J Sign Process Syst (2014) 76:235–247
DOI 10.1007/s11265-014-0886-z

Algorithm and Architecture of Fully-Parallel Associative Memories Based on Sparse Clustered Networks

Hooman Jarollahi · Naoya Onizawa · Vincent Gripon · Warren J. Gross

Received: 25 September 2013 / Revised: 24 February 2014 / Accepted: 19 March 2014 / Published online: 12 April 2014

Abstract Associative memories retrieve stored information given partial or erroneous input patterns. A new family of associative memories based on Sparse Clustered Networks (SCNs) has been recently introduced that can store many more messages than classical Hopfield-Neural Networks (HNNs). In this paper, we propose fully-parallel hardware architectures of such memories for partial or erroneous inputs. The proposed architectures eliminate winner-take-all modules and thus reduce the hardware complexity by consuming 65 % fewer FPGA lookup tables and increase the operating frequency by approximately 1.9 times compared to that of previous work. Furthermore, the scaling behaviour of the implemented architectures for various design choices are investigated. We explore the effect of varying design variables such as the number of clusters, network nodes, and erased symbols on the error performance and the hardware resources.

H. Jarollahi (✉) · W. J. Gross
Department of Electrical and Computer Engineering, McGill University, 3480 University Street, Montreal, Québec, Canada, H3A 0E9
e-mail: hooman.jarollahi@mail.mcgill.ca

W. J. Gross
e-mail: warren.gross@mcgill.ca

N. Onizawa
Research Institute of Electrical Communication, Tohoku University, 2-1-1 Katahira, Aoba-ku, Sendai, Miyagi, Japan, 980-8577
e-mail: nonizawa@m.tohoku.ac.jp

V. Gripon
Electronics Department, Télécome Bretagne, Brest, France
e-mail: vincent.gripon@telecom-bretagne.eu

Keywords Associative memories · Hopfield neural networks · FPGA-based VLSI implementation · Sparse clustered networks

1 Introduction

Associative Memories are structures that store data patterns such that, when a partial pattern is presented as an input, the memory can quickly recover the full pattern. Therefore, the function of an associative memory is different from indexed memories in which data is accessed by presenting an explicit address as input which is often determined after an exhaustive search operation. Associative memories eliminate exhaustive search operations and are thus attractive in certain applications such as data mining and implementation of sets, where the computations can benefit from their specific functioning [5, 9, 12, 15, 16].

Content-Addressable Memories (CAMs) are specific categories of associative memories in which two data fields are associated with each other: An input field containing pointers (tags) to an output field, where the actual data is stored. The architecture of the tag field is referred to as CAM in literature by which an input can be matched against the stored contents during the search process. CAMs are used in a variety of applications such as image processing [13] and network routers [3, 8]. However, CAMs consume large amounts of power due to the operation of parallel comparators performing brute-force search [14]. In applications where fully-associative CAMs are required such as Translation Lookaside Buffers (TLBs) [1, 2, 4], they only contain a few entries and dissipate significant amounts of the total power.

A classical state-of-the-art associative memory is the Hopfield Neural Network (HNN) [7]. HNNs are also known

as auto-associative memories, where inputs and outputs are not separated in the way they are represented in the network as opposed to their hetero-associative counterpart. In this paper, we always refer to auto-associative memories. An HNN is capable of storing (learning) the patterns (messages) by linking all of their constituent data bits, and storing the links in a memory module. Using the stored links, HNNs can then retrieve the full length of a previously stored pattern given only a part it.

Although HNNs demonstrate attractive learning and retrieving features, there exists major drawbacks with them: First, in order to increase the size of the memory, the length of the messages need to be unnecessarily increased. Therefore, the number of different messages HNNs can learn (diversity) is small since, due to a fixed number of available physical memory bits, they can only store few long messages instead of many short ones. Second, the ratio between the number of information bits that it can store (capacity) to the memory bits that it requires to store the links (efficiency) approaches zero as the memory size is increased.

A new class of associative memories, also known as Sparse Clustered Networks (SCNs) has been proposed in [5] that addresses the drawbacks of the HNNs. It proposes an algorithm for associative memories that features a large diversity and a significantly superior efficiency. SCNs are graphically modelled, similar to HNNs, by nodes (neurons) and connections between them. However, unlike HNNs, the neurons are grouped into clusters and the connection weights are binary [6].

A proof-of-concept hardware architecture of an SCN-based associative memory has been proposed in [10], where the authors show that when implemented in hardware, the architecture provides significant speedup compared to its software counterpart. However, the previous architecture requires resource-hungry Winner-Take-All (WTA) modules, which are based on a classical compare-and-select algorithm.

In this paper, we present an extended version of a previously proposed work in [11], where preliminary hardware implementation results for two reduced-comp-lexity architectures were demonstrated: an architecture useful for applications that recover partially erased input patterns, and an architecture suitable for applications that the input patterns may also contain erroneous bits. In this paper, essential design variables and the effects of their variations are also evaluated for the proposed architectures. The exploration includes (i) the effect of the number of erased symbols on the error performance of the message retrieval process, that is the number of correctly retrieved messages to that of the total inputs, (ii) the effect of increasing or decreasing the number of clusters on the allocated hardware resources and the error performance, and (iii) the effect of the number of iterations on the convergence of the output of the associative memory for various erasure conditions. Furthermore, the scalability of the hardware architectures and their limitations are discussed. The proposed architectures introduce new decoding methods in hardware that eliminate the requirement to employ the winner-take-all modules in [5, 6, 10]. Therefore, they improve the design complexity by reducing the required resources and increasing the maximum operating frequency in an attempt to improve the scalability. In addition, the proposed architectures have a similar error performance to that of [10]. The algorithm is explained from a hardware design perspective in Section 3, and the hardware implementations are introduced in Section 5. In Section 4 the design variables and the effects of their variation on the error performance and hardware resources are studied for the proposed algorithm and architectures. Section 6 summarizes the results, and is followed by conclusions in Section 7.

2 Review of Hopfield Neural Networks

Hopfield Neural Networks are biologically-inspired state-of-the-art family of associative memories which are represented by n nodes (neurons) that are fully interconnected by integer-weighted links. Each neuron is the place-holder of a bit in an input data (message). A link between the i-th and the j-th neuron in the network ($1 \leq i, j \leq n$) has an integer weight, $\leq w_{(i,j)}$. The number of messages the network has stored is referred as M in this paper. Messages to be stored and to be retrieved are n bits in length, such that the index of each bit corresponds to that of a neuron in the network. A set of M messages are stored in the network by adding the links between the nodes to the network, and storing them in a storage device according to the following algorithm:

$$w_{i,j} \leftarrow \begin{cases} \sum_{m=1}^{M} v_i^m v_j^m, & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases}, \tag{1}$$

where v_i^m is the binary value of the i-th binary bit in a message m ($1 \leq m \leq M$). Therefore, the hardware implementation of the HNN must account for $M + 1$ distinct values for each link in a network including $n(n-1)/2$ different link values by which the capacity and the efficiency of the network can be computed (see [5]).

