Hardware Implementation of CMAC Type Neural Network on FPGA for Command Surface Approximation

Sándor Tihamér Brassai, László Bakó

Sapientia - Hungarian University of Transilvania, Faculty of Technical and Human Sciences, Târgu–Mureş, Romania, 547367 Corunca, Şos. Sighişoarei 1C

E-mail: tiha@ms.sapientia.ro, lbako@ms.sapientia.ro

Abstract: The hardware implementation of neural networks is a new step in the evolution and use of neural networks in practical applications. The CMAC cerebellar model articulation controller is intended especially for hardware implementation, and this type of network is used successfully in the areas of robotics and control, where the real time capabilities of the network are of particular importance. The implementation of neural networks on FPGA's has several benefits, with emphasis on parallelism and the real time capabilities.This paper discusses the hardware implementation of the CMAC type neural network, the architecture and parameters and the functional modules of the hardware implemented neuro-processor.

Keywords: Neural networks, CMAC, Neural networks hardware implementation, FPGA

1 Introduction

Great interest has been manifested lately for the utilization of adaptive modeling and control, based on biological structures and learning algorithms. Control systems need to have high dynamic performance and robust behavior. These controllers are expected to cope with complex [1], uncertain and nonlinear dynamic processes. It is difficult to obtain a mathematical representation of uncertain and nonlinear dynamic processes that impose an intelligent modeling and control. For static system modeling one can use feed-forward static networks like Multi-Layer Perceptrons (MLP), Radial Basis Function (RBF). For dynamic system modeling, neural networks that show temporal behaviour can be used.

The major disadvantage of MLPs and the MLP-based dynamic networks is their slow training algorithm. This drawback may be an obstacle to apply them for real-time adaptive modeling problems.

Using networks with only a single trainable layer, the learning speed can be significantly increased.

CMAC (Cerebellar Model Articulation Controller) and RBF (Radial Basis Function) are networks with a single trainable layer and have better capabilities than multi-layer perceptrons. The most important properties of the CMAC type controller are the fast learning capability and the special architecture that allows digital hardware implementation.

2 CMAC Network

Cerebellar Model Articulation Controller networks play an important role in non-linear function approximation and system modeling. The main advantages of CMAC type networks compared to MLP, are their extremely fast learning and the possibility of low-cost digital implementation.

The CMAC network can be represented as a three layer system (Figure 1) with a normalized input space, basis functions, and output, and can be considered as an associative memory, which realizes two subsequent mappings.

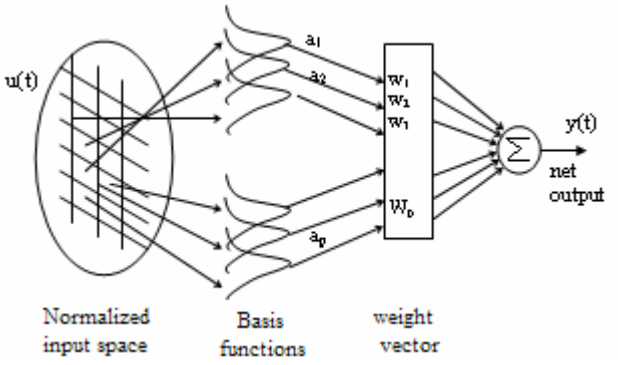

Figure 1
CMAC network as a three layer system

The first one - which is a non-linear mapping - projects an input space point **u** into a binary association vector **a.**

The association vectors always have C active elements, which means that C bits of an association vector have the value ‘1’ and the others have the value ‘0’. C is an important parameter of the CMAC network and it is much less than the length of the association vector (Figure 2).

In practical applications the two mappings are implemented by a two-layer network. The first layer is responsible for mapping the input points to the association vectors; this mapping is fixed (can be wired in hardware).

