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

by

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

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
What is the significance of this Thesis?

• There is a critical need for the ability to detect malicious network intrusions

• A Self Organizing Map (SOM) based solution known as the port agent architecture has been proposed to solve this problem [Kohlbrenner. 2011]

• A hardware implementation of the SOM is required for the port agent architecture that can process 1Gbps Ethernet traffic for anomalies
SOM Background

- Unsupervised learning algorithm conceived by Teuvo Kohonen and others in the early 1980’s
- Takes higher dimensional feature vectors and produces a matrix of reduced dimensionality (typically 2D) based on the similarities in the features of the input training vectors
  [Kohonen, 2001]
SOM Background [cont]

Figure 1 SOM Concept  - Source Note: [Kohlbrenner, 2011]
SOM Operation

- The SOM has a training phase and then a classification phase
- Training step 1: Assign random value to all of the nodes
- Training step 2: A feature vector from the training data set is compared to every element in the SOM matrix using a distance metric
SOM Operation [cont]

- Training step 3: The element in the matrix with the smallest computed distance to the input feature vector is selected as the best matching unit (BMU).
- Training step 4: Update the BMU and all elements in its neighborhood to more closely match the input feature vector.
  - The learning amount and neighborhood size shrink over time.
SOM Operation [cont]

• Training step 5: If a termination criterion is met (i.e. a sufficient number of vectors have been processed), the training is over, if not, the steps are repeated.
• The SOM can be used to classify data after training by presenting data to the SOM and determining the location of the BMU.
• The SOM can be used to detect anomalies in data by assessing the distance between the data input and the BMU.
Port Agent Architecture

- The port agent architecture uses a system of SOM based port agents to detect anomalous network traffic passing through the ports of an enterprise level network switch.

Figure 2 Port Agent Architecture - Source Note: [Kohlbrenner, 2011]
Port Agent Design

- Each port agent consists of two SOMs optimized for anomaly detection and classification by preloading an already trained map and using a constant learning rate and neighborhood size.

Figure 3 Port Agent Design - Source Note: [Kohlbrenner, 2011]
Thesis Objective

• The design modifications for the port agent SOM result in a smaller and faster SOM
• The objective of this thesis is to provide a design comparison between a conventional SOM and the port agent SOM
• Both designs are fully implemented and the results are used to show that the port agent SOM implementation is smaller than a conventional SOM and can process data faster
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
Step 1: SOM Initialization

- Data is processed and maintained in the SOM matrix as n-dimensional vectors
  - Weight Vector: $W_j = (w_1, w_2, \ldots, w_n)$
  - Input Vector: $V_i = (v_1, v_2, \ldots, v_n)$
- The SOM matrix is initialized by assigning a starting value to each $W_j$ in the matrix
  - The initial value of $W_j$ is often a random number
Step 2: 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 $W_j = (w_1, w_2, \ldots, w_n)$ in the matrix using a distance metric.

- A common metric used in the SOM is the Euclidean norm [Kohonen, 2001]

$$\|V_i - W_j\| = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \cdots + (v_n - w_n)^2}$$
Step 3: 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 in [Kohonen, 2001] as

\[ c = \arg\min_j \| V_i - W_j \| \]
Step 4: Weight Update Calculation

- Based on the BMU selected and the input feature vector $V_i$, each element in the SOM is updated.
- The amount an element is updated is calculated as a function of the discrete time coordinate $t$ as

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

- $t$ is incremented for every input vector processed.
Step 4: Weight Update Calculation [cont]

- $h_{cj}(t)$ is referred to as the neighborhood kernel and is typically computed as a Gaussian function [Kohonen, 2001] or a step function [Pena, 2004]
- Gaussian function

$$h_{cj}(t) = \lambda(t) \times \exp\left(-\frac{||r_j - r_c||^2}{2\sigma^2(t)}\right)$$

- $\lambda(t)$ 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
- $\sigma(t)$ is the neighborhood width and is a monotonically decreasing function of time
Step 4: Weight Update Calculation [cont]

- Step function

\[ h_{cj}(t) = \begin{cases} 
\lambda(t) & \text{if } j \in N_c(t) \\
0 & \text{if } j \notin N_c(t) 
\end{cases} \]

- \( N_c(t) \) 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.
BMU Neighborhood

Figure 4 BMU Neighborhood
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
Overview

- Researched previous hardware implementations in order to understand the design tradeoffs
- Categorized implementations based on their matrix topology, distance metric, BMU selection, and weight update function
Matrix Topology

- **Interconnected nodes vs. Independent nodes**

![Interconnected Topology (left), Independent Topology (right)](image)

- Interconnected topology requires more hardware [Melton, 1992]
Distance Metric

- **Euclidean Norm**
  \[\|V_i - W_j\| = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \ldots + (v_n - w_n)^2}\]

- **Manhattan Distance**
  \[\|V_i - W_j\| = |v_1 - w_1| + |v_2 - w_2| + \ldots + |v_n - w_n|\]

