An FPGA based adaptive weightless Neural Network Hardware

P. Lorrentz[a], W. G. J. Howells[b],
[a, b] *Department of Electronics, University of Kent, Canterbury, Kent, CT2 7NT, UK.*
[a]*Tel: +44(0)7813089916*
[a] *plorrentz@gmail.com*
[b] *W.G. J.Howells@kent.ac.uk*

K.D. McDonald-Maier[c]
[c]*Department of Computing and Electronic Systems, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK.*
[c] *kdm@essex.ac.uk*

Abstract

This paper explores the significant practical difficulties inherent in mapping large artificial neural structures onto digital hardware. Specifically, a class of weightless neural architecture called the Enhanced Probabilistic Convergent Network is examined due to the inherent simplicity of the control algorithms associated with the architecture. The advantages for such an approach follow from the observation that, for many situations for which an intelligent machine requires very fast, unmanned, and uninterrupted responses, a PC-based system is unsuitable especially in electronically harsh and isolated conditions, The target architecture for the design is an FPGA, the Virtex-II pro which is statically and dynamically reconfigurable, enhancing its suitability for an adaptive weightless neural networks. This hardware is tested on a benchmark of unconstrained handwritten numbers from the National Institute of Standards and Technology (NIST), USA.

1. Introduction

Learning and reasoning [16], in a digital hardware, may lead to adaptation and reconfiguration. Neural networks have shown to be well suited to learn from examples and adapt to non-linear environments, but many variants are rather resource intensive and therefore prohibitive in practical embedded applications [2]. However, one class of neural networks is more suited to implementation in hardware - the so-called weightless neural networks can be well matched to RAM (Random Access Memory) because their learning and recognition algorithms are mainly associated with reading from and writing to memory.

The aims and objectives of this paper are to present the architecture, and the implementation of an adaptive RAM-based neural network, called Enhanced Probabilistic Convergent Network (EPCN) [10], in a reconfigurable FPGA. Hardware implementations of EPCN is attractive for the following reasons:

1) Compact size and low power consumption compared to a PC based implementation (i.e. it becomes deployable in areas where a PC may not).
2) The binary weights of RAM-based neural networks are contained in RAM, and functions are converted to simple logic gates (AND, NOT, and OR). The logic combinations required to operate the EPCN are of lesser computational intensity than calculus operations.
3) Regular structure: Generally, RAM-based Artificial Neural Network (ANN) are easier to implement on hardware due to their regular structure.
4) The use of reconfigurable IC like FPGA to implement neural network allows fast prototyping and lends itself to modifications at low cost. This makes it a suitable testbed prior to large-volume production.
5) A well designed VHDL based hardware will allow a significant increase in processing thouroughput compared to a software based execution on a general processor.

When this project completes, it is envisaged that Machine Intelligent Quotient (MIQ) may be a measure of its performance. Chalfant [1] introduced a MIQ and proposes the analysis of system architecture and configuration as the criteria for its measurement. Commuri [16] maintains that an architecture of adaptation and learning at all levels of hierarchy simplifies the measurement of MIQ. Learning of a neural network by reconfiguration is demonstrated in [9] using a Virtex-II 6000 FPGA. In [2], Simoes employs ALTERA MAX + PLUS II in the implementation of Goal Seeking neuron (GSN) on Eraseable Programming Logic Device (EPLD). The EPLD is used in classification of British mail postal addresses. Spaanenburg [9] implements two neural networks, one is a feed-forward network to solve the problem of spatial and temporal computing. The second is implementation of Cellular Neural Networks (CNN) for image processing. The FPGA used is Virtex-II 6000 and the learning of these networks were made to depend on reconfiguration

