Implementation and Applications of Tri-State Self Organising Maps on FPGA

Kofi Appiah, *Member, IEEE,* Andrew Hunter, *Member, IEEE,* Patrick Dickinson, *Member, IEEE,* and Hongying Meng, *Member, IEEE*

***Abstract*—This paper introduces a tri-state logic Self Organising Map (bSOM) designed and implemented on a Field Programmable Gate Array (FPGA) chip. The bSOM takes binary inputs and maintains tri-state weights. A novel training rule is presented. The bSOM is well-suited to FPGA implementation, trains quicker than the original SOM, and can be used in clustering and classification problems with binary input data. Two practical applications, character recognition and appearance-based object identification, are used to illustrate the performance of the implementation. The appearance-based object identification forms part of an end-to-end surveillance system implemented wholly on FPGA. In both applications, binary signatures extracted from the objects are processed by the bSOM. The system performance is compared with a traditional SOM with real-valued weights and a strictly binary weighted SOM.**

***Index Terms*—binary SOM, FPGA, object recognition, character recognition.**

I. Introduction

ONE of the original motivations for research into neural networks is the observation that neural systems are massively parallel and can therefore potentially escape some of the inherent computational limitations of strictly serial architectures. However, most neural network research uses simulations on standard CPU architectures, and so does not address the architectural issues found in real parallel hardware. This paper introduces an architecture for self-organizing maps custom-designed for Field Programmable Gate Array (FPGA) implementation, that is designed to exploit the fine-grained parallelism of the FPGA while respecting its architectural limitations. The FPGA platform is chosen as it is reconfigurable, allowing easy custom-design of each implementation, and on-chip integration with other system functions.

The original Self Organising Map (SOM) proposed by Kohonen [1], [2] consists of two layers: the input and the competitive layers. It is an unsupervised neural network with competitive learning that captures the topology and probability distribution of input data, and can be used for a wide range of pattern recognition purposes including anomaly detection, clustering and classification [3], [4], [5]. In the vast majority of implementations the SOM input data and neurons are represented by real numbers (with floating-point representation), making it difficult to implement efficiently on FPGA, which in general do not have specialized floating point hardware, and therefore provide only inefficient implementations of real-numbers.

Weightless Neural Networks (WNNs) [6] are an alternative neural network architecture that directly exploit hardware capabilities (commercially available Random Access Memory) and use binary inputs and outputs. Instead of adjusting weights, learning is implemented by changing look-up table entries, providing very rapid training [7]. In WNNs, memory blocks play the role of the 'neurons' in the system. This approach to neural networks was pioneered by Aleksander [8], [9], and has since been further developed by Austin [10], and others [11]. An N input RAM node (RAM-based neuron) has 2^N memory locations addressed by an N-bit binary string. An N-bit binary input string will access only one memory location. Learning in RAM node is accomplished by writing the desired output into the corresponding look-up table. RAM networks are taught to respond with a "1" output for those patterns in the training set and only for those patterns. Generalization is achieved by sub-sampling the input space with multiple RAM-nodes (with cross-sampling of inputs), and aggregating the RAM-node outputs. A limitation of RAM-nodes is that a "0" output may be ambiguous, indicating either lack of a corresponding training example or existence of a counter-example [7].

To overcome this ambiguity Aleksander and Myers [12] developed the Probabilistic Logic Node (PLN) system. The PLN node uses a tri-state scheme with three levels (0, 0.5, 1) in which the value of 0.5 means that an output of 0 or 1 can be expected with equal probability if that node is addressed. The three levels in PLN are represented using two bits. PLN are initialized to 0.5 values; in training these are replaced with 0's or 1's. A further development is the Probabilistic RAM (pRAM) model [13] which uses fixed-point probability estimates as weights, which approximate the range [0,1]. Similar to other RAM-based networks, an N input pRAM node has 2^N memory locations addressed by the input vector. A number of nodes may be combined by aggregating the probabilities. The probabilistic training is based on frequency of class examples. In its basic form, the pRAM comprises of a number of memory locations, a comparator and a noise generator [14].

In SOM networks the neurons in the competitive layer each have a "weight vector" which represent a position in the input space, and therefore act as "prototype" vectors. During training and execution the "winning" neuron is identified as

K. Appiah, A. Hunter and P. Dickinson are with the Lincoln School of Computer Science, University of Lincoln, Lincoln, LN6 7TS, UK e-mail: (see http://www.lincoln.ac.uk/socs/).

H. Meng is with Brunel University, London, UK

that with the minimum distance from its prototype vector to the input vector using an appropriate distance metric, D. During execution, the winner-takes-all (WTA) algorithm is used and the winning neuron stands for the input [1], [2]. During training, the winning neuron and its topological neighbors' prototype vectors are adjusted towards the input vector, so that the prototypes come to represent cluster centres. The Euclidean distance is most frequently used as the distance metric.

Although the SOM uses real data inputs and outputs, in some 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 Haar filter responses are often used to produce a (long) binary signature. In this case the real-number representation of prototypes is inefficient, and arguably inappropriate. Most proposed SOM hardware implementations have adopted a real valued architectural model, modified to utilize the parallel nature of FPGAs [15]. Manolakos and Logaras [15] presented a parallel SOM architecture design following the systolic model, which is realized as a flexible soft IP core. Soft IP based FPGA processor cores generally have lower performance levels and higher resource utilization [16] than hard IP cores implementing the same functions, but they are highly flexible and can be customized for a specific application with relative ease [17]. In contrast, hard processor IP cores are generally highly-optimized and fine tuned, but difficult to port to other targets with equivalent performance [16] and over-specified for restricted tasks.

This paper presents a tri-state Self Organising Map (the bSOM), which takes a binary input vector and maintains tri-state weights. The design is implemented as a soft IP core in Handel C, where the number of neurons, the number of elements per input vector elements and the number of bits for data and weights are all tuneable parameters, maximizing flexibility and minimizing complexity. The architecture is well-suited to FPGAs, achieving very high training and execution speeds, and is easily integrated into a wider on-chip system. The architecture may be used for various pattern recognition tasks, including clustering and classification. We demonstrate its use in two applications: hand-written character recognition and moving object identification. In the latter application the bSOM is part of a larger on-chip system that includes feature extraction from colour video sequences to produce binary signatures.

Preliminary versions of this material have been presented in conference papers [18] [19]; this paper extends and integrates the presentation.

The remainder of this paper is divided into five sections. Section II gives an overview of hardware solutions to the implementation of SOM. This is followed by the details and training rules of the proposed bSOM in section III. Section IV describes the FPGA realisation of the proposed bSOM and section V presents the two practical applications of the bSOM with experimental results. We conclude in section VI with suggested future work.