The message retrieval (decoding) process is performed in an iterative process given a partial and/or erroneous input pattern that updates the value of each neuron v_i by retrieving the trained links according to the following rule:

$$v_i^* \leftarrow \begin{cases} +1, & \text{if } \sum_{m=1}^{M} w_{i,j} v_j \geq 0 \\ -1, & \text{otherwise} \end{cases}, \tag{2}$$

where v_i^* is the updated value of the i-th neuron at the end of an iteration. Antipodal format of the input bits permits an erased value in the input pattern (erasure) to be presented as zero. In other words, an erasure cannot contribute in favour of or against the activation of a neuron (setting its value to +1 or -1 respectively). It can be observed that the number of nodes is equal to the number of bits in a message. Furthermore, the capacity of the network depends on the number of neurons. Therefore, the number of bits in a message is dominated by the capacity of the network. In [5], the authors showed that the efficiency of HNNs, that is the number of memory bits stored to the ones used, tends to zero as the number neurons is increased.

3 Sparse Clustered Networks

First introduced in [5], and similar to HNNs, sparse clustered networks belong to a class of associative memories that can be modelled graphically by showing the associations between related patterns by lines and nodes. As shown in Fig. 1 SCNs can be represented as a graph consisting of a set of computing nodes (neurons) and binary connection weights as opposed to their integer-valued counterpart as in HNNs. More precisely, the network consists of n binary neurons arranged into c equally-partitioned clusters. Each cluster is associated with a portion of a message to be learned or retrieved. The network is c-partite with respect to the clusters previously defined, which means that two neurons in a same cluster cannot be interconnected. A partial input pattern is presented and a sparse set of neurons are activated which represent the matching learned pattern. The set is then encoded to form the full output pattern, also called a *clique*.

3.1 Message Training

In order to store messages in the SCN (training), a message m of K bits is divided into equal portions of κ bits each resulting in the creation of $c = K/\kappa$ sub-messages. Each cluster in the network consists of $l = 2^\kappa$ binary neurons, where each neurons represents a possible sub-message. A function that maps the binary value of each input sub-message to its equivalent integer number between 1 and l is considered in this work. This integer value is the index of the neuron to be activated in each cluster. Once the neurons corresponding to an input message are determined in all of the clusters, the corresponding binary connections are added to the network, and stored in a memory unit. In other words, once all the connections have been determined for the message to learn, a neural clique is constructed as shown in Fig. 1.

In this paper $w_{(c_i,l_j)(c_{i'},l_{j'})}$ refers the binary interconnection value between the j-th neuron of the i-th cluster to the j'-th neuron of the i'-th cluster.

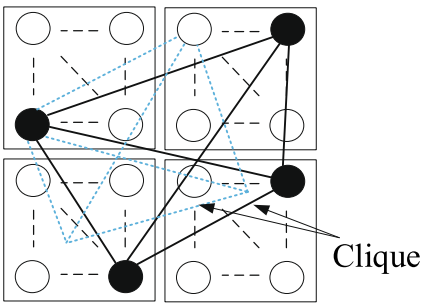


Figure 1 Graphical representation of neurons, clusters and cliques.

3.2 Message Retrieval

To retrieve a message, it is first determined which neurons should be activated given a partial or erroneous input pattern. Because the stored messages are matched against a partial input message, multiple neurons for a cluster might be activated. The interconnection weights between clusters are then used to allow the learned associations between portions of the learned message to affect which neurons should be activated. This process iterates until only one neuron per cluster is activated or the number of activated neurons is not changed. The active set of neurons is then encoded to form the output. The decoding process is performed in two stages: Local Decoding (LD) and Global Decoding (GD). Only global decoder is performed iteratively to retrieve the message.

3.2.1 Conventional Algorithm

Local Decoding In the conventional local decoding process [5, 6, 10], the indices of the neurons to be activated in each cluster are determined. Using the scalar product of a matrix g with each κ-bit sub-message I^i ($1 \leq i \leq c$), the j-th neuron of the i-th cluster is given a score, $s_{(i,j)}$:

$$s_{(i,j)} = \sum_{h=1}^{\kappa} g_{(j,h)}.(I^i)^T(h) \tag{3}$$

where $g_{(j,h)}$ denotes the h-th element of the j-th row of matrix g, and $(I^i)^T(h)$ denotes the h-th element of the transposed vector I^i. In this paper, g is an $l \times \kappa$ matrix containing ordered antipodal values representing integer numbers ranging from 1 to l. This ordering is consistent with the way a sub-message was mapped to a neuron in the training process.

The scoring process is then followed by finding the maximum value of the scores that would exceed a certain threshold value σ:

$$v^*_{(i,j)} \leftarrow \begin{cases} 1, & \text{if } s_{(i,j)} = s_{max} \\ & \text{and } s_{max} \geq \sigma \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where the value of σ depends on the number of erased bits in a sub-message. This step is referred to as winner-take-all (WTA) rule in the theory of neural networks where only the neuron(s) with maximum score(s) (s_{max}) are activated.

Due to the possible presence of erasures in an input message, it is possible that after local decoding process more than one neuron is activated in each cluster causing ambiguities in constructing a unique clique. These ambiguities are the attempted to be *disambiguated* in the global decoding process.

Global Decoding Following local decoding, global decoding is performed using the stored links and the activated neurons in the local decoding process. Similar to the local decoding process, each neuron is given a score but in a different computational procedure. In order to compute the score of the j-th neuron of the i-th cluster ($s_{(i,j)}$) in global decoding, the link values of all other neurons in other clusters to the neuron being globally decoded are first multiplied by the binary values of their attached neuron in other clusters. These values along with a scaled value of the neuron being globally decoded are then added together to compute the score:

$$\forall i, j, s_{(i,j)} = \sum_{i'=1}^{c} \sum_{j'=1}^{l} w_{(i',j')(i,j)} v_{(i',j')} + \gamma v_{(i,j)}. \tag{5}$$

The addition of the scaled value of the neuron being globally decoded is referred to as *the memory effect* [5], and is to consider the value of the local decoding process. The scale factor, γ is necessary to achieve good performance by enabling a memory effect such that a learned message is distinguished from the non-learned ones. In this paper $\gamma = 1$ is considered for simplicity in the hardware implementation. Once all the scores are computed, the WTA rule is then applied using Eq. (4) similar to the last step of the local decoding process. The process of computing scores and applying the WTA rule continues in an iterative process using Eqs. (5) and (4) respectively until either the values converge, i.e. more iterations are ineffective in changing the retrieved bits, or the maximum limit of the number of iterations is reached.

