Hardware Implementation of Hierarchical Temporal Memory Algorithm

Weifu Li
Department of Electrical Engineering
North Carolina State University
Raleigh, USA
wli25@ncsu.edu

Paul Franzon
Department of Electrical Engineering
North Carolina State University
Raleigh, USA
paulf@ncsu.edu

***Abstract*— In this paper, a hardware ASIC implementation of the Numenta Hierarchical Temporal Memory (HTM) algorithm is presented. Each column in the neural network is implemented as a processing element (PE). Neuron cells in columns are built as identical cell modules. Dedicated register files for each module cell are employed to replace the conventional centralized memory organization. A complete neural network is built as a matrix of PEs connected in the mesh network. Both first order and high order network are successfully performed on a 20×20 PE matrix using images from MNIST dataset as input patterns. The power and area of a single PE including 2 cell modules are 1.29 mW and 17511 μm^2 respectively. The average processing time in the proposed implementation is 4.52 μs in learning mode and 4.39 μs in inference mode. Compared to the performance of a software implementation on the 4 threads CPU, the ASIC version provides a 329.6x speedup in learning mode.**

Keywords—HTM Network; ASIC Design; Distributed Memory; MNIST Dataset;

I. INTRODUCTION

Hierarchical Temporal Memory (HTM) is an on-line machine learning algorithm inspired by the structural and algorithmic properties of the neocortex [1]. By modeling the process of human brain handling information such as vision, audio, and behavior, HTM network has the capacity to learn and recognize input patterns, and then can make a prediction based on the learned knowledge. As the name implies, HTM network is developed based on three key factors, hierarchy, temporal and memory. For hierarchy, a typical HTM network consists of several level organized as a hierarchy. The output data from lower levels are recombined at higher levels into more complex components. With regards to temporal, an HTM network is able to learn both spatial and temporal patterns from a continuous data stream, which makes the network output at a given time point not only depends on the current input but are also influenced by the previous one. Instead of storing all data in one large central memory, information in HTM network is distributed among columns within each level.

HTM algorithm was first proposed by Jeff Hawkins and his colleagues at Numenta in 2007 [2]. Two generations of HTM have been presented so far. In recently years, HTM has been applied across various areas, such as machine learning, pattern recognition, data mining and navigation [3]. In [4], the HTM network was designed to recognize 11 words in the TIDIGIS corpus. For pattern recognition, HTM has been implemented to deal with the traffic sign recognition [5]. Since each column in the network requires a large amount of computation, the total amount of computation is dramatically increased when dealing with large scale problem on the sequential processor. Though various hardware solutions, such as FPGAs in [6, 7], have been already proposed to improve the performance of HTM, the neural network size in [6] was relatively small and only the spatial pooling was implemented in [7]. So far, none full ASIC implementation of HTM as this work has been published.

In this paper, a processing element (PE) based design is proposed to meet the bandwidth requirement of memory and interconnection for parallel computing. In our proposed design, each column in the neural network is mapped on a single PE. Neuron cells in each column are built as multiple identical sub-modules packed in PEs. Dedicated storage devices, built as a series of register files, are assigned to each cell modules to replace the conventional central memory organization. The 2-dimensional (2-D) matrix of PEs connected through the mesh network is used to implement a complete neural network.

II. HIERARCHICAL TEMPORAL MEMORY ALGORITHM

A. HTM Operation Mode

In HTM, a neural network is modeled as a 2-D matrix of connected columns, each of which contains multiple neuron cells as shown in Fig.1. The HTM network can be operated in two modes: learning and inference.

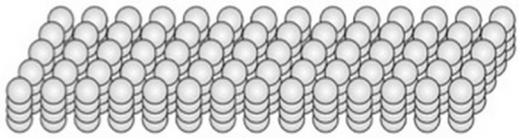

Fig. 1. HTM Neural Network

Generally, we gain the understanding of our practical world by learning both spatial and temporal patterns. The spatial pattern of a given input refers to the distribution of active bits and is represented by the combination of active columns. The temporal pattern can be considered as the time-based order of the input stream and is represented by the combination of cells in the active state. In learning mode, the spatial pattern and temporal patterns are stored in columns and cells respectively to generate a unique expression for each input. In addition, the

978-1-5090-1367-8/16/$31.00 ©2016 IEEE

connection strength between input bits and columns referred as proximal synapse permanence, and the boost value is updated. In temporal pooling, a series of connections to cells in learning state at previous time step (t-1) is generated for each cell in learning state at current time step (t) to memorize temporal pattern in learning mode. Meanwhile, the connection strength referred as distal synapse permanence is also modified.

