Design and Analysis of a Reconfigurable Hierarchical Temporal Memory Architecture

Abdullah M. Zyarah

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Design and Analysis of a Reconfigurable Hierarchical Temporal Memory Architecture

by

Abdullah M. Zyarah

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering

Supervised by

Dr. Dhireesha Kudithipudi
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
June 2015

Approved by:

Dr. Dhireesha Kudithipudi
Associate Professor - R.I.T. Dept. of Computer Engineering

Dr. Marcin Lukowiak
Associate Professor - R.I.T. Dept. of Computer Engineering

Dr. Mehran Mozaffari Kermani
Assistant Professor - R.I.T. Dept. of Electrical and Microelectronic Engineering

Dr. Sohail Dianat
Department Head - Professor - R.I.T. Dept. of Electrical and Microelectronic Engineering
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Design and Analysis of a Reconfigurable Hierarchical Temporal Memory Architecture

I, Abdullah M. Zyarah, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

__________________________________________________________________________

Abdullah M. Zyarah

__________________________________________________________________________

Date
Dedication

To my family...

May God protect you and keep you safe always...
Acknowledgments

Praise be to Allah, the Lord of Creations.

I would like to thank all people who helped me to get this work done. I am grateful to Dr. Dhireesha Kudithipudi for her unwavering guidance and for being patient with me during my work;

my committee members for reviewing this work and their valuable time;

Dr. Lukowiak, Dr. Patru, Mark Indovina, Prof. P.R. Mukund for their advices and extraordinary support.

I would also like to thank my professors and colleagues at the Department of Electrical Engineering, University of Baghdad, for believing in me and supporting me.

Finally, I would like to thank the Higher Committee for Education Development in Iraq for giving me the opportunity to pursue my master degree at RIT.
Abstract

Self-learning hardware systems, with high-degree of plasticity, are critical in performing spatio-temporal tasks in next-generation computing systems. To this end, hierarchical temporal memory (HTM) offers time-based online-learning algorithms that store and recall temporal and spatial patterns. In this work, a reconfigurable and scalable HTM architecture is designed with unique pooling realizations. Virtual synapse design is proposed to address the dynamic interconnections occurring in the learning process. The architecture is interwoven with parallel cells and columns that enable high processing speed for the cortical learning algorithm.

HTM has two core operations, spatial and temporal pooling. These operations are verified for two different datasets: MNIST and European number plate font. The spatial pooling operation is independently verified for classification with and without the presence of noise. The temporal pooling is verified for simple prediction. The spatial pooler architecture is ported onto an Altera cyclone II fabric and the entire architecture is synthesized for Xilinx Virtex IV. The results show that ≈ 91% classification accuracy is achieved with MNIST database and ≈ 90% accuracy for the European number plate font numbers with the presence of Gaussian and Salt & Pepper noise. For the prediction, first and second order predictions are observed for a 5-number long sequence generated from European number plate font and ≈ 95% accuracy is obtained. Moreover, the proposed hardware architecture offers 3902X speedup over the software realization. These results indicate that the proposed architecture can serve as a core to build the HTM in hardware and eventually as a standalone self-learning hardware system.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 List of Objectives</td>
<td>2</td>
</tr>
<tr>
<td><strong>2 Related Work</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Machine Learning</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Hierarchical Temporal Memory Overview</td>
<td>5</td>
</tr>
<tr>
<td>2.3 The First Generation of HTM (Zeta1)</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Hardware Realization of Zeta1</td>
<td>7</td>
</tr>
<tr>
<td>2.5 The Second Generation of HTM (CLA)</td>
<td>8</td>
</tr>
<tr>
<td>2.6 The Hardware Realization of CLA</td>
<td>8</td>
</tr>
<tr>
<td><strong>3 Hierarchical Temporal Memory</strong></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Hierarchical Temporal Memory</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Sparse Distributed Representation (SDR)</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Spatial Pooler</td>
<td>12</td>
</tr>
<tr>
<td>3.4 Spatial Pooler Operation</td>
<td>14</td>
</tr>
<tr>
<td>3.4.1 Phase 1: Initialization</td>
<td>14</td>
</tr>
<tr>
<td>3.4.2 Phase 2: Overlap and Inhibition</td>
<td>14</td>
</tr>
<tr>
<td>3.4.3 Phase 3: Learning</td>
<td>15</td>
</tr>
<tr>
<td>3.5 Temporal Pooler</td>
<td>15</td>
</tr>
<tr>
<td>3.6 Temporal Pooler Operation</td>
<td>16</td>
</tr>
<tr>
<td>3.6.1 Phase 1: Activate the cells of the columns</td>
<td>16</td>
</tr>
<tr>
<td>3.6.2 Phase 2: Set the cells of the columns in the predictive state</td>
<td>17</td>
</tr>
<tr>
<td>3.6.3 Phase 3: Synapses permanence update</td>
<td>17</td>
</tr>
</tbody>
</table>
4 Proposed Architecture Design .................................................. 18
  4.1 HTM Architecture ............................................................ 18
    4.1.1 Spatial Pooler Architecture .......................................... 19
      4.1.1.1 Competitive Learning Algorithm Based Architecture ... 19
      4.1.1.2 Multi-Core Architecture ....................................... 21
    4.1.2 Spatial and Temporal Poolers ...................................... 23
      4.1.2.1 Pipelined Multi-Core Architecture ......................... 23
  4.2 HTM Main Control Unit .................................................. 26
  4.3 HTM Column ............................................................... 28
  4.4 HTM Cell ................................................................. 32

5 HTM Verification .............................................................. 36
  5.1 Setup HTM parameters .................................................... 36
  5.2 The Verification Methodology ......................................... 38
    5.2.1 Training and Testing Sets ......................................... 38
    5.2.2 Training Process .................................................... 39
    5.2.3 Testing Process ..................................................... 39
    5.2.4 Spatial Pooler Verification ....................................... 40
    5.2.5 Temporal Pooler Verification .................................... 40
    5.2.6 Spatial Pooler Software Implementation ....................... 41

6 Results and Performance Analysis ........................................ 43
  6.1 Classification ............................................................ 43
    6.1.1 HTM Parameters Vs. Classification Accuracy .................... 44
    6.1.2 Classification Accuracy in the Presence of Noise ............ 45
  6.2 Prediction ............................................................... 46
    6.2.1 Prediction Accuracy ................................................. 47
    6.2.2 Prediction Accuracy in the Presence of Noise ............... 47
  6.3 Performance Analysis .................................................. 48
  6.4 Device Utilization Summary ............................................ 50

7 Conclusions and Future Work .............................................. 52
  7.1 Conclusions .............................................................. 52
  7.2 Future Work ............................................................. 53

A Run Time Equations of the Spatial Pooler ............................... 56
List of Tables

5.1 The spatial pooler parameters used for simulation and verification 36
5.2 The temporal pooler parameters used for simulation and verification 37
6.1 Impact of noise on the classification accuracy 46
6.2 FPGA resources utilization of the single HTM region of 100 columns (3 cells per column) 51
## List of Figures

3.1 A representation of three HTM regions (or levels when it refers to the role of the region in the hierarchy) arranged hierarchically. Each region is composed of columns of cells arranged vertically. .......................... 10

3.2 Sparse distributed representation of the spatial-temporal pooler. The blue cells represent the inactive cells while the yellow ones are the active cells that represent a particular input within a certain context ......................... 12

3.3 A column with four cells connected to the input space via the proximal segment synapses which are represented as small circles at the bottom. The filled circles represent the connected synapses while the unfilled ones are the inactive synapses. ......................................................... 13

3.4 HTM single cell connections. The cell receives input from its neighbor cells via the synapses of the distal segment and from the input space via the synapses of the proximal segment. ........................................... 16

4.1 The competitive learning algorithm based spatial pooler architecture. The architecture is structured from clusters (group of columns, shown as yellow cubes, and single LCU, shown as a blue cylinder) and UART interface block. 20

4.2 Parallel multi-core spatial pooler architecture. This architecture is structured from columns arranged in a 2-D array and a MCU bridged by a MUX. 22

4.3 The HTM network architecture. The architecture is structured from columns of cells (an abstract of the biological neuron) and MCU to move data between different HTM regions, receive the encoder data (preprocessed input), and generate the output of the HTM region. .................. 24

4.4 The pre-processing steps performed by encoder. It includes resizing the images and reshaping the final output to a 32x8 array. ..................... 26

4.5 The spatial pooler MCU RTL diagram. It is composed of memory blocks and a sorting unit. The memory blocks are used to store the columns’ overlaps and the winners, while the sorting unit is utilized to select the winning columns. ......................................................... 27

4.6 State machine of the MCU in HTM ............................................. 28

4.7 RTL representation of the spatial pooler column with partitioned memory. 29
4.8 The column receptive field: (a)PRNG receptive field (b)RRF receptive field.
4.9 State machine of a column in HTM.
4.10 The HTM cell RTL diagram.
4.11 State machine of HTM cell.