- **Hamming Distance**
  \[\|V_i - W_j\| = \text{Count 1's}(V_i \text{ xor } W_j)\]

- **Chess Book Norm**
  \[\|V_i - W_j\| = \max\{|v_1 - w_1|, |v_2 - w_2|, \ldots, |v_n - w_n|\}\]
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
- The binary tree search and the bit serial search were encountered most often
BMU Selection: Binary Tree Search

- Efficient for small maps, long path delay for large maps

Figure 6 Binary Tree Search – Source Note: [Pena, 2004]
BMU Selection: Bit Serial Search

- Compares the MSB for every element in parallel to determine which is the smallest

Algorithm:
```
repeat {
    if (own MSB == 1 AND line == 0) then
        { flag := 0 }
    shift D one bit left
} until all bits in D
```

Figure 6 Bit Serial Search
Weight Update Function

• Similar to the distance metric, the weight update function is often simplified for hardware implementations to eliminate the required multiplication

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

• The solution encountered most often was the replacement of \( h_{cj}(t) \) with a negative power of two which removes the need for a hardware multiplier
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
Design Approach

• The approach used was to develop a fully featured conventional SOM and then optimize this design for minimal size and maximum throughput for the port agent SOM

• This was accomplished by simplifying or altogether removing elements of the conventional SOM not required by the port agent SOM
SOM Requirements

- **Size:** 256 Nodes
  - 16 nodes x 16 nodes
- **Feature Vectors:** 64 bits
  - 8 x 8-bits
- **Throughput:** 2,976,190 vectors/second
  - Based on 1 Gbps Ethernet traffic

[Kohlbrenner, 2011]
SOM Architecture

• Topology: Fixed architecture of independent nodes that can process each input vector in parallel to maximize throughput
• Distance Metric: Manhattan
• BMU Selection: Bit serial search
• Weight Update: Negative power of two
SOM Architecture [cont]
Conventional SOM Datapath

- The datapath contains all of the processing elements of the SOM, the winner take all (WTA) BMU selection circuit, and the parameter scheduler.

![Figure 10 Datapath](image-url)
Conventional SOM Weight Update Logic

- Weight update function

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

- Implements negative power of two \( h_{cj}(t) \) from [Pena, 2004]

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

- \( D(R_j,R_{j^*}) \) is the Manhattan distance between the BMU \( j^* \) and the node being updated, \( j \)

- \( h_{cj}(t) \) decreases over time with increasing \( \alpha \) and \( \beta \)
Conventional SOM Distance Calculation Logic

- Calculates the Manhattan distance between the input vector and the weight vector

\[ \|v_i - w_j\| = |v_0 - w_0| + |v_1 - w_1| + \ldots + |v_7 - w_7| \]

- The distance is stored in a Parallel In Shift Out (PISO) register to be processed by the WTA BMU selection logic

Figure 13 Distance Calculation Logic
Conventional SOM WTA Circuit

- WTA circuit selects the BMU
- Requires 11 clock cycles to compare all MSBs
- Requires up to 256 clock cycles to randomly break ties
Port Agent SOM Overview

- 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

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

- The port agent SOM is loaded with an already trained map eliminating the need a monotonically decreasing neighborhood function

\[ h_{cj}(t) = \begin{cases} \left(\frac{1}{2}\right)^2 & \text{if } D(R_j, R_j^*) \equiv 0 \\ \left(\frac{1}{2}\right)^6 & \text{if } D(R_j, R_j^*) \equiv 1 \\ 0 & \text{if } D(R_j, R_j^*) > 1 \end{cases} \]

- \( h_{cj}(t) \) for the port agent SOM is derived from the conventional SOM by assuming the maximum values of \( \alpha = 2 \), \( \beta = 3 \), and \( D(R_j, R_{j^*}) = 1 \)
- Removal of \( \alpha \) and \( \beta \) eliminates the need for a parameter scheduler
- Reduces the complexity of the weight update logic
Conventional SOM Weight Update Logic

**Algorithm:**

if \((\text{dist}_\text{to}_\text{bmu} == 0)\) then

\[W' = W + [(V - W) \times \alpha]\]

else

\[W' = W + [(V - W) \times \alpha + \text{dist}_\text{to}_\text{bmu} + \beta]\]

---

**Algorithm:**

if \((\text{dist}_\text{to}_\text{bmu} == 0)\) then

\[W' = W + [(V - W) \times 2]\]

else if \((\text{dist}_\text{to}_\text{bmu} == 1)\) then

\[W' = W + [(V - W) \times 6]\]

else

\[W' = W\]

---

Figure 16 Weight Update Logic for Conventional SOM (left) Port Agent SOM (right)
Port Agent SOM Datapath

- The port agent SOM requires less signaling between the datapath and controller because of the removal of the parameter scheduler.