3.2.2 Proposed Local Decoding

It is possible to simplify the functioning of the conventional local decoding process [5, 6, 10] by considering the fact that the maximum possible value for a neuron in the local decoding process is directly dependent on the number of erased bits per cluster. Therefore, it can be derived to be equal to $\kappa - n_e$ where n_e is the number of erased bits. This simplification will eliminate the need to apply the WTA rule after finding scalar product of g and I as follows:

$$v_{(i,j)} \leftarrow \begin{cases} 1, & \text{if } s_{(i,j)} = \kappa - n_e \\ 0, & \text{otherwise} \end{cases}. \tag{6}$$

3.2.3 Proposed Global Decoding

It is possible to reduce the computational complexity of the conventional global decoding using two methods:

Method I As mentioned earlier, the local decoding process process can result in more than one activated neuron in the clusters with erasures or erroneous values. It can be shown that the score of a neuron being globally decoded can be determined to be (or remain) activated only if it receives at least one signal from every other cluster than itself. In other words, the ambiguities of other clusters will not have an effect on the value of the neuron being computed in global decoding. In [6], the scoring process for the new discovery is expressed as:

$$s(i, j) = \sum_{i'=1}^{c} \max_{1 \leq j' \leq l} \left(w_{(i',j')(i,j)} v(i', j') \right) + \gamma v(i, j). \tag{7}$$

Since the maximum value of $w_{(i',j')(i,j)} v(i', j')$ is equal to one, the decoding process can be expressed as:

$$v^*(i, j) \leftarrow \begin{cases} 1, & \sum_{i'=1}^{c} \bigvee_{j'=1}^{l} \left(w_{(i',j')(i,j)} v(i', j') \right) + \\ & \gamma v(i, j) \geq \sigma \\ 0, & \text{otherwise} \end{cases}, \tag{8}$$

where $\bigvee_{j'=1}^{l}$ performs an l-input logical OR operation, and σ is a threshold value that can be adjusted to fine-tune the error rate in case the input messages are erroneous instead of containing erased bits. If $\sigma = \gamma + c - 1$, this method can only be useful for retrieving inputs containing erased bits and not for the erroneous ones.

As an example, let us assume a network comprised of three clusters ($c = 3$), two neurons each ($l = 2$), $\gamma = 1$, and a single learned message "010". An erroneous input message "110" will activate a false neuron in the first cluster. Therefore, if $\sigma = c$, all neurons will switch off after the first iteration and therefore the network will not retrieve the message. On the other hand, if $\sigma = c - 1$, the network will be able to retrieve the correct message.

This algorithm is the basis of the first proposed hardware implementation (Architecture I) explained later in this paper.

Method II In scenarios where the input messages are not erroneous and only contain erased bits, a neuron is activated when its score is equal to $\gamma + c - 1$ during the global decoding. The global decoding process can then be expressed as:

$$v^*(i,j) \leftarrow \begin{cases} 1, \text{ if } \sum_{i'=1}^{c} \bigvee_{j'=1}^{l} \left(w_{(i',j')(i,j)} v(i',j')\right) \\ \qquad +\gamma v(i,j) = \gamma + c - 1 \\ 0, \qquad \text{otherwise} \end{cases} . \quad (9)$$

Method I (8) can therefore be altered to suit digital circuit implementation by removing the sum and using logic symbols as shown in:

$$v^*(i,j) \leftarrow \begin{cases} 1, \text{ if } \bigwedge_{i'=1}^{c} \bigvee_{j'=1}^{l} \left(w_{(i',j')(i,j)} v(i',j')\right) \\ \qquad \bigwedge v(i,j) \\ 0, \qquad \text{otherwise} \end{cases} , \quad (10)$$

where $\bigwedge_{i'=1}^{c}$ performs a c-input logical AND operation. This method can only be used for a case when bit erasures exist, and will not work for erroneous input corrections. This algorithm is the basis of Architecture II explained later in this paper. Equations (9) and (10) are equivalent when $\gamma \neq 0$.

This representation of the global decoding process is the basis of the second proposed hardware implementation (Architecture II). Software simulation results for confirming the validity of Eq. (9) are depicted in Fig. 2 comparing the error performance of the conventional global decoding algorithm with the proposed methods. Since the decoding algorithm of Method I and Method II is the same for erasure-only scenarios, the error performance of both methods are also equivalent.

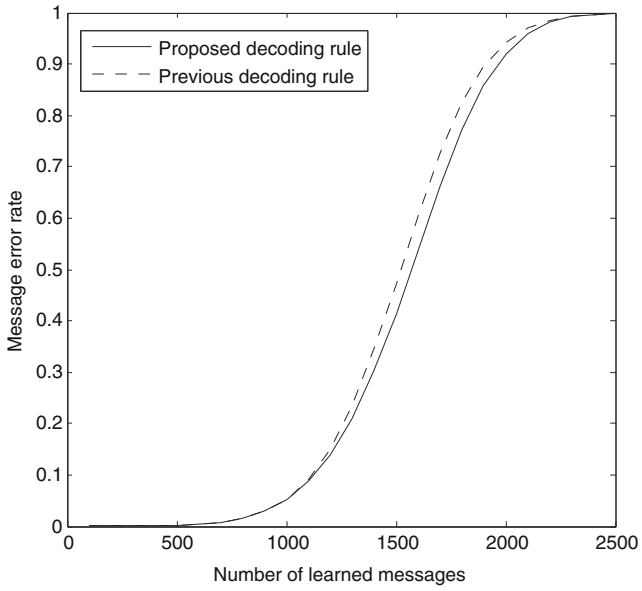


Figure 2 Comparison between message error rate of the previous [10] and the proposed methods ($l = 54$, $n = 432$, $it = 4$, $ce = 50$ %).

4 Design Space Exploration

There are a few choices one can make in designing an SCN-based associative memory, and performing the message retrieval process. The effect of the message density, and the number of iterations on the error performance given a fixed number of erased clusters (ce in percentage) has been studied in [5]. The effect of the number of bit erasures on the error performance was discussed in [10]. In this section we investigate the impact of a few other choices focusing on scenarios, where all of the bits for one or more sub-messages are erased rather than individual bits.

4.1 Number of Clusters

The number of clusters in an SCN-based associative memory determines the flexibility of the number of possible different locations of erasures. For example, if a K-bit message is used in an SCN network with c_1 or c_2 clusters ($c_1 < c_2$), the network with c_2 clusters will permit random erasures in a smaller range of lengths and is thus more flexible on the position of erasures. In Fig. 3, it is shown that given a fixed number of neurons and a fixed percentage of erasures in the message bits, increasing the number of clusters will also increase the error rate. Due to the reduction of the number of neurons per cluster, fewer connections are available that given the same number of stored messages, as in a case with smaller number of neurons per cluster, results in an increased density and thus an increased error rate.

