A HARDWARE IMPLEMENTATION OF THE SOM FOR A NETWORK INTRUSION DETECTION SYSTEM

by

Brent W. Roeder

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

KGaj — Dr. Kris Gaj, Dissertation Director

Dr. Jens-Peter Kaps, Committee Member

Brian L. Mark — Dr. Brian Mark, Committee Member

Dr. Andre Manitius, Department Chair

Dr. Lloyd J. Griffiths, Dean, Volgenau School of Engineering

Date: July 28, 2011

Summer Semester 2011
George Mason University
Fairfax, VA

A Hardware Implementation of the SOM for a Network Intrusion Detection System

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Brent W. Roeder
Bachelor of Science
Virginia Tech 2005

Director: Dr. Kris Gaj
Department of Electrical and Computer Engineering

Summer Semester 2011
George Mason University
Fairfax, VA

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A HARDWARE IMPLEMENTATION OF THE SOM FOR A NETWORK INTRUSION DETECTION SYSTEM

Brent W. Roeder, B.S.

Virginia Tech, 2005

Thesis Director: Dr. Kris Gaj

This thesis describes the research and development of a hardware implementation of the Self Organizing Map (SOM) for a network intrusion detection system. As part of the thesis research, Kohonen's SOM algorithm was examined and different hardware implementations for the SOM were surveyed. This survey resulted in the design and implementation of a conventional SOM, which was then modified for use as a detector of anomalous network traffic as part of a network intrusion detection system. The resulting implementation known as the port agent SOM is both smaller in area and supports higher data throughput than the conventional SOM, as was quantified through post place and route analysis. This thesis can serve as a tool for developing hardware implementations of the SOM, especially if their intended application is anomaly detection.

1 INTRODUCTION

There is a critical need for the ability to detect malicious network intrusions. A novel system known as the port agent architecture has been proposed that utilizes the Kohonen Self Organizing Map (SOM) as a tool for detecting these intrusions [1]. However, in order for this system to be realized, an efficient hardware implementation of the SOM must be researched and developed. This thesis lays the groundwork for the design of a hardware implementation of the SOM for the port agent architecture network intrusion detection system.

The SOM is an unsupervised learning algorithm conceived by Teuvo Kohonen and others in the early 1980's. As depicted in Figure 1, the SOM takes as input higher dimensional unlabeled feature vectors and produces a matrix of reduced dimensionality (typically 2D) based on the similarities in the features of the input training vectors [2].

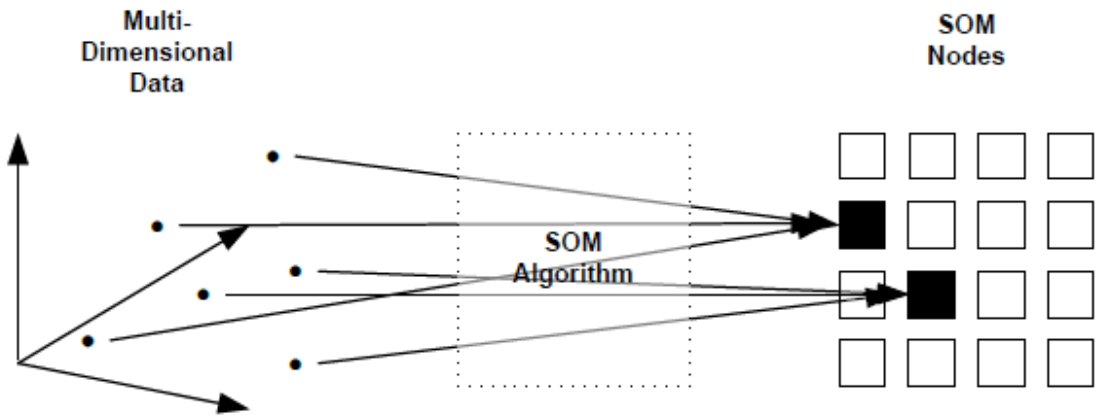


Figure 1 SOM Concept – Source Note: [1]

To accomplish this, the SOM must first be trained. Training a SOM is a relatively simple process. First, each element in the SOM matrix is assigned a random value or weight. Second, a feature vector from the training data set is compared to every element in the SOM matrix using a distance metric. Third, the element in the matrix with the smallest computed distance to the input feature vector is selected as the Best Matching Unit (BMU). Fourth, the BMU and elements in its neighborhood are updated to more closely match the input feature vector. The amount by which the elements are updated is dependent on the learning rate of the SOM and BMU's neighborhood size. In a conventional SOM, both of the learning rate and the neighborhood size decrease over time as measured by the number of training vectors processed. Finally, if a termination criterion is met (i.e. a sufficient number of vectors have been processed so that the

learning rate and neighborhood size are very small) the training is over, if not, steps two through four are repeated.

Training in the SOM is different than many artificial learning algorithms because it learns in an unsupervised fashion. That is, it does not require labeled data and requires no a priori information about the data it is trying to understand. The advantage of an unsupervised learning approach is that target outputs are not required. Because of this, the SOM can cluster data without user interaction directly from the input data. Simply put, the data to be clustered is the training data. Once trained, data can be presented to the SOM and the BMU calculated. The location of the BMU can be used to classify the type of data input and the distance between the input data and the BMU can be used to measure the degree of anomalousness. The SOM's ability to characterize data in this way has led to its use in speech recognition and translation, image recognition, speaker identification, radar target classification, and others [2].

Another application of the SOM's classification and anomaly detection capability is network intrusion detection. A system known as the port agent architecture is being researched and developed that will use a system of SOM based port agents to detect anomalous network traffic passing through the ports of an enterprise level network switch (see Figure 2).

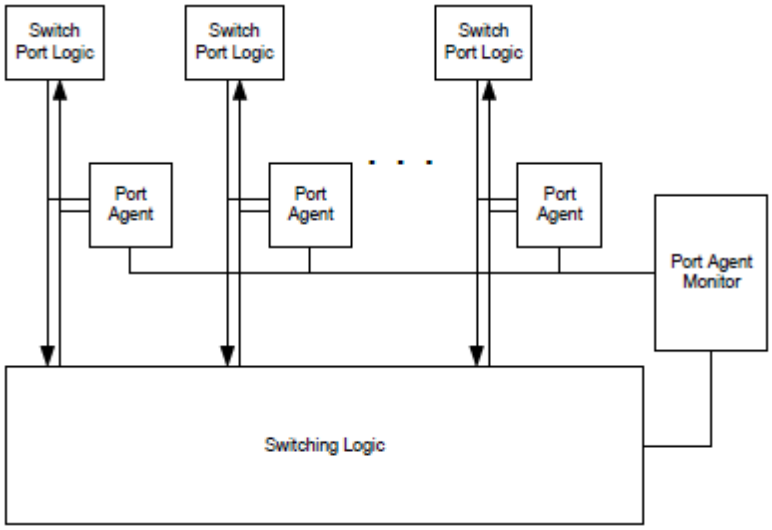


Figure 2 Port Agent Architecture – Source Note: [1]

The port agent will use two nearly identical SOMs. The first level SOM will classify network traffic and the second level SOM will detect anomalies in sequences of classifications (see Figure 3).

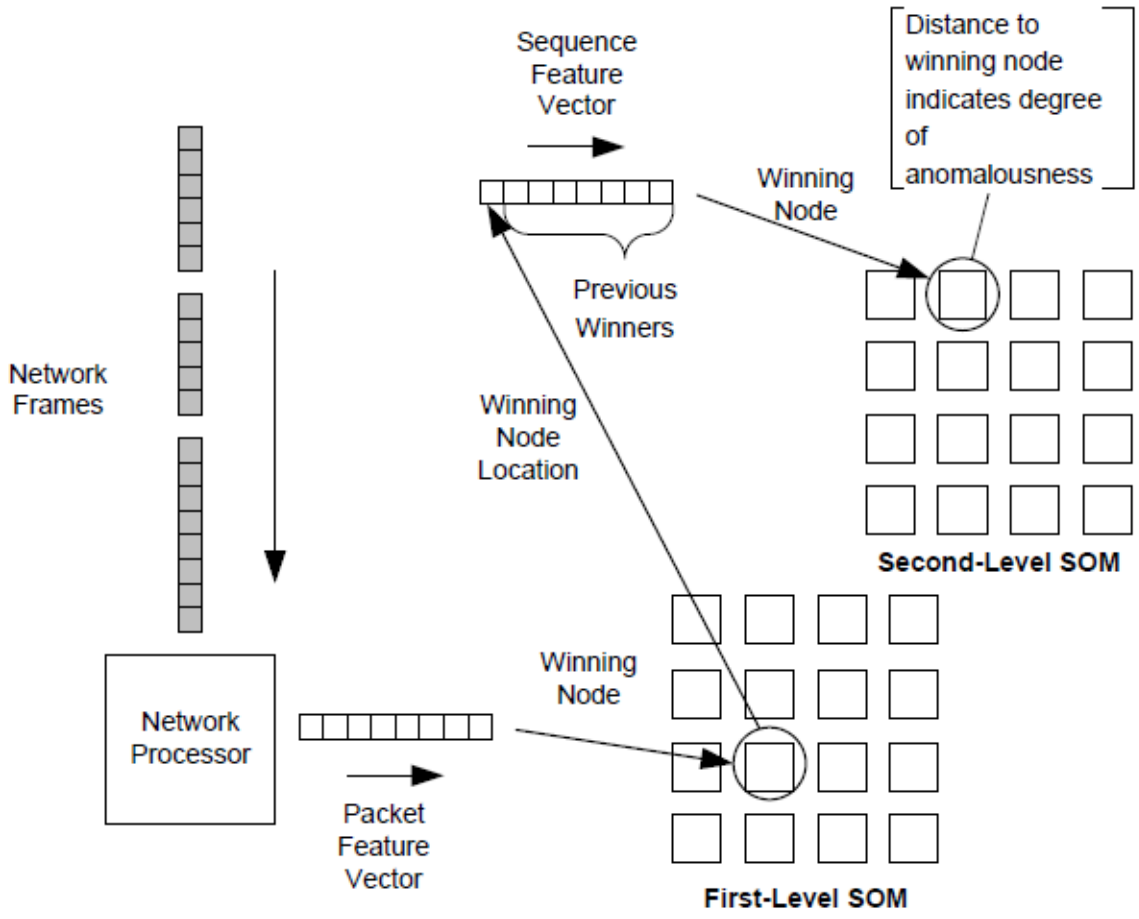


Figure 3 Port Agent Design – Source Note: [1]

The SOM design used for each of the levels in the port agent is based on a conventional SOM architecture, but optimized for classification and anomaly detection by being preloaded with an already trained matrix and using a constant learning rate and a constant neighborhood size. As mentioned, when training a conventional SOM, the learning rate and neighborhood size decease over time to zero until the SOM is no longer learning. Since the port agent SOM is preloaded with a trained map, it does not require this

capability. Rather than using a learning rate of zero in the port agent SOM, a small constant learning rate and neighborhood size is used so that small changes in the input data can be incorporated in the map while being processed for anomalies. This approach was validated in software in [3]. The modifications to the port agent SOM reduce its complexity in hardware and produce a design of reduced size and increased speed compared to a conventional SOM implementation. The objective of this thesis is to provide a design comparison between a conventional SOM and the port agent SOM. Both the conventional SOM and the port agent SOM designs will be fully implemented and the results will be used to show that the port agent SOM implementation is smaller than a conventional SOM implementation and can process data faster.

2 THE SOM ALGORITHM

The network intrusion detection system known as the port agent architecture will consist of anomaly detectors known as port agents that are based on an unsupervised learning algorithm called the SOM. Realizing the port agents in hardware requires an understanding of the computational requirements of the conventional SOM algorithm conceived by Kohonen.

As described in Chapter 1, in order for a SOM to be used it must first be trained. The first step required when training the SOM is matrix initialization. Data is processed and maintained by the SOM as n-dimensional weight vectors which have the form $W_j = (w_1, w_2 \ldots, w_n)$. Initializing the matrix requires assigning a starting value (usually random) to each of the weights in the matrix. Once initialized, training data can be presented to the SOM. The second step in the algorithm is the feature vector distance calculation. An input vector $V_i = (v_1, v_2 \ldots, v_n)$ is presented to the SOM and compared to each of the weight vectors W_j in the matrix using a distance metric. A common metric used in the SOM is the Euclidean norm [2].

$$\|V_i - W_j\| = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \cdots + (v_n - w_n)^2} \qquad (1)$$

The distance computed between the current weight of each element and the input vector is used to determine the best matching unit (BMU).

The third step of the SOM algorithm is BMU selection. BMU selection is accomplished by comparing the distance computed in the previous algorithm step for each element and selecting the index of the element whose weight is the closest to the input vector. This is formalized as

$$c = \text{argmin}_j\{\|V_i - W_j\|\} \tag{2}$$

by Kohonen in [2]. The BMU for each input feature vector V_i is calculated to determine the neighborhood in the matrix whose elements should be updated.

The fourth step of the SOM algorithm is the weight update calculation. Each element of the SOM matrix is updated as a function of the discrete time coordinate t as

$$W_j(t+1) = W_j(t) + \Delta W_j = W_j(t) + h_{cj}(t)[V_i(t) - W_j(t)] \tag{3}$$

where t is incremented for every input vector processed. The function $h_{cj}(t)$ is referred to as the neighborhood kernel and is also a function of the discrete time coordinate. The neighborhood kernel is typically computed as a Gaussian function [2] or a step function [4]. The Gaussian function is

$$h_{cj}(t) = \lambda(t) * \exp(-\frac{\|r_j - r_c\|^2}{2\sigma^2(t)}) \quad (4)$$

where $\lambda(t)$ is the learning rate and is a monotonically decreasing function of time, $\|r_j - r_c\|$ is the distance between the element to be updated and the best matching unit, and $\sigma(t)$ is the neighborhood width (see Figure 4). Similar to $\lambda(t)$, $\sigma(t)$ is a monotonically decreasing function of time.

An alternative to the Gaussian function is what is referred to as a step function [4]. The step function is

$$h_{cj}(t) = \begin{cases} \lambda(t) & if\ j \in N_c(t) \\ 0 & if\ j \notin N_c(t) \end{cases} \quad (5)$$

where $N_c(t)$ is a set of points within a boundary centered at the BMU c, and a radius $r = \sigma(t)$, calculated as a monotonically decreasing function of time (see Figure 4).

