On the Implmentation of a Fuzzy CMAC

Yu Changbin and Abdul Wahab
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{Brad, Abdulwahab}@pmail.ntu.edu.sg

Abstract

This paper presents a Novel Fuzzy Cerebellar Model Articulation Controller (CMAC) implementation using Field Programmable Gates Array (FPGA) available in a flexible software/hardware co-design platform. The novel fuzzy quantization technique used for reducing CMAC memory requirement is similar to Discrete Incremental Clustering (DIC), with modifications to meet the hardware constraint.

The objective of the project is therefore twofold: to propose a hardware-friendly fuzzy clustering technique based on DIC, to implement the Fuzzy CMAC in hardware cost-saving manner and exploit realization of its inherent parallelism in FPGA architecture.

Test result for classification of 2-spiral problem is presented, which demonstrates the validity and generalization capability of proposed architecture and its advantage of reduced memory requirement.

I. Introduction

Inspired by how brain works: artificial neural networks (NN) have attracted the growing interest of researchers, scientists, engineers and students in many different scientific and engineering areas. Many NN architectures were proposed over the years, among which we will look at the Cerebellar Model Articulation Controller (CMAC) [1, 2], first developed by Albus in an attempt to develop an efficient computational algorithm for use in manipulator control. Functionally, CMAC is a generalized Look-Up Table which able to perform multivariable approximation. Structurally, it is a three-layer feed-forward associative memory neural network. These features enable the CMAC very unique characteristics such as fast speed training and local generalization, which makes it particularly suitable for control applications, signal processing and pattern recognition.

The performance of CMAC is largely dependent on quantization of each input dimension, fine resolution is desired to ensure satisfactory performance, but results in large size of memory to be used and naturally, higher cost incurred. In many cases, CMAC are made impracticable due to its high memory requirement, such as for small-embedded systems, in which both space and/or cost of memory are of major concern.

Motivation to make CMAC used in embedded system such as commercialized electronics has come into sight of many researchers. Producing an affordable yet storage efficient FPGA based Novel CMAC hardware is the main objective of this project.

In order to exploit the parallel processing capability of the CMAC net, the CMAC can be implemented by means of FPGA chips, together with banks of RAM chips. In previous works, emphasis had been made on efficient migration of CMAC from software simulation to hardware realization [3, 4] without the fundamental changes to CMAC structure itself. It was concluded that more memory to be used to increase the input resolution, then better performance could be achieved. The CMAC input quantization and memory size dilemma still exist and yet to be solved. For small-embedded system, addition of RAM chips means both increase in cost and space, which may again yield CMAC-based application impracticable. Instead of reducing input resolution to avoid high memory requirement, we could be solved the problem in a different approach: non-linear quantization of input, which is used to make better memory utilization without lost to dynamic resolution.

Many clustering techniques can be used for non-linear quantization. A novel fuzzified quantization technique based on Discrete Incremental Clustering (DIC)[5] was chosen for its simplicity and effectiveness demonstrated in CVT control application [6]. However, DIC itself is a complicated algorithm with full real numbered arithmetic and requires large hardware for implementation. Hence, this project is specially addressed the problem and proposed the modified algorithm named Sim-DIC which is more suitable for hardware implementation. Finally, Sim-DIC clustering and CMAC are integrated to a Fuzzy Novel CMAC, which is highly hardware realizable and allows on-chip learning and flexible configuration. The proposed architecture is also implemented on low-end FPGA chip and benchmarked by classification problem.

Section II briefly presents the neural network model -CMAC, which maybe implemented in a hash mapping method or conventional Look-Up Table (LUT). Both methods are briefly introduced and compared. Section III explains how fuzzy quantization is achieved through discrete incremental clustering (DIC) and why it cannot directly prototyped to FPGA. Implementation is presented in section IV with special addressing to multiplier-free implementation for fast processing as well as testing Result for 2-spiral classification problem. The paper is concluded in Last Section.