II. Hardware Architectures for SOM

Hardware implementations of neural networks are essential to take full advantage of the inherent parallelism of neural network [20]. Software simulations are useful for investigating the capabilities of neural network models, and creating new algorithms, but they fall short where fast execution and training is required [21], and fall short as problem size scales up, creating a bottleneck [22]. There are two major approaches to implementing neural networks in hardware: analogue and digital implementations. Digital neural networks are more popular due to their greater accuracy, flexibility and relatively insensitivity to noise [23].

FPGAs provide an appealing platform for the implementation of digital neural networks, due to their reconfigurability and consequently small non-recurring engineering (NRE) cost. Neural architectures invariably need to be "tuned" for specific applications (e.g. number of inputs); this is difficult to accommodate in a specialised neural ASIC chip, but easily handled on an FPGA. Moreover, neural networks are rarely used alone, and can be integrated directly on the chip with other system functions (e.g. video or image input, feature extraction, control functions).

However, a key limitation of FPGAs is the cost of implementing arithmetic – particularly floating point operations – and most traditional neural networks are designed around real-valued arithmetic. This suggests that either efficient representations of real-values must be used, or that the problem should be recast to use a binary representation.

A popular approach is to use fixed point arithmetic to approximate real values. Pena and Vanegas [5] implemented a fixed-point version of the SOM on FPGA. They simplified the neighbourhood function and introduced a set of new learning. Raygoza-Panduro *et al.* [24] presented a fixed-point SOM based neuro-processor using a Xilinx Virtex II FPGA for the analysis and classification of tension deformation patterns of knee ligaments, capable of recognising different sequences of movement patterns for a knee joint with damage to the anterior cruciate ligaments.

Kurdthongmee [25] presented a modified SOM implemented on FPGA, used for image quantisation. They used unsigned integer arithmetic operations suitable for moderate density FPGAs. A similar implementation where the distance, neighbourhood and learning rate computation is replaced with a simplified version, was presented by Chang *et al.* [26] and Porrmann *et al.* [27]. An efficient SOM architecture based on a new Frequency Adaptive Learning algorithm, which efficiently replaces the neighbourhood adaptation function of the original SOM, was presented in [26]. The design was implemented on a Xilinx FPGA and is capable of quantising 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 [27]. Using Xilinx FPGAs, the implementation achieves a speed-up of up to 190 times (with five FPGA modules on the RAPTOR2000 system) compared to a software implementation on a state-of-the-art personal computer. A similar

system implemented on a Xilinx Virtex II XC2V300, aimed at reducing the training processing time of SOM, has been presented in [28]. The design consists of 16 units in the input layer. The number of neurons in the output layer is divided into three sections: the processing unit array, the address generator and the controller. Compared with a software implementation, the design achieves approximately 89% speed-up. However, these systems still have fairly low numbers of neurons and modest speed-up, reflecting the significant amount of silicon area required to deal with the fixed-point arithmetic.

Recognising these issues, Yamakawa *et al.* [4] proposed a binary weighted vector SOM based on FPGA. The proposed SOM used binary data for both input and weight vectors. The Hamming distance is used as the distance metric between input and weight vectors. However, as their input data actually consists of integers the weight vector was updated with priority given to the most significant bit (MSB), thus attempting to utilise a hybrid scheme that treats the weights as a direct representation of integer values in some functions, and as binary strings in others. This produces some peculiarities (e.g. in treating the least- and most-significant bits equally in the Hamming-distance calculation). Nonetheless, the implementation was five times faster than the real number weighted SOM in software and 140 times faster in hardware, and achieved comparable results [4]. This highlights a key principle that the most successful design will take account of the nature of the hardware architecture, as demonstrated by Austin's [22] ability to implement a fast system on a low-cost digital hardware.

III. The Tri-state SOM

This paper introduces a tri-state SOM (the bSOM), which combines concepts from the traditional SOM [1], [2] with the tri-state logic pioneered in the PLN. The bSOM has the same essential structure as a standard SOM – an input layer and a competitive layer – and is capable of the same wide range of applications as the SOM. The bSOM takes a binary vector input and maintains tri-state prototype vectors "weights" with $\{0, 1, \#\}$ as the possible values. We use # to represent a "don't care" state (signifying that the corresponding input vector bit is matched whether it is set or clear). The resulting architecture implements very efficiently on FPGA, and the additional logic state significantly improves performance compared to a strictly binary architecture. In comparison with WNNs, the weight vectors have the same length as the input binary vector, whereas a WNN uses 2^N memory locations per logic node; moreover there is no need to sub-sample the input space and combine outputs in a pyramid structure, so the input part of the architecture is relatively simple.

One of the functions of standard SOMs [1], [2] is to reflect topological information prevalent in high dimensional input data in the organization of the one or two dimensional map of neurons [29]. Each neuron in a SOM has a topological neighbourhood, typically one- or two-dimensional and of a defined shape (e.g. circle, square or hexagon in two dimensions), with size of the region specified by a "radius" parameter r. For ease of hardware implementation we have used a one-dimensional neighbourhoods in the bSOM. Given a binary input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, all the units in the competitive layer are "connected" by corresponding prototype vectors, $\mathbf{w_j} = (w_{j1}, w_{j2}, \ldots, w_{jn})$. The bSOM training algorithm is discussed below, and compared and contrasted with the original SOM algorithm [1], [2] and Yamakawa's [4] implementation.

In contrast to Yamakawa [4], we assume that the input is strictly binary, and we use a tri-state weight vector. We used a specialised distance metric, and a specialised probabilistic update rule during training, both of which are necessary to reflect our tri-state weight structure. In contrast, in [4] the basic Hamming distance is used as a distance metric, despite its unsuitability for binary representation of fixed point integer inputs, and the weight $\mathbf{w_j}(t+1)$ is updated by $w_{ji}(t+1) = \overline{w_{ji}(t) \otimes x_i}$, for $i \in [0..N]$ for N-bit input vector, with $\otimes$ representing the exclusive OR operation, but with priority given to the MSB to reflect the integer encoding.

A. Distance Computation

We used a modified version of the Hamming distance to compare input to prototype vectors, as shown in Equation 1, for an input vector $\boldsymbol{x}$ and weight vector $\boldsymbol{w}$.

$$H(x, w_j) = \sum_{i=1}^{n} \left\{ \begin{array}{ll} 0 & \text{if } w_{ji} = \# \\ (\overline{x_i} \wedge w_{ji}) \vee (x_i \wedge \overline{w}_{ji}) & \text{otherwise} \end{array} \right\}. \quad (1)$$

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

This equation implies that any input bit value "matches" a '#' in the prototype vector. A consequence of this is that prototype vectors may effectively represent a region rather than a point or, viewed alternatively, may be selective to distance in some dimensions while ignoring others. This is a powerful feature of the approach. We may think of tri-state prototypes as corresponding to schemata in Holland's Genetic Algorithm [30], and so we refer to the modified distance metric as the Schema distance, D.