5.1 The spatial pooler verification process block diagram. It is composed of spatial pooler that generates the SDR representation of the encoder output (pre-processing block) during the learning and testing processes and relays it to the SVM classifier. The results of SVM are observed graphically.
5.2 The spatial-temporal verification process block diagram. It has an HTM region that performs spatial and temporal operations on the pre-processed input data (encoder output) and generates the CLA output. Also, it exports the cells’ memory content to be mapped to an HTM region 3-D plot.

6.1 The MNIST database images and their sparse representation as generated by the spatial pooler.
6.2 The effect of the training set size on the classification accuracy.
6.3 The effect of the number of winning columns in the spatial pooler output vector on the classification accuracy in two cases, PRNG and RRF. The green line is the classification of the pre-processed images (Not SDR).
6.4 Classification accuracy of the spatial pooler with varying overlap threshold.
6.5 The prediction of number 2 at different points in time. This number is presented to the network as a part of a sequence between 0-4.
6.6 The first and second order prediction accuracy of HTM.
6.7 The effect of the presence of noise on the HTM prediction accuracy.
Chapter 1

Introduction

1.1 Motivation

The human brain has an inspirational mechanism in processing the data that surrounds us in the world. The enormous computational power of data being relayed and processed by the brain can not be achieved even by today’s best available super computers [1]. There have always been fundamental questions about why computers can not perform tasks as efficiently as the brain, and why they consume more power in comparison to the biological brain. One answer to these questions may be the way that our brain processes information which is completely different from conventional computers [2]. An example of these differences, which explains the remarkable speed of the brain in processing data, is the massive parallelism of the brain structure relative to computers. To this end, two approaches are adopted in neuromorphic computing to provide efficient computational systems. The first approach abstracts the processes that occur in the brain, whereas the other approach focuses on mimicking the structure and the algorithmic properties of the human brain regions that give it these salient features [2].

The neocortex of the brain is the region responsible for implementing the high-level cognitive tasks such as vision, hearing, language and all other tasks that today’s best computers can takes hours to accomplish. In 2003, Jeff Hawkins proposed a theory called a memory-prediction theory, known today as Hierarchical Temporal Memory (HTM). This theory aims to mimic the structure and the neocortex principles of operation. The theory has shown promising results in various disciplines and domains especially in applications
such as image recognition [3, 4], detecting anomalies in servers and in the movement of people [5], object classification, and prediction [6].

Unlike other neural network algorithms, HTM is a relatively complex algorithm that can learn from the world by finding patterns that are grouped together as sequences of events. It has a hierarchical structure of memory based regions that can be used to store and learn information. These regions are structured from building blocks which are just vertically aligned artificial neurons. These neurons are processing data in parallel and are homogeneous. Exploiting this parallelism in hardware will offer an energy/area efficient system. One of the technologies that provides this sort of environment is the Field Programmable Gate Array (FPGA), which has thousands of configurable logic blocks (CLBs) [7]. The FPGA can be used to build a parallel processing network that benefits from the HTM parallel architecture, and makes the algorithm perform cognitive tasks comparable to the human brain.

This thesis aims to develop a scalable hardware and self-learning platform based on HTM theory that replicates the structure and the algorithmic characteristics of the human neocortex. Moreover, it aims to study the effect of various hardware architectures, and effect of varying parameters, on the performance and the function of the HTM. The developed module will be a step forward to building an efficient and intelligent computing platform. Building such an intelligent system will have an enormous impact in several applications including pattern recognition and anomaly detection.

1.2 List of Objectives

- Developed and verified a scalable hardware platform framework for Hierarchical Temporal Memory (HTM) that can be utilized in various machine learning applications and especially in classification and prediction.

- Investigated the possible changes to HTM that do not affect the performance of the algorithm and enable efficient hardware implementation of the algorithm.
• Performed analyses on how changing the various parameters in HTM as well as the architectures might affect the classification accuracy of the SDR representation of the input.

• Developed different verification templates for the HTM algorithm to:
  
  – Evaluate the performance of the spatial pooler in classification using SVM classifier and MNIST handwritten database.
  
  – Evaluate the performance of the temporal pooler in predictions using European numberplate font numbers.
  
  – Investigate the feasibility of implementing the entire HTM network on the FPGA.

• Developed a MATLAB based software environment that facilitates debugging the spatial and temporal pooling.

• Proposed three different possible architectures for the HTM network. Two architectures exploit multi-core parallelism and the other one is hybrid variations with parallelism and pipelining. Two architectures have been verified with the spatial pooler only and one architecture is validated with overall HTM.
Chapter 2

Related Work

Recent research and evidence point to the human neocortex as the main region responsible for high-level cognitive tasks. This encourages researchers to pay more attention to this vital region of the brain in their goal to build a biological plausible intelligent systems. Inspired by the sophisticated functionality of the brain and experimental evidence, the HTM attempts at modeling and using the neocortex for a higher form of machine intelligence. In this chapter, the evolution of the HTM theory will be discussed as well as the previous hardware realizations.

2.1 Machine Learning

Machine learning is a field of artificial intelligence that explores and studies computational algorithms that can learn from and make predictions on different types of data [8]. The evolutionary bases of these learning algorithms have been primarily used for pattern recognition techniques and prediction applications. Based on the input data and the history, the machine learning algorithm creates a hypothesis that facilitates the system’s task to take decisions and to make predictions. There are several types of machine learning algorithms developed over time, but they are not designed to build an intelligent system as efficient as the human brain. The machine learning algorithms are primarily focused on solving problems such as (i) regression (ii) object recognition and (iii) classification.

The class of algorithms that are inspired by the structure and the operation of the biological brain is the artificial neural networks (ANN) [9]. They are composed of a massive
number of highly interconnected computational units known as artificial neurons. These neurons are connected through synapses that are weighted with values which reflect the strength of their connections. The weights of the synapses change over time during the learning process in order to store the acquired knowledge. This gives the algorithm the generalization feature which makes the network more generic and able to produce reasonable output for even unseen testing data.

Recent years have seen a rapid evolution in the field of machine learning, and new theories have been presented such as the hierarchical temporal memory. The HTM is proposed as a theory that can be used to build a "real" intelligence (as claimed by the author [2]). HTM attempts at mimicking the structure and the function of the human neocortex. The neocortex is the outer layer of the brain and it is mainly involved in performing the cognitive tasks such as learning languages and precipitation. It is composed of different regions or areas that work in the same principles. HTM tries to emulate these common principles with the ultimate goal of developing a new generation of intelligent machines that can learn, predict, and adapt easily in the working environment.

2.2 Hierarchical Temporal Memory Overview

The concept of HTM is proposed by Jeff Hawkins in 2003. In 2005, Hawkins starts a company called Numenta which proposed two generations of HTM: HTM-Zeta1 and HTM-CLA. Both generations are incorporated in two separate versions of a software development platform known as Numenta platform for intelligent computing (NuPIC) [10]. The first version of the software only supports HTM-Zeta1 algorithm and allows the users to perform experiments based on their needs. Unfortunately, this version of NuPIC is not user-friendly for non-programmers as the second generation which provides a better user interface to create experiments in the neuroscience discipline [11].

The first generation of HTM was published in a white paper by Dileep George and Bobby of Numenta in 2007 [12, 11]. The proposed work is based on the outlines stated by
the Hawkins’s theory in his book *On Intelligence*. In 2010 and after a remarkable effort, Hawkins, Ahmed, and Dubinsky published the second version of HTM in a white paper by Numenta [13]. This white paper describes the new HTM with emphasis on the prediction process. Moreover, it provides a pseudo code that explains how the spatial and temporal pooling processes work.

The fundamental operation of the two HTM generations is substantially different. The first generation is based on the spatial-temporal algorithm and Bayesian belief propagation, whereas the second generation is based on the cortical learning algorithm (CLA).

2.3 The First Generation of HTM (Zeta1)

This version of HTM is published in 2007. It outlines the HTM theory as described by Jeff Hawkins in his book *On Intelligence* [2] and his paper Hierarchical Temporal Memory: Concepts, Theory and Terminology, 2006 [14]. In this version, the HTM architecture is defined as a tree-shaped of hierarchical nodes that employ the Bayesian belief propagation algorithm. Each node is connected to at least one child and a parent node except the one at the bottom and top. The node at the bottom receives data from the sensory input and passes it to the higher ones where it gets combined to form invariant representations of the objects.