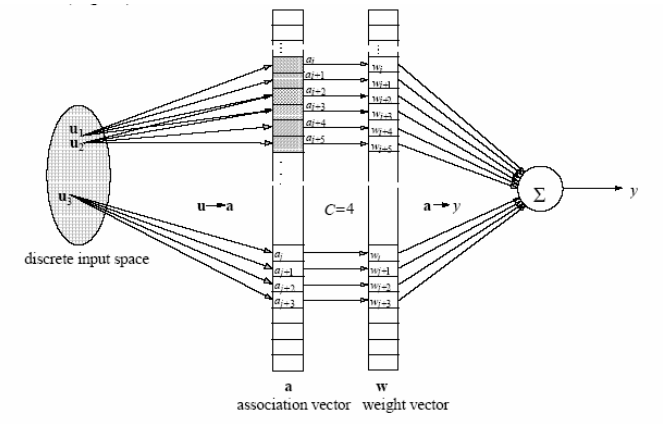

Figure 2
The mapping of a CMAC The trajectory tracking

The second layer is trainable and realizes a linear combination of the weight vector and the association basis function vector. The input variables are divided into overlapped regions and every region is subdivided into quantization intervals. The output value for an input point can be considered as a weighted sum of selected basis functions [2]. The resolution of the network and the shift positions of the overlapping regions are determined by this quantization.

A given value of an input variable activates all the regions where the input is within a quantization interval of a region.

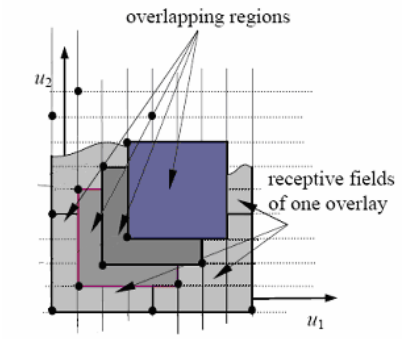

Figure 3
The receptive fields of a two-variable Albus CMAC

Every element in the association vector corresponds to a basis function. Each basis function has a so-called receptive field.

The shaded regions in Figure 3 are receptive fields of different basis functions [3]. If an input point is within a receptive field, the corresponding basis function is selected. The basis functions are grouped into overlays. One overlay contains basis functions with non-overlapping supports, but the union of the supports covers the whole input space. The different overlays have the same structure; they consist of similar basis functions in shifted positions. The positions of the overlays and the basis functions of one overlay can be represented by definite points.

In the original Albus scheme the overlay-representing points are in the main diagonal of the input space, while the basis function positions are represented by the sub-diagonal points as it is shown in Figure 3. The overlay representing points can be described as displacement vectors the elements of which are the coordinates of the definite points. Every input data will select C basis functions, each of them on a different overlay, so in an overlay one and only one basis function will be active for every input point. As every selected basis function will be multiplied by a weight value, the size of the weight memory is equal to the total number of basis functions or to the length of the association vector. As an element of the association vector can be considered as the value of a basis function for a given input, the output of the binary basis function is one if an input is in its receptive field and zero elsewhere:

$$a_i(u) = \begin{cases} 1 & if\ u\ is\ the\ receptive\ field \\ of\ the\ i-th\ basis\ function & \\ 0 & otherwise \end{cases} \tag{1}$$

The application of such functions means that if the basis functions are selected, the sum of the corresponding weights will form the network's output value independently of the exact position of the input point in the receptive fields of the basis functions:

$$y(u) = \sum_{i=1}^{C} a_i(u) w_i \tag{2}$$

The output of the network can be obtained without the need of any multiplication in the case of the binary basis function. As the output layer is the only trainable layer of the CMAC, and as this output layer performs linear operations, the simple LMS algorithm can be applied to train the network:

$$w(k+1) = w(k) + \mu\varepsilon(k)a(k) \tag{3}$$

where k is the discrete time index, ε(k) is the error of the network and a(k) is the association vector at step k. As it is a sparse binary vector only those weights will be modified during the training process which take part in the forming of the output value.

3 Parallel Implementations of Neural Networks

Fast implementations of neural network applications are useful because of the very high number of required arithmetic operations. Such implementations might use massively parallel computers as well as digital or analog hardware designs. This section briefly discusses the use of the various possible parallel devices.