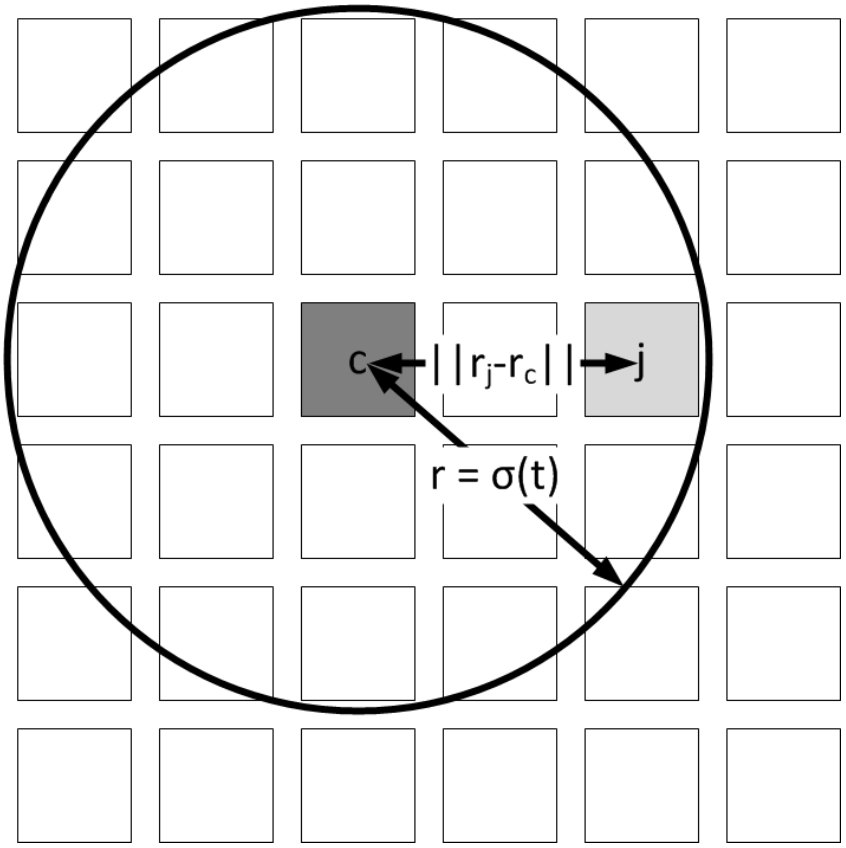


Figure 4 BMU Neighborhood

Steps two through four are repeated until a termination criterion is met. In many cases the criteria for termination is having the discrete time coordinate reach some threshold. Usually this threshold is chosen so that the learning rate and neighborhood are approaching zero and the SOM is no longer learning. As described in the previous chapter, once trained the SOM can be used to classify input vectors and measure their degree of anomalousness with the same computational steps as are used to determine the BMU.

3 SOM IMPLEMENTATION RESEARCH

The port agent architecture described in Chapter 1 requires a SOM implemented in hardware that is small enough to be integrated into a network switch yet fast enough to process 1 Gbps network traffic in real time. The design approach used to meet these requirements was to research and develop a fully featured conventional SOM in hardware and then optimize this design for minimal size and maximum throughput by simplifying or altogether removing elements of the conventional SOM not required by the port agent SOM. In order to understand the design tradeoffs for a conventional SOM hardware design, a survey of previous hardware implementations of the SOM was performed.

Hardware designs for the SOM can be coarsely divided into two categories namely analog and digital [5]. While examples of the former were examined [2][6][7], an overwhelming number of implementations found in the literature were for the latter. This is likely because it is more difficult to develop a SOM using analog circuitry that is sufficiently scalable for solving useful problems [8]. This, combined with the availability of high performance Field Programmable Gate Arrays [FPGAs] and their powerful development and simulation tools, eliminated analog implementations from consideration for the port agent SOM in favor of digital SOM implementations. Based on the survey performed, digital SOM implementations can be described and classified by their

topology, distance metric, BMU selection, and weight update function. The design tradeoffs for different SOM implementations are described in this context.

3.1 Topology

The majority of digital hardware implementations of the SOM in the literature are implemented as a matrix of processing elements or nodes with the distinguishing feature being whether the elements are interconnected with one another or not. The two generalized topologies are show in Figure 5.

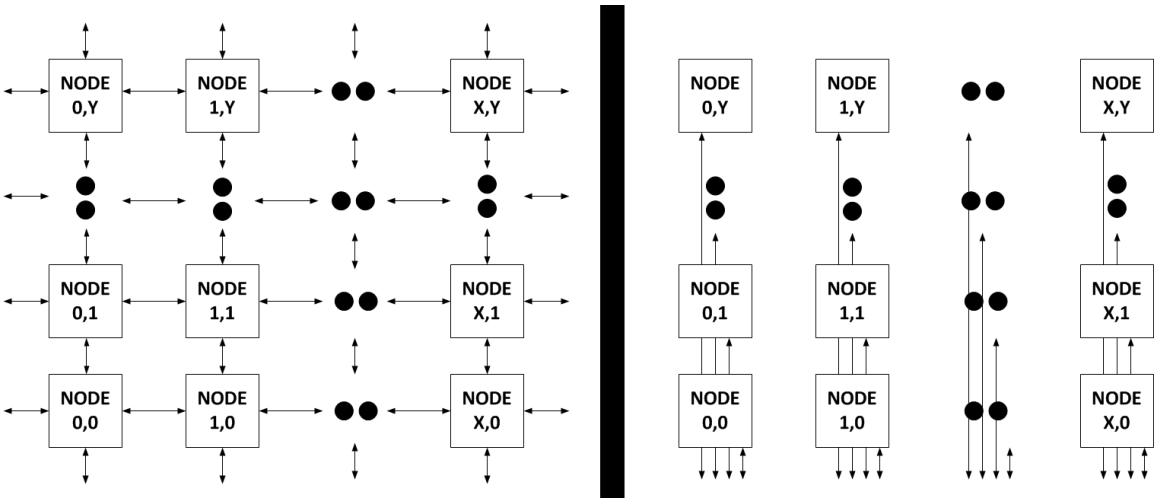


Figure 5 Interconnected Topology (left), Independent Topology (right)

Because SOMs are a form of competitive learning artificial neural network (ANN), some SOM implementations leverage an interconnected topology for their matrix. Two

examples of SOM implementations that use interconnected topologies are [9] and [10]. In [9], an existing hardware design used for ANNs known as MANTRA I is used for the SOM. In [10], a small SOM is developed using a traditional ANN interconnected architecture. Because interconnected topologies increase the hardware requirements of the SOM [8], most SOM implementations in the literature use an array of independent nodes where each node provides the distance metric and weight update functionality so that input vectors can be processed in parallel.

3.2 Distance Metric

While the Euclidean norm (see Equation 1) is often provided as the example distance metric in SOM algorithm descriptions, it is a hardware intensive function to implement because it involves the squaring of values and a square root. Therefore, most hardware implementations of the SOM use a computationally simpler function for calculating the distance between vectors. The most popular metric used based on the survey of available literature is the Manhattan distance. The Manhattan distance is calculated as

$$\|V_i - W_j\| = |(v_1 - w_1)| + |(v_2 - w_2)| + \cdots + |(v_n - w_n)| \quad (6)$$

and was used to replace the Euclidean norm in the majority of designs that were reviewed because it only requires the use of adders which can be implemented with a relatively simple digital circuit. While the Manhattan distance was used most often, other metrics were used as well. For instance, in [11], the Hamming distance was used. The Hamming

distance is calculated by counting the number of bit positions that are different between two values and like the Manhattan distance, requires fewer hardware resources than the Euclidean Norm.

3.3 BMU Selection

BMU selection requires the implementation of a search algorithm to select the SOM node whose weight vector is the smallest distance from the input vector (see Equation 2). Based on the literature surveyed, two popular ways of accomplishing this are the binary tree search and the bit serial search.

As the name suggests, the binary tree search finds the BMU by comparing the distance computed at every node using a binary tree [10]. An example implementation from [4] for a four element SOM is shown in Figure 6.

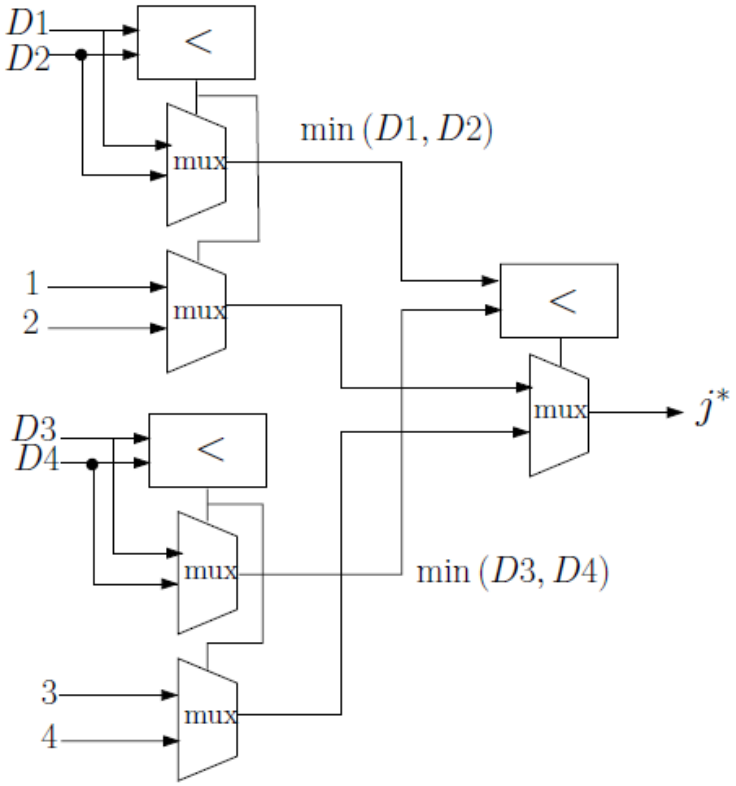


Figure 6 BMU Binary Tree Search – Source Note: [4]

The binary tree search is relatively efficient for small maps. However, the number of levels in the search hardware is the $\log_2$ of the number of elements in the map. Therefore, the path delay for large maps may be unacceptable. A more efficient method employed by many SOM implementations is the bit serial solution as shown in Figure 7. This is described in [2] as the optimal method for selecting a BMU.

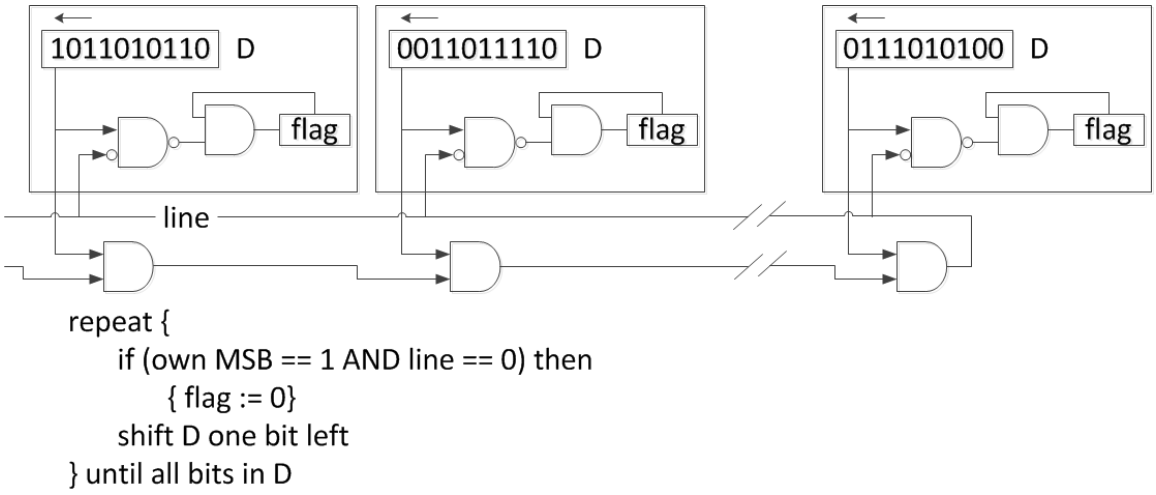


Figure 7 Kohonen's WTA Implementation – Source Note: [2]

In the implementation from [2] shown in Figure 7, each D is a representation of the difference between the input vector and the current weight of a node in the SOM. Initially, the flag for each difference, D, is set. The MSBs for all of the differences are compared. If there is at least one MSB equal to zero amongst all of the MSBs, any D with MSB of one has its flag reset. This process is iterated for all bits of D. Any D whose flag is set after the process is complete is a winning BMU. Unfortunately, the circuit as presented in [2] and shown in Figure 4 is flawed. Once a flag register is reset, it is not a possible BMU. Therefore, it should not continue to be compared to the other nodes' D. However, this was not accounted for by the BMU selection circuit in [2]. An additional OR gate is required to ignore all nodes that have been eliminated from the WTA competition for BMU (see Figure 8).

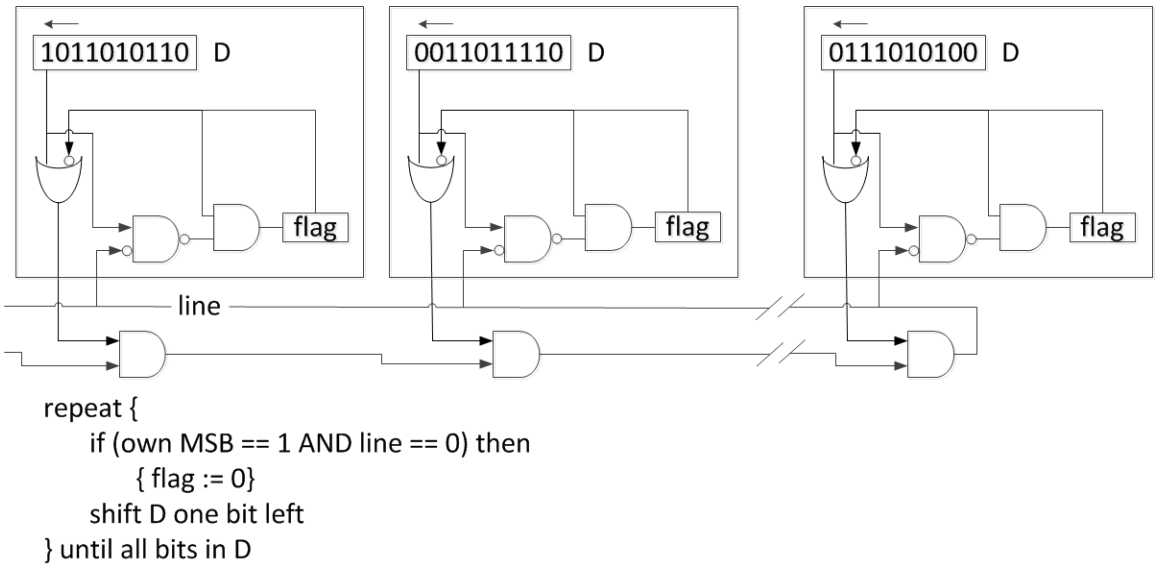


Figure 8 Improved BMU Selection Circuit

3.4 Weight Update Function

Similar to the distance metric, most hardware implementations of the SOM simplify the weight update function used in the literature (see Equation 3) in order to eliminate the hardware required for multiplication. Although alternative simplification methods exist such as the use of "Markov chains" [8][11], the solution encountered most often in the literature replaces the complex neighborhood kernel, $h_{cj}(t)$, with a negative power of two. Modifying the neighborhood kernel in this way replaces the multiplication required to implement the learning function with a divide by two implemented as a right bit shift [12] reducing the amount of hardware needed for the solution.

