HIGH-SPEED VISION PROCESSING AND COMPUTATION USING MEMRISTOR- BASED CELLULAR NEURAL NETWORKS

HIGH-SPEED VISION PROCESSING AND COMPUTATION USING MEMRISTOR-BASED CELLULAR NEURAL NETWORKS

A Dissertation Presented

by

VIGNESH RAVICHANDRAN

Submitted to the Graduate School of the

University of Massachusetts Amherst in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2025

Electrical and Computer Engineering

HIGH-SPEED VISION PROCESSING AND COMPUTATION USING MEMRISTOR-BASED CELLULAR NEURAL NETWORKS

A Dissertation Presented

By

VIGNESH RAVICHANDRAN

Approved as to style and content by:

_____

Qiangfei Xia, Chair

_____

Marco Duarte, Member

_____

J. Joshua Yang, Member

_____

Russell Tessier, Department Head

Electrical and Computer Engineering

DEDICATION

To my parents.

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartfelt gratitude to Professor Qiangfei Xia for being an encouraging PI and creating a supportive and close-knit lab space. His passion for research and optimism have been a great source of inspiration for me throughout my journey through grad school. He provided a platform from which I was able to dive into and explore new cutting-edge ideas and research directions. He supported me even when some of my ideas were more farfetched and untraditional. Moreover, he has always been very approachable and has been a great advisor, mentor, and friend. I would like to thank him for providing me with this opportunity and opening many more. Through his constant guidance and support I've seen myself grow not only as a researcher, but also as a person these past several years.

I would like to thank Professor Joshua J. Yang for initially introducing me to the field of neuromorphic computing and providing me the opportunity to intern with his research group during the summer of 2017. As it turned out, that summer has been a very influential period in my life; That research internship is also what sparked my initial interest in pursuing grad school in this field. Professor Yang's wisdom and guidance are much appreciated as are his ideas and collaboration during the early years of my PhD.

I would also like to thank Professor Marco Duarte for serving on my dissertation committee and providing valuable insight and feedback. He was also a great advisor for my undergraduate Senior Design Project "NeuroArm". His continued support and guidance are greatly appreciated.

Thank you to Dr. Thomas Collins and Dr. Jonathon Comeau at BAE Systems. Working internships under their guidance, I was able to further hone and expand my technical knowledge

and skillset outside of the lab. I am grateful for all that they have taught me and for the opportunities that they have provided me.

Thank you to all the lab mates I have had the pleasure of working with throughout the course of graduate school: Dr. Can Li, Dr. Hao Jiang, Dr. Shiva Asapu, Dr. Navnidhi Kumar Upadhyay, Dr. Rivu Midya, Dr. Zhongrui Wang, Dr. Mingyi Rao, Dr. Fatemeh Kiani, Dr. Puming Fang, Yi Huang, Ali Abdel-Maksoud, Alireza Jaberi Rad, Dr. Yunzhi Ling, Koushik Kumar, Wuyu Zhao, Yuhan Gao, Suyeon Jang, Zhizheng Wang, and Dr. Zheshun Xiong. I would also like to thank each of the undergraduate interns I have had the pleasure to work with and their collective contributions: Tina Maurer, Bryce Flannery, Remy Ai, Nia Heermance, Tergel Molom-Ochir, and Joshua Tackie, along with our high-school intern Alex Guo. All of these folks were always there for me to talk to and bounce ideas off of and helped keep me sane throughout the whole process. Without such a supportive lab group, I don't believe I could have made it through my Ph.D.

I would also like to thank Francis Caron and Wouter Schievink at UMass who have both been instrumental to me being able to conduct my research in the support and technical services they provided.

Lastly, I am grateful to all my family and close friends who have seen me go through hard times but have always supported me throughout my PhD journey: Magesh Ravichandran, Kousalya Ravichandran, Nithyanandan Ravichandran along with Ann Liptak, Joan Liptak, Dan-Michael Tiamzon, and Allison Chen. They have been there for me every single day, during the highest of highs and lowest of lows, and I am forever grateful to have these people in my life. This work would not have been possible without everyone's support. Thank you.

"From 3 to 30" –R.K.

ABSTRACT

HIGH-SPEED VISION PROCESSING AND COMPUTATION USING MEMRISTOR-BASED CELLULAR NEURAL NETWORKS

FEBRUARY 2025

VIGNESH RAVICHANDRAN

B.S., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS, AMHERST

Advisor: Professor Qiangfei Xia

Neuromorphic computing algorithms and brain-inspired architectures hold promise to move beyond the current limitations of contemporary von Neumann architectures that are hindering current Artificial Intelligence (AI) implementations in terms of processing speeds and power consumption. The Cellular Neural Network (CeNN) is one such bio-inspired architecture that allows for massively parallel, analog computing schemes for high-speed image processing. The network is composed of simple, repeating processing elements referred to as cells and takes advantage of reprogrammable local connectivity for direct neighborhood interactions as well as propagating global dynamics to achieve a wide variety of applications.

The first section will introduce the network fundamentals and underlying principles of network dynamics. Based off these fundamentals, a CeNN GUI software package with a digital twin simulator is written in python. The simulator is designed and tested for a wide variety of applications based off suggested hardware parameters. Simulated image and video processing approaching 100,000 frames per second are shown. Noisy image and video processing is also discussed, simulating the consequences of mismatch and variance in a fabricated network. Computational simulations for solving partial differential equations further demonstrate network

robustness and capabilities. These digital twin simulations inform us on network performance and provide insight into the potential behavior of fabricated hardware networks.

The second section focuses on the design and measurement of a prototype 5×5 hardware network based on pure CMOS using the TSMC 65nm process node. Edge detection and vertical/horizontal line detection algorithms are run demonstrating processing speeds greater than 20,000 frames per second. Performance merits are presented and compared to prior state of the art implementations.

Finally, emerging Ta/HfO_x drift memristor devices are introduced and the benefits of using such devices for neural networks (NN) are discussed. These two terminal devices are implemented into a "bridge synapse" architecture for efficient weight storage and intrinsic weight multiplication. A board-level CeNN prototype is built around these memristive devices and used to demonstrate horizontal line detection.

This dissertation discusses and contributes to the development of massively parallel, low-power, analog NN hardware with a focus on CeNNs. The CeNN digital twin provides a means for easy network modelling in software and proof of concept. The fabrication of hardware and testing of a pure CMOS IC as well as the board-level memristor-based implementation further explore and demonstrate the feasibility of parallel, reprogrammable network architectures for high-speed, analog processing.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Background

Since the inception of the modern computer, there has been a continued effort to realize a true thinking machine with capabilities akin to the biological human brain. Significant technological advancements over the past decades have brought us increasingly closer to achieving this lofty goal of a true Artificial Generalized Intelligence (AGI). Although we may not be quite at that level yet, modern Artificial Intelligence (AI) systems are becoming more capable and increasingly commonplace in today's data driven world. Impressive capabilities of AI based on contemporary technologies have already been demonstrated in several domains showcasing a wide variety of practical applications, including natural language processing, image generation and classification, protein folding, and autonomous decision making for applications such as self-driving cars [1], [2], [3], [4], [5]. With the boom in digital information from modern devices that are always online, AI is expected to continue improving and integrating further into our everyday lives utilizing ever growing datasets.

While AI has taken significant strides forward in recent years, it has become clear that this growth is unsustainable without further hardware and algorithmic advancements. Energy consumption for training AI models has exponentially increased as models have become more complicated, as shown in Figure 1.1, requiring billions of training parameters and massive training datasets to keep up with current demands and trends [6], [7], [8]. On the other hand,

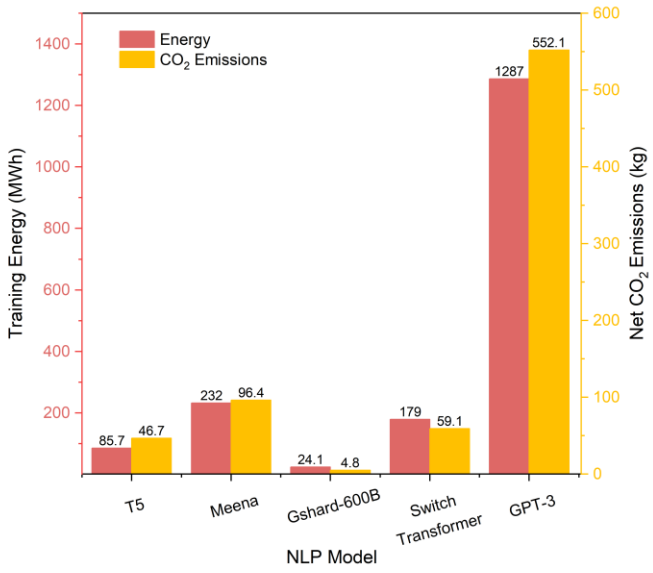


Figure 1.1: (Red) The energy consumption in MWh to train five different Natural Language Processing models [6]. (Orange) The corresponding net CO2 emissions to train the models [6].

edge computing has also become increasingly popular and takes advantage of local processing near or within remote devices, placing an even greater emphasis on low-power computation.

Modern AI implementations depend on conventional computing hardware that fall prey to the limitations of the von Neumann architecture [9], the classical computing paradigm that separates data operations and instruction memory. While the von Neumann architecture has been the backbone for traditional digital computers and has proven effective until now, big data processing and AI implementations have exposed the limitations of the architecture, severely hindering further growth in terms of computing throughput and energy efficiency.

Historically, computers have also been able to take advantage of Moore's law, the observation by Gordon Moore in 1965 positing that the number of transistors in an integrated circuit (IC) is expected to double roughly every two years [10]. In 1974 Robert Dennard further noted that metal oxide semiconductor field effect transistor (MOSFET) scaling maintained a

constant power density, suggesting that the power required was proportional to the size of the MOSFET [11]. Over the past several decades, significant hardware gains in terms of both speed and power can be attributed to these benefits of CMOS miniaturization. However, efforts to scale transistors further have begun to succumb to non-idealities and physical limitations due to quantum effects at the nanoscale signaling an end to this prosperous growth [12]. The upcoming field of neuromorphic computing aims to bypass these limitations using biologically inspired hardware and algorithms which promise to move past these impeding issues and usher in the next generation of high-performance computing technology [7], [8], [16].

1.2 Biology and Neuromorphic Computing

The field of neuromorphic computing aims to approach modern computing problems from a biologically plausible perspective [7], [16], [17], [18], [19]. The human brain is one of the most efficient computing systems we know of today, and while many details are still under study about this complex organ, at its core, processing within the brain primarily occurs using simpler, discrete computational cells. These cells, referred to as neurons, are able to use electrical and chemical signals to communicate amongst each other via synaptic connections allowing for impressive parallel processing capabilities with incredible power efficiency [20].

It is estimated that the human brain contains on the order of 10^{11} neurons and each neuron connects with up to 10,000 other neurons via synapses, yielding on the order of 10^{15} synapses [20], [21]. Biological neurons are able to generate spiking action potentials which propagate through their axons and transmit a signal to neighboring cells' dendrites via the synaptic junction and integrate signal information from the source neuron into the receiving neuron. Synapses are generally excitatory or inhibitory and can modulate how strongly they weigh transmitted signals over time based on the history of neural activity patterns. This dynamic ability for synapses to

strengthen or weaken is referred to as synaptic plasticity and is thought to be the basis of the brain's ability to learn and memorize [20].

The synaptic plasticity of the brain allows it to generalize and adapt to a variety of real-world challenges and problems. Moreover, the brain is able to directly process incoming analog stimuli in the analog domain using a combination of electrical and chemical signals, serving as an important proof-of-concept for massively parallel analog, in-memory computing. All of this is achieved without the need for a fully connected network: In fact, in an effort to optimize connectivity between frequently used neural circuits, the human brain utilizes synaptic pruning as it develops in an effort to get rid of underused and unnecessary synapses, further simplifying the connectivity between neurons [20].

The brain drives all of these processes and supporting structures between them using less than 20 watts of power, with each synaptic event estimated to utilize about 1-10 femtojoules (fJ) [21], [22], [23]. This results in an extremely efficient computing scheme, with the human brain estimated to operate roughly at equivalent computing speeds of between 10^{18} to 10^{25} floating point operations per second (FLOPS) [23]. While modern supercomputers, such as Frontier, have achieved processing speeds on the order of 10^{18} FLOPS [24], [25], the brain is able to perform at these incredible processing speeds while also demonstrating immense energy efficiency, using several orders of magnitude less power than today's large language models (LLM) and supercomputers which can consume hundreds of megawatts [6], [7], [25], enough to power thousands of residential homes [80]. This vast discrepancy highlights the motivation and the need for biologically-inspired neuromorphic architectures to ensure continued and sustainable development of future AI technologies.

Several prior efforts have already attempted to take advantage of neuromorphic architectures to fabricate bio-inspired chips. Works including IBM's TrueNorth [26], SpiNNaker [27], and Intel's Loihi [28], [29] all attempt to design and emulate the parallelism and spiking activity seen in the biological brain. IBM TrueNorth was introduced in 2015 and consists of 5.4 billion transistors that create 256 million synapses with 1 million spiking digital neurons. Consuming just 68 mW of power, this chip was well suited for ultra-low power video processing [26]. SpiNNaker (Spiking Neural Network Architecture) was developed by the University of Manchester using one million low-power ARM processors in an effort to create a massively parallel neuromorphic computer to simulate the real-time behavior of one million neurons [27]. Intel's Loihi chip utilized neuromorphic cores that implemented spiking neurons and synapses that can adapt to incoming information in real-time [28], [29]. The spiking-based computation on this chip was able to outperform conventional solutions on certain classes of problems.

While these bio-inspired architectures may be more efficient for traditional computing applications, building systems that use first principles of the brain would allow us to potentially develop architectures that may also inform on the biological brain itself. One such example is The Blue Brain Project: In an effort to better understand the emergence of biological intelligence, The Blue Brain Project utilized the IBM BlueGene supercomputer to simulate mammalian brains with a high level of biological accuracy [30].

These prior implementations already showcase the significant progress that has been made in terms of neuromorphic hardware and algorithms and the performance thus far seems promising. By using simpler processing elements in a massively parallel manner with reconfigurable synaptic connections, they attempt to leverage the basic operating principles of

the human brain and show potential to move beyond the immediate bottlenecks of modern AI systems.

1.3 Cellular Neural Networks

The Cellular Neural Network (CeNN), shown in Figure 1.2a, is an example of neuromorphic architecture. Proposed by Leon Chua in 1988 [13], [14], the CeNN is composed of simple, repeating computing elements referred to as cells, that are locally interconnected to each other via synapses. This architecture allows for massively parallel, reprogrammable, analog computing schemes that speed up inference times for applications such as image processing, while simultaneously reducing power consumption. The two-dimensional network topology is shown Figure 1.2a, showcasing the cells and interconnections between them.

The interconnection between cells in the network plays a crucial role in the network's ability to run a variety of different types of applications. These interconnections are created through weighted synapses amongst the cells. Cells are connected to their neighbors and communicate by transmitting analog signals to each other during network inference. Traditionally, the CeNN uses a connectivity radius of one ($r = 1$), a metric defining how large a cell's neighborhood is which details how many cells it is able to communicate with. With a radius of one, cells are only connected to their direct neighbors. Larger connectivity radii are possible and create larger, more complex connected neighborhoods, as shown in Figure 1.2b. Assuming a network of M rows and N columns, with M equal to N, if the connectivity radius exceeds N/2, this implies that every cell in the network is connected to every other cell, creating a fully connected scheme which corresponds to a Hopfield Network [15]. For general purposes,

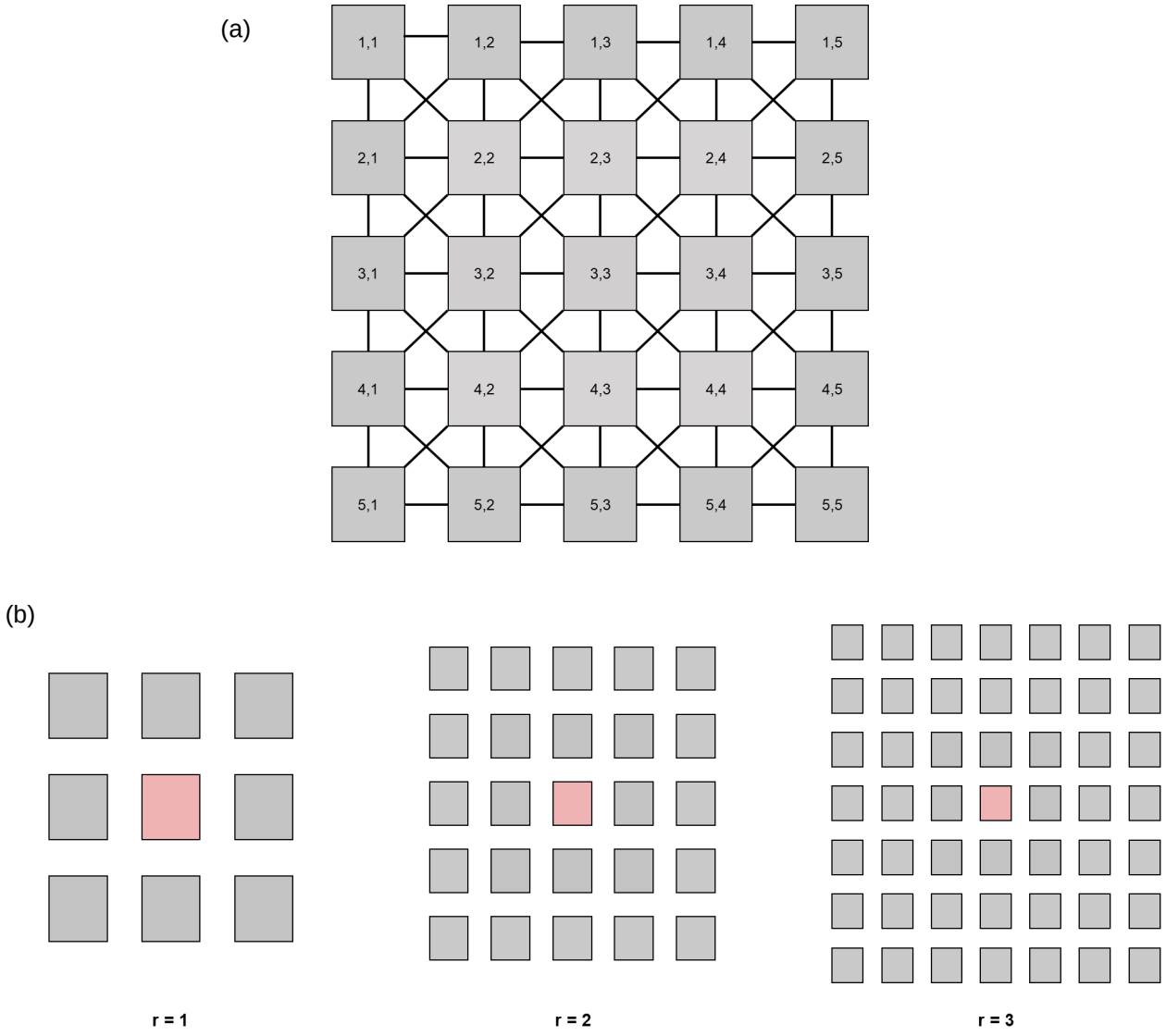


Figure 1.2: (a) General Cellular Neural Network topology. (b) Various network neighborhood sizes based on different connectivity radii, r. Adapted from [13].

however, the radius is usually set to one with cells only being connected to their eight immediate neighbors. For this dissertation, the radius will be set to one for all examples demonstrated.

The synapses of a given neighborhood form localized templates, designated the feedback (A) template and the feedforward (B) template. By manipulating the synaptic weights within

these templates, different tasks can be achieved. Through these local connections and the processing capabilities of each individual cell, complex dynamics arise in the network and certain templates result in global, time-dependent signal propagation throughout the network. By taking advantage of these dynamics, prior CeNN demonstrations have shown efficient image processing, DNA sequencing capabilities, brain tumor detection, partial differential equations (PDEs) solvers, and even mimicry of biological signal generation for robot locomotion [13], [14], [15], [77], [78], [88], [93], [97], [98], [114], [115]. Due to the reprogrammability and the vast variety of compatible algorithms of a fully realized CeNN chip, the ideal network paradigm has been labelled the CeNN universal machine, with capabilities matching that of a Turing Machine [85], [92], [96].

For image processing applications, the cells are arranged in a two-dimensional (2D) regular grid that allows for one-to-one mapping between each cell in the network and a pixel from the input image. During image processing, the A and B template weights are adjusted according to the application to be run, and the greyscale pixel intensities of the image are mapped to the network for processing. Since the weighted templates are stored locally to every cell and since each cell is able to accept and process an incoming pixel from the input image, this enables a massively parallel computing scheme that speeds up image-processing throughput: The input is fed in parallel into the network, the cells process simultaneously, and the output can be read in parallel after network inferencing. Generally, for image processing applications, network runtime is defined as the time it takes for the final cell to achieve steady-state.

Moreover, the repeating cell architecture and the simplicity of the local connections between only adjacent neighbors makes the network easier to fabricate and amenable to very large-scale integration (VLSI). This contrasts with other types of neural network designs that rely

on fully connected nodes, such as convolutional networks, that drastically complicate hardware design and increase on-chip area.

Several other advantages provide motivation for developing a hardware CeNN: several image processing and computation tasks can be handled by a single layer CeNN, unlike more traditional convolutional networks that require several layers for processing. While a single-layer perceptron was unable to solve Minsky's global connectivity problem [31], this task can be completed using a single layer CeNN [15]. Furthermore, for processing on a given layer in convolutional networks, a weighted kernel of a fixed size moves across the layer for processing in fixed stride steps. During computation, however, the need for striding bottlenecks the overall processing speed. Meanwhile, the CeNN is set up in a fundamentally different manner where all cells have a copy of the A and B template local to themselves, bypassing the need for strides which enables truly parallel processing as the cells can all process simultaneously. This type of architecture also more closely resembles biology, taking advantage of the simplicity of local synaptic connections between cells.

1.4 Prior Cellular Neural Network Implementations

Prior CMOS implementations have been demonstrated in the past for image processing applications, including ACE16K, SCAMP5, MIPA, the Toshiba SPS-02 camera, along with the FPGA implementations CESAR and NERO among others [32], [33], [34], [35], [36], [37], [38], [35], [39], [40], [41], [42]. These prior works show the feasibility of building such a network and successful demonstrations prove the potential this type of bio-inspired hardware possesses. They take advantage of the CeNN's parallel processing and reprogrammable nature and have demonstrated rapid processing speeds as fast as 100,000 frames per second (FPS) for image

processing tasks using mixed-mode signals. Along with rapid processing, some implementations achieved relatively large network sizes owing to the simpler fabrication of the local interconnections between cells.

ACE16K boasted a large 128×128 network manufactured on standard CMOS at the 0.35µm process node [32], [33]. Developed by a group at the Institute of Microelectronics Seville, the integrated circuit (IC) included an analog core for image processing, with several buffers driving analog and digital signals to the cell array. Though the cells are analog, the processer is digitally controlled and depends on digital to analog (D/A) and analog to digital (A/D) converters to read and write to the cells. For the weights between cells, a bank of programmable analog CMOS multipliers was used to connect cells to their eight direct neighbors. Using this setup, ACE16K was able to achieve image processing speeds greater than 1,000 FPS with readout's achieving precision of 8-bits with each cell using approximately 180µW of power. While this architecture does tout a true analog network, it was one of the earlier implementations of the CeNN, using the now outdated 0.35µm process node, resulting in a large cell footprint and high-power consumption. The network also operated on the order of 1000s of frames per seconds which is relatively slow compared to 10K-100K FPS processing demonstrated by more recent implementations.

Another hardware implementation was demonstrated by a group at The University of Manchester: the SCAMP5d vision chip [34], [35], [36]. The chip was fabricated on 0.18 µm to be a general-purpose programmable system touting 256×256 pixels using an embedded image sensor, with each pixel mapped to a single cell of the network. This setup enabled direct processing of incoming analog data from the image sensor within the camera itself, drastically reducing the latency due to data movement off chip and bypassing the need for power intensive

D/A conversions. The SCAMP5d network was a mixed-mode design with cells connected to their four cardinal neighbors. Analog registers allowed basic arithmetic operations (add, subtract, multiply), while 1-bit digital registers allowed for Boolean operations (OR, NOR, and NOT) and an external controller provided the cells with the instructions to execute based on the task to be completed. The output was flexible, allowing for readout in the form of 8-bit digital data or as analog frames. Using this chip, the group demonstrated image processing speeds of 100,000 FPS in an object recognition task locating closed-shape figures within a cluttered scene with each cell consuming about 18.8μW while the entire setup used only 1.23W at an instruction rate of 10MHz. However, the dependence on instruction rate is one of the limitations of this architecture: The network does not utilize time-continuous processing, instead using a microprocessor-type clock-based approach in which more complex algorithms may require many basic processing steps to execute. Moreover, the IC is controlled by an external instruction issuing device, requiring additional dedicated digital hardware in each cell to store and decode these instructions.