Inference within HTM network is the process of matching current input patterns with ones stored previously. In inference mode, the most similar expression of the current input pattern is searched within the patterns stored in learning mode. Since the same input pattern can be presented to the network multiple times in learning mode, there could be various expressions of one input pattern stored in the HTM network. However, only the expression matching both spatial and temporal information can be considered as correct. In addition, a prediction for the next possible pattern is made using cells in the predictive state. Similar to the pattern matching process in inference mode, predictions in HTM are context sensitive. No prediction can be made while facing a completely new pattern.

B. Algorithm Description

In both learning and inference mode, operations in HTM can be divided into two phases: spatial and temporal pooling. The flow charts of spatial and temporal pooling are shown in Fig.2 (a) and (b) respectively. Operations in dotted blocks are only performed in learning mode.

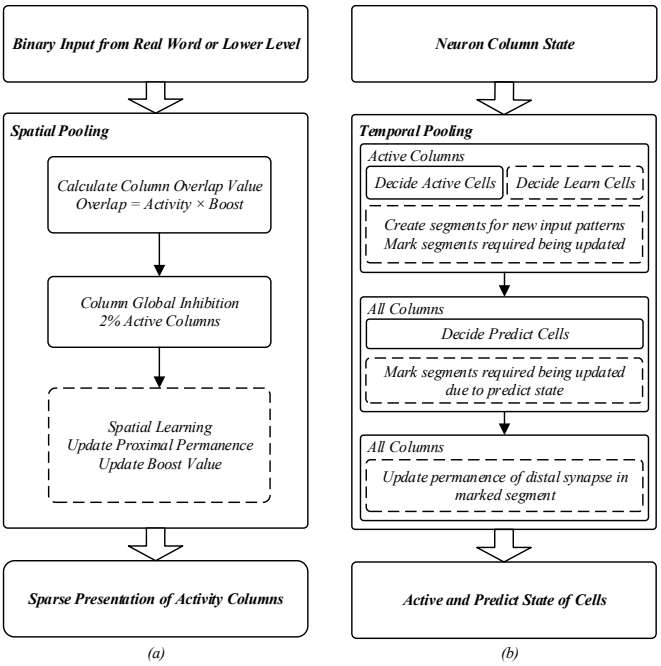

Fig. 2. Flow Charts of HTM Algorithm

1) Spatial Pooling

The main purpose of spatial pooling is to generate a sparse distributed representation (SDR) in columns. By employing the SDR, several desirable properties are integral to HTM.

- Though the number of active bits in the input patterns can be large, only a small percentage of the columns are required to be active, which can result in a great savings in both processing time and power consumption.
- In practice, input patterns in inference mode may not be exactly same as any previously stored pattern. Since the noise bits in input patterns have less influence on SDR, HTM network is more robust to the incomplete and noisy input.

In the learning mode, each proximal synapse of a column is mapped to a single bit of input data in current design. Only the input bit connected to a synapse with permanence above the threshold is considered as valid. The sum of valid input bits is multiplied by the boost value to calculate the overlap of each column. To generate the SDR, columns with a larger overlap inhibit its neighbor columns within a given radius, so that only a small percentage of columns can be active after inhibition. In the learning mode, the boost value of columns which are less active is increased. Proximal synapses connected to the active input bit have increases in their permanence while those connected to the inactive input bit get decreases in their permanence.

In the inference mode, all operations in the spatial pooling are same as that in learning mode. However, the boost value and proximal synapse permanence are fixed now.

2) Temporal Pooling

In temporal pooling, each neuron cell can have three states: learning, active and predictive. Neuron cells are only turned into learning state in learning mode. The distal segment used to present a series of connections between cells is introduced, each of which is referred as a distal synapse.