The nodes of the HTM have basically two main functions: the spatial pooling which is responsible for storing the common input patterns and the temporal pooling that groups the sequences of patterns that occur frequently together. For example, in image recognition applications, the nodes at the lower levels try to learn and remember the pixels of the image during the training phase. During the inference phase, the spatial pooler tries to determine how closely the input image matches the stored patterns, while the temporal pooler tries to place the sequential patterns that most likely occur together in one group. However, this version of HTM is no longer supported by Numenta since they started working on the new version of HTM which is based on CLA algorithm.
2.4 Hardware Realization of Zeta1

Zeta1 version of HTM is realized in hardware by several research groups and the focus is on accelerating the computational cognitive tasks performed by a system. In 2007, Kenneth et al. were able to realize the first generation of HTM on FPGA [3]. The authors made use of the parallelism between the HTM nodes to develop a reconfigurable hardware platform for image content recognition applications. The developed model has 81 parallel computational nodes arranged hierarchically in 3 layers. This architecture is realized on Virtex II Pro FPGA housed in a Cray XD1 and runs at 120 MHz. The results show that the hardware implementation has 148 speed up over the software implementation run at 2 GHz AMD Opteron processor.

In 2009, Melis et al. [15, 16] implemented (software) a user support system on Bayesian neural network (BNN) and HTM. They have found that although BNN performance is better than HTM, HTM has advantages over the BNN. HTM can provide a complete solution for ‘user intention estimation’ in contrast to the BNN which requires additional hardware to carry out the feature extraction tasks. The authors also prove that the regular structure of the HTM does not always manifest in the optimal solution for each application. Therefore, developing an HTM hardware that can adapt even at the level of hierarchy could provide more optimal results. In this work the authors proposed a possible a very large scale integration (VLSI) architecture that can be used for the HTM and supports the aforementioned features.

A Verilog implementation for the HTM fundamental unit, known as a node, is proposed in 2013 [17]. This paper discusses a possible implementation for a single node that can be used as a building block to create the entire HTM network. The authors proposed different cell architectures that can be utilized for the spatial pooler during its different modes of operation in addition to the temporal pooler implementation. Unfortunately, this work does not demonstrate the accuracy of the proposed design while performing spatiotemporal tasks. It is also not feasible or practical to apply these architectures for the CLA HTM.
2.5 The Second Generation of HTM (CLA)

This version of HTM is a result of continuous research and improvement of the first generation of HTM, Zeta1. HTM-CLA was released in 2010 as a draft version, then as a white paper in 2011. The paper describes the algorithm’s main functions in adequate detail that can be easily understood even by non-programmers. Moreover, the author provided a pseudo code that describes the flow of the main two operations in the HTM which is the spatial and temporal pooling.

This generation of HTM tries to mimic the architecture of the brain’s neocortex from the biological anatomy perspective [11]. In this version of HTM, the nodes are replaced by columns that contain several cells. The cells are connected to the network sensory input and among themselves. The first connection can change the cell state to be in active mode which is equivalent to generating the action potential in biological neurons, whereas the second sets a cell in the predictive mode that is similar to the steady rate of the action potential [13]. The active mode of the cells allows the HTM to represent the same input in various contexts. In contrast, the predictive state helps HTM to predict what will happen next based on the previous and current input data.

2.6 The Hardware Realization of CLA

To the best of our knowledge, there is no large scale hardware architecture of the HTM-CLA model or its sub-units. This is most likely because of the lack of a mathematical model for the second generation of the HTM as well as the complexity of the algorithm in contrast to the first generation, HTM-Zeta1. In 2012, Zhou and Luo [18] attempt to speed up the algorithm by parallelizing the HTM C code and implement it on the Adapteva Epiphany many-core hardware, which has 16 coprocessors. With this implementation, the computation time of the algorithm speed up by 13X in contrast to the single core implementation.
Chapter 3

Hierarchical Temporal Memory

3.1 Hierarchical Temporal Memory

The Hierarchical Temporal Memory (HTM) is a biologically inspired online machine learning theory that mimics the structure and the behavioral characteristics of the neocortex. The HTM theory was first proposed by Jeff Hawkins in his book *On Intelligence* (2003) as a theory that resulted from repackaging and reinterpreting the earlier theories such as BNN and the spatial-temporal algorithm [11].

HTM is basically a memory based system that can be trained on a sequence of events that vary over time. It has levels or regions arranged hierarchically just as in the neocortex. These regions are responsible for storing information and making predictions. Each region in the hierarchy is comprised of artificial neurons known as cells which incorporate the most important capabilities of the biological neuron. Within the same region, the cells are aligned vertically to form a columnar organization to the regions such that the cells within each column respond only to one particular input at a time.

The cell represents a basic building block in the HTM and similar to the biological neuron each cell has one proximal dendrite segment and several distal dendrite segments. The proximal segments connect the cell with either the input space or the layer immediately lower in the hierarchy, while the distal segments connect a cell with other neighboring cells within the same region. The segments are basically a group of synapses that have a scalar value assigned to it called permanence. The permanence determines the strength of a synapse’s connection and provides the system with the ability to learn patterns and to make
predictions.

Figure 3.1: A representation of three HTM regions (or levels when it refers to the role of the region in the hierarchy) arranged hierarchically. Each region is composed of columns of cells arranged vertically.

HTM, as can be inferred from its name has a hierarchical structure, can be trained on temporal sequences of data, and is a memory based system. Figure 3.1 depicts the architecture of the HTM with three regions\(^1\) (one region per level). These regions decrease in size higher up in the hierarchy. The size of the region reflects the amount of memory allocated to that region. The more memory is allocated to a region, the more complex information can be acquired. This is important since it influences the components assigned to a region. Here a component refers to the fundamental building unit of the objects. For example, the edges and corners are the components of the objects in the visual field. Since the lower layer of the hierarchy processes the presented data to the HTM network, typically more memory is allocated to this level to form more components out of the input data. These components are relayed to the higher levels to be reused and recombined to generate

\(^1\)Level and region are used interchangeably. Usually, "region" refers to the internal function of a region while "level" describes the role of the region in the hierarchy.
more complex objects. This mechanism of operation speeds up the training process and reduces memory usage.

HTM unlike other machine learning algorithms is an online adaptive learning system, i.e. it is not designed to be trained once or to be restricted to one class of data. HTM can efficiently extract and store the useful information of various types of data regardless of the source. In summary, HTM can be used across various application domains where the cognitive tasks need to be performed efficiently, such as pattern recognition and natural language recognition.

3.2 Sparse Distributed Representation (SDR)

The components’ states of any object can be coded with a desirable number of bits as a dense or sparse representation. In the dense representation, small number of bits are used to encode the states of the components. This means the difference between any two consecutive states is typically a single bit. This kind of representation can not be used to determine the common features between object’s component states and is also sensitive to noise. An alternative approach for dense representation is the sparse representation in which each state is represented with a large number of highly distributed bits. The benefit of such model is to increase the representation capacity and the number of the discrimination states that can be formed. Increasing the number of bits obviously means more memory capacity is needed to store the data. This problem can be avoided by making the representation sparse which means every component’s state is represented as a small number of active bits. This will significantly reduce the memory space required to store data and makes the system more robust to noise and highly computationally efficient [19, 20].

The HTM network works based on the principles of SDR. It has hundreds of cells (neurons) arranged in columns, and in regions. Every time an input is provided to the network, a small number of these neurons become active to represent that particular input. Typically, the number of the activated neurons at each time is no more than 2% as shown in Figure
3.2 in which the highlighted neurons represent the active neurons. This process allows the HTM region to have about the same number of active columns in each iteration and also enables the HTM to process data with a small amount of power consumption. The mechanism that helps the HTM to activate small portions of neurons at each time is the inhibition. Depending on the required distribution and locality of the active columns, the inhibition can be local (occurs within a particular radius of the region and only the columns within that radius take part) or global (covers the entire region and all the columns participate). In the local inhibition, the active neurons of higher activation potential inhibit their neighbor’s cells. In case of the global inhibition, the top 2% active neurons are selected to represent the input.

3.3 Spatial Pooler

The spatial pooling is the first and most critical stage of the CLA and it is responsible for converting the binary input data into sparse distributed representation (SDR). The SDR is crucial to learning sequences and making predictions in the HTM. The SDR representation of the input is achieved by activating a small number of columns to represent the input.
patterns. Typically, every column in the HTM region is either connected directly to the input space or other regions in the hierarchy. In case of region 1 (the lowest region in the hierarchy), the columns are connected to the input space. The columns are connected to the lower regions using proximal segments that are composed of several synapses as shown in Figure 3.3 [21]. Usually, all the cells of a single column share the same proximal segment. When the synapses are active and connected to a reasonable number of active bits in the input space, the proximal dendrite segment will be active. The activation of the proximal dendrite segment will nominate that column to compete with its neighboring columns to represent the input of the region. The column with the most active synapses and active inputs inhibits its neighbors and becomes ON (winner). The output of the spatial pooler is generated as a vector. The length of the vector corresponds to the number of the columns and the ON bits of the vector represent the winning columns.