MIPA4K was another example chip designed by a group at the University of Turku, Finland [37], [38]. Fabricated on the 0.13 μm process node, the network was a mixed-mode 64×64 processor array that also included an embedded image sensor. The processor cells consisted of optimized circuit cores dedicated to performing specific functionalities. The main grayscale core was used to efficiently implement non-linear filtering operations for low-level image data reduction in a fully analog and parallel manner. This core allowed cells to communicate in the cardinal directions and can process analog input directly from the integrated image sensor, enabling simple arithmetic operations, as well as iterative grayscale processing when more than one basic operation is needed. The binary core, on the other hand, permitted

connectivity to all eight direct neighbors and utilized 1-bit programmable weights to achieve logic operations. The dedicated hardware was optimized for high-speed processing has been demonstrated to run at speeds greater than 100,000 FPS for both single-step digital processing and gray-scale analog processing with each cell consuming about 6.5µW. Although MIPA4k shows great processing speed and power efficiency, the implementation of the network architecture is fairly intricate, with cells possessing a combination of binary processing cores and grayscale processing cores that were selected based on the type of network operation. The resulting cell architecture contains approximately 1500 transistors, drastically increasing the cell's footprint size.

Toshiba's SPS-02 (Smart Photo Sensor) was the first commercially available camera featuring a compact image processing system based on the CeNN [39]. Camera processing was enhanced by pre-processing using an integrated 144x176 CMOS CeNN which enabled high-speed applications like object orientation detection and presence detection among a variety of other industrial use cases.

Other types of implementations include CESAR and NERO which are based-off FPGA hardware [40], [41] from a group at the Dresden University of Technology. While not pure hardware implementations, the FPGA designs bypassed the primary limitations of the VLSI implementations: IC prototypes ran into certain hardware drawbacks, such as low-bit accuracy and volatility of analog memories that limit these prior works' ability to truly implement all functionalities of the ideal CeNN universal machine. CESAR was designed as a virtual network on which a time discrete CeNN universal machine could be run with 18-bit data representation for high accuracy modeling [40]. A higher accuracy and more precise weight tuning allowed for more complex computation beyond image processing such as partial differential equation (PDE)

solvers. NERO takes this one step further and implemented a scalable FPGA architecture processing up to 480×640 cells while maintaining the same 18-bit accuracy, allowing for efficient computation of sequential CeNN operations. This type of sequential processing reduced the need to access RAM, in turn reducing overall execution time of the virtual cells of the network allowing certain operations to be completed within nanoseconds, with the whole network capable of completing tasks on the order of microseconds [41].

1.5 Emerging Memristive Devices

In an effort to more faithfully emulate the behavior of biological networks, neuromorphic computing explores the use of novel electronic devices that fundamentally function in a manner akin to real neurons and synapses to further improve network architectures. One such type of device includes the emerging nanoscale memristor. First proposed by Leon Chua in 1971, he observed that the four fundamental circuit variables current, voltage, charge, and magnetic flux could have six possible relationships. Two were covered by basic physical laws and three were related by the existing three basic circuit elements that existed at the time: The resistor, capacitor, and inductor. There was still one missing relationship, however, that linked charge and magnetic flux. This led him to postulate the existence of a missing fourth basis circuit element, the memristor, a two terminal, passive device that relates charge and magnetic flux via its "memristance" [43].

It was not until 2008 when HP Labs demonstrated a Titanium Oxide (TiO_2) based device with the pinched hysteresis loop shown in Figure 1.3a to link theory with experiment [44] and since their inception, memristors have garnered significant attention for use in neuromorphic applications [45], [46], [47], [48], [49], [50]. Although a variety of different materials have been

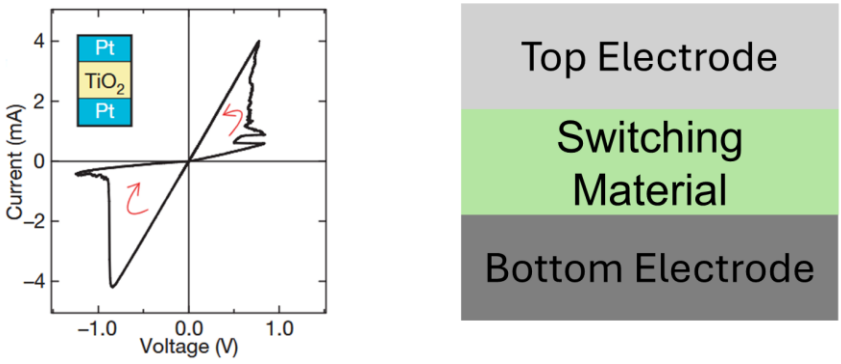


Figure 1.3: (a) Experimental I-V sweep of the TiO2 device created by HP Labs. Image courtesy of [44]. (b) The generalized stack structure of metal-oxide-metal memristor devices.

used, memristive devices tend to be composed of simple metal-oxide-metal structures, as shown in Figure 1.3b. The concept of memristors today has become more broadly generalized to include memristive systems and covers resistance switching devices as well as phase change memory devices, spintronic devices, and ferroelectric devices [51], [52], [53], [54], [55], [56], [57], [113].

One class of memristive devices referred to as diffusive memristors has proven to exhibit integrating and spiking behavior [46]. The resistance state of the device modulates in response to applied voltage pulses and resets given enough time, in a manner similar to how biological neurons are able to integrate information and fire an action potential. Such devices have been shown to be suitable for leaky integrate and fire (LIF) spiking applications that aim to more closely resemble neuronal spiking behavior [46], [102], [103], [104]. Moreover, these devices exhibit ionic dynamics similar to that found in biological synapses, enabling short term plasticity akin to what is observed in the brain [20], [46] further supporting their use in brain-inspired computing schemes. Along with LIF behavior and ionic plasticity, these devices have also been used to model more specialized sensory neurons, including artificial nociceptors capable of

detecting noxious stimuli that can exhibit localized sensitization [47]. Additionally, the nonlinear dynamics of diffusive memristors make them useful for use in reservoir computing (RC), in which temporal input information is mapped to a higher dimension feature space "reservoir" and used to demonstrate pattern classification capabilities [48].

Drift memristors are another class of memristive devices and are capable of zero static-power, non-volatile analog weight storage [52], [57], [66], [105], [106], [112]. Our studies and prior efforts for characterization have shown that these devices are able to achieve a wide range of stable analog resistance states and have good cycle-cycle uniformity. They also operate within a wide range of temperatures, ranging from 80K to 438K [57], [106] and exhibit radiation hardness [58], [59], [60] making them suitable for use in extreme environments like space. This class of devices has already been demonstrated in numerous studies implementing various arrays and neural networks [61], [62], [63], [64], [65], [66], [67], [68], [112] exhibiting features such as in-situ learning and spike timing dependent plasticity (STDP). Reconfigurable memristor crossbars composed of hafnium oxide memristors using a one transistor, one memristor (1T1R) setup have been shown to be capable of analogue vector matrix multiplication with large array sizes up to 128x64 elements for a variety of real-world applications [64], [66], [112].

Improvements to programming algorithms have also allowed extremely high precision tuning of these devices, with recent studies achieving up to 2,048 conductance states [67], [68]. Such high precision improves the accuracy of network inferencing and unlocks capabilities to handle more specialized problems, such as PDE solvers, which require higher precision during computation for high-fidelity results [68], [69]. Along with the impressive programming capabilities of drift memristors, their simple structure is amenable to 3D fabrication, allowing for

multilayer stacking that can efficiently implement sequential operations without increasing lateral chip area or incurring latency penalties due to extraneous data movement [70].

By using both the diffusive memristor and drift memristor in conjunction, bio-realistic long term plasticity was realized [46]. Furthermore, combining the non-volatile weight storage capabilities of drift memristors as synapses, and the temporal dynamics of diffusive memristors to emulate spiking neurons, a fully memristive network capable of unsupervised learning for pattern classification was demonstrated by Zhongrui Wang in 2018, further showcasing the capabilities of these devices for neuromorphic computing schemes [71].

1.6 Overview

Chapter 2 will focus on the CeNN GUI software package and the digital twin simulator built to integrate network fundamentals and operating principles and assess the complex dynamics exhibited by the CeNN. The simulator allows rapid testing across a variety of different applications for any given network size, providing insight into network convergence times and individual pixel behavior. The A and B template, along with the bias can be easily adjusted and provide insight into how changes in the synaptic weights influence the final output. Simulated image and video processing approaching 100,000 frames per second are demonstrated. The introduction of simulated noise improves understanding of mismatch and variance in a fabricated network. Variations in the output function and the neighborhood shapes of cells are described and preliminary simulations are explored in an effort to further minimize on-chip area and power consumption. Beyond image processing, the software package allows us to simulate even more complicated tasks, including a simulated CeNN for solving partial differential equations. The

software package provides a robust digital twin for network simulations as well as a graphical menu for users to interface with the custom fabricated CeNN hardware.

Chapter 3 shifts the focus to the design, fabrication, and measurement of a CMOS prototype 5×5 hardware CeNN at the 65nm process node and an attempt of designing a larger-scale network at the 28nm node. The analog cell and synapse architecture is covered, supported by SPICE simulations using TSMC's 65nm device models. The controller interface between the software GUI and the IC along with the supporting peripheral circuitry is discussed. Viability of the reprogrammable hardware is demonstrated by running edge detection and vertical/horizontal line detection algorithms, showing measured processing speeds greater than 20,000 frames per second in the fabricated hardware.

Chapter 4 delves into the use of Ta/HfO_x drift memristor devices in CeNNs in order to improve upon the pure CMOS implementation. The cell and synapse architecture used to incorporate these memristors is discussed, along with the custom write-and-check program scheme that is used to program the memristors to ensure precise programming of the devices before inferencing begins. Based on this architecture, a board-level CeNN prototype is built and used to demonstrate horizontal line detection.

Finally, chapter 5 provides concluding remarks and avenues for future research directions, including potential architectural improvements, a discussion on the integration of memristors on a pre-fabricated CMOS die, methods to better implement the hardware network IO, along with the motivation to expand the concept of the 2D CeNN to 3D space, challenges, and its potential applications.

CHAPTER 2

HYBRID ENVIRONMENT: GRAPHICAL INTERFACE AND DIGITAL TWIN SOFTWARE SIMULATOR

In this work a robust CeNN software package is presented, and the built-in software digital twin simulator is used to demonstrate a variety of different applications. Written primarily in python, the software package, referred to as the Hybrid Environment (HyE), contains a GUI that allows users to run a variety of network simulations and noise analyses while also serving as a streamlined software interface to control the custom fabricated hardware. CeNN simulations using the digital twin validate and demonstrate the CeNN's capabilities for high-speed image processing as well as partial differential equation (PDE) solving. The digital twin's performance informs on how the real hardware network behaves and provides understanding on how noise in the input, templates, and output function can affect processing. Additionally, a simulated convergence detection algorithm is used to visualize how certain image processing tasks converge over time to their final state, useful for understanding network dynamics in real-time. This comprehensive software package allows for rapid network simulations to validate a variety of CeNN applications, modeling both local cell interactions as well as global network dynamics that provides valuable insight of network behavior in the real-world fabricated hardware.

2.1 Introduction

Simulated implementations of the CeNN have been used for a variety of purposes in the past. CeNN simulations have been demonstrated both as mathematical models and as SPICE simulations to demonstrate proof-of-concept of network dynamics, validate various application

templates, solve 2D partial differential equation, as well as support the feasibility of a multilayer CeNN [13], [14], [15], [72], [73], [74], [75], [76], [77], [78], [84].

To replicate network dynamics, simulators must faithfully emulate cell dynamics. The network is able to take advantage of a 1:1 mapping of input pixels to processing cells, allowing for parallel processing as shown in Figure 2.1a and each cell is an individual processing unit capable of weighting, summing, and integrating these input signals into their cell states before finally using an output function to convert their state to their respective output. Assuming a connectivity radius of one, the cells are only able to interact with the nearest eight neighbors as shown in Figure 2.2 via the A template and the B template. The A template provides weight for the neighboring cell's outputs *Y*, while the B template provides weight for the input pixel data *U*. This weighted data is then accumulated within the cell and integrated as part of its own cell state. To model these dynamics, the digital twin utilized Kirchoff's current law (KCL) based off Leon Chua's original cell circuit model [13], shown in Equation 2.1:

$$C_x \frac{dV_x(t)}{dt} = -\frac{1}{R_x} V_x(t) + \sum A \odot \left(V_{y_{neighborhood}}\right) + \sum B \odot \left(V_{u_{neighborhood}}\right) + I_{bias} \quad 2.1$$

Where ⊙ denotes an elementwise matrix multiplication (Hadamard product) of the templates with their local neighborhood matrix. From this model, the inputs into a cell are represented in the form of voltages. These voltages are multiplied by resistive synapses that comprise the A and B templates, which convert them into the current domain following Ohm's law. The currents are then summed using the properties of KCL, allowing for accumulation of the weighted inputs. This summed current is then integrated as part of the cell state voltage at the state node,

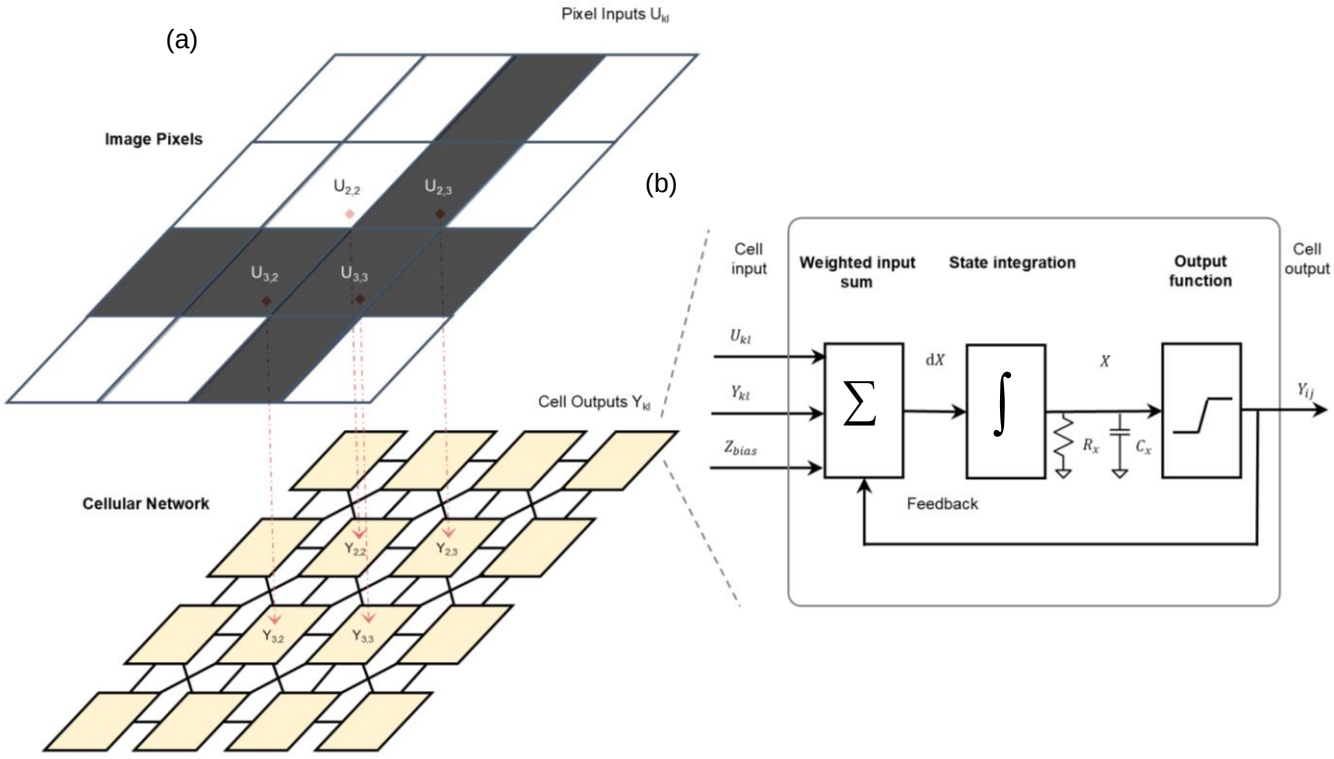


Figure 2.1: (a) One-to-one input pixel to network mapping for high-speed parallel image processing. (b) Block diagram modelling single cell dynamics.

composed of a state resistor (R_x) and state capacitor (C_x) which results in the cell's updated state voltage (V_x), visualized in Figure 2.1b. After integration, the cell state is passed through an output stage which clips the cell state voltage into a predictable range, shown in Figure 2.3, ensuring the output is bound modelled by the piecewise output Equation 2.2:

$$V_y = \begin{cases} -100mV & if\ V_x < -100mV \\ 100mV & if\ V_x > 100mV \\ V_x & else \end{cases} \qquad 2.2$$

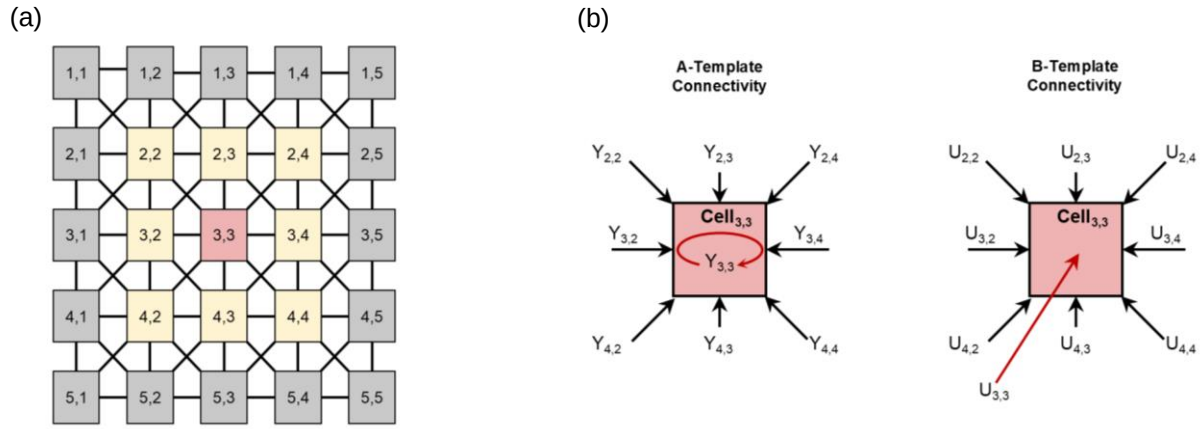


Figure 2.2: (a) Neighborhood synapse connections given a connectivity radius of 1. (b) A and B template inputs.

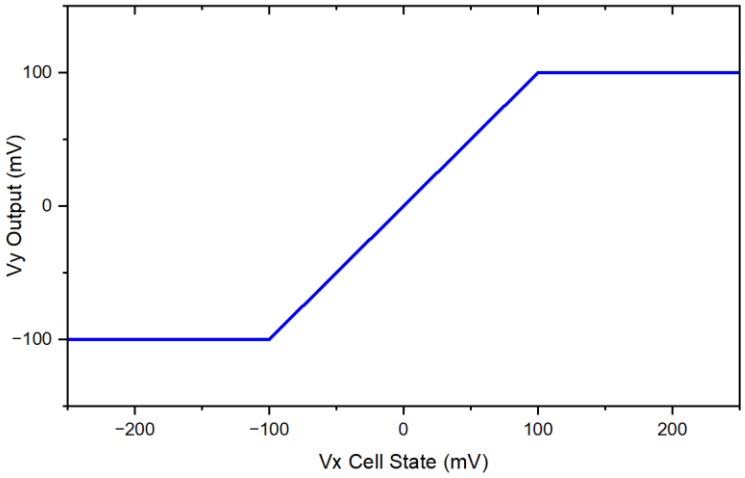


Figure 2.3: Standard piecewise output function used to convert the cell state voltage into the cell's output based on Equation 2.2.

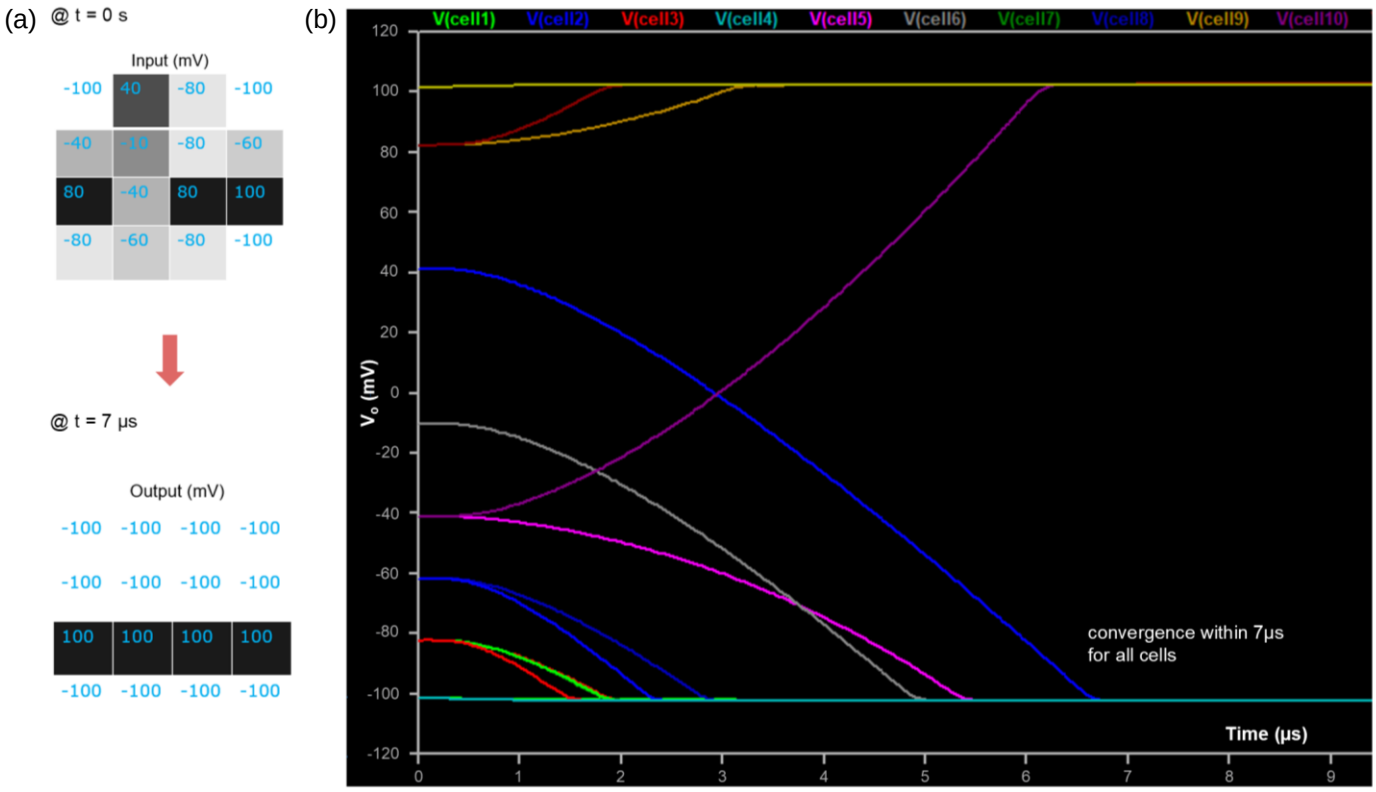


Figure 2.4: (a) SPICE simulation of the horizontal line detection task pixelwise input and output. (b) Visualization of individual cell outputs reaching convergence within 7µs (>140K FPS).

Network dynamics arise from the interactions between cells through their weighted synapses. Generally, the B-template computation is a linear matrix operation. However, the network also relies on complex time dynamics that are introduced via the A-template, which operates on the real-time neighboring cells' outputs as well as the cells feeding back into themselves. Since each of the cells within the network handle their own respective templates, processing can occur in parallel given appropriate input and weights. For image processing tasks, generally network convergence is realized once all cells have achieved a steady-state voltage: i.e. the last cell to converge dictates the total processing time of the network for a given image

processing task. Figure 2.4 shows the voltage outputs of individual cells converging to steady state in a 4×4 simulated SPICE network running the horizontal line detection (HLD) application.

The HyE was originally developed to provide an interface with the hardware, but quickly came to include a robust digital twin that could run a mathematically modelled ideal CeNN solver emulating various parameters of the hardware, such as the state R and C. Since the software was built in a modular manner, additional modules were added over time, including one for noise analysis and per pixel convergence visualization providing further insight into individual cell behavior and network dynamics. Once the hardware had been fabricated, the digital twin provided quick bias and offset simulations to give us a better understanding of how to fine tune the A and B templates to run on the hardware network given its intrinsic offsets and mismatches from the ideal.