3.5 Summary of Hardware SOM Designs Surveyed

The results of the SOM digital hardware design survey are summarized in Table 1. Where possible, each implementation is qualitatively classified based on the criteria described in this chapter (i.e. topology, distance metric, BMU selection, and weight update function). In addition, the number of elements implemented for each design is listed. In many cases, projections were made for the number of elements possible in the design. However, only SOM element counts that appeared to be realized in hardware are reported. When a design could not be classified for a particular criterion, it is listed as indeterminate.

Table 1 SOM Digital Hardware Implementations Surveyed

Ref.	**Topology**	**Distance Metric**	**BMU Selection**	**Weight Update**	**Number of Elements**
[4]	Independent	Manhattan /Chessboard	Binary Tree	Negative Power of Two	25
[5]	Independent	Euclidean	Bit Serial	Indeterminate	25
[8]	Independent	Manhattan	Binary Tree	Markovian	Indeterminate
[9]	Interconnected	Euclidean	Bit Serial	Indeterminate	Indeterminate
[10]	Interconnected	Indeterminate	Binary Tree	Indeterminate	16
[11]	Independent	Hamming	Binary Tree	Markovian	60
[12]	Independent	Manhattan	Bit Serial	Negative Power of Two	16
[13]	Independent	Manhattan	Bit Serial	Negative Power of Two	Indeterminate
[14]	Independent	Manhattan	Indeterminate	Negative Power of Two	4
[15]	Independent	Euclidean	Linear Search	Negative Power of Two	25

4 PORT AGENT SOM IMPLEMENTATION

The approach used to design a hardware implementation of the port agent SOM was to develop a fully featured conventional SOM in hardware and then optimize this design for minimal size and maximum throughput. This was done by simplifying or altogether removing elements of the conventional SOM not required by the port agent SOM. The conventional SOM implementation that was developed and the modifications made for the port agent SOM implementation are described in the subsequent sections of this chapter.

As with many of the SOM implementations researched, the throughput of a SOM can be maximized in hardware by taking advantage of its topologically fixed architecture to process each input vector in parallel using a topology consisting of an array of independent processing elements or nodes (see Figure 9)

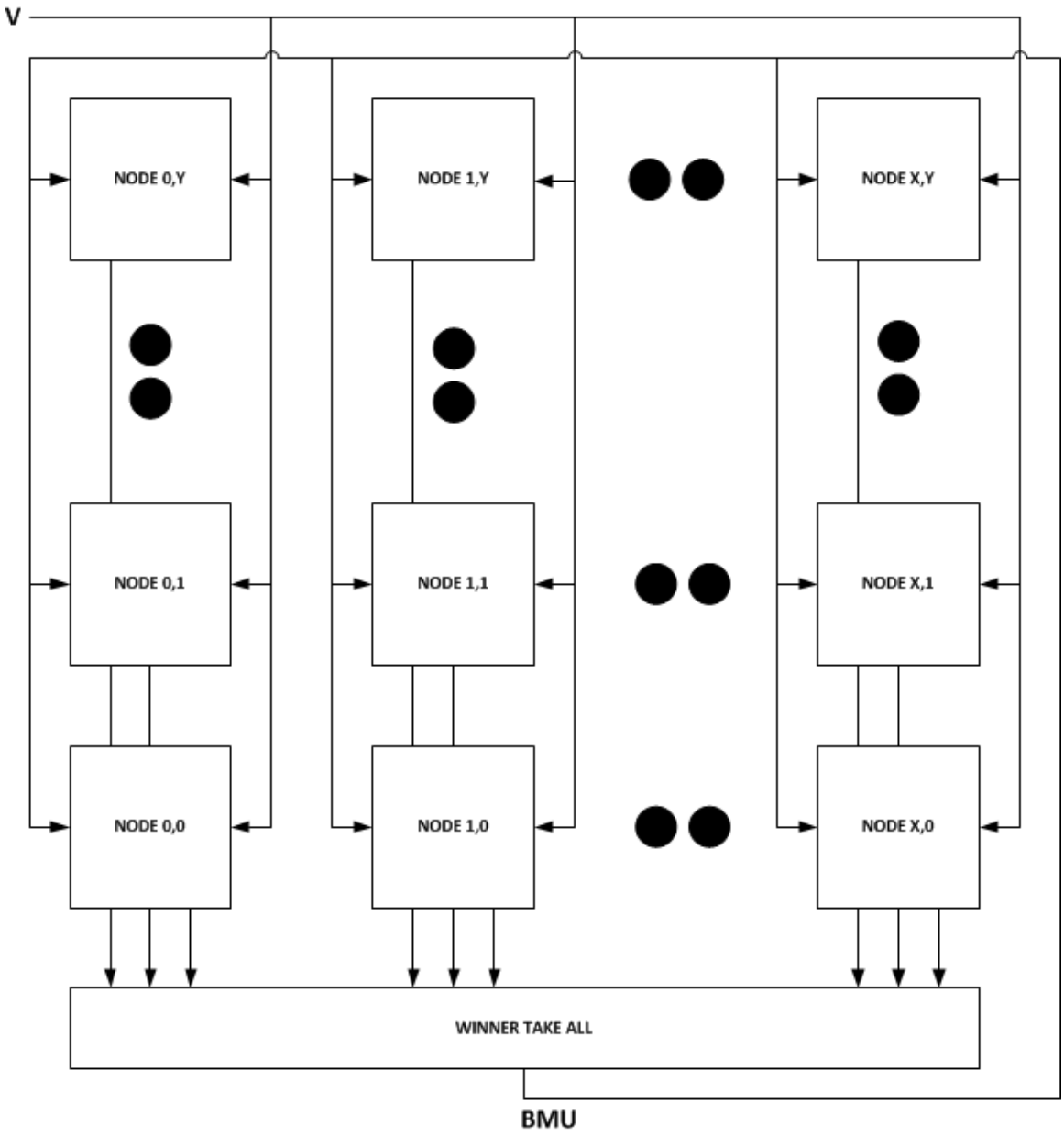


Figure 9 SOM Hardware Architecture

To achieve this, each node in the SOM must contain the logic required to calculate the distance between its own weight and each input vector as well as the amount that its

weight should be updated for each input vector. In addition, hardware must be implemented that can find the BMU based on the input weight distance produced by each of the SOM nodes. Based on the SOM implementation survey performed, a good choice for each of these functions is the Manhattan distance, the negative power of two based weight update function, and the bit serial BMU search respectively. By combining the distance and weight update implementation described in [4] with the BMU search implementation described in [2], these design choices can be realized. Therefore, this combination of designs was used when developing a conventional SOM implementation that served as the basis for the port agent SOM implementation.

4.1 CONVENTIONAL SOM IMPLEMENTATION

SOMs are specified by their dimensions (typically as X by Y) and the format of the feature vectors they process. The specification provided for the port agent was for a 16 node by 16 node SOM with 8 x 8-bit (64-bit) feature vectors. The conventional SOM implementation resulting from a combination of this specification and the background research is described by starting with the top level view of the hardware architecture and then delving into individual subsystems so that each functional unit can be explained.

The top level SOM architecture is shown in Figure 10.

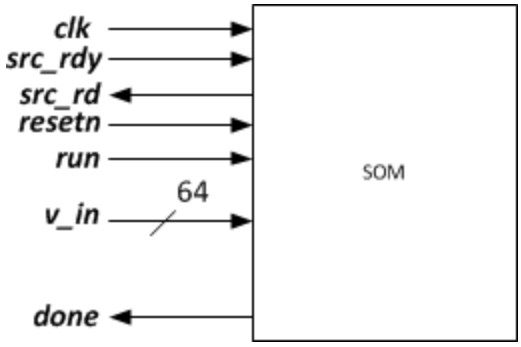


Figure 10 Conventional SOM Top Level Architecture

There are seven input/output (I/O) signals used to interface with the SOM as described in Table 2.

Table 2 Conventional SOM Top Level Signal Description

Signal	**Width**	**Description**
clk	1	Clock source for the SOM
src_rdy	1	Indicates the user is ready to input v_in
src_rd	1	Indicates the SOM is ready for v_in to be input
v_in	64	Training feature vector input
resetn	1	Asynchronous reset
run	1	Initiates execution of the SOM algorithm
done	1	Indicates the SOM execution is complete

The next level down in the conventional SOM hardware architecture is the datapath/controller (see Figure 11). The datapath contains all of the processing elements of the SOM. The controller manages the operation of the datapath and its functional units. Figure 11 shows the I/O signaling described previously as well as the signals that communicate information between the controller and the datapath. The purpose of each of these will be made clear later when the datapath's architecture is deconstructed.

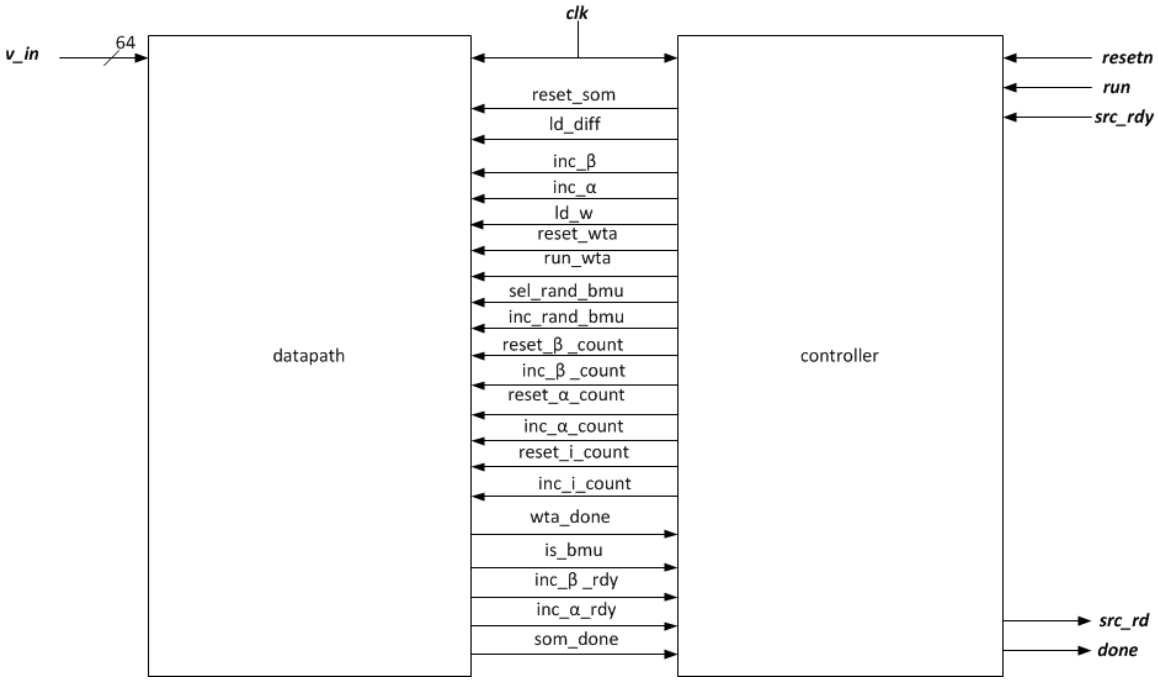


Figure 11 Conventional SOM Datapath/Controller Architecture

The datapath for the SOM contains the topologically fixed architecture of 16 x 16 processing nodes along with a WTA and parameter scheduler as shown in Figure 12.