Figure 3.3: A column with four cells connected to the input space via the proximal segment synapses which are represented as small circles at the bottom. The filled circles represent the connected synapses while the unfilled ones are the inactive synapses.
3.4 Spatial Pooler Operation

The operation of the spatial pooler can be separated into three distinct phases: initialization, overlap and inhibition, and learning.

3.4.1 Phase 1: Initialization

In this phase, the columns’ receptive field, which is the natural center a column may have over the input space, is determined as well as the permanence of the synapses. The permanence of the synapses is initialized with random values that are close to the permanence threshold, which is the minimum value of permanence that a synapse needs to be considered as a connected synapse. This speeds up the learning process because it activates the synapses after small numbers of training iterations.

3.4.2 Phase 2: Overlap and Inhibition

The winning columns that represent the input pattern are selected in this phase. This happens after determining the overlap of each column with the input space active bits. The overlap is computed by counting the number of active synapses that associate with active bits in the input space, multiplied by its boost (scalar value) as shown in 3.1, where $U$ is a vector mapped from the input space to a column and $P$ is the permanence vector of that column. If the overlap value exceeds a certain threshold ($\text{minOverlap}$), the column is nominated to compete against its neighbors to represent the input pattern. Usually, the nominated columns within each local region (inhibition radius) compete against each other and the winner columns are selected based on their overlap values and on the desired level of sparsity.

\[
\text{Overlap} = \text{Boost} \times \sum_{i=0}^{n-1} U_i \times P_i \quad (3.1)
\]
3.4.3 Phase 3: Learning

After determining the winning columns, the learning phase starts to update the permanence of the columns’ synapses as necessary, i.e. only the synapses of the active columns are updated. The approach followed in updating the permanence of the synapses is based on the Hebbian rule [22]. The rule implies that the connection of synapses to active bits must be strengthened by increasing their permanence, while the connection of synapses to inactive bits will be weakened by decreasing their permanence.

3.5 Temporal Pooler

The second stage of the CLA is the temporal pooler. This stage is responsible for learning sequences and making predictions. Usually, the cells of the winning columns are involved in this process. The active cells of the winning columns form synaptic connections (distal segment) with the prior active cells, such that cells can predict its active state by just examining the distal segments. The number of the distal segments that a cell might have depends on the number of the distinct pattern that can be predicted. The more distal segments a cell might have, the more connections it has with other cells and more patterns can be predicted.

Figure 3.4 depicts how a cell may be connected to other cells within the same region using distal dendrite segments with a limited set of potential synapses. As can be noticed from the figure, the cell is connected to other cells through three distinct distal segments that are created while learning three different input patterns. The first segment is established while learning the letter ‘A’, whereas the other segments are created while learning letters ‘B’ and ‘C’ respectively. The blue cells of the segments represent the current active cells with connected synapses, while the clear cells represent the same pattern when the synapses are disconnected.
3.6 **Temporal Pooler Operation**

The operation of the temporal pooler can be separated into three distinct phases: (1) activate the cells of the columns, (2) set the cells of the columns in the predictive state, and (3) synapses permanence update.

3.6.1 **Phase 1: Activate the cells of the columns**

Each cell in the HTM is connected to other cells of the same region by distal segments that rises as the number of learned patterns increases. The purpose of these segments is to determine the state of each cell at each point in time within a single column. Usually, the cells can be either in an inactive, predictive, or active state. In this phase, the cells are set to be in the active state. One cell per active column is selected to be active and learn the input pattern, if one of the column’s cells was in the predictive states in the previous time step. If the input is not predicted, i.e. there are no cells in the predictive state, all the cells of the active column are *bursting* to be in the active state. Furthermore, the cell with the largest number of active synapses, which is known as the best matching cell, is selected to be a learning cell to learn the input pattern.
3.6.2 Phase 2: Set the cells of the columns in the predictive state

The status of the distal segments of each cell is examined in this phase. The distal segments that are in the active state can turn its cell in the predictive state unless the cell is already activated by the feed-forward input. The status of the distal segment can be determined after computing the number of active synapses connected to the active cell. Once that number exceeds a particular threshold, the segment will be in the active mode. When the distal segment turns into the active mode, the segment synapses’ permanence are queued up to be updated in the next phase. The update of synapses occurs based on their column’s status and whether they are connected to active or inactive cells in the same region. After updating the synapses permanence, a new segment is added to the cell to predict more steps ahead.

3.6.3 Phase 3: Synapses permanence update

The update of the segment that is queued up in the second phase of temporal pooling is carried out in this phase (learning phase). The cells that are in the learning state have segments that are positively reinforced, while the cells that stop predicting have segments that are negatively reinforced.
Chapter 4

Proposed Architecture Design

HTM core structure consists of columns of vertically aligned cells (neurons). Each column functions independently and separately from its neighbor. To exploit this innate structure, a pipelined multi-core system is developed to provide a reconfigurable and flexible HTM architecture that can be ported onto different FPGA fabrics. The FPGA has a reconfigurable structure and provides the necessary parallelism to host the proposed architecture. For this work, both VHDL and Verilog are used for behavioral description of HTM algorithm design and for mapping it to hardware. In this chapter, the proposed architectures to realize HTM algorithm on FPGA are discussed in addition to the Register-Transfer level (RTL) design of the HTM columns and cells.

4.1 HTM Architecture

A hierarchical approach can be followed to implement HTM network on FPGA. This approach is based on developing standalone columns and cells, the building blocks in the HTM, then instantiate them in a design hierarchy to construct the entire network. Three building blocks are developed to build various architectures of HTM hardware platform: (i) column (ii) cell and (iii) control unit. Detailed architecture of these units will be discussed in the next sections in addition to the high-level modular architectures of the HTM: the spatial pooler architecture and the overall architecture (spatial pooler and temporal pooler).
4.1.1 Spatial Pooler Architecture

Three separate architectures are developed to implement the spatial pooler on the FPGA. The structure of the first is inspired from the competitive learning algorithm, whereas the second and the third are based on a multi-core system with and without pipelining, respectively.

4.1.1.1 Competitive Learning Algorithm Based Architecture

The competitive learning algorithm implies that within a cluster of columns, only the column that fits the input data best, i.e. has the highest overlap, is selected to be active \[19\]. The other columns will be inhibited and do not take part in the SDR representation. Typically the HTM region is divided into disjoint group of columns (cluster) that are connected to the input space. All the columns in the region have identical structure, utilize similar computational algorithms, and they are subjected to the constraint that no more than one column is connected to the same receptive field.

\[
\text{OverlapPerColumn} = \sum_{i=0}^{n-1} U_i \times P_i
\]  

\[
P(i) := \begin{cases} 
1 & P(i) > \text{PermTh}, \ i = 0, 1, ..., n - 1 \\
0 & \text{Otherwise}
\end{cases}
\]

Figure 4.1 illustrates the architecture of the spatial pooler that captures the competitive learning algorithm characteristics. As can be noticed, there are two main parts in the architecture: the spatial pooler clusters and the Universal Asynchronous Receiver/Transmitter (UART) interface. The spatial pooler processes the received data and encodes it into a sparse form. It has a scalable number of clusters that arrange into a 2-D array. Each cluster is mainly structured from a scalable number of columns and a Local Control Unit (LCU). The UART interface is responsible for moving data back and forth between a PC running MATLAB and the spatial pooler.
Figure 4.1: The competitive learning algorithm based spatial pooler architecture. The architecture is structured from clusters (group of columns, shown as yellow cubes, and single LCU, shown as a blue cylinder) and UART interface block.

Each column in the cluster is responsible for computing its overlap (as shown in 4.1) with a subset of the inputs and then passing it to the LCU. Once the LCU receives the overlap of the columns, it determines on the basis of these values which columns become active to sparsely represent the input. Afterwards, each cluster generates a vector of length corresponding to the number of columns in the cluster. The active bits of this vector point to the active columns within the cluster. The vectors of all the clusters are concatenated together and sent to the UART interface block. At the same time, the winning columns
are informed from their LCU to update their permanences according to the aforementioned Hebbian rule (Chapter 3).

