An Analysis of Hardware Configurations for an Adaptive Weightless Neural Network

P. Lorrentz[1], W. G. J. Howells[2], and K. D. McDonald-Maier[3]

***Abstract* - This paper examines the potential offered by adaptive hardware configurations of a class of weightless neural architecture called the Enhanced Probabilistic Convergent Network targeted on a Virtex-II pro FPGA which is re configurable.**

The reconfiguration and adaptive capability of the Enhanced Probabilistic Convergent Network is a highly adaptive architecture offering a very fast, automated, uninterrupted responses in potentially electronically harsh and isolated conditions. The hardware architecture is tested on a benchmark of unconstrained handwritten numerals from the Centre of Excellence for Document Analysis and Recognition .

***Index Terms* – Adaptive Neural Network, FPGA, Handwritten characters, Reconfiguration.**

I. Introduction

Research into reconfiguration of artificial neural networks (ANN) is an increasingly significant area of investigation. This arises partly due to the improvement in performance possibilities offered in that it becomes possible for an ANN, when implemented in digital hardware, to be capable of adaptation and reconfiguration during learning [15]. This may be in response to nonlinear environment. However, adaptation and reconfiguration may incur a high computation overhead, more so in practical applications [2]. This high computation overhead is however minimised in the class of neural network investigated in this paper, the weightless neural network. This follows from the observation that the less the computation requirement, the faster an ANN is able respond to new input. This reduction in response time becomes very large when the ANN is implemented in hardware.

Artificial Neural Networks come into being as an attempt by humans to model the functionality of the brain and the central nervous system. This quest has been fruitful, and has yielded very many groups of ANN. Characteristic of these groups is the learning and recognition algorithm. The learning algorithm of an ANNs develops its configuration to allow for efficient subsequent recognition. Those that do not utilise weights during leaning and recognition are termed *weightless* ANN and those that do are called weighted NN. Depending on whether there is a “teacher” or not, learning may be called supervised learning (when there is a teacher) or unsupervised learning [3]. ANNs may also be grouped depending on whether the error feedback is employed or not. Those possessing error feedback are called recurrent networks, those that are not are called feed forward networks. ANNs may also be grouped depending on the principle behind their implementation. Those whose behaviour closely mimic the intelligence of natural being e.g. the genetic algorithm, and those designed from mathematical concepts. Weightless neural networks, also called RAM based neural networks [4], are a subgroup of those designed from mathematical concept, in this case mathematical logic concept.

Bledsoe and Browning in their pioneering work [4] (around 1959) made the first attempt to base their design of neural network on mathematical logic concept. More sophisticated networks has naturally been developed subsequently. These include the implementation of Enhanced probabilistic Convergent Networks (EPCN). The EPCN is an enhanced form of PCN [17]. The specific enhancements are as detailed in [14]. EPCN is a feed forward neural networks incorporating supervised learning with the addition that the mathematical logic is minimised even further when EPCN is implemented in a hardware.

A harware implementation of ANN offers significant advantages to a purely software implementation due to increased speed. For a weightless NN, the mathematical logic is of a reduced complexity than is the case with alternative NN when implemented in a digital intergrated circuit (IC) - this allows an increase in speed. These advantages, amongst others, motivate the work of this paper.

The aims and objectives of this paper are two fold. One is to present the architecture and implementation of an adaptive RAM-based neural network, the Enhanced Probabilistic Convergent Network (EPCN) [14], in a reconfigurable FPGA. The second is to explore the reconfiguration and adaptive properties of the FPGA-based neural network. The principle EPCN system implemented here is generic and highly scaleable. It does lend itself well to reconfiguration and adaptation. This enhances its capability for prediction and recognition. It is thus suited to a multitude of applications.