II. The neural network model

The CMAC architecture can be considered as consisting of a single layer of memory locations, a memory addressing unit, weights adjusting unit and an output-summer. Simply speaking, the content of addressed memory locations (i.e. weights) are summed to provide network response to the input. Some applications might require input signals to be transformed to CMAC Input Space. CMAC is then defined by a series of mappings,

$$X \Rightarrow M \Rightarrow W \Rightarrow y$$

Where X is transformed input vector from external CMAC environment, M is the set of addressed memory locations, W is the set of contents (weights) of M, y is an one dimensional output. A function of

$$y = h(X)$$

is a well representation of overall mapping $X \Rightarrow y$. In the case CMAC nets consisting set of N CMACs operating on the same input to produce a vector mapping

$$X \Rightarrow Y$$

This, similarly, has all properties of the vector function

$$Y = H(X).$$

Hash Mapping

For Albus proposed CMAC, hash mapping scheme is used for indexing multiplayer CMAC, i.e $X \Rightarrow M$ mapping. To explain the concept of Hash Mapping, the case of a single input and single output (SISO) is considered. The variable of the transformed input space is denoted by *x*.

In the first layer, the input space is parted into five divisions, which division is the so-called *Cell*. Now, five weights from *W*11 to *W*15 are allocated to each *Cell*. Then, the second layer can be added by shifting the first layer to *S*21. The *Slk* means the amount of shift in the direction of the *k*th degree of freedom (DOF) on the *l*th layer. Therefore, *S*11 necessarily becomes zero because the first layer is the basis. Subsequent layer can be added in a similar manner. The result to input *x* would be summation of weights of all indexed cells, one from each layer.

The more layers are used, the better the output resolution can be achieved; but the required size of memory increases according to the number of layers. One should be careful when adds in more layers so that none of the shifted layers coincide with one another.

Multiple Layers of weight matrix also enables generalization property and hence speed up learning process. This is easy to see as one weight in a specified layer can be indexed by a number of different inputs close to each other, or neighborhoods.

Conventional Look-Up Table

Since the basic idea of CMAC is to store information in a Look-Up Table (LUT) manner, one can simply use conventional LUT scheme, which simply has one cell for each and every possible input index in every dimension. To compare the memory requirement of both approaches, we have the ratio of the two memory structures,

$$\delta = \frac{N_{(TLU)}}{N_{(hash)}} = (R/C)^{N-1} = K^{N-1}$$

with the assumption that the *N* input dimensions are of the same resolution, each can produce *R* different outputs, the number of layers is *K,* assume that the number of cells in each layer is the same:

$$C = C_1 = C_2 = \Lambda = C_k$$

One can easily know from this ratio that the hash mapping becomes more effective as the dimensionality of the input space increases. It is also worth to point out that memory requirement for hash mapping is further reduced when more layers are used (increasing *K*), but this also increases the complexity of the addressing algorithm, which makes it hardware expensive. In addition, such an algorithm is usually topologically non-adaptable to hardware implementation, when number of layers changes, the whole addressing scheme need to be changed.

Thus LUT scheme was chosen for its simplicity and flexibility in hardware realization. And the indices could be calculated as follows[10]:

Given a two-dimensional input (x_i, x_j), the winning neurons' locations can be calculated in

$$(a_i^k, a_j^k) = \left(\left\lceil \frac{(i_{\max} - K)}{x_{i\max}} x_i + k \right\rceil, \left\lceil \frac{(j_{\max} - K)}{x_{j\max}} x_j + k \right\rceil \right)$$

Where (a_i^k, a_j^k) are the calculated index for the i and j axis for layer *k,* $x_{i\max}$ and $x_{j\max}$ are the maximum value of the inputs in respective dimension, *k* = { 1, 2 … *K*} is the Layer number, $i_{\max}$ and $j_{\max}$ are the maximum addressable memory locations in respective axis, K is the number of Layers used, $\lceil x \rceil$ is the ceiling function, denotes least integer that is greater than or equal to x

