Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Neural Network Chip for Color Correction

Jen-Hwa Kao*, Jar-Shone Ker** and Yau-Hwang Kuo*

Institute of Information Engineering*
Department of Electrical Engineering**
National Cheng Kung University
1, University Road, Tainan
Taiwan, R.O.C.
kuoyh@cad.iie.ncku.edu.tw

Abstract

Color correction is a complex nonlinear function approximation problem. In this paper, we employ the inverse plant control model and CMAC neural network paradigm to solve this problem. Experimental results have revealed excellent effect and fast speed on color correction by using CMAC-based approach. To support real-time applications, a digital higher-order CMAC chip is developed. This chip adopts systolic structure to realize the weight address generator, based on a novel weight address calculation formula which can sharply reduce the required weight memory size. Besides, a fast B-spline receptive field evaluation method is proposed to compute the receptive field values for a recursive formula. To provide the function of dynamic weight adjustment, an on-chip learning module is also embedded into the CMAC chip. An FPGA-based add-on card has exhibited a performance faster than software approach about twenty-five times.

1. Introduction

With the rapid progress of electronic color scanning and printing technologies, considerable excitement has been caused by the generation of documents containing color material for individual users. The color content in documents can range from colored texts to full-color pictorial images. This widening use of color has given rise to consideration of how color can be faithfully represented in the output of various printers. In other words, a color digital image printing system attempts to reproduce on output accurately those colors presented in the original image.

Considering an original image C and the corresponding printed image C', the difference between them is called distortion, which is not what we desire. A color correction mechanism H is then added between scanner and printer to eliminate the mismatch between C and C'. This paper intends to develop a real-time hardware system for the purpose of color correction.

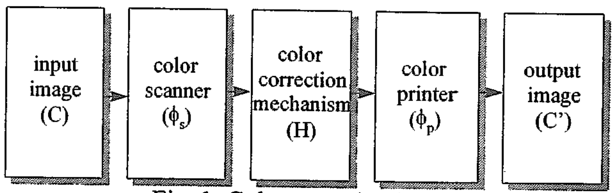


Fig. 1. Color reproduction system.

As shown in Fig. 1, let ϕ_s and ϕ_P be the invertible mappings which characterize the scanner and printer respectively. ϕ_s converts the original color image C to output data of scanner. ϕ_p acts as an inverse operator of ϕ_s which produces the final output color image from the output data of H. The overall process of color reproduction can be characterized by the following equation [1-2]:

$$C' = \phi_p \circ H \circ \phi_s \circ C \qquad (1)$$

where $\circ$ denotes functional composition operator.

In order to achieve the faithful reproduction of color images, the color correction mapping, H, should be chosen such that the appearance of any output color image C' is identical to the appearance of its corresponding original color image C, i.e., C' = C. Therefore, Eq. 1 becomes

$$C = \phi_p \circ H \circ \phi_s \circ C \qquad (2)$$

that is,

$$H = \phi_p^{-1} \circ \phi_s^{-1} = (\phi_s \circ \phi_p)^{-1} \qquad (3)$$

Obviously, H would be a complex nonlinear function. Since the higher-order CMAC model is quite suitable in nonlinear function approximation, we employ it to serve as the generalized inverse mapping H.

2. Hardware architecture of higher-order CMAC

A CMAC neural network [3-7] is a perceptron-like association memory, which is capable of multi-dimensional nonlinear function approximation, with overlapping receptive fields. Representation of a non-linear function, $y = f(x)$, by a CMAC model is accomplished by using two primary mappings, $S : X \rightarrow A$ and $P : A \rightarrow Y$, where X is a continuous M-dimensional input space, A is an N_A-dimensional association cell space, and Y is a one dimensional output space. The function S is usually fixed and maps each point in the input space X onto an association vector $\alpha \in A$ that has N non-zero elements. These N non-zero elements are called the active association cells which will affect the value of output. The $N_A \times 1$ association cell vector is defined as $\alpha = S(x)$.

The function $P(\alpha)$ computes a scalar output, y, by projecting the association vector determined by $S(x)$ on a $N_A \times 1$ vector, w, whose components are attached to their corresponding association cells, that is,

$$y = p(a = a^T w = \sum_{i=1}^{N_A} w_i \cdot S(x)$$

The mapping $S(x)$ can be characterized by the following three sub-mappings:

$R: X \rightarrow T$

$Q: X \rightarrow L$

$E: L, T \rightarrow A$

where R is a receptive field function, Q is a quantization function, and E is an embedding function. T is a matrix of receptive field activation values, and L is an array containing column vectors for identifying the locations of maximally activate receptive fields along each input dimension.

