International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:2, No:1, 2008

Local Linear Model Tree (LOLIMOT) Reconfigurable Parallel Hardware

A. Pedram, M. R. Jamali, T. Pedram, S. M. Fakhraie, and C. Lucas

***Abstract*—**Local Linear Neuro-Fuzzy Models (LLNFM) like other neuro- fuzzy systems are adaptive networks and provide robust learning capabilities and are widely utilized in various applications such as pattern recognition, system identification, image processing and prediction. Local linear model tree (LOLIMOT) is a type of Takagi-Sugeno-Kang neuro fuzzy algorithm which has proven its efficiency compared with other neuro fuzzy networks in learning the nonlinear systems and pattern recognition. In this paper, a dedicated reconfigurable and parallel processing hardware for LOLIMOT algorithm and its applications are presented. This hardware realizes on-chip learning which gives it the capability to work as a standalone device in a system. The synthesis results on FPGA platforms show its potential to improve the speed at least 250 of times faster than software implemented algorithms.

***Keywords*—**LOLIMOT, Hardware, Neuro-Fuzzy Systems, Reconfigurable, Parallel.

I. Introduction

INTELLIGENT and highly autonomous systems are playing a great role in both industrial and academic settings of today. Artificial neural networks, fuzzy systems [1] and neuro fuzzy systems [2] ,[3], demonstrate some main fields of computational intelligence, which have many applications ranging from engineering to economics, from forecasting to control and system identification [4], [5]. These techniques provide powerful tools for non-parametric analysis of nonlinear systems model-free processing and control of uncertain systems and plants. In many cases, proofs that these systems are universal approximators and after learning act in asymptotically optimal manner in Bayesian senses are readily available. On the negative side, the black box and model-free approach underlying the use of these tools means that there are few theorems and methods for assuring stability and robustness or carrying out post optimality analysis on the total systems where these tools have been used. Despite these applications and facilities, many practical applications require a large computational power to overcome complexity or real-time constraints. Hardware implementation is a good solution for these restrictions [4], [7] and [8].

In order to take advantage of redundancies in the solution and improve the performance of these systems to let them take critical roles in real-time and dynamic system control and prediction and fully exploit the parallel nature of these types of fuzzy systems a dedicated hardware is needed. The hardware should also have high classification rate and be able to solve different types of problems with its reconfigurable architecture.

Generally there are three methods for training neuro-fuzzy systems: off-line, on-line and on-chip training. On-line training runs the learning with an algorithm on an external computer interfaced to the hardware device to be trained. The latter computes the feed forward step, while the external computer updates weights for training step. On-chip training trains the network on the chip itself, by means of appropriate circuits which must be implemented on the chip. No external computer is needed. This solution requires a more complex hardware [8].

In this paper, the hardware implementation of the LOLIMOT algorithm is presented [3]. This algorithm divides the input space into local linear models which has a higher performance and needs lower neuron count compared to normal neural networks in terms of learning a mapping of the input space [9]. This algorithm can be used in all TSK neuro-fuzzy networks with real-time applications such as [10] and [11], which persuade the need of dedicated hardware systems to implement this algorithm on it. The nature of the algorithm is based on matrix computations. Although, matrix computations are elaborated in several hardware implementations [12], there is not an efficient dedicated hardware implementation for LOLIMOT algorithm that gathers the matrix computation implementations in a single framework. On the other hand, exploiting the native parallelism of the whole LOLIMOT algorithm is the other main motive for implementation of a dedicated hardware. While preserving the ability of the single matrix operations in our proposed hardware system such as matrix multiplication and matrix inversion, also our implemented hardware system is a dedicated hardware realization of whole LOLIMOT algorithm. We employ fine and important hardware design techniques to improve the performance of the realized LOLIMOT hardware. It can be used as a matrix-computation system with the capability of realizing matrix multiplication and inversion and the other operations that can be categorized in this class. The processing engines of our LOLIMOT hardware system are design in a way that can bear the computation requirement of the on-chip learning. In fact, in the learning phase they are reconfigured to accomplish the computations required for learning from the input samples. One of the main advantages of our hardware realization is on-chip learning capability.

