ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2009. No. 8(96)*
ELEKTRONIKA IR ELEKTROTECHNIKA

AUTOMATION, ROBOTICS
T 125
AUTOMATIZAVIMAS, ROBOTECHNIKA

Modeling and Analysis of Neuro–Genetic Hybrid System on FPGA

M. NirmalaDevi, N. Mohankumar
VLSI Design Research Group, Department of Electronics & Communication Engineering, School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, TamilNadu, India, phone: + 91–422– 2656422, fax.+91–422–2656273
e–mails: nirmala_amrita@rediffmail.com, nirmala_amrita@yahoo.co.in, mk.mohankumar@gmail.com
S. Arumugam
Nandha Institutions, Erode, TamilNadu, India,
e–mail: arumugamdote@yahoo.co.in

Introduction

Artificial Neural Networks (ANNs) are computational models of a fractional part of biological nervous system. ANN is computation intensive to suit complex applications [1] though its structure is simple. Most of the ANN applications use Feed Forward (FF) architecture with gradient–based learning like Back Propagation (BP) algorithm [2] or modified BP algorithm [3]. As the complexity of the network increases, the search space appears with more and more local optima and gradient–based learning may not always lead to global minima. Moreover BP needs complex operation, which restricts the search coverage. To improve the global convergence capability, an Evolutionary Algorithm (EA) [4] can be used. It refers to a special class namely; Evolutionary Artificial Neural Networks (EANN's) in which evolution is another fundamental form of adaptation in addition to learning [5]. EANN can be exploited to design the architecture, learn weight, adapt the learning rule and extract the rule from ANN [6]. EA was broadly classified as Evolutionary Strategies (ES), Evolutionary Programming (EP) and Genetic Algorithms (GA), though many other types have emerged in the recent past [7]. The work presented here uses GA to simultaneously evolve the structure and weight of ANN. Capability of GA in the exploitation of information guides the direction of search towards feasible region and hence it converges at global optima.

Although the software implementation exactly replicates the given algorithm, relatively inexpensive massive parallelism is exhibited when ANN is realized in hardware using Very Large Scale Integration (VLSI). A digital implementation of Genetic Algorithm based Neural Network (GANN), to solve Parity Function and Character Recognition problem is explained here. To solve N–bit parity functions, the Feed Forward Neural Network (FFNN) needs N neurons in the hidden layer [8]. Stork and Allen [9] reduced it to two hidden units with diode–like Activation Functions (AF). Few others [10, 11] had tried to solve the function without imposing any constraint on AF. On the other hand, genetically evolved Neural Network (NN) proposed in this paper, solves the N–bit parity function with *N/2* neurons in the hidden layer. The second application of GANN realizing Character Recognition, exhibits improved performance in terms of faster convergence with acceptable error. Thus the GANN to solve benchmark problems of 8–bit Parity and Character Recognition is proved to consume lesser area with increased speed and reduced error. By altering the topology and links, the designed Neuro–Genetic Hybrid System can solve any function with binary inputs. Main objective of the research is to implement the evolved neural network in VLSI hardware exhibiting the following characteristics;

- Simplicity and regularity of the structure with minimal interconnections
- Expandability and design scalability to combine more modules together
- Reduction in silicon area by replacing multiplier with comparator in the hidden neuron
- Absence of learning unit due to the genetic evolution of structure and weight, minimizing the hardware further
- Solving the parity function with reduced hidden layer neurons and recognition of character in lesser number of generations

The review of GANN, algorithmic development, architecture design and observed results are presented in the following sections. Simulated results of GANN ensure the successful implementation of neurohardware.

Review of GANN

Genetic evolution of NN may broadly be classified as non–invasive and invasive technique. Non–invasive method combines GA and gradient learning,

where the former evolves the structure and the latter adapts the weights. Since it involves gradient method, proper initialization and network implementation is needed to overcome the local minima problem [12]. On the other hand, invasive method uses GA for both weight and topology evolution of ANN. A purely non–invasive approach with a constructive algorithm is demonstrated in [13] to evolve Cooperative NN Ensembles (CNNE), using incremental learning. This reduces redundancy and maintains diversity to offer a better solution. Smalz and Conrad [14] proposed a method to assign fitness to individual neurons of ANN, rather than the whole network. Odri, S. V., Petrovacki, D. P. & Krstonosic, G. A [15] developed a non–population based learning algorithm, which could alter the architecture of ANN.

