A Binary Self-Organizing Map and its FPGA Implementation

Kofi Appiah Andrew Hunter Hongying Meng and Shigang Yue
Mervyn Hobden Nigel Priestley Peter Hobden and Cy Pettit

***Abstract*— A binary Self Organizing Map (SOM) has been designed and implemented on a Field Programmable Gate Array (FPGA) chip. A novel learning algorithm which takes binary inputs and maintains tri-state weights is presented. The binary SOM has the capability of recognizing binary input sequences after training. A novel tri-state rule is used in updating the network weights during the training phase. The rule implementation is highly suited to the FPGA architecture, and allows extremely rapid training. This architecture may be used in real-time for fast pattern clustering and classification of binary features.**

I. Introduction

THE original Self Organizing Map (SOM) proposed by Kohonen [1] consists of two layers; the input and the competitive layers. It is an unsupervised neural network with competitive learning models that captures the topology and probability distribution of input data, which facilitates clustering and classification in pattern recognition[2], [3], [4].

The SOM is typically implemented on a standard von Neumann architecture computer. For large input dimensionality and training set size execution speeds are reasonable, but training is rather slow, as the SOM training algorithm typically requires thousands of iterations, each of which involves the calculation of the Euclidean distance of each of the input vectors to each of the neuron prototype vectors. Hardware implementation is therefore of interest. Fortunately, the structure is fairly easy to convert into hardware processing units executing in parallel [5]. However, a direct implementation of the standard SOM onto hardware results in large designs, which consume substantial hardware internal resources (slices, registers and look-up table (LUT) units), limiting the scale of network implementation.

The SOM algorithm presented in [1] is based on a competitive learning algorithm, the winner-take-all (WTA) network, where an input vector is represented by the closest neuron prototype vector, which is assigned during training to a data cluster centre. The prototype vectors are stored in the "weights" of the neural network. The architecture consists of topologically organized array of neurons, each with $N-$dimensional weight vector, where N is also the dimensionality of the input vector. The basic principle of the SOM is to adjust the weight vectors until the neurons represent the input data, while using a topological neighbourhood update rule to ensure that similar prototype occupy nearby positions on the topological map.

Kofi Appiah, Andrew Hunter, Hongying Meng and Shigang Yue are with the Department of Computing and Informatics, University of Lincoln, UK and Mervyn Hobden, Nigel Priestley, Peter Hobden and Cy Pettit are with e2v Technologies, Lincoln, UK.

This work was supported by TSB under BRAINS Project

During training, the "nearest" neuron prototype vector to the input vector is identified – this is called the "winning" neuron – using a distance metric, D. The Euclidean distance is most frequently used as the metric.

For a given network with M neurons and $N-$dimensional input vector $\mathbf{x}$, the distance for neuron with weight vector $\boldsymbol{w_j}$ $(j < M)$ is given by

$$D_j^2 = \sum_{i=0}^{N-1} (\mathbf{x_i} - w_{ji})^2. \quad (1)$$

The vector components of the winning node $\boldsymbol{w_k}$ with minimum distance D_k is then updated as follows

$$\Delta w_{ji} = \eta(\mathbf{x_i} - w_{ji}). \quad (2)$$

where η is the learning rate. The topological ordering property is imposed by also updating weight vectors of nodes in the neighbourhood of the winning node. This can be achieved by the following learning rule

$$\Delta wji = \eta N_j(\mathbf{x_i} - w_{ji}). \quad (3)$$

where N_j is a neighbourhood function (defining the region around $\boldsymbol{w_k}$) based on the topological displacement of neighbouring neuron from the winning neuron. The size of N_j decreases as training progresses.

In the vast majority of implementations, the SOM input data and neurons are represented by real numbers, making it difficult to implement on a hardware architecture like the Field Programmable Gate Array (FPGA). However, in many applications the data is either presented as a binary string, or may be conveniently recoded as such (a "binary signature"). For example, in image processing applications a bank of Haar filters produces a long binary signature. In this paper we present a new learning algorithm which takes binary inputs and maintains tri-state weights (neuron) in the SOM. We also present the FPGA implementation of this binary Self Organizing Map (bSOM). The bSOM is designed for efficient hardware implementation, having both greatly reduced circuit size compared to a real-valued SOM, and exceptionally fast execution and training times.

In section II, we review previous implementations of SOM on hardware architectures. The novel bSOM algorithm is then presented in III, followed by its FPGA implementation in section IV. Section V, presents the experimental results in software and hardware, and we conclude in section VI.

II. Hardware architectures for Kohonen's map

Software simulations are very useful for investigating the capabilities of neural network models [6], and are suitable

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

for many applications, but are limited in the size of network implementation, particularly where very fast execution and training is required. Hardware neural networks can be implemented using analogue or digital systems [12].

The popularity of digital implementations stems from the fact they are more accurate, more flexible and are less sensitive to noise than analogue ones [7] – notwithstanding the analogue inspiration from theoretical neural models. The computational complexity of the SOM algorithm [1] prevents it from training in real-time on single processor architectures, for many real-time applications. The FPGA provides a suitable platform for the implementation of a digital version of the SOM neural network, due to its reconfigurability and smaller non-recurring engineering (NRE) costs.