B. Winner Take All (WTA)

The unit with the smallest Schema distance to the input is defined as the winning neuron. We use the #-count (number of #'s in the weight string) as a tie-break when the Schema distances of multiple neurons to the input vector are the same – the winner is the neuron with the lowest #-count. This implies that we prioritise prototypes with a more specific representation.

C. Neighbourhood Selection and Weight Update

As in the original SOM and in [4], a neighbourhood $\mathcal{N}$ of neurons around the winning neuron $\mathbf{w}$ is selected and updated; the size of the neighbourhood progressively decreases. We use a probabilistic update rule, as follows:

- A bit in the weight vector is only updated if it is different from its corresponding input vector bit.
- An update probability is used for each iteration during training. This value decreases linearly as training progresses.

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

a) T

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

b) T_e

Fig. 1. (a) Left: the conditional Markov transition matrix for an equi-probable bit (T). (b) Right: the effective Markov transition matrix (T_e).

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. 2. The conditional Markov transition matrix after different iterations.

- A bit is updated by changing its value from 1 to #, 0 to # or # to (0 or 1) depending on the input bit value.

The behaviour of an individual bit can be modelled as a Markov chain with a conditional Markov transition matrix (T). Figure 1(a) illustrates the case where the probability that a particular bit is set, when that neuron wins, is 0.5. If the probability of applying the conditional Markov transition matrix is given as $p = 1 - \alpha$ (where α is the update rate), the resulting effective Markov transition matrix (T_e) for a bit to change is as shown in Figure 1(b). 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 2. The illustrated transition matrix settles after the 12th iteration. This supports the observation that the bSOM requires few iterations to converge, as compared to the original SOM and that presented in [4].

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

TABLE I
SPECIFICATION OF THE BSOM AS IMPLEMENTED ON FPGA.

IV. TRI-STATE SOM ON FPGA ARCHITECTURE

The most critical aspect of any hardware design is the selection of an architecture which provides the most efficient and effective implementation [26]. The specifications of the circuit implemented on FPGA is given in Table I, with its corresponding block diagram in Figure 3. The circuitry is made up of five basic blocks: the weight initialisation, pattern input, Winner Take All, neighbourhood update and display

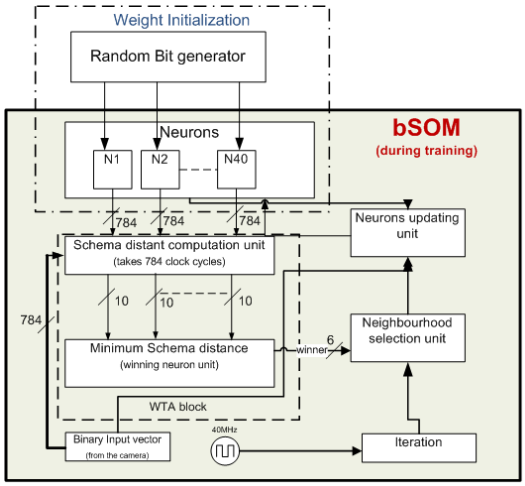

Fig. 3. A block diagram of the bSOM design on FPGA.

blocks. The circuitry is parameterized by the input bit width, N, and requires only a simple reconfiguration for a different design. Three of the five blocks run in parallel: the pattern input, Winner Take All and display (output) block. The weight initialisation block is triggered only at start-up. Similarly, the neighbourhood update block is triggered when a winning node is identified for an input binary vector. Details of the five basic blocks are presented in the following sections.

A. Weight Initialisation block

This block is used to randomly initialize all the weight vectors in the network. All the neurons in the network are initialised in parallel bit-by-bit; hence, it takes as many clock cycles as there are bits in the binary input vector to complete the initialisation. The hardware architecture presented here has been tested with binary image characters of size 28×28, totalling $N = 784$ bits (and also with binary signatures from moving objects with N = 768 bits). The sizes of the input and weight vectors are all set to N bits and can easily be altered for any input size. The presented implementation takes exactly N clock cycles to completely initialise all the neurons.

B. Pattern Input block

This block is used to acquire the binary input vector (or binary image) from an external source. The size of the input vector, N, is pre-configured and the input is complete when a total of N bits is read from the input source. This binary data is stored in the input vector and then passed onto the schema distance computation unit for further processing.

C. Winner Take All block

This block is made up two parts, the distance computation unit and the winning neuron unit. The distance computation unit is used to compute the Schema distance between the input binary vector and all neurons in the bSOM. The Schema distance between the input vector $\mathbf{x}$ and a neuron $\mathbf{w}_j$, as shown in Equation 1 is a bitwise operation, and hence takes

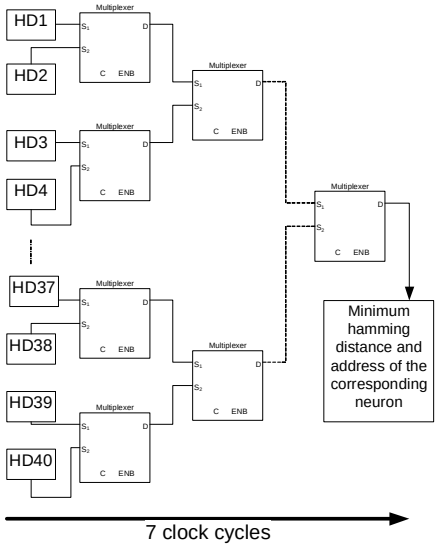

Fig. 4. Structure of the minimum Schema distance unit in the WTA unit.

as many clock cycles as there are bits in the input vector. Since the Schema distances for all the neurons are computed in parallel, it takes exactly N clock cycles to complete the distance computations for all the neurons in the network.

The winning neuron unit uses the results from the Schema distance computed in the distance computation unit to identify the winning neuron. The design, as shown in Figure 4, uses a tree-structured series of comparators to select the minimum of a pair of two inputs. For an implementation with 40 neurons, the design takes exactly seven clock cycles to compute the node with the minimum Schema distance.

D. Neighbourhood update block

This block is used 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 neighbourhood size is initialized to 4, and decrements every $I/4$ iterations until it reaches a minimum of 1, where I is the total number of iterations. The update requires a random number generator, which is complex to implement in hardware and computationally expensive. To avoid these costs, a look-up table 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. Mismatching bits in the neuron vector are updated as discussed in Section III-C. A # is implemented as binary '10'.

E. Output display blocks

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

F. Implementation Platform

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.