• General purpose parallel computers. Fine-grain parallel implementations on massively parallel suffer from the connectivity of standard neural models which results in costly information exchanges. Coarse grain parallel implementations are mainly applied to neural learning, so that their efficiency suffers from the sequentiallity of standard learning algorithms such as stochastic gradient descent.

• Dedicated parallel computers. Neuro-computers are parallel systems dedicated to neural computing. They are based on computing devices such as DSPs (digital signal processors), or neuroprocessors. Their use suffers from their cost and their development time: they rapidly become out-of-date, compared to the most recent sequential processors. Most well-known neurocomputers are described in [5, 6].

• Analog ASICs. Many analog hardware implementations have been realized. They are very fast, dense and low-power, but they introduce specific problems, such as precision, data storage, robustness. On-chip learning is difficult.

• Digital ASICs. Many digital integrated circuits have also been designed for neural networks. Compared to analog chips, they provide more accuracy, they are more robust, and they can handle any standard neural computation. They usually implement limited parts of neural networks, so as to be included in neuro-computer systems ([4, 8]).

• The FPGA solution. The appearance of programmable hardware devices, algorithms may be implemented on very fast integrated circuits with software-like design principles, whereas usual VLSI designs lead to very high performances at the price of very long production times (up to 6 months).

FPGAs, such as Xilinx FPGA ([9]), are based on a matrix of configurable logic blocks (CLBs). Each CLB contains several logic cells that are able to implement small logical functions (4 or 5 inputs) with a few elementary memory devices (flip-flops or latches) and some multiplexers. CLBs can be connected thanks to a configurable routing structure. In Xilinx FPGAs, CLBs can be efficiently connected to neighbouring CLBs as well as CLBs in the same row or column. The configurable communication structure can connect external CLBs to input/output blocks (IOBs) that drive the input/output pads of the chip.

An FPGA approach simply adapts to the handled application, whereas a usual VLSI implementation requires costly rebuildings of the whole circuit when changing some characteristics. A design on FPGA requires the description of several operating blocks. Then the control and the communication schemes are

added to the description, and an automatic 'compiling' tool maps the described circuit onto the chip. Therefore configurable hardware appears as well- adapted to obtain efficient and flexible neural network implementations.

Neural Networks on FPGAs: Specific Assets

As stated above, FPGAs offer a cheap, easy and flexible choice for hardware implementations. They also have several specific advantages for neural implementations:

- Reprogrammable FPGAs permit prototyping
- FPGAs may be used for embedded applications, when the robustness and the simplicity of neural computations is most needed, even for lowscale productions.

4 Implemented CMAC Network Block Modules

The most important characteristics are the number of inputs, inputs' limits, number of internal points, generalization parameters, the basis function's type. For a better use of the capacity of the FPGA instrument the above mentioned parameters are a multiple or exponent of two.

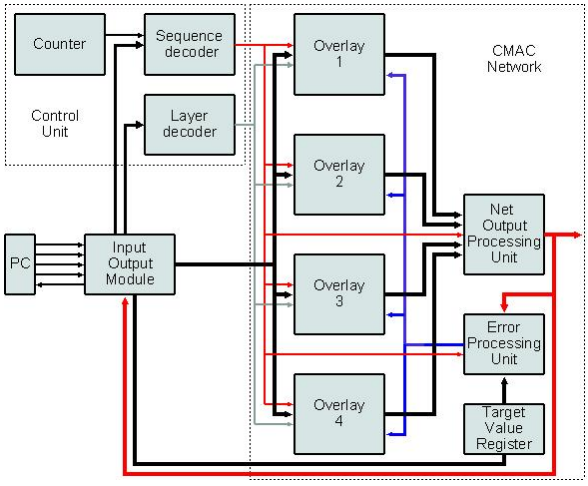

Figure 4
Implemented CMAC network block schemtic

The implemented network consists of the following main modules: the input/output module, the control module, and the network itself. The input output module assures the weight initialization, the data introduction and extraction. The control unit drives the network both in the output elaboration and the training

phase. The network itself is composed of several functional subunits, which were separately designed in VHDL. The modular design assures a network with a high flexibility and easy manageability. Figure 4 shows the implemented CMAC network block's schematic.