For a standard CMAC neural network, the receptive field activation value of each component t, $t \in T$, is equal to either 0 or 1. Thus, R maps the input x to a binary-stated activation matrix T. Both the quantization and embedding functions are used to assign T to its expected association cell vector. Generally, the receptive field functions are not limited to be the rectangular functions whose value is either 0 or 1. The receptive field functions in terms of high-order polynomials, i.e., cubic B-spline, may be considered in the CMAC-based function approximation. By formulating CMAC neural networks with B-spline receptive field functions, Lane had developed the higher-order CMAC neural network model to learn both functions and function derivatives [8].

Given N partitions of the real interval $x \in [a,b]$, a nth order spline function $\xi(x)$ of the form

$$f(x) \cong \xi(x) = \sum_{j=1}^{N+n-1} w_j \cdot B_{n,j}(x) \tag{4}$$

can be constructed to approximate $f(x)$ by using linear combinations of B-splines $B_{n,j}(x)$'s weighted by coefficients w_j's. The sequence of normalized B-splines, $B_{n,1}(x)$, $B_{n,2}(x)$, ..., $B_{n,N+n-1}(x)$, constructed on a knot set { λ_0, λ_1,..., λ_N} forms a basis set for all polynomial spline of order n on N partions of the interval $x \in [a,b]$. The $B_{n,j}(x)$ can be obtained from the recurrence relation.

$$B_{n,j}(x) = \left[\frac{x - \lambda_{j-n}}{\lambda_{j-1} - \lambda_{j-n}}\right] B_{n-1,j-1}(x) + \left[\frac{\lambda_j - x}{\lambda_j - \lambda_{j-n+1}}\right] B_{n-1,j}(x) \tag{5}$$

where $B_{1,j}(x) = \begin{cases} 1 & \text{for } x \in [\lambda_{j-1}, \lambda_j) \\ 0 & \text{otherwise} \end{cases}$

The B-spline index j is associated with the region of local support $\lambda_{j-1} \leqq x < \lambda_j$. Once the partition number of the active interval $[\lambda_{j-1}, \lambda_j)$ has been determined for B-splines of order n = 1, the recurrence relation of Eq. 5 can be used to generate all nonzero B-spline of higher order. The multi-dimensional receptive field functions can be obtained by multiplying the one-dimensional receptive functions contained in each tensor product, that is, $R(x) = R^1(x_1) \otimes R^2(x_2) \otimes ... \otimes R^M(x_M)$, where $\otimes$ denotes the tensor product, x_i is the input component of x, and $R^i(x_i)$ denotes the B-spline receptive field function of scalar variable x_i.

A generic hardware architecture of higher-order CMAC neural networks is shown in Fig. 2. Basically, it is composed of four processing units: multi-dimensional receptive fields evaluation module, weight cell addressing module, output generation module and learning module.

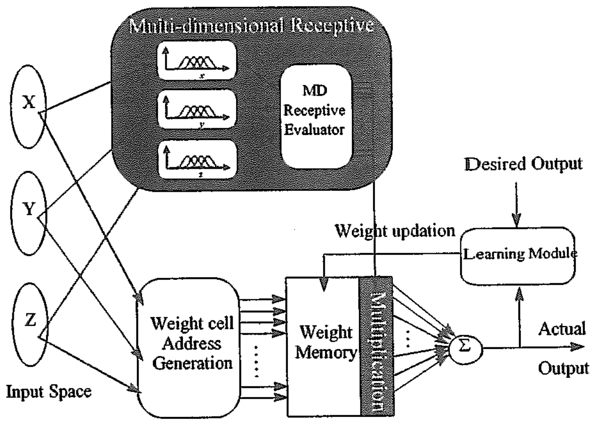


Fig. 2. Architecture of higher-order CMAC neural networks.

2.1. Fast B-spline receptive field evaluation methodology

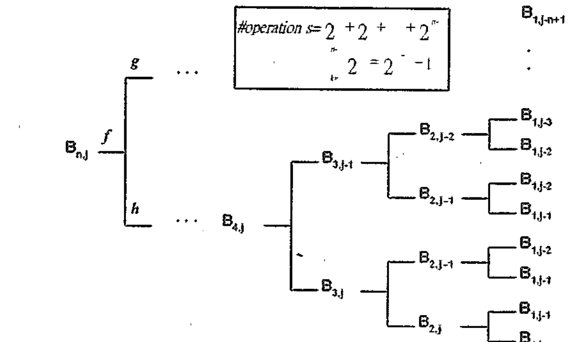
The feature of incorporating B-spline receptive field function with CMAC model contributes to well approximation capability on nonlinear continuous control functions. Obviously, it is a recurrence formulation for the evaluation of $B_{n,j}(x)$. The value of $B_{n,j}(x)$ can be computed when the two values, $B_{n-1,j-1}(x)$ and $B_{n-1,j}(x)$, are available. Therefore, the computation flow is equivalent to traverse a binary tree for the evaluation of the value, $B_{n,j}(x)$, of root node. Since the input space is quantized in uniform resolution, i.e., the width, D, between two adjacent knots are all the same, instead of maintaining the coordinates of knots for the evaluation of $B_{n,j}(x)$ we just need to maintain the value of resolution D on each input dimension. The recurrence formulation can be simplified into the following form