In temporal pooling, neuron cells in an active column are turned into the active and learning state if they were in the predictive state at t-1. Otherwise, all cells in the active column are turned into the active state to indicate an "unexpected" pattern. In that case, the cell having the best segment (segment has the largest number of active synapse) are turned into the learning state to create a new distal segment. The number of synapses connected to cells in the active state is considered as the distal segment activity. The distal segment with a segment activity above the threshold is active. To predict the next possible pattern, neuron cells in all columns with any active segment are turned into the predictive state. Meanwhile, the active segment is marked.

Finally, the distal segments with a mark are updated based on the cell state. For marked segments in the learning cell, the permanence of distal synapses connected to active cells is increased while that of the other synapses is decreased. For cells in predictive state at t- 1, the permanence of synapses in the marked segments is decreased if those cells aren't turned into the predictive state at t.

In the inference mode, all operations of the temporal pooling are same as that in the learning mode. However, the permanence update or segment creation is not allowed in the inference mode.

III. HARDWARE IMPLEMENTATION OF HTM

A. Design Element

The following design elements were implemented in this design in order to maximize performance and performance per unit of power.

1) Mapping Method

Numerous parallel operations take place in the various levels of HTM, such as the region level, the column level and the cell level [8]. In this work, each single column is implemented as a PE, and a connected 2-D matrix of such PEs is used to model a HTM network. Neuron cells in each column are implemented as identical sub-modules.The reasons why we use such a mapping method is as follow:

- In the HTM network, all columns, regardless the position, shares the same operations in both spatial and temporal pooling. In addition, since most of these operations in each column is independent, all columns in the network can be performed in parallel.
- For a given column, the operations of temporal pooling can be performed on all neuron cells in parallel. As a result, mapping each cell onto a single sub-module allows full exploration of parallelism in HTM.
- In the proposed design, a neural network is modeled as a matrix of PEs connected through a mesh network. Considering the independence of individual column, the proposed design is scalable to solve large scale problem.

2) Cell Number Per Column

As opposed to what is possible in the software implementation, the number of cells per column need to be decided before the manufacture of current design. In HTM, the network which only has one cell per column is named as first order network. In the first order network, the same spatial pattern can only result in the same combination of active cells in temporal pooling. In contrast, the network which has multiple cells per column is named as high order network. The high order network allows the various combination of active cells for the same spatial pattern. For instance, 100 active columns with 2 cells per column can be encoded in 2^{100} ways in temporal pooling.

Based on the analysis above, the number of cells per column should depend on the specific target application. Typically, the first order network is ideally suitable for the static spatial inference [1], such as the image recognition. The high order network can be applied to the time-based sequence recognition, such as music or language recognition [4].

3) Memory Usage

Since sufficient memory space is required in HTM to store the input patterns while in learning mode, the memory design is an important concern in the ASIC implementation. Different from the traditional computer system, storage devices in HTM are distributed within columns and cells. In this work, each distal synapse uses 3 bytes to store the location of a connected cell and permanence. New distal segments are only created when a completely new input pattern is presented to the HTM network. For a given column, the memory size for each cell module is calculated as (1)

$$Mem\ Size = \frac{N_{input} \times N_{syn} \times P_{active} \times 3}{N_{cell}}\ Bytes \quad (1)$$

The N_{input}, N_{syn}, N_{cell} and P_{active} is the number of new input patterns, the distal synapse count per cell, the number of cells per column, and the maximum column active frequency respectively. For the column active frequency, if a column has been turned into the active state for 30 times while learning a dataset with 100 input patterns, then the active frequency of this column is 30%. If the memory becomes full, the distal segments created for the oldest input pattern are overwritten to store the segments for the new incoming one.

B. Configuration of PE

The schematic of a PE is shown in Fig.3. In our proposed design, each PE consists of four parts: spatial pooler, temporal pooler, logic controller and the router.