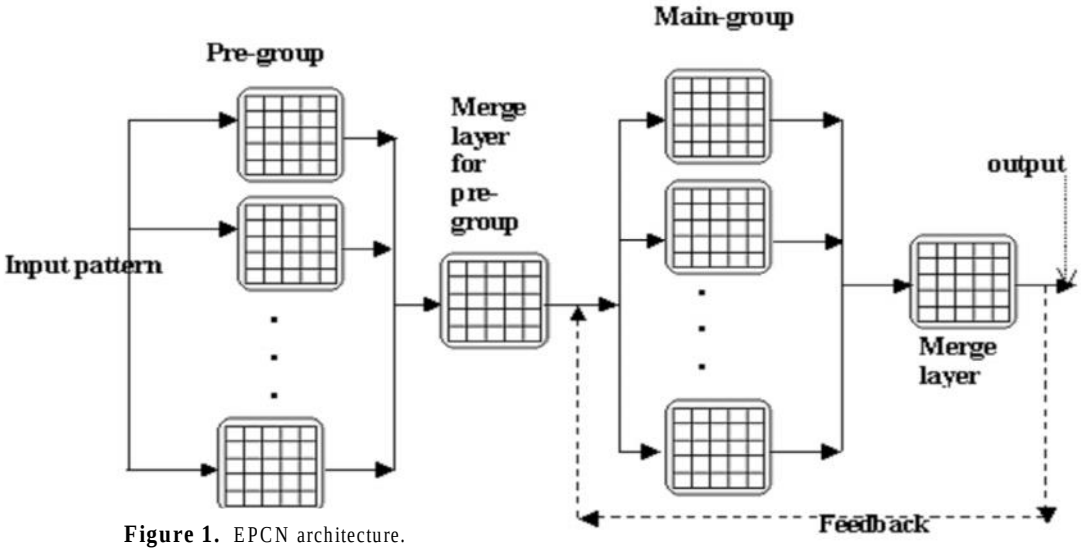


Figure 1. EPCN architecture.

capability of this FPGA. Freeman [11], designed a co-processor based on a binary neural unit known as Correlation Matrix Memory (CMM) which is used for approximate high-speed search and match operations on large datasets. Botelho [17] implements Goal Seeking Neuron (GSN), a RAM-based neural network, on Khepera mobile robot for control and navigation. The RAM-based neural networks in [9],[17], designed on FPGA were deployed in autonomous systems. Most of these systems are application dedicated systems, often for one purpose only. However, the principled EPCN system is generic and highly scaleable. The EPCN when implemented on FPGA could be employed both for prediction and recognition It would thus become suited to a multitude of applications. In this paper however, the hardware is tested on a benchmark of unconstrained handwritten integers from National Institute of Standards and Technology (NIST), USA.

The remainder of this paper is organized as follows. Section 2 introduces the EPCN architecture. The FPGA-based Hardware architecture of EPCN, and its implementation are presented in section 3. The functional testing, and results are presented in section 4. The paper concludes with an analysis of the state of project in section 5, and areas of further research and development are specified in section 6.

2. EPCN – The Enhanced Probabilistic Convergent Network

Weighted Neural Network are those Neural Networks whose performances depend on weights and adjustment of the weights. Conversely, Neural Networks whose performance and system parameters are independent of weights (and their adjustments) are called weightless Neural Networks or sometimes RAM-based Neural Networks [5]. The Enhanced Probabilistic Convergent Network (EPCN) belongs to this latter category.

The architecture of EPCN consists primarily of four component layers of neurons termed the *pre-group*, a *merge-layer* for the *pre-group*, the *main-group*, and *merge-layer* for the *main-group*, as shown in Figure 1. It includes an optional feedback path (represented by dashed arrows) from the merge layer of the main group to the main-group. Each group of the EPCN consist of a number of layers with each member layer consisting of component neurons which are themselves made up of storage locations known as RAM-locations, as shown in Figure 2.

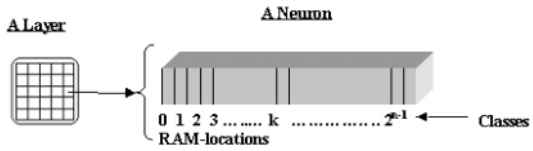


Figure 2. An EPCN neuron.

Each storage location is divided into separate values for each pattern class under consideration for the neuron. The EPCN has a learning algorithm for which the pre-group layers play a vital role. The *pre-group layers* are subsequently merged to give a *merged layer* for the pre-group, [15],[18]. The merged *pre-group layers'* outputs are passed to the *main-group layers*, ending in a merged layer for the main-group.