Even though this architecture gives the SDR representation of the input, the number of input discriminable states that can be encoded is limited because each cluster’s output is associated with a particular region in the input space. It gets active only when there are a significant number of active bits in its region. If it is assumed that there always are enough active bits in each column’s receptive field, the number of possible distinct states that can be generated using this architecture can be calculated using (4.3). In addition to this, the columns in this architecture are directly connected to the input space. Although this could speed up the computation of column overlaps, the interconnect area, the power dissipation and eventually the cost will be increased too. As a result, the number of the columns or clusters that can be scaled will be limited. For instance, for a 100-column, 16 synapses/column, the required interconnects will be 1,600.

\[
UniqueSDCoding/\text{Cluster} = \frac{m!}{w!(m-w)!} \tag{4.2}
\]

\[
UniqueSDCoding = (UniqueSDCoding/\text{Cluster})^{\text{Number of clusters}} \tag{4.3}
\]

where, \(m\) is the number of columns per cluster and \(w\) is the number of the active columns in \(m\)

### 4.1.1.2 Multi-Core Architecture

This model is inspired from a 2-D mesh network. Unlike the mesh network, in this architecture there is no need for the cores to communicate with each other since they are performing computations on unrelated set of data. In this model, the spatial pooler is composed of identical cores arranged in 2-D array and a main control unit (MCU) as shown in Figure 4.2. The core corresponds to a single column in an HTM region, while the MCU
is a bridge that assists columns receiving data from the encoder and it also determines the winning columns. Each column in this architecture receives the input data via the MCU after being processed by the encoder. In this work, the encoder is only responsible for resizing the input vectors and annotating the location of active bits of the input data. This optimizes the memory space required to store the input data. The input data is used to compute column overlaps. This is necessary to determine the activation level of columns and eventually the winning columns that represent the input data in a sparse form.

![Figure 4.2: Parallel multi-core spatial pooler architecture. This architecture is structured from columns arranged in a 2-D array and a MCU bridged by a MUX.](image)

In this model, there is no local control unit for each group of columns to decide the winners, but all the columns send their overlap values to the MCU which finalizes the selection step and determines the winners. The MCU receives the column overlaps through a multiplexer that randomly selects the inputs to be relayed to the MCU and then to a sorting unit within the MCU. The sorting unit is responsible for selecting the winning columns. The selection of the winning columns depends on the global inhibition. Every
column competes with all other columns in the region, and 10% of the columns are picked to form the SDR of the input. The selection is based on the level of activations, the more activations a column has, the more likely it is selected to be a winner.

When the columns have different overlap values, which is not always the case, the competition among columns is ruled by the level of activations. Whenever two or more nominated columns have similar overlap values, they will be randomly chosen to be winners. Once the winners are selected, the spatial pooler output will be produced from the MCU. Moreover, the list of the winning columns is sent to all columns through the H-Tree main bus. The columns that are in the list, their synapses’ permanence will be updated for learning, while the columns that are not in the list their synapses’ permanence will remain unchanged.

As discussed earlier, the columns relay their data to MCU via the MUX. The MUX is used to govern the data flow from columns to MCU and also to overcome the direct connection of columns to the MCU, which requires a vast number of interconnects and ports linearly proportional to the number of columns. Although the MUX could help in reducing the interconnects and the ports, it forms a bottle neck as the number of cores increases. Additionally, building such a multiplexer requires a vast amount of logical units to realize it in hardware, and it also adds a significant latency that slows down the communication between the columns and the MCU.

4.1.2 Spatial and Temporal Poolers

In this section, the overall HTM architecture will be introduced. This architecture merges both the spatial and temporal poolers in one single design that represents a standalone HTM region.

4.1.2.1 Pipelined Multi-Core Architecture

The high-level architectural framework of this model is a multi-core system which comprises of a MCU and columns arranged in a 2-D array with pipelined structure (shown in
Figure 4.3). The MCU works as a bridge connecting an HTM region with either the input (encoder) or the other regions within the hierarchy, whereas the columns represent computational units that emulate the HTM region columns as described in [13]. All the columns in the network have identical structure and utilize similar computational algorithms. The inheritance of the parallelism from the multi-core system makes the model computationally powerful, whereby the concept of pipelining speeds-up moving data from the columns to the MCU. These features increase the flexibility of the developed architecture and make it well suited to be considered as a scalable architecture.

Figure 4.3: The HTM network architecture. The architecture is structured from columns of cells (an abstract of the biological neuron) and MCU to move data between different HTM regions, receive the encoder data (preprocessed input), and generate the output of the HTM region.

In this design, the entire HTM region is modeled, and that includes both the spatial and temporal pooling processes. Spatial pooling is performed first, in which the data are represented in SDR form. Similar to the previous design, the columns of the spatial pooler are receiving data from the encoder via the MCU. Then, each column starts computing
its overlap value based on the number of active synapses connected to the active bits in its receptive field. This overlap information is sent to the MCU in a pipelined fashion. Usually, each column does not pass its overlap value to the next stage in the pipeline until it receives the overlap of the columns proceeding it in the same row. The MCU in this model is responsible for controlling the data movement from one stage to the next stage in the pipeline and thereby maintains sequence. This reduces data bus width to the MCU as there is only transfer of overlap value (without columns’ numbers). When the MCU gets the overlap of all columns of the region, the winners will be selected using the sorting unit that is used in the previous design. Once the winning columns are determined, the list will be sent to the columns to update their permanence, and also to start the next stage of the CLA algorithm which is the temporal pooling. In this phase, the cells within the winning columns are selected to represent the input within the context in a way that similar inputs will most likely get similar representation at columns’ level and different representations at the cells’ level. After selecting the cells, their positions within the spatial pooler array will be sent to the MCU in order to generate the final output of the CLA network and also to resend it to other cells in the region.

The pipelined multi-core architecture is developed to overcome the limitations of the designs in 4.1.1.1 and 4.1.1.2. This design is developed to have all salient features of the HTM such as the scalability, flexibility, and speed. However, similar to the multi-core architecture and unlike competitive learning algorithm based architecture the number of possible distinct representations that can be achieved from such architecture can be computed using (4.4) [19]

\[
UniqueSDRencoding = \frac{n!}{w!(n-w)!}
\]  

(4.4)

where, \(n\) is the length of the spatial pooler output vector and \(w\) is the number of the active bits in \(n\).
4.2 HTM Main Control Unit

The main control unit in the previous architecture works as a bridge for relaying data from the HTM encoder to the CLA network, as well as from the CLA network to the output. Moreover, it is responsible for the spatial and temporal pooler final computations such as determining the winning columns and generating the CLA output vector. Our focus in this section is the compatibility between the main control units and the other units in the HTM region of the pipelined-parallel architecture. The main control unit receives the data fetched to the network after being processed (in MATLAB) by the encoder (Figure 4.5). The encoder resizes the input data, like an image, to be 256 bits and reshapes it to a 32x8 array as shown in Figure (4.4). This will speed up storing the input data and reduce the computation time.

The MCU receives the data from the encoder sequentially one packet at a time, where the packet refers to a single byte in the input vector. Once a packet is received, it is stored in a register to be relayed to the MUX and then to the columns via the H-Tree bus (as shown in Figure 4.3). Once the columns receive the input data, their overlap values will be computed and sent back to the MCU. The MCU starts receiving the columns’ data through the pipelined buses and stores them in the Overlap Array as shown in Figure 4.5. Then, the overlap values are randomly extracted from the array to be fetched in the sorting
Figure 4.5: The spatial pooler MCU RTL diagram. It is composed of memory blocks and a sorting unit. The memory blocks are used to store the columns’ overlaps and the winners, while the sorting unit is utilized to select the winning columns. The sorting unit consists of a series of comparators and registers. The task of these comparators is to find the top 20% column overlaps and store them in the registers that in turn pass it to the Winning Col. RAM. Then, the list of the winners will be sent back to the columns to update their synapses’ permanences during the learning phase. After the learning phase of the spatial pooling, the temporal pooling will start, in which the MCU plays the role of information distributor. Typically, the temporal pooler starts with selecting the winning cells that represent the input within their context. Once the cells are selected, they will be sent to the MCU via the pipelined network formed by the columns. The MCU is responsible for receiving this data and dispersing it back to the network so that each cell knows the current active cells. This process helps the cells, which are in inactive mode and have active distal segments, making predictions. While the cells are busy, the MCU
makes use of this time and generates the final output of the HTM region and starts the next iteration.

Figure 4.6: State machine of the MCU in HTM

4.3 HTM Column

The cells in the column are activated due to feed-forward input or the lateral input of the neighboring columns via the synaptic connections during the training process. Designing such dynamical connections in hardware is challenging because of the rigid interconnects in hardware and thereby lacks flexibility. To overcome this limitation, a virtual synapse concept is developed in this work. The virtual synapses emulate the characteristics of the physical interconnects such as its address and permanence using a memory unit associated with each column. Although this realization increases the computation time, it significantly
reduces the interconnect size and offers dynamic connectivity to port onto the FPGA.

Figure 4.7: RTL representation of the spatial pooler column with partitioned memory.