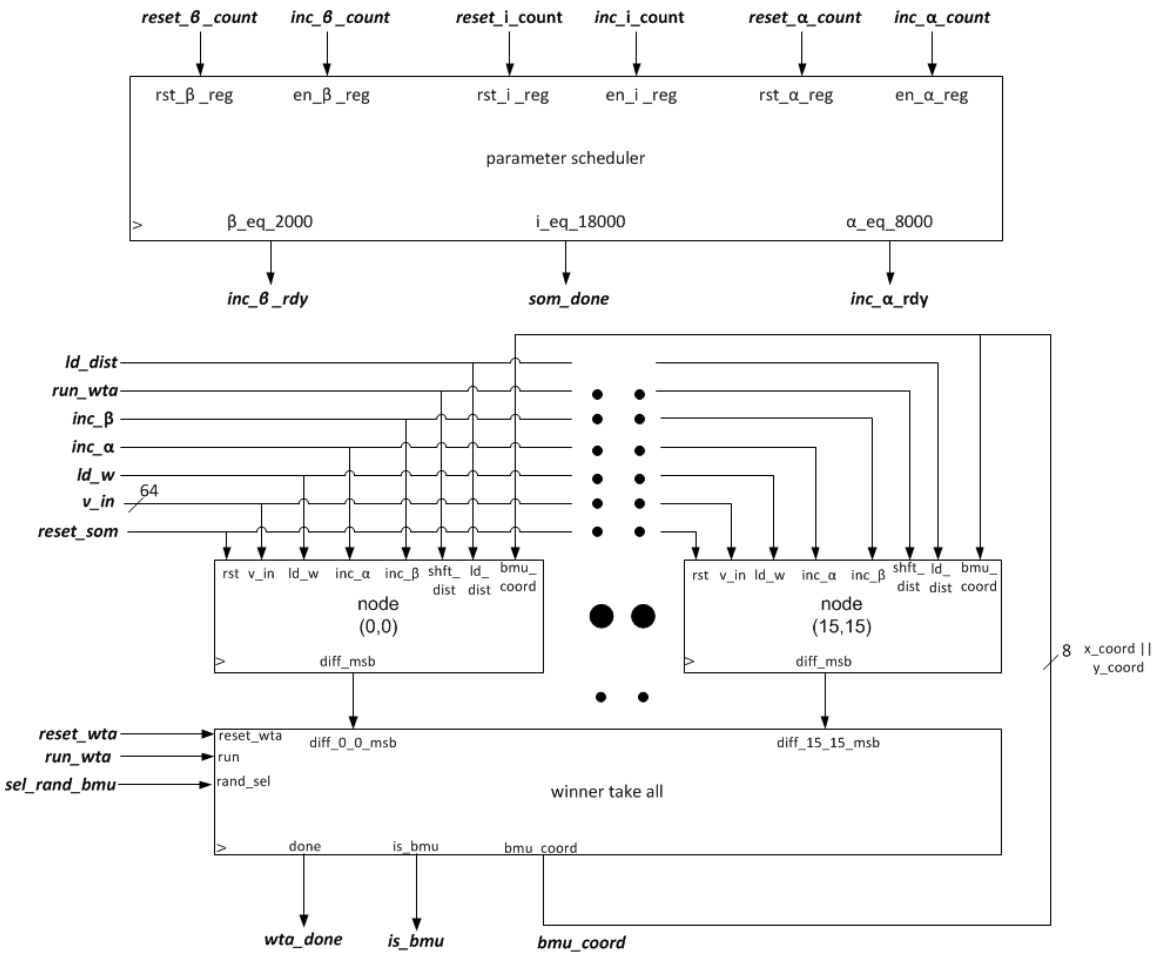


Figure 12 Conventional SOM Datapath

The purpose of the parameter scheduler logic (see Figure 13) is to maintain three counters i, α, and β used by the weight update function. The i counter keeps track of how many feature vectors the SOM has processed and can be thought of as the discrete time coordinate described in Chapter 2. The α and β counters maintain a count of the number

of feature vectors processed using the present α and β parameter values. The scheduler utilizes comparators to set flags to indicate to the SOM controller that 2,000 feature vectors have been processed with the present value of β, 4,000 feature vectors have been processed using the present value of α, and 18,000 feature vectors have been processed in total and the SOM processing is complete.

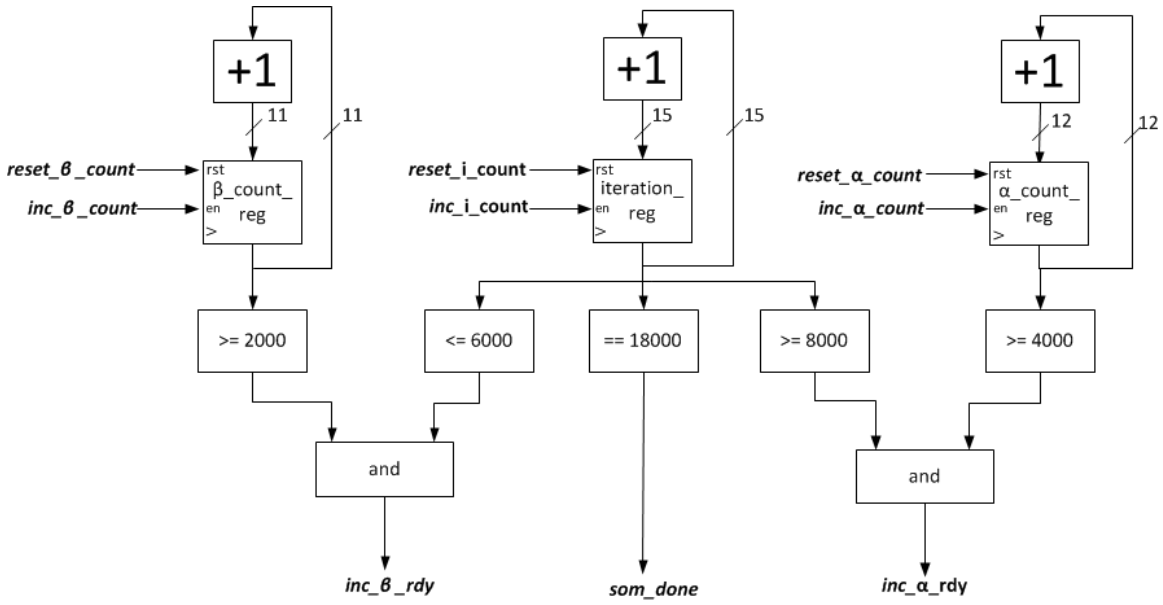


Figure 13 Conventional SOM Parameter Scheduler

The α and β parameter values are maintained by each of the nodes in their weight update logic and are used to calculate the neighborhood kernel (see Figure 14).

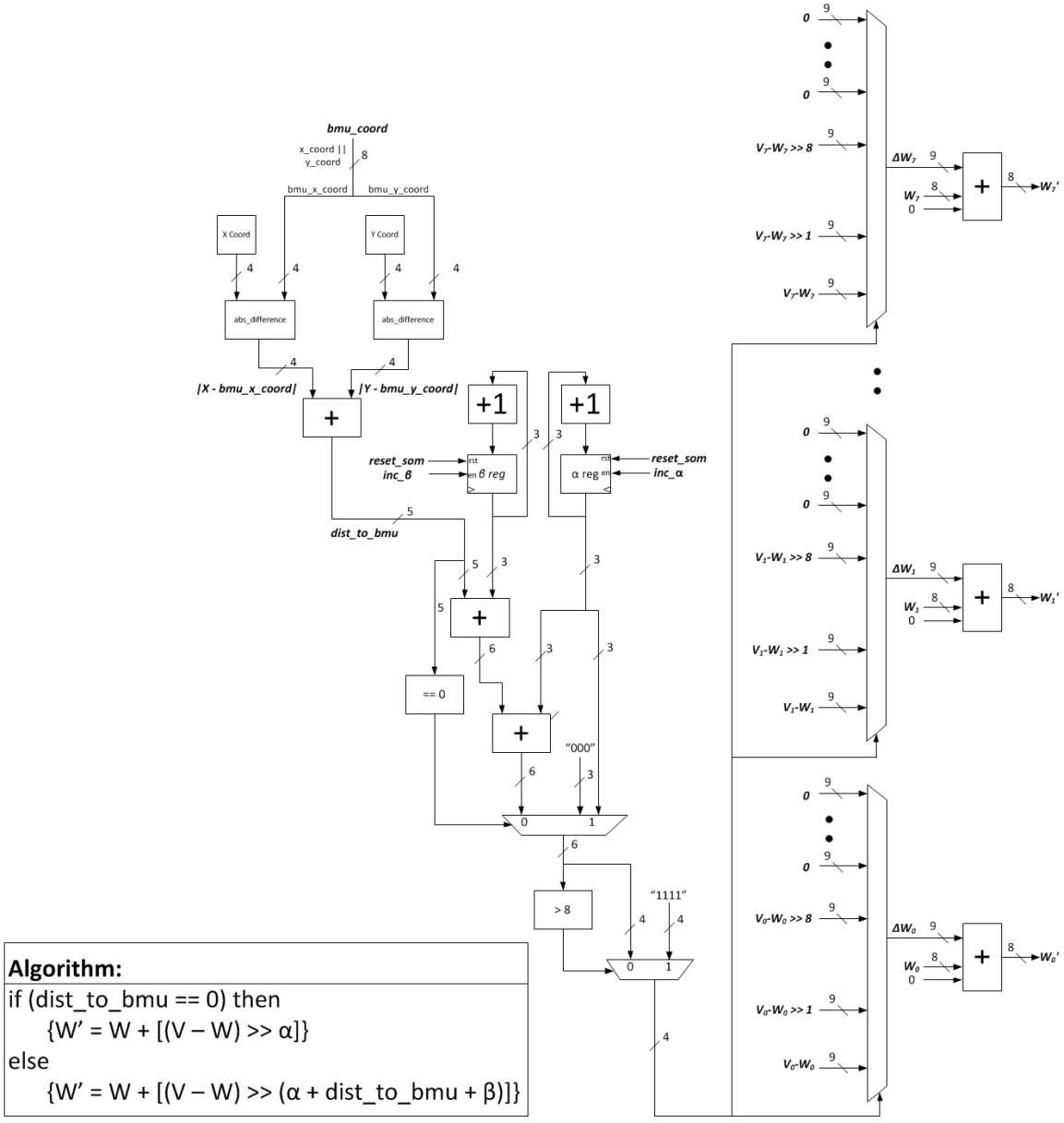


Figure 14 Conventional SOM Weight Update Logic

As shown in Equation 3 in Chapter 2, the weight update logic in each node calculates the offset $\Delta W_j = h_{cj}(t)[V(t) - W_j(t)]$ to apply to the weight of the node. To eliminate the

need for multiplication logic while still allowing for a monotonically decreasing learning rate and neighborhood size, a negative power of two implementation of the neighborhood kernel $h_{cj}(t)$ from [4] was selected. This function is shown in Equation 7.

$$h_{cj}(t) = \begin{cases} \left(\frac{1}{2}\right)^{\alpha} & if\ j = j^* \\ \left(\frac{1}{2}\right)^{\alpha + D(R_j, R_{j^*}) + \beta} & if\ j \neq j^* \end{cases} \tag{7}$$

In Equation 7, j^* represents the BMU, j represents the node being updated, and $D(R_j, R_j*)$ represents the Manhattan distance between the two nodes (see Equation 6). $D(R_j, R_j*)$ is labeled as *dist_to_bmu* in Figure 14. Because the parameter scheduler increases the value of α and β over time, $h_{cj}(t)$ monotonically decreases over time as desired.

The Manhattan distance is also used to determine the distance between the input vector presented to the SOM and the weight of each element in the SOM for determining the BMU. This logic is shown in Figure 15.

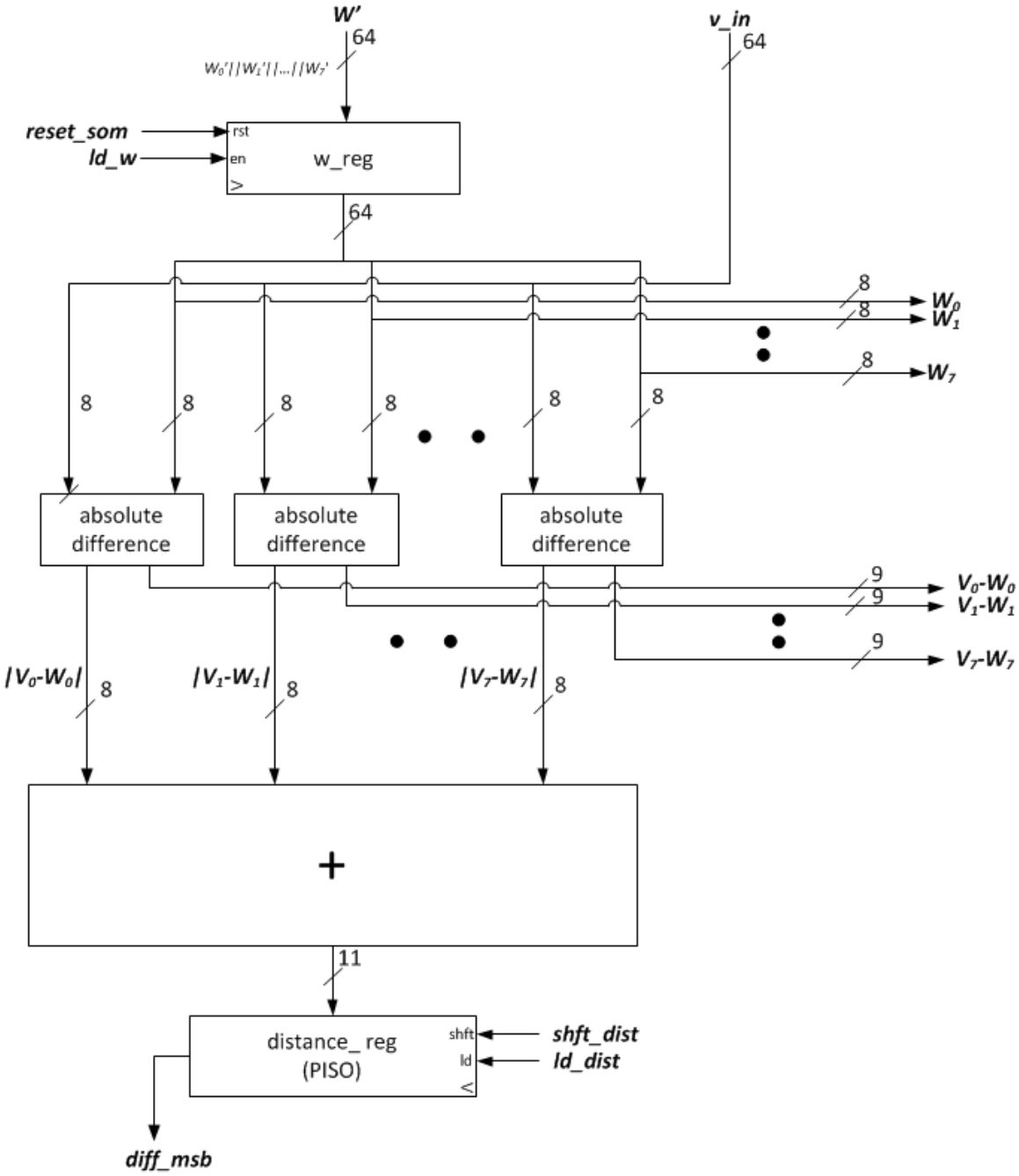


Figure 15 Conventional SOM Distance Calculation Logic

The Manhattan distance between the input vector and the node weight is stored in the Parallel In Shift Out (PISO) register labeled *distance_reg* in Figure 15. This register allows the distance calculated at each node to be processed serially (one MSB at a time) by the WTA circuit.

The distance calculation requires as intermediate values absolute difference and signed difference values using the absolute difference logic as shown in Figure 16.