During learning and recognition, an integer number called the *division* is required for *adjustment* purposes. The term *adjustment* refers to multiplying the integer value in a RAM-location by the *division* and dividing by the number 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. Both learning and recognition algorithms are now presented

2.1 Training algorithm

Within the EPCN architecture, learning is mainly the process of presenting the individual training patterns for connectivity formation and recording the address so formed. The layers within the pre-group of the network are all trained independently. For a given fixed number of pattern classes, only the layers within the pre-group are trained. Learning proceeds as follows for each sample pattern in the training set:-

(1) Prior to the commencement of training, all locations are initialised to zero.
(2) Each pattern sample is subsequently presented to the network in turn.
(3) For each neuron within a pre-group layer, an address is formed using the elements of the sample training pattern.
(4) Depending on the address so formed, the respective RAM-location is incremented for the given pattern class.
(5) Subsequent to the completion of training, an *adjustment* phase occurs to normalise the natural number in each RAM-location.

2.2 Recognition algorithm

Analogously to the learning algorithm, the recognition procedure consists of first presenting to the network the pattern to be classified. Recognition proceeds as follows:-

(1) As for the training process, an address is formed for each neuron using the elements within the pattern meant for recognition.
(2) The contents of the corresponding location so addressed will be modified.
(3) An averaging process is employed on the main-group layers which subsequently produce a merge layer for the main group.
(4) After the merge, an *adjustment* is required, this employs the number *division*.
(5) The output of the main-group is fed back iteratively. As the iteration proceeds, the probability measures for the classes tend to stabilise. If the network reaches a repeating sequence of states rather than a single converged state, iteration may be limited.

2.3 Other similar weightless Neural networks

Almost all weightless NN are derived from either of these units

- WISARD discriminator [4]
- Correlation matrix memory [6]

To date, these units are used in various combination to design weightless neural networks. A typical state-of-the-art design is employed by Bin Azar [3] who utilizes a WISARD discriminator to design a weightless NN for robot navigation. [3] states that WISARD discriminator does not exhibit generalisation inherently. In his implementation, memorization and generalisation abilities were achieved by setting 120 neurons manually. The CMM relies on bitwise OR of input space during learning, and dot-product during recall phase. This find application in the design of C-NNAP [6].

Following is a comparison between EPCN designed in this paper and a typical state-of-the-art design [3].

Table 1. Comparison of FPGA based, typical weightless neural network, and EPCN

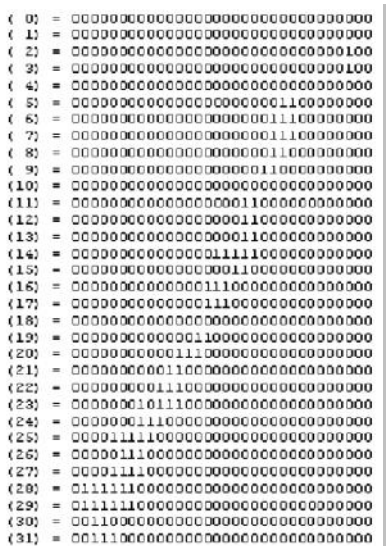
Typical weightless NN	EPCN
Discriminator or correlation matrix memory utilized	No discriminator or correlated matrix memory utilized
May not be an n-tuple classifier	An n-tuple classifier
Does not favour generalisation	Inherent generalisation
May require manual training e.g. [3]	Does not permit manual training.
Binary (“1” or “0”) output	Vector output – scaled probability values. Positive integer output only.

3. The FPGA-based hardware architecture of EPCN

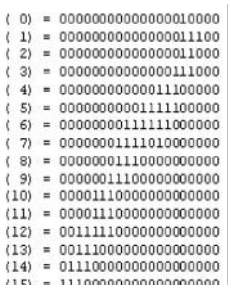
In this section, the architecture of EPCN is proposed that forms a complex hierarchical systems. The design is sub-divided into pre-processing input data, core modules of EPCN, hashing function (unit), reconfiguration, and memory management.