However, a floating-point representation of neurons in a neural network presents significant difficulties for implementation on FPGAs, despite the current advances in FPGA technology [13], since floating point multipliers and the computation of nonlinear excitation functions is complex and consumes large resources [7] [8]. A number of authors have sought to mitigate this problem by introducing simplifications to the SOM algorithm; Pena *et. al.* [4] implemented a digital version of the SOM on FPGA by replacing the Euclidean distance computations with a Cityblock (Manhattan distance) computation to avoid the expense of hardware multiplication. In addition, they simplified the neighbourhood function and introduced a set of new learning parameters.

A similar implementation of the SOM, where the distance, neighbourhood and learning rate computation is replaced with a simplified version, has been presented by Chang *et. al.* [9] and Porrmann *et. al.* [10]. An efficient SOM architecture based on a new Frequency Adaptive Learning (FAL) algorithm, which efficiently replaces the neighbourhood adaptation function of the conventional SOM, has been presented in [9]. The design was implemented on a Xilinx FPGA and is capable of quantizing a 512×512 pixel colour image in about 1.003sec at 35MHz clock rate without the use of sub-sampling.

A design based on the universal rapid prototyping system RAPTOR2000 for the acceleration of SOM is presented in [10]. Using Xilinx FPGAs, the implementation achieves a speed-up of up to 190 (with five FPGA modules on the RAPTOR2000 system) compared to a software implementation on a state of the art personal computer, for typical applications of self-organizing maps. A similar system, implemented on a Xilinx Virtex II XC2V300, aimed at reducing the training processing time of SOM, has been presented in [11]. The design consists of 16 units in the input layer, N neurons in the output layer and is divided into three sections: the processing unit array, the address generator and the controller. Compared with an all software implementation, the design achieves approximately 89% speed-up.

Other forms of neural networks have also been designed and implemented on hardware architectures such as FPGA. In [17], Nedjah *et. al.* proposed the design of a feed-forward neural network on FPGA using a stochastic process to implement the computation performed by the neurons. In the implementation, the multiplication and addition of stochastic values are achieved by an ensemble of XNOR and AND gates respectively. In the proposed stochastic model, a long probabilistic bit-stream whose density of set bits is proportional to the encoded numeric value is used to represent a number.

Merchant *et. al.* [13] designed an intrinsic embedded online evolution system using Block-based neural networks(BbNN)[15]; a grid based network structure of interconnected block-based neurons. Each neuron block can have up to 3 inputs, 3 outputs and 9 synaptic weights and biases depending on the internal configuration determined by the network structure. The design has been implemented on a Xilinx Virtex II Pro FPGA running at 40MHz, using a LUT based BbNN implemented on the block RAM.

A modified version of Boolean k-nearest neighbour (BKNN), a supervised classifier using Boolean Neural Networks, with binary inputs and outputs, has been implemented on FPGA by Liu *et. al.* [14]. The modification omits the iterative classification procedure and is characterised by a one-shot training and a single classification sweep to obtain the answer. The design has been verified with Xilinx ISE 6, targeting XC2S100E Xilinx Spartan2E FPGA.

To entirely avoid numeric weights in the SOM, while maintaining the level of performance as well as speed up in training and using SOM for real-time application, Yamakawa *et. al.* in [3] proposed a binary weighted vector SOM and simulated it in hardware. The proposed SOM uses binary data for both input and weight vectors. The Hamming distance is used to calculate the distance between the input and weight vectors, to identify the winning neuron in the network. However, the weight vector is updated with priority given to the most significant bit, thus attempting to treat the weights as a direct representation of integer values.

The use of the binary weighted SOM on FPGA proves to be very successful compared to the others. The design of the binary weighted SOM is five times faster than the real number weighted SOM in software and 140 times faster in hardware[3]. This highlights a key principle – that the most successful design will take account of the nature of the hardware architecture. A novel binary SOM that follows this principle is presented in the following section.

III. Proposed binary SOM algorithm

In this section we introduce the binary Self Organizing Map (bSOM). This takes a binary vector input, and maintains tri-state vector weights with {0, 1, # } as the possible values. The # represents a "don't care" state, which signifies that the corresponding input vector bit may be either set or clear. The weight vectors have the same length as the input binary vector. The bSOM has the same essential structure as a standard SOM, with an input layer and a competitive layer – see figure 1. Given a binary input vector $\mathbf{b_i} = (b_1, b_2, \ldots, b_n)$, all the units in the competitive layer are "connected" by corresponding prototype vectors, $\mathbf{w_j} = (w_{j1}, w_{j2}, \ldots, w_{jn})$.