It is important to point out that the expressions $(i_{\max} - K)$ and $(j_{\max} - K)$ ensure when the input (x_i, x_j) is at maximum, the addressing scheme will not allow any memory overflow. The overflow problem can also be solved from hardware prospective, by adding the memory locations in each axis to

$$R' = R + K - 1$$

Then the calculation for determining winner neurons is further simplified to

$$\left(a_i^k, a_j^k\right) = \left(\left\lceil \frac{x_i}{x_{i\max}} + k \right\rceil, \left\lceil \frac{x_j}{x_{j\max}} + k \right\rceil \right)$$

The number of layers and localization topology can simply be adapted by varying the value of *K* and how it is selected. Therefore In this project, the LUT addressing scheme with multiple winning neurons is used for its simplicity in hardware implementation.

III. Fuzzy quantization

The *uniform quantization* problem remains in the conventional CMAC presented in previous section. It can be described such that the CMAC input space is quantized into equal-size regions while inputs at problem space are not necessarily uniform. When problem inputs appear with uneven degree of variation, or clustering effect, then uniform quantization of such inputs could be associated with large quantization error and not efficient for storage.

Moody [7] uses multiple CMACs with different resolution to resolve the quantization error problem. His idea is to start learning with Low Resolution CMAC, and CMAC with higher resolution is then used in same input space if the result is unsatisfied. This expansion process stops only when error rate is reduced to an acceptable level, and output of this structure is the summation of all CMACs' outputs. This approach solved minimum acceptable resolution problem but fundamentally the uniform quantization characteristic remains unchanged, further more, memory requirement increases and becomes unpredictable at first instance when problem is given.

Kim and Lin [8] proposed an adaptive quantization technique for input space by using *mapping function*. In this approach, training of CMAC stills starts with uniform quantization, and subsequently input intervals with large variation are compressed and those with small variation are extended. Hence learning accuracy is improved and storage requirement is reduced. However, the mapping function used to convert inputs is globally adjusted and it requires prior knowledge of the problem inputs in order to determine which are the intervals to be compressed or extended.

A fuzzified MCMAC (Modified CMAC) proposed by Shu [6] employs Fuzzy Quantization to replace uniform quantization. This is achieved by the use of a novel clustering technique named Discrete Incremental Clustering (DIC) [5], which dynamically forms the fuzzy clusters by using only input information from same neighborhood (or intervals). The theory is stated as:

"The input space is divided into several regions, and quantization ratio in that region is proportional to the number of fuzzy sets that covers the region."

It can be illustrated using figure 1.

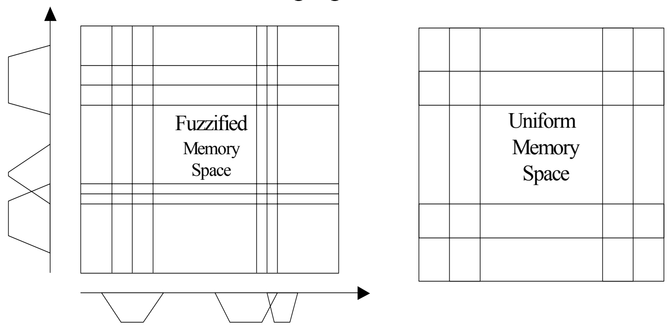

Figure 1. Memory Space Structures

Note the trapezoidal-shaped fuzzy sets for both input dimension are created as indexing rule base by DIC. The DIC is a novel clustering techniques that requires only raw numerical values of training inputs without any pre-processing. In most DIC implementations, trapezoidal-shaped fuzzy sets are used and each fuzzy set belonging to the same input dimension has little or no overlapping in kernels with its immediate neighbors. The DIC performs clustering on a local basis, that is, the number of fuzzy sets created is dimensional specific. This concept is similar to ART clustering techniques but in DIC will not "recreate" any existing fuzzy set for a particular dimension. Hence, DIC ensures that fuzzy set can uniquely identify fuzzy labels which formulate a consistent rule base for Fuzzy CMAC.