3.1 Pre-processing

The EPCN’s pre-processing steps include the reading in of the input data, or querying a terminal from where the input is to come. The EPCN expects the input values to be expressed in binary number. A compression algorithm, the Lempel-Zif algorithm [7] is included in the pre-processing steps. Most of the common identical data points in the classes will be removed by Lempel-Zif algorithm in order to permit real-time rescaling of input pattern as and when required. For example, the input, Figure 3(a), is pre-processed resulting in Figure 3(b) during input processing.



(a)



(b)

Figure 3. The pre-processing; 3 (a) is pre-processed resulting in 3(b).

3.2 The EPCN

The EPCN is here described using the hierarchical system of design. The design is synthesised by Xilinx ISE during which EPCN is analysed and converted into digital circuit components. Figure 4 shows the main block modules constituting the EPCN architecture. In Figure 4, the train-block and the recognise-block are both connected to the input pre-processing block via the control unit and the hashing function that produces the addresses. The control unit initiates pre-processing when data are available at the input. After the completion of pre-processing of the input training data, it initiates the training processes (section 2.1). The train-block signals a finish flag when training completes. On reception of learn-flag-complete, the control unit checks the recognition input for data. When data is present, pre-processing is done for the pattern meant for recognition. When the pre-processing-stop flag is detected by the control unit, the recognition block starts the recognition processes (section 2.2). The output block is monitored by the control unit through a feedback system. Iteration of the recognition processes stops when values in the output block are stable, by querying the output block, or after a pre-defined number of iteration steps.

The overriding majority of the EPCN block architecture consist of memory. Its functional behaviour is concentrated on data flow from and to these memory locations. The memory location in EPCN is described as single-port block RAM driven by a registered read address and a synchronous write operation.

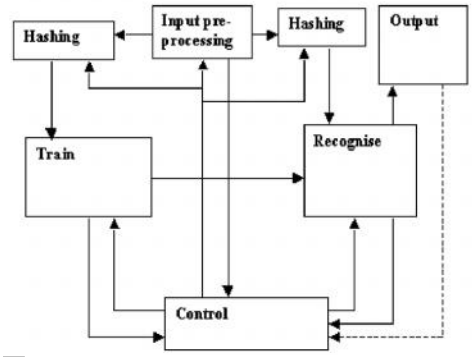


Figure 4. The block-diagram of EPCN FPGA architecture.

3.3 Hashing

A hashing function is often used to search and retrieve information from memory. Such a hashing function as used in [11] is based on bit-folding, XOR, and pseudo-random number generator.

In this paper however the hashing function implemented is based on XOR and Maximum-length Shift-register code [7]. Maximum-length shift-register codes generates a systematic code with desired output length;

$$n = 2^m - 1 \text{ -------- } (1)$$

where m is the information bit derived 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;

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

The maximum-length shift-register codes (MLSR) are dual of cyclic Hamming codes.

Bits in the pattern become the information bits. The hashing is used for address (connectivity) formation. Data will be written to or retrieved from the LUT-RAM location whose address is so formed. Examples to illustrate this are given below.

Example I: when m = 3; equation (1) becomes

$$n = 2^m - 1 = 2^3 - 1 = 7;$$

This means that 7 addresses are required. In equation (2) the parity polynomial h(p) becomes;