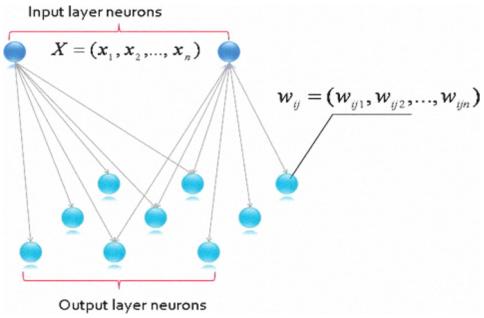

Fig. 1. Structure of the Original SOM[18].

The bSOM training algorithm is discussed below, and compared and contrasted with the original SOM algorithm and Yamakawa's [3] implementation.

A. Distance Computation

The Euclidean distance computation, equation 1, is used in the original SOM to calculate the distance between the input vector and the neuron prototype vectors. The implementation of this equation is not only difficult to realise in hardware, but also unnecessary for binary vectors. Following [3], we use the Hamming distance H, as shown in equation 4, for an input vector $\boldsymbol{x}$ and weight vector $\boldsymbol{w_j}$.

$$H(x, w_j) = bits\{(\dot{x}_i \wedge w_{ji}) \vee (x_i \wedge \dot{w}_{ji}) \mid i = 1, \ldots, n\} \quad (4)$$

where $\dot{x}_i$ and $\dot{w}_{ji}$ are the bit inverse of x_i and w_{ji} respectively.

B. Winner Take All (WTA)

Analogously to the original SOM, the unit with the smallest Hamming distance to the input is defined as the winning neuron; see equation 5. Since the weight is a tri-state vector, a # is considered as a matching bit irrespective of the input bit's value. The total number of #'s in the weight vector is stored and used when selecting the winning unit in the competitive layer. When there is a tie or when two neurons have the same Hamming distance to the input vector, the neuron with the minimum number of #'s is chosen as the better match.

$$w_c = arg\ min_j(H(x, w_j)) \quad (5)$$

C. Neighbourhood Selection

As in the original SOM and in [3], a neighbourhood N_c of neurons around the winning neuron w_c is selected and updated with the winning neuron. The size of the neighbourhood is inversely proportional to the iteration value.

D. Updating Weight Vectors

The winning neuron and its neighbourhood are updated as shown in equation 3. In bSOM, a probabilistic update is used. The probabilistic update in the bSOM is summarised as follows:

- A bit in the weight vector is only updated if it is different from its corresponding input vector bit.
- An update value is generated for each iteration during training. This value decreases as training progresses.
- A random number is then generated and if the number is greater than the update value, the bit is updated.
- A bit is updated by changing its value from 1 to #, 0 to # or # to (0 or 1) depending on the input bit value.

	0	1	#
0	0.5	0	0.5
1	0	0.5	0.5
#	0.5	0.5	0

Fig. 2. The conditional Markov transition matrix

The bit transition can be modelled as a Markov chain with a conditional Markov transition matrix (T) as shown in figure 2. If the probability of applying the conditional Markov transition matrix is given as $p = 1 - update\ rate$. The resulting effective Markov transition matrix (T_e) for a bit to change is as shown in figure 3. If T is a regular transition matrix, then as n approaches infinity, $T^n \rightarrow S$, where S is a matrix with constant vectors, as shown in figure 4. The transition matrix settles after the 12th iteration. Solving for

	0	1	#
0	1-0.5p	0	0.5p
1	0	1-0.5p	0.5p
#	0.5p	0.5p	1-p

Fig. 3. The effective Markov transition matrix

X in $(T \rightarrow \lambda I)X = 0$ where $\lambda = 1$ and X is a vector representing the three states (0, 1, #); $X_1 = X_2 = X_3$. This shows that increasing the number of training iterations makes no significant difference to the final results, confirming that the bSOM requires fewer iterations to converge, as compared to the original SOM and that presented in [3]. The following section gives the architectural and implementation features of the proposed bSOM algorithm.

T^2

0.5000	0.2500	0.2500
0.2500	0.5000	0.2500
0.2500	0.2500	0.5000

T^{12}

0.3335	0.3333	0.3333
0.3333	0.3335	0.3333
0.3333	0.3333	0.3335

T^{13}

0.3334	0.3333	0.3334
0.3333	0.3334	0.3334
0.3334	0.3334	0.3333

T^{14}

0.3334	0.3333	0.3333
0.3333	0.3334	0.3333
0.3333	0.3333	0.3334

Fig. 4. The conditional Markov transition matrix after the 2nd, 12th, 13th and 14th iterations respectively.

TABLE I

SPECIFICATION OF FPGA CIRCUIT DESIGN.

Network Size	40 neurons
Input vectors	784 bits
Neuron vectors	784 bits
Initial weights	Random
Maximum neighbourhood	4 neurons

IV. FPGA ARCHITECTURE AND IMPLEMENTATION

The most critical aspect of any hardware design is the selection and design of the architecture that provides the most efficient and effective implementation [9]. The specifications of the circuit implemented on FPGA is given in table I with its corresponding block diagram in figure 5. The circuitry is made up of five basic blocks, namely the weight initialization, pattern input, Winner Take All, neighbourhood update and the display blocks.