The HyE also included a “stitching” algorithm module. This algorithm was devised to allow an MxN hardware network process an image with larger dimensions than the simulated network itself as may be the case with a small prototype hardware network as shown in Figure 2.5. Generally, each pixel of an image is mapped to a certain cell of the network, but this is not possible if the image to be processed is larger than network size. The algorithm itself effectively uses the dimensions of the image and uses a partition of the network with dimensions that are factors of the image dimensions. As an example, a 20x20 image could use a 5x5 network for processing. In this particular case, in a manner similar to how a convolutional network uses a kernel that strides across an image, this partition of the network is able to process chunks of the image. Once all the individual chunks have been processed, they are “stitched” back together to showcase the entire processed image.

While prior simulator implementations have demonstrated the capabilities of the CeNN, they are fairly rudimentary and tend to be specific in their application. The HyE software discussed in this section aims to provide a much more robust and comprehensive simulator along with an easy-to-use interface for the custom hardware.

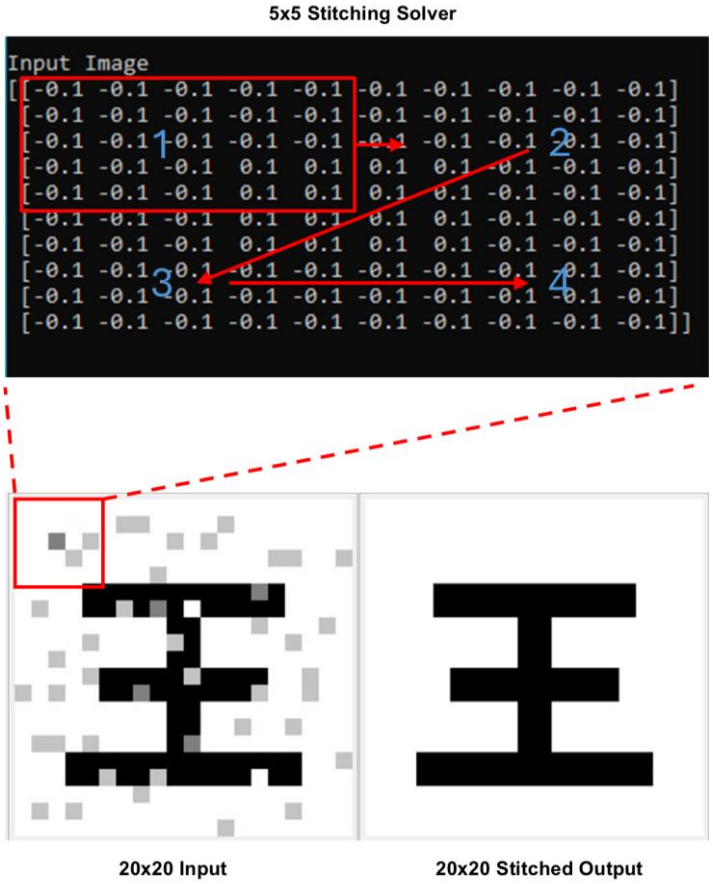


Figure 2.5: Proposed stitching solver to process a larger image (20×20) on a smaller CeNN (5×5).

2.2 Methods

The software platform provided a full-fledged greyscale CeNN digital twin simulator as well as a hardware interface contained within the GUI. The system block diagram is shown in Figure 2.6a. Based on the network's mathematics, the digital twin was written from scratch and modeled the dynamics of its hardware counterpart. For image and video processing tasks, the simulator accepted a variety of common file types, including .jpg, .png. and .mp4 which can conveniently be selected from a local computer directory for processing.

Fundamentally, the digital twin was a virtual, discrete-time CeNN [90], [91] with a pre-defined timestep that determined overall simulation fidelity. Smaller timesteps resulted in higher fidelity compared to a true time-continuous network but were more resource intensive than simulations using larger timesteps, generally requiring a longer runtime to achieve convergence. The change in voltage of the cells, *dv*, based on the cell dynamics was computed using the Forward Euler method in the following form with a predefined timestep *dt*:

$$dv = \left[-\frac{1}{R_x}V_x(t) + \sum A \odot \left(V_{y_{neighborhood}}\right) + \sum B \odot \left(V_{u_{neighborhood}}\right) + I_{bias}\right] \times \frac{dt}{C_x} \quad 2.3$$

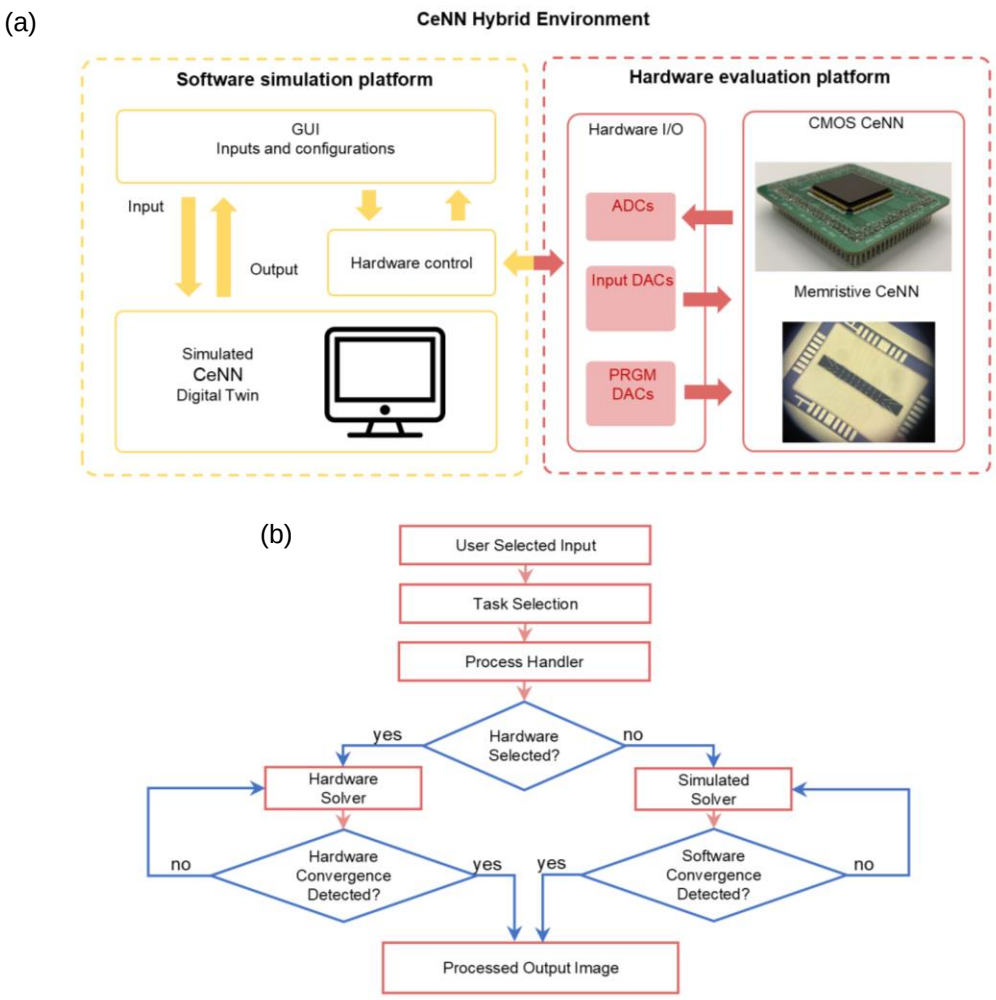


Figure 2.6: (a) Hybrid Environment (HyE) software package block diagram. (b) Digital twin software flow.

Along with the timestep and the total simulation stop time, several other user-defined parameters like the desired A and B templates and the input and initial conditions to be processed are selected and translated into the appropriate format for I/O into the software model or the hardware network. Input files are converted into greyscale and mapped onto a simulated network with the same dimensions as the file. The greyscale conversion was dependent on the light level of the respective pixels from the input file and the new grey-scale pixels are converted into corresponding voltages that are provided as the input for the simulated cells, with -100mV

corresponding to a white pixel, 100mV corresponding to a black pixel, and analog greyscale values in between. While this implementation of the network only allowed for greyscale simulation, it should be noted that color images can be run by either separating the RGB values of each pixel and running the network sequentially for the red values, the green values, and the blue values with the results recombined at the end to restore a processed colored image. Additionally, three separate networks can be run in parallel to handle each of the RGB colors. This is similar to how color vision is handled in the human eye: Specialized light sensing cells called cones are located on the biological retina and these color receptors react specifically to red, blue, or green wavelengths of light depending on the type of cone [79]. Video processing followed the same steps as single image processing, with the additional steps of parsing the video into its constituent frames, processing those individual frames, and then combining the processed frames back into a video.

While users can define a simulation STOP time, convergence detection is enabled during network runtime to ensure the CeNN operations are halted once cells have reached steady state to prevent unnecessary processing overhead. This was via the use of a steady-state detection algorithm. Detection was achieved by making a pixel-wise comparison of the previous frame ($t-1$) and the current frame (t). If any pixel had a different value during this frame comparison, it could be assumed that the network has not converged since that cell had not reached steady state. However, if there were no different values during frame comparison, convergence had been achieved and the task was marked as completed and displayed on the GUI even if the user-defined STOP had not been reached. The software flow of the simulator is shown in Figure 2.6b. The simulated network is based off Equations 2.1 and 2.2 and is capable of simulating an

arbitrarily large network given enough computing resources. Various image processing tasks have been simulated and are showcased in Figure 2.7 and Figure 2.8.

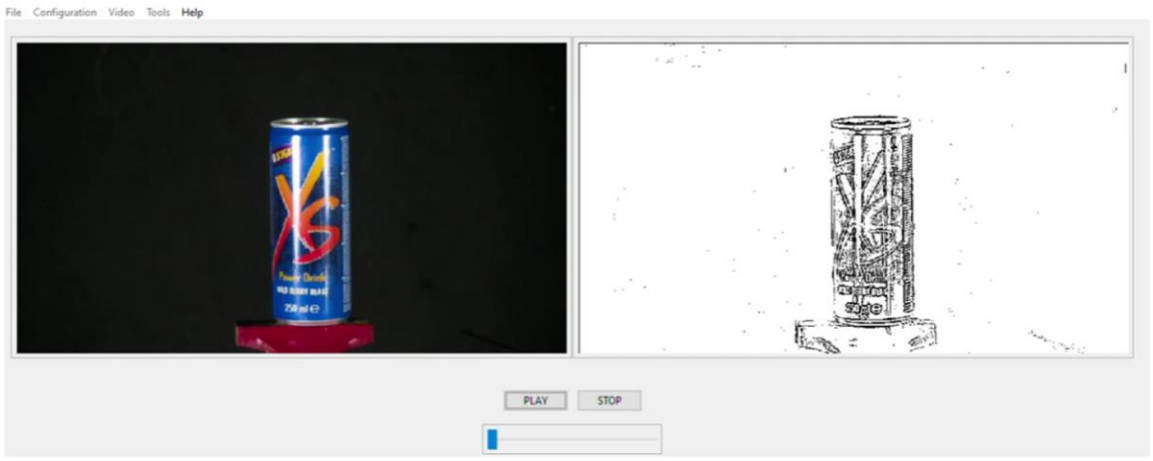
Many of the applications demonstrated in this dissertation have been adapted for use in our digital twin simulator from prior works which formally proved the feasibility of these tasks for the CeNN [13], [14], [15], [72], [73], [74], [75], [77], [78]. Certain applications require only the use of the A-template, some tasks require only the B-template, while others require both. In certain cases, the linear operation of the B-template is sufficient to complete the task without the need for time dynamics. Edge detection is one such application, whose templates are shown below:

Edge Detection Template

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \text{Bias} = -0.1$$

All A-template elements are set to zero, so the real-time cell outputs do not affect their neighbors, and the template only relies on neighboring input pixel values to extract edge information via the B-template. Because this task does not have a time dependence, the cells are expected to converge very quickly and in a simulated environment using discrete time, can be completed within one timestep.

(a)



(b)

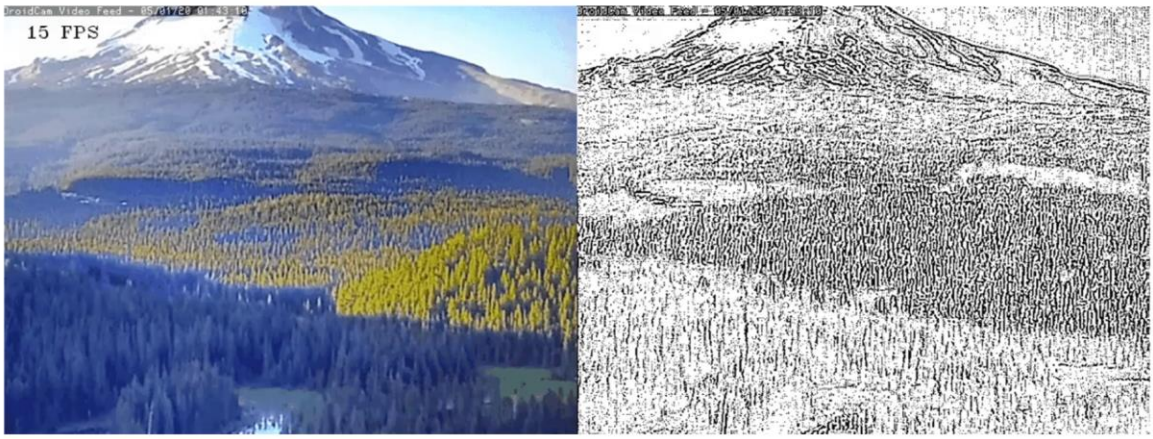


Figure 2.7: (a) Digital twin GUI simulated video edge detection using a local .mp4 file. (b) Real-time video edge detection processing from camera feed.

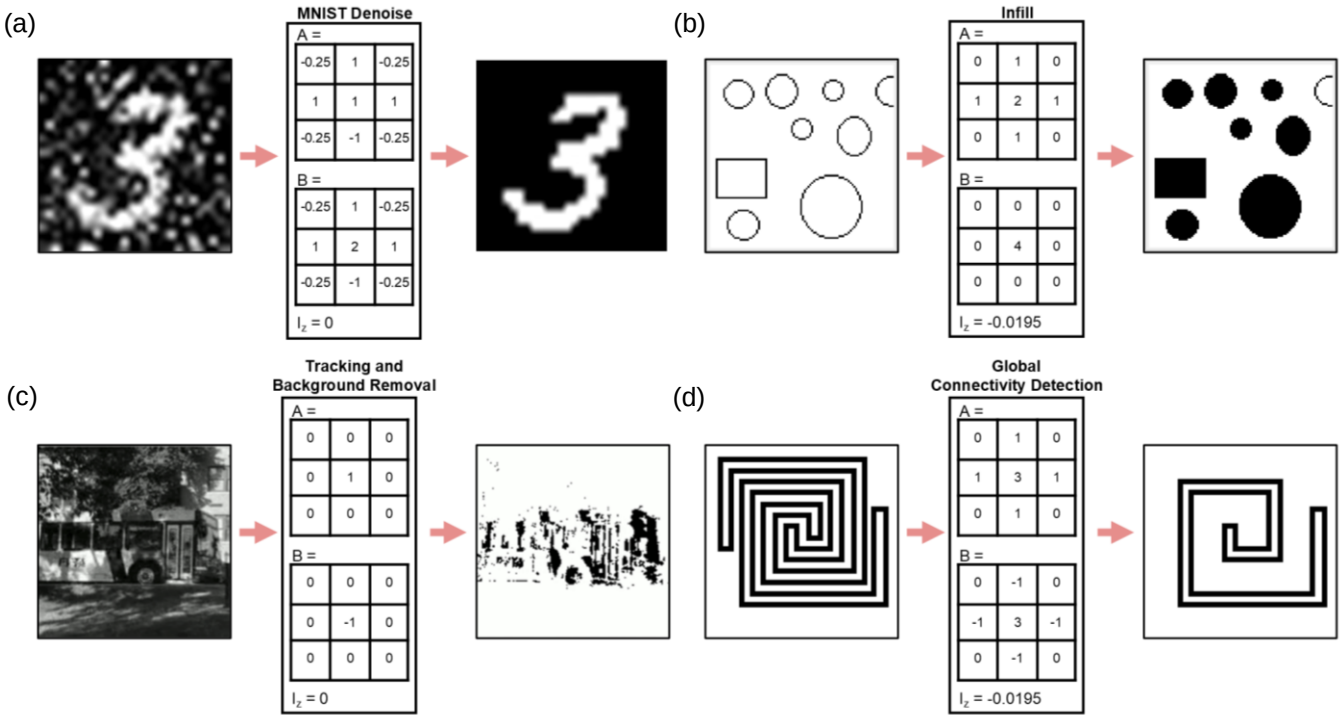


Figure 2.8: Various simulated applications: (a) MNIST denoising, (b) infill, (c) tracking and background removal, and (d) global connectivity detection. Some templates were modified slightly for better performance using the digital twin simulator.

On the other hand, tasks such as vertical line detection and horizontal line detection [13],[14] only require the A-template to function. As the names suggest, these can be used to filter out horizontal or vertical lines in an image. The horizontal and vertical line detection (HLD and VLD) templates are shown below:

Horizontal Line Detection Template

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{Bias} = 0$$

Vertical Line Detection Template

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{Bias} = 0$$

In this case only a single image is used and is mapped as the initial condition of the network. For these line detection templates, horizontal (or vertical) lines can be extracted from the image mapped to the network using only the A-template. The cells are able to look at their immediate neighbors on the left and right for HLD (top and bottom for VLD) and use the following rules to determine convergence: If its neighbors are both white pixels, it too will converge to a white pixel. If one of the neighbors is a black pixel and the other is white, the cell will converge to a stable state based off its initial condition. If it starts as a black pixel, it will remain black and vice versa for white pixel. If both neighbors are black pixels, the cell will converge to a black pixel given enough time.

Still other tasks require the use of the A and B templates. One such task is object tracking, a task that utilizes both templates and requires setting the input and initial conditions. For this case, both templates have all their weights except their center element set to zero,

implying that cells are not affected by their neighbors' real-time output or their neighbors' pixel input, only using their own pixel input and initial cell state. The operation is effectively a logical difference between two frames, detecting the difference between the initial condition of the network (set to the current frame at *t*) and the input image (the frame at *t+1*). Pixels that remain the same value between frames will converge to white, leaving only pixels that have changed between these frames:

Tracking/Logical Difference Template

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{Bias} = 0$$

Other applications utilize more elements within the A-template and depend on the non-linear time dynamics that the feedback template provides. One example includes the hole filler application. The hole filling application [73] is a binary operation that looks at the shapes in an image and determines if a shape is closed or open. These shapes are defined by a black pixel outline with at least a width of one pixel and if the shape is a closed shape, the hole filling template will "fill" the shape with black pixels.

Hole filler Template

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{Bias} = -0.1$$

The initial condition of the network is set to all black pixels and the input is the image containing the shapes to be filled. By using the templates above, the task is completed in an outside-in manner, showcasing the network's property of global propagation with respect to time. Pixels will converge white if and only if they start as white pixels and their cardinal neighbors are less

than 100mV. Pixels will converge black if they were originally a black pixel in the input image or if their cardinal neighbors are all black pixels.

Another application that requires both templates is global connectivity detection. Global connectivity is a binary task that also exhibits propagating dynamics and aims to distinguish between black objects on a white background [15]. The input into the network is the original image and the initial condition of the network is the same as the input image, with some black pixels of one of the objects changed to white. This object is referred to as the marked object and during network runtime the marked object is deleted without affecting the rest of the image, making it obvious if multiple objects in the original image are connected or not.

Global Connectivity Detection Template

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{Bias} = -0.4$$

Using this template, black pixels of the marked object converge to white over time from the marked point. Propagation occurs in all directions from the marked point, but only change black pixels to white pixels following a local rule: Pixels that have at least one neighboring cell whose output is white and input is black will themselves converge into a white pixel while leaving the rest of the pixels unchanged. This process continues until, given enough time, all cells of the marked object display as white pixels, leaving behind only objects that were separate from the marked object.

Noise reduction is another important task that utilizes both templates. In image processing, one of the predominant problems is the introduction of noise while capturing images via cameras and other image sensors. The CeNN has demonstrated to be capable of running noise reduction using an averaging operator [14], [84] and was effectively implemented using the

digital twin to denoise MNIST digits that were corrupted with Gaussian noise and then had Salt and Pepper noise further added. The standard Gaussian noise reduction templates are shown below:

Gaussian Noise Reduction Template

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \text{Bias} = 0$$

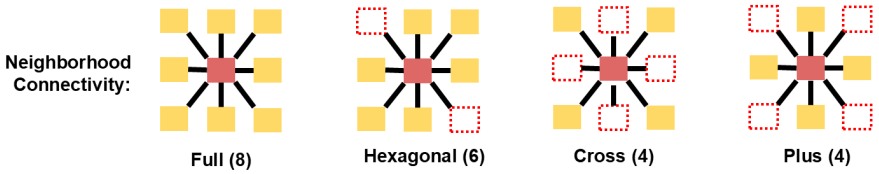
It should be noted that the templates for denoising will change slightly depending on the type of noise present in the image. By modifying the Gaussian noise removal template, the CeNN was able to run as a denoising preprocessor, cleaning up the noisy digits whose output was then fed into a hardware memristor-based classification network designed by Tetramem. Following the denoising step, it was observed that the classification accuracy of the digits had increased compared to without the denoising CeNN processor, supporting that the fast CeNN preprocessing step was effective at reducing noise.

Normally the network operates using a neighborhood connectivity radius of one, implying that the A and B templates are 3x3 in size. However, in an effort to study and further simply the hardware implementation, certain modifications to the neighborhood connections and output functions were simulated and performance was observed as described in Figure 2.9a. Various template “shapes” were tested, the motivation being that these modified templates require fewer synapses in the neighborhood. Fewer synapses simplify VLSI implementations and require less area on-chip as well as lower overall power consumption. Instead of a “full” template with connections to all eight direct neighbors, various template shapes including cardinal neighbors only, diagonal neighbors only, and a few other shapes were tested. These

simulations were all benchmarked against the full neighborhood benchmark based on pixel similarity of the processed image. In this particular comparison, 100 frames were processed using the edge detection task and pixel percent differences were recorded and analyzed.

Similarly, the various output functions used were simulated and compared to the original piecewise function shown in Figure 2.9b. Different output functions introduce different levels of hardware complexity, and in turn area and power consumption based on the number of additional components required to implement a certain function. The various output functions were used for image processing tasks and a pixel-wise comparison was made to the standard piecewise function's processed image.

(a)



(b)

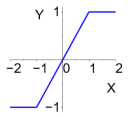
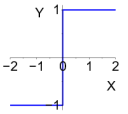
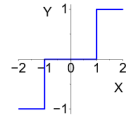
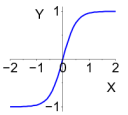
Output Function	Plotted Function	Function Definition
Standard Piecewise		$f(x) = \begin{cases} V_x \leq -1, & V_y = -1 \\ -1 < Vx < 1, & Vy = Vx \\ V_x \geq 1, & V_y = 1 \end{cases}$
Binary		$f(x) = \begin{cases} V_x \leq 0, & V_y = -1 \\ V_x > 0, & V_y = 1 \end{cases}$
Trinary		$f(x) = \begin{cases} V_x \leq -1, & V_y = -1 \\ -1 < Vx < 1, & Vy = 0 \\ V_x \geq 1, & V_y = 1 \end{cases}$
Tanh		$f(x) = tanh(2V_x)$

Figure 2.9: (a) Various simulated template shapes to reduce connectivity. (b) Simulated output functions and their definitions.

While the CeNN shows promise for image and video processing tasks, it is also capable of running more complex algorithms, including a solver for PDEs [77], [78]. One example is the 2D Heat equation problem. This specific problem is used to describe the distribution of temperature u(x, y, t) on a 2D surface as it evolves with respect to time based on Equation 2.4.

$$\frac{\partial^2 u}{\partial t^2} = \alpha * \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.4}$$

Where α is the thermal diffusivity of the material, *u* is the temperature at time *t*, *x* is the position on the x-axis, and *y* is the position on the y-axis of the 2D plane. To map the problem on to the CeNN, the problem is first discretized with respect to space and time. Each spatial grid point corresponds with a cell *C(i,j)*. Using the explicit finite difference scheme shown in Equation 2.5, the temperature of each cell at the next time step *m+1* can be computed.

$$\frac{u_{i,j}^{m+1} - u_{i,j}^{m}}{\Delta t} = \alpha * \left(\frac{u_{i-1,j}^{m} - 2 * u_{i,j}^{m} + u_{i+1,j}^{m}}{\Delta h^2} + \frac{u_{i,j-1}^{m} - 2 * u_{i,j}^{m} + u_{i,j+1}^{m}}{\Delta h^2}\right) \tag{2.5}$$

The equidistant grid spacing, $h = \Delta x = \Delta y$, represents the distance between adjacent cells in the x and y directions. Equation 2.5 is than adapted to the CeNN by using a finite difference approximation for spatial derivatives using the A template, given by:

$$u_{i,j}^{m+1} = u_{i,j}^{m} + A * u_{i,j}^{m} \tag{2.6}$$

$$\text{where A} = \frac{\alpha \Delta t}{\Delta h^2} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{2.7}$$

2.3 Results and Discussion

Using the network's capability to perform ideal network simulations, various image processing applications have been demonstrated. Using the same simulator, video processing was achieved using a local file on the computer as well as using a live camera feed, shown in Figure 2.7. An iteration counting module allowed the visualization of cell convergence for various time-dependent templates, including the hole filler and global connectivity tasks, shown in Figure 2.10. Various hardware modifications were made to simplify network topology and reduce on-chip area by adjusting the template shapes and reducing the number of direct connections within a neighborhood and simulating various output functions aside from the original piecewise function as shown in Figure 2.9. Edge detection performance differences along with topological differences between the various template shapes and output functions are recorded in Table 2.1 and Table 2.2.