G. Training Speed

To compare the training speeds of bSOM and cSOM on the FPGA architecture, a simplified version of the cSOM has been implemented on the Xilinx Virtex-4 FPGA. In the simplified version of the cSOM the Manhattan distance is used instead of the Euclidean distance. Also to accommodate the fine grain learning in cSOM, 8 bits are used to represent values ranging from 0 to 1 in fixed point format. The design for the bSOM can be clocked at 40MHz and 25MHz for the cSOM. The resource utilisation of the two implementations are given in table II. The cSOM implementation takes 3 times as many clock cycles as the bSOM, due to the intermediate arithmetic operations required for updating the 8-bit fixed point memory locations. At 25MHz the cSOM is capable of training the system with approximately 10,000 patterns per second. Also, at 40MHz the bSOM implementation can be trained with approximately 25,000 patterns per second; representing a 2.5 fold improvement over the training time of the cSOM on FPGA. The clock frequencies of 40MHz and 25MHz also include the design for controlling the external logic for the VGA and the camera. This is the actual hardware test and the most stable clock frequencies for the two implementations. The frequencies could be much higher without the requirement to interface these devices. Table II gives the details of the resource utilisation of the FPGA implementations for the 784-bit character recognition problem.

Resource		**bSOM**		**cSOM**	
Name	**Total**	**Used**	**Per.(%)**	**Used**	**Per.(%)**
Flip Flops	135,168	4,095	3	7,522	5
4 input LUTs	135,168	18,387	13	35,947	26
bonded IOBs	768	147	19	147	19
Occupied Slices	67,584	11,468	16	23,413	34
RAM16s	288	43	14	64	22
BlockRAM	5,184	790	15.25	1,145	22.09

TABLE II
RESOURCE UTILISATION OF BSOM AND CSOM, USING VIRTEX-4 *XC4VLX160*, PACKAGE *FF1148* AND SPEED GRADE *-10*.

V. APPLICATIONS AND EXPERIMENTAL RESULTS

The performance of the bSOM has been verified using two practical applications: handwritten character recognition, and moving object identification. To verify the performance of the bSOM, the MNIST database of handwritten digits [31], sample shown in Figure 5, was used to test the implementation both

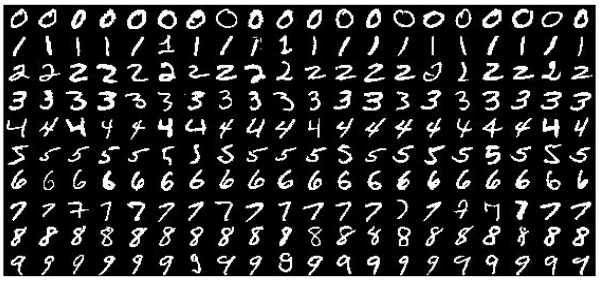

Fig. 5. Sample of the MNIST database.

in PC simulations and on the FPGA hardware architecture. A comparison on the PC between the original SOM as presented by Kohonen in [1], [2] (herein referred to as the cSOM), a strictly binary SOM (BSOM) and the proposed tri-state SOM (bSOM) algorithms is also given in this section. Although the bSOM is meant for hardware implementation, it has been implemented on a PC using MATLAB to enable comparison with the original SOM. To illustrate the importance of the tri-state $(0, 1, \#)$ rather than binary $(0, 1)$ representation, the BSOM version uses the Hamming distance metric, but otherwise is implemented with the same parameters as the bSOM.

A. Handwritten Character Recognition

To illustrate the comparative performance of the bSOM in cluster analysis and topological ordering, we have tested the system on the MNIST handwritten character dataset [31]. To evaluate how effective the clustering is, after training the neurons were visually inspected and labels (0 to 9) were assigned to each neuron. A labelled independent test data (10,000 numeric characters) was then used to test the classification accuracy of the three hand-labelled SOMs.

The software based simulation of the bSOM was 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 neurons, neighbourhood size and learning rate – for all three models, to determine the number of neurons required to represent all 60,000 patterns in the dataset (see Figure 5).

Table III illustrates the influence of the different parameters of the cSOM and bSOM performance. Although the bSOM performs better than the cSOM, there is significant improvement in performance for the cSOM as the number of iterations increases. Increasing the number of neurons in the network increases the performance of both the cSOM and bSOM, with some of the neurons left unused for larger networks.

Experiments were conducted with the number of neurons ranging from 10 to 100 in steps of 10. The bSOM results improve with increasing numbers of neurons until performance plateaus at 80 neurons (with minimal improvement thereafter). The initial neighbourhood size (4) was determined using the cSOM, and adopted for the other implementations.

After empirically selecting parameter values, tests were conducted to compare the convergence of the bSOM, cSOM and BSOM at 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200 and 500 iterations. The experiment was repeated ten times at each iteration count with the exception of 500, which (due to computational load) was repeated only five times. In the case of the original SOM, repetitions did not make any significant difference to the results, whereas the results of the strict binary and tri-state SOM showed some variability.

Iterations	No. of Nodes	Correct Classification (%)	
		cSOM	bSOM
1	100	11.35	59.8
2	100	67.47	77.82
3	100	74.36	76.37
4	100	75.95	81.15
5	100	78.38	81.69
6	100	79.49	82.86
7	100	79.87	81.22
8	100	78.96	80.87
9	100	79.64	81.43
9	200	86.27	88.30
9	300	86.81	89.46
10	50	75.67	76.22
10	60	76.20	75.29
10	70	76.71	76.11
10	80	77.90	77.97
10	90	78.82	80.35

TABLE III
THE PERFORMANCE OF THE CSOM AND BSOM FOR VARIOUS NETWORK SIZES AND NUMBERS OF ITERATIONS.

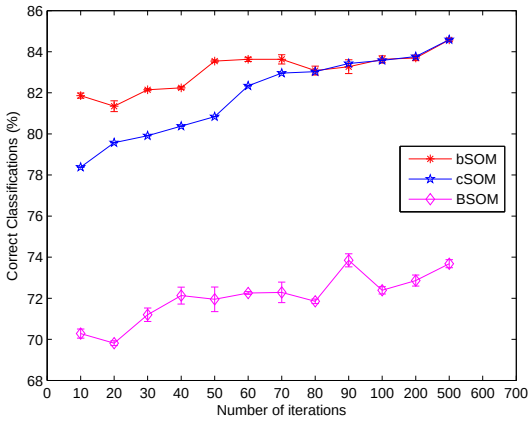

Fig. 6. Performance of the three different SOM implementations for different iteration counts. The average correct classification rate is illustrated with error bars.

Figure 6 illustrates the results. The BSOM has markedly inferior performance. The cSOM and bSOM appear to have similar performance at high iteration counts; however, the bSOM performs better at low iteration counts and plateaus around 50 iterations; increasing the number of iterations beyond this point does not make a significant difference. The cSOM appears to plateau after 700 iterations (not illustrated) with a performance level of 89%; it took approximately 50 hours to complete one training run at this number of iterations, and we did not repeat the experiment.