DIC technique overcomes several limitations encountered in many other partition-based clustering techniques. It does not require prior knowledge about number of clusters to be created for a given set of data. DIC performs clustering on a local basis and can significantly increase the memory efficiency yielding reduced the memory requirement. Moreover, DIC provides a platform for Fuzzy CMAC to formulate consistent fuzzy rule base.

However, DIC itself is NOT a hardware friendly algorithm, for it has large computational overhead and difficult-to-implement non-linear functions. With limited precision, limited computational resources and restricted arithmetic that can be performed on FPGA, it is really challenging to have DIC, or rather, DIC-like algorithm to be implemented from a hardware prospective.

There are many costly computations (other than addition, subtraction and bit-wise operations) involved in DIC algorithm, to name a few, non-linear sinusoidal function, division, multiplication. One of the major problems of FPGA implementing Neural Networks, affecting both performance and number of gates used, is the presence of multiplier. Thus the preliminary feature of proposed Sim-DIC is that it must be multiplier free.

- Some multiplication can be replaced by bit-wise operation, such as bit shift which equivalent to multiply (or divide) a Power of 2 Number.
- The nonlinear functions are implemented using Look-Up Tables (LUT) with proper discretization of entries.
- The Membership Function is also implemented in LUT with consideration for data precision.

Three major effects of these simplifications and modifications are founded in the testing result. They are:

- Reduced Accuracy and precision
- Reduced flexibility in fine tuning of parameters
- Increased Learning Speed on very small circuit

We shall address these effects again in next section.

IV. Implementation & Result

The Fuzzy CMAC with Sim-DIC was implemented on Student RC100 Development Board (Celoxica) featured with one Spratarn II 20K gates FPGA chip from Xilink[9]. The project was conducted in software/hardware co-design manner, as in, the proposed novel architecture was first implemented using C language, tested and tuned for performance; then the C code was migrated to Celoxica Handle-C language, which is a C-like High Level Language can subsequently be compiled and translated into HDL; then the Bitmap is generated from HDL file by Xilink Design Manager for running in specific FPGA.

Note that this project is to demo the novelty and feasibility of implementing a Fuzzy CMAC on FPGA, so the data transfer from PC to RC100 Board was not of main concern. Thus we prepared the input data in raw file and pre-load to onboard Flash RAM for use. And for the result, which is also in raw format, is loaded to PC and converted to Text file for easy viewing and analysis.

The Sim-DIC clustering Phase and CMAC Learning Phase of Fuzzy CMAC is performed by FPGA, however, due to large hardware created for both Sim-DIC and CMAC, two phases are *time-partitioned* in the design. So FPGA is first configured for creating clusters and store them back into RC100 onboard Flash RAM, and the re-configured to a CMAC enabled Fuzzy Quantization which is realized by indexing through clusters. This way we enhanced the functional density of FPGA, which is effective for further cost saving

Note that this project is to demo the novelty and feasibility of implementing a Fuzzy CMAC on FPGA, so the data processing was not of main concern. Thus we prepared the input data in raw file and pre-load to onboard Flash RAM for use. And the result, also in raw format, is loaded to PC and converted to Text file for easy viewing and analysis. Figure below illustrates the overall implementation strategy in block diagram (Figure 2).

Our implementation of Sim-DIC uses 73% of slices (logic elements) of the Spartan II (FG456) with a highest frequency at 21.44MHz. The Fuzzy CMAC (learning phase) with indexing through clusters is made up 41% of slices and runs at 31.74MHz. In contrast, conventional CMAC with uniform quantization can be implemented using only 36% slices but runs at close frequency of 33.05MHz. Table 1 below shows a summary of important implementation statistics running on a P4-2.4 512M PC

Table 1. Statistic of Current Implementation

	Sim-DIC	Fuzzy CMAC	CMAC (pure)
Size (# of Gates)	74,132	34,946	29,037
Device utilization (%slices)	73%	41%	36%
Max clock	21.439 MHz	31.736 MHz	33.050 MHz
Period	46.643 ns	31.510 ns	30.257ns
Build Time	47.1 sec	23.6 sec	20.5 sec
Netlist->.BIT time	1 min 28.4 sec	42.5 sec	40.2 sec