Due to the importance of hardware reconfiguration of ANNs, and its relative scarcity, some efforts has been dedicated to this area. Simoes [2] employs an ALTERA MAX + PLUS II on Eraseable Programming Logic Device (EPLD) to implement a weightless NN called the Goal Seeking neuron (GSN), in the classification of British mail

[1] Department of Electronics, University of Kent, Canterbury, Kent, CT2 7NT,UK. Tel: 07813089916 E-mail: plorrentz@gmail.com

[2] Department of Electronics, University of Kent, Canterbury, Kent, CT2 7NT, UK, Tel: +44 1227 764000 x 3389. Fax: +441227 456084, E-mail: W.G.Howells@kent.ac.uk

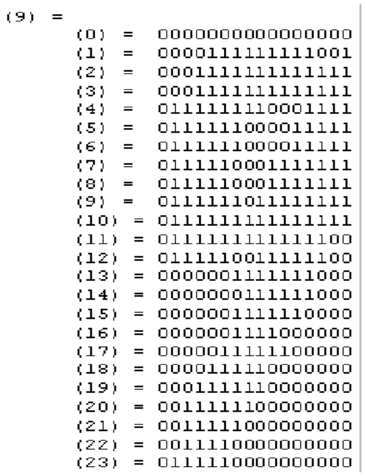
[3] Department of Computing and Electronic Systems, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK, e-mail: kdm@essex.ac.uk

postal addresses. Spaanenberg [9] shows, by employing the Virtex-II 6000 FPGA, learning of NN by reconfiguration. Botelho [16] implements Goal Seeking Neuron (GSN) on FPGA. THE GSN, a RAM based neural network, was deployed on Khepera mobile robot for control and navigation. RAM based neural networks in [9],[16], designed on FPGA were deployed in autonomous systems.

The remainder of this paper is organized as follows. Section II presents an overview of the EPCN, while section III introduces its hardware configurations. The experiments to test the configuration possibilities of FPGA based Hardware architecture of EPCN and its results are presented in section IV. The analysis of results obtained are presented in section V. The paper concludes with areas of further research and development in section VI.

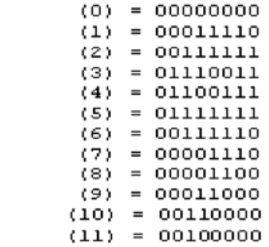
II. Overview of EPCN

The EPCN consist of two groups of layers with a single standalone layer subsequent to each group. The first group is called pre-group layer while the second group is called main-group layer. The single layer that succeeds the pre-group is its merge layer, while the single layer that succeeds the main-group is called main-group merge layer.



```
(9) =
        (0)  =  0000000000000000
        (1)  =  0000111111111001
        (2)  =  0001111111111111
        (3)  =  0001111111111111
        (4)  =  0111111110001111
        (5)  =  0111111000011111
        (6)  =  0111111000011111
        (7)  =  0111110001111111
        (8)  =  0111110001111111
        (9)  =  0111111011111111
        (10)  = 0111111111111111
        (11)  = 0111111111111100
        (12)  = 0111110011111100
        (13)  = 0000001111111000
        (14)  = 0000000111111000
        (15)  = 0000001111110000
        (16)  = 0000001111000000
        (17)  = 0000011111100000
        (18)  = 0000111110000000
        (19)  = 0001111110000000
        (20)  = 0011111100000000
        (21)  = 0011111000000000
        (22)  = 0011110000000000
        (23)  = 0111110000000000
```

Figure 1(a). An example of input data. This numeral "9" is part of CEDAR database used in this paper



```
(9) =
        (0)  = 00000000
        (1)  = 00011110
        (2)  = 00111111
        (3)  = 01110011
        (4)  = 01100111
        (5)  = 01111111
        (6)  = 00111110
        (7)  = 00001110
        (8)  = 00001100
        (9)  = 00011000
       (10)  = 00110000
       (11)  = 00100000
```

Figure 1(b). A compressed version of numeral "9" .The effect of compression algorithm employed is noticeable.

The EPCN typically operates on binary threshold images as shown in Figure 1(a).