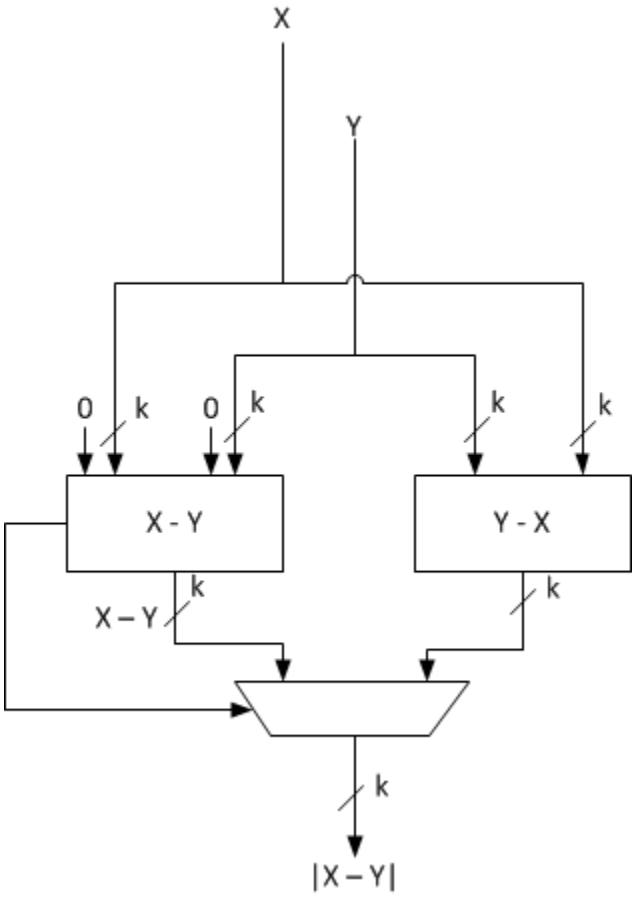


Figure 16 Conventional SOM Absolute Difference Logic

The value calculated by the distance logic and stored in the distance register in each of the 256 SOM nodes is processed by the WTA logic (see Figure 17) to search for a BMU.

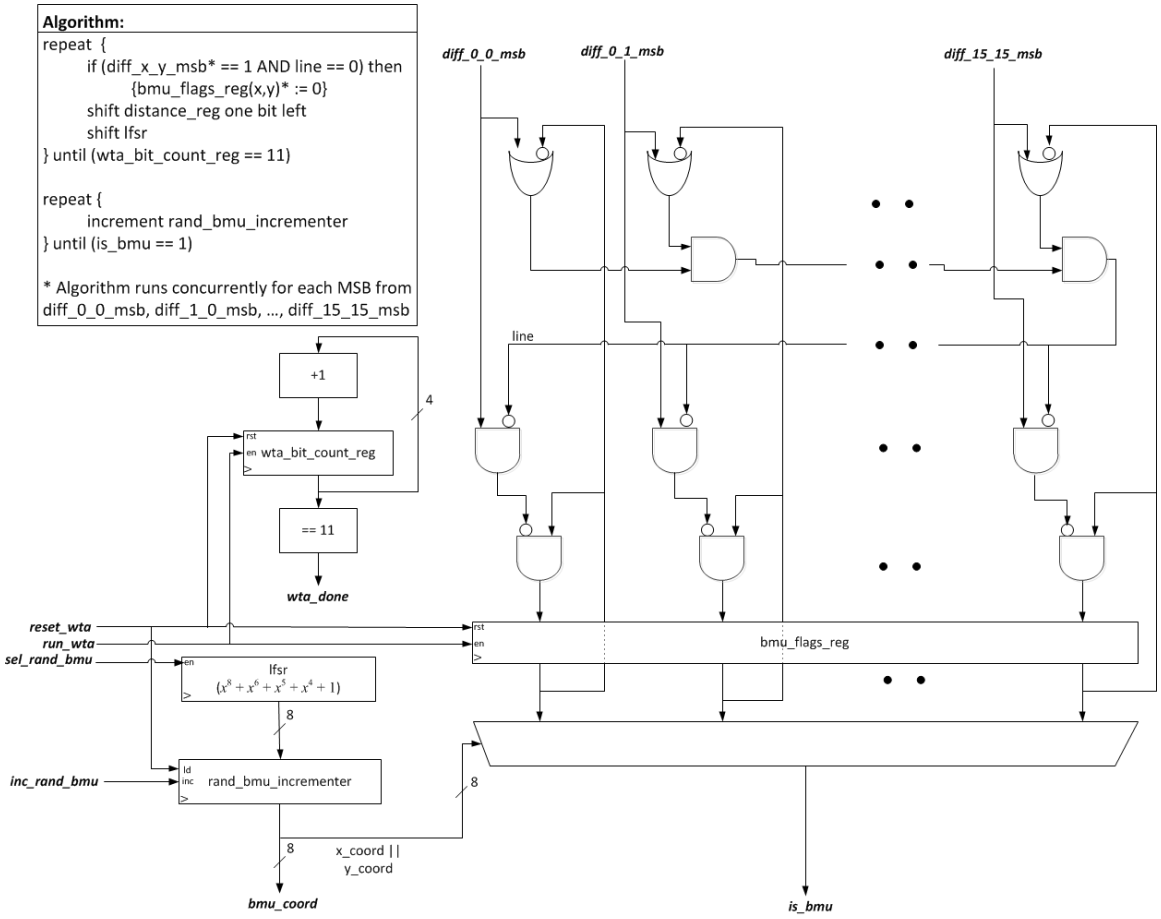


Figure 17 Conventional SOM WTA Logic

The WTA logic is based on the improved BMU selection circuit shown in Figure 8 and described in Chapter 3 but with additional logic added to count the number of bits

processed and to randomly select a BMU when there are ties. Because the Manhattan distances calculated at each node are eleven bits wide (see Equation 6 and Figure 15), the WTA circuit is clocked eleven times with the signal labeled *run_wta* in Figure 17 set. After all eleven MSBs for each node have been processed; a node is indicated as a BMU if its flag in the BMU flags register is set. Because there is an opportunity for ties where more than one BMU flag is set, tie breaking logic was designed. The tie breaking logic in the WTA circuit is a random number generator consisting of an 8-bit Linear Feedback Shift Register (LFSR). The LFSR is used to select a starting point to find a BMU flag that is set. This value is stored and then incremented until a set BMU flag is found. This may require up to 256 attempts to find a BMU flag that is set. Once a set BMU flag is found the BMU search is complete as indicated by the signal labeled *is_bmu* being set. The value labeled *bmu_coord* contains the coordinates of the BMU.

All of the functional units that comprise the datapath are controlled by an algorithmic state machine (ASM). The detailed ASM implemented in the conventional SOM controller is shown in Figure 18.

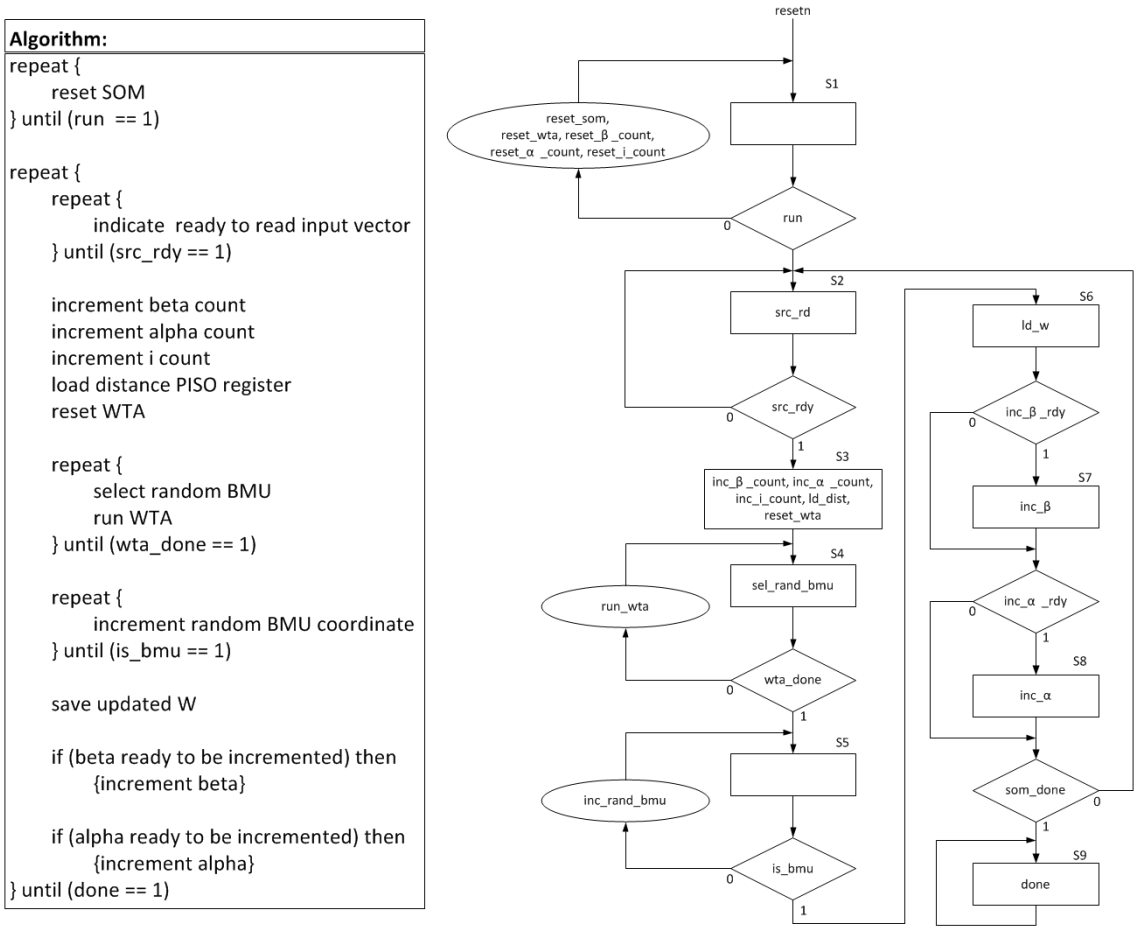


Figure 18 Conventional SOM Controller ASM

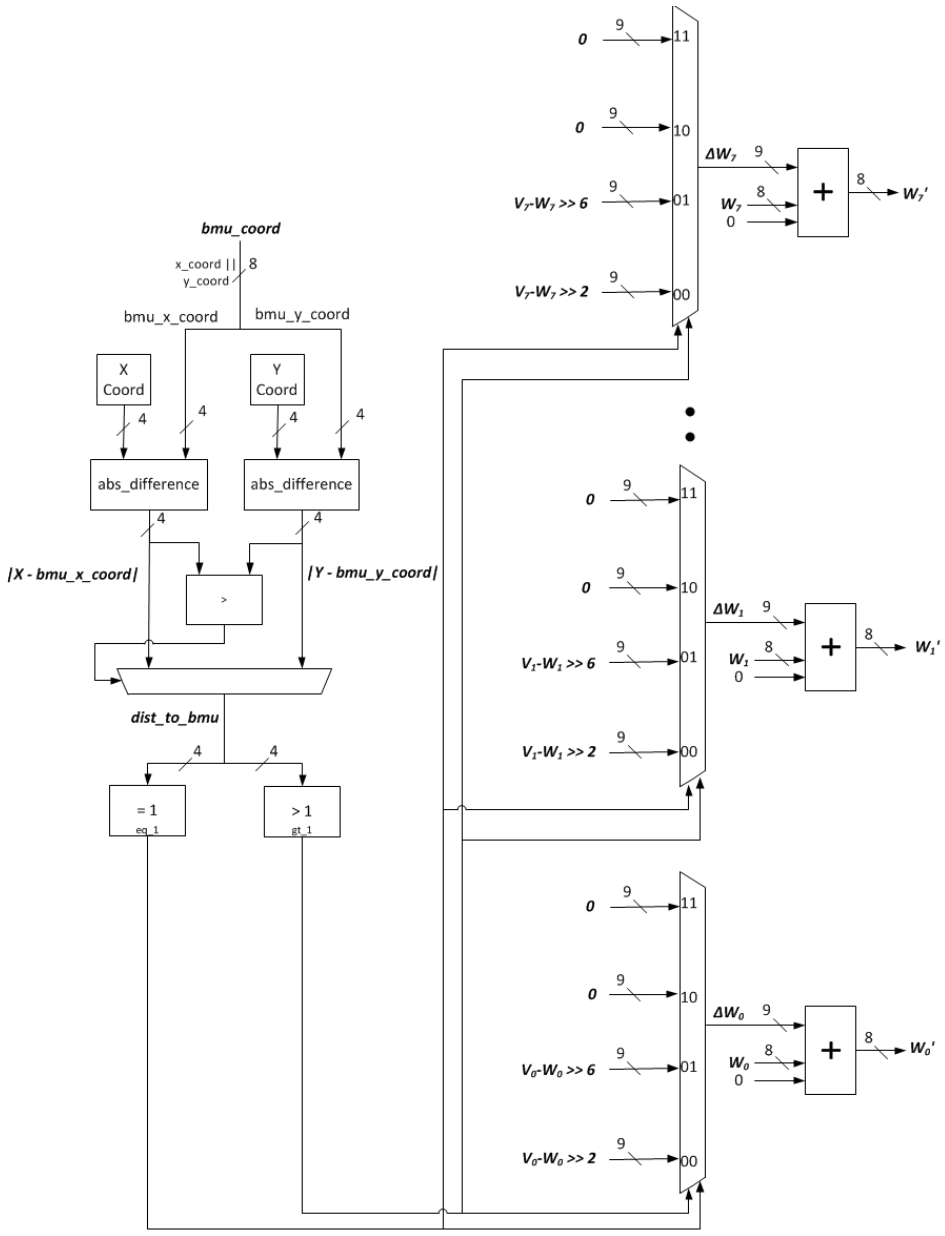
4.2 PORT AGENT ARCHITECTURE SOM IMPLEMENTATION

The port agent SOM is a modified version of the conventional SOM that has been optimized for area and speed by simplifying the weight update logic. The conventional SOM implementation weight update function is dependent on a neighborhood size that decreases monotonically over time making for a relatively complex weight update circuit (see Figure 14). However, the port agent SOM is initialized with weights from a trained

map so that it can use a neighborhood function that stays constant over time as shown in Equation 8.

$$h_{cj}(t) = \begin{cases} \left(\frac{1}{2}\right)^2 & if\ D(R_j, R_j*) == 0 \\ \left(\frac{1}{2}\right)^6 & if\ D(R_j, R_j*) == 1 \\ 0 & if\ D(R_j, R_j*) > 1 \end{cases} \quad (8)$$

The port agent SOM's neighborhood function (Equation 8) is derived from the conventional SOM's neighborhood function (Equation 7) by assuming the maximum values of α and β in the conventional SOM implementation (two and three respectively). Because α and β are not required in the port agent SOM's neighborhood function, the parameter scheduler can be eliminated from the implementation and the complexity of the weight update logic is reduced significantly (see Figure 19).