Manuscript received April 20, 2006.

A. Pedram, M. Jamali, C. Lucas, S. M. Fakhraie, Department of Electrical and computer Engineering University of Tehran, Tehran, Iran. (e-mail: a.pedram@ece.ut.ac.ir, m.jamali@ece.ut.ac.ir, tarannom.pedram@siemens.com, fakhraie@ut.ac.ir, lucas@ipm.ir).

II. Mathematical Description

The network output is calculated as a weighted sum of the outputs of the local linear models, where the validity function is interpreted as the operating point dependent weighting factors. The validity functions are typically chosen as normalized Gaussians.

The most important factor for the success of LOLIMOT is divide and conquer strategy that is used in it. LOLIMOT is an incremental tree-construction algorithm that partitions the input space by axis-orthogonal splits [3].

In this section, a mathematical formulation of LLNFM with LOLIMOT learning algorithm is described [3]. The fundamental approach with locally linear neuro-fuzzy models is dividing the input space into small subspaces with fuzzy validity functions. Any produced linear part with its validity function can be described as a fuzzy neuron. Thus the whole model is a neuro-fuzzy network with one hidden layer and a linear neuron in the output layer which simply calculates the weighted sum of the outputs of locally linear models. The Network structure is shown in Fig. 1.

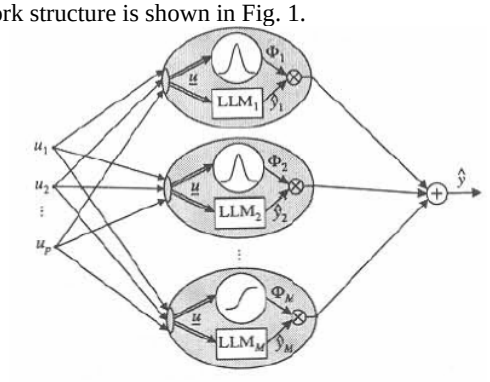

Fig. 1 Network structure of a static local linear neuro-fuzzy with M neurons for p inputs [4]

In (1) input-output relation of LLNFM is represented. In this formula M is number of neurons, $\underline{u} = [u_1 \;\; u_2 \;\; ... \;\; u_p \;\;]^T$ is the model input, p is number of input dimension, N is the number of input samples and ω_{ij} denotes the weights of i th neuron [3].

$$\hat{y}_i = \omega_{i0} + \omega_{i1}u_1 + \omega_{i2}u_2 + \omega_{i3}u_3 + ... + \omega_{ip}u_p \quad , \quad \hat{y} = \sum_{i=1}^{M} y_i \Phi_i(\underline{u}) \tag{1}$$

The validity functions are chosen as normalized Gaussians. Normalization is necessary for a proper interpretation of validity functions. In (2) and (3) validity function formulation is represented.

$$\Phi_i(\underline{u}) = \frac{\mu_i(\underline{u})}{\sum_{j=1}^{M} \mu_j(\underline{u})} \tag{2}$$

$$\mu_i(\underline{u}) = \exp\left(\frac{(u_1 - c_{i1})^2}{-2\sigma_{i1}^2}\right) \times \cdots \times \exp\left(\frac{(u_p - c_{ip})^2}{-2\sigma_{ip}^2}\right) \tag{3}$$

Each Gaussian validity function has two parameters; centre c_{ij} and standard deviation σ_{ij} .Also there are $M \times p$ weight parameters of the nonlinear hidden layer. Optimization or learning methods are used to adjust fine tuning of two sets of parameters, ω_{ij} weights and the parameters of validity functions.