Pre-processing will sometimes yield a reduced (in size) pattern, Figure 1(b), which will be employed in subsequent processes such as learning. Learning of EPCN is supervised, generally following the principle of forming addresses for the RAM neurons from within the input pattern and assigning class identifiers to the addresses so formed [14]. This corresponds to the Train-block in Figure 2.

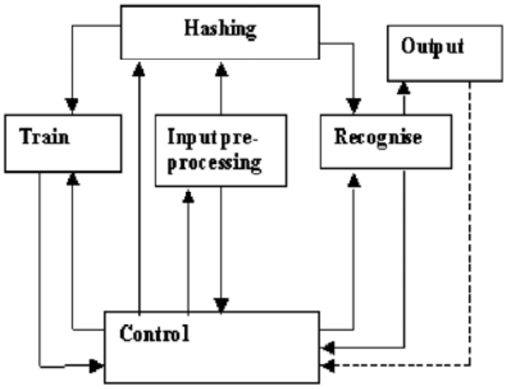


Figure 2. The block-diagram of EPCN FPGA architecture.

The recognition process of EPCN is mainly an averaging process which occurs in the main-group layer and is represented as recognise block in Figure 2. The learning and recognition algorithm of EPCN is identical to the learning and recognition algorithm of PCN [17]. For learning and recognition to proceed connectivity (addresses to RAM location) are formed from input space. This is done in FPGA by hashing (represented by the Hashing block in Figure 2). The output of the hashing block is a series of addresses, as shown in Figure 3, that represents RAM locations to be modified.

III. The Hardware Architecture

Various options were considered with respect to medium of hardware implementation. Considering that fabrication of a dedicated hardware EPCN does not lend itself easily to modification of structure and topology of EPCN. But certain advanced FPGA, e.g. Virtex-II pro, offers more possibilities for modification to the structure and topology of EPCN easily. An equivalent software implementation is as good and as fast as the CPU (and FPU). That is, it could be very slow depending on computational demands and exceptions handling.

The EPCN is an adaptive NN, and to maintain this characteristics, an FPGA which is re-configurable is required. The Virtex-II pro satisfy these conditions. The FPGA, Virtex-II pro requires the logic equations of EPCN, and this is described in VHDL using Xilinx ISE. By the employment of VHDL, the EPCN was described in a hierarchical system that consist of packages and modules. Of these modules, the ones relevant to re-configuration and adaptive behaviour of EPCN are the hashing module, and reconfiguration module.

A. Hashing

In [11] a hashing function based on bit-folding, XOR, and pseudo random number generation was developed.

In this paper, the hashing function used is derived from XOR, and Maximum-length Shift-register code. Maximum-length shift-register codes [5] generate a systematic code with desired output length;

$$n = 2^m - 1 \qquad (1)$$

where m is the information bit obtainable from input pattern. The code words are normally generated by m-stage digital shift register with feedback. The generation of the code words depend on parity polynomials h(p) given by;

$$h(p) = p^k + h_{k-1}p^{k-1} + \dots + h_0p^0 \qquad (2)$$

The MLSR is preferred to other alternatives because it helps to easily reproduce addresses formed. The hashing is used for address (connectivity) formation. When an address is hashed, the corresponding RAM location will be read from or written to as the case may be. Examples to illustrate this are given below.

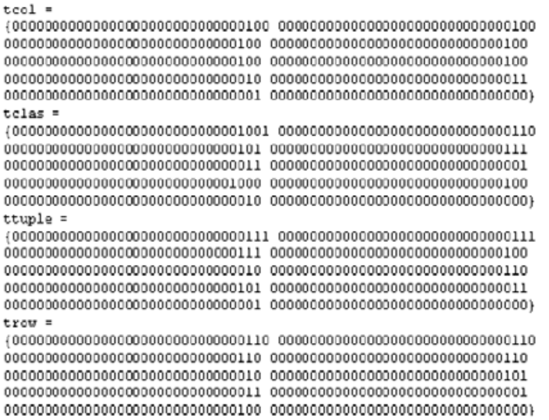
Example I: Suppose the information bit m = 2, and it is required to generate addresses in the range 0 to 3. Equation (1) becomes

