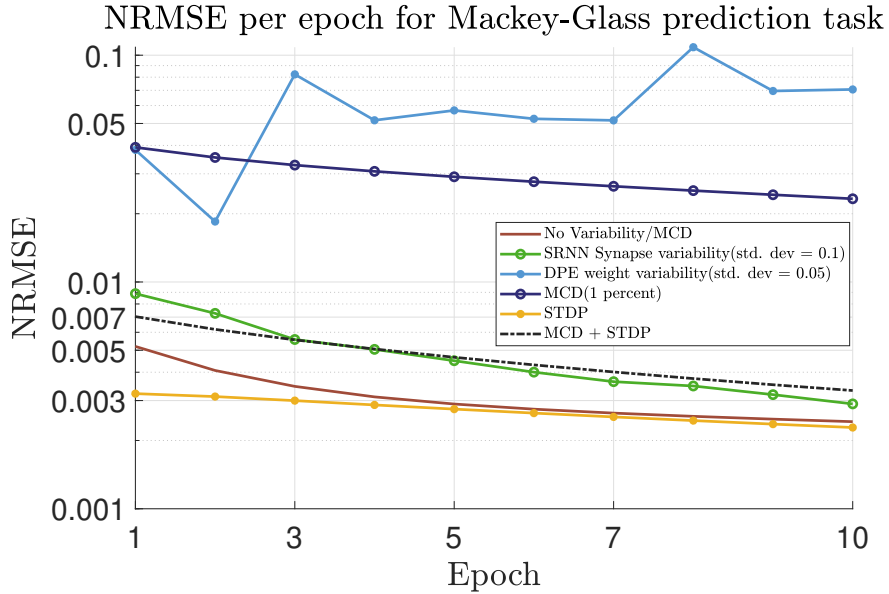


Figure A.7: The Normalized Root Mean Square Error (NRMSE) per Epoch for the Mackey-Glass (MG) dataset reveals the impact of different sources of noise, stochastic effects, and plasticity mechanisms, internal and external to the Reservoir Computing (RC) system. Variations in NRMSE across epochs illustrate how factors such as external sources of noise, synaptic plasticity mechanisms, and stochastic behaviors affect the predictive performance of the RC system [93]

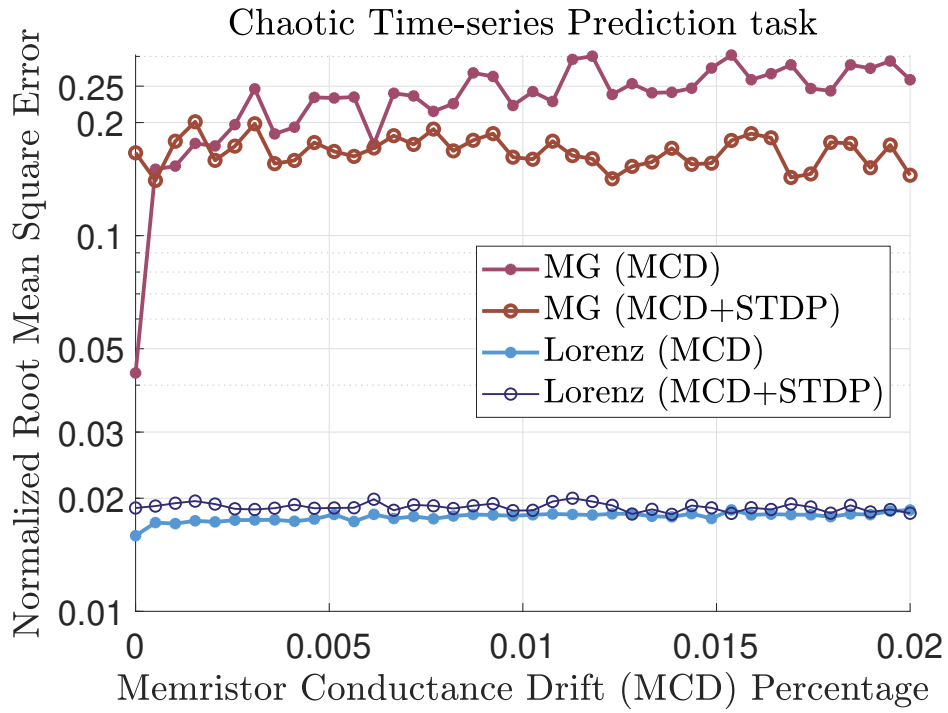


Figure A.8: NRMSE plotted against Memristor Conductance Drift with and without Spike-Timing-Dependent Plasticity (STDP). This comparison highlights the impact of memristor conductance drift, with and without the application of STDP, on the predictive accuracy of the system. Understanding these relationships sheds light on the efficacy of STDP in mitigating the effects of memristor conductance drift on prediction performance [93]

when Spike-Timing-Dependent Plasticity (STDP) is applied concurrently with MCD. This trend is further corroborated by Fig. A.7, where MCD alone leads to significantly higher error compared to the scenario where MCD is applied alongside STDP, or when STDP is applied alone. At the same time, for the Lorenz dataset, the NRMSE with MCD is superior to that with MCD+STDP applied. Nevertheless, the overall trend demonstrates that the NRMSE increases with an escalation in MCD percentage for both applications, whereas it remains relatively stable when MCD is applied in conjunction with STDP.