Local optimization of linear parameters is simply obtained by Least squares technique. The global parameter vector contains $M \times (p+1)$ elements. In (4) these parameters are shown.

$$\underline{\omega} = \lfloor \omega_{10} \;\; \omega_{11} \;\; .. \omega_{1p} \;\; \omega_{20} \;\; \omega_{21} \;\; .. \omega_{M0} \;\; .. \omega_{Mp} \;\rfloor \tag{4}$$

Associated regression matrix X for N measured data samples is formulated in (5) and (6). Thus the weights will be obtained by solving (7) and (8) as shown in (9).

$$\underline{X} = [\underline{X}_1 \underline{X}_2 ... \underline{X}_M] \tag{5}$$

$$\underline{X}_i = \begin{bmatrix} 1 & u_1(1) & ... & u_p(1) \\ 1 & u_1(2) & ... & u_p(2) \\ \vdots & \vdots & & \vdots \\ 1 & u_1(N) & ... & u_p(N) \end{bmatrix} \tag{6}$$

$$\underline{\hat{y}} = \underline{X} Q \underline{\hat{\omega}} \tag{7}$$

$$Q_i = \begin{bmatrix} \Phi_i(\underline{u}(1)) & 0 & ... & 0 \\ 0 & \Phi_i(\underline{u}(2)) & ... & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & ... & \Phi_i(\underline{u}(N)) \end{bmatrix} \tag{8}$$

$$\hat{\omega} = (X^T Q X)^{-1} X^T Q y \tag{9}$$

Several methods can be used to optimize the premise rules. Tree based methods are appropriate for their simplicity and intuitive constructive algorithm. LOLIMOT is an incremental based on three iterative steps: first the worst Local Linear Model is defined according to local loss functions. This LLM neuron is selected to be divided. In the second step all divisions of this LLM on input space are constructed and checked. Finally the best division for the new neuron must be added. For further information about LOLIMOT algorithm refer to [3]. In Fig. 2 the first five iterations of LOLIMOT algorithm for a two dimensional input space is shown.

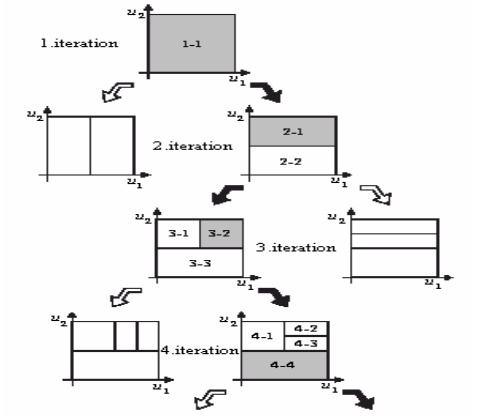

Fig. 2 Operation of the LOLIMOT algorithm in the first five iterations for a two dimensional input space [4]

The computation complexity of LOLIMOT grows linearly with number of neurons. This computation complexity is comparable with other algorithms [3]. Time complexity of LOLIMOT is represented in (10).

$$O(2Mp(p+1)^3) \approx O(2Mp^4) \quad (10)$$

The remarkable properties of locally linear neuro-fuzzy model, its transparency and intuitive construction, lead to the use of least squares for rule antecedent parameters and incremental learning procedures for rule consequent parameters [3].

A. Complexity Analysis

The main complexity of LOLIMOT algorithm is to calculate Equation (10) for $p \times \left\lfloor \frac{M+1}{2} \right\rfloor \times M$ times, which by itself consists of different types of matrix computations such as matrix-matrix multiplication, matrix-vector multiplication and matrix inversion. It is noticeable that the matrix in (9) is a diagonal matrix which makes $(\underline{X}^T_{(p+1)\times N} Q_{N\times N})$ a simple multiplication of each column of X^T by the same diagonal index element of matrix *Q*, this means (11) is a matrix power operation on matrix X^T with extra multiplication of diagonal element of matrix *Q* into each column of X^T. In this paper, we name the operation (11) "*pow*." The main computations of the algorithm take place in (11) and (12).

$$(\underline{X}^T_{(p+1)\times N} Q_{N\times N} \underline{X}_{N\times(p+1)})_{(p+1)\times(p+1)} \quad (11)$$

$$(\underline{X}^T_{(p+1)\times N} Q_{N\times N} \underline{y}_{N\times 1})_{(p+1)\times 1} \quad (12)$$