Noise analysis on the network was also carried out using the digital twin: Noise on the input, output, along with random weight variations on the templates can be introduced. Input noise was relatively easy to introduce, using a Gaussian noise filter on the input image to be processed before feeding it into the CeNN. Output noise was introduced by adding Gaussian noise to the cell output function. For synaptic weight noise, an algorithm was specifically developed that emulates the variance due to mismatch or programming error while template weights are being set. Various noisy simulations can be seen in Figure 2.11. All noise functions can be calibrated to closely resemble the noise that may be present in a fabricated network and can be used independently or in conjunction.

Preliminary studies using a software CeNN as a pre-processer with a hardware classifier were done here using a noisy MNIST dataset as shown in Figure 2.12: MNIST digits were

artificially noised using a combination of gaussian and salt and pepper noise. A simulated CeNN pre-processor was used to denoise these images using the digital twin and the cleaned images were then passed on to a hardware classifier using the lab's Tetramem memristor-based evaluation kit. Passing the noisy MNIST digits directly to the classifier resulted in poor classification accuracy, but using the CeNN denoise pre-processor improved the accuracy significantly, showing the potential of using such networks in conjunction. The high-speed nature of the CeNN also ensures minimal overall system latency during the pre-processing step.

Due to the digital twin being capable of high floating point precision operations, PDEs were also able to be solved by modifying network weights. Figure 2.13 shows a 2D heat equation PDE example. Preliminary studies show when compared to CPU based solvers using traditional numerical methods for solving, the CeNN solver outperforms standard methods on the same hardware in terms of execution time, especially at higher spatial resolution and smaller temporal resolution. This further demonstrates the capability of the CeNN as a robust network that can be used for a variety of tasks ranging from vision processing to complex, nonlinear PDE problems.

2.4 Conclusion

The software digital twin demonstrated robust simulation performance and took advantage of the CeNN's dynamics to implement a variety of tasks, ranging from static image processing, video processing, and PDE solving by adjusting the weighted templates, the input, and the network's initial conditions without changing the network topology. Tasks that were previously unachievable using a single-layer perceptron networks, such as global connectivity, have been demonstrated to be solvable using a single-layer CeNN and functional modules allow us to track how convergence occurs, providing further insight into real-time network dynamics.

Moreover, network-level simulations show promise to be able to compute PDEs in a manner more closely related to how PDEs naturally evolve in real-time, improving efficiency over traditional numerical solvers, and further taking advantage of the analog, real-time nature of the CeNN.

Table 2.1: Neighbor connections and edge detection performance difference between the various simulated template shapes.

Template Shape	**Number of Connected Neighbors**	**Average % px Difference for Edge Detection**
Full	8	Benchmark
Hexagonal	6	1.51%
Cross	4	3.31%
Plus	4	1.24%

Table 2.2: Hardware complexity and edge detection performance difference between the various simulated output functions.

Output Function	**Avg % px Difference for Edge Detection**	**Number of Transistors Required to Implement**	**Estimated Area on 65nm node**
Standard Piecewise	Benchmark	24	11.69μm^2
Binary	0.23%	12	5.844μm^2
Trinary	0.00013%	24	11.69μm^2
Tanh	0.074%	0	0μm^2 (No additional hardware used)

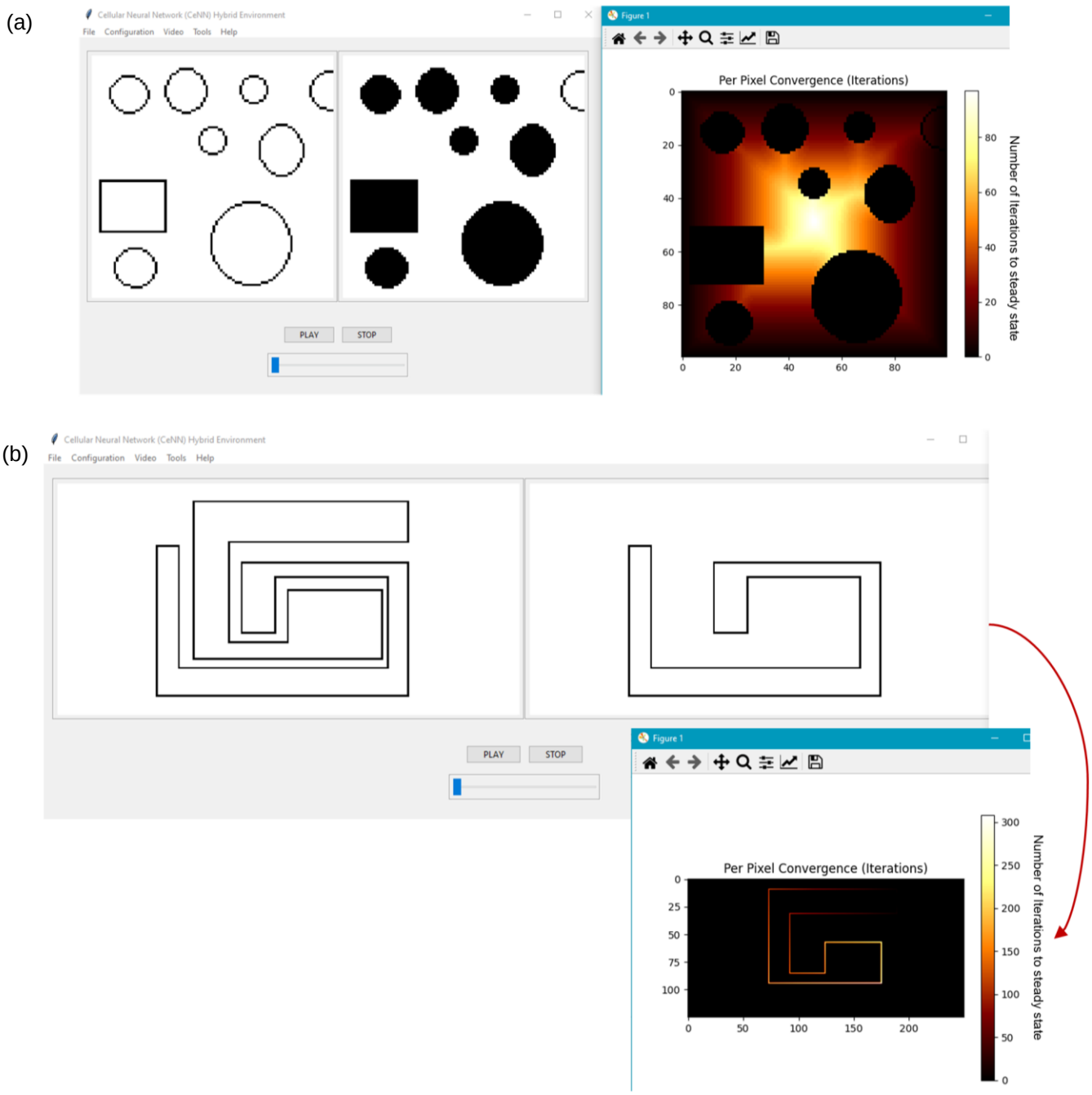


Figure 2.10: (a) Per pixel iteration for hole filler task. (b) Per pixel iteration for global connectivity task. Convergence heatmaps represent pixels that take longer to converge as lighter colors.

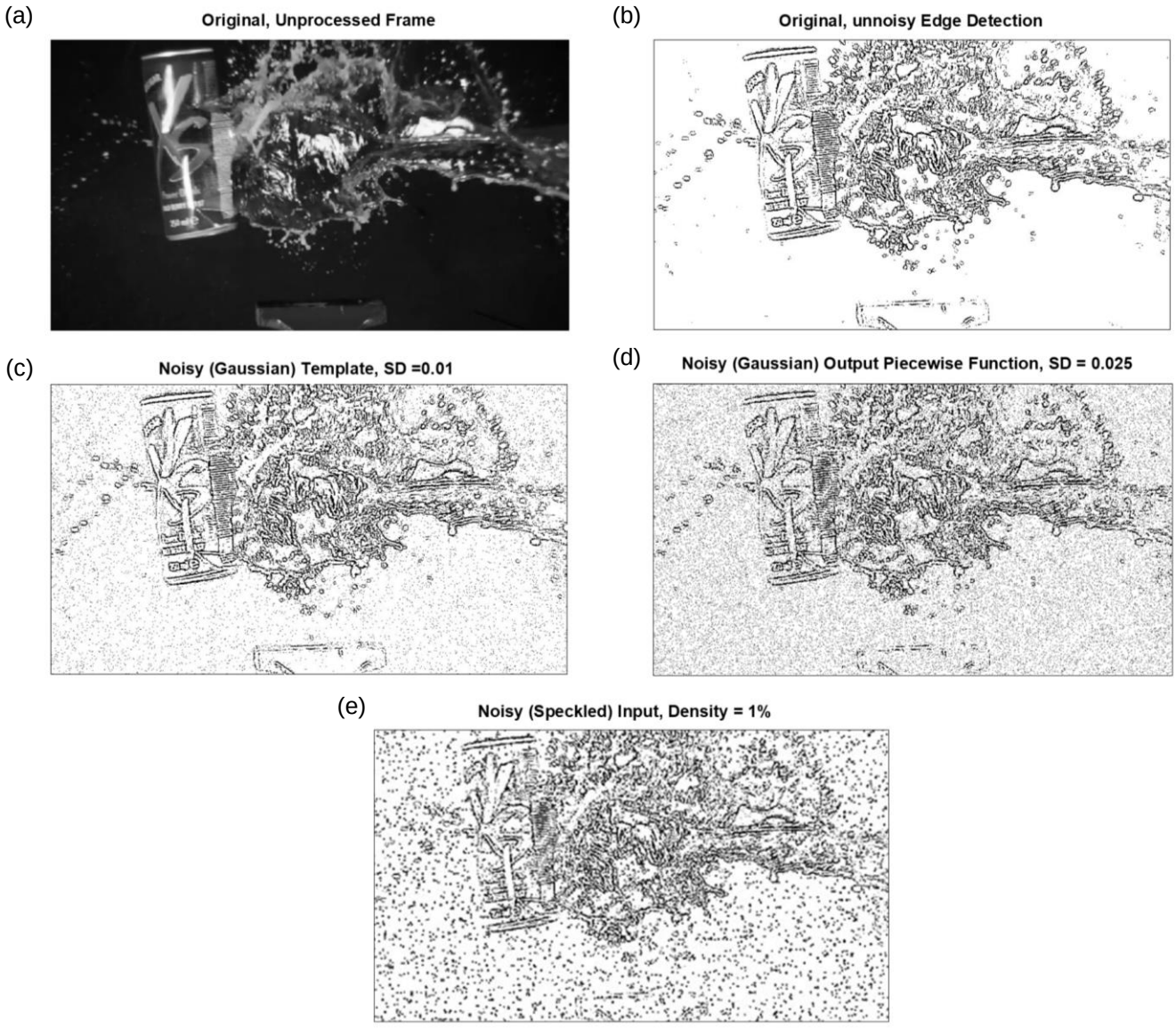


Figure 2.11: Introducing noise at different levels of processing: (a) Original input. (b) Ideal edge detection. (c) Processing with Gaussian noise added to templates. (d) Processing with Gaussian noise added to piecewise output function. (e) Processing with Speckled input noise added to input image.

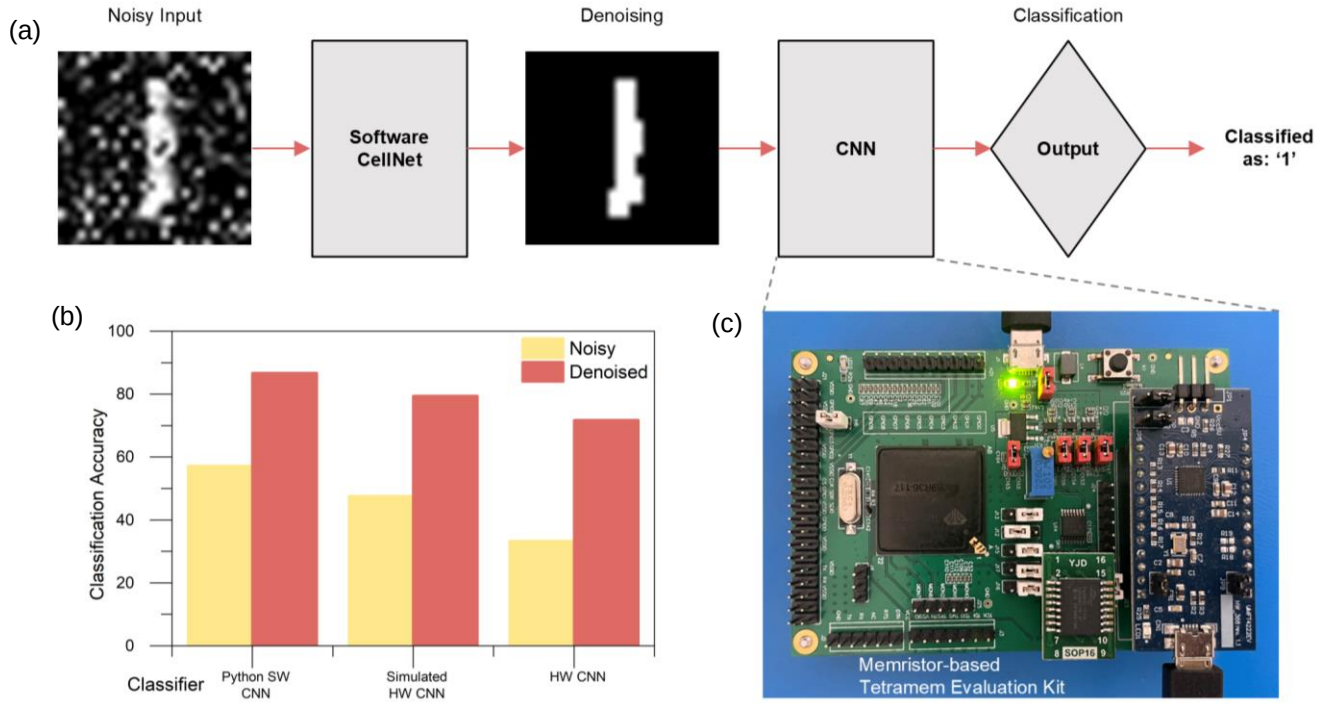


Figure 2.12: (a) CeNN as a pre-processor for MNIST digit denoising to improve memristor-based hardware classification accuracy. (b) Classification accuracy before and after CeNN pre-processing on 10K MNIST digits with Gaussian plus Salt and Pepper noise showing improvement after denoise step. SW is software, HW is hardware. (c) Tetramem evaluation kit used for hardware classification.

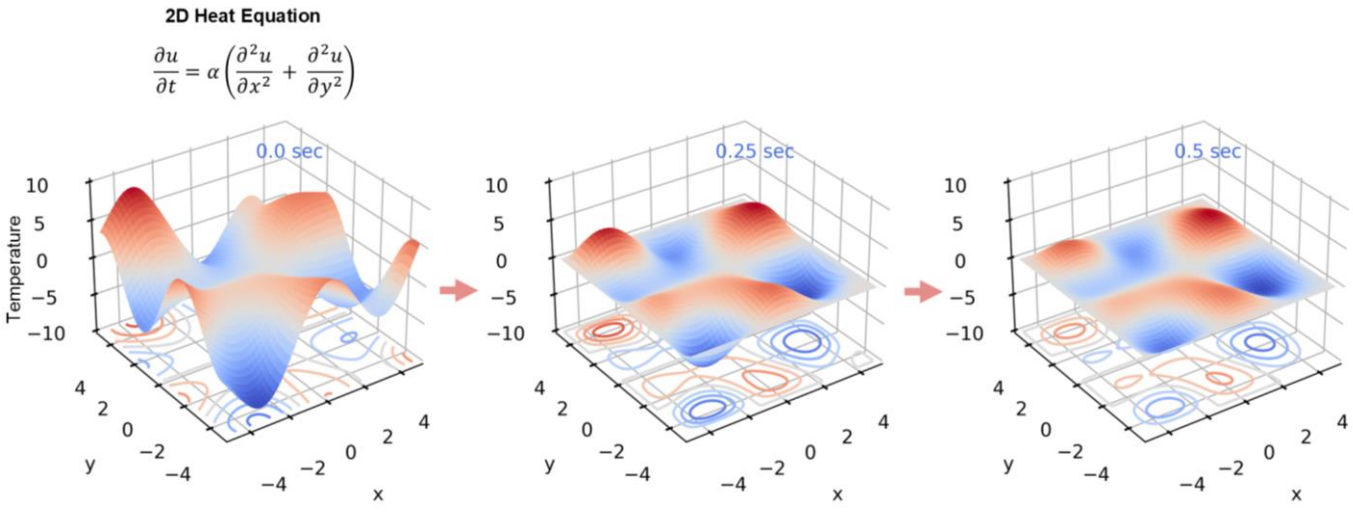


Figure 2.13: Simulated 2D Heat Equation PDE using CeNN using spatial discretization and intrinsic cell dynamics.

CHAPTER 3

CMOS-BASED HARDWARE CELLULAR NEURAL NETWORK

After mathematical modeling and running simulations to validate network topology and performance, an analog-mode IC was designed using the 65nm PDK provided by Taiwanese Semiconductor Manufacturing Company (TSMC). Although the HyE digital twin simulations have shown promising results, the next step was to create non-traditional computing hardware and take advantage of the physical benefits of the CeNN and to better characterize their advantages over traditional computing hardware. The fabricated dies were packaged and interfaced with driving circuitry for testing and network inference. Synaptic weights were successfully reprogrammed for image processing applications and used to showcase different tasks on a single chip, including edge detection and vertical line detection. The primary aim of our integrated circuit (IC) was to create an analog-mode implementation that could operate asynchronously, all while simplifying the reprogrammable synapses in order to reduce on-chip area and power requirements. From the 65nm IC results, deeper scaling network design was attempted on the 28nm process node in an effort to build an 8x32 network. Furthermore, our chip architecture was designed with memristor integration in mind to allow for the development of hybrid CMOS/memristor chips as a future next step.

3.1 Introduction

Prior CMOS-based hardware implementations have been demonstrated [32], [33], [34], [35], [36], [37], [38], [39], [88], [94], [95], [101], [110] and showcase impressive performance

capabilities of the CeNN. However, many of these prior implementations fall prey to complicated mixed-mode block design, dependence on synchronized clock signals, complex analog multiplier designs to handle weighted operations, or focus on binary-mode operations, neglecting the intrinsic analog advantage of the CeNN. We taped out a small-scale network of 5×5 cells at the 65nm process node to demonstrate a purely analog, asynchronous prototype while also attempting to simplify its architecture while capturing the key benefits of the CeNN, shown in Figure 3.1.

3.2 Methods

The purpose of this hardware network was to design and build a simplified CMOS architecture that was amenable to parallel processing and allowed for easily reprogrammable synaptic weights between the synapses. Initial simulations focused on building a simple, easy to program analog synapse based on a CMOS bridge architecture, dubbed the "FET bridge". A single CMOS synapse is composed of four transistors in a bridge arrangement plus an additional three transistors that form the differential amplifier for voltage to current conversion. The bridge itself serves as two parallel voltage dividing circuits, using a reference voltage V_{ref} and a program voltage V_{prgm} to bias the FET bridge gates, thus allowing for easy modulation of the synaptic weight. This in turn produces a top node voltage (V_A) and a bottom node voltage (V_B) shown in Figure 3.2a. These two voltages are used as the input into the differential amplifier and the voltage difference between them modulates the total current output of synapse based on the internal gain of the differential amplifier *A* as modelled by Equation 3.1:

$$\boldsymbol{I_{synapse} = A_{DiffAmp} \times (V_A - V_B)} \qquad 3.1$$

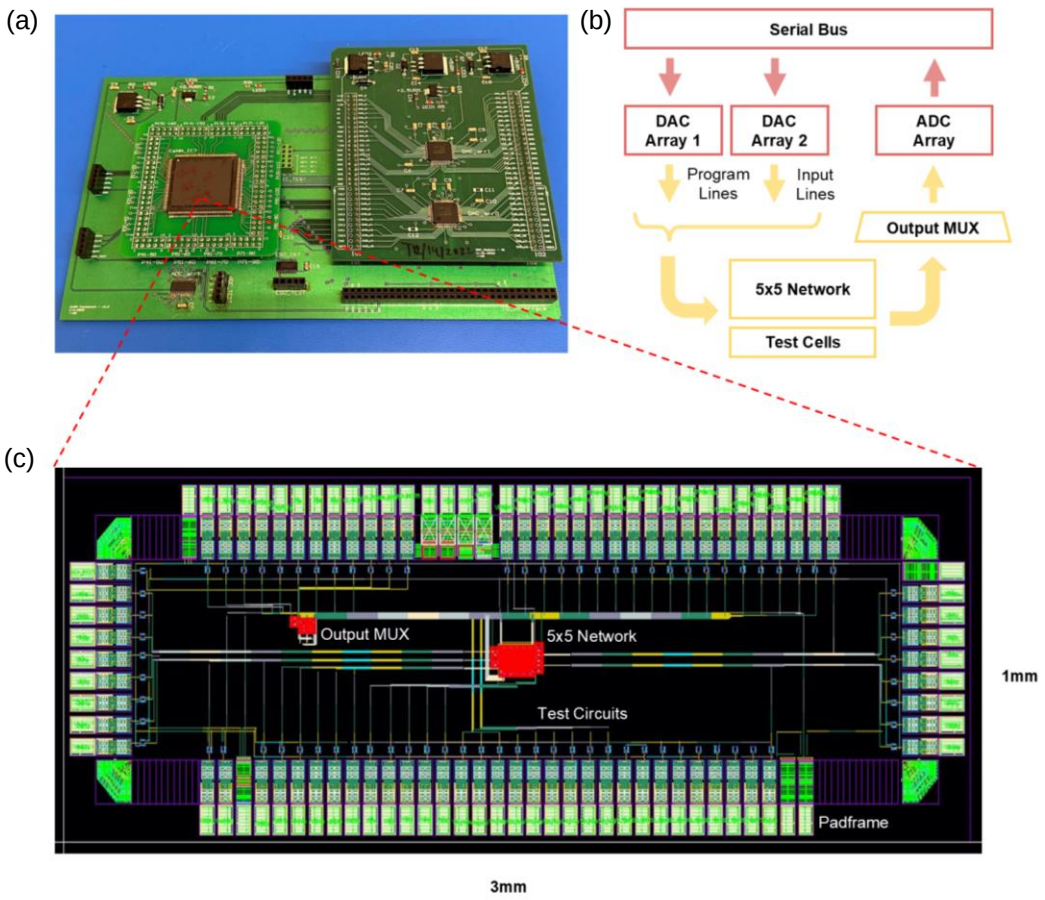


Figure 3.1: (a) Fabricated hardware setup including packaged IC and supporting PCBs. (b) Hardware block diagram. (c) Layout of the whole 65nm IC.