Three of the five blocks run in parallel. These are the pattern input, Winner Take All and the display (output) block. The weight initialization block is triggered only at start-up. Similarly, the neighbourhood update block is triggered when a winning node w_c is identified for an input binary vector. Details of the five basic blocks are presented in the following sections.

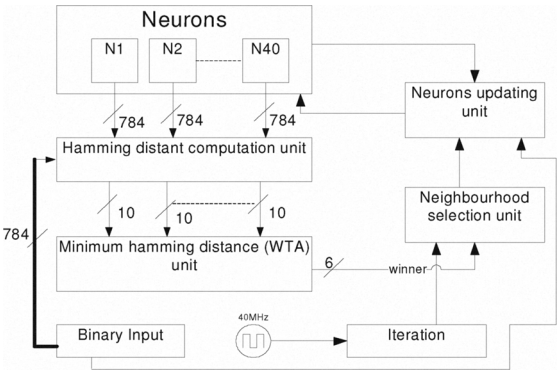

Fig. 5. A block diagram of the design circuit.

A. Weight Initialization block

This block is used to randomly initialize all the weight (neuron) vectors in the network. All the neurons in the network are initialized in parallel bit-by-bit; hence it takes as many clock cycles as there are bits in the binary input vector to complete the initialization. The hardware architecture presented here has been test with binary image characters of size 28×28, totalling 784 bits. The sizes of the input and weight vectors are all set to 784 bits and can easily be altered for any image size. The presented implementation takes exactly 784 clock cycles to completely initialize all the neurons.

B. Pattern Input block

This block is used to acquire the binary input vector (or binary image) from an external camera. The size of the input vector 784 is pre-programmed and the input is complete when a total of 784 bits is read from the camera. This binary data is stored in the input vector and then passed onto the WTA block for further processing.

C. Winner Take All block

This block is made up two parts, the Hamming distance computation unit and the winning neuron unit.

1) Distance computation unit: This unit is used to compute the Hamming distance between the input binary vector and all the (40) neurons in the bSOM. The Hamming distance between the input vector x_i and a neuron w_j, as shown in equation 6 is a bitwise operation and hence, takes as many clock cycles as there are bits in the input vector. Since the Hamming distance for all the 40 neurons are computed in parallel, it takes exactly 784 clock cycles to compute the Hamming distance for all the neurons in the network.

$$H_{ij} = \sum_{k=1}^{784} H_{ijk}, \text{ where } w_{jk} \neq \#. \tag{6}$$

where k is the total number of bits in the input vector and $j \in (1 \cdots 40)$ is the address of the neuron. It is worth noting that the neuron vector is tri-state and the # state is ignored when computing the Hamming distance. Thus, for a neuron with 784 #'s, the Hamming distance will always be 0.

2) Winning neuron unit: This unit uses the results from the Hamming distance computed in section IV-C.1 to identify the winning neuron. The design, as shown in figure 6, uses a series of comparators to select the minimum of every two input Hamming distances. For an implementation with 40 values, the design takes exactly seven clock cycles to compute the node with the minimum Hamming distance.

D. Neighbourhood update block

This block is use to select the neighbourhood of the winning neuron and to update the neurons in the specified region. The size of the neighbourhood reduces as training progresses. In the hardware implementation the maximum size of the neighbourhood is set to 4, and decreases as training progresses. The iterations count determines the size of the neighbourhood; for example, if the total number of iterations is set to 100, then for the first 25 iterations the neighbourhood is set to 4, then 3 in the second 25 iterations (thus iteration 26 to 50) and then 1 in the last 25 iterations.

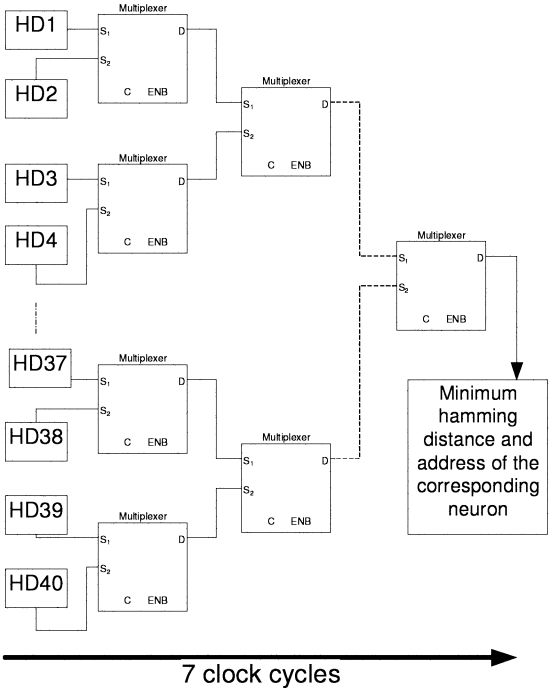

Fig. 6. Structure of the WTA unit.