$$n = 2^m - 1 = 2^2 - 1 = 3;$$

This means that 3 addresses are required. In equation (2) the parity polynomial h(p) becomes;

$$h(p) = p^3 + h_2p^2 + \dots + h_0p^0 \qquad (3);$$

In equation (3) it is seen that the coefficient of p^3 is 1. Then h_i, (i = 0,1,…3) is such that $h_{k-1}p^{k-1}$ is an integer between 3 and 0. This case is shown in Figure 3 , as values assigned to "tcol" variable, where tcol represents addresses of a layer. There it is seen that all values between 0 and 3 have been generated. To distinguish between wanted zeros and unwanted zeros, wherever h(p) has values greater than 3 or less than 0, this is set to 2^n and this make the location inaccessible.



```
tcol =
{00000000000000000000000000000100 00000000000000000000000000000100
00000000000000000000000000000100 00000000000000000000000000000100
00000000000000000000000000000100 00000000000000000000000000000100
00000000000000000000000000000010 00000000000000000000000000000011
00000000000000000000000000000001 00000000000000000000000000000000}
tclas =
{00000000000000000000000000001001 00000000000000000000000000000110
00000000000000000000000000000101 00000000000000000000000000000111
00000000000000000000000000000011 00000000000000000000000000000001
00000000000000000000000000001000 00000000000000000000000000000100
00000000000000000000000000000010 00000000000000000000000000000000}
ttuple =
{00000000000000000000000000000111 00000000000000000000000000000111
00000000000000000000000000000111 00000000000000000000000000000100
00000000000000000000000000000010 00000000000000000000000000000110
00000000000000000000000000000101 00000000000000000000000000000011
00000000000000000000000000000001 00000000000000000000000000000000}
trow =
{00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000010 00000000000000000000000000000101
00000000000000000000000000000011 00000000000000000000000000000001
00000000000000000000000000000100 00000000000000000000000000000000}
```

Figure 3. Formation of addresses by hashing from input patterns. This is prior to the learning process.

Example II: Suppose ten connectivity are required none of which should be greater than 10?.

Answer: Recall that $2^4 > 10 > 2^3$. So that when m = 4; equation (1) becomes

$$n = 2^m - 1 = 2^4 - 1 = 15;$$

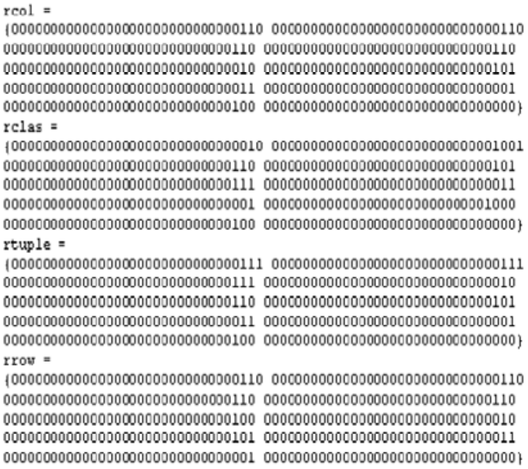
and in equation (2), the parity polynomial h(p) becomes;

$$h(p) = p^{15} + h_{14}p^{14} + \dots + h_0p^0 \qquad (4)$$

In equation (4) it is seen that the coefficient of p^{15} is 1. And h_i, (i = 0,1,…15) is such that $h_{k-1}p^{k-1}$ is an integer between 15 and 0. Since no connectivity should be greater than 10, h(p) is set to 2^n for those values that are not required. An example is shown in Figure 4, here the variable "rclas" shows all addresses derived lies between 0 and 10 inclusive. The "rclas" represents addresses of a neuron in the recognition phase.