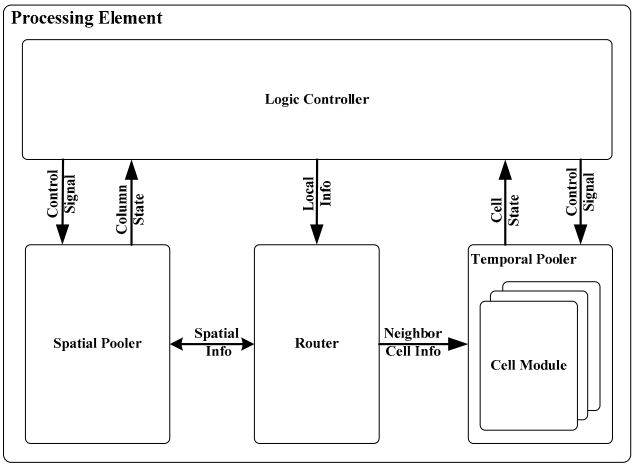

Fig. 3. Schematic of Processing Element

The spatial pooler is built to implement the operations in spatial pooling. To calculate the overlap value of each PE, the valid input bits mapped on each PE are summed up and then multiplied by the corresponding boost value. During inhibition, the local overlap value of each PE is broadcasted to all other PEs within the inhibition radius. Meanwhile, a counter in the spatial pooler is accumulated by one every time an overlap value larger than the local one is received. The column active flag can only be set if the number in this counter is smaller than the desire density. The spatial pooler in each PE is also used to count the number of times that the corresponding column is turned into active state to update the boost value and proximal synapse permanence in learning mode.

The temporal pooler is a combination of multiple identical sub-modules referred as cell module, one for each neuron cell. Each cell module can be divided into two parts: control logic and segment managers as shown in Fig.4. The control logic is used to collect and process the data received from all segment managers, such as segment activity and segment count, to find the best segment within each cell. The segment manager is designed to create, store and update the distal synapses stored

in register files. Each distal synapse including the location and states of connected cell occupies one register file index.

The basic function of the logic controller is to maintain correct operation order among various modules using the Finite State Machines (FSMs). In addition, the logic controller is also used to collect and process data received from all cell modules in temporal pooling. Due to the latency difference between active and inactive cells, the logic controller is responsible for the timing synchronization among cell modules.

The router in the proposed design is used to implement the optimized mesh network discussed in next section. During inter-PE communication, each PE is able to receive 4 packets simultaneously. Meanwhile, the 4 packets received previously are forwarded to the neighbor PE through corresponding output port. In order to provide enough processing time, the packets can stay in each PE for several clock cycles.

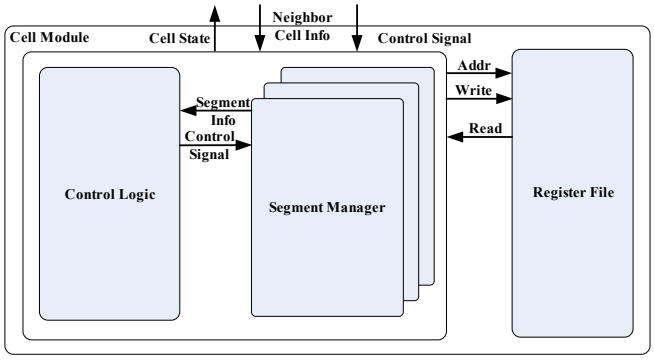

Fig. 4. Schematic of Cell Module

C. Inter-PE Connection

Besides sufficient memory space, the proposed design also requires an efficient interconnect topology to implement the cross-PE operations, such as column inhibition, distal segment creation, and boost value modification. In this paper, a mesh network is proposed to implement the inter-PE communication within the network. For a given packet, the communication process is divided into multiple routing cycles, each of which consists of multiple routing steps. The number of routing cycles for each PE is equal to the corresponding operation radius, such as the inhibition radius and learning radius. In the proposed mesh network, all PEs within the same level are allowed to send and receive multiple packets in parallel.

Each PE in the network has four input ports and four output ports connected to the four nearest PEs. During the first step of each routing cycle, local information, such as the overlap value and cell state, is sent to all neighboring PEs simultaneously. Then, the data received from neighbor PEs are forwarded. In addition to the required information, a 4-bit data used to record the travel distance is added to each packet. For a given packet, the travel distance is accumulated by one for each PE it passes through. As the travel distance of a packet is increased to the current target distance, the forward direction of each packet is changed by 90 degrees as shown in Fig.5. Current routing cycle is considered as done once the travel distance is equal to the target distance again. For each packet, similar routing cycle is repeated multiple times with different target distance until the packet passes through all PEs within the operation radius.