The update requires a random number generator, which is not only complex to implement in hardware but also computationally expensive. To avoid these costs, an LUT with 2000 randomly generated numbers has been implemented on the FPGA. For a mismatched bit between the input vector and the neuron to be updated, one of the 2000 values is selected using the iteration count. If the number of iterations exceeds 2000, the last 10 bits of the iteration count is used to address the random number in the LUT.

Mis-matching bits in the neuron vector are updated as discussed in section III-D; thus a 1 changes to #, a 0 changes to # and a # changes into 0 or 1 depending on the binary input value. Note, a # is implemented as '10' or decimal 2.

E. Output display blocks

This block displays the neurons (weights) as binary image on an external Video Graphics Array (VGA) for visual verification. It runs in parallel with the input and WTA blocks. It runs at the refresh rate for the VGA used, typically 60Hz.

The bSOM architecture discussed here has been implemented on a Xilinx Virtex-4 FPGA chip (XC4VLX160) with approximately 152,064 logic cells with embedded RAM totalling 5,184 Kbits. The design and verification was accomplished using the Handel-C high level descriptive language. Compilation and simulation were achieved using the Agility DK design suite. Synthesis – the translation of abstract high-level code into a gate-level net-list – was accomplished using Xilinx ISE tools.

The entire design can be clocked up to 40MHz, making it possible to train the binary Self Organizing Map with up to 25,000 patterns of size 784bit in a second after initialization. The clock frequency of 40MHz also includes the design for controlling the external logic for the VGA and the camera. This is the actual hardware test and the most stable clock frequency. The frequency could be much higher without the requirement to interface these devices. Table II gives the details of the resource utilization of the FPGA implementation.

TABLE II
IMPLEMENTATION RESULTS FOR THE BSOM, USING VIRTEX-4 *XC4VLX160*, PACKAGE *FF1148* AND SPEED GRADE *-10*.

Resource		Total Used	
Name	**Total**	**Used**	**Per.(%)**
Flip Flops	135,168	4,095	3
4 input LUTs	135,168	18,387	13
bonded IOBs	768	147	19
Occupied Slices	67,584	11,468	16
RAM16s	288	43	14

V. EXPERIMENTAL RESULTS

This section describes some of the experiments conducted on the algorithm to verify its correctness, and compares it with other implementations. The MNIST database of handwritten digits[19], sample shown in figure 7 is used to test the implementation on both PC simulation and on the FPGA hardware architecture. A comparison on the PC between the original SOM as presented by Kohonen in [1] (herein referred to as the conventional SOM), a strictly binary SOM and the proposed tri-state SOM (bSOM) algorithms is also given in this section.

Even though the bSOM is meant for hardware implementation; for simulation and a fair comparison with the conventional SOM, we have also implemented the bSOM on a PC using MATLAB. To justify the use of tri-state $(0, 1, \#)$ rather than just binary $(0, 1)$, we have also implemented a version of the binary SOM with only 0's and 1's excluding the third state #. The solely binary implementation uses the same rules as in the tri-state (or bSOM) implementation and it is herein referred to as the strict binary SOM.

A. Software Simulation

The software based simulation of the bSOM has been achieved on a PC with a general purpose processor clocked at 2.8GHz and 2GB of SDRAM. Initial experiments were conducted to empirically select control parameters – number of neuron, neighbourhood size and learning rate – for all three implementations of the SOM (the conventional SOM, strict binary SOM and tri-state SOM (bSOM)).

To determine the number of neurons required to represent all 60,000 patterns in the dataset; see figure 7. We experimented with different numbers of neurons from 10 to 100 in steps of 10. This experiment was primarily based on the bSOM and also applies for the conventional SOM algorithms. The results improve with increasing numbers of neurons until performance begin to plateau at 80 neurons for the bSOM (with minimal improvement thereafter).

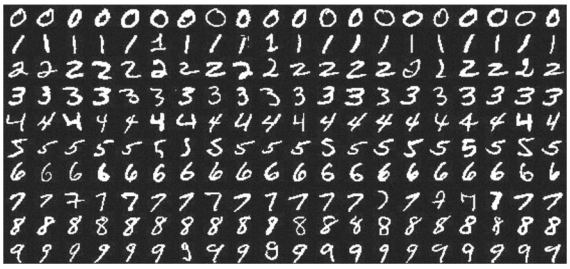

Fig. 7. A subset of the MNIST database of handwritten digits.

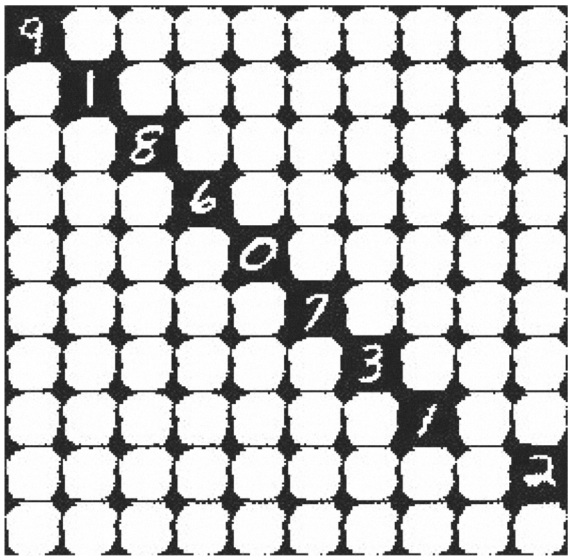

Fig. 8. Results of the cSOM with neighbourhood size of 10.