To distinguish between wanted zeros and unwanted zeros, wherever h(p) has values greater than 10 or less than 0, this is set to 2^n and this make the location inaccessible.

Other addresses are derived similar to example I and II. Recall that information bits in a pattern characterise that pattern, and thus the connectivity is reproducible.



```
rcol =
{00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000010 00000000000000000000000000000101
00000000000000000000000000000011 00000000000000000000000000000001
00000000000000000000000000000100 00000000000000000000000000000000}
rclas =
{00000000000000000000000000000010 00000000000000000000000000001001
00000000000000000000000000000110 00000000000000000000000000000101
00000000000000000000000000000111 00000000000000000000000000000011
00000000000000000000000000000001 00000000000000000000000000001000
00000000000000000000000000000100 00000000000000000000000000000000}
rtuple =
{00000000000000000000000000000111 00000000000000000000000000000111
00000000000000000000000000000111 00000000000000000000000000000010
00000000000000000000000000000110 00000000000000000000000000000101
00000000000000000000000000000011 00000000000000000000000000000001
00000000000000000000000000000100 00000000000000000000000000000000}
rrow =
{00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000110 00000000000000000000000000000110
00000000000000000000000000000100 00000000000000000000000000000010
00000000000000000000000000000101 00000000000000000000000000000011
00000000000000000000000000000001 00000000000000000000000000000000}
```

Figure 4. Formation of addresses by hashing from input patterns. This is prior to the recognition process.

B. Reconfiguration

The structural architecture of EPCN and the size of its neuron is adaptive, changing with learning and classification. During learning and classification, an integer number called the *division* is required for *adjustment* purposes. [14] The term *adjustment* refers to multiplying the integer value in a RAM location by the *division* and dividing by the number of training patterns per class. The *adjustment* is necessary for all classes to be treated equivalently when the number of pattern per class varies between classes. *Tuple-size* is the number of bits sampled from input data (at once) that characterize features of that data. For a *tuple-size* of n, $2^n - 1$ bits are sampled.

In practice the maximum size and structure of EPCN is naturally limited by the available hardware resources. The number of *pre-group layers*, the number of *main-group layers*, the *tuple-size*, and the *division* are often referred to as system parameters.

The size of pre-group layer and the size of main-group layers are modifiable alongside the reconfiguration process.

The number *division* used during various *adjustment* phases could be chosen within a value from 1 to 2^{15} . This is the binary address range that fits in memory on FPGA. The possibility of the variability in system parameters are vital to static and dynamic reconfiguration. Modification to the value assigned to *division* is done by prefixing "constant divisn" with a "generic" statement. This is normally done before a training and a recognition session pair.

The EPCN reconfiguration file is stored in Programmable Read Only Memory (PROM). Since the golden configuration is stored in revision 0 for FPGA's self-test, the EPCN reconfiguration file is stored in revision 1. The source-select switch is used to select any of the revision at any time required.

The FPGA is pre-programmed with various possible configuration options. The config-select, SW8, is a group of three switches, the combination of which gives the selection of one of eight possible configurations of EPCN. The source-select, SW9, is a group of two switches, the combination of gives the selection of source of configuration for EPCN.

Using the config-select switches in conjunction with config-source switches it was found that it supports to a maximum of:

- Tuple-size = 4;
- Pre-group layers = 5;
- Main-group layers = 5;
- Class = 15;
- Number of neuron per layer 20-by-15;

The detection of pattern boundaries is automatically and dynamically done by the control unit (figure 2).

The functional activities of the pre-processing unit and the hashing function (unit) are monitored by the control unit to ensure that the size of the pattern used within the EPCN fall within the maximum neuron size possible. Secondly, it is always possibile to adjust every pattern size appropriately before hashing. This solves the boundary problems. The solution to the boundary problems increases the range and type of input sources and reconfiguration flexibility of EPCN, which will be experimented on in section IV.

IV. Experiments And Results

.