McDonnell, J. R. & Waagen, D. [16] tried invasive approach to evolve the interconnectivity with weight values of ANN, to solve binary mapping and 3–bit parity problem. Improved GA for tuning the structure and parameters of an ANN is explained in [17]. Generation of three offsprings using different mutation operation leads to the improvement. Structural and weight learning by mutation is employed in GNARL algorithm to construct Recurrent NN [18]. Based on GNARL algorithm, a Mutation–based Genetic NN (MGNN) is implemented in [7]. The invasive method of using GA for simultaneous weight and topology evolution is adopted here. The hardware friendly hybrid system is implemented in VLSI and its merits are discussed with results.

Algorithmic development of GANN

The factors influencing GANN evolution are encoding scheme, genetic operators, fitness function and stopping criteria. Presented work follows binary encoding scheme to simplify the digital hardware implementation. The genetic operators namely; two–point crossover and uniform mutation with probability of *pc* and *pm* of 0.8 and 0.01 respectively are applied. The fitness function of GANN is chosen to be minimization of mean squared error. Convergence of the network is ensured and computation is stopped when the fitness variation in consecutive iteration is insignificant.

Primarily in GA, a set of initial encoded schedules known as chromosomes is randomly created. Each schedule is valuated for "fitness". Then, processes based on natural selection, crossover, and mutation are repeatedly applied on a population of binary strings which represent potential solutions. Over time, the number of above–average individuals increases and better–fit individuals are created, until a good solution to the problem at hand is found [19].

Pseudo code of GANN

The procedure followed in the algorithm is explained with example

1. Initialize the random population (weights and bias of network);
2. Generate input and target vectors to suit the application;
3. Assign chromosomes from the population to the network of initial topology;
4. Simulate the network;
5. Evaluate the fitness function or Mean Square Error (MSE);
6. Increment the number of hidden neurons and repeat the process, until a best fit population is obtained;
7. Do two point crossover on the population to exchange information between parents:
   For example, if p1 and p2 are the parents
   p_1 = [11111111], p_2 = [00000000];
   and the crossover points are 3 and 6 then
   Child = [11100011];
8. Get the best population, choose parent and mutate to gain the lost material:
   For example, let the parent be *P*: 000**0**000**0**
   After uniform mutation *Child*: 000**1**000**1;**
9. Stop if the condition is satisfied and plot the required results.

The algorithm is executed with an initial population of chromosomes of length 41 and 50 for the 8–bit Parity Function and Character Recognition problem respectively.

Application Examples

Eight–bit Parity Functions. To verify the performance of the evolved neurohardware, eight–bit parity problem is chosen. Any change in input alters the output and hence the parity problems become harder to solve. It is one of the benchmark problems due to its simple definition but great complexity [20]. Feedforward networks with one hidden layer require eight neurons in the hidden layer that is reduced to four, when genetically evolved. Hence FFNN with *N/2* neurons in hidden layer is implemented in this paper and the performance is validated. Design of four hidden neurons to realize three–bit parity is now utilized to solve eight–bit parity function as shown in Fig. 1.

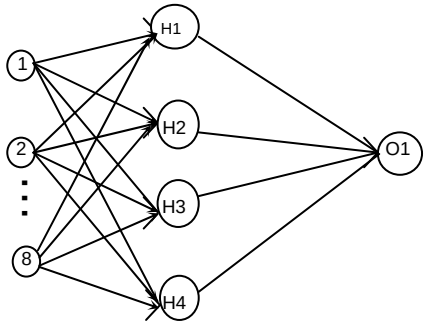

Fig. 1. Architecture to solve 8–bit parity function: 1~ 8 – Input Layer Neurons; H1~H4 – Hidden Layer Neurons; O1 – Output Layer Neuron

Topology evolution and network convergence for varying number of hidden layer neurons of 8–bit parity function is presented in Table 1.