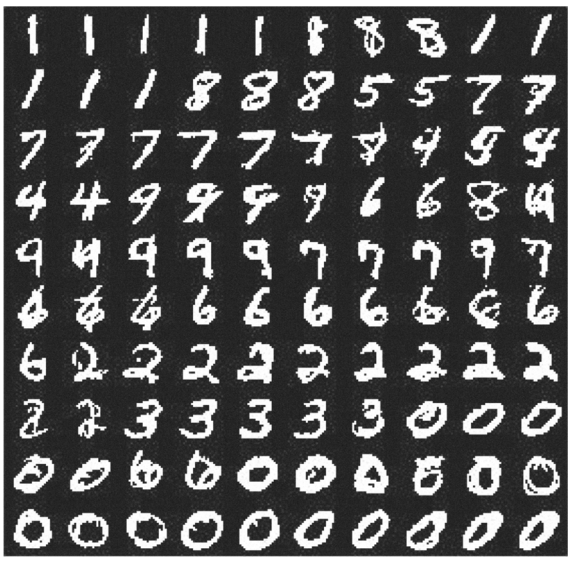

Fig. 9. Results of the bSOM with neighbourhood size of 10.

The neighbourhood size was determined primarily using the conventional SOM. Increasing the neighbourhood size increases the number of unused neurons in the conventional SOM. Even though the bSOM did not suffer from the same effects, for a fair comparison, the optimal neighbourhood size for the conventional SOM was adopted for all the other implementations. Figures 8 and 9 show the neurons after training the conventional SOM (CSOM) and tri-state SOM (bSOM) respectively, with the neighbourhood size set to 10 .

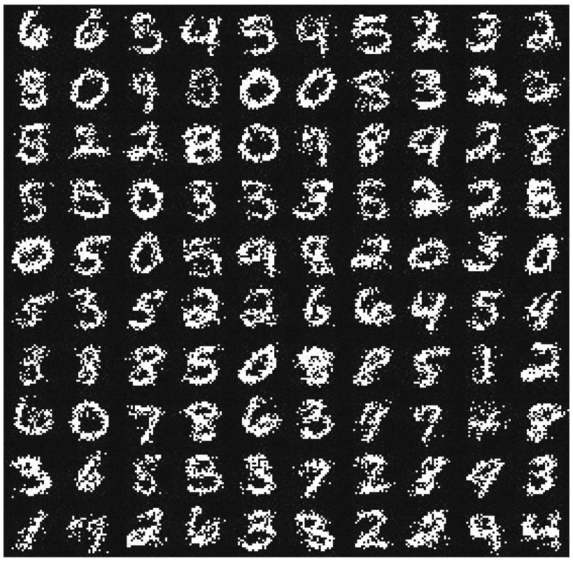

Fig. 10. Results with the [19] dataset; 100 neurons and 100 iterations for strict binary SOM implementation.

After the selection of appropriate parameters, various tests were conducted to compare the performance of the tri-state SOM (bSOM), with the conventional SOM (CSOM) and a strict binary SOM algorithms. The main aim of this experiment it to compare the convergence of the tri-state SOM with the others. Tests for 10, 100, 200 and 500 iterations on the MNIST dataset[19] were conducted. The experiment was repeated for a number of times. In the case of the conventional SOM and the strict binary SOM, repetitions did not make any difference, whereas the results of the tri-state SOM showed some variability.

Six runs have been conducted on the 10, 100 and 200 iterations, with three runs for the 500 iterations. Figures 13 and 14 show the best and average results for the various runs for the three implementations. From the two graphs, the tri-state SOM plateaus after 100 iterations. Increasing the number of iterations does not make much difference for the tri-state SOM. In contrast to the tri-state SOM, the conventional SOM improves as the number of iteration increases and the tri-state version outperforms the conversional SOM for smaller iterative values. The performance level of the strict binary version is inferior as compared to either the conventional or the tri-state SOMs.