$$h(p) = p^7 + h_6p^6 + \ldots + h_0p^0 \quad \text{--------(3);}$$

In equation (3) it is seen that the coefficient of p^7 is 1. h_i, (i = 0,1,2,... 7) is such that $h_{k-1}p^{k-1}$ is an integer between 7 and 0. This is a constraint to be satisfied. Most of the values of $h_{k-1}p^{k-1}$ will be zero. Looking at Figure 5, it is seen that non of the values assigned to "ttuple" is greater than 7.

```
tcol =
(00000000000C0000000000000001000 00000000000000000000000000001000
00000000000C0000000000000000100 00000000000000000000000000000010
00000000000C0000000000000000110 00000000000000000000000000000101
00000000000C0000000000000000111 00000000000000000000000000000011
00000000000C0000000000000000001 00000000000000000000000000000000)
tclas =
(00000000000C0000000000000000011 00000000000000000000000000000001
00000000000C0000000000000001000 00000000000000000000000000000100
00000000000C0000000000000000010 00000000000000000000000000001001
00000000000C0000000000000000110 00000000000000000000000000000101
00000000000C0000000000000000111 00000000000000000000000000000000)
ttuple =
(00000000000C0000000000000000111 00000000000000000000000000000111
00000000000C0000000000000000111 00000000000000000000000000000100
00000000000C0000000000000000010 00000000000000000000000000000110
00000000000C0000000000000000101 00000000000000000000000000000011
00000000000C0000000000000000001 00000000000000000000000000000000)
trow =
(00000000000C0000000000000001000 00000000000000000000000000001000
00000000000C0000000000000000101 00000000000000000000000000000111
00000000000C0000000000000000011 00000000000000000000000000000001
00000000000C0000000000000000100 00000000000000000000000000000010
00000000000C0000000000000000110 00000000000000000000000000000000)
```

Figure 5. This are (an instance of) connectivity derived from input pattern by hashing. Here "ttuple" and "tclas" are for neuron location, while "trow" and "tcols" are pre-group layer addresses.

Example II: What if ten addresses 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) gives

$$h(p) = p^{15} + h_{14}p^{14} + \ldots + h_0p^0 \quad \text{--------(4);}$$

In equation (4) it is seen that the coefficient of p^{15} is 1. h_i, (i = 0,1,2,... 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) equates to zero for those values that are not required.

In practice however, zero is a valid address. To distinguish between unwanted zeros and wanted zeros as addresses, the register values in the location whose values are not required are set to 2^n . Then these locations are not accessed.

The information bits in a pattern characterise that pattern, and thus the connectivity is reproducible.

3.4 Reconfiguration

The network structure and size changes with learning and classification. In practice the maximum size and structure is limited dependant on the hardware resources. The number of *pre-group layers*, the number of *main-group layers*, and the *division* (see section 2) could be referred to as system parameters. Different EPCNs, characterized by different system parameters, are obtainable from this single implemented adaptive system. This is done by specifying different system parameters during reconfiguration. Preliminary experimentation has revealed that the available resource of the specific FPGA used 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 design is such that these values are not exceeded. The control unit mediates between the pre-processing unit and the hashing function (unit) to ensure that the size of the pattern used within the EPCN does not exceed the maximum neuron size possible. The possibility of reducing every pattern size below this value (20-by-15) always exists. This solves the boundary problems. The pre-processing is such that vital information is retained within the pattern after pre-processing.

```
package mathneuro2 is
.
.
.
constant tuple:integer:=3;
constant divisn:integer:=1000;
.
.
.
end mathneuro2;

entity epcnvl is
    Port ( ...
   noslay:in std_logic_vector(2 downto 0);
      ...
      );
end epcnvl;
```

Figure 6. This shows system parameters that are mainly responsible for reconfiguration

The number of *pre-group layers* is changeable before a training session begins. The number of neurons per layer is automatically determined from input training patterns after pre-processing. The number of *main-group layers* is changeable before a recognition session begins. The number of neuron per main-group layer is automatically determined from the "recognition" pattern after pre-processing.

The number *division* (see section 2 for definition) used during various *adjustment* (see section 2) phases could vary from 1 to 2^{15}. The possibility of the variability in system parameters are vital to static and dynamic reconfiguration. Varying the number of *pre-group layer* is done by specifying a value to the variable "noslay" before training begins, see Figure 6. So also, varying the number of *main-group layer* is done by specifying a value to the variable "noslay" before recognition begins. Changing the value assigned to *division* is done by prefixing "constant

divisn" (Figure 6)with a "generic" statement. This is done before a training and a recognition session pair.

3.5 Memory management

As indicated previously, the memory location in EPCN is described as single-port block RAM driven by a registered read address and a synchronous write operation. in Xilinx's LUT-RAM.