Samples of the resulting topological maps with 100 neurons in each network (after 100 iterations) are illustrated in Figure 7 for cSOM, BSOM and bSOM. We note that the specific assignment of neurons to patterns is not significant,

Fig. 7. Results of the three implementations using the [31] dataset. Left: original SOM (cSOM); middle: tri-state SOM (bSOM); right strict binary SOM (BSOM). In cSOM grey-levels are used to illustrate weight magnitude; in BSOM # is represented by mid-grey.

Iter.	MNIST dataset			Binary Signatures		
	cSOM	bSOM	BSOM	cSOM	bSOM	BSOM
10	78.38%	**81.86%**	70.28%	81.84%	**84.41%**	68.92%
20	79.57%	**81.35%**	69.89%	83.06%	**84.56%**	68.99%
30	79.91%	**82.15%**	71.18%	84.50%	**84.85%**	75.81%
40	80.38%	**82.24%**	72.12%	84.05%	**84.05%**	69.00%
50	80.84%	**83.54%**	71.94%	83.98%	**85.03%**	74.49%
60	82.34%	**83.62%**	72.25%	84.70%	**85.91%**	77.94%
70	82.96%	**83.60%**	72.28%	85.03%	**85.74%**	79.80%
80	83.03%	**83.08%**	71.86%	**85.01%**	84.58%	70.35%
90	**83.43%**	83.21%	73.82%	**85.20%**	84.40%	77.12%
100	83.58%	**83.63%**	72.38%	**85.15%**	84.58%	76.79%
200	**83.77%**	83.69%	72.86%	84.68%	**86.44%**	76.13%
300	–	–	–	**86.71%**	84.23%	69.17%
400	–	–	–	**87.33%**	86.05%	83.17%
500	**84.59%**	84.57%	73.68%	**87.42%**	86.89%	78.56%

TABLE IV
LEFT: AVERAGE PERFORMANCE OF THE THREE SOM IMPLEMENTATIONS FOR 12 DIFFERENT ITERATION COUNTS USING THE MNIST DATASET [31]. RIGHT: AVERAGE PERFORMANCE FOR 14 DIFFERENT ITERATIONS USING THE TARGET IDENTIFICATION DATABASE.

	Iter.	FN%	FP%	TN%	TP%	PCC%	JC%
tSOM	10	3.64	2.89	82.72	10.75	93.47	62.24
cSOM	10	4.23	3.36	82.25	10.16	92.42	57.27
BSOM	10	5.22	5.64	79.97	9.17	89.15	45.82
tSOM	100	3.58	2.91	82.70	10.81	93.51	62.50
cSOM	100	4.16	3.42	82.19	10.23	92.43	57.49
BSOM	100	5.27	5.37	80.24	9.12	89.36	46.18
tSOM	200	3.59	2.87	82.73	10.81	93.54	62.56
cSOM	200	4.18	3.40	82.21	10.21	92.42	57.39
BSOM	200	5.16	5.76	79.85	9.23	89.08	45.84
tSOM	300	3.63	2.87	82.73	10.77	93.50	62.37
cSOM	300	5.25	5.31	80.29	9.15	89.44	46.41
BSOM	300	5.15	5.75	80.03	9.07	89.10	45.41

TABLE V
THE PIXEL LEVEL ACCURACY MEASURE FOR DIFFERENT ITERATIONS USING THE 3 IMPLEMENTATIONS. ITER. IS THE NUMBER OF ITERATIONS.

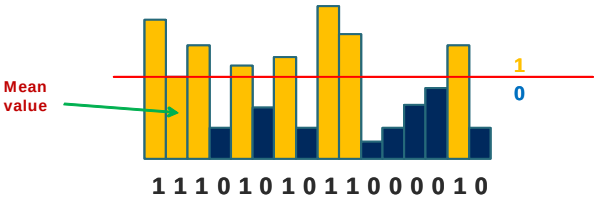

Fig. 8. A sample 16 bin histogram. The binary signature is extracted by thresholding at the mean bin frequency value.

although the one-dimensional topological ordering is reflected (neighbourhood runs across rows in a scan-line fashion) in the clustering. The cSOM captures ambiguities well (appearing as "blurring" of the pattern in some nodes). The bSOM can also achieve this to some extent, whereas the BSOM reflects only specific patterns. It is the ability of the bSOM to capture at least some level of ambiguity which distinguishes it from the BSOM in terms of performance.

Table IV (left hand side of table) shows the average performance level from figure 6 numerically. Although in this experiment the SOM's have been used for clustering and a post-hoc analysis of correct classification conducted, for comparison we list the performance of various classifiers on this dataset, as presented in the literature. Reported accuracy levels are 88% using a linear classifier (1-layer NN) [31], 84% using Sparse Distributed Memory [32], 94% using Support Vector Machine and 99.5% using a two stage pattern recognition architecture using feature extraction (a large convolutional neural network with unsupervised pre-training) [33].

The performance measure in table IV is the object level classification measure. Table V gives the pixel level accuracy measure using the MNIST dataset for four different iterations with the three implementations. This is the pixel level comparison of the 10,000 test data and the represented neurons. Table V also gives two statistical measures derived from the False Negative (FN), False Positive (FP), True Negative (TN) and True Positive (TP). The Jaccard Coefficient (JC) neatly characterizes the performance in a single measure that takes into account both FP and FN errors and it is computed as $\frac{TP}{TP+FP+FN}$. Percentage Correct Classification (PCC), a widely used method for assessing a classifiers performance has also been give in table V and is computed as $\frac{TP+TN}{TP+FP+TN+FN}x100$.

B. Target identification.

Our second implementation illustrates the bSOM as a component of a surveillance system. The system, fully implemented on FPGA, analyzes real-time video, applies background differencing [34], segments multiple objects and tracks them. The tracking and segmentation modules yield individual objects, represented using a bounding box. The bSOM is used to perform appearance-based target identification – a number of known objects (individuals) are learned by the system, and during tracking the bSOM is used identify which object(s) is/are in view.

The objects are represented using a simple binary signature, extracted from the colour histograms – this is frequently sufficient to identify individual objects from a reasonably small set. However, the approach generalises to more sophisticated feature extraction techniques. A 768 bin histogram is generated; 256 bins for each of the RGB colour components. To convert this into a binary signature the average bin frequency, μ_{bin}, is computed; any bin with a value greater than or equal to μ_{bin} is represented as binary 1, 0 otherwise (see Figure 8).