Figure 17 Datapath/Controller (top) and Datapath (bottom)
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
Overview

- Validation was accomplished by implementing the SOM in VHDL and verifying through simulation using Aldec Active-HDL 8.3 SP1
- SOM matrix was preloaded with a map generated using the software implementation from [Kohlbrenner, 2011]
- Three test vectors were selected to test the three neighborhood scenarios possible in the port agent SOM
- Results were compared to a software implementation of the weight update function

Figure 19 Port Agent SOM Test Case 1 (left), 2 (middle), and 3 (right)
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification

Results and Analysis
Future Work
Summary and Conclusions
Overview

• Both the conventional SOM and port agent SOM were targeted for the Xilinx Virtex-6 xc6v1x760 and Altera Stratix IV GT ep4s100g4f45i1 FPGA parts
• Xilinx ISE version 12.4 was used for the Virtex-6 implementation
• Altera Quartus II – 10.1 was used for the Stratix IV implementation
• The Automated Tool for Hardware Evaluation (ATHENa) version 0.6.1 [Gaj, 2010]
### Virtex-6 Results

#### Table 4 Minimum Area Implementation for the Virtex-6

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

#### Table 5 Maximum Throughput Implementation for the Virtex-6

<table>
<thead>
<tr>
<th>Resource</th>
<th>Conventional SOM</th>
<th>Port Agent SOM</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>127,318</td>
<td>91,832</td>
<td>-35,486 (-28%)</td>
</tr>
<tr>
<td>Slices</td>
<td>45,909</td>
<td>25,565</td>
<td>-20,344 (-44%)</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>23,624</td>
<td>23,575</td>
<td>-49 (-0.2%)</td>
</tr>
<tr>
<td>Max Frequency</td>
<td>51.6 MHz</td>
<td>105.3 MHz</td>
<td>+53.7 MHz (+104%)</td>
</tr>
</tbody>
</table>
# Stratix IV Results

Table 6 Minimum Area Implementation for the Stratix IV

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

Table 7 Maximum Throughput Implementation for the Stratix IV

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

• The port agent SOM is more than two times as fast and less than ¼ the size of the conventional SOM on the Virtex-6
• The port agent SOM is more than 30% faster and less than ¼ the size of the conventional SOM on the Stratix IV
• Small improvements in size to each node is magnified since there are 256 nodes in the design
Analysis [cont]

- The port agent SOM latency for each vector on Virtex-6 is 1,349 ns
  - This equates to 741,549 vectors/second
- The port agent SOM latency for each vector on the Stratix IV is 887 ns
  - This equates to 1,127,464 vectors/second
Analysis [cont]

- Software implementation was evaluated on a Linux Fedora Core 13 Virtual Machine (VM)
- Eight Core Dell T7400 with 6GB of memory
  - The VM was configured to have two CPUs and 2GB of memory
    - Each CPU in the VM was a four core E4520 running at 2.5 GHz
- The software port agent SOM was able to process 71,942 vectors/second
- Port agent SOM hardware implementation is more than 10 times faster than a functionally equivalent software implementation from [Kohlbrenner, 2011]
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
BMU Selection Time

- The port agent SOM as implemented fails to process the 2,976,190 vectors/second required for 1 Gbps Ethernet traffic.
- This is largely because of the 128 clock cycles required by the BMU search circuit to randomly choose a BMU.

Figure 24 BMU Search Circuit
The BMU search circuit can be clocked faster than the rest of the port agent SOM.

Static timing analysis has demonstrated that the BMU search circuit can be clocked at 517 MHz on the Virtex-6.

At this clock rate the average time required to process an input vector would be reduced from 1349 ns to 381 ns on the Virtex-6.

Assuming the Stratix IV supports this rate, the time to process an input vector is reduced from 887 ns to 385 ns or 2,985,074 vectors/second.

This would be sufficient for processing 1 Gbps traffic.
Initial Matrix Weight Configuration

- The port agent SOM as designed is preloaded with hard coded pre-trained initial weights.
- The port agent SOM requires the ability to be preloaded with a user configurable set of values during periodic system maintenance.
- I/O ports and the associated control logic must be added in the future to support this requirement.
A Hardware Implementation of the SOM for a Network Intrusion Detection System

Introduction
SOM Algorithm
SOM Implementation Research
Port Agent SOM Implementation
Port Agent SOM Design Verification
Results and Analysis
Future Work
Summary and Conclusions
Summary and Conclusions

• The research and development of a hardware implementation of the SOM for a network intrusion detection system was described.
• A survey of conventional SOM implementations in hardware resulted in the design of a conventional SOM.
• The conventional SOM was 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 on the Virtex-6 and Stratix IV FPGA devices.
• The port agent SOM is more than two times as fast and less than ¼ the size of the conventional SOM on the Virtex-6.
• The port agent SOM is more than 30% faster and less than ¼ the size of the conventional SOM on the Stratix IV.
• The hardware implementation of the port agent SOM is 10 times as fast as the software implementation.
• By adding a separate clock for the BMU search circuit that port agent SOM will be able to support 1 Gbps Ethernet traffic as desired.
A Hardware Implementation of the SOM for a Network Intrusion Detection System

References


References [cont]


References [cont]


<table>
<thead>
<tr>
<th>References [cont]</th>
</tr>
</thead>
</table>