The memory unit in the column can be divided into 3 partitions. The first partition (16 bytes) stores the permanence value of each synapse, which is loaded to the memory block via SynpPerm register, while the second partition (16 bytes) has the column synapses’ addresses in the input space. Typically, the synapses’ addresses are generated from the MCU either by using a pseudo random number generator (PRNG) or using ROM, which has previously generated receptive fields of all HTM columns. In the PRNG version, each column can establish synaptic connections anywhere in the input space without restrictions (Figure 4.8). This approach gives all the columns the same likelihood to be active. Moreover, the column activation over time will be more consequent so that all the columns take part in representing the input data and avoid any dead columns. The second approach, which is called ROM Receptive Field (RRF), is captured from [13] in which each column has
a receptive field around its center in the input space and the neighbor columns are sharing some locations in the input space (Figure 4.8). Although this method can give better encoding results in terms of giving similar input similar representations, the columns that are connected to the low active areas in the input space are less likely be active. For this reason, these column activations need to be boosted up often to participate in representing the input data. The aforementioned operations occur during a phase known as the initialization phase. Following this phase, the third partition (32 bytes) will be loaded with the data received via the Active Bits/Winners Addr bus.

![Figure 4.8: The column receptive field: (a)PRNG receptive field (b)RRF receptive field.](image)

At this point, each column has all the data needed to begin the next phase, overlap phase. In this phase, the column’s overlap is computed. This process starts loading a synapse permanence, and its address in the input space, into the registers, PermV, and SynpAddr respectively as shown in Figure 4.7. The 6-bit SynpAddr is always padded with a zero at the MSB. Afterwards, the permanence of the selected synapses will be observed. Whenever the synapse’s permanence is more than the threshold (PermTh), the input vector packet that has the synapse address will be loaded in ActiAddr register. If the synapse
address in the input vector is logic ‘1’, the overlap counter will be increased by one. Otherwise, a new synapse address will be read from the memory to be checked. This process continues until all the synapse addresses are examined with their corresponding input bit addresses. Once all the columns’ overlaps are computed, they will be transferred via the pipeline register to the MCU in order to select the winning columns.

Figure 4.9: State machine of a column in HTM

During this time all the columns will be in the steady state until the winning columns
are determined. The MCU sends the list of the winners to each column in the region. Each column then checks if it is a winner. If so, the column updates the permanence of its synapses based on the Hebbian rule [22]. The column starts reading the permanence of the individual synapses and passes it to an \((+1/−1)\) unit that determines whether the connection should be strengthened or weakened. If the column is not a winner, its synapses’ permanence will remain unchanged.

### 4.4 HTM Cell

The column’s cell is responsible for representing the input data within the context and making predictions. This occurs during the temporal pooler phase of the CLA algorithm. Typically, to develop an HTM network that can predict high order sequences multiple cells are required. At every point in time, one of these cells is selected to be active. This feature facilitates the hardware realization of a single cell unit. This unit consists of a single cell and a shared memory block with three partitions. Each partition is used to store the information of its corresponding cell in the column.

The operation of the cell unit begins during the temporal pooling process. It starts with the initialization process in which the synapses’ permanences are initialized around the permanence threshold. The permanence threshold is sequentially loaded from a register (Initial. Perm.) into the synapses’ permanence location in the memory via a 4-to-1 MUX (shown in Figure 4.10). After the initialization, the cells will be in the ideal state until the winning columns of a particular input are determined. Each column informs its cells about its status via the command line which is 3 bits wide. The cells decode the received command via a set of AND gates and perform the necessary tasks based on the received commands. If the column is a winner, it is expected to receive “111” through the command line, otherwise, “101” is received. When the cell unit receives the winning flag from its column, one of the cells will be selected to represent the input. This is if one of the winning column’s cells was in the predictive state. If none of the cells were in the predictive state, all
the cells are set to be active, i.e. *bursting* will take place. The process of selecting the active cells occurs in phase-1, while in phase-2 the predictive cells are determined considering that the column should be inactive in this case.

Phase-1 starts by examining the cells’ status in the previous time step, which is stored in $CellsTimeLine$ memory. If one of the cells was in the predictive state, it is selected to be active. If the cell is set to be active due to active cells that were in the learning state, this cell is selected to be a learning cell as well. When none of the column’s cells are in the predictive state, the bursting will happen and all the cells of the column will be ON, and then the best matching cells are selected as in $Burst\_Block$ to represent the current fetched input. After this, the cells’ location that are active due to the previous input, which are already stored in the $PriorMem$ partition, will be transferred to the $SegmentUpdate$ partition via the $M.~Buffer$ register. The active cells’ numbers then will be transferred to the column to be sent via the pipeline to the MCU (Not shown in the schematic). The MCU re-distributes the winning cell numbers to all the cells in the network. If the received cells are selected as a winner, it starts the phase-3 to establish a new segment or updates their synapse permanence for learning, otherwise phase-2 starts.

Phase-2 is responsible for setting the network cells in the predictive state and updating the $SegmentUpdate$ partition to add the current active cell locations and the ones prior. Whenever a cell in not in an active state and has one or more active segment, the prediction will be possible. This happens in $Prediction\_Block$ in Figure 4.10. This process starts by reading the stored segment addresses and the current active cell addresses and checks for matching percentage. Whenever there is a reasonable match, more than 50% in this work, the cell of the matched segment is set to be in the predictive state by setting the cell status to be ”11”. Otherwise, none of the cells will be in the predictive state and the current active segment will be adopted in the next time step in phase-3.

Once phase-2 or phase-1 is finished, phase-3 will start. This phase starts by checking
the cells’ status in the previous and current time step. If one of the cells is in the predictive state and currently active, this means the current input matched the predicted and the synapses permanence will be strengthened. Yet if there was a cell in the predictive state and currently inactive, its synapses permanence will be weakened. The strength of the synapses occur in the Learning Block while the weakening happens outside it. However, in the Learning Block, the Segment partition is compared with the SegmentUpdate. If there is a match the SynpMatchVec will be filled with the location of the matched addresses to strengthen their connection, otherwise, it will be transferred to the Segment partition.
Figure 4.11: State machine of HTM cell.
Chapter 5

HTM Verification

The spatial and temporal pooler of the HTM are verified separately and are tested for two different applications based on their nature of operation. Since the spatial pooler is utilized for encoding, it has been tested with classification. Whereas, the temporal pooler is tested with the prediction since its structure is more suited for temporal problems. This chapter investigates the verification strategy used and the analysis techniques adopted for setting the HTM parameters.

5.1 Setup HTM parameters

There are several parameters and variables that determine the structure of the HTM network, its performance, and the hardware resources. In this section, the strategy followed to set various HTM parameters is discussed:

Table 5.1: The spatial pooler parameters used for simulation and verification

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of columns</td>
<td>100</td>
</tr>
<tr>
<td>Number of winning columns</td>
<td>10-20%</td>
</tr>
<tr>
<td>Number of column’s synapses</td>
<td>16</td>
</tr>
<tr>
<td>Synapse permanence threshold</td>
<td>127</td>
</tr>
<tr>
<td>Permanence dec/inc factor</td>
<td>1</td>
</tr>
<tr>
<td>Inhibition type</td>
<td>Global</td>
</tr>
<tr>
<td>OverlapMin</td>
<td>2</td>
</tr>
</tbody>
</table>

- Synapses’ permanences and permanence threshold: According to [13], the synapses’
Table 5.2: The temporal pooler parameters used for simulation and verification

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cells per column</td>
<td>3</td>
</tr>
<tr>
<td>Number of segments per cell</td>
<td>3</td>
</tr>
<tr>
<td>Number of synapses per segment</td>
<td>10</td>
</tr>
<tr>
<td>Synapse permanence threshold</td>
<td>127</td>
</tr>
<tr>
<td>Permanence dec/inc factor</td>
<td>1</td>
</tr>
</tbody>
</table>

permanences are set to be random values close to the permanence threshold. The purpose is to form or un-form synaptic connections after a small number of training iterations. Typically, the permanence is a scalar value ranging between 0 – 1. Implementing this in hardware requires specialized floating point unit which is complex and resource intensive. The permanence value can be optimized for hardware by setting it between 0 – 255, with a midpoint threshold of 127. The synapses are considered active only if their permanence value is greater than the threshold.

- **MinOverlap**: This value determines whether a column is considered during the inhibition process or not based on its overlap value. The MinOverlap is opted to be a low value ranged between 2-7 so that only the columns that represent the input better will be chosen to participate in input SDR representation.

- **Number of columns**: The number of columns is selected to achieve at least the minimum requirements of HTM constraints such as the sparsity level and semantic meaning in representing the input, and the unique representation of different inputs. The test platform for this work consists of 10x10 columns with, 20% maximum active columns at any given point of time. This can give a sparsity level of 20% as shown by equation 5.1.

\[
\text{SparsityLevel} = \frac{w}{n}
\]  

(5.1)
where,

\( w \) is the total number of ON bits;

\( n \) is the input vector length.