4.2 Number of Erased Clusters

Another important factor in the error performance is the number of erasures in the input message (erasure rate). If we assume uniform random distributions of inputs, a particular position of the erasures does not have any advantages or disadvantages on the error performance. In Fig. 4, the impact of various erasure conditions on various network parameters is shown for a fixed number of neurons in total and different number of clusters. The first observation is that the difference of the effect on the error performance for 25 % compared to 50 % erasures is more dramatic for a network with 4 clusters compared to another network with 8

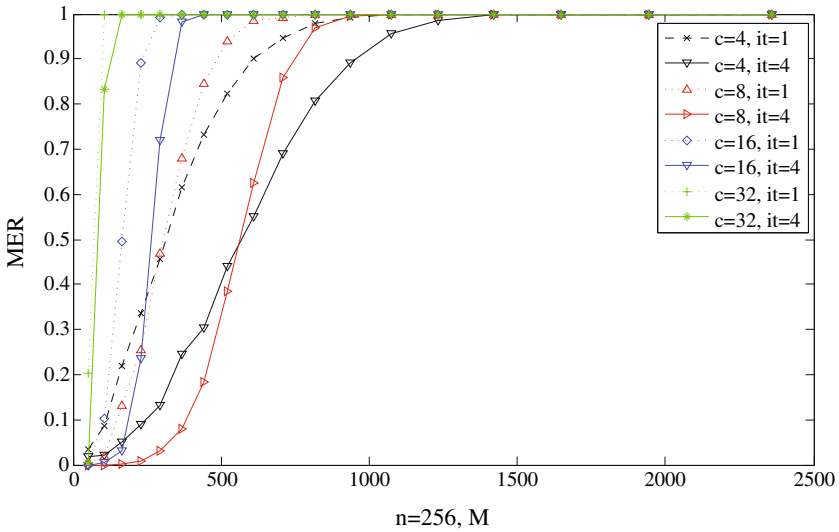


Figure 3 The effect of increasing the number of clusters given a fixed number of neurons and the number erased clusters on the message error rate (ce = 50 %).

clusters regardless of the number of iterations. The second observation is that for a network with 4 clusters, only a single iteration is sufficient to achieve convergence given inputs with 25 % erased clusters. However, increasing the number of erased clusters will require more iterations to achieve a converged output.

4.3 Iterations

After local decoding, global decoding is performed iteratively to achieve convergence or a desired error level. As more neurons are disambiguated in each iteration, the deactivated neurons cause further reduction of ambiguities in the next iteration by resulting in 0's in the outputs of the AND operations of Eq. (9). Figure 5 depicts the effect of the number of iterations on the error performance for various network parameters given a fixed number of neurons in all networks. It can be observed that as the network converges after the second iteration for a network with 4 clusters, whereas for a network with 8 clusters, larger number of iterations are required to reach convergence. Therefore, iterations have a higher effect on convergence for networks with larger densities.

5 Proposed Hardware Implementation

In designing the hardware architecture of SCN, algorithmic and design parameters are selected according to the input characteristics (length, erasure locations, erroneous and/or erasure bits), available hardware resources, speed, and performance requirements. In Section 4, we discussed how different parameters influence the error performance. In this section, we investigate the explored parameters in hardware implementation and suggest suitable architectures accordingly. Considering the evaluation results in Section 4

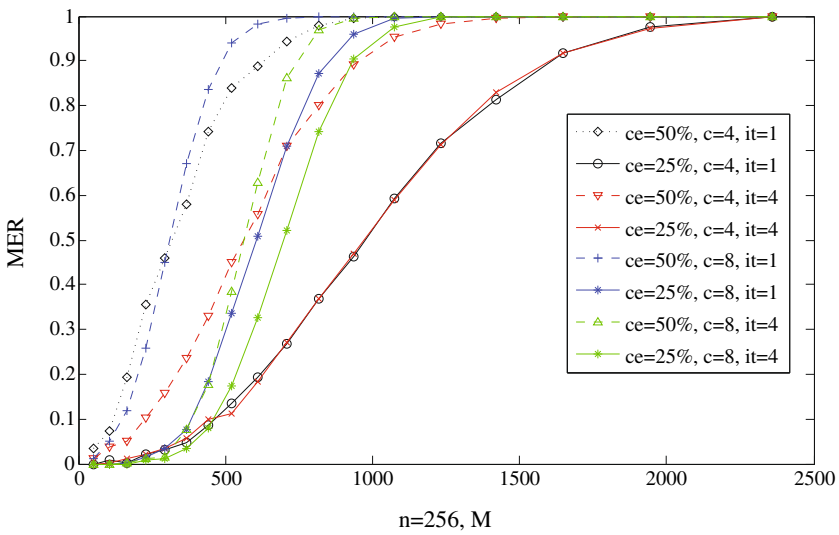


Figure 4 The effect of increasing the number of erased clusters on the message error rate for different sizes of clusters and given a fixed number of neurons.

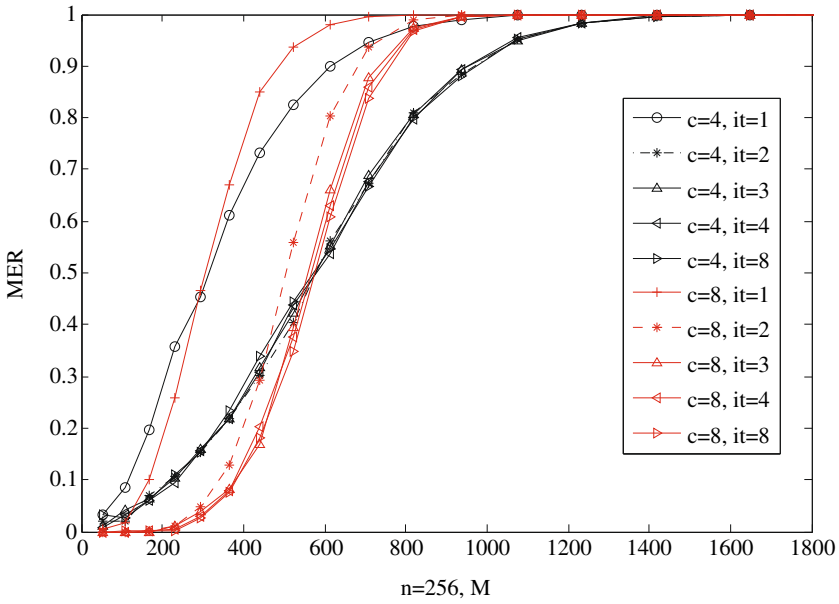


Figure 5 The effect of increasing the number of iterations for two different number of clusters ($ce = 50\ \%$).

to develop an initial intelligent guess for the parameters, the designer needs to iteratively evaluate the parameters of the design back-and-forth using Section 4 and this section to meet a specific hardware and algorithmic requirement.

The number of nodes, clusters, iterations are thus to be selected first by using software simulations according to the required diversity, message lengths, erasure locations. Once an initial intelligent guesses are made, the designer can tune the design parameters after running software simulations according to Section 4 to meet the error performance requirements. Then, the desired hardware solution can be selected by reviewing this section. If the parameters that were initially selected are too resource-hungry or slow, the designer must go back to Section 4 and update the parameters (eg. increase the number clusters without increasing the total number of nodes).