4.1 The Input/Output Module

The input/output module: Due to the fact that, by construction, the utilized FPGA development board uses the PC parallel port for downloading the configuration bit-stream, the easiest and most forthcoming way to exchange data with the CMAC network also is to use the same port. The issue that arises by doing so, is the limited number of bits available, which requires a custom serial protocol to be put in place. Hence, a synchronous, full-duplex serial communication module, the input/output module has been implemented. It manages the delivery of the initial weight values, the network input and the retrieval of the network output.

4.2 The Control Unit

The control unit is composed of a binary counter a sequence decoder and a layer decoder. This unit elaborates the different signals to control the network in different phases.

For each training point a time step is composed of seven cycles. Four cycles are used to compute the neural network output and three to calculate and update the new weight values (Figure 5). The computation cycles take place after the input and target values have been uploaded.

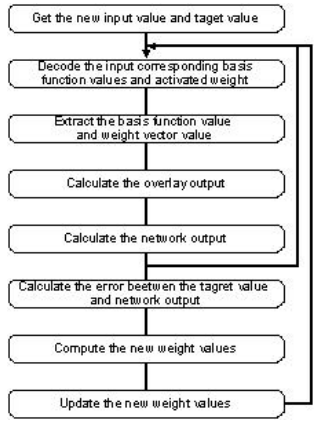

Figure 5
Flowchart of the neural network computation

4.3 CMAC Network Overlay Module

Two versions of the network have been developed. In the first version the weight values were stored in internal registers and the used triangular basis functions were implemented using logical elements (Figure 6). The main drawback of this approach is the fact that the most of the flip-flop type resources of the FPGA have been consumed by the weight storage. As a result it was impossible to implement large networks by this means.

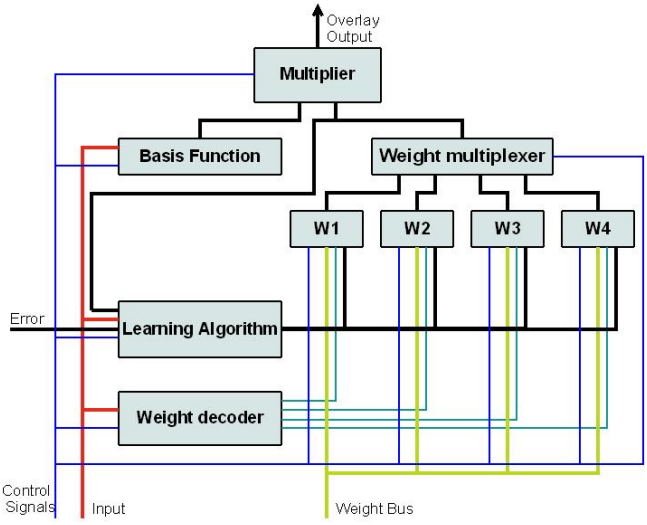

Figure 6
Overlay module structure I

In the second version of the network architecture we used Block RAMs to store weights and the basis function values (Figure 7).

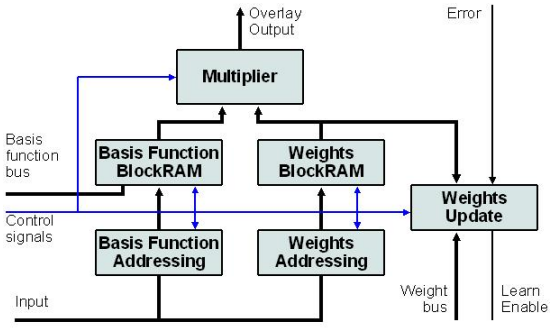

Figure 7
Overlay module structure II

By applying this change of FPGA resource utilization strategy, several advantages have emerged. The implementation has become more flexible as several basis function types can be now easily programmed.

The weight address and basis function address can be easily computed by integer division and by the rest of the integer division. In our Block RAM implementation the above mentioned operation can smoothly be implemented by selecting the higher or the lower bits of network input value.