Algorithm:

```
if (dist_to_bmu == 0) then
    {W' = W + [(V – W) >> 2]}
else if (dist_to_bmu == 1) then
    {W' = W + [(V – W) >> 6]}
else
    {W' = W}
```

Figure 19 Port Agent SOM Weight Update Logic

Many levels of the SOM architecture were affected by the removal of the parameter scheduler. For instance, signaling was removed between the datapath and controller (see Figure 20) that was previously required to maintain the values of i, α, and β.

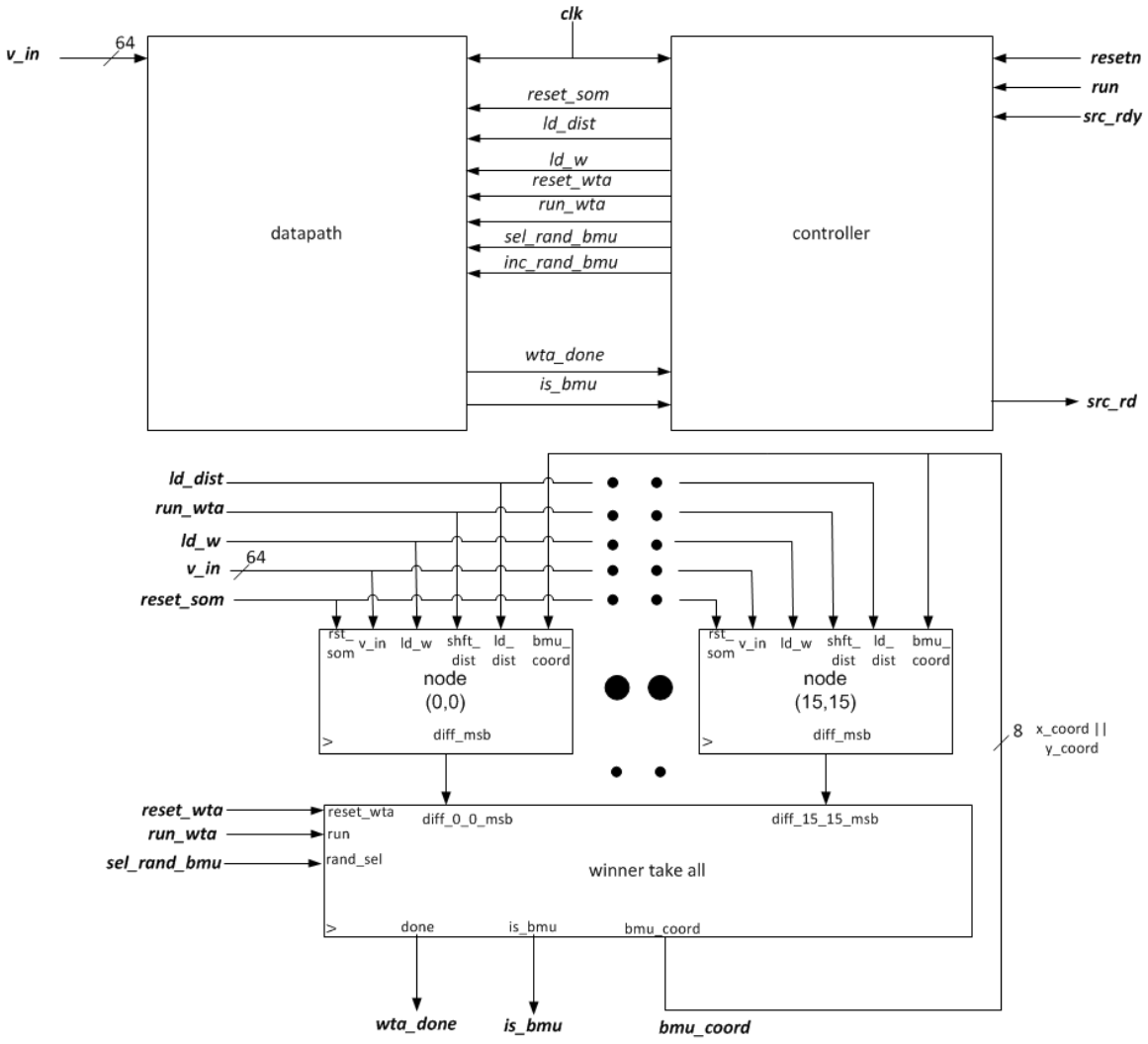


Figure 20 Port Agent SOM Datapath/Controller(top) and Datapath (bottom)

With the removal of a significant amount of logic from the datapath, the ASM controller for the port agent SOM was able to be simplified as well (see Figure 21).

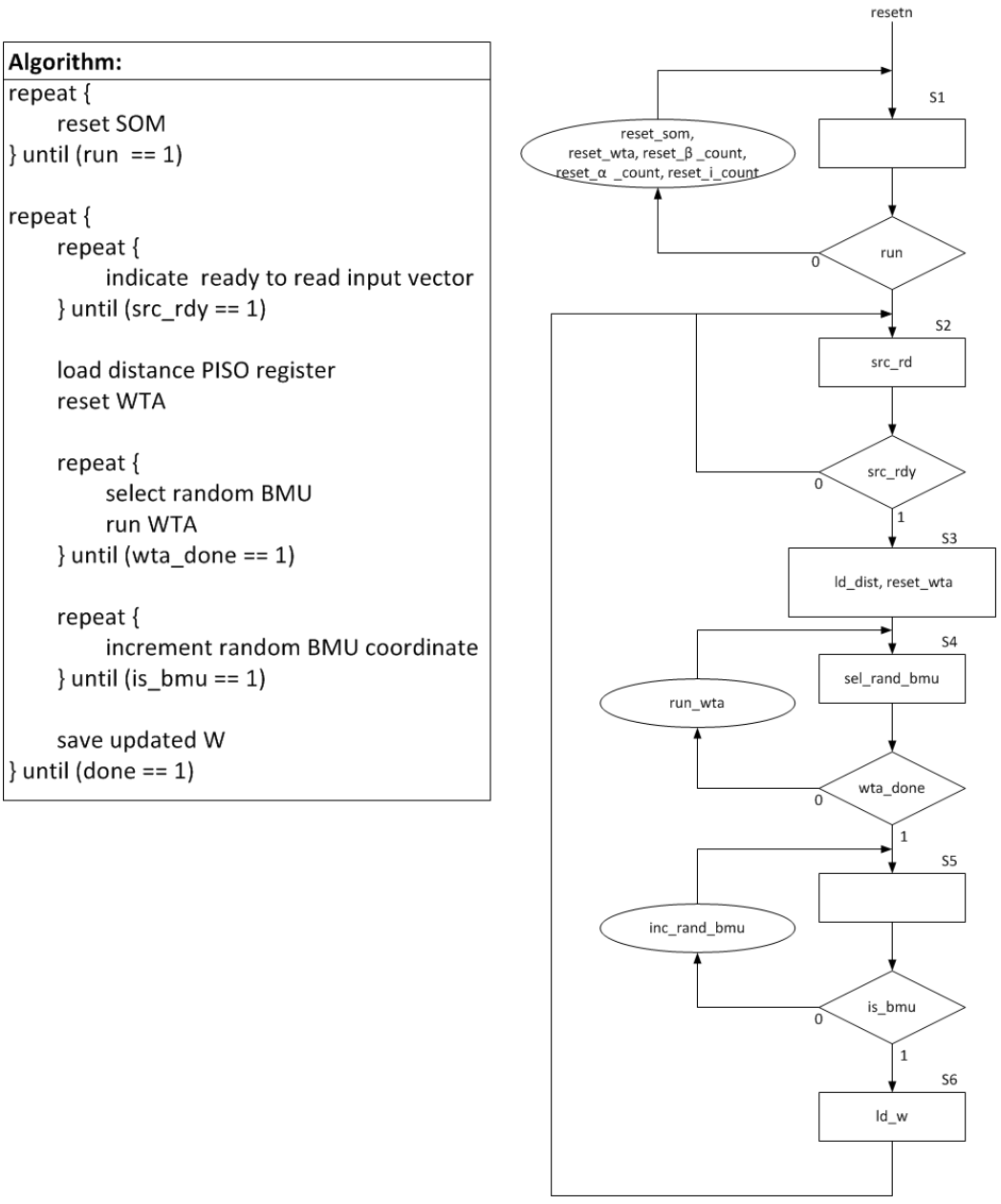


Figure 21 Port Agent Architecture SOM Controller ASM

The optimizations made to the conventional SOM architecture provided a significantly smaller and faster implementation for the port agent SOM. Quantitative evidence of this is provided in the next chapter.

5 PORT AGENT SOM DESIGN VERIFICATION

After the port agent SOM design was complete it was implemented in VHDL and verified through simulation using Aldec Active-HDL 8.3 SP1. Design verification was accomplished by first preloading the port agent SOM matrix with a map generated using the software implementation from [3] and then presenting vectors to the SOM for processing. The map updates produced from these vectors were validated with results from a software implementation of the weight update function.

5.1 TEST VECTOR SELECTION

The test vectors presented to the port agent SOM were chosen to verify the three scenarios possible when processing input vectors with the port agent SOM (see Figure 22).

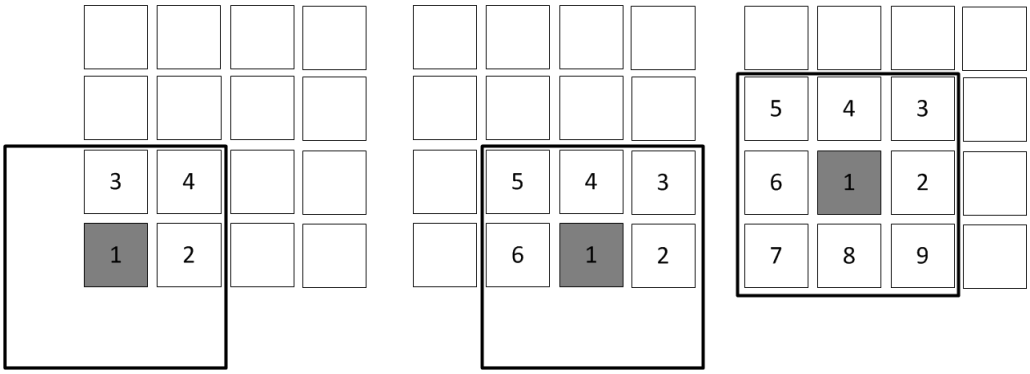


Figure 22 Port Agent SOM Test Case 1 (left), 2 (middle), 3 (right)

Because the neighborhood size in the port agent SOM has a constant radius of one, the neighborhoods that result from an input vector can contain 4, 6, or 9 elements depending on where the BMU is found. Based on the pre-trained map that was loaded into the port agent SOM, three specific test vectors were chosen to ensure each of these cases was tested.

As described in the previous chapter, the SOM that was instantiated for simulation was composed of a 16 node by 16 node matrix designed to process 8 x 8-bit (64-bit) feature vectors. Because of the size of the map and its feature vectors, it is difficult to display the entire port agent SOM matrix. However, for the verification process it is only necessary to display the portions of the map that were affected by the input vectors. Figure 23 shows the subset of the port agent SOM that was exercised during validation after being initialized with a pre-trained map.

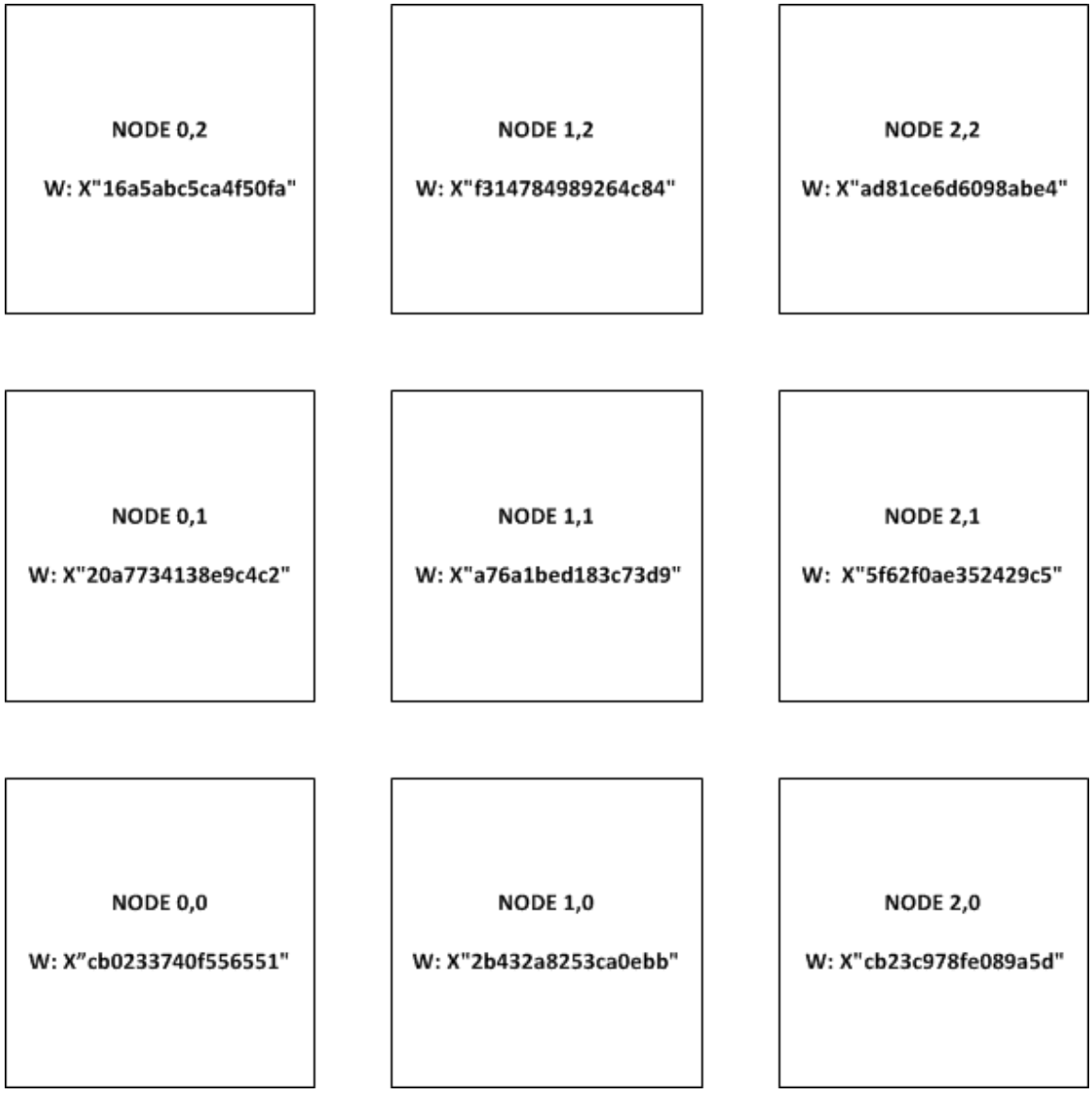


Figure 23 Pre-trained Port Agent SOM Partial Map

In order to ensure the three cases in Figure 22 were tested, the test vectors chosen were V_1: X"cb0233740f556551", V_2: X"2b432a8253ca0ebb", and V_3: X"a76a1bed183c73d9" for test case one, two, and three respectively. These vectors correspond to the initial

values for nodes (0,0), (1,0), and (1,1) guaranteeing they would be the BMU for each vector allowing each of the three cases in Figure 22 to be tested.