Two architectures are proposed: (i) The first architecture (Architecture I) incorporates the proposed local decoding in Section 3.2.2, with the proposed global decoding in Method I (8), (ii) The second architecture (Architecture II) incorporates the proposed local decoding process with the proposed global decoding in Method II (10).

5.1 Design Hierarchy

A top-level system diagram of the proposed hardware architectures is depicted in Fig. 6. There are two types of possible input messages: training or retrieving. The input messages contain antipodal values (-1 or $+1$) or zeros to indicate erasures. Therefore, two bits are required to represent these values: $00 \rightarrow$ Erased, $01 \rightarrow +1$, and $10 \rightarrow -1$.

The difference between this design hierarchy and that reported in [10] is in the way both the local and the global decoders are implemented. A threshold generator module is designed to generate the required threshold values for local decoder based on the number of erased bits as shown

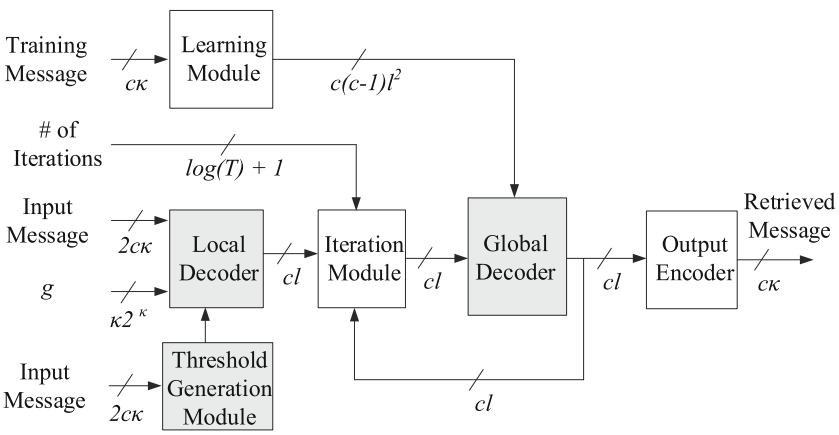


Figure 6 Simplified block diagram of the top level hierarchy with shaded regions to show the improved modules compared to [10].

in Section 3.2.2. The global decoder module has also been re-designed in two possible ways to implement the proposed global decoding algorithms discussed in Section 3.2.3.

Once the network is trained with the messages, the message retrieval process can be initiated by presenting partial input patterns to the network. Local decoding and the iterative global decoding processes are then performed to generate the output results. An output encoder module is dedicated to convert the l-bit values of the neurons in each cluster to κ bits. If at the end of the last iteration more than one neuron is activated, a sequential encoder can be implemented to sequentially output the values. However, in this paper the latter scenario is not implemented i.e. it is assumed that the last iteration will produce a single activated neuron in each cluster.

5.2 Architecture of the Training Module

The training of the network is achieved by storing the binary links in c two-dimensional register arrays of size $(l \times (c-1)l)$ bits as shown in Fig. 7. The links associated with the messages are sequentially stored to the FF array in a similar way to Static Random Access Memories (SRAMs), with the difference that in SRAMs, simultaneous access to data for different addresses is not possible. A parallel array of OR gates are employed to merge the values of the links for different messages in the register array by first reading the appropriate values of the link values and performing the logical OR on them with the incoming inputs. This array of OR gates is necessary since when consecutive messages are trained into the network, one can overwrite the links associated to that of the previous ones otherwise. A multiplexer (OR Mux)is employed that selects which row of the register array must be OR-ed with the new input depending on the values of the inputs. The row and column decoders are simply one-hot decoders that convert each κ bits of the message into 2^{κ} bits.

The registers are accessed independently to permit parallel computation of the values of the neurons in global decoding. Since a neuron in each cluster is connected to all the neurons in other clusters than itself, and that each connection requires one bit of storage device, the number of memory bits required to store these connection is given by $c(c-1)l^2$.

5.3 Proposed Architecture for Local Decoding

The proposed architecture of the local decoder is based on the algorithm presented by Eq. (6) and depends on the number of erased bits and the total number of bits in a message . The local decoding process is initiated by first determining the threshold value from which, if a neuron's score exceeds,

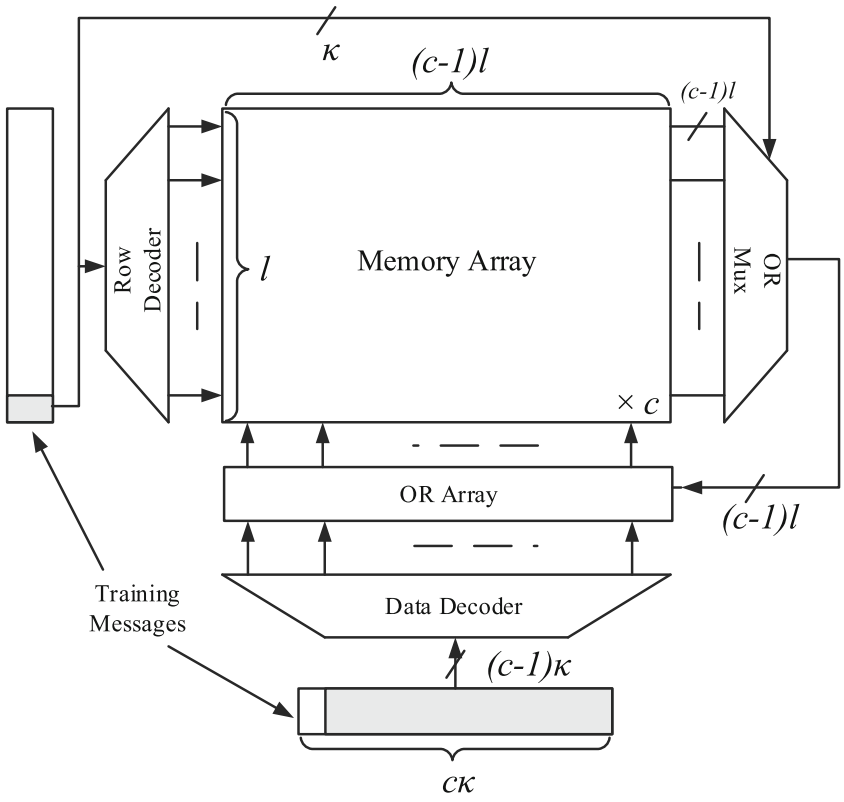


Figure 7 Architecture of the training module consisting of an array of parallel independently-accessed flip flops storing the binary links between neurons in different clusters.