TCLX – Character Recognition. Character recognition is a trivial task for humans, but for computers it is extremely difficult. The main reason for this is, the many sources of variability. Characters represented by 7 × 5 matrix was realized using pulse–coupled neurons [21]. To reduce complexity, a 3 × 3 array of pixels is chosen

[22] to represent the characters T, C, L and X in nine bits as shown in Fig. 2. A black pixel and white pixel denotes one and zero, respectively.

Table 1. Evolution of 8–bit parity function

Network Topology	Fitness value	Generations
8–3–1	0.256	45
8–4–1(best network)	**0.25**	**50**
8–5–1	0.265	63
8–6–1	0.256	67
8–7–1	0.25	55
8–8–1	0.25	70

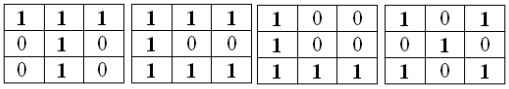

Fig. 2. A 3 × 3 array of characters – TCLX

The four combinations of two output neurons are assigned to denote the four characters, as shown in Table 2.

Topology evolution of Character Recognition converges to 9–4–2 with least possible error, as shown in Fig. 3 and Table 3 respectively.

Table 2. Binary assignment of TCLX

	Binary representation	Output
T	1 1 1 0 1 0 0 1 0	0 0
C	1 1 1 1 0 0 1 1 1	0 1
L	1 0 0 1 0 0 1 1 1	1 0
X	1 0 1 0 1 0 10 1	1 1

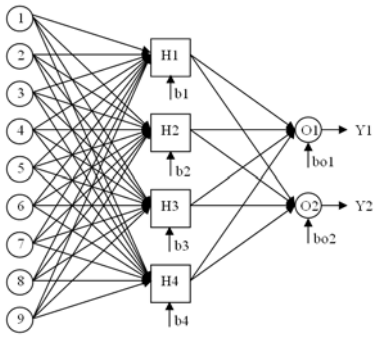

Fig. 3. Architecture 9–4–2 of character recognition

Network with optimized number of hidden layer neurons converges fast within 55 generations at an acceptable error. Individual modules of the realized hardware are explained.

Table 3. Evolution of Character Recognition

Network Topology	Fitness value	Generations
9-3-2	0.005077	50
9-4-2(best network)	**0.004432**	**55**
9-5-2	0.007234	71
9-6-2	0.02604	60
9-7-2	0.072332	67
9-8-2	0.053875	63

Architectural Design

Algorithmic flowcharts of computational models namely; Multiplier and Adder are presented in Fig. 4 and Fig. 5 respectively. A simple Comparator is also designed. These modules are combined together to realize hidden layer neuron, output layer neuron and then the complete network. Optimized weights are converted to the equivalent 32–bit single–precision IEEE 754 format, to make it compatible. It has three components namely sign (*s*), exponent (*e*) and mantissa (*m*). While realizing the network in hardware the hidden neuron structure is defined and replicated when required. Design of hidden neuron differs from output neuron.

Computation Modules of Neurons

Multiplier Module. Multiplication is executed as follows. Extract the sign(*s*), exponent (*e*) and mantissa (*m*) of multiplier and multiplicand. Add the exponents and store the result along with the carry bit. XOR the sign bits and multiply these two 23–bit mantissa. Adjust the exponent depends on the higher order bit of the multiplied result. Assign the sign bit for the result based on the two operands and normalize the resulting mantissa. Place the resulting sign, exponent and mantissa into 32–bit format

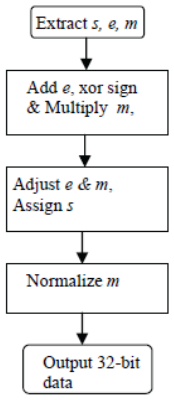

Fig. 4. Flowchart of multiplier

AdderModule. Extract the sign, exponent and mantissa of two operands. Compare the exponent and mantissa to check the absolute value and align the decimal points for addition or subtraction.

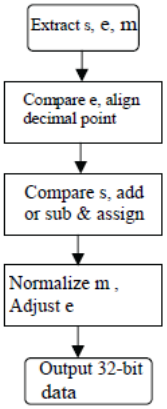

Fig. 5. Flowchart of adder

Compare the sign bit and add if equal; else subtract the numbers to get final result. Assign the sign, based on the magnitude. Normalize the resultant mantissa and adjust the exponent of the result. By interconnecting the individual modules, the network is realized. Though the explanation below refers to Character Recognition, the hidden layer and output layer neurons are designed to solve both the problems.