5.2 TEST VECTOR PROCESSING

The vector V_1: X"cb0233740f556551" was presented as the first test vector to the port agent SOM. As predicted, the BMU was determined by the port agent SOM to be node (0,0) and its neighborhood included nodes (0,0), (1,0), (0,1), and (1,1). The map that resulted from this test vector is shown in Figure 24.

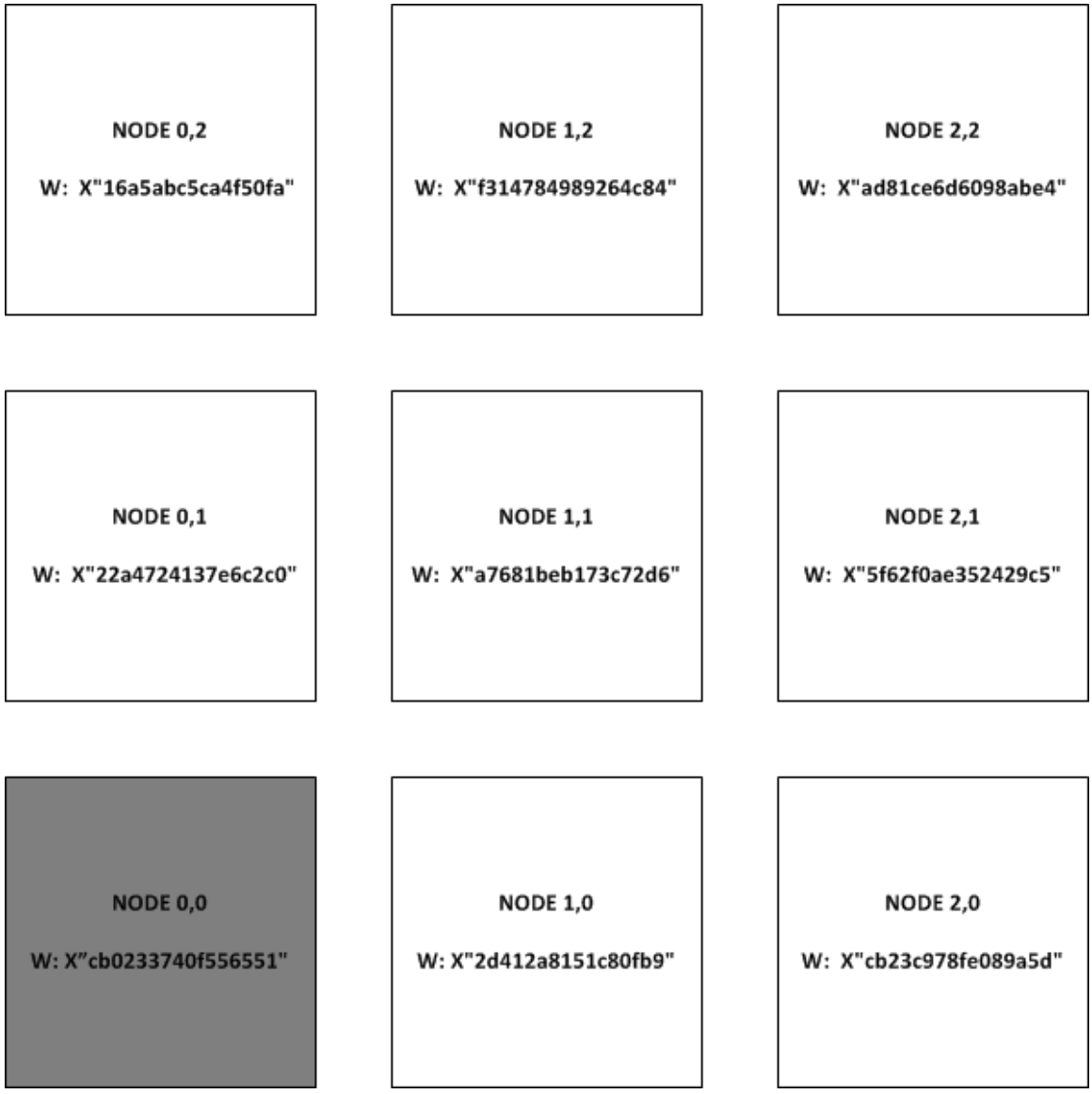


Figure 24 Port Agent SOM Resulting from V_1

It is evident from Figure 24 that the nodes in the neighborhood around the BMU node (0,0) changed from their initial value and all nodes outside of the neighborhood remained unchanged. Recall from the previous chapter that in the port agent SOM, each node in the

neighborhood is updated by adding $(V - W) \gg 2$ when the node being updated is the BMU; $(V - W) \gg 6$ when the node is a distance of one from the BMU; and zero when the node being updated is greater than a distance of one from the BMU. For instance, the updated value of node (1,0) is X"2d412a8151c80fb9" from its initial value of X"2b432a8253ca0ebb" for the input vector V_1: X"cb0233740f556551". Because node (1,0) is a distance of one from the BMU for V_1, $(V - W) \gg 6$ must be added to the initial value of node (1,0) to obtain its new weight. The calculations performed by the port agent SOM are shown in Table 3 for test vector V_1.

Table 3 Weight Update Validation Example

V_1 (hex)	**cb**	**02**	**33**	**74**	**0f**	**55**	**65**	**51**
W (hex)	**2b**	**43**	**2a**	**82**	**53**	**ca**	**0e**	**bb**
V (dec)	203	2	51	116	15	85	101	81
W (dec)	43	67	42	130	83	202	14	187
V – W (dec)	160	-65	9	-14	-68	-117	87	-106
V – W >> 6 (dec)	2	-2	0	-1	-2	-2	1	-2
W’ = W +[(V-W) >> 6] (dec)	45	65	42	129	81	200	15	185
W’ = W +[(V-W) >> 6] (hex)	**2d**	**41**	**2a**	**81**	**51**	**c8**	**0f**	**b9**

The remaining two cases were run similarly. Test vector V_2: X"2b432a8253ca0ebb" was presented as the second test vector to the port agent SOM that resulted from test vector V_1. As expected, the BMU was determined by the port agent SOM to be node (1,0) and its neighborhood included nodes (0,0), (1,0), (2,0), (0,1), (1,1), and (2,1). The map that resulted from this test vector is shown in Figure 25.

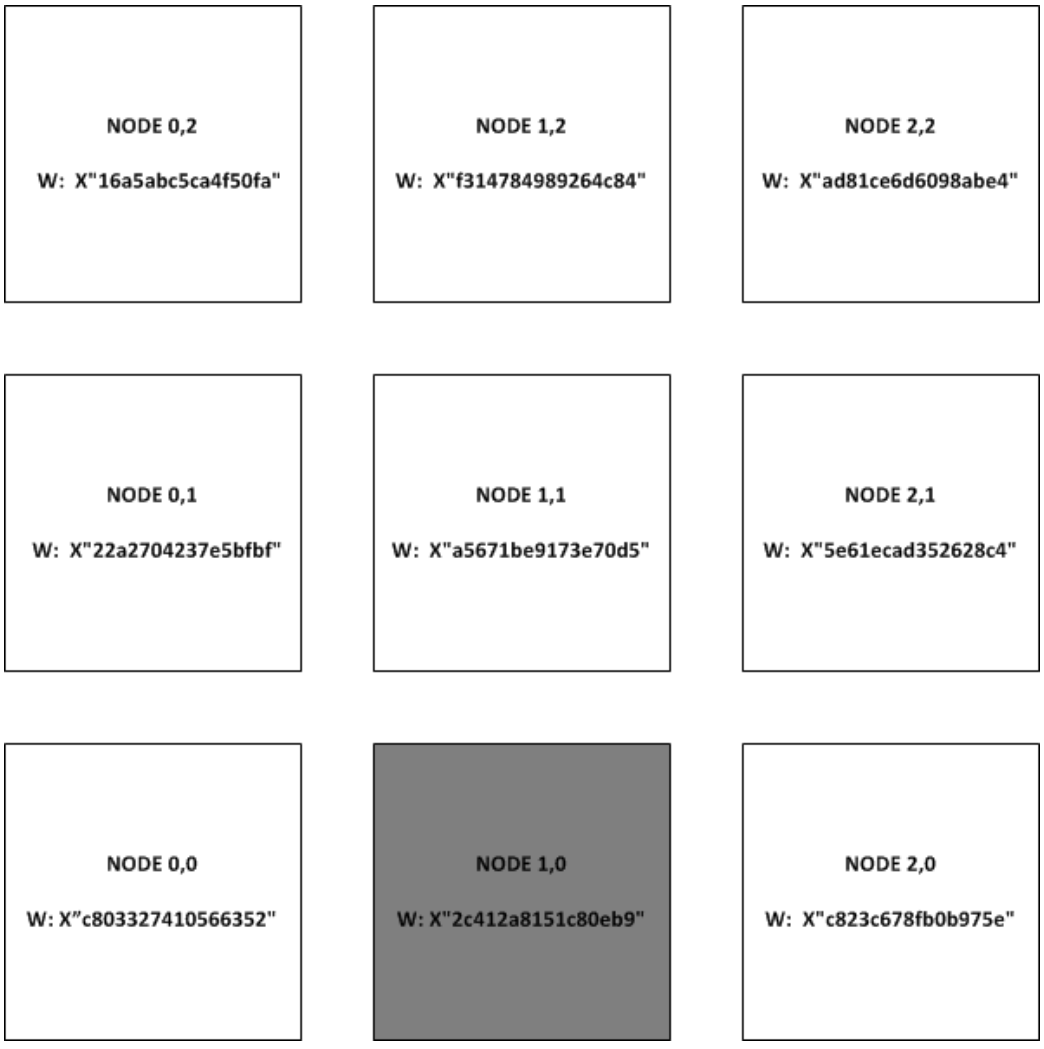


Figure 25 Port Agent SOM Resulting from V_2

The final test vector presented was V_3: X"a76a1bed183c73d9" and was presented to the map resulting from V_2. The BMU was determined by the port agent SOM to be node (1,1) and its neighborhood included nodes (0,0), (1,0), (2,0), (0,1), (1,1), (2,1), (0,2), (1,2), and (2,2). The map that resulted from this test vector is shown in Figure 26.

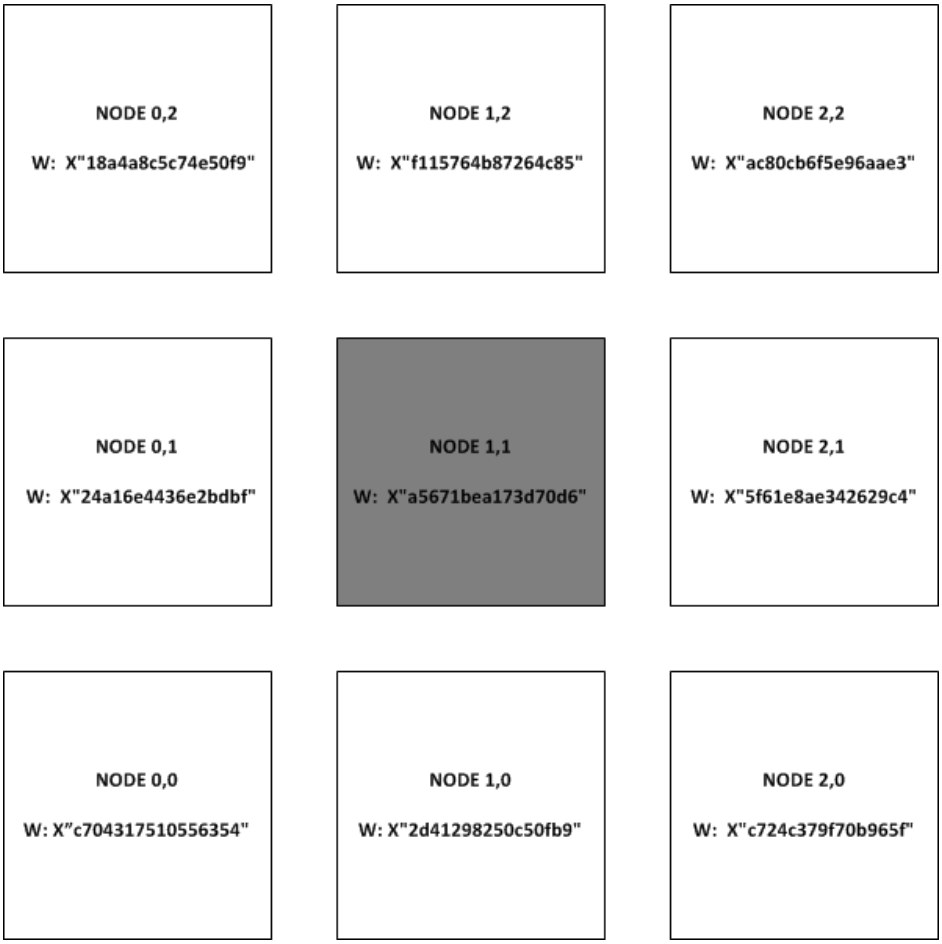


Figure 26 Port Agent SOM Resulting from V_3

The port agent SOM values resulting from the three test vectors for the three different test cases were confirmed with a software implementation of the weight update function validating the port agent SOM design.