For all of the parameters specified above, the number of the unique representations that can be obtained is computed using equation (5.2). For example, in case of 100 columns with 20 of them active, there are \( 5.36 \times 10^{20} \) possible encodings. Although it is obvious that the dense encoding capacity is more than that in sparse encoding, this difference tends to be meaningless as the number of columns grows up [23].

\[
UniqueSDRencoding = \frac{n!}{w!(n-w)!} 
\]  

(5.2)

- **Number of synapses**: There are 16 synapses per column. The number of synapses is related to the number of columns and the input vector length. Fewer synapses, which ensure a reasonable overlap, are required for small input vectors and columns.

- **Number of cells**: The number of cells is selected to be 3 per column. This enables the network to represent more sequences and make more predictions.

### 5.2 The Verification Methodology

#### 5.2.1 Training and Testing Sets

Two data sets are observed to verify the HTM spatial and temporal pooling operations separately and independently. The first set is the MNIST database [24]. MNIST is a standard database of hand written digits. It composes of 60,000 training and 10,000 testing examples. Each example in the training and testing sets has an image of a hand written single number of size 28x28 pixels. In order to reduce the amount of data being stored in the hardware and the training time, these images are pre-processed using an encoder.
The encoder resizes the images to 16x16 and sets a threshold at 80 to convert them to binary images. The preprocessing of images is performed outside the developed prototype in MATLAB. The generated binary vector of the image is sent to the HTM network via a testbench whenever the network is simulated in ModelSim, or via a UART port when the model is required to be tested on the FPGA chip. The only part of the encoder that is developed in this work is the Pre-SP. This model is responsible for marking the active bits indices of the input vector and sending it to the network when it is convenient to do so.

The second database, which is utilized to evaluate the performance of the spatial pooler and to debug the operation of the temporal pooler, is the numbers of the European numberplate font (EUNF) [6]. The numbers of the set, which range between 0-9, are resized to 19x14 pixels and threshold is set at 80 to convert them to binary images. The images are fetched to the network using the same techniques aforementioned.

5.2.2 Training Process

This process involves training the HTM network on particular sets of data, which are the MNIST and the EUNF databases. The process starts by setting the HTM parameters to match the desired application and then fetching the training examples to the network one vector at a time after being processed by the encoder. Once the data is received by the network region, it will be processed by two operations. The first operation is the spatial pooling that generates the SDR representation of the input vectors and updates the active columns synapses permanence. In the second operation, which is the temporal pooling, the SDR representations of the input are stored and the distal synaptic connections among cells are formed.

5.2.3 Testing Process

During the spatial pooling operation, the process involves testing how an HTM network may respond to unseen input patterns, i.e verifying the generalization feature of the HTM. This process is performed after training the network on a given set of training examples,
and it implies fetching unseen patterns to the network and observing the results. During the temporal pooling, a sequence of 5 numbers is presented consecutively to the network several times. In every iteration, the content of the cells’ memory is exported to MATLAB in a CSV file to keep tracking the variations that occur in the network and how it responds to previously seen input.

5.2.4 Spatial Pooler Verification

The main fundamental task of the spatial pooler is to encode the input data into a binary SDR representation. The encoding process is subjected to constraints such that similar inputs may obtain similar or at least close representations. In order to verify whether the spatial pooler performs the intended functions aforementioned or not, two verification steps are observed. The first step checks if the spatial pooler generates sparse representations. This test is easy and can be applied by utilizing any traditional debugging method. The other set focuses on observing the semantic meaning within the generated SDR representations. This part is performed using SVM classifier and MNIST database as shown in Figure 5.1. The testing starts by setting the spatial pooler parameters and pre-processing the MNIST images by the encoder. Then, the output of the encoder is presented to the network for training, testing, and observing the classification accuracy (SVM classifier output). Different sets of HTM parameters are observed and its influence on the classification accuracy will be discussed in the results and analysis chapter.

5.2.5 Temporal Pooler Verification

The temporal is developed to learn sequences and to make predictions, consequently, it is verified with the prediction application. Unlike the spatial pooler, the temporal pooler is too difficult to debug traditionally. Thus, a software debugging environment using MATLAB is developed in addition to a specialized VHDL process. This is to read the cells’ memory, where all the significant information associated with columns’ cells are stored,
Figure 5.1: The spatial pooler verification process block diagram. It is composed of spatial pooler that generates the SDR representation of the encoder output (pre-processing block) during the learning and testing processes and relays it to the SVM classifier. The results of SVM are observed graphically.

and save them in a CSV file. The developed MATLAB code generates a 3-D plot to visualize an HTM region graphically and to facilitate the debugging process. The 3-D plot has 3 layers of cells aligned vertically and arranged in a 2-D array (as shown in Figure 5.2). The code reads the status of the cells at the current time step, and the two before and maps it to the 3-D plot so that the status of the cell in each time step can be visually seen and easily tracked.

5.2.6 Spatial Pooler Software Implementation

In order to facilitate the verification process and to check the FPGA results of the HTM spatial pooler, a replica of hardware implementation is created in MATLAB. The developed model works similarly to the hardware except that it works in sequential fashion. The MATLAB model is used along with the SVM classifier to observe the influence of tuning spatial pooler parameters on the sparsity level and the semantic meaning of the SDR
Figure 5.2: The spatial-temporal verification process block diagram. It has an HTM region that performs spatial and temporal operations on the pre-processed input data (encoder output) and generates the CLA output. Also, it exports the cells’ memory content to be mapped to an HTM region 3-D plot.

representations.
Chapter 6

Results and Performance Analysis

In this chapter the proposed architecture design is exhaustively analyzed for performance when different algorithmic tasks are emulated.

6.1 Classification

The spatial pooler fundamental tasks in the HTM are: (i) converting the input data into SDR (ii) giving similar inputs similar representations. The first feature is significant to distinguish the common features between inputs [19], whereas the second feature is necessary in the next phase of the CLA, temporal pooler, to perform successful predictions. One approach that can be followed to verify the functionality of the spatial pooler is classifying its output. An SVM classifier with a second-order polynomial kernel and MNIST database are used for verification. The resized MNIST numbers are asserted to the spatial pooler and encoded with 100 bits and 20% of them are active (Fig. 6.1). Then, the output of the spatial pooler is relayed to the SVM classifier. For this test, 90% of classification accuracy is achieved. Figure 6.2 shows the changes in the classification accuracy with the training set size. As can be noticed, incrementing the training set size reduces the generalization error of the model and eventually increases the classification accuracy. More detailed analysis of the classification accuracy with varying parameters is presented in the following subsections.
Figure 6.1: The MNIST database images and their sparse representation as generated by the spatial pooler.

Figure 6.2: The effect of the training set size on the classification accuracy.

6.1.1 HTM Parameters Vs. Classification Accuracy

Two parameters are studied in this work which influence how a region may encode the input data. These parameters are: the Overlap threshold (\textit{MinOverlap}) and the number of winning columns. Figure 6.3 illustrates the effect of the number of the winning columns on the classification accuracy. As can be noticed, the encoding capacity increases with the increase of the winning columns which leads to improved classification accuracy. This suggests that a larger number of columns do not necessarily achieve higher classification
accuracy.

Figure 6.3: The effect of the number of winning columns in the spatial pooler output vector on the classification accuracy in two cases, PRNG and RRF. The green line is the classification of the pre-processed images (Not SDR).

Figure 6.4 shows how the classification accuracy is inversely proportional to the overlap threshold. The overlap threshold has a significant influence on the number of active columns that participate in the input representation. With higher overlap threshold fewer columns are used for the representation. This eventually leads to decrease in classification accuracy. It is worth noticing from the previous figures that classifying the images without SDR gives better accuracy. This is because the images are compressed to less than half their original sizes.

6.1.2 Classification Accuracy in the Presence of Noise

In this test, numbers from the EU numberplate font are utilized to evaluate how spatial pooler classifies in the presence of noise. The numbers from the data set, which range from 0-9, are resized to 19x14 pixels after injecting a particular type of noise. Two types of noise with different density levels are added to distort the images. Table 6.1 shows
the noise type, its intensity, and its effectiveness on the classification accuracy. As can be seen, the increment in the noise level negatively affects the accuracy because of the false representation that may be generated due to the injected noise. However, the results show that the system can handle noise with a density of 20%.