A routing example with an operation radius of 2 is shown in Fig.5. The arrows here indicate the moving track of a packet from the PE in blue. The black arrow and blue arrows stand for the track of first and second routing cycle respectively. As the packets from the blue PE arrives at the yellow PE, the travel distance becomes equal to the target distance of first routing cycle. As a result, the forward direction is changed to cover all PEs within the operation radius. For instance, packets received from the right-side input port are forwarded through the top-side output port rather than the left-side one in the yellow PE now.

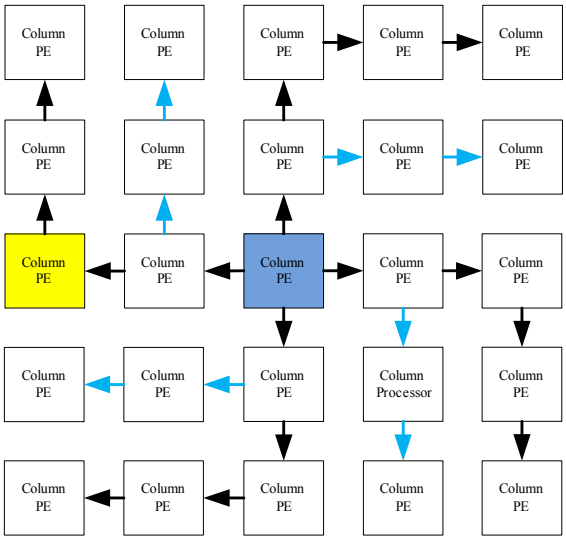

Fig. 5. Routing Example With a Radius of Two

IV. Functionality Test and Performance Evaluation

To test the functionality of this hardware implementation, a 20×20 column matrix was simulated in the ModelSim. The MNIST dataset from Yann LeCun's website [9] is chosen as our input pattern. Each binary image in MNIST dataset is normalized to a 20×20 grayscale pixel matrix. Each pixel is 8 bit. An example of the image in MNIST dataset is shown in Fig.6.

Fig. 6. Example of Image in MNIST Dataset

In our test, each PE is connected to one pixel. For a given input pattern, recognition is considered as successful if the active cells in inference mode are same as the learning cells in learning mode. Prediction is considered as successful if the

cells in the active state at t are the same as cells in predictive state at t-1 in inference mode.

A. Fisrt Order Network

In the first order network test, the same input pattern should always result in the same combination of cells in the active and predictive state.

In the first test, images for digits from 0 to 9 are presented to the HTM network one by one in learning mode. Then, the same input pattern is presented again in inference mode. The simulation results in TABLE I show that the proposed design can successfully recognize each input pattern.

In the second test, image "4" is repeated twice at different positions of the input sequence as shown in TABLE II. In reference mode, every time image "4" is presented to the HTM, cells used to represent image "7" and "8" are all turned into the predictive state. It proves that the first order network wasn't able to distinguish between the two identical input patterns while making a prediction.

TABLE I. TEST ONE FOR FIRST ORDER NETWOK

Network Operation Mode	*Input Pattern Sequence*
Sequence in learning mode	0, 1, 2, 3, 4, 5, 6, 8, 9
Sequence in inference mode	0, 1, 2, 3, 4, 5, 6, 8, 9
Prediction Sequence	1, 2, 3, 4, 5, 6, 8, 9

TABLE II. TEST TWO FOR FIRST ORDER NETWOK

Network Operation Mode	*Input Pattern Sequence*
Sequence in learning mode	0, 1, 2, 3, 4, 7, 5, 4, 8, 9
Sequence in inference mode	0, 1, 2, 3, 4, 7, 5, 4, 8, 9
Prediction Sequence	1, 2, 3, 4, 7&8, 5, 4, 7&8, 9

B. High Order Network

In high order network test, the same input pattern can result in different combinations of cells in the active and predictive state. In the proposed design, there are two cells per column.

The same input sequence is presented to the HTM network in the first test of high order network. Since the two image "4" are represented by different combinations of active cells now, the high order network can make a correct prediction for each image "4" as shown in TABLE III. The test is successful.