B.2 Image recognition task

In Fig. A.9, the training accuracy per epoch are presented for the Iris dataset for various scenarios, including applied variability, Memristor Conductance Drift (MCD), and Spike-Timing-Dependent Plasticity (STDP) in the SRNN, is depicted. Interestingly, the classification tasks appear to benefit from MCD, unlike chaotic series prediction tasks. In this image classification case, MCD seems to function as an additional plasticity mechanism alongside STDP. This is evident from the rapid convergence of the MCD+STDP curve, achieving convergence in minimum epochs. Conversely, synapse variability exhibits a detrimental effect, resulting in the lowest accuracy among all cases. Thus, for classification tasks, synapse variability within the reservoir does not yield the same beneficial impact observed in chaotic prediction tasks. Nonetheless, despite the varied impacts of variation, noise, and drift, the system displays resilience to variability and eventually converges to an acceptable accuracy value for all cases.

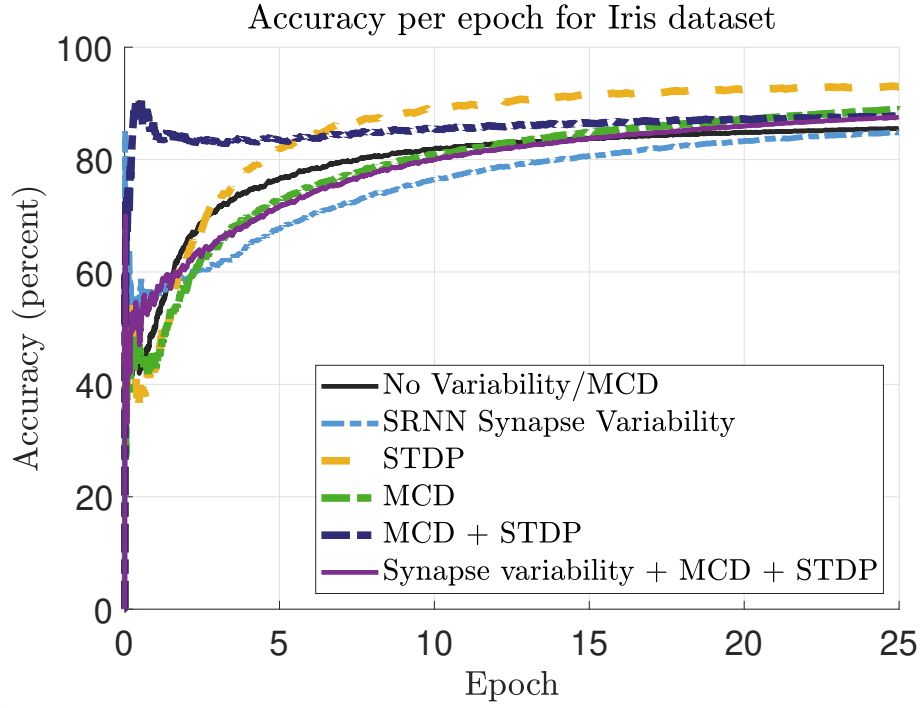


Figure A.9: The accuracy per epoch for the Iris dataset is compared across various scenarios, including synapse variability, Memristor Conductance Drift (MCD), and the application of Spike-Timing-Dependent Plasticity (STDP). This analysis provides insights into how these factors influence the classification performance of the system over the training epochs, offering valuable information for optimizing the performance of the Reservoir Computing system on image recognition tasks [93]

Appendix B

Verilog-A Model for HfO₂ Memristor

```
// VerilogA for model_mem, memr_with_forming, veriloga
//Copyright of the model is maintained by the developers.
    This model is distributed under the terms of the
    Creative Commons Attribution-NonCommercial 4.0
//International Public License
    https://creativecommons.org/licenses/by-nc/4.0/legalcode.
    If you choose to use this model, you are kindly requested to cite.

// Last modified : 02/03/2023 by Manu Rathore

`include "constants.vams"
`include "disciplines.vams"

module rr_hfo_manu (p,n);

    inout          p;          //positive pin
    inout          n;          //negative pin
```

```

electrical      p, n;

//memristor parameters
parameter real HRS          = 50e3;
// high resistance state
parameter real LRS          = 5e3;
// low resistance state
parameter real Vtp          = 0.75;
// positive threshold voltage
parameter real Vtn          = -0.75;
// negative threshold voltage
parameter real tsw_p        = 1e-8;
// time to switch under +V bias
parameter real tsw_n        = 1e-6;
// time to switch under -V bias
parameter real V_rst_fail   = -1.8;
// Reset fail occurs at this applied reset voltage value
parameter real R_rst_fail   = 500;
// (Ohms) Resistance of memristor after reset fail occurs

//window parameters
parameter real alpha_LRS    = 1.05;
parameter real alpha_HRS    = 0.95;
parameter real theta_HRS    = 0.85;
parameter real beta_HRS     = 0.2;
parameter real theta_LRS    = 2.1;
parameter real beta_LRS     = 0.05;

//Fitting coefficients

```

```

parameter real C_LRS      = 7.29;
parameter real C_HRS      = 6.75;
parameter real P_LRS      = 3;
parameter real P_HRS      = 3;

parameter real Rinit      = 50e3; // initial resistance

parameter real HRS_rate   = 0; //Aging rate?
parameter real HRS_nom_spr = 0.001; //Nominal spread

//spread/distribution specification for memristor parameters
parameter real HRS_stddev_rel = 0.05 from [0:0.3];
parameter real LRS_stddev_rel = 0.05 from [0:0.3]; //0.02;
parameter real tsw_n_stddev_rel = 0.02 from [0:0.3]; //0.02;
parameter real tsw_p_stddev_rel = 0.02 from [0:0.3]; //0.02;
parameter real Vtn_stddev_rel = 0.001 from [0:0.3]; //0.02;
parameter real Vtp_stddev_rel = 0.001 from [0:0.3]; //0.02;

//forming parameters
parameter integer form = 0 from [0:1];
//Memristor model include forming : switch : can take values 0 and 1
parameter real Rf      = 1M;
parameter real Vf      = 2.1;
parameter real Tf      = 2m;
parameter real Icomp = 56e-6;

real LRS_Aging_Rate;    //Aging rate of LRS
real LRS_nom_spr;       //this also should be a parameter

// local variables