Table 6.1: Impact of noise on the classification accuracy

<table>
<thead>
<tr>
<th>Noise Type</th>
<th>Noise Density or Variance</th>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salt&amp;Pepper</td>
<td>10% density</td>
<td>100</td>
</tr>
<tr>
<td>Salt&amp;Pepper</td>
<td>20% density</td>
<td>90</td>
</tr>
<tr>
<td>Gaussian</td>
<td>10% variance</td>
<td>100</td>
</tr>
<tr>
<td>Gaussian</td>
<td>20% variance</td>
<td>90</td>
</tr>
</tbody>
</table>

### 6.2 Prediction

The HTM region has 3 cells per column. This gives the network the capability of predicting first and higher order sequences. In this work, only the first order and second
order sequences are verified. Figure 6.5 (a, b, and c) shows the prediction steps of number 2 in a sequence between 0-4. Each subplot displays the number fetched to the network and a region of HTM that has 100 columns of 3 cells. In this context, subplots (a), (b), and (c) show the number presented to the network on the left and its representation at column and cell levels on the right. As can be noticed, only 10 columns are active in each time step to represent the input. It is also worth noting in Figure 6.5(a) that all the cells of the winning columns are active. This indicates that the network has never seen this particular input before and for that reason there is a bursting state. The cells that have black color will learn this particular input so that it can be predicted in the next time steps. After fetching the same sequence several times, the network can start predicting the next number. Figure 6.5(b) and (c) show the prediction of number 2 at time interval t+1 and t+2.

6.2.1 Prediction Accuracy

Figure 6.6 shows the changes in the prediction accuracy of the HTM network for a 5-number long sequence. Notice that the prediction starts after fetching the same sequence a couple of times since synapse permanences values at initial time step were 126 or 127. Once the segment has synapses with permanences over 128, it can be involved in the prediction step. This is to avoid having random predictions. Figure 6.6 also illustrates that the second order prediction starts after fetching the sequence four times because the cell should establish connections to predict the first order sequences then second order sequences.

6.2.2 Prediction Accuracy in the Presence of Noise

The presence of noise has a relatively small effect on the prediction accuracy in HTM due to the robust representation of input data as SDR. In this work, $\approx 25\%$ of salt & pepper noise is added to the same sequence used in the previous test. Figure 6.7 explains the variation in the prediction accuracy in the presence of noise. It is obvious that the noise causes a reduction in accuracy, but this is still relatively small since it is only $\approx 12\%$ and $\approx 20\%$ for the first and second order predictions, respectively.
Figure 6.5: The prediction of number 2 at different points in time. This number is presented to the network as a part of a sequence between 0-4.

### 6.3 Performance Analysis

The proposed architecture is synthesized for Virtex IV Xilinx FPGA, which runs at 80 MHZ. The MATLAB version runs on 2.9GHZ Intel i7, 8-core MACBook Pro. It was found that training spatial pooler on FPGA takes only 6.493 µs/sample, while the MATLAB model takes 25.34 ms/sample. In case of temporal pooler, it was found that the training requires only 5.052 µs/sample. Since the developed HTM hardware platform offers parallel
Figure 6.6: The first and second order prediction accuracy of HTM

Figure 6.7: The effect of the presence of noise on the HTM prediction accuracy

processing for the current data spatial pooling and the previous fetched data temporal pooling, the spatial pooler training time can be considered the dominant time required to train
the overall HTM network. However, it is important to note that the newer versions of Virtex architecture will yield significant improvements in speed.

The overall computation time of the spatial pooler is represented in the equations (6.1) and (6.2), which estimates the spatial pooler training time and the effect of increasing the number of columns on the training time. The equations consider the time needed to store the input data to the network \( T_{\text{StoreInputData}} \), the time that columns need to compute their overlap values and update the synapses’ permanences \( T_{\text{OverlapComp.}} \), and the time required to move the data back and forth between the columns and the MCU \( T_{\text{DataPropagation}} \).

\[
T_{\text{SPCompTime}} = T_{\text{StoreInputData}} + T_{\text{OverlapComp.}} + T_{\text{DataPropagation}} \tag{6.1}
\]

\[
T_{\text{Training}}(\mu s) = \frac{1}{30} \times \text{NumberOfColumns} + T_{\text{SPCompTime}} \tag{6.2}
\]

### 6.4 Device Utilization Summary

Table 6.2 shows the FPGA resource utilization of the a single HTM region. The FPGA, XC5VLX330T, has 207360 logic slices and 324 block RAM. The number of used slices by each column, including the cells, is 1183. This can be added to the resources used by the MCU to get 120260 slices as a total for the entire HTM network with 100 columns. Also, it can be noticed from the RTL diagram of each column that there is a single memory block of 64 bytes. Every two blocks can be merged together in a single dual port block RAM. This is added to the block RAMs used by column’s cells (3 cells per column). As a result, for 100 columns of 3 cells, only 150 blocks are needed. This means that the network can be scaled up to 175 columns for the given FPGA.
Table 6.2: FPGA resources utilization of the single HTM region of 100 columns (3 cells per column)

<table>
<thead>
<tr>
<th>Unit</th>
<th>Block RAM Utilization</th>
<th>Slice Logic Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTM Region (SP+TP)</td>
<td>150 - 46%</td>
<td>120260 - 57%</td>
</tr>
<tr>
<td>Column (3 cells)</td>
<td>2 - 0%</td>
<td>1183 - 0%</td>
</tr>
<tr>
<td>Cell Unit</td>
<td>2 - 0%</td>
<td>949 - 1%</td>
</tr>
<tr>
<td>MCU</td>
<td>0</td>
<td>3089 - 1%</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusions and Future Work

7.1 Conclusions

Inspired by the organization and the behavioral characteristics of the brain, HTM attempts at implementing high-level cognitive tasks such as vision, predictions, etc. The main contributions of this work are the development of a scalable hardware and self-learning platform for the HTM algorithm. The scalability of the proposed design is only limited by the available resources since the training time is slightly affected by the expansion of the HTM network. Moreover, this work introduces the virtual synapses concept to overcome the hardware limitations such as the lack of dynamic interconnects and scalability.

Different experiments were explored to study the effects of changing different HTM parameters and the noise on the HTM spatial and temporal pooling functions and performance. The results show that the hardware realization provides a training speed up to $3902X$ faster than the software implementation. Furthermore, the experimental results demonstrate that spatial pooling with large number of winning columns could improve the classification accuracy, in contrast to the overlap threshold which should be small. The spatial pooler can handle 20% of noise without affecting its performance significantly. On the other hand, the temporal pooling is verified with the prediction application and the results show that the proposed HTM hardware can perform first and second order predictions efficiently even in the presence of noise.
7.2 Future Work

There are several aspects in HTM that need to be explored in the future. One of these is evaluating HTM algorithm performance in classification and prediction applications with larger number of columns, probably double or triple what is proposed in this work. Also, in this work only an HTM with a single level is utilized. The results are promising to explore multi-level HTM. This probably optimizes the invariant representation feature in HTM and allows us to exploit it in other applications.

In this work, three main units are developed to realize the overall HTM network in hardware. These units work in parallel with the presence of dependability at required points in time. In the future, the parallelism can be extended further within the columns and cells. Moreover, different possible hardware architectures, such as cellular automata, can be explored to remove the MCU from the proposed architecture and eventually reduce the utilized resources.

It is also worth developing specialized components that are more suitable to HTM architecture, which may significantly reduce the resources used and speed up the computations as well. For example, developing a multi-port RAM where more than 3 or 4 read/write operations can be performed at the same time.
Bibliography


[9] Carsten Peterson and Thorsteinn Rb’gnvaldssonzl. “An Introduction to Artificial Neural Networks.” In: ().


Appendix A

Run Time Equations of the Spatial Pooler

\[ T_{\text{Initialization}} = T_{\text{ClockCycle}} \times \text{NumSynapsesPerColm} \times \text{NumColm} \quad (A.1) \]

\[ T_{\text{StoreInputData}} = T_{\text{ClockCycle}} \times \frac{\text{InputVectorLength}}{8} \quad (A.2) \]

\[ T_{\text{OverlapComp}} = (\text{NumNonActiveSynapses} \times 2 \times T_{\text{ClockCycle}}) + \] \[ (\text{NumActiveSynapses} \times 5 \times T_{\text{ClockCycle}}) \quad (A.3) \]

\[ T_{\text{FlagPropagation}} = \text{NumPipeStages} \times T_{\text{ClockCycle}} \quad (A.4) \]

\[ T_{\text{DataPropagation}} = \text{NumPipeStages} \times 2 \times T_{\text{ClockCycle}} \quad (A.5) \]

\[ T_{\text{SelectWinners}} = 3 \times T_{\text{ClockCycle}} \times \text{NumColm} \quad (A.6) \]

\[ T_{\text{TransWinners}} = T_{\text{ClockCycle}} \times \text{NumWinners} \quad (A.7) \]

\[ T_{\text{Delay}} = \text{DelayCount} \times T_{\text{ClockCycle}} \quad (A.8) \]