TCLX – Character Recognition

The requirements of ANN such as parallelism and performance are related to the silicon area consumed to realize the network. Different types of parallelism are mentioned like Layer parallelism, Neuron parallelism and Synapse parallelism [23]. Neuron parallelism that requires one multiplier per neuron is realized in the architecture presented here. Hence the neurons of same layer computes in parallel and computation between layers are executed sequentially. Character Recognition topology is already shown in Fig. 3.

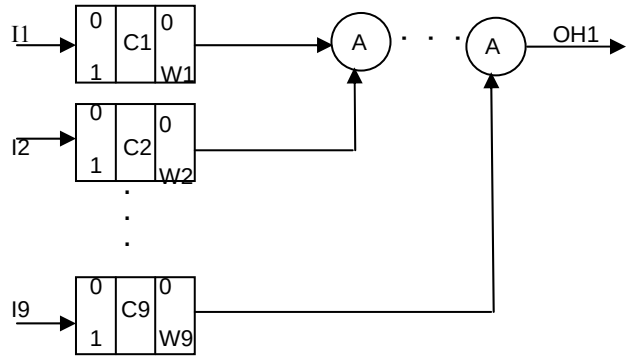

Fig. 6. Hidden layer neuron: I1 ~ I9 – inputs; W1 ~ W9 – weights, C1 ~ C9 – comparators, A – Adder; OH1 – output

Hidden Layer Neuron for Character Recognition: The output of the hidden layer neuron is the multiplied sum of input and weight. Being a binary input, it need not be multiplied by weight. A module to check whether the input is zero or one (i.e.) a comparator will serve the purpose. Each hidden layer neuron comprises nine comparators and eight adders to solve character recognition problem as shown in Fig 6. Hidden layer neuron accepts binary input and the comparator will either pass 0 or the corresponding weight to the adder as a result of input and weight multiplication. For example, if the 9–bit input is 000011111 then the output of the nine comparators from top to bottom would be 0, 0, 0, 0, W_5 ~ W_9. Comparators are used instead of multipliers to simplify the architecture, which reduces the area significantly and improves the speed. Output of comparators is added and the hidden neuron output OH_1 is passed on to the next layer for further computations. Multipliers are used only in output layer neuron of the proposed design.

Output Layer Neuron for Character Recognition: Each neuron in the output layer utilizes four multipliers and three adders as shown in Fig. 7.The weights are multiplied with the hidden neuron outputs and added to yield the final output. Proper threshold is set in the output adder, to decide the discrete binary output. Multiplier and Adder modules are explained already with flowchart.

Results of simulation

A Neuro–genetic hybrid system is implemented in hardware to solve parity function and character recognition problems. Results after simulation exhibit satisfactory performance.

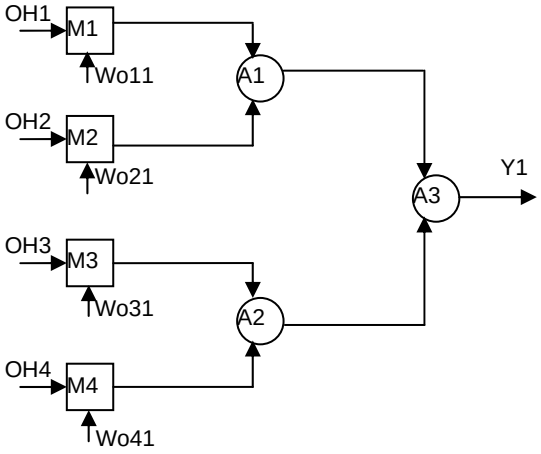

Fig. 7. Output layer neuron: OH1 ~ OH4 – outputs from hidden layer neuron; Wo1 ~ Wo4 – weights; A1 ~ A3 – Adders; M1 ~ M4 – Multipliers & Y1 – final output

Eight–bit Parity Function

Evolved weight set of input–hidden layer and hidden–output layer neurons for parity function is shown in Table 4 and Table 5 respectively. The internal computation modules are ensured for proper functionality and interconnected together to realize the application.