```

```

real delR;                // resistance spread (HRS - LRS)
real delt;               // simulation time step
real HRS_rnd;            // ith HRS
real LRS_rnd;            // ith LRS
real RFS_rnd;
real Rm;                 // memristance
real Rm_tmp;             // temp memristance variable
real Rm_mid;             // mid point between LRS/HRS
//real switch_val;       // denote state of memristor switching
real time_last;          // previous simularion time reading
real tsw_p_rnd;
real tsw_n_rnd;
real Vtn_rnd;            // negative th voltage for ith switch
real Vtp_rnd;            // positive th voltage for ith switch
real Vwr;                // input voltage

real HRS_nom;            // nominal high res, trends down w/ time
real LRS_nom;
real tsw_p_nom;
real tsw_n_nom;
real Vtn_nom;
real Vtp_nom;

real HRS_spread;
real LRS_spread;
real tsw_p_spread;
real tsw_n_spread;
real Vtp_spread;
real Vtn_spread;

```

```

real HRS_min;      // impose a lower bound on the
                    //HRS due to the Aging rate
real HRS_after_Aging;    // value of HRS after Aging phenomena
real HRS_max;
real LRS_max;
real LRS_min;
real LRS_after_Aging;    //value of LRS after Aging phenomena

//temperature variables
real temp_room;
real HRS_temp_change_rate;
real LRS_temp_change_rate;
real HRS_spread_temp_change_rate;
real Vt_temp_change_rate;
//
real HRS_spread_rate_temp;
real temp_curr;
real deltemp;
// forming variables
real f;
real tmp;
real c;
real x, y;

// reset fail variables
real rst_fail_flag;

integer rnd;
// temp storage for random number seeds

```

```

analog begin
  @ ( initial_step or initial_step("dc") ) begin
    delt          = 0;
    time_last     = 0;
    rnd           = $random;      // generate random number for seed
    HRS_nom       = HRS;
    LRS_nom       = LRS;
    HRS_rnd       = HRS_nom;
    LRS_rnd       = LRS_nom;
    RFS_rnd       = R_rst_fail;
    tsw_p_rnd     = tsw_p;
    tsw_n_rnd     = tsw_n;
    f = form;
    tmp = 0;
    if (f == 0)
      Rm = Rinit;
    else
      Rm = Rf;
    delR          = HRS_rnd - LRS_rnd;
    Rm_mid = LRS_rnd + delR/2;
    LRS_Aging_Rate = 0;    //Aging rate of LRS
    LRS_nom_spr = 0.001;   //this also should be a parameter
    HRS_spread    = abs(HRS * HRS_stddev_rel);
    LRS_spread    = abs(LRS * LRS_stddev_rel);
    tsw_p_spread  = abs(tsw_p * tswp_stddev_rel);
    tsw_n_spread  = abs(tsw_n * tswn_stddev_rel);
    Vtp_spread    = abs(Vtp * Vtp_stddev_rel);
    Vtn_spread    = abs(Vtn * Vtn_stddev_rel);
    HRS_min = HRS; //25e3;//HRS; //arbitrary choice for minimum HRS
    HRS_max = HRS; //2e5;//HRS;
  end
end

```

```

LRS_max = LRS; //1e4;
LRS_min = LRS; //2e3;//LRS;

//LRS_Aging_Rate = 0.25e6;

//Aging rate of LRS...should be slower then HRS aging
//initialize the parameters
temp_room = 300.15;
HRS_temp_change_rate = 0;

// negative temperature coefficient of HRS, slower than LRS
LRS_temp_change_rate = 0;
// lies between temp-co of Ti & W, not a bad estimate
HRS_spread_temp_change_rate = 0;
//HRS variance increases with increasing temperature
Vt_temp_change_rate = 0;
//threshold voltage decreases with increasing temperature
    temp_curr = $temperature;
Vtp_nom = Vtp;
    Vtn_nom = Vtn;
    Vtp_rnd      = Vtp_nom;
    Vtn_rnd      = Vtn_nom;
    HRS_spread_rate_temp = HRS_stddev_rel;
// nominal threshold voltage changes for different temperature
//Later in this code, generate Vtp_rnd, Vtn_rnd
Vtp_nom = Vtp_nom + Vtp_nom
            * Vt_temp_change_rate
            * ($temperature - temp_room);
Vtn_nom = Vtn_nom + Vtn_nom
            * Vt_temp_change_rate

```

```

        * ($temperature - temp_room);

Vtp_rnd = Vtp_nom;
Vtn_rnd = Vtn_nom;
////////change in nominal value of HRS and LRS
HRS_nom = HRS * (1
                + HRS_temp_change_rate
                * ($temperature - temp_room));

LRS_nom = LRS * (1
                + LRS_temp_change_rate
                * ($temperature - temp_room));

////////change in nominal value of HRS spread rate
HRS_spread_rate_temp = HRS_spread_rate_temp
                      + HRS_spread_rate_temp
                      * HRS_spread_temp_change_rate
                      * ($temperature-temp_room);

HRS_spread = abs(HRS * HRS_spread_rate_temp);
rst_fail_flag = 0;
end

delt      = $abstime - time_last;
time_last = $abstime;
Vwr       = V(p,n);

// for temperature changes that occurs within a simulation
deltemp   = $temperature - temp_curr;
temp_curr = $temperature;

Vtp_rnd   = Vtp_rnd + Vtp_rnd * Vt_temp_change_rate * deltemp;
Vtn_rnd   = Vtn_rnd + Vtn_rnd * Vt_temp_change_rate * deltemp;
HRS_spread_rate_temp = HRS_spread_rate_temp