Results of tests conducted using the MNIST dataset [19], with 100 neurons in each network and the training repeated

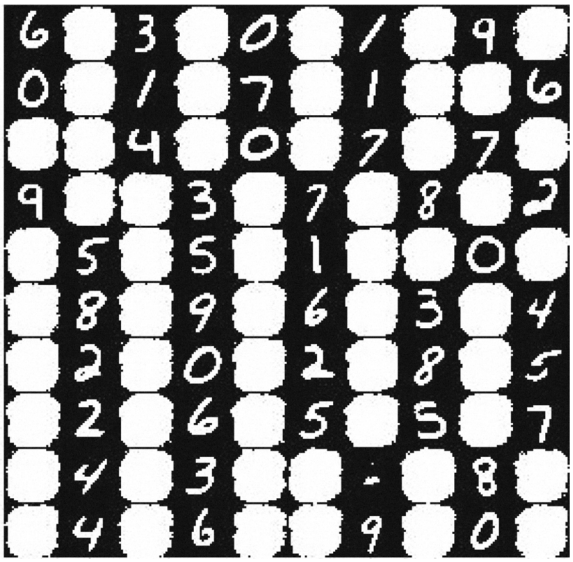

Fig. 11. Results with the [19] dataset; 100 neurons and 100 iterations for the original SOM implementation.

for 100 iterations are shown in figures 10, 11 and 12 for the strict binary, conventional and tri-state SOMs respectively. From the figures, the tri-state evenly represents the dataset, while the results of the original SOM is visually the best, and the strict binary version is the worse. The 60,000 binary image patterns from the MNIST dataset [19] are converted into binary strings each with 784 bits longs. The binary signatures are then used to train binary networks each with 100 nodes; hence figures 10, 11 and 12 are the binary representations of the 60,000 binary patterns using the the strict binary, conventional and tri-state SOMs respectively.

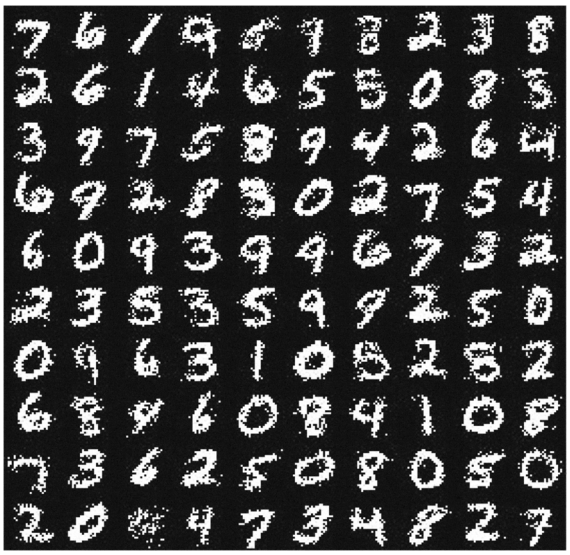

Fig. 12. Results with the [19] dataset; 100 neurons and 100 iterations for the tri-state SOM implementation.

Tables III and IV, show the performance level in terms of the number of binary images correctly classified by the three implementations. Corresponding graphs for the two tables are shown in figures 13 and 14, where TSOM is the tri-state SOM, CSOM is the convensional SOM and BSOM is the strict binary SOM.

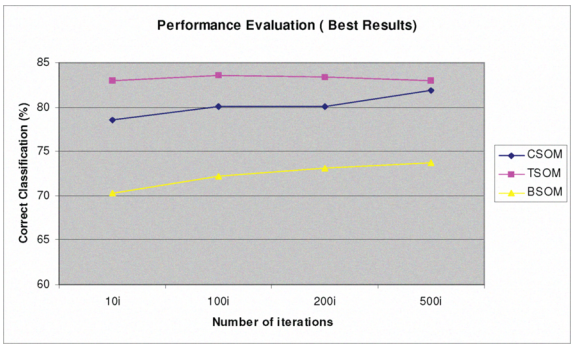

Fig. 13. A graph showing the best performance for the 3 implementations.

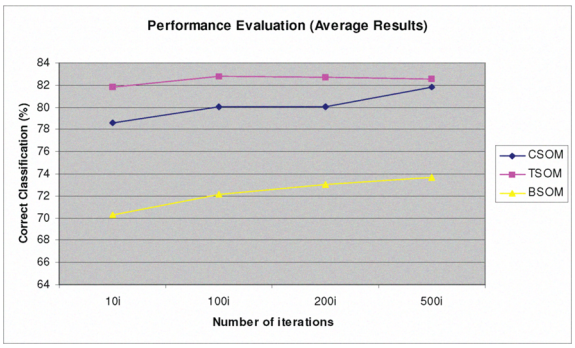

Fig. 14. A graph showing an average performance for the 3 implementations.

The software implementations were written in MATLAB. The original SOM (conventional SOM) runs slightly faster on the PC with the MATLAB implementation than the tri-state SOM. It is worth noting that MATLAB is not the most optimal for a tri-state implementation on the PC. However, the hardware implementation speed of the bSOM far outperforms the conventional SOM algorithm.

TABLE III
A TABLE SHOWING THE BEST PERFORMANCE OF THE 3 IMPLEMENTATIONS FOR 4 DIFFERENT ITERATIONS.

Design	10i	100i	200i	500i
CSOM	78.58%	80.02%	80.03%	81.84%
TSOM	82.95%	83.54%	83.39%	83.02%
BSOM	70.28%	72.18%	73.06%	73.68%

B. Hardware Simulation

The MNIST dataset [19] has been used to test the implementation on FPGA. The dataset is available as binary strings. We first convert the binary strings into binary images. The binary dataset is then converted into a video stream and fed to the hardware implementation. Results of the hardware

TABLE IV

A TABLE SHOWING THE AVERAGE PERFORMANCE FOR THE 3 IMPLEMENTATIONS FOR 4 DIFFERENT ITERATIONS.