The top signal '*Reset'* assigns weights, when it is high and the network produces output when reset is low. A clock 'clk' of 100 ns is applied, to trigger the network on positive edge.

Table 4. Weight set of input–hidden layer for parity function

Input & Hidden Neurons	**H1**	**H2**	**H3**	**H4**
1	0.8357	1.236	0.9488	1.2441
2	0.8896	1.2732	0.3288	0.6875
3	0.5753	0.705	2.0304	–0.2142
4	0.691	0.8228	1.3995	0.3744
5	0.7944	1.4901	0.0055	–0.0421
6	0.8678	0.9432	0.8898	0.5914
7	0.8965	0.4306	0.3753	–0.4368
8	0.3099	0.6362	0.2716	–0.048

Table 5. Output layer weight and bias of parity function

Hidden layer	**Output layer weights**	**Bias**
1	0.886	1.3336
2	-0.178	-0.2937
3	-0.2722	1.7109
4	0.0429	0.0551
Output layer	--	0.0925

The design was downloaded in SPARTAN 3E – XC3S 500E – 5CP132 using Xilinx ISE 8.1i and verified using Leonardo Spectrum tool.

Table 6. Device utilization summary–parity generation

Logic Utilization	Used	Available	Utilization
No. of Slices	1369	4656	29%
No. of Slice flipflops	320	9312	3%
No. of 4 input LUTs	2407	9312	25%
No. of bonded IOBs	11	92	11%
No. of GCLKs	8	24	33%

TCLX – Character recognition

Similarly the evolved weight sets for Character Recognition is presented in Table 7 and Table 8. Proper convergence of network recognizes the binary patterns applied and hence the characters as displayed.

The signals as mentioned in Parity function are displayed for character recognition after simulation. Feasibility of the GANN promises that the network after synthesis could be configured on Field Programmable Gate Arrays (FPGAs) through the commercial Place & Route.

If an eight–bit input, *'Inp'* is applied, the output, *'outtresh'* displays the even parity output on the rising edge of next clock cycle. Input to hidden neuron and hidden to output neuron weight set is also displayed.

Table 7. Weights and bias of hidden – output layer for character recognition

Neurons of Hidden layer	Output layer 1	Output layer 2	Hidden layer Bias
1	0.346	0.5228	–0.3896
2	0.6464	–1.1363	1.4192
3	0.4366	–0.0633	0.9215
4	0.0854	–0.4011	1.0728
Output layer Bias	2.2747	2.7092	–

Table 8. Weight set of input–hidden layer of character recognition

Neurons of Hidden layer & Input layer	1 →	2	3	4
1	-0.0826	0.4273	0.0031	0.8917
2	0.1731	-2.5914	0.8914	0.8123
3	-0.076	-1.6177	2.1266	0.2951
4	0.2617	-0.513	1.4674	1.9791
5	1.2786	-1.6695	2.1176	-0.3155
6	0.4235	2.0695	0.7347	-0.8926
7	-0.5047	-1.2504	2.9228	1.7954
8	1.8538	-1.3483	0.6531	-3.6263
9	-0.3787	-2.6797	0.3509	1.3465

The design was downloaded in SPARTAN 3E – XC3S 500E – 5CP132 using Xilinx ISE 8.1i and verified using Leonardo Spectrum tool. Table 9 shows the device utilisation summary for TCLX character recognition.

Table 9. Device utilization summary–character recognition

Logic Utilization	Used	Available	Utilization
No. of Slice Latches	877	9,312	9%
No. of 4 input LUTs	5,213	9,312	55%
Logic Distribution			
No. of occupied Slices	2,817	4,656	60%
No. of Slices containing only related logic	2,817	2,817	100%
No. of Slices containing unrelated logic	0	2,817	0%
Total No. of 4 input LUTs	5,364	9,312	57%
No. used as logic	5,213		
No. used as a route-thru	151		
No. of bonded IOBs	13	92	14%
IOB Flip Flops	2		
No. of GCLKs	20	24	83%
Total equivalent gate count for design	45,144		
Additional JTAG gate count for IOBs	624		
Total memory usage (KB)	76104		