```

```

        + HRS_spread_rate_temp
        * HRS_spread_temp_change_rate
        * deltemp;

HRS_spread = abs(HRS * HRS_spread_rate_temp);
// HRS_nom    = (HRS_nom + HRS_nom * HRS_temp_change_rate * deltemp);
// LRS_nom    = (LRS_nom + LRS_nom * LRS_temp_change_rate * deltemp);
HRS_nom    = (Vwr <= Vtn_rnd) ? ((-2382247.51227144 * Vwr * Vwr * Vwr)
                                + (-13207439.75417665 * Vwr * Vwr)
                                + (-23665132.52825601 * Vwr)
                                + (-13505746.41170909)) : HRS_nom;
LRS_nom    = (Vwr >= 0.9*Vtp_rnd) ? ((2.44650748e+11 * I(p,n) * I(p,n))
                                - (1.32395756e+08 * I(p,n))
                                + 1.94736040e+04) : LRS_nom;

LRS_rnd    = $rdist_normal(rnd,LRS_nom,LRS_spread);
HRS_rnd    = $rdist_normal(rnd,HRS_nom,HRS_spread);

// Effect of current exceeding the compliance
if (I(p,n) > Icomp) begin
LRS_nom = LRS_nom - (0.7e3/35e-6)*(I(p,n)-Icomp);
if (LRS_nom < 0.1 * LRS) begin
LRS_nom = 0.1 * LRS;
end
end

Rm          = Rm;
delR        = HRS_nom - LRS_nom;

////////// Forming phase
if (f == 1) begin
if (Vwr < Vf) begin

```

```

c = 0;
tmp = 0;
end
else if (Vwr >= Vf)begin
c = c+ delt ;
if (c >= Tf)begin
    tmp = 1;
end
end
if (Vwr >= 0  && Rm != LRS_nom && tmp == 1) begin //Vtp
    Rm_tmp = Rm -  delt* (delR/tsw_p);
if (Rm_tmp <= LRS_nom) begin
Rm_tmp      = LRS_nom;
f      = 0;          //formed
end
    end else begin
Rm_tmp = Rm;
end
    end

//////////either forming done or no forming to be included for the model
if (f == 0) begin //1
    if (Vwr >= Vtp_rnd && Rm > LRS_rnd) begin //2
Rm_tmp = Rm - delt * C_LRS
        * (delR/tsw_p_rnd)
        * pow (((Vwr-Vtp_rnd)/Vtp_rnd),P_LRS)
        / (1+exp((theta_LRS * LRS_rnd - Rm)
            /(delR)/beta_LRS));
if (Rm_tmp <= LRS_rnd) begin //3

```

```

rnd = $random;          // generate random seed
//////////impose a upper bound of LRS aging factor
LRS_after_Aging = LRS_nom + LRS_Aging_Rate*time_last;
//$display("age = %r\n",LRS_after_Aging);
if(LRS_after_Aging>LRS_max)
LRS_after_Aging = LRS_max;
        // update trending upward (Aging) nominal low resistance value
x = I(p,n);
LRS_nom    = (2.44650748e+11 * x * x)
              - (1.32395756e+08 * x)
              + 1.94736040e+04;
              // fitted characteristic
        //$rdist_normal(rnd,LRS_after_Aging,LRS_after_Aging*LRS_nom_spr);
LRS_spread = abs(LRS_nom * LRS_stddev_rel);
if (LRS_nom >= LRS_max)
        LRS_nom = LRS_max;
if(LRS_nom <= LRS_min)
LRS_nom = LRS_min;
        // regenerates LRS, Vtn and tsw_n ONLY when switching complete
LRS_rnd      = $rdist_normal(rnd,LRS_nom,LRS_spread);
Vtn_rnd      = $rdist_normal(rnd,Vtn,Vtn_spread);
tsw_n_rnd    = $rdist_normal(rnd,tsw_n,tsw_n_spread);
// update parameters, set switch_val to stop switching
delR         = HRS_rnd - LRS_rnd;
Rm_mid = LRS_rnd + delR/2;
Rm_tmp      = LRS_rnd;
        end //3
        end /*2*/ else if (Vwr <= Vtn_rnd && Rm < HRS_rnd) begin //4
                if (Vwr > V_rst_fail) begin //5
Rm_tmp = Rm + delt *C_HRS