The *pow* is a Symmetric matrix, so the computation of *pow* is reduced to about half of the number of elements of the result matrix. Consider that pow_{ij} is the i *th* row and j *th* column element of *pow* which is computed with the Equation (13).

$$pow_{ij} = \sum_{k=1}^{N} X_{ki} \times X_{kj} \times Q_{kk} \quad (13)$$

Equation (13) indicates that the matrix power can be computed in N steps such that in the l *th* step, the l *th* sample vector (l *th* row) of matrix X and the l *th* diagonal element of matrix Q are needed to produce $X_{li} \times X_{lj} \times Q_{ll}$ for different i's and j's. All step's results are added together to make the final result of *pow*. Total amount of computations in each step is about $\frac{(p+1)\times(p+2)}{2}$ multiply accumulate operation which is the number of different combinations of X_{li} and X_{lj}, In the other hand, elements of matrix in (12) can be computed with the same manner shown in Equation (14) ,

$$(\underline{X}^T_{p\times N} Q_{N\times N} \underline{y}_{N\times 1})_{i1} = \sum_{k=1}^{N} X_{ki} \times Q_{kk} \times y_{k1} \quad (14)$$

B. Matrix Inversion

The second issue in computing (10) is to compute $(\underline{X}^T Q \underline{X})^{-1}$ or the inverse of *pow*. While matrix power is a Symmetric Positive Definitive (SPD), its inverse can be computed by decomposing it into its Cholesky factor. Instead of seeking arbitrary lower and upper triangular factors **L** and **U**, Cholesky Decomposition constructs a lower triangular matrix **L** whose transpose $\mathbf{L}^T$ can itself serve as the upper triangular part [13]. In other words $pow=L.L^T$ and $pow^{-1}=L^{-T}.L^{-1}$. The *pow* is a $(p+1)\times(p+1)$ matrix and is very small compared to matrix X so computing its inverse is not a bottleneck in this problem, while Matrix inversion can be solved by matrix multiplication engine in $O(n^3)$.

C. Updating the Coefficients

The μ of new neurons have just a difference in one of the dimensions or exponential phrases. For example if the x th dimension is the dimension which should be divided into two neurons. The new μ s can be computed as presented in Equations (15) to (18).

$$temp = \frac{\mu_{old}}{\exp\left(\frac{(u_x - c_{ix})^2}{-2\sigma_{ix}^2}\right)} \quad (15)$$

$$\mu_{i(new)} = temp \times \exp\left(\frac{(u_{x1} - c_{ix1})^2}{-2\sigma_{ix1}^2}\right) \quad (16)$$

$$\mu_{M+1} = temp \times \exp\left(\frac{(u_{x2} - c_{ix2})^2}{-2\sigma_{ix2}^2}\right) \quad (17)$$

$$\left(\sum_{j=1}^{M+1} \mu_j(\underline{u})\right)_{new} = \left(\sum_{j=1}^{M} \mu_j(\underline{u})\right)_{old} - \mu_{i(old)} + \mu_{i(new)} + \mu_{M+1} \quad (18)$$

For other neurons just the normalization dividend is changed and as described in Section 2 the input matrix of each Neuron is the same as others. The only difference is the matrix Q_i which is different for neurons and as shown in Figure 4. Each diagonal element of Q_i matrix is fed to the processing group just in one clock cycle and the other inputs are matrix X's elements.

III. Architecture Design

This design can be mapped in to a scalable and also reconfigurable array of processing elements by providing suitable connections between processing elements in order to take advantage of data dependencies. The key idea in the new design is to fill the presented design with additional PEs and make a matrix of processing engines which is in a regular arrangement of processing elements. Then the matrix of PEs is reduced to a chain of PEs and the computations of each step are done in multiple cycles. This new method is presented in theorem 1.

Theorem 1: Total computations of matrix power for matrix X_{pp} (*p* is even) can be done with *p/2* processing units and a multiplier in *p/2+1* steps and data rate is two elements per each step. PEs are connected in an array, each PE contains two multipliers, two adders and two *(p/2+1)* registers.