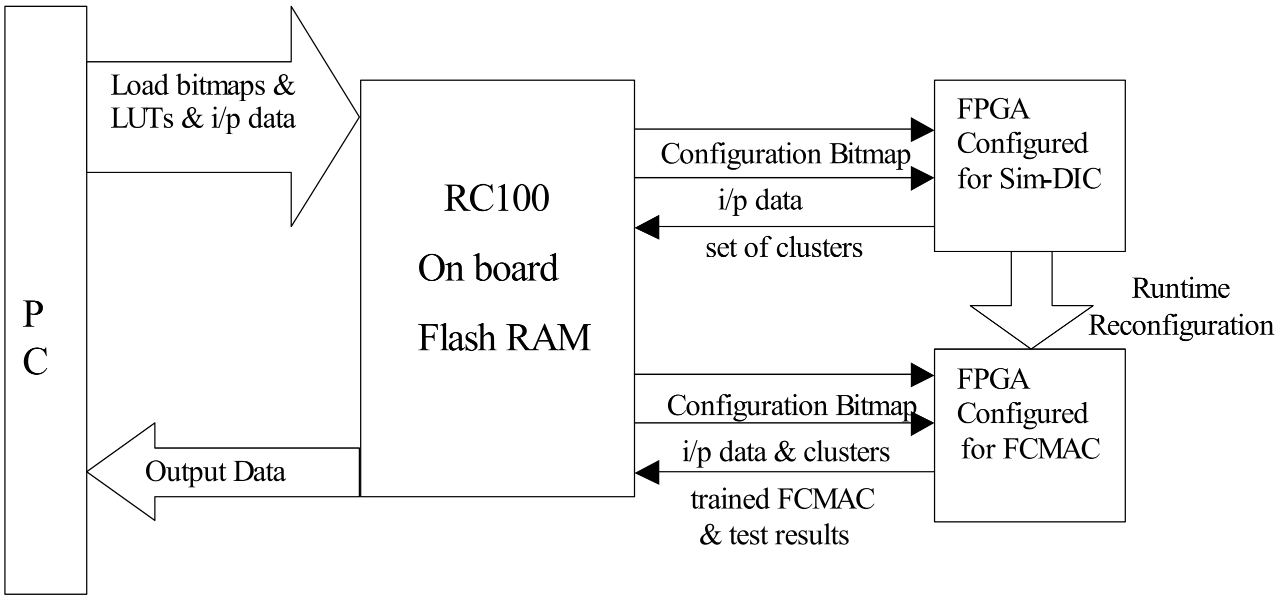

Figure 2. FCMAC Implementation Diagram

Table 2. Test Result on Recall Performance

	Test Set A (194)			Test Set B (770)		
Architecture	Correct	Errors	Classification Rate	Correct	Errors	Classification Rate
FCMAC on PC (double precision)	194	0	100%	770	0	100%
64X64, CMAC	194	0	100%	770	0	100%
40x40 , CMAC	188	6	96.91%	766	4	99.48 %
30x30 , CMAC	189	5	97.42%	766	4	99.48 %
20X20, CMAC	180	14	92.78%	675	95	87.66 %
12x12 , CMAC	154	40	79.38%	625	145	81.17%
11x13, FCMAC	178	16	91.75%	691	79	89.74%
17x20, FCMAC	191	3	98.45%	769	1	99.87%
28x27, FCMAC	194	0	100%	770	0	100%

The classification of two intertwined spirals problem, first developed by Alex Wieland, is a complex task for neural networks. We adopted this problem as a validation benchmark for feasibility of using FPGA implemented Fuzzy CMAC for practical applications with defined error rate. The standard training set contains 194 points (set A) with 97 points for each spiral and testing set consists of 770 points (set B) with 385 each. The recall capability of proposed novel FCMAC is investigated. We conducted Supervised Training for CMAC or Fuzzy CMAC with desired value to be either 0 (for class 0) or 1 (for class 1).