Conclusion

Genetic Algorithm based Neural Network is designed to solve eight–bit Parity function and TCLX Character Recognition applications. The GA learning methods with different number of hidden layer neurons was investigated for ascertaining how change in neural network architecture contributes to the overall fitness of the input–weight combinations. Combined evolution of structure and weights for different functions are tried and implemented successfully. Genetic learning of weight makes the hardware implementation of FFNN more feasible. GANN without multiplier in hidden layer contributes to further reduction in hardware. Thus the GANN to solve benchmark problems of 8–bit Parity and Character Recognition is proved to consume lesser area with increased speed and reduced error. Since fixed control parameters may hinder the convergence on the latter stage of generation, dynamic assignment of parameters like the rate of genetic operators is identified for further research. And the high potential GA can be exploited to produce 'Pareto–optimal' solutions for the multi–objective optimization.

References

1. **Liu J., Liang D.** A Survey of FPGA–Based Hardware Implementation of ANNs. Proc.IEEEConf. – 2005. – P. 915–918.
2. **Yao X., Liu Y.** A New evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks. – 1997. – No. 8(3). – P. 694–713.
3. **Hikawa H.** A Digital hardware pulse–mode neuron with piecewise–linear activation function.IEEE Transactions on Neural Networks. – 2003. – No.14(5). – P. 1028–1037.
4. **Rumelhart D., McClelland J.** Parallel distributed processing:Explorations in microstructure of cognition. Cambridge, MA: MIT Press. – 1986.
5. **Yao X.** A review of Evolutionary Artificial Neural Networks. Int. J. Intell. Syst. – 1993. – No. 8(4). – P. 539–567.
6. **Martinetz T. M., Berkovich S. G., Schulten K. J.** Neural–gas Network for Vector Quantization and its Application to

Time–series Prediction. IEEE Trans. on Neural Networks, 1993. – No. 4. – P. 558–569.
7. **Palmes P. P., Hayasaka T., Usui S.** Mutation–Based Genetic Neural Network. IEEE Trans. on Neural Networks, 2005. – No.16(3). – P. 587–600.
8. **Hertz, J., Krogh, A., Palmer R.** Introductionto the theory of neural computation. Reading, MA:Addison–Wesley. – 1991.
9. **Stork, D. G., Allen, J. D.** How to solve the N–bit parity problem with two hidden units. Neural networks. – 1992. – No. 5. – P. 923–926.
10. **Hohil M.E., Liu D., Smith S. H.** Solving N–bit parity problem using Neural networks. Neural Networks, 1999. – P. 1321–1323.
11. **Lavretsky, E.** On the exact solution of the parity–N problem using ordered neural networks. Neural Networks, 2000. – No. 13. – P. 643–649.
12. **Gori M. Tesi A.** On the problem of local minima in backpropagation. IEEE Trans. on Pattern Anal. Mach. Intell , 1992. – No. 14(1). – P. 76–86.
13. **Islam M. Yao X. Murase K.** A Constructive Algorithm for training cooperative neural network ensembles. IEEE Trans. on Neural Networks, 2003, – No. 14(4). – P. 820–834.
14. **Smalz, R., Conrad, M.** Combining evolution with credit apportionment: A new learning algorithm for neural nets. Neural Networks, 1994. – No.7(2). – P. 341–351.
15. **Odri S. V., Petrovacki D. P. Krstonosic, G. A.** Evolutional development of multi level neural network. Neural Networks, 1993. – No. 6(4). – P. 583–595.
16. **McDonnell J. R. Waagen D.** Evolving neural network connectivity. Proc. Amer. Power Conf. 1993. – P. 863–868.
17. **Leung F. H. F., Lam H. K., Ling S. H., Tam P. K. S.** Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm. IEEE Trans. on Neural Networks, 2003. – No.14(1). – P. 79–88.
18. **Angeline P. J., Saunders G. M. Pollack J. B.** An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. on Neural Networks, 1994, – No.5(1). – P. 54–64.
19. **Srinivas M., Patnaik, L. M.** Genetic algorithms: A survey. IEEE Computer, 1994, – No.27. – P. 17–27.
20. **Franco L., Cannas S. A.** Generalization properties of modular networks: Implementing the parity function. IEEE Transactions on Neural Networks, 2001. – No.12(6). – P. 1306–1313.
21. **NirmalaDevi M. Arumugam S.** Modeling Of Pulse–Coupled Neurohardware Using Simulink. Int. J. Electrical Engineering, 2007. – P. 76–84.
22. **Maeda Y. Tada T.** FPGA Implementation of a Pulse Density Neural Network With Learning Ability Using Simultaneous Perturbation. IEEE Trans. on Neural Networks, 2003, – No.14(3). – P. 688–695.
23. Reyneri L. M. Implementation Issues of Neuro–Fuzzy Hardware: Going Toward HW/SW Co design. IEEE Trans. Neural Networks, 2003. – No.14(1). – P. 176 – 194.