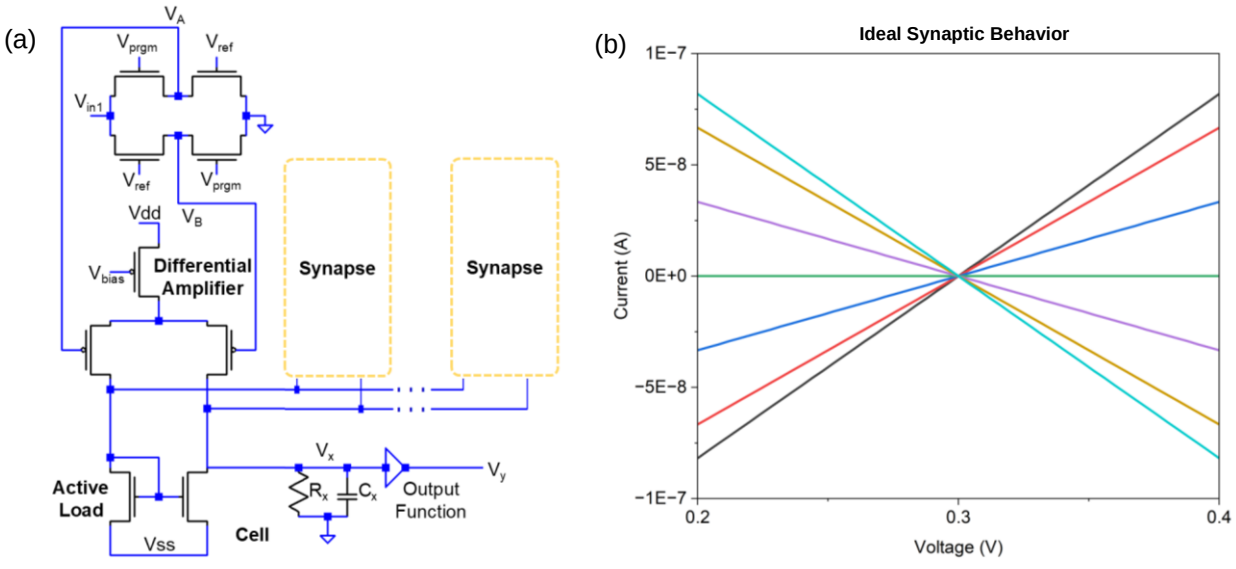


Figure 3.2: (a) FET Bridge synapse architecture and (b) ideal synapse performance.

To ensure low-power operation, each synapse's differential amplifier was given a 10nA biasing current. This sets an upper current limit for each synapse which in turn imposed a maximum power budget for each cell. Figure 3.2b shows the ideal performance of a single synapse across various positive, negative, and zero weights.

Once the FET bridge synapses were verified, the next step was to design the summing and integration blocks, the cell body or soma of the cell. To keep the architecture simple, a basic active load was designed capable of accepting current inputs from 19 synapses: nine each from the A-template and B-template along with an additional biasing synapse. The active load fed into the state resistor and state capacitor which via their intrinsic RC dynamics allowed for state integration from the current fed by the active load. The final functional block to consider was the output function which converted the cell state into the cell output. It was noted during simulations that the active load's output naturally saturated given a large enough input from the synapses. The output response followed that of the tanh function

Various simulations discussed in Chapter 2 using the digital twin compared the tanh output function to the ideal piecewise function to understand potential performance implications. These simulations suggested that the tanh function was sufficient to ensure bounded output convergence of the cells in the network, instead of adding on additional circuitry that would introduce further complexity and power draw. This was an important step to ensure that the tanh output function would in fact be suitable for use in the hardware network without degrading performance significantly.

Once the cell was designed, it was copied over 25 times to build a prototype 5×5 network. Network level simulations followed individual cell simulations to validate the design and connectivity between cells. Running these sims also demonstrated that the tanh function was

indeed a close approximation to the ideal piecewise output function and elucidated network metrics in terms of processing speed and power requirements.

Cadence Virtuoso is used for schematic level circuit design whose netlist is extracted and simulated using Spectre to validate performance. The TSMC 65nm process design kit (PDK) includes a variety of device simulation models that are based off real-world measurements and performance. Using these models, initial network simulations were performed at room temperature (300K) with no mismatch enabled to observe ideal circuit behavior, as shown in Figure 3.3. Once the design was verified, Monte Carlo simulations with carried out to observe the effect of process and device-level mismatch on the performance from ideal conditions.

After several rounds of adjustments to minimize the difference between mismatched devices and ideal, the design was ready for layout. Layout is the process of taking the abstract circuit design and physically laying out the corresponding transistors, wires, and other circuit components onto a predefined area representing the physical IC. While laying out, certain parameters need to be kept in mind, including size of devices used, device position matching to reduce performance mismatch, and length of wire runs which introduces parasitic resistance and capacitance. All these parameters can affect circuit performance of the final chip and need to be carefully considered. To account for some of these factors, “post-layout” simulations are run to observe how these influence simulated IC behavior. While this is not a perfect representation of the final chip behavior, it still provides valuable insight and improves confidence in the circuit design and respective layout before finally being sent to the fabrication company for tape out.

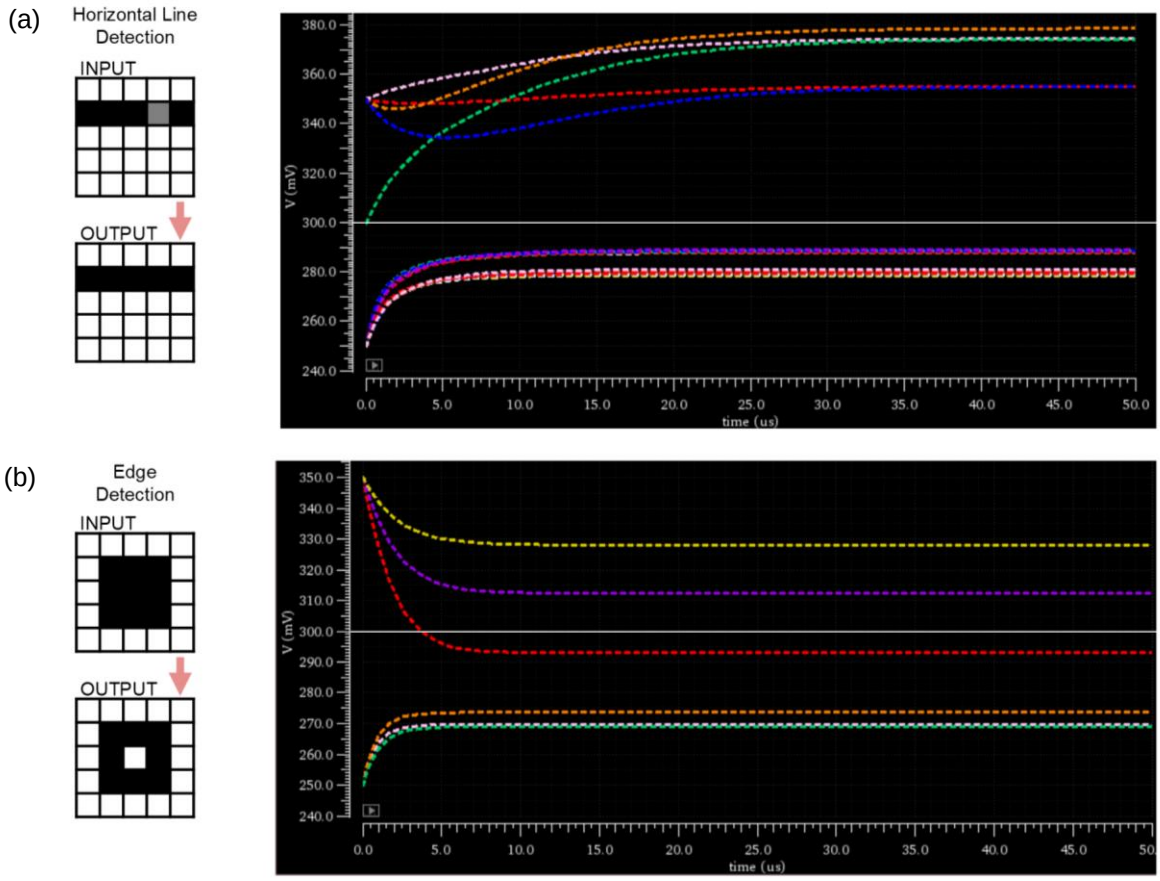


Figure 3.3: (a) Horizontal line detection SPICE simulation showing cells converging within 40µs (25K FPS). (b) Edge detection SPICE simulation showing cells converging within 10µs (100K FPS). Simulations run using 5×5 network designed with TSMC 65nm PDK components. Tanh output function is used with 300mV set as pixel state threshold.

Due to the size constraint of the chip, only so many IO pads could be included on the periphery of the IC. These pads are the chip-level interface on which input signals and output signals can be written and read via. We estimated that only around 90 pads with a pad pitch of 60µm would be able to fit on the 1mm×3mm die. For true parallel IO, each cell in the network would require one pad for direct input and one pad for readout, resulting in 50 pads used for just network IO. However, programming signals and signals to other supporting circuitry required several pads and it became clear during the design process some of the IC signals would have to be time multiplexed. The network input was allowed the full 25 pads, but an array of 8:1 multiplexors was designed to multiplex each column of the network. Since each column is composed of five cells, five of the multiplexor inputs were connected to their respective cell's output, and the remaining three inputs were connected to either test circuits or ground. This resulted in cutting the output pad requirement from 25 pads, down to five. By connecting the select lines of all the multiplexors and using an ADC array connected to the five output pads, this enabled the read out a single row of a network concurrently. Through this pad optimization, the final number of pads taped out on the chip was 88.

While the chip was being designed, supporting circuitry was designed in parallel, in the form of custom PCBs. Microcontroller Firmware and PC software were also written to allow for seamless interfacing between the IC and the HyE program that in turn allowed the supporting circuitry to drive the IC for testing and measurements. Post fabrication, the IC was wirebonded and packaged into a standard Quad Flat Package (QFP) which was then soldered directly to a "IC PCB" which provided a socketing mechanism via through hole mounts to the primary motherboard PCB which hosted all the supporting peripheral circuitry to drive the chip. The

measured IC along with the fully assembled supporting PCBs and microcontroller are shown in Figure 3.1.

3.3 Results

SPICE simulations within Cadence Virtuoso verified the network topology on various applications as seen in Figure 3.3 before the chip was sent out for fabrication. Before image processing applications could be run on the network, synaptic weights and synapse reprogrammability had to be verified. Hardware measurements of the synapses and cells are shown in Figure 3.4a demonstrating the measured synaptic mismatch across four different test cells. Following that, cell state voltages had to be defined. For this work, 300mV was used to represent a state of 0, corresponding to a neutral grey state, 325mV was used to represent a state of 1, the black pixel value, and 275mV was used to represent a state of -1, the white pixel value. While the voltages of the cell states were analog, oscilloscope readings on the outputs of the cells made it clear that there was too much noise to accurately gauge finer analog states as measured in Figure 3.4b, with a calculated signal-to-noise ratio (SNR) of 18.6dB, posing another hardware challenge that had to be accounted for.

Table 3.1: IC Hardware figure of merits compared to prior hardware implementations.

	FET Bridge (This work)	**ACE16K [32,33]**	**SCAMP-5 [34,35,36]**	**MIPA4K [37,38]**
Transistors/Cell	132	198	174	1500
Technology Node	65nm	350nm	180nm	130nm
Memory	Analog	Analog & Digital	Analog & Digital	Analog & Digital
Area/Cell	260 μm^2	5549 μm^2	977 μm^2	4392 μm^2
Estimated Power/Cell	1.39 µW	180 µW	18.8 µW	6.50 µW
Tested Image Processing Speeds	>20K FPS (as tested)	>1K FPS	100K FPS	>100K FPS

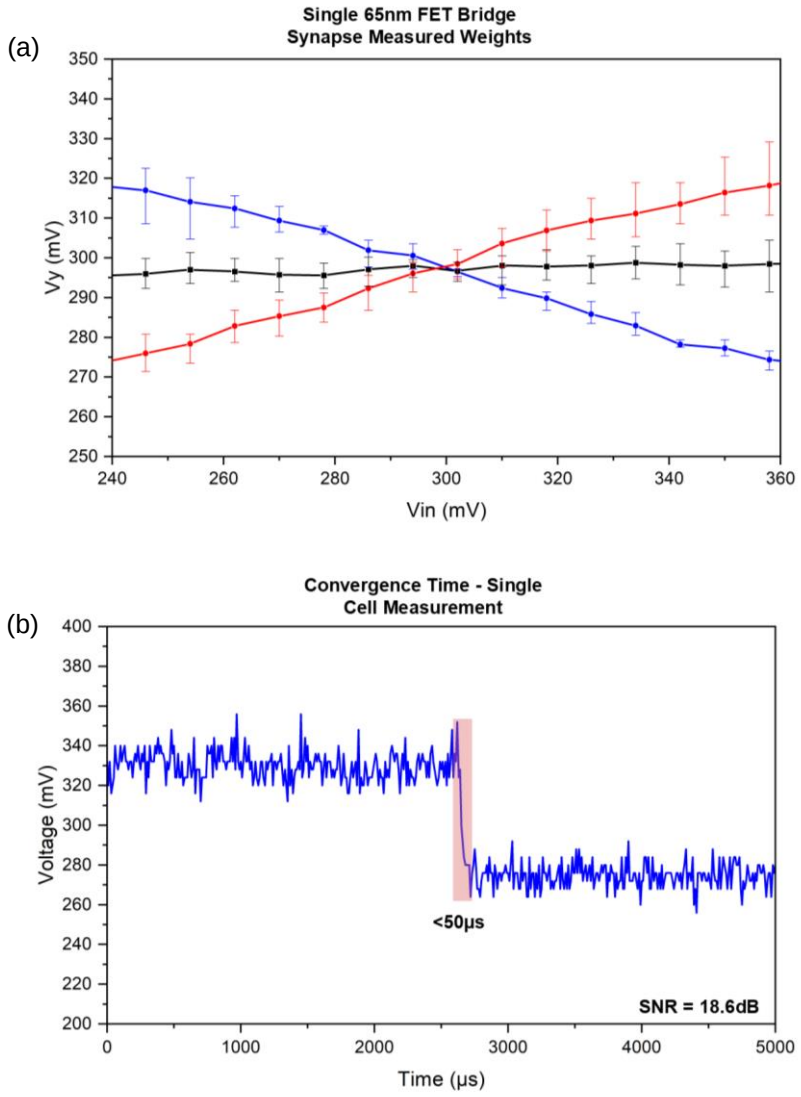


Figure 3.4: (a) Synaptic mismatch across hardware cells. (b) Hardware network single cell output measurement showing 50µs convergence time (20K FPS) for edge detection.

In light of this, the software interface was written to interpret ranges of analog voltages as one of three states: black, white, or grey with user adjustable thresholds between the states in an attempt to accommodate for the noise and mismatch present in the IC. The analog CMOS network hardware was demonstrated running edge detection and vertical line detection showing convergence times under 50µs. The measured input and output for these tasks are shown in Figure 3.5. Furthermore, our pure CMOS implementation is compared to prior implementations in Table 3.1 comparing network processing speeds, along with other pertinent metrics such as transistor count per cell and estimated power draw.

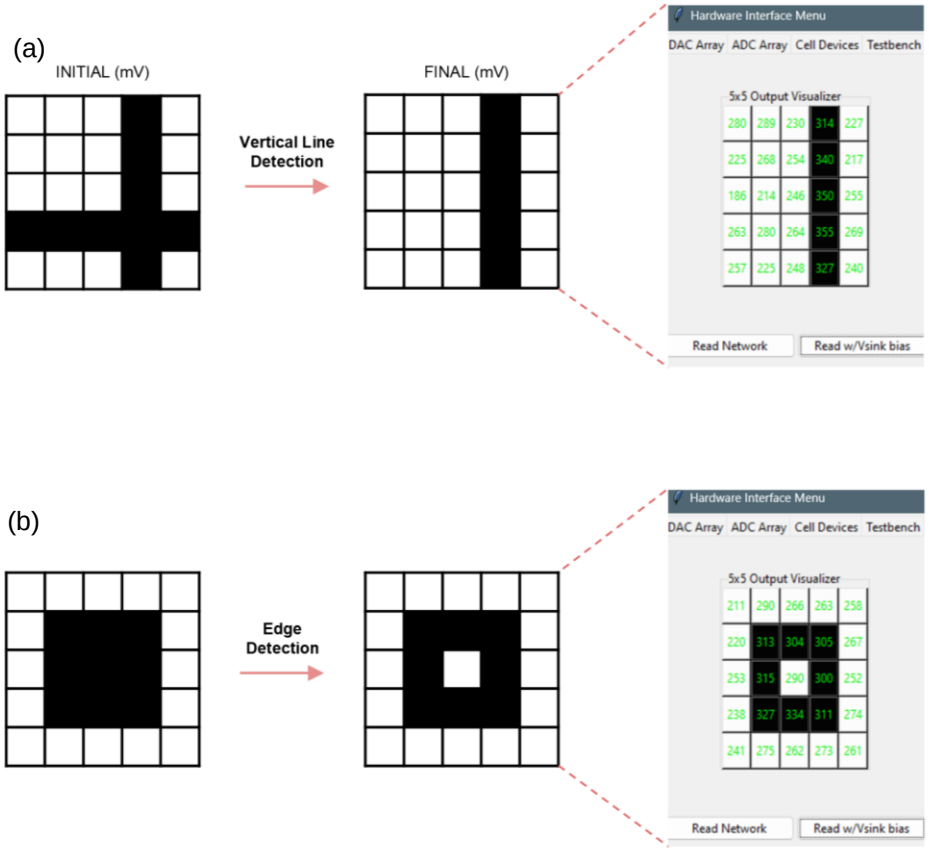


Figure 3.5: (a) Vertical line detection using the IC hardware. (b) Edge detection using the same IC after reprogramming the synapses. Hardware interface cutout shows measured cell voltages.

3.4 Discussion

It is postulated that the noise observed within the IC is partially due to thermal noise, along with potential crosstalk from the digital select signals driving the output multiplexors. Furthermore, the narrow output range less than 100mV that the cells operate within inherently decreases SNR. While using a smaller voltage range helped reduce the power consumption of the cell, the detrimental effect of noise on small amplitude analog signals can be mitigated by increasing the output voltage range of the cells to improve SNR.

While decreasing the output range of the cells causes a decline in SNR, in general, it is observed that using a smaller operating range helps alleviate degeneration due to device-level mismatch, better maintaining the linear, analog multiplication expected of the synapses. Due to each cell containing 19 synapses, mismatch between the synapses accumulates quickly along with any mismatch in the active load from cell to cell which introduces further nonidealities in performance. To abate the effect of mismatch, especially in the case when certain cells were operating well outside of the nominal operating range of 275mV to 325mV, individual reference biasing could be used which allowed these problem cells to be brough back into the expected range and perform as expected. A module was included in the HyE that sampled the reference bias for each cell sequentially and would increase or decrease the reference value incrementally while monitoring the cell's output to fine tune its reference bias.

Using the design of the 65nm process node, a 28nm IC was designed to create a larger 8x32 CeNN. A significant challenge of this design, however, was getting input onto the IC due to pad constraints. Ideally, the input should be fed in parallel, as was the case with the 65nm design. However, the number of unique cells in this larger network exceeded the number of allowable IO pads on the die's periphery and required the use of time multiplexing and voltage

sampling of the inputs. While individual component functionality was verified, at the system-level, the chip was unable to move the necessary data to the input of the CeNN, rendering it unable to operate or produce meaningful results. The cause of this issue is expected to be in the sample-and-hold circuits that were used to hold incoming data for each of the cells.

For future tape-outs, better layout practices can be followed to further reduce noise and circuit-level mismatch. In analog design, various reference generators are designed to drive circuits and rely on the "matching" of transistors in terms of their width and length to provide accurate and precise reference values. In practice, however, physically matching these sensitive transistors is non-trivial due to variability introduced during the fabrication process. To address this fundamental issue in analog IC design, over the years various best practices have been implemented. Matched transistors are commonly placed physically closer to each other, and other techniques like common centroid layout [111] are used, wherein matched transistors have their centers aligned. Such techniques tend to improve matching and reduce variation between the devices. While we tried our best to ensure we followed these best practices for this IC, there are still several design and layout adjustments that can be made to improve the overall performance of the prototype chip.

3.5 Conclusion

This pure CMOS prototype demonstrated the feasibility of building a CeNN on an IC using common commercial fabrication techniques, while taking advantage of the parallel and analog nature of the network. Pre-layout simulations helped verify the initial design and the use of an alternative tanh cell output function instead of the standard piecewise function. Post-layout simulations provided insight on the R and C parasitics introduced into the design and their

impact on network performance arising from the physical layout and wiring of the circuits in preparation for tape-out.

Following the fabrication and packaging of the chip, parallel input and processing were successfully demonstrated and measured for edge detection and vertical line detection tasks by reconfiguring the weights using the same hardware network while operating at speeds over 20,000 frames per second. While this prototype demonstrated the hardware-level proof of concept, further improvements on the IC can be made to boost performance and functionality of the hardware network by further mitigating the effect of noise and device-level mismatch.

CHAPTER 4

MEMRISTOR-BASED CELLULAR NEURAL NETWORK

In this study, the simplified architecture of the pure CMOS CeNN was further expanded and adapted to introduce HfOx drift memristor devices into the hardware network at the board level. These drift memristors offer a variety of advantages, including low-power, non-volatile analog weight storage, and were used within the synapses interconnecting the cells. Custom PCBs were designed based off the network topology and sent out for fabrication by third party vendors. These PCBs were then populated with CMOS/Memristor hybrid circuitry using both commercially available components and memristive devices fabricated in-house. To ensure compatibility, memristor devices were wirebonded to a chip-carrier with a corresponding through-hole footprint on the PCBs, creating a bridge topology which allowed seamless integration between CMOS circuits and memristor devices. After validation of the individual PCB components, a small-scale 4×4 network was built, and a synaptic programming algorithm was devised showing high programming accuracy for each addressable memristive device. The HyE software on the PC drives the board-level setup and was used to demonstrate horizontal line detection on the memristor-based hardware. While simulations have been run in the past using memristive devices in CeNNs, to our knowledge this is the first time to date true memristor-based CeNN hardware computation has been demonstrated, providing valuable insight into the advantages of such a network along with the challenges attributed to designing and scaling up such a network.

4.1 Introduction

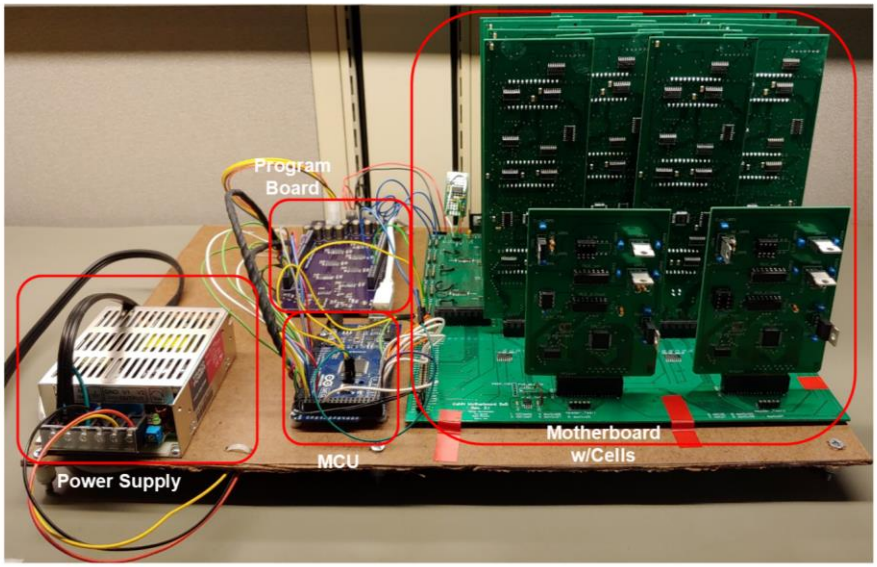
Although various CMOS implementations of the CeNN exist as discussed in previous chapters, our proposed architecture based off a memristor synapse design offers several advantages and further improvements: Reduced circuit complexity, improved multiplier linearity, and better power efficiency. Previously, non-volatile "drift memristors" have proven to be great candidates for low-power weight storage, resistive state retention, and intrinsic linear vector matrix multiplication (VMM) via Ohm's law [52], [57], [61], [62], [63], [64], [65], [66], [89] owing to their high switching endurance and repeatable switching behavior. These devices have also been measured to operate at a wide variety of temperature ranges with effectively no degradation to their non-volatile storage capabilities [57], further exhibiting their robustness. These qualities make them promising synaptic devices for neural networks and provide motivation to use these devices within the board-level CeNN setup shown in Figure 4.1a.

Simulations suggest the proposed architecture based on these devices maintain high-speed convergence times as low as 6μs [13] while requiring significantly fewer transistors than prior CMOS designs as reported in Table 4.1. Memristor-based synaptic weight storage within the network would provide a compact and low power synapse solution all while allowing the CeNN to be reconfigured to allow a single SoC to handle a multitude of tasks.

Table 4.1: Transistor and voltage line count based on CeNN topology.