The memory management is handled by the *control unit*, see Figure 4. During a read operation from a location, a write operation is disabled with respect to that location. During a write operation to a location, any read operation from that same location is disabled.

As concerning the buses, a read/write enable/disable operation depends on the addresses formed from the pattern. The FPGA consists of configurable logic blocks (CLBs). These CLBs are inter-connected by buses. Activating a bus, and which bus is being activated depends on if its address is formed. Activating a bus is necessary prior to a read/write operation, otherwise a read/write operation is not possible on that bus. A read and write addresses, with respect to one bus will not be formed at the same time. It is either a read address or a write address. It ensures that values are not written to and read from the same bus at the same time. This is commonly referred to as bus contention.

4. Results

The EPCN was designed and implemented using Xilinx ISE 9.21i. The NIST data base[1] has been used for testing the functionality of EPCN and has been found suitable for use. Testing was done by instantiating the EPCN in a test-bench and associating the NIST handwritten data set with its input.

The prototyping boards is linked to the computer via a USB programming cable. Auto-recognition of the programming cable by Xilinx ISE enables the download of the bit file generated from EPCN, and configuration of the board by impact (component of Xilinx ISE) using PROM file generated from EPCN.

To display the output of EPCN on board the FPGA, RS232 cable is connected via the com port and linked to the HyperTerminal at a baud rate of 2400 – 9600. Recognition results are displayed on the HyperTerminal in a PC. Internal to the virtex-II pro FPGA is a 2MB SDRAM. External SDRAM at 2GB is also attached. This makes possible an increased number of LUTS generated and utilized during learning and recognition of EPCN in FPGA.

[1] National Institute of Standards and Technology (NIST) in Gaithersburg USA. NIST provide the handwritten simple form (HFS) of numerals, which were binarised and resize to 32-by-32.

An example of the FPGA resource utilisation of EPCN is shown in Table 2. These are the resource requirements for the following EPCN architecture:
Tuple-size = 3;

- Pre-group layers = 3;
- Main-group layers = 2;
- Class = 10;
- Number of neuron per layer 20-by-15;

Table 2. An example of resource utilisation showing the conversion of EPCN to gate-level components.

Logic Utilisation	Used	Available	Used %
Total Number Slice Registers (SR)	1,612	27,392	5%
Total Number SR used as Flip-flops	1,611	1,612	99.9%
Number of 4 input LUTs	3,532	27,392	12%
Number of occupied Slices	2,492	13,696	18%
Number of Slices containing only related logic	2,492	2,492	100%
Number of bonded IOBs	24	416	5%
Number of GCLKs	2	16	12%

From Table 2, it is seen that the resource utilisation is relatively low for the given example network.

Using the NIST handwritten integers, the EPCN is trained on 0 to 9, and during recognition requested to recognise "1". Data for training are selected from the training set while patterns for recognition were selected from recognition set. Both training set and recognition set form disjoint sets.

```
grp.dssout = {0000000000000000 0000011010001101 0000011010001101
0000010100010110 0000001001011001 0000000000000000 0000000000000000
0000000001111010 0000011010100010 0000110101101100
0000000000000000}
          (0) = 0000000000000000
          (1) = 0000011010001101
          (2) = 0000011010001101
          (3) = 0000001010010110
          (4) = 0000001001011001
          (5) = 0000000000000000
          (6) = 0000000000000000
          (7) = 0000000001111010
          (8) = 0000011010100010
          (9) = 0000110101101100
```

Figure 7. Wrong recognition: A recognition result from EPCN when trained on "0" to "9", and shown "1" in recognition phase.