The experimentation carried out here explores various configurations of EPCN. The EPCN was designed and implemented using Xilinx ISE. It was then testes in software by simulations prior to these experiments. The Source of database used in these experiments is:-

- The centre of Excellence for Document Analysis and Recognition (CEDAR), University at Buffalo, State University of New York, USA. Department of Computer Science. Unconstrained handwritten numbers from CEDAR were resized and binarised to 16-by-24 in dimension.

The config-select switch consist of three switches while the source-select switch is made up of two switches. In any session, learning or recognition, a combination of the three switches on SW8 yields eight possible configurations which enables variation of configuration and system parameters of EPCN architecture. The config-source, consist two switches which are used to select sources of configuration. The various configurations of EPCN in these experiments resides in PROM (in Revision 1) and are fetched during reconfiguration automatically.

Preliminary investigations, that includes the available size of both the internal and external synchronous dynamic random access memory (SDRAM), has revealed that hardware resources supports maximum of 5 layers of pre-group and maximum of 5 layers of main-group. Guided by these hardware resource constraints, the experiment aims to explore various configuration possibility of EPCN and to determine the possible optimum configuration of EPCN. To this end, three experiments were performed on FPGA based EPCN using the database mentioned above. They are:-

- A case where division = 1000; main-group layers = 3; pre-group layer increase from 1 through to 5.
- A case where division = 1000; pre-group layer = 3; main-group layer increase from 1 through to 5.
- In the third experiment, the main-group layers = 3; pre-group layer = 3; division is increase from 100 through to 700.

Results of these experiments were recorded. They are graphically displayed in Figures 5,6, and 7.

These same experiments has been performed on the software version of EPCN [14], by employing the same CEDAR database.

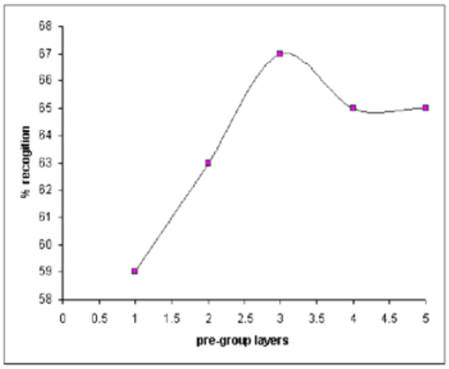


Figure 5: A plot of % recognition against number of pre-group layer; division = 1000; main-group layers = 3; pre-group layer increase from 1 through to 5.

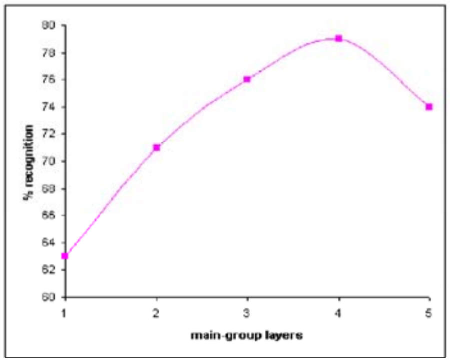


Figure 6: A plot of % recognition against number of main-group layer; division = 1000; pre-group layers = 3; main-group layer increase from 1 through to 5.

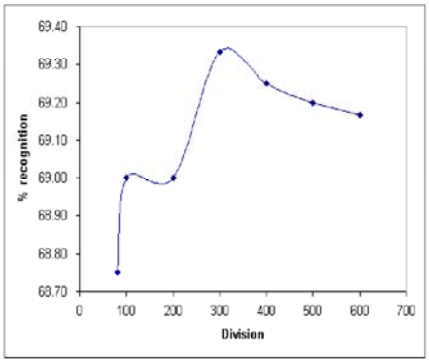


Figure 7: A plot of % recognition against *division*; the main-group layers = 3; pre-group layer = 3; division is increase from 100 through to 700.

V. Analysis

The advantages of the FPGA implementation are that it is able to exploit the reconfiguration and adaptive capability of the EPCN which is advantageous for many situations for which an intelligent machine requires very fast, automated, uninterrupted responses, and in potentially electronically harsh and isolated conditions.