In the case of multiple inputs the weight addressing is accomplished by simple mapping of the weights addresses on higher and lower memory address positions.

5 Experimental Results

These modules were developed in VHDL description language without using any schematics. The implemented networks were parameterized. Any other network can be generated by modifying the network parameters defined in the top level module.

For an easy development and tests an interface was created in Matlab with a driver implemented in Visual C++. The Visual C++ module contains the following functions: basis function upload, weights' initial value upload, inputs and target value uploads, the network output download to PC which, were accessed from Matlab.

Multiple tests and multiple networks with different parameters were tested. Table 1 contains the network parameters used in the performed experiments. The figures in the following section present some measurement results. These figures were recorded as results of a network with two input variables.

Table 1

Network parameters

Parameter name	Parameter value
Number of inputs	2
Number of bits per input variable	8
Basis functions receptive filed dimension	32x32
Number of overlays	4
Number of bits for weight value	6
Number of bits for basis function values	4

In the next figures 3D plots are presented for two target surfaces (Figure 8 and Figure 13), with a two variable function approximation. The subsequent figures contain a few samples of the learning process, starting with the response given using randomly initialized weights (Figure 9 respectevly Figure 14). As one can easily see, how the network gradually learns its tasks (in Figures 10, 11 for the

first surface and in Figures 15, 16 for the second surface) and how the error decreases (Figures 12 and 17). It should be mentioned that all parameters use integer representation. Obviously better accuracy could be achieved by using floating point or fixed point arithmetics.

Table 2
Resource utilization

Nr of resources	Used	Available	Used %
Slices	132	7680	1.7
Slice Flip Flops	107	15360	0.7
4 input LUTs	232	15360	1.5
Bonded IOBs	65	173	37
BRAMs	8	24	33
MULT18X18s	4	24	16.5
GCLKs	3	8	37

As it can be seen from Table 2, the resource utilization of the current implementation is very low, a version with higher precision would comfortably fit on the FPGA circuit.