Proof:

The key idea in Theorem 1 is to bring *p/2* of the *N th* sample at first step and compute their multiplication and then bring each of the other *p/2* elements in a step and let the processing units pass their input data to their neighbours in order to reuse it.

Each result of the MAC unit is saved in its own register and will be accumulated with the next *p* dimensional sample temporary value. The total amount of computations in *(p/2+1)* clock cycles is equal to the desired amount of computation to calculate matrix *pow* in each step, which is $(\frac{p}{2}+1)+(\frac{p}{2}+1)\times\frac{p}{2}\times 2=\frac{p+2}{2}\times(p+1)$ as shown in Fig. 3. Just one additional step is needed to compute X^TQy in which the input of all MACs is *y*.

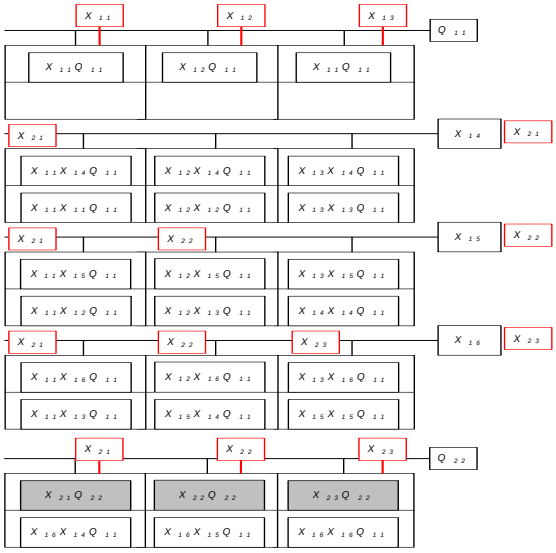

Fig. 3 Input Data Sequence in one step for p= 6

Each Mac unit can be pipelined in order to increase the clock frequency. The number of pipeline stages of MAC units can be *p/2+1* stages while each result will be used to be added to the next sample's respective result which is ready *p/2+1* clock cycles later.

This hardware also allows parallel matrix multiplication. The input data rate is again two elements per each step and no hardware overhead is forced. The difference is that there is no data transfer between processing elements and all elements work independently. Each step in matrix multiplication can be *p/2* or *p* clock cycles which depend on the number of registers in each processing unit. The architecture of each PU is displayed In Fig. 4.

The whole system for LOLIMOT contains a few processing unites (PU)s ; each unit contains some processing engines and one multiplier. These units are able to be combined and make a bigger group or work standalone with same input data. The system can support any number of *p* but maximum utilization of processing units is for *p* ranged between 8 and 32. It contains a distributed cache and an input cache, both connected to processing engines with a common bus. The distributed cache contains the temporary results and is distributed among processing groups. These two buses give the ability to the system to do different types of computations concurrently in system.

IV. Experimental Results

In this section software analysis of the main three factors of the network effects are discussed for a Speed Identification of Induction Motor by means of d-q axis Stator Current and Voltage[14].The duration of training run on the general purpose computer with software and our proposed hardware are compared too. It is noticeable that the reconfigurable design of this hardware utilizes the processing elements for every input dimensions.

The fixed parameters of the network are M=50 neurons, P=8 dimensions, and N=200 input samples. Each time one parameter is changed and two others are kept fixed. The feed-forward training duration shown in the tables are multiplied by 1000.