$$\begin{aligned} B_{n,j}(x) &= \frac{(x-(j-n)\cdot D)\cdot B_{n-1,j-1}(x)+(j\cdot D-x)\cdot B_{n-1,j}(x)}{(n-1)\cdot D} \\ &= f(g(x,B_{n-1,j-1}(x)),h(x,B_{n-1,j}(x))) \end{aligned}$$

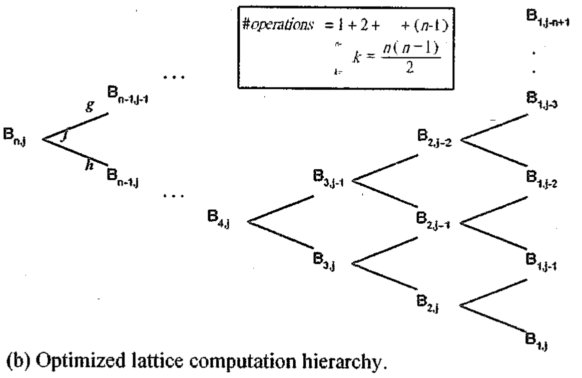
$$\begin{aligned} \text{where } f(x,y) &= \frac{x+y}{(n-1)\cdot D}, \\ g(x,y) &= (x-(j-n)\cdot D)\cdot y, \\ h(x,y) &= (j\cdot D-x)\cdot y, \text{ and} \\ B_{n,j}(x) &= \begin{cases} 1 & \text{for } x\in[\lambda_{j-1},\lambda_j) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

A spline operator, S, is defined to evaluate the corresponding B-spline value of each node in the computation hierarchy. The S operator evaluates the value of $f(g(x,y),h(x,z))$, which is defined in above. The recurrence computation hierarchy, as shown in Fig. 3(a), is un-suitable for hardware implementation because a stack data structure has to be maintained. For such a binary tree computation hierarchy, there are 2^{n-1}-1 S operations. We have derived an equivalent lattice computation hierarchy, as shown in Fig. 3(b), to efficiently evaluate the value of a B-spline function $B_{n,j}(x)$. Only n(n-1)/2 S operations are needed. Obviously, the lattice computation hierarchy not only reduces the number of S operations, but also can be implemented with a rather simple computation unit, which is shown in Fig. 4.

To evaluate the value of $B_{n,j}(x)$ with the S operator and a set of data storages, we just need to arrange the data storages into a set of bi-directional shift registers and properly shift them right or left to prepare the desired operands for the S operator. As shown in Fig. 3(b), each evaluation of $B_{n,j}(x)$ takes only n(n-1)/2 computation cycles.



(a) Original recurrence computation hierarchy.



(b) Optimized lattice computation hierarchy.

Fig. 3. Simplification of B-spline evaluation.

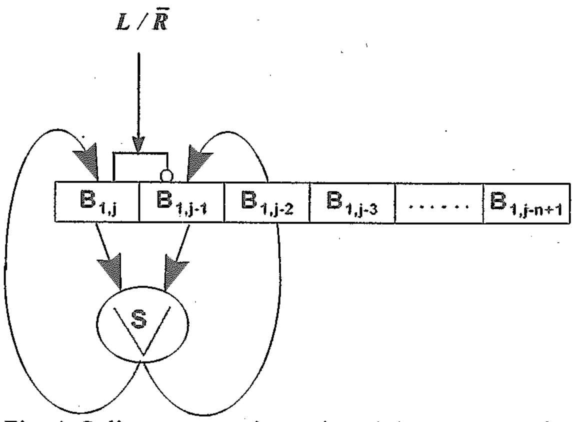


Fig. 4. Spline computation unit and data storages for evaluating $B_{n,j}(x)$.

Because of the heavily demands on the evaluation of $B_{n,j}(x)$, the higher-order CMAC chip is designed with two operation phase (warm-up phase and stand-by phase) to speed up the overall performance. A B-spline receptive field table is dynamically built during the warm-up phase. Due to the superiority feature of finite support for a B-spline function, we only have to record the nonzero values between two adjacent knots. Hence, the table has D * G entries, where D is the width between two adjacent knots and G is the number of non-zero B-splines activated in the interval $[\lambda_{j-1},\lambda_j)$. To get the non-zero B-spline values for a given input vector, we just need to retrieve G pre-evaluated values from the B-spline table instead of spending computation time in unnecessary evaluations. Such a methodology contributes to the promotion in overall performance. After the table is constructed, the operation mode will be automatically switched into the stand-by phase for the processing of input streams.