A binary feature vector (binary signature), $\mathbf{x}= \{x_1, x_2, \ldots, x_N\}$ for $N = 768$ is generated as in

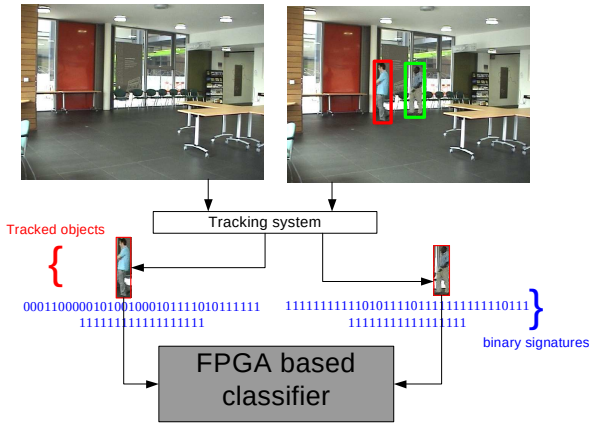

Fig. 9. Processing of objects for identification. The tracking system detects objects and constructs a bounding box. A binary signature is extracted from the colour histogram of each object, and fed to the bSOM for identification.

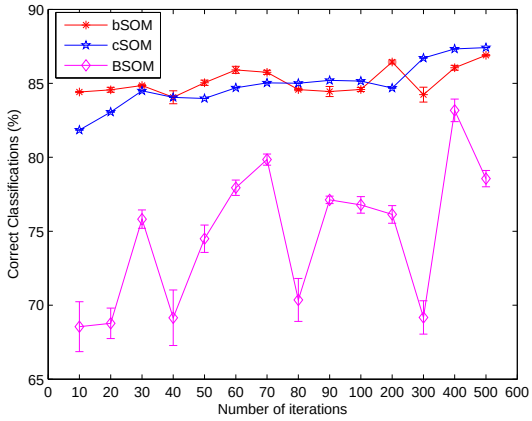

Fig. 11. Performance of the three different SOM implementations for 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400 and 500 iterations using binary signatures extracted from the colour histogram for nine objects.

Equation 2.

$$x_i = \begin{cases} 1 & \text{if } bin_i >= \mu_{bin} \\ 0 & \text{otherwise} \end{cases} . \quad (2)$$

To test the short term-recognition of the bSOM with signatures extracted from the colour histogram, a limited number of objects (nine people) have been used to train a fixed size bSOM. The bSOM is trained using binary signatures collected from all moving objects in a training sequence. The number of unique objects that appear in the scene determines the number of neurons required in the bSOM. Ideally, the number of neurons should be the same as the total number of unique objects. However, due to partial occlusion, camera jitter, over segmentation and under segmentation, the appearance and hence the histogram for an object may vary from frame to frame, so that each individual is represented by multiple nodes.

After training the network with binary signatures extracted from $2,248$ manually-labelled objects, we use a win-frequency based algorithm to automatically label nodes for object identification. For each node, we count how many of the training patterns for which it "wins" the competition correspond to each known object. The node is labelled as the most frequently-assigned object. To test the performance, $1,139$ manually labelled independent test data are used. During the testing phase, the winning neuron is identified. If the minimum Schema distance exceeds a threshold value set during training the object is classified as unknown; otherwise it is identified using the node label.

The object identification system has been tested with video data recorded over a period of two hours with a total of 18,122 frames. The video was recorded in an indoor environment, very close to the exit of a building. Typically people enter the building and leave at the same exit point. The scene has normal office furniture, which partially occlude the moving object in some locations. There is some variation in lighting conditions, particularly around the wide transparent windows, see Figure 9. Frames from the first 30 minutes with nine different persons entering the building were used to train the system. A tracking system is used to segment and extract the pixels of all moving objects as shown in figure 9. Objects with less than 768 pixels are filtered as noise, which also avoids values of μ_{bin} less than 1 in Equation 2. Figure 10 shows three of the nine objects used to train the bSOM. In the figure, the actual object being tracked is shown to the left with its binary signatures over the period of time that it appears in the scene shown to the right. The binary signatures are shown as images and each row in the image to the right corresponds to the 768 bits representing the binary signature for its colour histogram.

Tests were conducted with the number of neurons ranging from 10 to 100 in increments of 10. For networks with more than 50 neurons, the recognition level for both the bSOM and cSOM exceeds 90%, but some neurons do not get used; 40 neurons was adequate for good performance. There were nine distinct objects, therefore roughly four neurons per object in this environment. The average performance of the cSOM, bSOM and BSOM using 40 neurons are presented to the right of Table IV; these figures are also illustrated in figure 11. The performance is consistent with the overall observations on the MNIST dataset; the bSOM and cSOM have comparable performance; BSOM has significantly worse performance; the bSOM trains relatively quickly, although the difference is not so marked on this dataset.

C. Statistical Significance of Results

This section examines the statistical significance of the performance of the three SOM implementations (cSOM, bSOM and BSOM) presented in the sections V-A and V-B. We used the Wilcoxon rank-sum test to determine whether there is any significant difference between the classification performance of the three algorithms. A one-tailed test was used to test whether higher average performance by one algorithm over another was statistically significant.

Table VI shows the Wilcoxon statistic (z), ϱ (the asymptotic significance) values and the significance for all the 12 iterations for the MNIST dataset. The ϱ values from Table VI suggest that bSOM significantly outperform cSOM for iterations less than 80 at the 5% significance level. There is no