```

```

        * (delR/tsw_n_rnd)
        * pow (((Vwr-Vtn_rnd)/Vtn_rnd),P_HRS)
        / (1 + exp((Rm - theta_HRS * HRS_rnd)
        /(delR)/beta_HRS));

if (Rm_tmp >= HRS_rnd) begin //6
rnd      = $random;          // generate random seed
//impose a lower bound of HRS aging factor
HRS_after_Aging = HRS_nom - HRS_rate*time_last;
if(HRS_after_Aging<HRS_min)
HRS_after_Aging = HRS_min;
        // update trending downward (Aging)
        //nominal high resistance value

y = V(p,n);
HRS_nom    = (-2382247.51227144 * y * y * y)
            + (-13207439.75417665 * y * y)
            + (-23665132.52825601 * y)
            + (-13505746.41170909);
            //fitted 3rd order polynomial

HRS_spread = abs(HRS_nom * HRS_spread_rate_temp);
if (HRS_nom <= HRS_min)
HRS_nom = HRS_min;
if (HRS_nom >= HRS_max)
HRS_nom = HRS_max;

        // regenerate HRS, Vtp and tsw_p ONLY when switching complete
HRS_rnd      = $rdist_normal(rnd,HRS_nom,HRS_spread);
Vtn_rnd      = $rdist_normal(rnd,Vtn,Vtn_spread);
tsw_n_rnd    = $rdist_normal(rnd,tsw_n,tsw_n_spread);
// update parameters, set switch_val to stop switching
delR         = HRS_rnd - LRS_rnd;
Rm_mid = LRS_rnd + delR/2;

```

```

Rm_tmp          = HRS_rnd;
    end //6
    end /*5*/ else begin //7
        rst_fail_flag = 1; end //7
    end /*4*/ else if ((Vwr <= V_rst_fail && Rm > RFS_rnd)
        || rst_fail_flag) begin //4
Rm_tmp = Rm + delt *C_LRS
        * (delR/tsw_n_rnd)
        * pow (((Vwr-V_rst_fail)/V_rst_fail),P_LRS)
        / (1 + exp((Rm - theta_LRS * RFS_rnd)
        /(delR)/beta_LRS));
if (Rm_tmp >= RFS_rnd) begin //6
rnd      = $random;
        // generate random seed
RFS_rnd      = $rdist_normal(rnd,R_rst_fail,abs(R_rst_fail * 0.001);
Vtp_rnd      = $rdist_normal(rnd,Vtp,Vtp_spread);
tsw_p_rnd    = $rdist_normal(rnd,tsw_p,tsw_p_spread);
// update parameters, set switch_val to stop switching
delR          = HRS_rnd - RFS_rnd;
Rm_mid        = RFS_rnd + delR/2;
Rm_tmp        = RFS_rnd;
    end //6
    end //4
    else begin //8
        Rm_tmp = Rm;
    end //8
end //1

Rm = Rm_tmp;

```

```
    I(p,n) <+ Vwr / Rm;  
end          // end analog  
  
endmodule
```

Appendix C

System Verilog Code for Reservoir Computing Design

A Network-on-Chip: spics_net.sv

```

/*****
Project name:
    Neuropipe
File name:
    spics_net.sv
Purpose: Implementation of circuit-switched network
Date:      Last modified
           05/19/2022
Author:
    Manu Rathore
Contact:
    E-mail: mrathore@utk.edu
*****/
```

```

'timescale 1ns/1ns

// N = total number of inputs
// n = N inputs grouped into collection of "n" signals
// n_box = N/n : total number of instances in stage 1
// k = total number of instances in stage 2 (k_box)
// M = total number of outputs
// m = M outputs grouped into collection of "m" signals
// m_box = M/m : total number of instances in stage 3

module spics_net #(parameter N=32,
                    M=32,
                    n=8,
                    m=8,
                    k=n+m-1,
                    n_box = N/n,
                    m_box = M/m)
    (input [N-1:0] I,
     input [((n_box*k*($clog2(n)))+(k*m_box*($clog2(n_box)))
            +(m_box*m*($clog2(k))))-1:0] S,
     output reg [M-1:0] Output);

    // number of blocks
    * select width per block (based on number of inputs and outputs
    = number of outputs * log base 2 (number of inputs))
    parameter sel_width = ((n_box*k*($clog2(n)))
                          +(k*m_box*($clog2(n_box)))
                          +(m_box*m*($clog2(k))));
    parameter sel_width_stg1 = ((n_box*k*($clog2(n))));
    parameter sel_width_stg2 = (k*m_box*($clog2(n_box)));

```

```

parameter sel_width_stg3 = ((m_box*m*($clog2(k))));

wire [sel_width_stg1-1 : 0] sel_stg1
    = S[sel_width-1 : sel_width
        -(n_box*k*($clog2(n)))] ;

wire [sel_width_stg2-1 : 0] sel_stg2
    = S[(sel_width-(n_box*k*($clog2(n)))-1)
        : sel_width -(n_box*k*($clog2(n))
            -(k*m_box*($clog2(n_box))))] ;

wire [sel_width_stg3-1 : 0] sel_stg3
    = S[(sel_width
        -(n_box*k*($clog2(n)))
        -(k*m_box*($clog2(n_box)))-1) : 0] ;

wire [n_box*k-1:0] stg1_2;
wire [k*m_box-1:0] stg2_3;

genvar gi,gj,gk;

generate
    for (gi = 0; gi < n_box; gi = gi + 1) begin : STAGE_1

        switch #(.in(n),
            .out(k))

        Stg1_n (.I(I[(n*(gi+1))-1 : (n*(gi+1))-n]),
            .S(sel_stg1[((sel_width_stg1/n_box)*(gi+1))-1