6 RESULTS AND ANALYSIS

In order to demonstrate the improvements in size and speed achieved by optimizing the conventional SOM design for the requirements of the port agent SOM, both designs were described with VHDL, implemented, and optimized. Both designs were targeted for the Xilinx Virtex-6 and Altera Stratix IV FPGA parts. The toolset used for implementation was the Xilinx ISE version 12.4 for the Virtex-6 and Altera Quartus II version 10.1 for the Stratix IV. The Automated Tool for Hardware EvaluatioN (ATHENa) version 0.6.1 was used to optimize both implementations [16][17].

6.1 IMPLEMENTATION RESULTS

The smallest and fastest implementations produced by ATHENa for both designs for the Virtex-6 xc6vlx760 are compared in Table 4 and Table 5 respectively.

Table 4 Minimum Area Implementations for the Virtex-6

Resource	Conventional SOM	Port Agent SOM	Delta
LUTs	123,962	91,832	-32,130 (-26%)
Slices	41,696	25,565	-16,131 (-39%)
Flip Flops	23,620	23,575	-75 (-0.3%)
Max Frequency	46.1 MHz	105.3 MHz	+59.72 MHz (+128%)

Table 5 Maximum Throughput Implementations for the Virtex-6

Resource	Conventional SOM	Port Agent SOM	Delta
LUTs	127,318	91,832	-35,486 (-28%)
Slices	45,909	25,565	-20,344 (-44%)
Flip Flops	23,624	23,575	-49 (-0.2%)
Max Frequency	51.6 MHz	105.3 MHz	+53.7 MHz (+104%)

In addition to being targeted for the Virtex-6 xc6vlx760, the conventional SOM and the port agent SOM were implemented for the Stratix IV GT ep4s100g4f45i1. The results for the smallest and fastest implementations produced by ATHENa for both designs for the Stratix IV GT ep4s100g4f45i1 are shown in Table 6 and Table 7 respectively.

Table 6 Minimum Area Implementations for the Stratix IV

Resource	Conventional SOM	Port Agent SOM	Delta
ALUTs	122,186	82,863	-39,323 (-32%)
Logic Util	145,447	88,774	-56,673 (-39%)
Flip Flops	19,529	19,482	-47 (-0.2%)
Max Frequency	105.2 MHz	160.1 MHz	+54.9 MHz (+52%)

Table 7 Maximum Throughput Implementations for the Stratix IV

Resource	Conventional SOM	Port Agent SOM	Delta
ALUTs	124,611	82,863	-41,748 (-34%)
Logic Util	156,035	88,774	-67,261 (-43%)
Flip Flops	19,529	19,482	-47 (-0.2%)
Max Frequency	116.3 MHz	160.1 MHz	+43.8 MHz (+38%)

Clearly the port agent SOM was significantly smaller and faster than the conventional SOM implementation on the Virtex-6 and Stratix IV. This is largely due to the improvements made to the SOM node implementation. Since there are 256 nodes in the implementation, small reductions in each node produced a large overall improvement in the size and speed of the design.

6.2 IMPLEMENTATION ANALYSIS

The port agent SOM must be able to process 2,976,190 vectors/second in order to detect anomalies in 1 Gbps Ethernet traffic. This equates to 336 ns of allowable latency for each vector [2]. The latency in the port agent SOM is calculated by multiplying the number of clock cycles required to process each vector with the clock period for the implementation. On average, each vector processed by the port agent SOM requires 142 clock cycles to produce a result. The majority of the clock cycles are needed to randomly select a BMU. Recall from Chapter 4 that the WTA circuit requires 11 clock cycles to determine one or more BMUs for an input vector and may require up to 256 clock cycles to randomly select a BMU if there is a tie. However, if we assume that on average it requires 128 clock cycles to break a tie, BMU selection requires 139 clock cycles. The remaining three clock cycles are used to read the input vector, initialize the WTA circuitry, and then store the updated node weights (see Figure 21). Based on the number of clock cycles required to process each vector and the maximum frequency for the Virtex-6 and Stratix IV implementations, the time required to process each input vector on the targeted devices is 1,349 ns and 887 ns respectively. This equates to 741,549 vectors/second for the Virtex-6 and 1,127,464 vectors/second on the Stratix IV. In both cases the performance of the port agent SOM implementation does not meet the requirement to process 1 Gbps Ethernet traffic due in large part to the latency incurred when finding a random BMU. A solution to this problem has been devised and will be discussed in more detail in the next chapter.

Although the port agent SOM hardware implementation cannot process 1 Gbps Ethernet traffic without additional research and development, it is significantly closer to meeting the requirement than the software implementation described in [3] that was used to validate the approach even when running on a high performance platform. The software implementation of the port agent SOM was evaluated on a Linux Fedora Core 13 Virtual Machine (VM) on an eight core Dell T7400 with 6GB of memory. The VM was configured to have two CPUs and 2GB of memory, and each CPU in the VM was a four core E4520 running at 2.5 GHz. Using this hardware, the software port agent SOM application was demonstrated to process one million feature vectors in 13.9 seconds or 71,942 vectors/second. This is less than $1/10^{th}$ the performance of the SOMs implemented in hardware for this thesis and highlights the need for a hardware implementation of the SOM for the network intrusion detection application.

7 FUTURE WORK

Although the port agent SOM shows promise for use in the intended network intrusion detection system, the design must be improved somewhat before it can be deployed. At a minimum, the port agent SOM must be improved so its WTA logic can find the BMU faster and it must allow for its initial map to be externally configurable. Both of these improvements will be described in more detail in the subsequent sections.

7.1 BMU Selection Time

Much of the latency in both the conventional SOM implementation and the port agent SOM implementation described in Chapter 4 is the number of cycles required to determine the coordinates of the BMU once they have been calculated. This is because the BMU is stored in a 256-bit flag register that must be serially searched over 256 clock cycles for the winner once it has been calculated (see Figure 27).

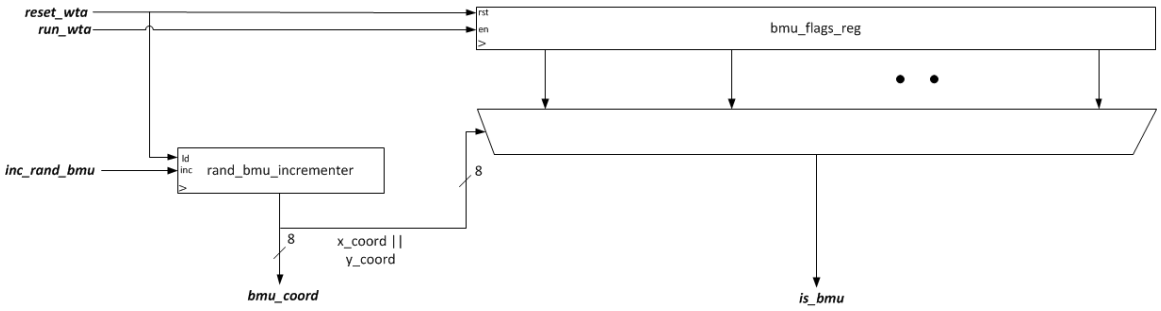


Figure 27 BMU Search Circuit

While this simple calculation can be performed very quickly by inspection using a 256 input multiplexer, it is currently clocked at the same speed as the rest of the SOM circuitry. Static timing analysis has demonstrated that the BMU search circuitry can be clocked at 517 MHz on the Virtex-6 which significantly reduces the penalty paid for clocking the circuit up to 256 times to determine the coordinates of the BMU. If the BMU search circuit was clocked at this rate for the 128 cycles it requires on average to find a BMU, the time required to process each input vector would be reduced from 1,349 ns to 381 ns on the Virtex-6. Assuming that the BMU search circuit can be clocked as fast as the Virtex-6 on the Stratix IV, as was the trend with the other port agent SOM circuitry, the time required to process each input vector would be reduced from 887 ns to 335 ns, allowing for 2,985,074 vectors/second and making it possible for the port agent SOM to process 1 Gbps Ethernet traffic as required for the network intrusion detection application.

7.2 Initial Matrix Weight Configuration

The first step in the SOM algorithm is to establish the initial weights for every element in the SOM matrix. In a conventional SOM this is accomplished by setting each weight to a random value. As described in Chapter 4, the port agent SOM is preloaded with an already trained matrix so that its weight update logic can be simplified significantly. Once realized and tested, the port agent SOM can only be preloaded with one set of weight values. It is evident from the top level architecture (see Figure 20) that there is no I/O port for loading initial weights. In order to support the ability to preload an initial set of weights representing an already trained matrix, I/O ports must be added and the associated logic must be developed for the port agent SOM.

8 CONCLUSIONS

This thesis described the research and development of a hardware implementation of the SOM for a network intrusion detection system. A survey of conventional SOM implementations in hardware resulted in the design of a conventional SOM, which was then modified for use as a detector of anomalous network traffic. The resulting implementation known as the port agent SOM was validated with software and fully implemented along with the conventional SOM for the Xilinx Virtex-6 and Altera Stratix IV FPGA devices. Comparison of the two implementations demonstrated that the port agent SOM required significantly less resources than the conventional SOM and could process input vectors much faster. Additionally, the port agent SOM was demonstrated to process input vectors more than 10 times as fast as a software implementation that provided the same functionality. Based on the number of clock cycles required to process each vector and the clock frequency compatible with the port agent SOM implementation on the Virtex-6 and Stratix IV, the time required to process an input vector on the targeted devices is 1,349 ns and 887 ns respectively. This equates to 741,549 vectors/second for the Virtex-6 and 1,127,464 vectors/second on the Stratix IV. While the current port agent SOM implementation cannot process the 2,976,190 vectors/second required to detect anomalies in 1 Gbps Ethernet traffic, it was shown that with the addition of a faster clock for the BMU search circuit this requirement can be met.

Although work remains to realize a deployable port agent SOM, the research performed for this thesis has succeeded in laying the groundwork for the design of a viable hardware implementation of a SOM for a network intrusion detection system.

REFERENCES

REFERENCES

1. P. Kohlbrenner, B. Roeder, K. Gaj, "Design of a Self Organizing Map (SOM) Based Anomaly Detector for Distributed Network Intrusion Detection," *Technical Report*, George Mason University, 2001.

2. T. Kohonen, *Self-Organizing Maps,* New York: Springer, 2001.

3. P. Kohlbrenner, "Self Organized Maps Verses Off-line Discord Detection for Network Intrusion Detection," *CS 795 Project Report*, George Mason University, 2011.

4. J. Pena, M. Vanegas, A. Valencia, "Parallel FPGA implementation of self-organizing maps," *Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on*, vol., no., pp. 709- 712, 6-8 Dec. 2004.

5. K. Khalifa, B. Girau, F. Alexandre, and M.H. Bedoui, "Parallel FPGA implementation of self-organizing maps," *Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on*, vol., no., pp. 709- 712, 6-8 Dec. 2004.

6. B. Hochet, V. Peiris, G. Corbaz, M. Declercq, "Implementation of a neuron dedicated to Kohonen maps with learning capabilities," *Custom Integrated Circuits Conference, 1990., Proceedings of the IEEE 1990* , vol., no., pp.26.1/1-26.1/4, 13-16 May 1990.

7. D. Macq, M. Verleysen, P. Jespers, J. Didier-Legat, "Analog implementation of a Kohonen map with on-chip learning," *Neural Networks, IEEE Transactions on* , vol.4, no.3, pp.456-461, May 1993.

8. M. Melton, T. Phan, D. Reeves, D. Van den Bout, "The TInMANN VLSI chip," *Neural Networks, IEEE Transactions on* , vol.3, no.3, pp.375-384, May 1992.

9. P. Ienne, M. Viredaz,, "Implementation of Kohonen's self-organizing maps on MANTRA I," *Microelectronics for Neural Networks and Fuzzy Systems, 1994., Proceedings of the Fourth International Conference on*, vol., no., pp.273-279, 26-28 Sep 1994.

10. M. Perez, W. Luque, F. Damiani, "Design of a 4×4 Kohonen neural net-VHDL description," *Devices, Circuits and Systems, 1995. Proceedings of the 1995 First IEEE International Caracas Conference on*, vol., no., pp.135-138, 12-14 Dec 1995.

11. K. Appiah, A. Hunter, M. Hongying, Y. Shigang, M. Hobden, N. Priestley,P. Hobden, C. Pettit, “A binary Self-Organizing Map and its FPGA implementation,” *Neural Networks, 2009. IJCNN 2009. International Joint Conference on* , vol., no., pp.164-171, 14-19 June 2009.

12. M. Porrmann, U. Witkowski, U. Ruckert, “A massively parallel architecture for self-organizing feature maps,” *Neural Networks, IEEE Transactions on*, vol.14, no.5, pp. 1110- 1121, Sept. 2003.

13. H. Onodera, K. Takeshita, K. Tamaru, "Hardware architecture for Kohonen network," *Circuits and Systems, 1990., IEEE International Symposium on*, vol., no., pp.1073-1077 vol.2, 1-3 May 1990.

14. A. Rajah, M. Khalil Hani, "ASIC design of a Kohonen neural network microchip," *Semiconductor Electronics, 2004. ICSE 2004. IEEE International Conference on*, vol., no., pp. 4 pp., 7-9 Dec. 2004.

15. A. Tisan, S. Oniga, C. Gavrincea, A. Buchman, "FPGA implementation of a self-organized map with on-chip learning," *Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference on*, vol., no., pp.81-86, 22-24 May 2008.

16. K. Gaj, J.P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, B.Y. Brewster, “ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware using FPGAs,” 20th International Conference on Field Programmable Logic and Applications, Milano, Italy, Aug. 31st - Sep. 2nd, 2010.

17. About ATHENa. [Online]. Available: http://cryptography.gmu.edu/athena/

CURRICULUM VITAE

Brent W. Roeder graduated from W.T. Woodson High School, Fairfax, Virginia, in 1995. He received his Bachelor of Science in Mathematics from Virginia Tech in 2000 and his Bachelor of Science in Computer and Electrical Engineering from Virginia Tech in 2005. He is currently employed as an engineer by McQ Inc. in Fredericksburg Virginia.