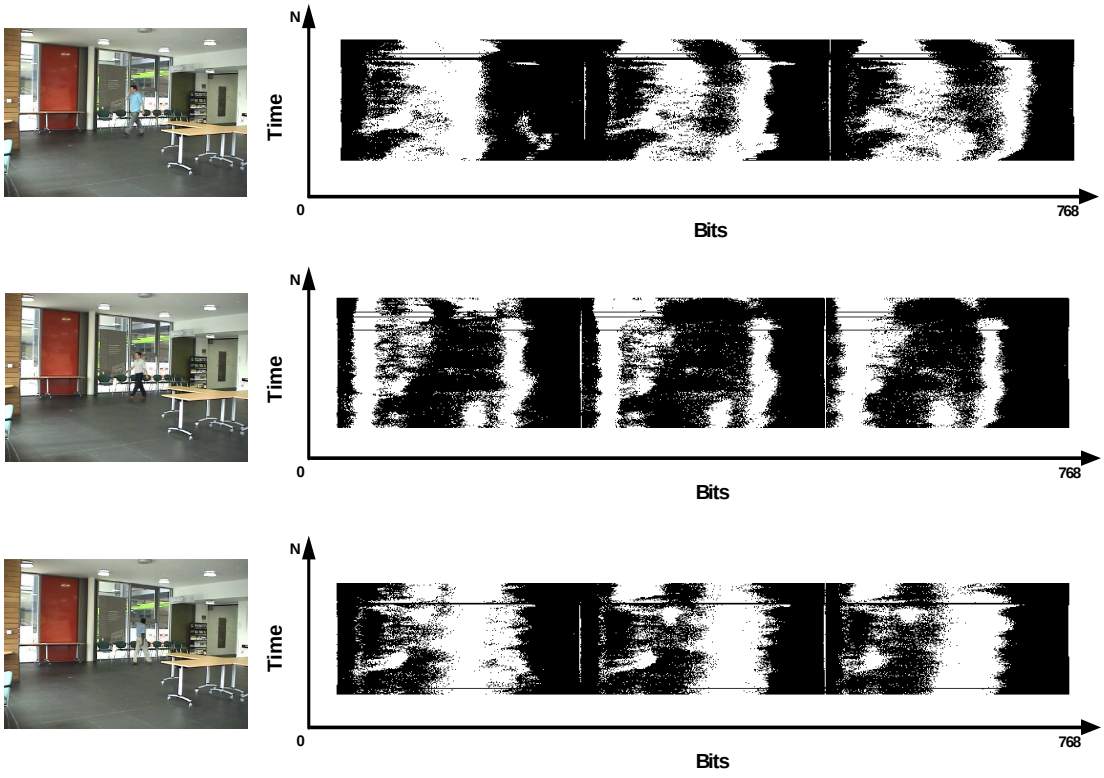

Fig. 10. Sample binary signatures from three of the nine moving objects. A representative frame for each object is shown to the left. The corresponding binary signature and its time evolution is shown to the right; each 768-bit wide row represents the signature on a single frame. The signature varies over time but shows fairly high level of consistency, and signatures of different objects are distinctly different.

Iter.	Mean Rank cSOM	Mean Rank bSOM	Rank Sum	z	Significance 1-tailed (ϱ)	
10	5.50	15.50	55	-3.99	0.000	$\succ$
20	5.50	15.50	55	-3.99	0.000	$\succ$
30	5.50	15.50	55	-4.01	0.000	$\succ$
40	5.50	15.50	55	-4.01	0.000	$\succ$
50	5.50	15.50	55	-4.01	0.000	$\succ$
60	5.50	15.50	55	-3.99	0.000	$\succ$
70	6.50	14.50	65	-3.19	0.000	$\succ$
80	10.50	10.50	105	0	0.519	$-$
90	12.50	8.50	125	1.58	0.057	$-$
100	12.50	8.50	125	1.57	0.057	$-$
200	11.50	9.50	115	0.7673	0.222	$-$
500	10.50	10.50	105	0	0.519	$-$

TABLE VI
WILCOXON RANK-SUM TEST FOR THE MNIST DATASET. SIGNIFICANT DIFFERENCES ARE INDICATED BY ϱ LESS THAN 0.05, $\succ$ INDICATES WHERE BSOM OUTPERFORMS CSOM; $-$ INDICATES NO SIGNIFICANT DIFFERENCE BETWEEN BSOM AND CSOM.

Iter.	Mean Rank cSOM	Mean Rank bSOM	Rank Sum	z	Significance 1-tailed (ϱ)	
10	5.50	15.50	55	-4.00	0.000	$\succ$
20	6.50	14.50	65	-3.19	0.000	$\succ$
30	5.50	15.50	55	-4.00	0.000	$\succ$
40	12.50	8.50	125	1.66	0.041	$\prec$
50	6.50	14.50	65	-3.19	0.000	$\succ$
60	5.50	15.50	55	-4.00	0.000	$\succ$
70	6.50	14.50	65	-3.19	0.000	$\succ$
80	15.50	5.50	155	4.00	0.000	$\prec$
90	14.50	6.50	145	3.19	0.000	$\prec$
100	12.50	8.50	125	1.58	0.057	$-$
200	5.50	15.50	55	-4.00	0.000	$\succ$
300	14.50	6.50	145	3.19	0.000	$\prec$
400	15.50	5.50	155	4.00	0.000	$\prec$
500	15.50	5.50	155	4.00	0.000	$\prec$

TABLE VII
WILCOXON RANK-SUM TEST FOR THE OBJECT IDENTIFICATION BINARY SIGNATURES. SIGNIFICANT DIFFERENCES ARE INDICATED BY ϱ LESS THAN 0.05, $\succ$ INDICATES WHERE BSOM OUTPERFORMS CSOM; $-$ INDICATES NO SIGNIFICANT DIFFERENCE BETWEEN BSOM AND CSOM; $\prec$ INDICATES CSOM OUTPERFORMS BSOM.

statistical significance between the performance for iteration greater than or equal to 80. This test shows that bSOM trains more quickly than cSOM, but that ultimate performance is comparable.

Table Table VII shows the Wilcoxon rank-sum test results for the object identification problem. As with MNIST, bSOM outperforms cSOM for smaller iterations (10–70), with the exception of iteration 40. However, cSOM outperforms bSOM for higher iterations (80–500), with the exception of iteration 100 and 200. There is no statistically significant difference at iteration 100. We conclude that bSOM trains more quickly than cSOM on this dataset, but ultimately cSOM has marginally higher performance.

VI. Conclusion

We have presented a new neural network architecture, the tri-state self-organising map, which is suitable for clustering, anomaly detection and classification. By utilising the concept of tri-state logic, originally presented in weightless neural networks, we can produce an efficient system which has comparable performance to a traditional real-valued SOM in handling binary input data (in contrast to a simple binary system using the Hamming distance), with significantly greater computational efficiency. The bSOM is particularly well-suited to an FPGA platform, trains in less iterations than the original SOM and has a much lower implementation foot-print. We have demonstrated the potential use of the bSOM in hand-written character recogntion, and in security surveillance systems as an object identification system using binary signatures extracted from colour histograms. The work presented here forms part of an end to end surveillance system fully implemented on FPGA. In the future we will demonstrate further applications of the tri-state SOM, integration with more sophisticated binary signature extraction algorithms, and integrated self-optimisation, including on-line learning.

Acknowledgment

This work was supported by the UK's Technology Strategy Board (TSB) under the BRAINS Project. Special thanks goes to the project team; Prof. Derek Sheldon, Nigel Priestley, Mervyn Hobden, Peter Hobden and Cy Pettit.