CeNN Architecture:	**Memristor Bridge Topology**	**Pure CMOS Topology**
# Transistors/Cell:	132	78
# Active voltage lines during inference:	22	5

(a)



(b)

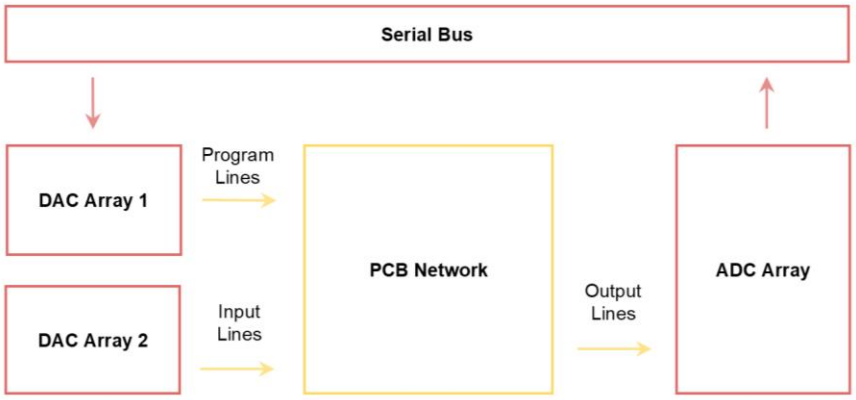


Figure 4.1: (a) Memristor-based CeNN board-level implementation. (b) Block diagram of hardware implementation.

For the CeNN, synaptic weights between cells would depend on the programmed resistance states of the memristors. Generally, prior memristor-based works used topologies in which devices could be individually addressed and programmed using a "write-read-verify" feedback loop [64], [65], [66], [67], [68], [112]. A programming pulse train of "SET" and "RESET" pulses applied to the Top Electrode (TE) of the two-terminal device modulates the resistance state of the device. SET pulses allow the device to modulate into a Lower Resistance State (LRS) using a compliance current (CC) limit to hit a desired state. In other words, for Ta/HfOx based memristive devices, the programmed state is directly correlated to the CC applied during the SET pulse: the higher the CC, the lower the resistance of the device observed. This CC limit is usually provided by a transistor with a controllable gate voltage which can also be completely turned off to mitigate sneak-path currents while other devices are being programmed. Both these reasons provide motivation for one transistor, one memristor setups (1T1R) that have shown robust operation and successful network inferencing for a variety of different applications.

On the other hand, the RESET pulse does not depend on the CC limit and effectively "resets" the state of the device to a High Resistance State (HRS). A small amplitude READ voltage is applied to the device after each SET/RESET pulse, and the resulting current is used to verify the state of the device before the programming algorithm decides whether the desired state has been hit or if another SET/RESET pulse is required. By using this programming scheme, the drift memristor can be fine-tuned, allowing for greater than 5-bit precision [67], [68].

At the semiconductor level Ta/HfOx resistance switching is explained by a combination of oxygen deficiency and metallic filament formation [57] in the conduction channel. As its name suggests, the device is essentially a stack of tantalum, followed by a hafnium oxide

dielectric, and finally an inert platinum layer visualized in Figure 4.2a. After the device is initially fabricated, it undergoes a "FORM" step, where a higher amplitude positive voltage is applied to the tantalum top electrode (TE) of the device and the platinum bottom electrode (BE) is grounded. This causes Ta to oxidize into Ta^{x+} cations. The concentration gradient of tantalum along with the electrical field due to the applied positive voltage causes the cations to diffuse and dope the HfOx layer. Simultaneously, O^{2-} anions in the oxide layer are attracted towards the top tantalum layer, resulting in the introduction of oxygen vacancies in the oxide layer. The movement of these two ions contributes to the formation of a localized conduction channel, shown in Figure 4.2b, which is responsible for lowering the resistance state of the device.

When a negative RESET voltage is applied after this forming process, the corresponding electric field reverses the ionic migration, moving O anions towards the platinum BE, while also causing the Ta cations to move back towards the tantalum TE. By reducing the oxygen vacancies and the amount of Ta cations in the switching layer, the device's resistance state increases, returning it to a High Resistance State (HRS). Following the RESET pulse, a positive SET voltage can be applied to the TE and this undoes the reset process described during the forming step, resulting in a device with a lower resistance state once again. By controlling the amount of current allowed to pass through the device via the CC limit, the extent to which ionic migration occurs, and in turn, the conductive channel composition can be modulated, giving rise to a range of programmable resistance states in the device.

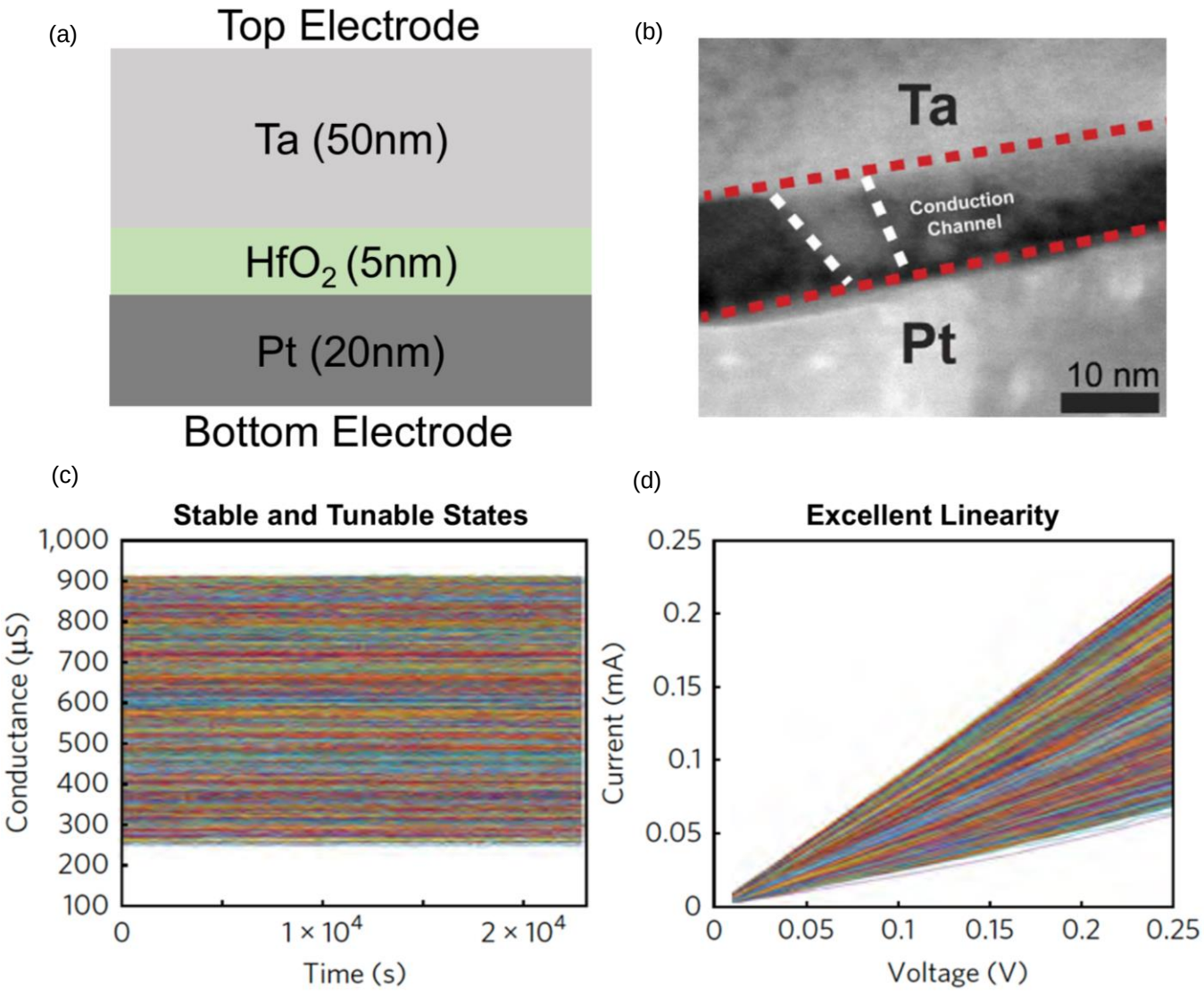


Figure 4.2: (a) Ta/HfO_x device stack (not to scale). (b) Localized conduction channel observed directly via STEM. (c) A single device capable of analog tuning and multiple stable states. (d) Highly linear operation of the device. Images courtesy of [57] and [66].

4.2 Methods

All PCBs were designed using Altium Designer and fabricated via third-party vendors including Sunstone Circuits and OSH Park, LLC. This board-level implementation, shown in Figure 4.1, utilized off-the-shelf CMOS components, along with memristors that were fabricated in-house. Several PCBs were used to build the network to ensure modularity and easy debugging, and all boards were assembled and soldered in-house. Each cell circuit was an individual PCB and was verified independently using custom testing circuitry. The cell architecture itself was based off the CMOS IC, but also included an operational amplifier circuit on the output of cells. This circuitry served two important functions: First, it enabled the implementation of the piecewise output function that converts the cell state V_x into the cell's bound output V_Y. Second, it served as a voltage-level shifter to ensure all memristors were operating within their nominal voltage range to prevent perturbing their set states during inferencing and served as overvoltage protection to maximize the devices' lifespan. This was achieved by using a rail-to-rail amplifier whose output range was set by its power voltages. Using the pre-determined gain of the amplifier, the input voltage within a specific range was allowed to pass through unaffected but would saturate and "clip" the voltage if the magnitude was too large in either the positive or negative direction, effectively recreating the piecewise function behavior modelled by Equation 2.2. The clipping stage was then followed by the level shifter stage, which utilized a voltage divider and voltage follower to keep shift the cell's output within a predefined range of -200mV to 200mV, the nominal operating voltage for the memristive synapses. Figure 4.3 shows an experimental demonstration of this output amplifier set up.

The largest PCB was the main motherboard which provided the interconnecting lines between cells and the necessary supporting power and processing signals. For input and output, two separate “I/O boards” housing DAC and ADC arrays were used that connected to the motherboard and wrote cell inputs and read their outputs, all while receiving commands from the host PC using an AtMega2560 microcontroller. An additional PCB was dedicated to programming the memristor devices on the cells using a custom programming scheme designed around the bridge synapse architecture as shown in Figure 4.4.

During the initial design phase of this board-level implementation, time was dedicated to exploring the various ways memristive devices could be integrated in the network in a meaningful and efficient manner. Prior simulated CeNN works have suggested using memristive devices within the cell’s integration node as well as within the synapses [13], [14], [76], [86], [87], [107], [108], [109]. Based on the characteristics of our labs Ta/HfO_x drift memristors, it was decided to design the hardware integrating them as part of a modified memristor bridge synapse [107], [108], [109]. Simulated memristor bridge topologies were demonstrated to be an effective way to build reprogrammable synapses that allowed for positive, zero, and negative weights. Non-volatile memristors were used for zero-power weight storage, and the introduction of a differential amplifier further allowed the synapse to perform analog multiplication based off Equation 3.1.

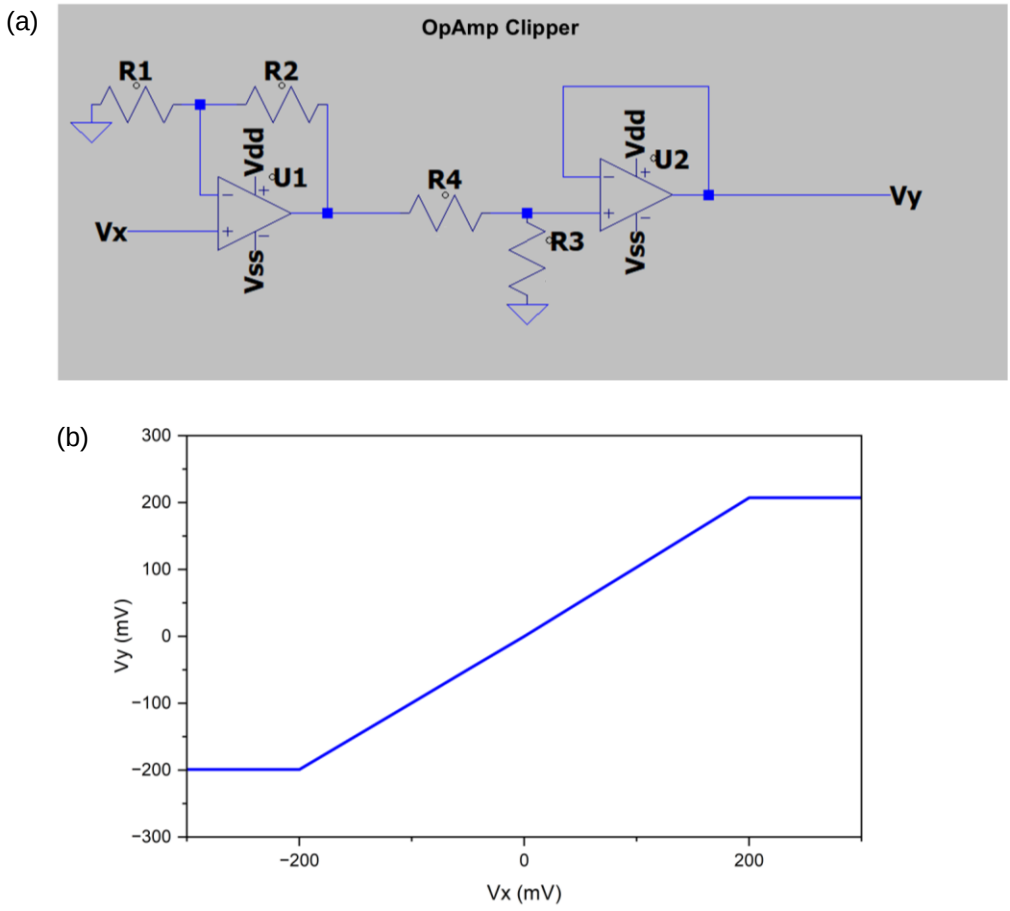


Figure 4.3: (a) Cell output function amplifier circuitry to ensure synaptic memristor devices were operating within their nominal voltage range. (b) Measured piecewise function performance.

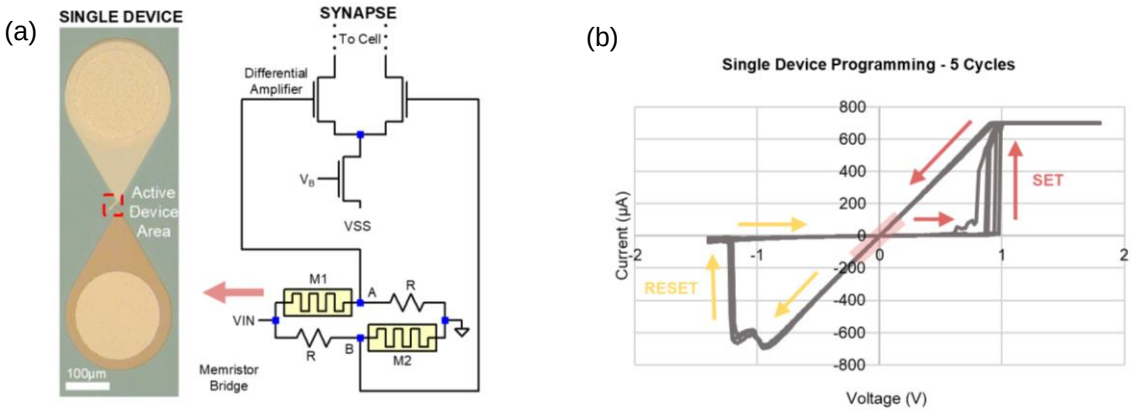


Figure 4.4: (a) Bridge synapse architecture used in the memristor-based CeNN implementation. (b) Single memristor I-V cycling, showing the SET and RESET process.

Similar to the FET bridge synapse, two voltage dividers form the bridge topology, as seen in Figure 4.4a. In our modified bridge, looking in from the input side, the first voltage divider is composed of a memristor and a series resistor R. The second voltage divider is designed in a similar manner, but has the components flipped, with the resistor in series with the memristor. The top voltage divider results in a corresponding voltage at node V_A and the bottom voltage divider produces a corresponding voltage at node V_B which are fed into the respective input terminals of the synapse's differential amplifier. The differential amplifier takes the difference in voltage between V_A and V_B and multiplies that difference by the internal gain of the amplifier, resulting in the amplifier's output current. Since the memristors within the synapse can be tuned in an analog manner, this enables the synapse to hold different synaptic weights: Adjusting the memristor's weights cause the node voltages V_A and V_B to change. Since the differential amplifier's output current is proportional to the difference between V_A and V_B, this implies that changing the memristors' state directly changes the weight stored on the synapse. The network operating range is then set correspondingly to ensure that once the memristor states have been set during the programming step, inferencing signals do not change their conductance levels.

Various programming schemes were tested during the design of the memristor-based network. Using two memristors and two standard resistors per synapse drastically simplified synaptic programming—The setup made it so that only one memristive device per branch of the synapse had to be programmed. In an attempt to further simplify the programming algorithm and hardware, a study was conducted to program the two memristors in parallel. Assuming that both memristors in a single synapse were set to the same state, a shared programming pulse was applied to both devices' TE and the CC was controlled by matched transistors at the BE with the same gate voltage applied as pictured in Figure 4.5. Though this parallel setup allowed for fast

programming of multiple devices, there was a tradeoff: Even while using the same compliance current for both devices, there was still some variation between the final state of the devices as described in Table 4.2, enough to degenerate network performance and ultimately this parallel programming scheme had to be shelved.

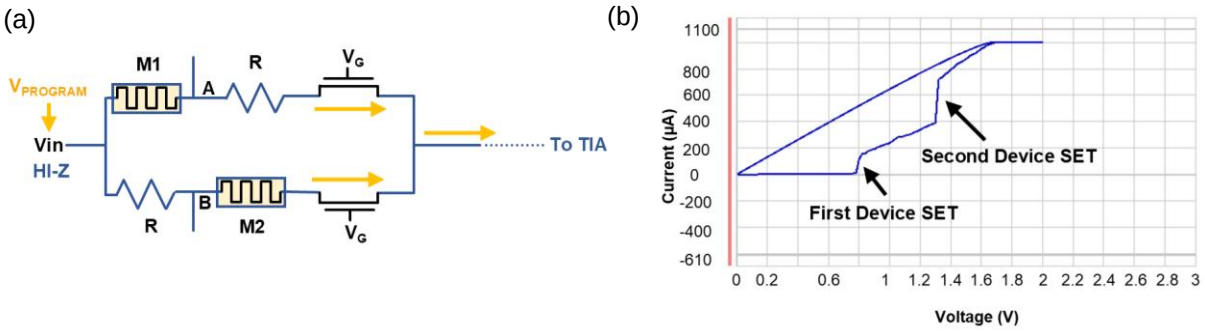


Figure 4.5: (a) Synapse parallel programming scheme programming two devices simultaneously. (b) I-V read from common TIA node as both devices are programmed.

Table 4.2: Parallel programming resistance state measurements of devices M1 and M2.

Gate Voltage V_G (V)	**M1 State (Ω)**	**M2 State (Ω)**
1	3030	1785
1.1	1612	1250
1.2	1109	1008

A serial programming scheme using the more traditional program and verify was then studied and implemented that was able to program the devices sequentially in a precise and accurate manner, shown in Figure 4.6a-d. The memristor in the top branch received its programming pulse to its top electrode via the Vin node and had its bottom electrode grounded via itself and its reference resistor, totally isolating the device while programming. Meanwhile, the memristive device on the bottom branch had its top electrode receiving programming pulses through the node between itself and its reference resistor, and its bottom electrode connected directly to the synapse's ground. A switch in the form of a standard CMOS transmission gate was implemented on the top branch of the bridge to further ensure that programming of the bottom device leads to no sneak-path currents traveling between the two memristors during programming. Sneak-path currents are a common issue studied within arrays of memristive devices where attempting to program a single device leads to unexpected and usually unwanted "sneak" currents through neighboring devices, potentially altering their set state and deteriorating the accuracy of device state readouts. The selector switch helped mitigate this by floating the devices that are not actively selected for programming. Once the devices were programmed, the selector switch closed, completing the bridge topology, and network inferencing began.

The resulting programming scheme had the advantage of being able to address each individual memristor in every cell, allowing for precise and accurate programming using the write-read-verify method. Programming of individual devices was achieved within a 2% tolerance of the targeted resistance state. Although a single device could be programmed precisely, a major drawback with this programming scheme was that it is performed sequentially, one device at a time. This drastically increased programming time for a larger network, but for the small-scale 4x4 memristor network prototype, however, this was a justifiable trade off.

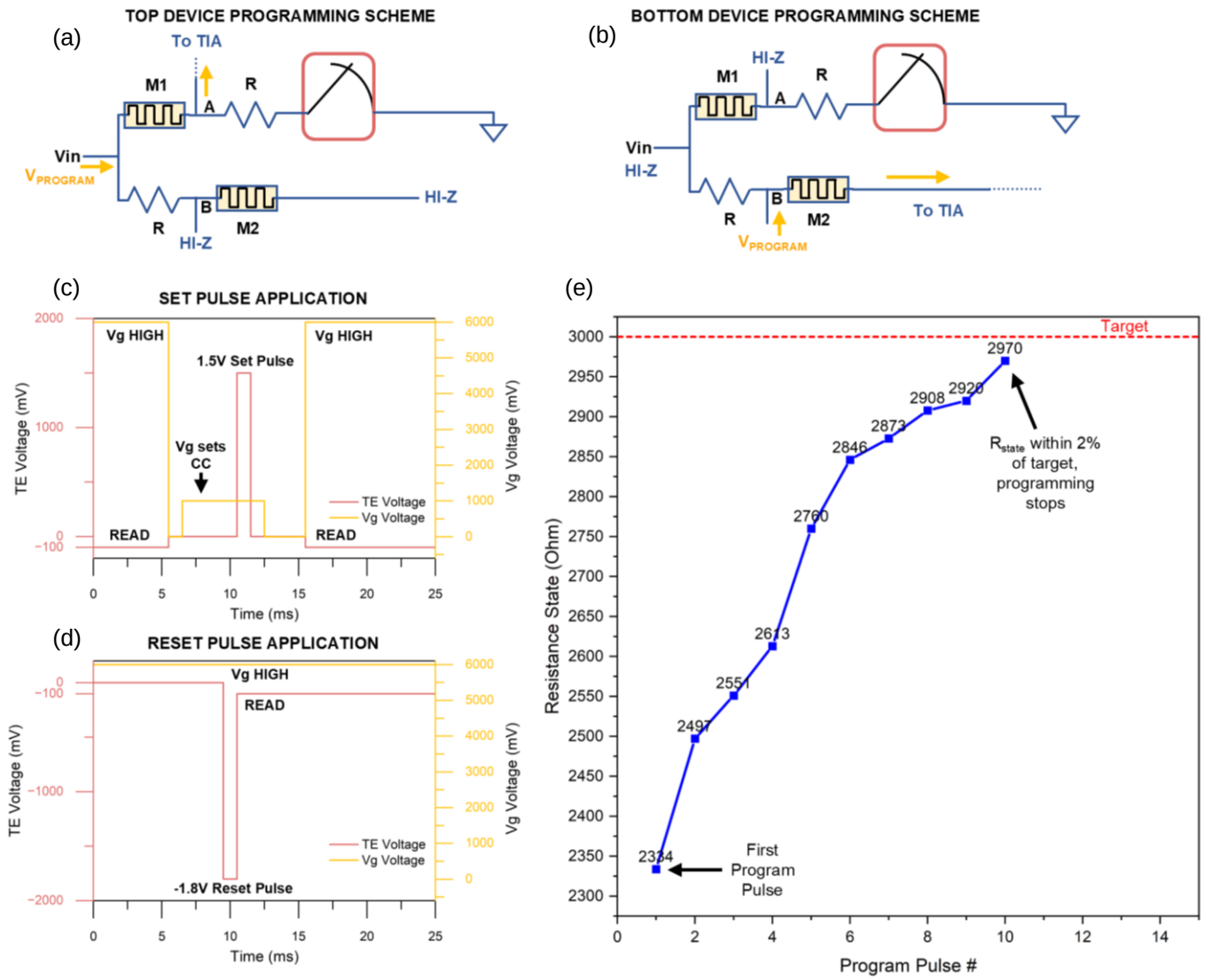


Figure 4.6: (a) Synapse sequential programming scheme for the top device. (b) Synapse sequential programming scheme for the bottom device. (c) SET pulse waveform showing TE voltage and gate voltage of CC transistor. (d) RESET pulse waveform showing TE voltage and gate voltage of CC transistor. (e) Experimental incremental programming using program and verify technique that enables programming resitance states within a tight tolerance.

Cold temperature measurements were also run on the memristor devices to study the behavior of the devices at temperatures as low as 82K. An MMR setup with a vacuum chamber was used to directly measure fabricated Ta/HfO_x drift memristors at low temperatures. The devices were placed within the vacuum chamber which achieved a vacuum of 1.6e-5torr during operation. The two terminals of a single device were probed and driven by a Keysight B1500 Semiconductor Analyzer which allowed for the verification of single device cycling and programming at low temperatures. The measured experimental data was then used to create drift memristor models for use in low-temperature SPICE simulations in Cadence Virtuoso to observe the simulated effect of temperature on the memristor bridge and verify its operation at low temperatures. MMR measurements temperatures as low as 77K were attempted, but the MMR setup did not physically allow temperatures below 82K. 77K is the condensation point of nitrogen, with liquid nitrogen being commonly used for cooling thermal cameras and vision sensors. Memristors operating nominally at these temperatures ensures CeNN functionality for such use cases.