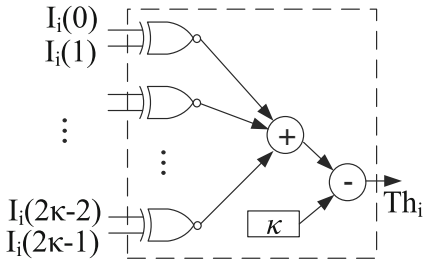


Figure 8 Architecture of the threshold generator used by the local decoder.

it will be activated. This threshold value is generated using the threshold generator module shown in Fig. 8. Since each bit in a sub-message is replaced by two bits to allow representation of an erased bit, the XNOR gates evaluate the two-bit equivalent of each bit in a sub-message to determine if it is erased. An adder simply adds the number of erasures such that $\kappa - n_e$ can be calculated afterwards using a simple subtractor. The output of the threshold generator can vary depending on the number of erased bits in a cluster.

The generated threshold values are used as inputs into the local decoder module shown in Fig. 9. The local decoder module activates the neuron(s) in each cluster and no longer requires resource-hungry WTA modules as in [10]. In the architecture of each local decoder module for each neuron, two sets of XOR gates along with a κ-bit adder are used to add the scalar product of a row in the g matrix with the negative and positive parts of each sub-message separately. Then these values are subtracted to calculate the final value of the scalar product. In the hardware architecture considering the g matrix, although the values are antipodal, a single bit is sufficient to represent a bit as no erasures can occur in the input messages. After calculating the score for each neuron, a comparator compares the score with the threshold value that is calculated in parallel using the threshold generator module. If the score is larger than or equal to the threshold, the corresponding neuron is activated accordingly.

5.4 Proposed Architectures for Global Decoding

The proposed global decoding architectures are depicted in Figs. 10, and 11. These architectures are referred to as global decoding Architecture I and Architecture II in this paper, and are based on Eqs. (8) and (10) respectively. A global decoding unit is implemented for each neuron. To elaborate on the behavior of these structures, let us assume that the value of the j-th neuron of cluster i is to be computed after the local decoding process has finished i.e. a preliminary decision has been made on which neurons should be activated. The global decoding unit updating the value of a neuron has three types of inputs: 1) The binary values of the links from all neurons excluding the one in the i-th cluster, 2) $v_{(i',j')}$, the binary values of the neurons in clusters excluding the i-th cluster, and 3) The binary value of the neuron being globally decoded ($v_{(i,j)}$) to account for the memory effect. In the hardware implementation, the memory effect coefficient, γ, is considered to be equal to 1 (also discussed in [5, 10]), which will simplify the hardware implementation.

The basic operation of the global decoding architecture I is similar to the previous architecture in [10], in a sense that decoding the value of a target neuron in a cluster is achieved by first computing the binary multiplication of the values of the binary connections to the target neuron, obtained from the memory, and the neural values attached to the neurons in the adjacent clusters. These multiplications are performed using two-input AND gates operating in parallel. The existence of such parallel computations in hardware

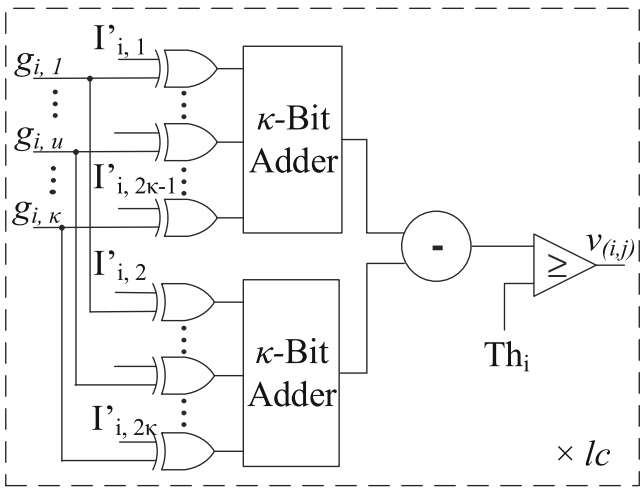


Figure 9 Architecture of the Local Decoder.

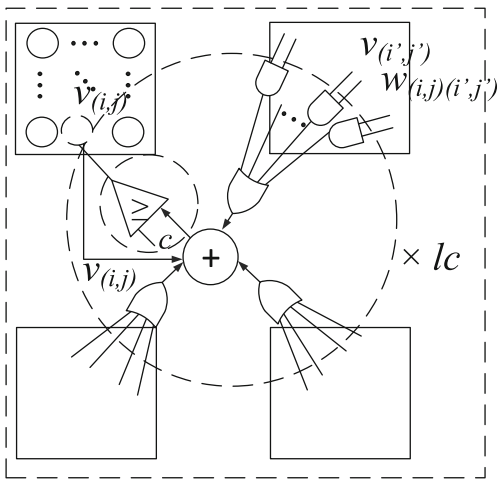


Figure 10 Hardware implementation of the proposed Global Decoder Architecture I.

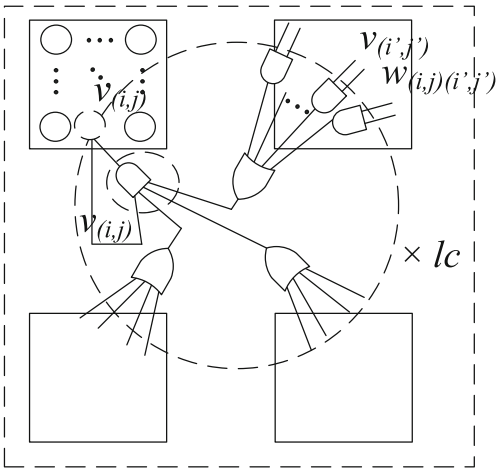


Figure 11 Hardware implementation of the proposed Global Decoder Architecture II.

is an advantage over a similar implementation in software. However, unlike the previous method for finding the maximum value using the compare-and-select method, the binary multiplication outputs are used as inputs to l-input OR gates which generate only one signal per adjacent cluster for each neuron being globally decoded. Then, the output from the OR gates, as well as the neural value obtained from the local decoder are added together and compared against a threshold value. This threshold value is at most equal to c and depends on the number of erasures and erroneous bits in each cluster. The target neuron will remain activated if the comparator outputs a '1'. If it is possible in an application to pre-determine that the inputs can never be erroneous and can only contain erased bits, the adder adding the ORed signals from each cluster can be removed and replaced by an c-input AND gate. This AND gate will result in a fixed threshold. This condition is presented in Architecture II and as shown in Fig. 11.

6 Circuit Evaluation

The proposed architectures have been implemented using an Altera Stratix IV (EP4SGX230KF40C2) Field Programmable Gate Array (FPGA), and were implemented using the same network parameters (128 neurons, 8 clusters) as in [10], as well as other variations of the network parameters (256 neurons, 8 clusters and 4 clusters) to investigate the scaling behaviour. Furthermore, the maximum size of the network the FPGA could fit was also implemented consisting of 432 neurons and 8 clusters. The maximum achievable number of neurons would not necessarily permit a larger number of input lengths compared to that of a 256-neuron network. However, it can have its own applications that can directly map the symbols to the neurons without a requirement for conversion of κ-bit inputs to l bit outputs. After verification of the results using a similar method described in [10], the hardware complexities, and the performances were compared with that of the previous work.