We first benchmark the problem using an FCMAC simulated on P4-2.0G Personal PC, with precision set to Double. In this case, both set can be classified correctly, but both running time and memory consumption are relatively large. Then we tested the FPGA implemented CMAC and found out that using 16bit integer representation, the problem can be solved using 64x64 CMAC. Since the purpose is to further reduce the memory requirement on FPGA, and such a reduction will become more significant as input dimensions increases. We then reduced the CMAC size, by having 40, 30, and 20 cells in each dimension, and as predicted, the classification rate drops. And if the problem requires a classification rate above 95%, the minimum CMAC size required is 30x30.

With aid of Sim-DIC, the proposed Novel Fuzzy CMAC is able to classify the data correctly when cells drop to maximum 30 per input dimension, and still sustain a classification rate above 98% when cells were further reduced to max 20 per input dimension. We can see that Fuzzy CMAC still functions when cells go as low as 11x13, given classification rate about 90%. In same case for 12x12 CMAC, only about 80% data can be classified due to poor resolution (Table 2).

More experiments are now in progress. To further demonstrate the viability of proposed Fuzzy CMAC and its advantage over conventional CMAC in clustering effect, hardware cost saving and reduced memory requirement. Test for higher dimensional problems will also be conducted.

V. Conclusion

In this paper, we proposed a hardware-friendly fuzzy clustering technique based on DIC, namely Sim-DIC. We also implemented a novel Fuzzy CMAC in FPGA using simple prototyping platforms. Handle-C is used for fast migration of C-simulation on PC to hardware. The architecture and implementation is tested using two-spiral problem and proved to be of better performance over conventional CMAC of same size. The FPGA implementation of Fuzzy CMAC proved that the integration of Fuzzy Techniques and Neural Network can be realized in simple hardware form (Student RC100 Development Board). And such hardware FNN can subsequently be used in real-time, embedded systems and/or for the sole purpose of fast processing through FPGA's internal true parallelism.

Refernces

[1] J.S.Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Transaction of ASME, Journal of Dynamic System, Measurement, and Control*, Vol. 97, pp. 220-227, 1975

[2] J.S.Albus, "Data Storage in the Cerebellar Model Articulation Controller(CMAC)". *Transaction of ASME, Journal of Dynamic System, Measurement, and Control*, Vol 97, pp.228-233, 1975

[3] G.Horváth & F.Deák, "Hardware Implementation of Neural Networks Using FPGA Elements", *Proc. of The International Conference on Signal Processing Application and Technology*. Vol. II. pp. 56-63, Santa Clara, Ca. 1993.

[4] G.Horváth & T.Szabó, "Higher order CMAC and its efficient hardware realization " , *Proc. of the International ICSC/IFAC Symposium on Neural Computation*, pp. 72-78, Vienna, 1998.

[5] W.L.Tung & C.Quek, "DIC: a novel discrete incremental clustering technique for the derivation fuzzy membership functions", *Proc. Of 7th Pacific Rim International Conference on Artificial Intelligience*, Tokyo, 2002.

[6] Z.G.Shu, "Fuzzy Associative Memory for the automatic control of a CVT control in an Automobile", *HYP report*, SCE, NTU, 2002.

[7] J.Moody, “Fast Learning in Multi-resolution hirecharies”, *Advances in Neural Information Processing System*, Vol. 1, pp.29-38, D.S Touretzky, Morgan Kaufmann Publishers, 1989.
[8] H. Kim & C.S Lin, “ use of adaptive resolution for better CMAC Learning”, *International Joint Conference on Neural Networks, IJCNN*, 1, pp.517-522, 1992
[9] S.M, R.G & S.B, "RC100 Hardware Manual", Ver. 1.2, Celoxica Ltd., 2001.
[10] A. Wahab and C. Quek, “Novel Noise Modeling Using AI Techniques ”, in *Intelligent Systems: Technology and Applications*, Cornelius T. Leonades, N.W: CRC press, 2003, III, ch.9, pp.297-327.