4.3 Results

The programming algorithm was verified to allow device programming within a tolerance of 2%. Figure 4.7b shows various positive, negative, and zero weights that were experimentally achieved using the Ta/HfO_x memristor bridge with the serial programming scheme. HyE and SPICE simulations suggest that the high precision weight tuning of the memristive devices exceeded the requirements for synaptic bit precision within a variety of

CeNN tasks as reported in Table 4.3 and Table 4.4 and the 2% programming tolerance is within acceptable limits to minimize network convergence errors.

Device operation was also experimentally measured as low as 82K and a SPICE model was developed based off these measurements to create a bridge synapse SPICE simulation, with low-temperature synaptic performance shown in Figure 4.7c and 4.7d, showing virtually no degradation to analog programming and linear, non-volatile storage capabilities. Low temperature measurements on state retention and cycling for a single device are shown in Figures 4.7e and 4.7f. The cold temperature MMR measurements suggest that a significant portion of noise in these devices may be attributed to thermal noise, at least at HRS: During state retention tests, a general trend was observed where devices programmed to HRS and read at lower temperatures exhibited less variation from the average programmed value of the device as shown in Figure 4.8. Read variance was improved when the device was set to LRS as well, but it was not as significant as in the case of HRS: This is due in part because relatively small deviations in the small currents sensed at HRS from noise implies fairly large swings in the overall resistance state based off Ohm's law. Meanwhile, when the device is at LRS, proportionally larger deviations are required to show significant changes in the device state. This study indicates that lowering the temperature, however, seems to reduce the magnitude of state variation attributed to noise.

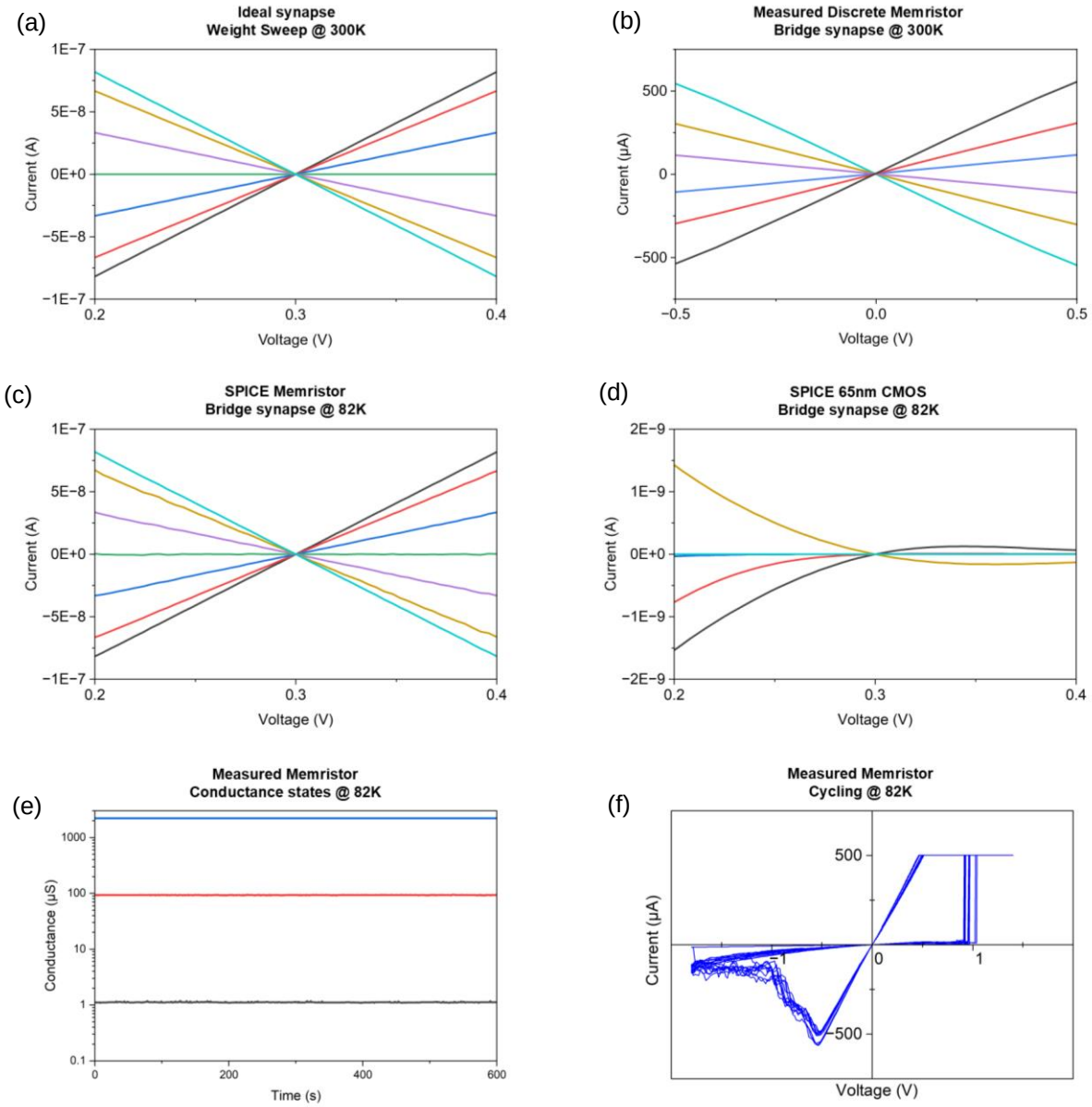


Figure 4.7: (a) Ideal bridge weight demonstration. (b) Experimental demonstration of positive, negative, and zero weights using memristor bridge synapses at 300K. (c) SPICE simulated 82K memristor bridge synapse performance using Ta/HfO_x memristor devices compared to (d) 82K FET bridge. FET bridge weights show drastic degradation. (e) Measured conductance states of a single device at 82K. (f) Measured I-V cycling of a single device at 82K over 10 cycles.

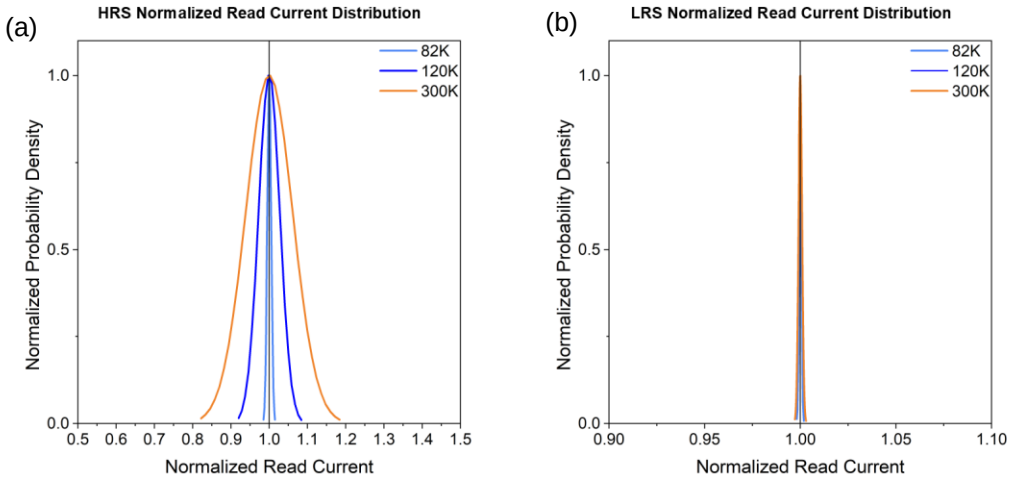


Figure 4.8: (a) Distribution of normalized current of devices set to HRS at various temperatures showing reduced variation at lower temperatures. (b) Distribution of normalized current of devices set to LRS at various temperatures.

Table 4.3: CeNN synaptic bit precision requirements by application.

Application	**Templates Required**	**Synapse Bit Precision**
Average	A & B	3-bit
Edge Detection	B	4-bit
LOGIC AND, OR, Difference	A & B	2-bit
LOGIC NOT	B	2-bit
Horizontal/Vertical Line Detection	A	3-bit
Noise Reduction	A & B	3-bit
Infill	A & B	3-bit
Global Connectivity	A & B	4-bit

Table 4.4: Average pixel error based on device programming tolerance for video edge detection.

1-sigma device tolerance (%)	**2**	**5**	**10**
Average Error (%)	0.18	3.01	14.1

Furthermore, reducing the operating temperature of the device not only allows the device to continue nominal operation, but prior works have also demonstrated that reduced temperatures tend to increase the state retention time of these devices as well [57]. This implies that the memristor's overall performance may be improved at lower temperatures, further supporting the use of these devices for applications that require cold-temperature operation, such as the use of infrared imaging sensors where the temperature of the sensor itself affects performance.

Once memristor programming was verified at the individual cell level, it was used to set up the appropriate templates across the network for HLD processing as shown in Figure 4.9. Processing was completed within one millisecond, but the true network speed is expected to be converging on the order of 100µs—The ADC arrays used for this board-level study bottlenecked the overall setup, introducing significant latency due to reading their input channels in a serial manner. With 16 active cells, 16 unique ADC channels were used and each channel was sequentially read.

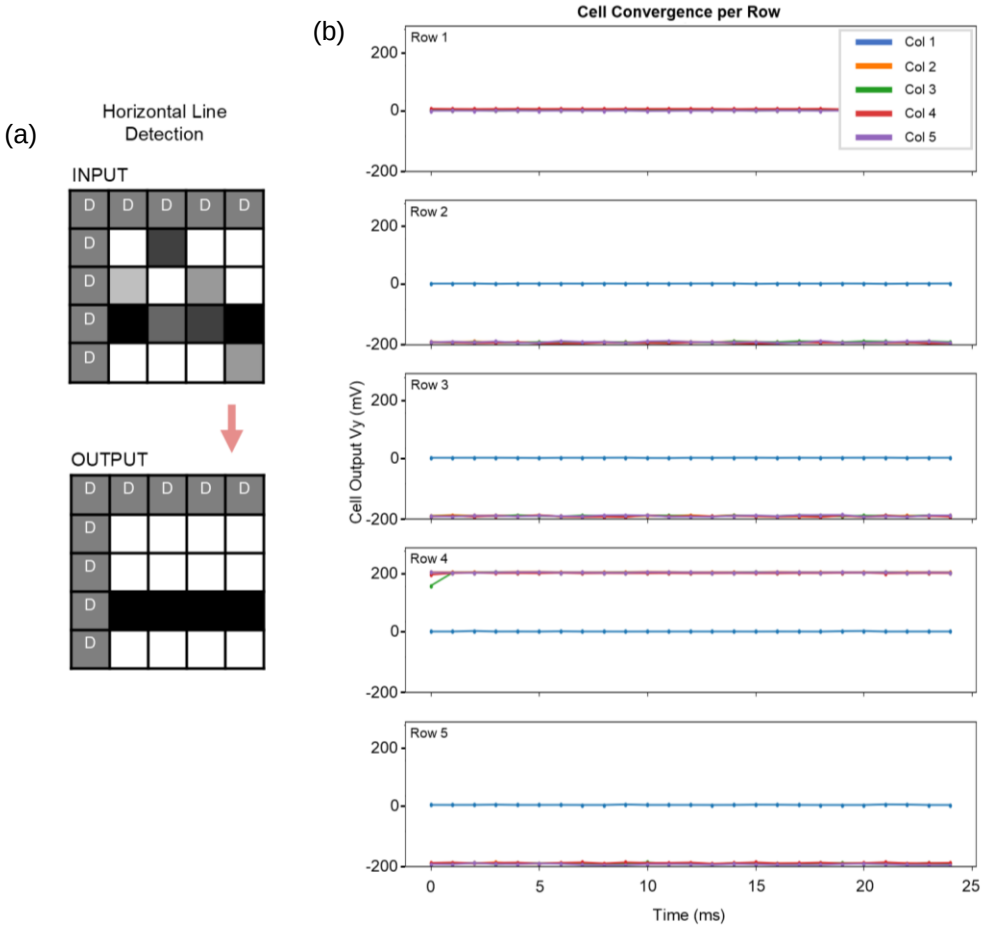


Figure 4.9: CeNN memristor-based hardware demonstration of the horizontal line detection. (a) Pixelwise input and output. (b) Visualization of individual cell outputs reaching convergence under 1ms. Each plot represents a row of the network, with different colored lines representing different column

4.4 Discussion

The board-level hybrid CMOS/memristor implementation showcased the advantages and limitations of using such a network. While the overall architecture was designed in a manner similar to the 65nm CMOS IC, major deviations from that design include the use of operational amplifiers for a piecewise cell output function and the introduction of passive resistors and memristors into the synapse architecture, replacing the pure CMOS FET bridge.

Ta/HfOx memristive devices have proven to be great candidates for a variety of synaptic applications in a variety of networks, including CeNNs, showing reprogrammable and robust operation. However, one of the primary challenges while building this board-level network was addressing individual devices and programming; Whereas the FET bridge-based network was able to simply use common voltage lines to set the synaptic weights by biasing the MOSFET gates in a parallel manner, programming the devices in the memristor bridge required additional dedicated hardware in the form of the current compliance FET, addressable multiplexors, and TIA, increasing the overall algorithmic complexity and slowing down programming time compared to the FET bridge synapse. Even with these limitations, the zero-power, analog weight storage, linear cold temperature operation, and intrinsic radiation hardness make this class of memristors suitable for use as non-volatile storage devices in a wide variety of different environments that an edge network, like the CeNN, may be deployed in. As CeNN tasks are defined by the synaptic weights of the network, this is critical to ensure proper network operation in extreme environments.

It should also be noted that the weights in this study assumed that both memristors were tuned to the same resistance state. While this did allow for positive, negative, and zero weights, uncoupling the state of the drift memristors in a single synapse would allow for finer tuning of

the synapse weights, increasing synaptic bit precision and range of achievable weights that these synapses can use.

The programming scheme chosen in this study emphasized precision and accuracy over programming speed, but as the network scales, programming speed will become a critical parameter to consider in order to ensure that the network can be programmed within a timely manner. Along with programming the network in a shorter period of time, high-speed programming opens up an avenue for a single network to handle more complicated tasks that are an aggregate of several simpler tasks: One task would be run to completion and the network output could be fed back as the input of the network to process a second task sequentially, effectively allowing for back-to-back "stacking" of different tasks.

4.5 Conclusion

The first of its kind demonstration using the board-level CeNN to create a hybrid CMOS/memristor architecture was experimentally verified, demonstrating sub-millisecond processing of the horizontal line detection task. While most of the topology is similar to that of the 65nm CMOS IC network implementation, the integration of memristors into this network provided a platform to study and validate the use of memristors within the bridge synapse to create an efficient and robust weight storage topology. A standalone programming board and accompanying algorithm enabled the serial programming of network memristors within a tolerance of 2% of the target resistance state.

Furthermore, setting the weights based off the simulations using the CeNN digital twin validated the programming of the devices and operation of the network for horizontal line detection. These findings show the feasibility of such a hybrid memristor/CMOS hardware

network and may pave the way for robust edge networks that could be used in a variety of extreme real-world scenarios, such as aboard a satellite in space for low-power image processing.

Although the readout using off-the-shelf ADC components posed as the major bottleneck to the overall speed of the network, this board-level prototype validated both the hardware and driving software for network inferencing as well as synaptic device programming. Next steps would be to move from this verified board-level prototype to an IC tape-out, allowing for a larger network and memristors integrated into the architecture as part of the back end of line (BEOL) features.

CHAPTER 5

SUMMARY AND FUTURE RESEARCH DIRECTION

By taking advantage of massively parallel, real-time analog computation, the CeNN significantly improves processing speeds and energy efficiency for a variety of vision tasks, as well as computational modeling tasks to model real-world problems like PDEs. The re-programmability of network weights allows a single IC to handle a multitude of tasks saving area and provides a robust solution for low-power edge-computing applications. Furthermore, the local connectivity of cells simplifies circuit topology and reduces on-chip area while simultaneously better modeling what is observed in biological systems.

Chapter 2 presented the comprehensive HyE GUI package that was built from the ground up and provides a robust digital twin simulator for rapid testing of different CeNN applications. The GUI also provided a custom interface for the fabricated hardware, allowing for READ and WRITE commands to be sent to the physical network during hardware operation. The digital twin provided a discrete time approximation for real-world hardware performance and was used to process image and videos for a variety of tasks, including edge detection, object tracking, global connectivity, and hole filling. Moreover, the floating-point precision offered by the digital twin enabled the use of more complex computational tasks, including 2D partial differential equations modeling.

Chapter 3's study on implementing the CeNN using a traditional CMOS process node demonstrated the feasibility of creating a network using commonly available fabrication techniques. The IC itself was composed of a pure analog network with simplified architecture, that was interfaced with DACs and ADCs via external peripheral circuitry for network I/O. The HyE GUI interface provided a human readable means to read and write to the hardware to set

biases, weights, and the input for processing as well as a platform that easily visualized the network output once all cells had converged. Hardware re-programmability and high-speed processing surpassing 20,000 FPS was demonstrated on tasks including edge detection and VLD/HLD on a single chip.

Finally, Chapter 4 discussed the successful integration of emerging drift memristor devices along with standard CMOS circuitry at the board-level in a study to prototype a first-of-its-kind real world memristor-based CeNN. HfOx drift memristors were used to provide a zero static-power, non-volatile means of synaptic weight storage and had the additional benefit of improving the linearity of analog weight multiplication of the synapses. Furthermore, a programming algorithm was devised that allowed programming these devices within a 2% tolerance, enabling precise tuning of synaptic weights. Board modularity provided a means for easy validation and debugging of components before full system integration. Using this board-level network, HLD was successfully demonstrated converging on the order of 100s of microseconds.

This dissertation's findings contribute new insight on the retinally-inspired cellular neural network, emphasizing network operation across a variety of platforms, ranging from the simulation-level digital twin and SPICE circuits to the hardware-level pure CMOS IC and board-level memristor-based prototype to experimentally demonstrate network programming and computation. These works provide direction and guidance for future research and innovation in the field of CeNNs, and more broadly, neuromorphic computing.

Several avenues of research have been explored here, but future research on the hardware CeNN would include further improving the IC hardware discussed in Chapter 3, including increasing the operating speed at the IC level, while scaling up the network size. This can be

achieved by further reducing routing parasitics and improving the SNR of the cells by using a larger operating range and better optimizing the overall IC layout. Additionally, to bypass the limitation imposed by the relatively small number of pads to interface with the internal circuits on the IC, photodiode sensors could be integrated on-chip as well to reduce the number of pads required simply input into the network. If a single sensor was integrated along with a single cell, this would enable true parallel input, removing all latencies associated with using a multichannel DAC to produce network input. Such a chip would also be able to react in real-time to the changing inputs, with dynamic scenes being sensed, providing more insight into the CeNN's operating principles and hardware dynamics.

Along with scaling up the network to allow for larger tasks to be processed, memristor integration into the IC would be another avenue for future research. While Chapter 4 presented a board-level memristor-based network, the next major milestone would be to design and fabricate an IC using these devices. Memristor integration could be a BEOL process after CMOS components are fabricated at a traditional semiconductor foundry. Though the serial programming scheme discussed in Chapter 4 may be useful for smaller-scale IC networks, it will be imperative to further develop this algorithm to allow faster serial programming or develop more precise parallel programming schemes. This both ensures that the network can be programmed in a timely manner, but also potentially opens the ability to process aggregate tasks that depend on a combination of simpler CeNN tasks in real-time, enabling more complex applications.

Implementing memristors into the hardware implementation may also allow for higher synaptic bit precision. With higher precision, it would be possible to run a wider variety of tasks using the hardware, including partial differential equations solvers, that take advantage of the

time-continuous nature of the cell dynamics, which would allow for better efficiency while computing as well as improve solution accuracy as it bypasses the limitations of using discrete numerical approximations to model such problems.

In a similar vein, our lab was also able to realize preliminary 3D CeNN simulation in an effort to compute the solution to 3D PDEs, namely the 3D Navier-Stokes equations that models fluid flow in the lid-driven cavity problem shown in Figure 5.1. Such a network would allow for real-time processing using three-dimensional weighted templates, with cells communicating with their 26 immediate neighbors in 3D space. While this network takes on the complexities of being connected to more neighbors than the 2D CeNN, it drastically improves the networks capabilities and would have profound implications in fields such as aerospace and medical imaging among other domains that rely heavily on processing in 3D space. As our understanding of the 3D network improves, a hardware realization may be feasible as 3D synapse topologies are explored and fabrication techniques improve.

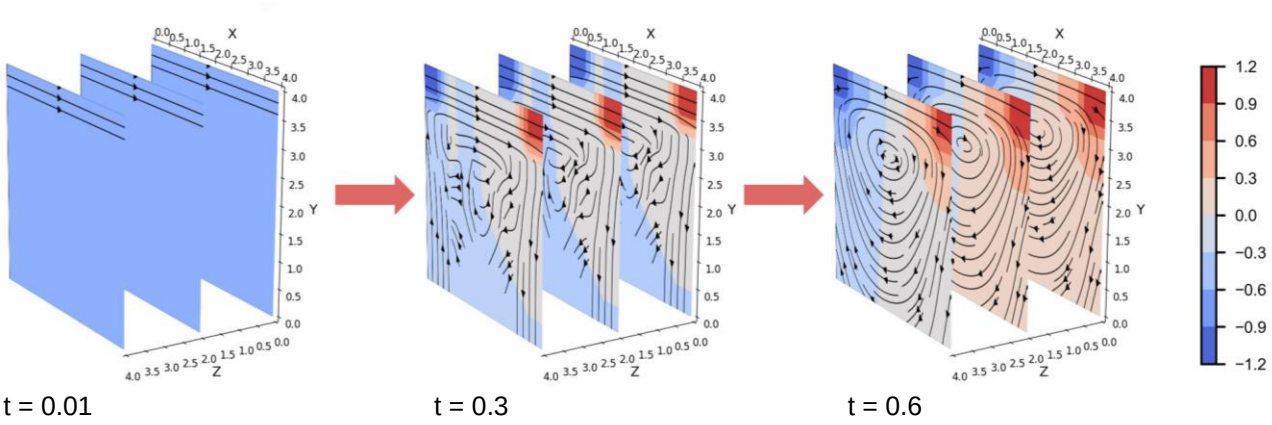


Figure 5.1: Simulated CeNN solver for 3D Navier-Stokes equations modeling the lid driven cavity problem showing a fluid's flow direction and velocity. Processing at different time steps is shown, with warmer colors representing areas of high pressure and cooler colors representing lower pressure. This simulation demonstrates proof of concept of a 3D network.

Network inferencing can be further optimized by incorporating weight training and learning algorithms within the CeNN [99], [100]. While the templates allow for generalized processing tasks, certain scenarios like low-lighting conditions for image processing, may require the network to adapt to allow adequate output results. This may include adjusting the weights or cell biasing to allow cells to more easily converge to their correct steady state, reducing pixel errors and noise. Furthermore, an adaptable CeNN would also be able to serve as a filter for image preprocessing during object recognition tasks. By cascading a frontend CeNN with a backend classifying deep neural network [81], [82], [83], the CeNN would allow for a variety of image transformations that improve the classification accuracy of the classifying network. Chapter 2 discussed a simple MNIST model that used this setup, but more complicated processing applications can be explored to take advantage of such a connected system.

The Cellular Neural Network has demonstrated to be a robust and practical neural network capable of a variety of different vision and computation tasks. The outlined future research directions take advantage of cutting-edge hardware advancements, including emerging nanoscale memristors, along with massively parallel algorithms and computational techniques that show promise to provide new insight into network implementations. This thesis provides a foundation on CeNN implementations on which future research can build upon. Continued development in this area may usher in a new generation of parallel, efficient computing using a true CeNN universal machine.

BIBLIOGRAPHY

[1] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," 2020, *arXiv*. doi: 10.48550/ARXIV.2005.14165.

[2] R. Thoppilan *et al.*, "LaMDA: Language Models for Dialog Applications," 2022, *arXiv*. doi: 10.48550/ARXIV.2201.08239.

[3] E. Talpes *et al.*, "Compute Solution for Tesla's Full Self-Driving Computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar. 2020, doi: 10.1109/MM.2020.2975764.

[4] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

[5] Z. Zhong et al. "High Performance Offline Handwritten Chinese Character Recognition Using GoogLeNet and Directional Feature Maps" 2015 13th International Conference on Document Analysis and Recognition (ICDAR)

[6] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," 2021, *arXiv*. doi: 10.48550/ARXIV.2104.10350.

[7] S. J. B. Yoo *et al.*, "Towards Reverse-Engineering the Brain: Brain-Derived Neuromorphic Computing Approach with Photonic, Electronic, and Ionic Dynamicity in 3D integrated circuits," 2024, *arXiv*. doi: 10.48550/ARXIV.2403.19724.