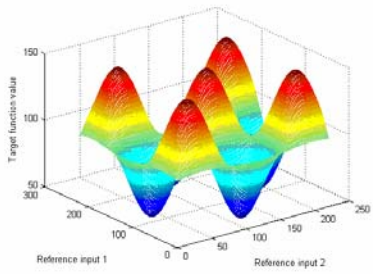

Figure 8
Reference surface

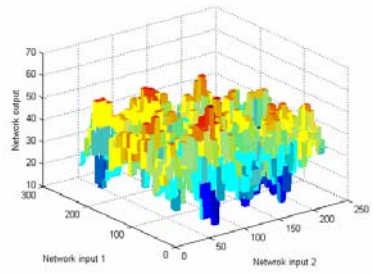

Figure 9
Initial form of surface

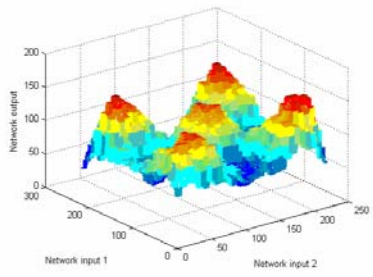

Figure 10
Learned surface after 20 epochs

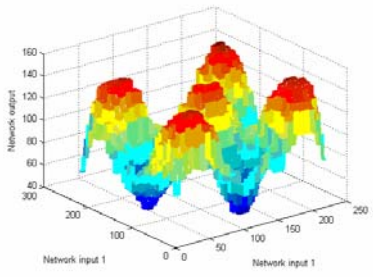

Figure 11
Learned surface after 40 epochs

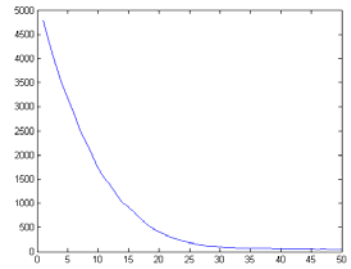

Figure 12
Squared aproximation error

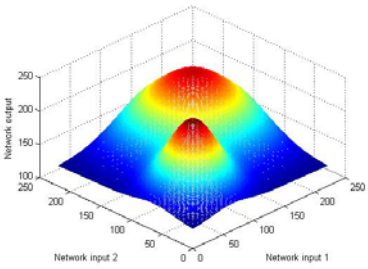

Figure 13
Reference surface

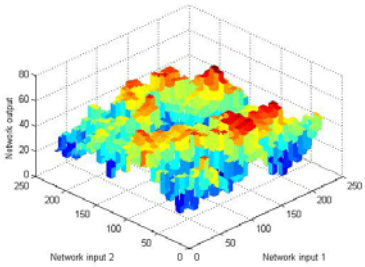

Figure 14
Initial form of surface

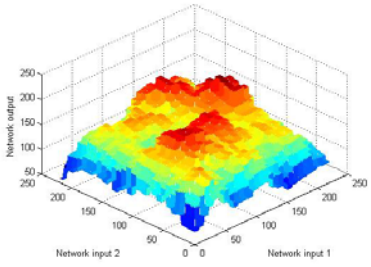

Figure 15
Learned surface after 40 epochs

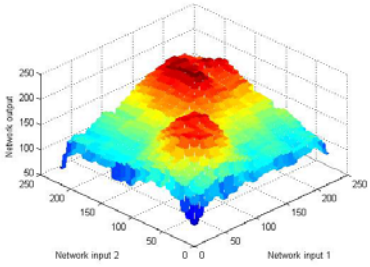

Figure 16
Learned surface after 100 epochs

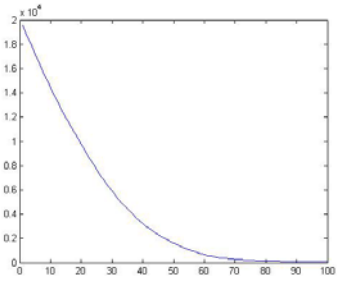

Figure 17
The trajectory tracking

Conclusions

A CMAC type hardware implemented network with one and two inputs has been developed. Due to the nature of the platform (FPGA), a very flexible architecture took shape, where most of the parameters can be modified. In the hardware implemented CMAC controller the follow error can be decreased by increasing the number of bits used for parameter representation, and for input coding.

Using an FPGA with more resources, the presented controller can easily be modified to use more than two inputs. The developed network is very fast, in 8 clock cycles it can obtain the network output. One of the main novelties of this implementation is that on-chip dynamic learning is performed without significant loss of efficiency and precision while maintaining reasonably low FPGA resource utilization.

References

[1] J. S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," Transaction of the ASME, Sep, 1975, pp. 220-227

[2] Horváth Gábor, Neuralis hálozatok és műszaki alkalmazások, Műegyetemi Kiadó, Budapest

[3] Horváth, G. Szabó, T. “Kernel CMAC With Improved Capability”, IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume: 37, Issue: 1, 2007, pp. 124-138, ISSN: 1083-4419

[4] W. Eppler, T. Fisher, H. Gelnmeke, T. Becher, G. Kock. High Speed Neural Network Chip on PCI-Board. In Proc. MicroNeuro, 1997, pp. 9-17

[5] T. Nordstrem, B. Svensson. Using and Designing Massively Parallel Computers for Artificial Neural Networks. Journal of Parallel and Distributed Computing, 14(3), 1992, pp. 260-285

[6] M. Schaefer, T. Schoenauer, C. Wo I ff, G. Hartmann, H. Klar, U. Ruckert. Simulation of Spiking Neural Networks - Architectures and Implementations. Neurocomputing, 2002, (48), pp. 647-679

[7] Brassai Sándor Tihamér, Dávid László, Bakó László, Hardware Implementation of CMAC-based Artificial Network with Process Control Application, Timişoara, Transaction on Electronics and communication, Scientific buletin of the „Politehnica” University of Timisoara, 2004, pp. 209-213, ISSN 1583-3380

[8] J. Wawrzynek, K. Asanovi~, N. Morgan. The Design of a Neuromicroprocessor. IEEE Trans. on Neural Networks, 1993, 4(3):394-399

[9] Xilinx, editor. The Programmable Logic Data Book. Xilinx, 2002