```

```

        : ((sel_width_stg1/n_box)*(gi+1))
        -(sel_width_stg1/n_box)]),
    .Output(stg1_2[(k*(gi+1))-1 : (k*(gi+1))-k]));

    end

endgenerate

generate
    for (gj = 0; gj < k; gj = gj + 1) begin : STAGE_2
        switch #(.in(n_box),
            .out(m_box))
        Stg1_n (.I(stg1_2[(n_box*(gj+1))-1 : (n_box*(gj+1))-n_box]),
            .S(sel_stg2[((sel_width_stg2/k)*(gj+1))-1
                : ((sel_width_stg2/k)*(gj+1))
                -(sel_width_stg2/k)]),
            .Output(stg2_3[(m_box*(gj+1))-1
                : (m_box*(gj+1))-m_box]));

    end

endgenerate

generate
    for (gk = 0; gk < m_box; gk = gk + 1) begin : STAGE_3
        switch #(.in(k),
            .out(m))
        Stg1_n (.I(stg2_3[(k*(gk+1))-1 : (k*(gk+1))-k]),
            .S(sel_stg3[((sel_width_stg3/m_box)*(gk+1))-1
                : ((sel_width_stg3/m_box)*(gk+1))
                -(sel_width_stg3/m_box)]),
            .Output(Output[(m*(gk+1))-1 : (m*(gk+1))-m]));

    end

endgenerate

```

```
endmodule
```

B Network-on-Chip: switch.sv

```

/*****
Project name:
    Neuropipe
File name:
    switch.sv
Purpose:
    nxm switch module for implementation of spics_net network
Date:
    05/23/2023
Author:
    Manu Rathore
Contact:
    E-mail: mrathore@utk.edu
*****/
```

```

`timescale 1ns/1ns
module switch #(parameter in=2,
                    out=1)
    (input [in-1:0] I,
     input [(out*($clog2(in)))-1:0] S,
     output reg [out-1:0] Output);
```

```

parameter n = $clog2(in);
reg [32:0] sel [out-1:0];
wire [n-1:0] select_wire [out-1:0];

genvar i;
generate
    for (i=0;i<out;i++) begin
        assign sel[i] = i*n;
        assign select_wire[i] = S[(sel[i])+:n];
        param_mux #(.in(in),
                    .sel(n))
        MUX_INST ( .A(I),
                  .Sel(S[(sel[i])+:n]),
                  .Y(Output[i]));
    end
endgenerate

endmodule

module param_mux #(parameter in=2,
                  sel=1)
    (input [in-1:0] A,
     input [sel-1:0] Sel,
     output reg Y);

    assign Y = A[Sel];

endmodule

```

C Reservoir SRNN: reservoir.vams

```
/* -----  
  
* File Name : reservoir.vams  
  
* Purpose :  
  
* Creation Date : 23-11-2023  
  
* Last Modified : 22-02-2024  
  
* Created By : Manu Rathore  
  
-----*/  
  
'include "disciplines.vams"  
'include "constants.vams"  
  
'timescale 1ns / 1ps  
  
//=====//  
  
//=====//  
// Module Definition //  
//=====//  
  
module reservoir  
    // Optional Parameter Declaration  
    #(parameter number_of_neurons=25,  
        parameter number_in_neurons=10,
```

```

parameter number_out_neurons=25,
parameter G_PS_PER_CORE = 3,
parameter conn_density=0.1,
parameter threshold=1,
parameter k_neighbors=4,
parameter probability_random_conn = 0.03
)

// Port Declaration
(
    // Inputs //////////////////////////////////////

    // Global Clock and Reset
    input wire clk,
    input wire rst_n,
    input wire reservoir_run_en,

    input reservoir_fire_in,

    // Inouts////////////////////////////////////////
    inout Vth,
    inout Vncm,
    inout Vst,
    inout Vrst,
    inout VDD3V3,
    inout VDD0V6,

    // Outputs //////////////////////////////////////

```

```

        output [number_out_neurons-1:0] reservoir_fire_out

);

//=====//
// Variable Declaration //
//=====//

integer i, j, k;
genvar gi, gj;

//=====//
// Signal Declaration //
//=====//
////////////////////////////////////
wire [(G_PS_PER_CORE*number_of_neurons
      +number_out_neurons)-1 : 0] spics_net_fire_out;
// One input from interface, all others internal
electrical reservoir_fire_in;
wire [number_of_neurons-1:0] neuron_fire_out;
parameter integer n_box=(number_of_neurons+number_in_neurons)/8;
parameter integer m_box=(G_PS_PER_CORE*number_of_neurons
      +number_out_neurons)/8;
parameter integer config_width
      =((n_box*8
        *($clog2(number_of_neurons+number_in_neurons)))
        +(8*m_box*($clog2(n_box)))
        +(m_box

```

```

                                *(G_PS_PER_CORE*number_of_neurons
                                +number_out_neurons)
                                *($clog2(8))));

reg [config_width-1:0] nw_config;
//=====//
// Component Instantiation //
//=====//
// Generate neurons //////////////////////////////////////

// neuron Core Blocks //////////////////////////////////////

generate
    for (gi = 0; gi < number_of_neurons; gi = gi + 1)
    begin : NCORE_INPUT_ARRAY_GEN
        Core_16x1 NCORE_INST (
            .Vout(neuron_fire_out[gi]),
            .read(spics_net_fire_out[(G_PS_PER_CORE
                *(gi+1)+number_out_neurons)-1
                : (G_PS_PER_CORE
                    *(gi+1)+number_out_neurons)
                    -G_PS_PER_CORE]),
            .CLK(clk),
            .En(reservoir_run_en),
            .EnB(!reservoir_run_en),
            .Res_Neu(!rst_n),
            .Vth(Vth),
            .Vncm(Vncm),
            .Vrst(Vrst),
            .Vst(Vst),
            .form(form),