Design	10i	100i	200i	500i
CSOM	78.58%	80.02%	80.03%	81.84%
TSOM	81.86%	82.83%	82.69%	82.57%
BSOM	70.28%	72.18%	73.06%	73.68%

are displayed on a VGA for visual inspection as well as written onto memory for further verification. Networks of size 10 to 60 neurons have been successfully implemented on the FPGA platform running at 40MHz. Networks with up to 100 neurons have been simulated on the hardware emulator and will be transferred onto the physical device in due course.

VI. CONCLUSION

In this paper we have proposed a novel binary SOM algorithm suitable for FPGA implementation, and designed the digital hardware based on the algorithm. The features of the algorithm include a binary input and neuron vectors. At the reported processing speed of 40MHz, the hardware tri-state SOM is 30 times faster than the original SOM implemented on a state-of-the-art PC. Generally, SOMs with neighbourhood update have topology-preserving nature. Unfortunately, this has not been the case with MNIST dataset [19]; a behaviour subject to further analysis.

REFERENCES

[1] T. Kohonene, *Self-Organizing Maps.* Springer,New York, 1995.

[2] H. C. Card, G.K. Rosendahl, D. K. McNeill and R. D. McLeod *Competitive Learning Algorithms and Neurocomputer Architecture* IEEE Transaction on Computers, Vol. 47 No. 8 August 1998

[3] T.Yamakawa, K. Horio and T. Hiratsuka *Advanced Self-Organizing Maps using Binary weight vector and its digital hardware design* Proc. of the 9th Int. Conf. on Neural Information, 2002.

[4] J. Pena and M. Vanegas *Digital Hardware Architecture of Kohonen's Self Organizing Feature Maps with Exponential Neighboring Function* IEEE International Conference on Reconfigurable Computing and FPGA, 2006.

[5] W. Kurdthongmee *A novel hardware-oriented Kohonen SOM image compression algorithm and its FPGA implementation* Journal of Systems Architecture, Volume 54, Issue 10, October 2008, Pages 983-994, ISSN 1383-7621.

[6] P. Moerland and E. Fiesler *Hardware-Friendly Learning Algorithms for Neural Networks: an Overview* Proc. of 5th international conf. on microelectronics for neural networks and fuzzy systems, 1996.

[7] A. Muthuramalingam, S. Himavathi, E. Srinivasan *Neural Network Implementation Using FPGA:Issues and Application* International Journal of Information Technology Volume 4 Number 2, Winter 2008.

[8] R. Molz, P. Engel, F. Moraes, L. Torres and M. Robert *A fast prototyping neural network model for image classification* Proc. of Design of Circuits and Integrated Systems, 2000.

[9] C. Chang, M. Shibu and R. Xiao *Self Organizing Feature Map for Color Quantization on FPGA* FPGA implementations of neural networks - Springer, 2006.

[10] M. Porrmann, U. Witkowski and U. Ruckert *Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware* FPGA implementations of neural networks - Springer, 2006

[11] R. AGUNDIS, G. GIRONES, C. PALERO and D. CARMONA *A Mixed Hardware/Software SOFM Training System* Computacin y Sistemas, No. 004, april 2008.

[12] J. Starzyk, Z. Zhu and T. Liu *Self-Organizing Learning Array* IEEE Trans. on Neural Networks, vol. 16, no. 2, pp. 355-363, March 2005.

[13] S. Merchant, G. Peterson and S. Kong *Intrinsic Embedded Hardware Evolution of Block-based Neural Networks*, Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms, June 26-29, 2006,

[14] J. Liu, B. Li and D. Liang *Design and Implementation of FPGA-Based Modified BKNN Classifier* IJCSNS International Journal of Computer Science and Network Security, VOL.7 No. 3, March 2007.

[15] S. Moon and S. Kong *Block-based neural networks* Neural Networks, IEEE Transactions on Volume 12, Issue 2, March 2001.

[16] P.De Souto,T. Ludermir and M. Campos *Encoding of Probabilistic Automata into RAM-Based Neural Networks* In Proceedings of the IEEE-INNS-ENNS international Joint Conference on Neural Networks, 2000.

[17] N. Nedjah and L. de Macedo *Stochastic Reconfigurable Hardware for Neural Networks* In Proc. of IEEE Euromicro Symposium on Digital System Design, 2003.

[18] B. Magomedov *Self-Organizing Feature Maps (Kohonen maps)* http://www.codeproject.com/KB/recipes/sofm.aspx

[19] Y. LeCun and C. Cortes *The MNIST database of handwritten digits.* http://yann.lecun.com/exdb/mnist/

[20] J. Geusebroek, G. Burghouts, A. Smeulders *The Amsterdam Library of Object Images* International Journal of Computer Vision, 2005.