These implementations demonstrate how the ratio of Logic-to-Memory ratio (LMR) is affected by in the proposed architecture as the number of neurons and clusters are scaled. As the number of clusters is doubled, LMR is slightly increased by 8 % for $n = 128$ and $n = 256$, whereas for the largest network the FPGA can fit, it is increased by 18 %. However, increasing the number of neurons for a fixed number of clusters results in a relatively constant value for the LMR.

The evaluation results of the circuit implementation are summarized in Table 1. Figure 12 depicts the FPGA results of error performance for various densities. Each point obtained from the FPGA is associated with the average MER resulted by applying the same number of test vectors as the number of learned messages for 20 different random networks. It also shows software simulation results with similar parameters to verify the correct functional behavior or the hardware implementation. Since in the hardware implementations the number of neurons in the clusters are equal, i.e., the sub-messages have equal lengths, the message densities are calculated according to:

$$d = 1 - \left(1 - \frac{1}{l^2}\right)^M, \tag{11}$$

where M is the number of learned messages.

Table 1 FPGA resource allocation comparing SCN architectures with design parameters reported in [10].

	Previous work [10]	Architecture I	Architecture II
Memory usage dedicated to w (bits)	14,336	14,336	14,336
Registers	15, 783/182, 400(9 %)	15, 048/182, 400(8 %)	15, 035/182, 400(8 %)
Combinational Look-up Tables (LUT)	35, 224/182, 400(19 %)	13, 244/182, 400(7 %)	12, 341/182, 400(7 %)
Total pins	169/888(19 %)	169/888(19 %)	169/888(19 %)
Slow 900 mV 85C maximum frequency (MHz)	107.15	203.78	205.21
Training, Retrieving delay (per message)	10 ns, 50 ns	5 ns, 25 ns	5 ns, 25 ns

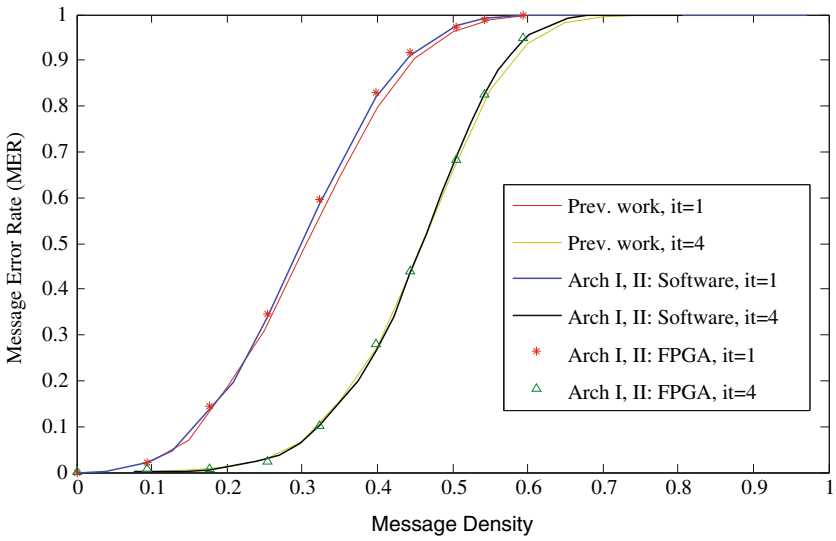


Figure 12 Comparison between the average MER of the proposed architectures and that of [10] using FPGAs, and software simulation ($n = 128$, $c = 8$, Number of erased clusters (random position): 4, FPGA number of networks per density: 20).

Implementing a network with $n_1 = 128$ neurons and 8 clusters, Architectures I and II achieve 62.4 and 65.0 % fewer LUTs respectively compared to the previous work [10], while the number of registers are similar. With a larger network comprised of $n_2 = 432$ neurons and 8 clusters, the previous architecture, unlike the proposed ones in this paper, can no longer fit within the same area as shown in Table 2. This comparison shows that growth of the number of neurons is larger than the growth of the number of LUTs. The maximum clock frequency in the FPGA has also been improved by ≈ 1.9 times resulting in lower computational delay.

Table 2 FPGA resource allocation comparing two networks ($n = 128$) and ($n = 432$).

	$n = 128$	$n = 432$
Registers (Prev. [10])	15,783	160,745
Registers (Proposed Arch. II)	15,035	161,331
LUT (Prev. [10])	35,224	359,127
LUT (Proposed Arch. II)	12,341	147,808

Table 3 FPGA resource allocation comparing three Architecture II-based networks with different number of clusters.

	Parameter	$n = 128$	$n = 256$	$n = 432$
c = 4	Registers	13,420	51,230	143,720
	LUT	10,170	38,520	111,410
	LMR	0.758	0.752	0.775
c = 8	Registers	15,035	58,672	161,331
	LUT	12,341	47,849	147,808
	LMR	0.821	0.816	0.916

Table 3 demonstrates the effect of variations in the number of clusters on the FPGA resources given a fixed number of neurons in the network. It also shows how the resources grow with the increase of the number of neurons given two choices for the number of clusters. As the number of connections in this network has a quadratic relationship with the number of neurons, the number of registers also follows a similar trend. Furthermore, since there exists at least one 2-input AND gate per register, and that the number of AND gates dominates the hardware resources, the number of LUTs also demonstrates a quadratic relationship with the number of neurons. It is also interesting to note that the number of different messages that can be stored in the network also has a quadratic relationship with the number of neurons [5].

7 Conclusion

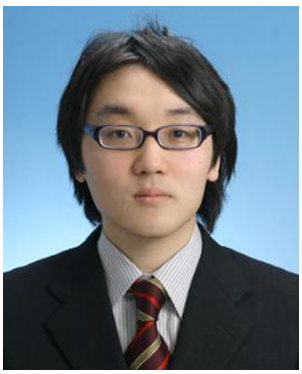
In this paper, we proposed the algorithm and hardware architecture of fully-parallel associative memories based on sparse clustered networks. The proposed architectures are suitable to either recover partially erased input patterns, or to correct erroneous input bits respectively. The proposed architectures can be used in a variety of applications such as search engines, data mining, and implementation of sets. The proposed architectures introduce data reconstruction (decoding) methods that reduce the hardware complexity by consuming 62.4 and 65 % fewer lookup tables respectively compared to those of the previous work. Furthermore, the proposed architectures increase the operating frequency approximately 1.9 times compared to that of previous work while having a similar error performance. We introduced novel hardware techniques that eliminated the resource-hungry Winner-Take-All circuits in the previous

architecture, and replaced it with simpler logic without sacrificing the error performance or the speed. The results were compared with that of previous work and their functions were verified using a similar verification strategy in the previous work. Additionally, we investigated the effect of varying the design variables to meet the required error performance and memory capacity. The design variables include the number of network nodes and clusters, the number of data entries, severity of erasures, and the effectiveness of iterations. The impact of such choices on the hardware resources were also discussed.