[8] Y. Huang, V. Ravichandran, W. Zhao, and Q. Xia, "Towards Energy-Efficient Computing Hardware Based on Memristive Nanodevices," *IEEE Nanotechnology Mag.*, vol. 17, no. 5, pp. 30–38, Oct. 2023, doi: 10.1109/MNANO.2023.3297106.

[9] J. Von Neumann, "First draft of a report on the EDVAC," *IEEE Annals Hist. Comput.*, vol. 15, no. 4, pp. 27–75, 1993, doi: 10.1109/85.238389.

[10] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[11] R. H. Dennard, F. H. Gaensslen, Hwa-Nien Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions," *Proc. IEEE*, vol. 87, no. 4, pp. 668–678, Apr. 1999, doi: 10.1109/JPROC.1999.752522.

[12] M. Schulz, “The end of the road for silicon?,” *Nature*, vol. 399, no. 6738, pp. 729–730, Jun. 1999, doi: 10.1038/21526.

[13] L. O. Chua and L. Yang, “Cellular neural networks: theory,” *IEEE Trans. Circuits Syst.*, vol. 35, no. 10, pp. 1257–1272, Oct. 1988, doi: 10.1109/31.7600.

[14] L. O. Chua and L. Yang, “Cellular Neural Networks: Applications,” IEEE Transactions on Circuits and Systems, vol. 35, no. 10, pp. 1273–1290, 1988. doi:10.1109/31.7601

[15] L. O. Chua and T. Roska, Cellular Neural Networks and Visual Computing: Foundations and Applications. Cambridge: Cambridge University Press, 2005.

[16] C. Mead, “Neuromorphic electronic systems,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990, doi: 10.1109/5.58356.

[17] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, “Physics for neuromorphic computing,” *Nature Reviews Physics*, vol. 2, no. 9, pp. 499–510, Jul. 2020, doi: 10.1038/s42254-020-0208-2.

[18] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, “Opportunities for neuromorphic computing algorithms and applications,” *Nat Comput Sci*, vol. 2, no. 1, pp. 10–19, Jan. 2022, doi: 10.1038/s43588-021-00184-y.

[19] J. B. Aimone *et al.*, “A review of non-cognitive applications for neuromorphic computing,” *Neuromorphic Computing and Engineering*, vol. 2, no. 3, p. 032003, Sep. 2022, doi: 10.1088/2634-4386/ac889c.

[20] D. Purves, G. J. Augustine, and D. Fitzpatrick, Eds., *Neuroscience*, Sixth edition. New York Oxford: Oxford University Press, Sinauer Associates is an imprint of Oxford Universitiy Press, 2018.

[21] M. A. Hofman, “Evolution of the human brain: when bigger is better,” *Front. Neuroanat.*, vol. 8, Mar. 2014, doi: 10.3389/fnana.2014.00015.

[22] A. Sandberg, “Energetics of the brain and AI,” 2016, *arXiv*. doi: 10.48550/ARXIV.1602.04019.

[23] A. Sandberg and N. Bostrom, “Whole Brain Emulation: A Roadmap,” Future of Humanity Institute, Oxford University, Technical Report #2008-3, 2008. [Online]. Available:

www.fhi.ox.ac.uk/reports/2008-3.pdf

[24] https://www.ornl.gov/news/ornl-celebrates-launch-frontier-worlds-fastest-supercomputer

[25] D. Schneider, “The Exascale Era is Upon Us: The Frontier supercomputer may be the first to reach 1,000,000,000,000,000,000 operations per second,” *IEEE Spectr.*, vol. 59, no. 1, pp. 34–35, Jan. 2022, doi: 10.1109/MSPEC.2022.9676353.

[26] F. Akopyan *et al.*, “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[27] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker Project,” *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014, doi: 10.1109/JPROC.2014.2304638.

[28] M. Davies *et al.*, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018, doi: 10.1109/MM.2018.112130359.

[29] G. Orchard *et al.*, “Efficient Neuromorphic Signal Processing with Loihi 2,” in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, IEEE, Oct. 2021, pp. 254–259. doi: 10.1109/SiPS52927.2021.00053.

[30] H. Markram, “The Blue Brain Project,” *Nat Rev Neurosci*, vol. 7, no. 2, pp. 153–160, Feb. 2006, doi: 10.1038/nrn1848.

[31] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*, Expanded ed. Cambridge, Mass: MIT Press, 1988.

[32] A. Rodriguez-Vazquez *et al.*, “ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs,” *IEEE Trans. Circuits Syst. I*, vol. 51, no. 5, Art. no. 5, May 2004, doi: 10.1109/TCSI.2004.827621.

[33] G. L. Cembrano, A. Rodríguez-Vázquez, S. Espejo Meana, and R. Domínguez-Castro, “ACE16k: A 128×128 FOCAL PLANE ANALOG PROCESSOR WITH DIGITAL I/O,” *Int. J. Neur. Syst.*, vol. 13, no. 06, pp. 427–434, Dec. 2003, doi: 10.1142/S0129065703001765.

[34] S. J. Carey, A. Lopich, D. R. W. Barr, B. Wang, and P. Dudek, “A 100,000 fps vision sensor with embedded 535GOPS/W 256×256 SIMD processor array,” in *2013 Symposium*

on VLSI Circuits, Jun. 2013.

[35] S. J. Carey, D. R. W. Barr, B. Wang, A. Lopich, and P. Dudek, “Mixed signal SIMD processor array vision chip for real-time image processing,” *Analog Integr Circ Sig Process*, vol. 77, no. 3, pp. 385–399, Dec. 2013, doi: 10.1007/s10470-013-0192-x.

[36] J. Chen, S. J. Carey, and P. Dudek, “Scamp5d Vision System and Development Framework,” in *Proceedings of the 12th International Conference on Distributed Smart Cameras*, Eindhoven Netherlands: ACM, Sep. 2018, pp. 1–2. doi: 10.1145/3243394.3243698.

[37] J. Poikonen, M. Laiho, and A. Paasio, “MIPA4k: A 64×64 cell mixed-mode image processor array,” in *2009 IEEE International Symposium on Circuits and Systems*, Taipei, Taiwan: IEEE, May 2009, pp. 1927–1930. doi: 10.1109/ISCAS.2009.5118161.

[38] M. Laiho, J. Poikonen, and A. Paasio, “MIPA4k: Mixed-Mode Cellular Processor Array,” in *Focal-Plane Sensor-Processor Chips*, Á. Zarándy, Ed., New York, NY: Springer New York, 2011, pp. 45–71. doi: 10.1007/978-1-4419-6475-5_3.

[39] TOSHIBA TELI CORPORATION, “Sps02 Toshiba Smart Photosensor.” [Online]. Available: https://www.toshiba-teli.co.jp/pdf/industrial-camera-en/op-sps02_e.pdf

[40] J. Muller, R. Becker, J. Muller, and R. Tetzlaff, “CESAR: Emulating Cellular Networks on FPGA,” in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, Turin, Italy: IEEE, Aug. 2012, pp. 1–5. doi: 10.1109/CNNA.2012.6331409.

[41] R. Braunschweig, J. Muller, J. Muller, and R. Tetzlaff, “NERO mastering 300k CNN cells,” in *2013 European Conference on Circuit Theory and Design (ECCTD)*, Dresden, Germany: IEEE, Sep. 2013, pp. 1–4. doi: 10.1109/ECCTD.2013.6662202.

[42] A. Zarandy, A. Horvath and P. Szolgay, "CNN Technology-Tools and Applications", IEEE Circuits and Systems Magazine, vol. 18, no. 2, pp. 77-89, 2018. Available: 10.1109/mcas.2018.2821771

[43] L. Chua, “Memristor-The missing circuit element,” *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971, doi: 10.1109/TCT.1971.1083337.

[44] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008, doi: 10.1038/nature06932.

[45] N. K. Upadhyay, S. Joshi, and J. J. Yang, “Synaptic electronics and neuromorphic computing,” *Sci. China Inf. Sci.*, vol. 59, no. 6, p. 061404, Jun. 2016, doi: 10.1007/s11432-016-5565-1.

[46] Z. Wang *et al.*, “Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing,” *Nature Mater*, vol. 16, no. 1, pp. 101–108, Jan. 2017, doi: 10.1038/nmat4756.

[47] J. H. Yoon *et al.*, “An artificial nociceptor based on a diffusive memristor,” *Nat Commun*, vol. 9, no. 1, p. 417, Jan. 2018, doi: 10.1038/s41467-017-02572-3.

[48] R. Midya *et al.*, “Reservoir Computing Using Diffusive Memristors,” *Advanced Intelligent Systems*, vol. 1, no. 7, p. 1900084, Nov. 2019, doi: 10.1002/aisy.201900084.

[49] J. J. Yang, D. B. Strukov, and D. R. Stewart, “Memristive devices for computing.,” Nat. Nanotechnol., vol. 8, no. 1, pp. 13–24, 2013.

[50] B. Rajendran, Y. Liu, J. S. Seo, K. Gopalakrishnan, L. Chang, D. J. Friedman, and M. B. Ritter, “Specifications of nanoscale devices and circuits for neuromorphic computational systems,” IEEE Trans. Electron Devices, vol. 60, no. 1, pp. 246–253, 2013.

[51] L. Chua, “Resistance switching memories are memristors,” Appl. Phys. A Mater. Sci. Process., vol. 102, no. 4, pp. 765–783, 2011

[52] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chung, I.-K. Yoo, and K. Kim, “A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5−x/TaO2−x bilayer structures,” Nat. Mater., vol. 10, no. 8, pp. 625–630, 2011.

[53] A. Chanthbouala, V. Garcia, R. O. Cherifi, K. Bouzehouane, S. Fusil, X. Moya, S. Xavier, H. Yamada, C. Deranlot, N. D. Mathur, M. Bibes, A. Barthélémy, and J. Grollier, “A ferroelectric memristor,” Nat. Mater., vol. 11, no. 10, pp. 860–864, 2012.

[54] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, “Memristive switching mechanism for metal/oxide/metal nanodevices,” Nat. Nanotechnol., vol. 3, no. 7, pp. 429-433, 2008.

[55] M. Wuttig and N. Yamada, “Phase-change materials for rewriteable data storage.,” Nat. Mater., vol. 6, no. 11, pp. 824–832, 2007.

[56] D. Kuzum, R. G. D. Jeyasingh, B. Lee, and H. S. P. Wong, “Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing,” Nano Lett., vol. 12, no. 5, pp. 2179-2186, 2012.

[57] H. Jiang *et al.*, “Sub-10 nm Ta Channel Responsible for Superior Performance of a HfO2 Memristor,” *Sci Rep*, vol. 6, no. 1, p. 28525, Jun. 2016, doi: 10.1038/srep28525.

[58] D. R. Hughart *et al.*, “A Comparison of the Radiation Response of TaOx and TiO2 Memristors,” *IEEE Trans. Nucl. Sci.*, vol. 60, no. 6, pp. 4512–4519, Dec. 2013, doi: 10.1109/TNS.2013.2285516.

[59] W. M. Tong *et al.*, “Radiation Hardness of TiO2 Memristive Junctions,” *IEEE Trans. Nucl. Sci.*, vol. 57, no. 3, pp. 1640–1643, Jun. 2010, doi: 10.1109/TNS.2010.2045768.

[60] M. L. McLain and M. J. Marinella, “Overview of the radiation response of anion-based memristive devices,” in *2015 IEEE Aerospace Conference*, Big Sky, MT: IEEE, Mar. 2015, pp. 1–11. doi: 10.1109/AERO.2015.7119304.

[61] Q. Xia and J. J. Yang, “Memristive crossbar arrays for brain-inspired computing,” *Nat. Mater.*, vol. 18, no. 4, pp. 309–323, Apr. 2019, doi: 10.1038/s41563-019-0291-x.

[62] G. Pedretti *et al.*, “Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity,” *Sci Rep*, vol. 7, no. 1, p. 5288, Jul. 2017, doi: 10.1038/s41598-017-05480-0.

[63] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” *Nature*, vol. 521, no. 7550, pp. 61–64, May 2015, doi: 10.1038/nature14441.

[64] C. Li *et al.*, “In-Memory Computing with Memristor Arrays,” in *2018 IEEE International Memory Workshop (IMW)*, Kyoto: IEEE, May 2018, pp. 1–4. doi: 10.1109/IMW.2018.8388838.

[65] P. Yao *et al.*, “Fully hardware-implemented memristor convolutional neural network,” *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020, doi: 10.1038/s41586-020-1942-4.

[66] C. Li *et al.*, “Analogue signal and image processing with large memristor crossbars,” *Nat Electron*, vol. 1, no. 1, Art. no. 1, Dec. 2017, doi: 10.1038/s41928-017-0002-z.

[67] M. Rao *et al.*, “Thousands of conductance levels in memristors integrated on CMOS,” *Nature*, vol. 615, no. 7954, pp. 823–829, Mar. 2023, doi: 10.1038/s41586-023-05759-5.

[68] W. Song *et al.*, “Programming memristor arrays with arbitrarily high precision for analog computing,” *Science*, vol. 383, no. 6685, pp. 903–910, Feb. 2024, doi: 10.1126/science.adi9405.

[69] M. A. Zidan *et al.*, “A general memristor-based partial differential equation solver,” *Nat Electron*, vol. 1, no. 7, pp. 411–420, Jul. 2018, doi: 10.1038/s41928-018-0100-6.

[70] P. Lin *et al.*, “Three-dimensional memristor circuits as complex neural networks,” *Nat Electron*, vol. 3, no. 4, pp. 225–232, Apr. 2020, doi: 10.1038/s41928-020-0397-9.

[71] Z. Wang *et al.*, “Fully memristive neural networks for pattern classification with unsupervised learning,” *Nat Electron*, vol. 1, no. 2, pp. 137–145, Feb. 2018, doi: 10.1038/s41928-018-0023-2.

[72] T. Matsumoto, L. O. Chua, and H. Suzuki, “CNN cloning template: shadow detector,” *IEEE Trans. Circuits Syst.*, vol. 37, no. 8, Art. no. 8, Aug. 1990, doi: 10.1109/31.56083.

[73] T. Matsumoto, L. O. Chua, and R. Furukawa, “CNN cloning template: hole-filler,” *IEEE Trans. Circuits Syst.*, vol. 37, no. 5, Art. no. 5, May 1990, doi: 10.1109/31.55004.

[74] T. Matsumoto, L. O. Chua, and H. Suzuki, “CNN cloning template: connected component detector,” *IEEE Trans. Circuits Syst.*, vol. 37, no. 5, Art. no. 5, May 1990, doi: 10.1109/31.55003.

[75] M. Radvanyi, B. Varga, and K. Karacs, “Advanced crosswalk detection for the Bionic Eyeglass,” in *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, Berkeley, CA: IEEE, Feb. 2010, pp. 1–5. doi: 10.1109/CNNA.2010.5430281.

[76] X. Hu, G. Feng, S. Duan, and L. Liu, “A Memristive Multilayer Cellular Neural Network With Applications to Image Processing,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 28, no. 8, Art. no. 8, Aug. 2017, doi: 10.1109/TNNLS.2016.2552640.

[77] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, “Simulating nonlinear waves and partial differential equations via CNN. I. Basic techniques,” *IEEE Trans. Circuits Syst. I*, vol. 42, no. 10, pp. 807–815, Oct. 1995, doi: 10.1109/81.473590.

[78] T. Kozek *et al.*, “Simulating nonlinear waves and partial differential equations via CNN. II. Typical examples,” *IEEE Trans. Circuits Syst. I*, vol. 42, no. 10, pp. 816–820, Oct. 1995, doi: 10.1109/81.473591.

[79] P. K. Brown and G. Wald, “Visual Pigments in Single Rods and Cones of the Human Retina,” *Science*, vol. 144, no. 3614, Art. no. 3614, Apr. 1964, doi: 10.1126/science.144.3614.45.

[80] U.S Nuclear Regulatory Commission (NRC)., https://www.nrc.gov/docs/ML1209/ML120960701.pdf Feb. 2012

[81] A. Canziani et al. “An Analysis of Deep Neural Network Models for practical applications.” Arxiv 2017

[82] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[83] T. Sen and H. Shen, “Fault Tolerant Data and Model Parallel Deep Learning in Edge Computing Networks,” in *2024 IEEE 21st International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, Seoul, Korea, Republic of: IEEE, Sep. 2024, pp. 460–468. doi: 10.1109/MASS62177.2024.00067.

[84] T.-J. Su, J.-C. Cheng, M.-Y. Huang, T.-H. Lin, and C.-W. Chen, “Applications of Cellular Neural Networks to Noise Cancelation in Gray Images Based on Adaptive Particle-swarm Optimization,” *Circuits Syst Signal Process*, vol. 30, no. 6, pp. 1131–1148, Dec. 2011, doi: 10.1007/s00034-011-9269-x.

[85] L. O. Chua and T. Roska, “The CNN paradigm,” *IEEE Trans. Circuits Syst. I*, vol. 40, no. 3, pp. 147–156, Mar. 1993, doi: 10.1109/81.222795.

[86] M. Di Marco, M. Forti, and L. Pancioni, “Complete stability of feedback CNNs with dynamic memristors and second-order cells,” *Circuit Theory & Apps*, vol. 44, no. 11, pp. 1959–1981, Nov. 2016, doi: 10.1002/cta.2205.

[87] M. Di Marco, M. Forti, and L. Pancioni, “Convergence and Multistability of Nonsymmetric Cellular Neural Networks With Memristors,” *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 2970–2983, Oct. 2017, doi: 10.1109/TCYB.2016.2586115.

[88] P. Arena, S. Castorina, L. Fortuna, M. Frasca, and M. Ruta, “A CNN-based chip for robot locomotion control,” in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS ’03.*, Bangkok, Thailand: IEEE, 2003, p. III-510-III–513. doi: 10.1109/ISCAS.2003.1205068.

[89] R. Liu, H.-Y. Lee, and S. Yu, “Analyzing inference robustness of RRAM synaptic array in low-precision neural network,” in *2017 47th European Solid-State Device Research Conference (ESSDERC)*, Leuven, Belgium: IEEE, Sep. 2017, pp. 18–21. doi: 10.1109/ESSDERC.2017.8066581.

[90] H. Harrer and J. A. Nossek, “Discrete-time cellular neural networks,” *Circuit Theory & Apps*, vol. 20, no. 5, pp. 453–467, Sep. 1992, doi: 10.1002/cta.4490200503.

[91] H. Harrer, J. A. Nossek, and R. Stelzl, “An analog implementation of discrete-time cellular neural networks,” *IEEE Trans. Neural Netw.*, vol. 3, no. 3, pp. 466–476, May 1992, doi: 10.1109/72.129419.

[92] T. Roska and L. O. Chua, “The CNN universal machine: an analogic array computer,” *IEEE Trans. Circuits Syst. II*, vol. 40, no. 3, pp. 163–173, Mar. 1993, doi: 10.1109/82.222815.

[93] C. Rekeczky, I. Szatmari, P. Foldesy, and T. Roska, “Analogic cellular PDE machines,” in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No.02CH37290)*, Honolulu, HI, USA: IEEE, 2002, pp. 2033–2038. doi: 10.1109/IJCNN.2002.1007452.

[94] T. Roska and A. Rodriguez-Vazquez, “Review of CMOS implementations of the CNN universal machine-type visual microprocessors,” in *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, Geneva, Switzerland: Presses Polytech. Univ. Romandes, 2000, pp. 120–123. doi: 10.1109/ISCAS.2000.856273.

[95] A. Zarandy *et al.*, “CNN technology in action,” in *Proceedings of the 2000 6th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA 2000) (Cat. No.00TH8509)*, Catania, Italy: IEEE, 2000, pp. 79–81. doi: 10.1109/CNNA.2000.876824.

[96] K. R. Crounse and L. O. Chua, “The CNN Universal Machine is as universal as a Turing Machine,” *IEEE Trans. Circuits Syst. I*, vol. 43, no. 4, pp. 353–355, Apr. 1996, doi: 10.1109/81.488819.

[97] P. Arena, L. Fortuna, and L. Occhipinti, “A CNN algorithm for real time analysis of DNA microarrays,” *IEEE Trans. Circuits Syst. I*, vol. 49, no. 3, pp. 335–340, Mar. 2002, doi: 10.1109/81.989167.

[98] A. A. Abdullah, B. S. Chize, and Y. Nishio, “Implementation of an improved cellular neural network algorithm for brain tumor detection,” in *2012 International Conference on Biomedical Engineering (ICoBE)*, Penang, Malaysia: IEEE, Feb. 2012, pp. 611–615. doi: 10.1109/ICoBE.2012.6178990.

[99] F. Zou, S. Schwarz, and J. A. Nossek, “Cellular neural network design using a learning algorithm,” in *IEEE International Workshop on Cellular Neural Networks and their Applications*, Budapest, Hungary: IEEE, 1990, pp. 73–81. doi: 10.1109/CNNA.1990.207509.

[100] H. Harrer, J. A. Nossek, and F. Zou, “A learning algorithm for time-discrete cellular neural networks,” in *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, Singapore: IEEE, 1991, pp. 717–722 vol.1. doi: 10.1109/IJCNN.1991.170484.

[101] L. Ravezzi, G. -F. Dalla Betta, and G. Setti, “Compact CMOS implementation of a low-power, current-mode programmable cellular neural network,” *Circuit Theory & Apps*, vol. 29, no. 3, pp. 299–310, May 2001, doi: 10.1002/cta.147.

[102] R. Brette and W. Gerstner, “Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity,” *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, Nov. 2005, doi: 10.1152/jn.00686.2005.

[103] C. Teeter *et al.*, “Generalized leaky integrate-and-fire models classify multiple neuron types,” *Nat Commun*, vol. 9, no. 1, p. 709, Feb. 2018, doi: 10.1038/s41467-017-02717-4.

[104] J. Lin *et al.*, “A Memristor-Based Leaky Integrate-and-Fire Artificial Neuron With Tunable Performance,” *IEEE Electron Device Lett.*, vol. 43, no. 8, pp. 1231–1234, Aug. 2022, doi: 10.1109/LED.2022.3184671.

[105] K. M. Kim *et al.*, “Voltage divider effect for the improvement of variability and endurance ofTaOx memristor,” *Sci Rep*, vol. 6, no. 1, p. 20085, Feb. 2016, doi: 10.1038/srep20085.

[106] S.-G. Ren, Y.-B. Xue, Y. Zhang, W.-B. Zuo, Y. Li, and X.-S. Miao, “High-temperature tolerant TaO *X* /HfO2 self-rectifying memristor array with robust retention and ultra-low switching energy,” *Applied Physics Letters*, vol. 124, no. 3, p. 033501, Jan. 2024, doi: 10.1063/5.0190308.

[107] H. Kim, M. Pd. Sah, C. Yang, T. Roska, and L. O. Chua, “Memristor Bridge Synapses,” *Proc. IEEE*, vol. 100, no. 6, Art. no. 6, Jun. 2012, doi: 10.1109/JPROC.2011.2166749.

[108] S. P. Adhikari, Changju Yang, Hyongsuk Kim, and L. O. Chua, “Memristor Bridge Synapse-Based Neural Network and Its Learning,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 23, no. 9, pp. 1426–1435, Sep. 2012, doi: 10.1109/TNNLS.2012.2204770.

[109] S. Duan, X. Hu, Z. Dong, L. Wang, and P. Mazumder, “Memristor-Based Cellular Nonlinear/Neural Network: Design, Analysis, and Applications,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 26, no. 6, Art. no. 6, Jun. 2015, doi: 10.1109/TNNLS.2014.2334701.

[110] M. Anguita, F. J. Pelayo, E. Ros, D. Palomar, and A. Prieto, “VLSI implementations of CNNs for image processing and vision tasks: single and multiple chip approaches,” in *1996 Fourth IEEE International Workshop on Cellular Neural Networks and their Applications Proceedings (CNNA-96)*, Seville, Spain: IEEE, 1996, pp. 479–484. doi: 10.1109/CNNA.1996.566621.

[111] A. K. Sharma *et al.*, “Common-Centroid Layouts for Analog Circuits: Advantages and Limitations,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France: IEEE, Feb. 2021, pp. 1224–1229. doi: 10.23919/DATE51398.2021.9474244.

[112] C. Li *et al.*, “Efficient and self-adaptive in-situ learning in multilayer memristor neural networks,” *Nat Commun*, vol. 9, no. 1, p. 2385, Jun. 2018, doi: 10.1038/s41467-018-04484-2.

[113] L. O. Chua and Sung Mo Kang, “Memristive devices and systems,” Proc. IEEE, vol. 64, no. 2, pp. 209–223, 1976, doi: 10.1109/PROC.1976.10092.

[114] Q. Xia, V. Ravichandran, T. Maurer, “Memristive Cellular Neural Network,” United States Patent and Trademark Office #US 2024/0046083 A1, 2024.

[115] V. Ravichandran, Y. Huang, B. Flannery *et al*., “Memristive Cellular Neural Networks for Ultrafast in-Pixel Computing,” Nature Communications, In-preparation, 2024.