```

```

        .DACV(DACV),
        .col_set(col_set),
        .col_reset(col_reset),
        .col_form(col_forming),
        .set(set),
        .row_reset(row_reset));

    end
endgenerate

// Network connections
spics_net #( .N(number_of_neurons),
             .M(G_PS_PER_CORE*number_of_neurons+number_out_neurons),
             .n(8),
             .m(8),
             .k(8))
NETWORK_INST ( .I({neuron_fire_out,
                  {number_in_neurons{reservoir_fire_in}}}),
               .S(nw_config),
               .Output(spics_net_fire_out));

assign reservoir_fire_out = spics_net_fire_out[number_out_neurons-1:0];

endmodule

```

D Output Readout Layer: Output_layer.sv

```

/* -----

* File Name : Output_layer.sv

```

* Purpose :

* Creation Date : 27-11-2023

* Last Modified : 12-03-2024

* Created By : Manu Rathore

-----*/

'timescale 1ns / 1ps

//=====//

// Module Definition //

//=====//

module Output_layer

 // Optional Parameter Declaration

 #(parameter number_of_inputs = 32,

 parameter symbol_width = 2,

 parameter error_width = 10,

 parameter weight_width = 8,

 parameter default_width = 8,

 parameter mu = 8'b00000001,

 // mu parameter for controlling weight update rate

 parameter weight_intial = 8'b00000000

)

 // Port Declaration

 (

 // Inputs //////////////////////////////////////

```

// Global Clock and Reset
input wire clk,
input wire rst_n, // active low
// Run and train enable
input wire reservoir_run_en,
input wire reservoir_train_en,
input wire [number_of_inputs-1:0] reservoir_fire_out,
// Expected output for training session
input [symbol_width-1:0] expected_symbol_out,
input [symbol_width-1:0] sampled_symbol_out,
input [error_width-1:0] error_sig_sampler,
// Inouts////////////////////////////////////
//inout wire [symbol_width-1:0] symbol_out

// Outputs //////////////////////////////////////
output reg [error_width-1:0] error_sampled,
output reg [symbol_width-1:0] symbol_out

);

//=====//
// Variable Declaration //
//=====//

integer i, j, k, weight;

//=====//
// Signal Declaration //

```

```

//=====//
//wire error_sig_sampler;
reg [symbol_width-1:0] calculated_symbol_out;
reg [default_width-1:0] calculated_ip_error_offset;
reg [number_of_inputs-1:0] reservoir_fire_out_sampled;
reg [weight_width-1:0] weight_coeff_next [number_of_inputs-1:0];
reg [weight_width-1:0] weight_coeff [number_of_inputs-1:0];
reg [weight_width-1:0] weight_coeff_d1 [number_of_inputs-1:0];
reg [weight_width:0] monitor_weight [number_of_inputs-1:0];
wire [error_width-1:0] error ;

// Sample reservoir output
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        reservoir_fire_out_sampled <= 'd0;
    end else if (reservoir_train_en) begin
        reservoir_fire_out_sampled <= reservoir_fire_out;
    end
end

always @(*) begin
    error[error_width-3:0] = error_sig_sampler
                        - calculated_ip_error_offset;
    error[error_width-1:error_width-2] = sampled_symbol_out
                        - expected_symbol_out;
end

// Weight update
always @(error) begin

```

```

if (rst_n && reservoir_train_en
    && (expected_symbol_out != sampled_symbol_out))
begin
    if (error < (2**error_width-1)/4) begin
        for (i = 0; i < number_of_inputs; i = i + 1) begin
            monitor_weight[i] = weight_coeff_next[i]
                + (reservoir_fire_out[i]
                    ? ((error>>1)) : 'd0);
            weight_coeff_next[i] = monitor_weight[i][weight_width]
                ? (2**weight_width-1) :
                    monitor_weight[i][weight_width-1:0];

            // mu * error * reservoir_fire_out[i]
        end
    end else begin
        for (i = 0; i < number_of_inputs; i = i + 1) begin
            monitor_weight[i] = weight_coeff_next[i]
                + (reservoir_fire_out[i]
                    ? ((error>>2) ) : 'd0);
            weight_coeff_next[i] = monitor_weight[i][weight_width]
                ? (2**weight_width-1)
                    : monitor_weight[i][weight_width-1:0];
        end
    end
end

end else if (rst_n & reservoir_run_en) begin
    monitor_weight <= monitor_weight;
    for (i = 0; i < number_of_inputs; i = i + 1) begin
        weight_coeff_next[i] = weight_coeff_next[i];
    end
end

```

```

end else if (!rst_n) begin
    monitor_weight <= monitor_weight;
    for (i = 0; i < number_of_inputs; i = i + 1) begin
        weight_coeff_next[i] = 'h00;
    end
end
end

always @(negedge clk or negedge rst_n) begin
    if (!rst_n) begin
        for (i = 0; i < number_of_inputs; i = i + 1)
            weight_coeff_next[i] <= 'd0;
        for (i = 0; i < number_of_inputs; i = i + 1)
            weight_coeff[i] <= 'd0;
        for (i = 0; i < number_of_inputs; i = i + 1)
            weight_coeff_d1[i] <= 'd0;
        for (i = 0; i < number_of_inputs; i = i + 1)
            monitor_weight[i] <= 'd0;
    end else begin
        for (i = 0; i < number_of_inputs; i = i + 1)
            weight_coeff[i] <= weight_coeff_next[i];
        for (i = 0; i < number_of_inputs; i = i + 1)
            weight_coeff_d1[i] <= weight_coeff[i];
    end
end

// Symbol out from reservoir_fire_out
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        symbol_out = 'd0;
    end
end