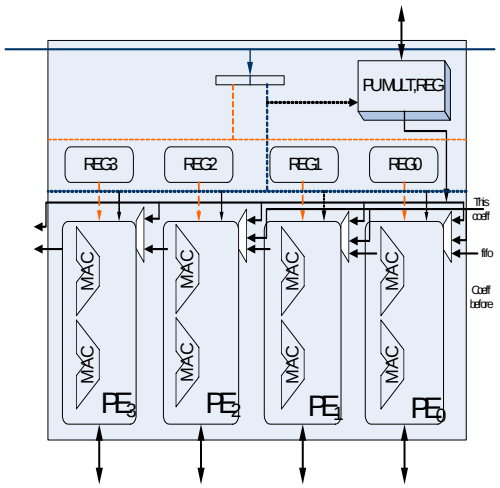

Fig. 4 PU architecture

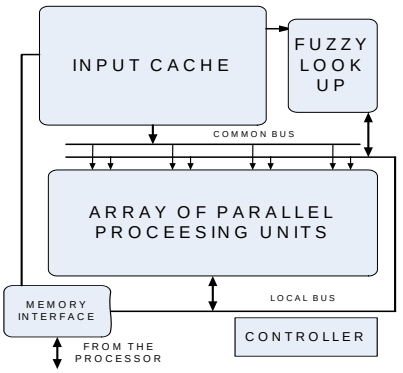

Fig. 5 LOLIMOT engine is offered as an IP core

In Fig. 6 the effect of dimension of input data is shown. The amount of complexity grows exponentially when the dimension is increased.

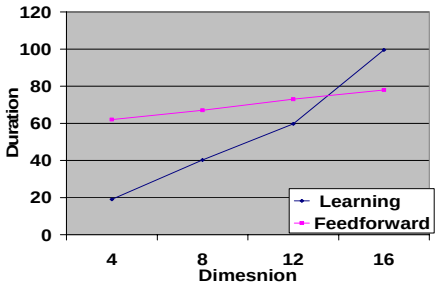

Fig. 6 Variation of training time with input dimension

In Fig. 7 the effect of input samples number on duration is shown.

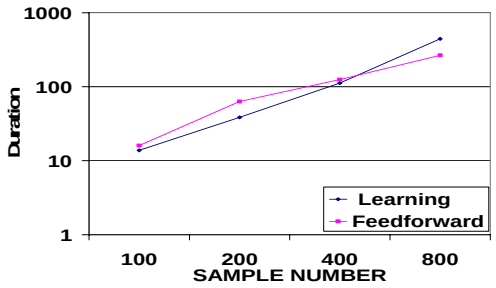

Fig. 7 Variation of training time with size of input space

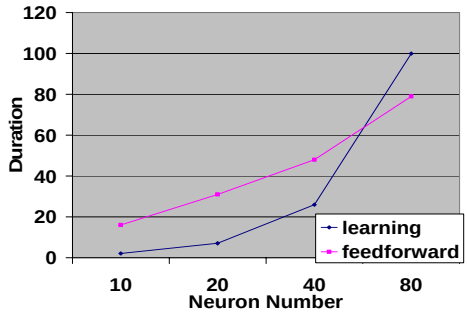

Fig. 8 Variation of training time with the number of Neurons

TABLE I
SYNTHESIS RESULTS FOR SOME STRATIX FPGA DEVICE

	Logic Cell Usage	Memory Bit Usage	Clock Frequency	Device	Total No of LCs	Total No of Memory Bits
1PU-8bit	16.65 %	0.45%	92.5 MHz	EP1S10F 780C	10570	920448
1PU-16bit	52.38 %	0.89%	76.4 MHz	EP1S10F 780C	10570	920448
1PU-32bit	46.84 %	0.48%	63.3 MHz	EP1S40F 1020C	41250	342374 4

The last factor is the number of neurons in the network its effect behaves different in training phase and in feed-forward phase. In the training phase the duration grows exponentially but in feed-forward it grows linear when the number of neurons is increased. The results are shown in Fig. 8.

The synthesis results on Stratix FPGA are presented in Table I and the total amount of clock cycles for training the algorithm are presented in Table II.

TABLE II
NUMBER OF CLOCK CYCLES USED FOR TRAINING OF A SINGLE PU SYSTEM

M=20	p=4	p=8	p=16	p=32
N=200	776K	2038K	13841K	84552K
N=1000	3384K	11806K	70091K	447560K

Form Table II and from Fig. 8 we can pick N=200 M=20 and p=8 to compare the duration of training the network with our proposed design with 16 bit fixed point words and the general purpose computers.