TABLE III. TEST ONE FOR HIGH ORDER NETWOK

Network Operation Mode	*Input Pattern Sequence*
Sequence in learning mode	0, 1, 2, 3, 4, 7, 5, 4, 8, 9
Sequence in inference mode	0, 1, 2, 3, 4, 7, 5, 4, 8, 9
Prediction Sequence	1, 2, 3, 4, 7, 5, 4, 8, 9

C. Perfomance Evaluation

The synthesis results of a single PE for the 45nm Nangate opencell technology is summarized in TABLE IV. There are two cell modules in each PE in the current version. The power and area of the register files are not included. Since only a few extra logic units are needed for adding more module cells into a PE, the power and area should be increased almost linearly.

TABLE IV. SYNTHESIS RESULT OF PE WITH TWO CELL MODULES

Subject	*PE*	*Cell Module*
Clock Period(ns)	10	10
Power (mW)	1.29	0.52
Area (μm^2)	17511	7382

For a 10ns clock period, the proposed design takes 4.52μs to process one input pattern in learning mode and 4.39μs in inference mode. Regardless of input pattern size, the operation latency stays almost same with constant operation latency.

In [3], a 16×16 column matrix is implemented. The average processing time in learning mode versus the number of thread is summarized in TABLE V. Compared to the performance of a 4 threads CPU, the proposed hardware can offer a speedup of 329.6x.

TABLE V. PERFORMANCE COMPARISION

Software Implementation		*Time per Input Pattern on Hardware Implementation*	*Speedup*
Number of Thread	*Time Per Input Pattern*		
1	3.06 ms	4.52 μs	676.9X
2	2.20 ms		486.7X
3	1.57 ms		347.3X
4	1.49 ms		329.6X

V. CONCLUSION

The major contribution of this paper is to present the design of an ASIC implementation of HTM algorithm. The neural column is implemented as a PE including multiple identical cell modules. Multiple PEs are connected together through the mesh network to deal with the large-scale application in the proposed design. Algorithm-driven design features include the mapping method, memory usage and setting cell number per column to fully exploit the parallelism in the HTM algorithm. Both first order and high order network are successfully tested using the MNIST dataset. The design was synthesized with a 45 nm library. Compared to the software implementation on a 4 threads general CPU, the average processing time of the ASIC is improved by 329.6x in learning mode.

ACKNOWLEDGMENT

The author would thank Lee Baker and Joshua Schabel for their distribution on this work. This work was sponsored in part by DARPA and AFRL under the agreement FA8650-15-7518.

REFERENCES

[1] J. Hawkins, S. Ahmad and D. Dubinsky, "Hierarchical Temporal Memory including HTM Cortical Learning Algorithms," Numenta, December 2011.

[2] J. Hawkins and S. Blakeslee, "On Intelligence", Henry Holt, New York, 2004.

[3] Xi Zhou, Yaoyao Luo, “Implementation of Hierarchical Temporal Memory on a Many-Core Architecture”, School of Information Science, Computer and Electrical Engineering, Halmstad University, December, 2012.

[4] J. V. Doremalen and L. Boves, "Spoken Digit Recognition using a Hierarchical Temporal Memory," Brisbane, Australia, 2008, pp. 2566-2569.

[5] W.Melis and M.Kameyama, “A study of the different uses of colour channels for trafficsign recognition on hierarchical temporal memory,” *International Conferenceon Innovative Computing, Information and Control (ICICIC)*, pp. 111–114, 2009.

[6] P. Vyas, M. Zaveri, “Verilog implementation of a node of hierarchical temporal memory”, *Asian Journal of Computer Science & Information Technology*, vol.3, no.7, 2013.

[7] A. M. Zyarah, D. Kudithipudi, “Reconfigurable hardware architecture of the spatial pooler for hierarchical temporal memory," 2015 28th *IEEE International System-on-Chip Conference (SOCC), Beijing, 2015, pp. 143-153.009*

[8] L. Bengtsson, A.Linde, T.Nordstrom, B.Svensson, and M.Taveniku, “The REMAP Reconfigurable Architecture: A Retrospective”, *FPGA Implementations of Neural Networks, Springer Verlag*, pp. 325-360, 2006.

[9] http://yann.lecun.com/exdb/mnist/