```

```

        calculated_ip_error_offset = 'd0;
    end else begin
        calculated_ip_error_offset = 'd0;
        for (i = 0; i < number_of_inputs; i = i + 1) begin
            calculated_ip_error_offset = calculated_ip_error_offset
                + (reservoir_fire_out[i]
                    ? weight_coeff[i] : 'd0);
        end
    end
end
end
end

endmodule

```

E Top-level Module: top_reservoir.sv

```

/* -----

* File Name : top_reservoir.sv

* Purpose :

* Creation Date : 23-01-2024

* Last Modified : 15-03-2024

```

* Created By : Manu Rathore

-----*/

`timescale 1ns / 1ps

//=====//

// Imports //

//=====//

/*

`include "./reservoir_pkg.sv"

`include "./ncore_pkg.sv"

import reservoir_pkg::*;

import ncore_pkg::*;

*/

//=====//

//=====//

// Module Definition //

//=====//

module top_reservoir

 // Optional Parameter Declaration

 #(parameter number_of_neurons=32,

 parameter number_in_neurons=10,

 parameter number_out_neurons=32,

 parameter conn_density=0.1,

 parameter threshold=1,

 parameter symbol_width = 2,

```

parameter error_width = 2,
parameter weight_width = 2,
parameter mu = 8'b00000001,
// mu parameter for controlling weight update rate
parameter weight_intial = 8'b00000000
)
// Port Declaration
(
    // Inputs //////////////////////////////////////
    input wire signal_in,
    // Global Clock and Reset
    input wire clk,
    input wire rst_n,
    input wire reservoir_run_en,
    input wire reservoir_train_en,
    // Expected output for training session
    input [symbol_width-1:0] expected_symbol_out,

    // Inouts////////////////////////////////////
    inout Vth,
    inout Vncm,
    inout Vst,
    inout Vrst,
    inout VDD3V3,
    inout VDD0V6,

    // Outputs //////////////////////////////////////
    output wire [error_width-1:0] error_sampled,

```

```

        output wire [symbol_width-1:0] symbol_out,
        output wire [symbol_width-1:0] sampled_symbol_out

    );

//=====//
// Signal Declaration //
//=====//
    wire [number_out_neurons-1:0] reservoir_fire_out;
    wire [11:0] calculated_ip_error_offset;
    //wreal error_sig;
//=====//
// Module Instantiation //
//=====//

    reservoir RESERVOIR_INST (.clk(clk),
        .rst_n(rst_n), // active low
        .reservoir_run_en(reservoir_run_en),
        .Vth(Vth),
        .Vncm(Vncm),
        .Vst(Vst),
        .Vrst(Vrst),
        .VDD3V3(VDD3V3),
        .VDD0V6(VDD0V6),
        .reservoir_fire_in(signal_in),
        .reservoir_fire_out(reservoir_fire_out)
    );

```

```

Output_layer #(.number_of_inputs(number_out_neurons))
OUTPUT_LAYER_INST(.clk(clk),
    .rst_n(rst_n), // active low
    .reservoir_run_en(reservoir_run_en),
    .reservoir_train_en(reservoir_train_en),
    .reservoir_fire_out(reservoir_fire_out),
    .expected_symbol_out(expected_symbol_out),
    .sampled_symbol_out(sampled_symbol_out),
    .error_sampled(error_sampled),
    .error_sig_sampler(error_sig),
    .symbol_out(symbol_out),
    .calculated_ip_error_offset(calculated_ip_error_offset)
);

```

```

sampler #(.symbol_width(2)) SAMPLE_INST (.clk(clk),
    .rst_n(rst_n), // active low
    .reservoir_run_en(reservoir_run_en),
    .reservoir_train_en(reservoir_train_en),
    .reservoir_fire_in(signal_in),
    .expected_symbol_out(expected_symbol_out),
    .sampled_symbol_out(sampled_symbol_out),
    .error_sig(error_sig),
    .calculated_ip_error_offset(calculated_ip_error_offset)
);

```

```

endmodule: top_reservoir

```

Vita

Manu Rathore is a dedicated researcher with a profound passion for Integrated Circuit Design, and a keen interest in Electrical and Computer Engineering. With six years of semiconductor design industry experience, including her role as a Mixed-Signal Design Engineer at NXP Semiconductors in Bangalore, India, Manu's journey has been marked by a commitment to advancing technology.

Manu Rathore holds a Master's degree in Communication Engineering from BITS, Pilani, India, and a Bachelor's degree in Technology from Banasthali University, India, which have collectively provided a solid foundation for her engineering career and academic pursuits. With this educational background, along with six years of industry experience, Manu brings a wealth of knowledge and expertise to her doctoral research at the University of Tennessee with the TENNLab Neuromorphic Computing Research group. Her interdisciplinary approach and dedication to advancing technology underscore her commitment to making significant contributions to the field of neuromorphic computing.

Manu Rathore's doctoral research at the University of Tennessee, with the TENNLab Neuromorphic Computing Research group, delves into three crucial areas of neuromorphic computing. Firstly, she explores brain-inspired connectivity for Neuroprocessor architectures, aiming to replicate the intricate neural connections found in the human brain. Secondly, Manu investigates nonlinear signal processing using the dynamical properties of reservoir computing, seeking to harness the power of complex dynamics for efficient information processing. Lastly, her research encompasses device modeling in neuromorphic Integrated Circuits, aiming to develop accurate models to better understand and optimize the behavior of these cutting-edge circuits. Through her multifaceted approach, Manu aims to advance the understanding and capabilities of neuromorphic computing, paving the way for innovative applications and technologies.

With her relentless curiosity and dedication, Manu Rathore is poised to make significant contributions to the advancement of technology and the scientific community as a whole.