Result of type figure 7 shown above is obtained when a pattern is shown to the network for recognition. Figure 7 shows the result when one input pattern is shown to the network for recognition. In this figure, the numbers 1,2,3,...,9, on the left-hand-side represents the classes while the binary numbers to the right-hand-side represents the probability (scaled by *division*) with which the

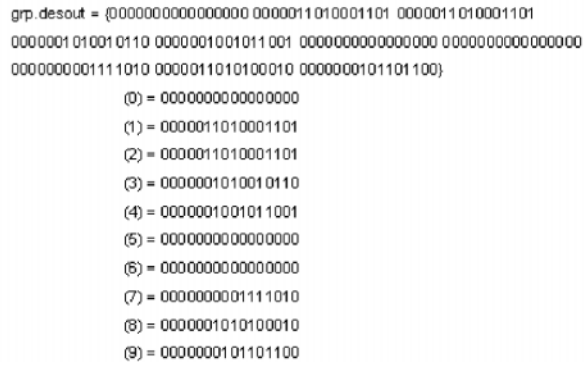


Figure 8. Ambiguous state: A recognition result from EPCN when trained on "0" to "9", and shown "1" in recognition phase.

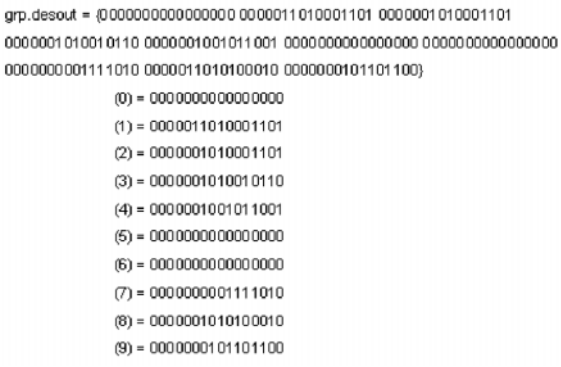


Figure 9. Correct recognition: A recognition result from EPCN when trained on "0" to "9", and shown "1" in recognition phase.

pattern belong to that class. By varying the configuration of EPCN and showing it the same pattern for recognition, different other results are obtainable as shown in Figures 8 and 9. Three possibilities exist in recognition processes of EPCN. They are correct classification (Figure 9), ambiguous classification (Figure 8), and wrong classification (Figure 7). The binary numbers in Figure 8 and 9 has same meaning as in Figure 7.

5. Analysis

From the design and the example resource utilisation, Table 2, it could be inferred that static and dynamic variation both of precision (word length) and system parameters are possible. As these are fully supported by available resources, up to a certain maximum (see the sub-section 3.4).

Figures 7,8 and 9 are result types obtainable from EPCN. Figure 7 is a case of wrong classification, while Figure 8 is an abiguous state. Figure 9 shows correct recognition. These results are obtained when character "1"' is shown to EPCN. The different outputs, due to changes in system parameters, is indicative of the possibility of changes in desicion due to changes in environment.

A pattern of 15-by-20 in size infers 300 neurons per layer. And there are many of such layer in any instance. This demonstrates the possibility of implementing an advanced and large weightless neural network, the EPCN, wholly on FPGA, the pre-processing steps inclusive.

6. Conclusion

The EPCN has been shown to be portably and wholly implement-able on FPGA. Pre-processing steps have been included in this design. The results demonstrate the possibility of implementation of a large, advanced, and adaptive weightless EPCN in a re-configurable FPGA.

Areas for further research include introduction of machine intelligent quotient (MIQ) as a means of self-assessment, and dynamic parameter tuning of the network.

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] Hannan Bin Azhar,M.A., Dimond K.R, Design of an FPGA based adaptive neural controller for intelligent robot navigation Digital System design,2002, Proceedings. Euromicro Synposium 2002.

[4] Igor Aleksander, from Wizard to Magnus: A family of weightless virtual neural machines. Dept. of Electrical and Electronic Engineering, Imperial College of Science Technology and Medicine, London, UK.

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

[6] J.V. Kennedy, J.Austin, R.Pack, B.cass: C-NNAP: A dedicated processor for Binary Neural Networks; Advanced Computer Architecture Group, Dept. of Computer Science, University of York, Heslington, York, YO1 5DD, UK.

[7] John G. Proakis: Digital Communication, 4th Edition, McGraw Hill Int. Edition, 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: A novel weightless neural based Multi-classifier for large classification, *Neural processing letters (submitted)*, 2007.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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