Figure 5 shows that the maximum percentage recognition occurs when the pre-group layer is 3. Figure 6 shows that the maximum percentage recognition occurs when the main-group layer is 4. Figure 7 shows that the maximum percentage recognition occurs when the division is 300.

Comparing Figures 5, 6, and 7, it may be observed that the performance is least dependent on *division* and that performance is most dependent on *main-group layers*. These results are identical to the result obtained from the PC-based EPCN [14] when same input databases are used hence demonstrating the validity of the FPGA implementation. Further investigation and experimentation shows that the optimum system parameters are:-

Main-group layers = 4;
Pre-group layers = 3;
Division = 300;

These values are naturally dependent on the database employed and the number of classes.

Also it is noteworthy that the hardware is of the order of 10^5 faster than an equivalent software implementation. A comparison between the speed of the FPGA-based EPCN, an optically enhanced Multilayer percepteron (MLP) [6][7], and a software based EPCN is shown in Table 1.

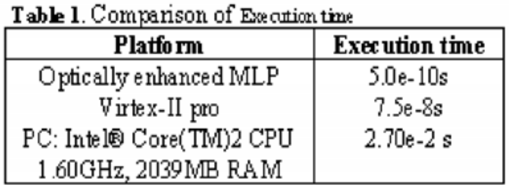


Table 1. Comparison of Execution time

Platform	Execution time
Optically enhanced MLP	5.0e-10s
Virtex-II pro	7.5e-8s
PC: Intel® Core(TM)2 CPU 1.60GHz, 2039MB RAM	2.70e-2 s

There is clearly a substantial gain in speed by the FPGA-based EPCN over a software implementation.

Different databases may give rise to different results (in Table 1). The results in Table 1 also depend on cofiguration complexity and on source of database employed. The order of magnitude appears more general and thus more reliable.

Hardware constraints has been considered, and compared to an equivalent software EPCN. These are tabulated in Table 2.

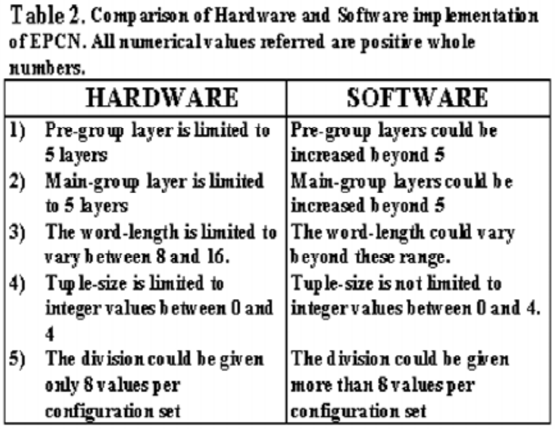


Table 2. Comparison of Hardware and Software implementation of EPCN. All numerical values referred are positive whole numbers.

HARDWARE	SOFTWARE
1) Pre-group layer is limited to 5 layers	Pre-group layers could be increased beyond 5
2) Main-group layer is limited to 5 layers	Main-group layers could be increased beyond 5
3) The word-length is limited to vary between 8 and 16.	The word-length could vary beyond these range.
4) Tuple-size is limited to integer values between 0 and 4	Tuple-size is not limited to integer values between 0 and 4.
5) The division could be given only 8 values per configuration set	The division could be given more than 8 values per configuration set

From Table 2 and Figures 5,6 and 7, it is deductable that the possibility of 16-bit word-lenght has a great effect on the identical result obtainable both from the hardware and the software EPCN.

VI. Conclusion

The FPGA based EPCN has been shown to be adaptive and reconfigurable. The results obtained here are comparable in performance terms to that of software-based EPCN.

A shortcomming of these experiments is that interaction effects of these parameters were not investigated. This may be considered as an area of further experimentation and development.