References

1. Agarwal, A., Hsu, S., Mathew, S., Anders, M., Kaul, H., Sheikh, F., Krishnamurthy, R. (2011). A 128x128b high-speed wide-AND match-line content addressable memory in 32nm CMOS. In *2011 Proceedings of the ESSCIRC (ESSCIRC)* (pp. 83–86). doi:10.1109/ESSCIRC.2011.6044920.
2. Chang, Y.J., & Lan, M.F. (2007). Two new techniques integrated for energy-efficient TLB design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *15*(1), 13–23. doi:10.1109/TVLSI.2006.887813.
3. Chao, H. (2002). Next generation routers. *Proceedings of the IEEE*, *90*(9), 1518–1558. doi:10.1109/JPROC.2002.802001.
4. Clark, L., & Chaudhary, V. (2010). Fast low power translation look aside buffers using hierarchical NAND match lines. In *Proceedings of 2010 IEEE international symposium on circuits and systems (ISCAS)* (pp. 3493–3496). doi:10.1109/ISCAS.2010.5537832.
5. Gripon, V., & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, *22*(7), 1087–1096. doi:10.1109/TNN.2011.2146789.
6. Gripon, V., & Berrou, C. (2012). Nearly-optimal associative memories based on distributed constant weight codes. In *Information theory and applications workshop (ITA)*, 2012 (pp. 269–273). doi:10.1109/ITA.2012.6181790.
7. Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, *79*(8), 2554–2558.
8. Huang, N.F., Chen, W.E., Luo, J.Y., Chen, J.M. (2001). Design of multi-field IPv6 packet classifiers using ternary CAMs. In *Global telecommunications conference (GLOBECOM)* (Vol. 3, pp. 1877–1881). doi:10.1109/GLOCOM.2001.965900.
9. Jarollahi, H., Gripon, V., Onizawa, N., Gross, W.J. (2013). A low-power content-addressable memory based on clustered-sparse networks. In *Proceedings of the 24th IEEE international conference on application-specific systems, architectures and processors (ASAP)* (pp. 305–308). doi:10.1109/ASAP.2013.6567594.
10. Jarollahi, H., Onizawa, N., Gripon, V., Gross, W.J. (2012). Architecture and implementation of an associative memory using sparse clustered networks. In *IEEE international symposium on circuits and systems (ISCAS)* (pp. 2901–2904). Seoul, Korea. doi:10.1109/ISCAS.2012.6271922.
11. Jarollahi, H., Onizawa, N., Gripon, V., Gross, W.J. (2013). Reduced-complexity binary-weight-coded associative memories. In *Proceedings of the 2013 IEEE international conference on acoustics, speech, and signal processing (ICASSP)* (pp. 2523–2527). doi:10.1109/ICASSP.2013.6638110.
12. Knoblauch, A. (2010). Optimal synaptic learning in non-linear associative memory. In *The 2010 international joint conference on neural networks (IJCNN)* (pp. 1–7). doi:10.1109/IJCNN.2010.5596604.
13. Komoto, E., Homma, T., Nakamura, T. (1993). A high-speed and compact-size JPEG Huffman decoder using CAM. In *Symposium on vlsi circuits, digest of technical papers* (pp. 37–38). doi:10.1109/VLSIC.1993.920528.
14. Pagiamtzis, K., & Sheikholeslami, A. (2006). Content-addressable memory (CAM) circuits and architectures: a tutorial and survey. *IEEE Journal of Solid-State Circuits*, *41*(3), 712–727. doi:10.1109/JSSC.2005.864128.
15. Qadir, O., Liu, J., Timmis, J., Tempesti, G., Tyrrell, A. (2011). Hardware architecture for a bidirectional hetero-associative protein processing associative memory. In *2011 IEEE congress on evolutionary computation (CEC)* (pp. 208–215). doi:10.1109/CEC.2011.5949620.
16. Tutanescu, I., Anton, C., Ionescu, L., Serban, G., Mazare, A. (2012). Hybrid error detecting and correcting system using hardware associative memories. In *Communications and information systems conference (MCC), 2012 Military* (pp. 1–5).

Hooman Jarollahi received the B.A.Sc. and M.A.Sc. degrees in Electronics Engineering from Simon Fraser University, Burnaby, BC, Canada in 2008 and 2010 respectively. He is currently pursuing a Ph.D. degree at the Department of Electrical and Computer Engineering of McGill University, Montreal, QC, Canada. His research interests are design and hardware implementation of energy-efficient and application-specific VLSI systems, such as associative memories and content-addressable memories. During 2012 and 2013, he was a visiting scholar at the Research Institute of Electrical Communication (RIEC) of Tohoku University, Sendai, Miyagi, Japan. He was the recipient of Teledyne DALSA award in 2010, for which he presented a patented architecture of a power and area-efficient SRAM. He is a Student Member of the IEEE.



Naoya Onizawa received the B.E., M.E. and D.E. degrees in Electrical and Communication Engineering from Tohoku University, Japan, in 2004, 2006 and 2009, respectively. He is currently an Assistant Professor in Frontier Research Institute for Interdisciplinary Sciences at Tohoku University, Japan. He was a postdoctoral fellow at Tohoku University from 2009 to 2011 and at University of Waterloo, Canada in 2011 and at McGill University, Canada from 2011 to 2013. His main interests and activities are in the energy-efficient VLSI design based on asynchronous circuits and multiple-valued circuits, and their applications, such as LDPC decoders, associative memories, and Network-on-Chips. He received the Best Paper Award in IEEE Computer Society Annual Symposium on VLSI in 2010. Dr. Onizawa is a Member of the IEEE.

Vincent Gripon is a permanent researcher with Tlcom Bretagne (Institut Mines-Télécom), Brest, France. He obtained his M.S. from École Normale Suprieure of Cachan and his Ph.D. from Télécom Bretagne. His research interests include information theory, neuroscience, and theoretical and applied computer science. His intent is to propose models of neural networks inspired by information theory principles, what could be called informational neurosciences. He is also the co-creator and organizer of an online programming contest named TaupIC which targets French top undergraduate students.

Warren J. Gross received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, Ontario, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, Ontario, Canada, in 1999 and 2003, respectively. Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada. His research interests are in the design and implementation of signal processing systems and custom computer architectures. Dr. Gross is currently Chair of the IEEE Signal Processing Society Technical Committee on Design and Implementation of Signal Processing Systems. He has served as Technical Program Co-Chair of the IEEE Workshop on Signal Processing Systems (SiPS 2012) and as Chair of the IEEE ICC 2012 Workshop on Emerging Data Storage Technologies. Dr. Gross served as Associate Editor for the IEEE Transactions on Signal Processing. He has served on the Program Committees of the IEEE Workshop on Signal Processing Systems, the IEEE Symposium on Field-Programmable Custom Computing Machines, the International Conference on Field-Programmable Logic and Applications and as the General Chair of the 6th Annual Analog Decoding Workshop. Dr. Gross is a Senior Member of the IEEE and a licensed Professional Engineer in the Province of Ontario.