About 2038 thousand clocks are needed at 76.4 Mhz speed which takes 0.026 seconds for training the network with 1-PU system and 200 input samples. In the other hand computing the network with software lasts 7.06 seconds which is about 271 times longer. It is obvious that if the number of Pus in the system increases the duration time is divided by their number. At least there are 2 PUs in the system which can improve the training almost two times faster.

V. CONCLUSION

In this paper, hardware implementation of local linear neuro-fuzzy model with LOLIMOT algorithm was presented. The learning method used in our system was an on-chip implementation of LLNFM. The architecture supports both feed-forward phase and leaning phase computations with reconfigurable processing engines. The architecture supports matrix inversion, multiplication and matrix power. Synthesis results showed that the hardware overhead for learning phase is acceptable. Despite this low overhead, the engines alleviate the on-chip learning problem with their high performance which is at least 250 times faster. Our hardware is designed and implemented in a reusable manner which can be employed in many other neuro-fuzzy systems that require high-performance matrix computations.

REFERENCES

[1] T. Takagi, M. Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” *IEEE Tran. Systems, Man and Cybernetics*, vol. 15, pp. 116-132, 1985.
[2] J. R. Jang, “ANFIS: Adaptive network based fuzzy inference system,” *IEEE Tran. Systems, Man and Cybernetics*, vol. 23, no. 3, 1993, pp. 665-685.
[3] O. Nelles, *Nonlinear system identification*. Berlin: Springer Verlag, 2001.
[4] C. Lucas, R. M. Milasi and B. N.Araabi, "Intelligent modeling and control of washing machine using LLNF modeling and modified BELBIC," *Asian Journal of Control*, vol.8, no.4, December 2005.
[5] O. Nelles, “Local linear model tree for on-line identification of time variant nonlinear dynamic systems,” *International Conference on Artificial Neural Networks (ICANN)*, pp. 115-120, Bochum-Germany, 1996.
[6] L. M. Reyneri, “Implementation issues of neuro-fuzzy hardware: Going toward hw/sw codesign,” *IEEE Trans Neural Networks*, vol. 14, no. 1, pp. 176–194, Jan. 2003.
[7] S. M. Fakhraie and K. C. Smith, *VLSI-Compatible Implementations for Artificial Neural Networks*. Norwell, Massachusetts: Kluwer Academic Publishers, 1997.
[8] L. M. Reyneri, Neuro-fuzzy hardware: design, development and performance, in *Proc. Of FEPPCON III*, Skukuza (South Africa), 12-15 July 1998.

[9] O. Nelles, “Nonlinear system identification with local linear neuro-fuzzy models,” PhD Thesis, TU Darmstadt, Shaker Verlag, Aachen, Germany, 1999.
[10] C. H. Lee, W. Y. Lai, C. C. Chen, “Lossless image coding via adaptive Takagi-Sugeno fuzzy neural network predictor”, Proc. of the. *2004, IEEE. International Conference on Networking, Sensing. &. Conrrol. Taipei,. Taiwan, March. 21-23, 2004*.
[11] I. Nedeljkovic, “Image classification based on fuzzy logic,” *in Proc. Geo-Imagery Bridging Continents XXth ISPRS* Congress, 12-23, Turkey, Istanbul, july, 2004.
[12] W. Xiaofang, S. G. Ziavras,” Performance optimization of an FPGA-based configurable multiprocessor for matrix operations,” *Proc. 2003 IEEE International Conference on Field-Programmable Technology (FPT)*, pp 303 – 3-6, dec. 2003.
[13] B. P. Flannery, *Numerical Recipes in C/C++*, William H. Press, 2005.
[14] T. Abbasian, F.R. Salmasi, M.J. Yazdanpanah, “Stability analysis of sensorless IM based on adaptive feedback linearization control with unknown stator and rotor resistances,“ *Industry Applications Conference, 2005. Fourtieth IAS Annual Meeting*, vol. 2, pp 985 – 992, Oct. 2005.