REFERENCES

[1] E.C. Chalfant, S. Lee: Measuring the Intelligence of Robotic Systems: An Engineering Perspective, Proc. Int. Symposium on Intelligent Systems, Gaithersburg MD, October 1999.

[2] E.V. Simões, L.F. Uebel, D.A.C. Barone: Hardware Implementation of RAM Neural Networks, Pattern Recognition Letters, no. 17, pp. 421 – 429, 1996.

[3] Haykins, S. (1994), *Neural Networks, A Comprehensive Foundation*, MacMillan Publishing, Englewood Cliffs, NJ., 1994.

[4] J. Austin (Ed.) :RAM-based neural networks, *World Scientific*, 1998.

[5] John G. Proakis: Digital Communication, 4th Edition, McGraw Hill Int. Edition, 2001.

[6] K. D. Maier, C. Beckstein, R. Blickhan, D. Fey, and W. Erhard: A multi-layer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits, in Proc. Of 6th Int. conf. On Parallel Interconnects, Anchorage, Alas. (IEEE Computer Society Press, Alamitos, Calif., 1999), pp. 73 – 80, 1999 .

[7] K. D. Maier, C. Beckstein, R. Blickhan, D. Fey, and W. Erhard: Standard cell-based implementation of a digital optoelectronic neural-network hardware, Applied Optics, Vol. 40, No. 8, 10 March 2001.

[8] L. Shang, Z. Yi, L. Ji: Binary Image Thinning Using Autowaves Generated by PCNN, Neural Processing Letters, 25:49 – 62, 2007.

[9] L. Spaanenburg, R. Alberts, C.H. Slump, B.J. VanderZwaag: Natural learning of neural networks by reconfiguration, pp. 273-284 (2003), in: Rodriguez-Vazquez, A., Abbott, D. and Carmona, R. (eds.), SPIE Int. Symp. On Microtechnologies for the new Millennium, Vol. 5119 (Maspalomas, Gran Canaria, Spain), 2003

[10] M. Fayyazi, Z. Navabi: Using VHDL Neural Network Models for Automatic Test Generation, 2nd Workshop on Libraries, Component Modeling and Quality Assurance, Toledo, Spain, April 1997.

[11] M. Freeman, J. Austin: Designing a binary neural network co-processor, Digital System Design, 2005. Proceedings. 8th Euromicro Conference on Volume, Issue, Page(s): 223 – 226, 30 Aug.-3 Sept. 2005

[12] M. Freeman, M. Weeks, J. Austin: AICP: Aura Intelligence co-processor for Binary Neural Networks, IP-SOC, Grenoble, France, 2004.

[13] P. Lorrentz, W. G. H. Howells, K. D. McDonald-Maier: Design and analysis of a novel weightless neural based Multi-classifier, World Congress on Engineering, 2007.

[14] P. Lorrentz, W.G.H. Howells, K.D. McDonald-Maier: Enhanced Probabilistic Convergent Network, in: Proceedings of the 6th International Conference on Recent Advances in Soft Computing (RASC 2006), pp. 267 – 272, 2006.

[15] S. Commuri, Y. Li, D. Hougen, and R. Fierro: Evaluating intelligence in unmanned ground vehicle teams, Performance Metrics for Intelligent Systems Workshop (PerMIS'04), NIST, Gaithersburg, MD, 2004.

[16] S.S.C. Botelho, E.V. Simões, L.F. Uebel, D.A.C. Barone: High Speed Neural Control for Robot Navigation, IEEE International Conference on systems, man, and cybernetics, pp.421-429, Beijing, China, October 1996.

[17] W.G.J. Howells, M.C. Fairhurst, Faud Rahman: *An exploration of a new paradigm for weightless RAM-based neural networks*, Electronic, Connection Science, Voln. 12, No.1 pp.65-9 , 2000.

[18] Xilinx University Program Virtex-II Pro Development System: Hardware Reference Manual, UG069, March 2005.