Received 2009 07 01

M. NirmalaDevi, N. Mohankumar, S. Arumugam. Modeling and analysis of Neuro–Genetic Hybrid System on FPGA // Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 8(96). – P. 69–74.

Simultaneous evolution of the architecture and adaptation of weights of an Artificial Neural Network is executed using Genetic Algorithm (GA) to overcome the local minima problem. Absence of learning unit simplifies the Very Large Scale Integration (VLSI) realization of evolved Neural Network (NN). Potential of the Neurohardware is tested on two benchmark circuits; Eight–bit even Parity function and nine–bit Character Recognition. Binary input facilitates the use of comparators instead of multipliers in the hidden layer neuron, reducing the hardware complexity. While evolving the parity function using GA, the number of hidden layer neuron is reduced to half, which in turn reduces the silicon area appreciably. Character Recognition Network converges faster with acceptable error. Simulated results ensure that the designed Neuro–Genetic Hybrid System is not only fast and accurate but also hardware friendly. Ill. 7, bibl. 23, tabl. 9 (in English; abstracts in English, Russian and Lithuanian).

М. Нирмаладеви, Н. Моганкумар, С. Арумугам. Моделирование и анализ FPGA неуро-генетических систем // Электроника и электротехника. – Каунас: Технология, 2009. – № 8(96). – С. 69–74.

Одновременная эволюция архитектуры и адаптация искусственных нейроновых сетей используется в генетическом алгоритме с целью решения местных. Условно маленьких проблем. Недостаток обучения компенсирует применение в развитых нейронных сетях технологии очень большой степени интеграции. Потенциальное “Neuro” техническое обеспечение тестировалось в двух испытательных цепях. Двоичная система облегчает применение компараторов вместо умножителей, раньше применявшихся в скрытом нейронным слое (уменьшается сложность аппаратуры). Применение генетического алгоритма расширяет масштаб функций, в два раза уменьшает количество скрытых нейронных слоев, что позволяет значительно уменьшить площадь силиция. Результаты моделирования приводят к тому, что спроектированная неврогенетически смешанная система бывает не только быстрая и точная, но и легко совместимая с техническим оборудованием. Ил. 7, библ. 23, табл. 9 (на английском языке; рефераты на английском, русском и литовском яз.).

M. Nirmaladevi, N. Mohankumar, S. Arumugam. FPGA neuroninių genetinių sistemų modeliavimas ir analizė // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2009. – Nr. 8(96). – P. 69–74.

Vienalaikė dirbtinių neuroninių tinklų architektūros ir adaptacijos evoliucija naudojama genetiniame algoritme siekiant įveikti vietines palyginti mažas problemas. Mokymų trūkumą kompensuoja labai didelės integracijos laipsnio technologijos taikymas išsivysčiusiuose neuroniniuose tinkluose. Potenciali techninė „Neuro“ įranga buvo testuojama dviejose bandomosiose grandinėse. Dvejetainė sąsaja palengvina taikyti komparatorius vietoj taikytų daugiklių paslėptame neuroniniame sluoksnyje (aparatūra prastinama). Taikant genetinį algoritmą plečiamas funkcijų mastas, paslėptų neuroninių sluoksnių kiekis sumažintas perpus, kas leidžia gerokai sumažinti silicio plotą. Modeliavimo rezultatai rodo, kad suprojektuota genetiniškai mišri neuro sistema veikia ne tik greitai ir tiksliai, bet ir yra lengvai derinama su technine įranga. Il. 7, bibl. 23, lent. 9 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).

DOI: 10.5755/j02.eie.9965