References

[1] T. Kohonen, *Self-Organizing Maps*, Springer, New York, 1995.
[2] T. Kohonen, *Self-Organizing Maps*, Springer Series in Information Sciences, Third Extended Edition, 2001.
[3] 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. 3, August 1998.
[4] K. H. T.Yamakawa and T. Hiratsuka, "Advanced self-organizing maps using binary weight vector and its digital hardware design," in *Proceedings of the 9th Int. Conf. on Neural Information*, 2002.
[5] J. Pena and M. Vanegas, "Digital hardware architecture of Kohonen's Self Organizing Feature Maps with exponential neighboring function," in *Proceedings of IEEE International Conference on Reconfigurable Computing and FPGA*, 2006.
[6] T. G. Clarkson, D. Gorse, and J. G. Taylor, "pram automata," in *Proceedings of 1990 IEEE International Workshop on Cellular Neural Networks and their Applications*, December 1990, pp. 235–243.
[7] T. Ludermir, A. de Carvalho, A. Braga, and M. de Souto, "Weightless neural models: a review of current and past works," *Neural Computing Surveys*, vol. 4, pp. 41 –61, 1999.
[8] W.-k. Kan and I. Aleksander, "A probabilistic logic neuron network for associative learning," pp. 156–171, 1989.
[9] I. Aleksander, M. D. Gregorio, F. Fran?a, P. Lima, and H. Morton, "A brief introduction to weightless neural systems," in *Proceedings of the 17th European Symposium on Artificial Neural Networks*, 2009, pp. 299–305.
[10] J. Austin, *RAM-based Neural Network*, 9th ed. World Scientific, 1998, pp. 43–50.
[11] E. do Valle Simoes, L. F. Uebel, and D. A. C. Barone, "Hardware implementation of ram neural networks," *Pattern Recognition Letters*, vol. 17, no. 4, pp. 421 – 429, 1996.
[12] I. Aleksander and C. Myres, "Learning algorithms for probabilistic logic nodes," in *Abstract of the First INNS*, 1989, p. 205.
[13] T. Clarkson, D. Gorse, J. Taylor, and C. Ng, "Learning probabilistic ram nets using vlsi structures," *IEEE Transactions on Computers*, vol. 41, pp. 1552–1561, 1992.
[14] T. Clarkson, Y. Guan, J. Taylor, and D. Gorse, "Generalization in probabilistic ram nets," *Neural Networks, IEEE Transactions on*, vol. 4, no. 2, pp. 360 –363, march 1993.
[15] I. Manolakos and E. Logaras, "High throughput systolic som ip core for FPGAs," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 61–64.
[16] R. Cofer and B. Harding, "Basics of core-based fpga design: Part 1 ? core types and trade-offs," *EE Times Design*, 2011.
[17] J. G. Tong, I. D. L. Anderson, and M. A. S. Khalid, "Soft-core processors for embedded systems," *IEEE 18th International Confernece on Microelectronics (ICM)*, pp. 170 –173, 2006.
[18] K. Appiah, A. Hunter, H. Meng, S. Yue, M. Hobden, N. Priestley, P. Hobden, and C. Pettit, "A binary self-organizing map and its fpga implementation," in *International Joint Conference on Neural Networks*, June 2009, pp. 164 –171.
[19] K. Appiah, A. Hunter, P. Dickinson, and H. Meng, "Binary object recognition system on FPGA with bsom," in *Proceedings of 23rd IEEE International System on Chip Conference (SOCC)*, September 2010.
[20] R. Gadea, J. Cerdá, F. Ballester, and A. Mocholí, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in *ISSS '00: Proceedings of the 13th international symposium on System synthesis*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 225–230.
[21] P. Moerland and E. Fiesler, "Hardware-friendly learning algorithms for neural networks: an overview," in *Proceedings of 5th international conference on microelectronics for neural networks and fuzzy systems*, 1996.
[22] J. Austin, "High speed image segmentation using a binary neural network," in *Neurocomputation in Remote Sensing Data*. Springer-Verlag, 1996.
[23] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural network implementation using FPGA : issues and application," *International Journal of Information Technology*, vol. 4, no. 2, 2008.
[24] J. Raygoza-Panduro, S. Ortega-Cisneros, and E. Boemo, "FPGA implementation of a synchronous and self-timed neuroprocessor," in *Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2005)*. IEEE Computer Society, 2005.
[25] W. Kurdthongmee, "A novel hardware-oriented Kohonen SOM image compression algorithm and its FPGA implementation," *The EUROMICRO Journal of Systems Architecture*, vol. 54, no. 10, pp. 983–994, October 2008.
[26] C. Chang, M. Shibu, and R. Xiao, *Self Organizing Feature Map for Color Quantization on FPGA*. Springer, 2006, pp. 225–246.
[27] M. Porrmann, U. Witkowski, and U. Ruckert, *Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware*. Springer, 2006, pp. 247–270.
[28] R. Agundis, G. Girones, C. Palero, and D. Carmona, "A mixed hardware/software sofm training system," *Computaci?n y Sistemas*, vol. 4, April 2008.
[29] S. Kumar, *Neural Networks a classroom approach*. McGraw-Hill, 2004.
[30] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.
[31] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," 1998.
[32] H. Meng, S. Yue, A. Hunter, K. Appiah, M. Hobden, N. Priestley, and P. Hobden, "A modified sparse distributed memory model for extracting clean patterns from noisy inputs," *International Joint Conference on Neural Networks, IJCNN*, 2009.
[33] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *The 20th IEEE International Conference on Conmputer Vision, ICCV*, September 2009.
[34] K. Appiah and A. Hunter, "A single-chip FPGA implementation of real-time adaptive background model," *IEEE International Conference on Field-Programmable Technology*, pp. 95–102, December 2005.

Dr. Kofi Appiah is a Senior Researcher at the University of Lincoln in England. He has previously studied at the Kwame Nkrumah University of Science and Technology, Kumasi - Ghana where he obtained a B.Sc. in Computer Science, the Royal Institute of Technology, Stockholm - Sweden where he obtained an M.Sc. in Electrical Engineering, and University of Oxford, Oxford - England, where he obtained an M.Sc. in Computer Science. He was awarded his PhD by the University of Lincoln in 2010. His research interests included computer vision, and highly parallel software and hardware architectures. Kofi has been a member of the IEEE since 2004.

Prof. Andrew Hunter studied for his B.Sc. Mathematics and Computing and Ph.D. Computer Graphics at Bath University. He worked for several years in industry, in computer graphics and CAD/CAM software, before returning to academia. Prof. Hunter held posts at Sunderland and Durham Universities, before joining the University of Lincoln and becoming Head of Department of Computing in 2004, became Dean of Research in 2007 and Dean for Science, Technology and Engineering in 2010. Professor Hunter has published over 80 academic papers, including more than 20 in international journals. He has also developed several freeware and commercial artificial intelligence software packages.

Dr. Patrick Dickinson is a Senior Lecturer at the University of Lincoln. He has a PhD from Lincoln for his work in computer vision, and has previously studied at the Universities of Oxford and Southampton. His current research interests include computer vision applied to the surveillance of wildlife, games AI and the use of games metrics. He is a member of the Centre for Computer Vision and Robotics, and co-founder of the Games Research Group at Lincoln. Patrick has been a member of the IEEE since 2004.

Dr. Hongying Meng is a Lecturer at Brunel University. Before that, he held research positions in several UK universities including University College London (UCL), University of Lincoln, University of York and University of Southampton. He received his Ph.D. in Communication and Electronic Systems from Xi'an Jiaotong University and was a lecturer in Electronic Engineering Department of Tsinghua University in China. Dr Meng has a wide research interests including digital signal processing, machine learning, computer vision, embedded systems and affective computing. He is a member of IEEE and a Fellow of The Higher Education Academy (HEA).