2.2. Weight cell address generation

The extended formula [9] for weight cell address is shown in the following form:

$$PN_j = \sum_{m=1}^{M-1}\left(\prod_{i=m+1}^{M}\left(1+\left\lceil\frac{Q_i - J_{i,j}}{G}\right\rceil\right)\times\left\lceil\frac{q_i - J_{i,j}}{G}\right\rceil\right)+\left\lceil\frac{q_M - J_{M,j}}{G}\right\rceil$$

for $j = 1 \ldots N$. (6)

where J is a $M \times N$ addressing matrix which is used to specify the desired addressing scheme, M is the number of input dimensions, and N is equal to the number of specific non-zero terms in the hyper-cube which is determined by the location of input vector and the value of G. Each entry $J_{i,j}$ refers to the relative coordinate of the corresponding hyper-cube on dimension i of the j-th non-zero term. Those specific non-zero terms constitute the addressing scheme for determining how many weight cells contribute to the response of an input vector and evaluating where they are. By changing the contents in the addressing matrix, any desired addressing scheme can be realized without modifying the hardware. Therefore, it results in a highly flexible weight addressing mechanism.

2.3. Systolic structure for weight cell address generation

To efficiently implement the address generation unit, the operations performed in Eq. 6 are rearranged into the following equivalent recurrence form:

$R_i(0) = 0$

$$R_j(i) = \left(1+\left\lceil\frac{Q_i - J_{i,j}}{G}\right\rceil\right) \times R_j(i-1) + \left\lceil\frac{q_i - J_{i,j}}{G}\right\rceil$$

for $i = 1 \ldots M$, and $PN_j = R_j(M)$ (7)

A systolic-array architecture is then derived to implement the extended direct addressing mechanism. The corresponding systolic schedule is illustrated in Fig. 5. In Fig. 5, the PEs in the same row, i, evaluate the corresponding component $R_j(i)$ in column j, for $j = 1$ to N, according to the values of Q_i, q_i, $J_{i,j}$, and the component $R_j(i-1)$ which is generated by its precedent PE. The PEs in the same row can be operated concurrently or in any sequential order. However, for the PEs in the same column j there exists data dependency between adjacent PEs, so that the PEs has to operated sequentially in the ascendant order of i.

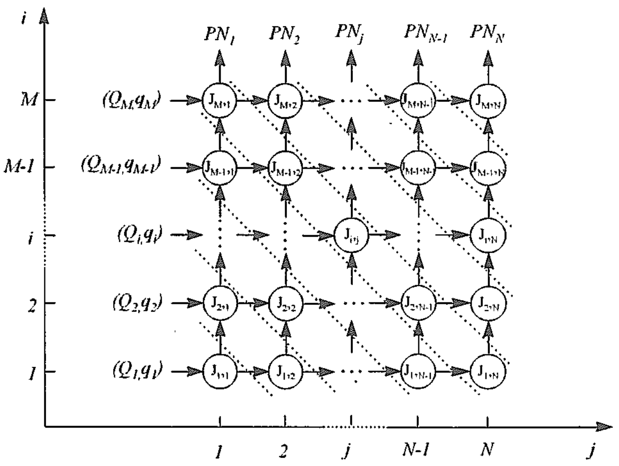


Fig. 5. Systolic schedule for direct addressing mechanism.

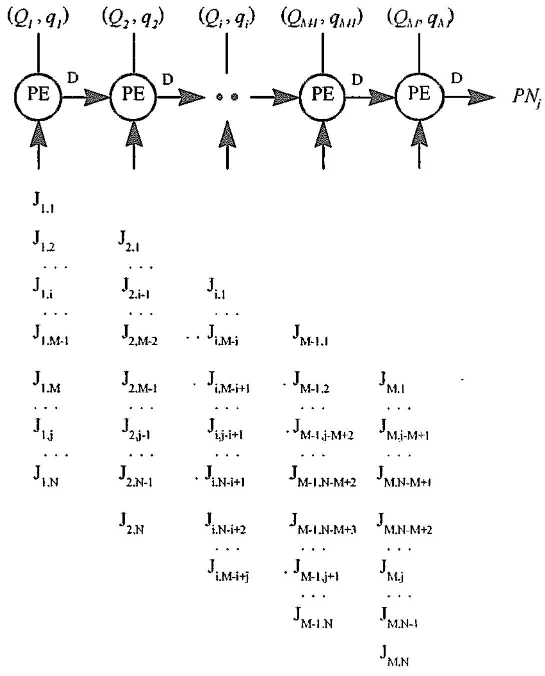
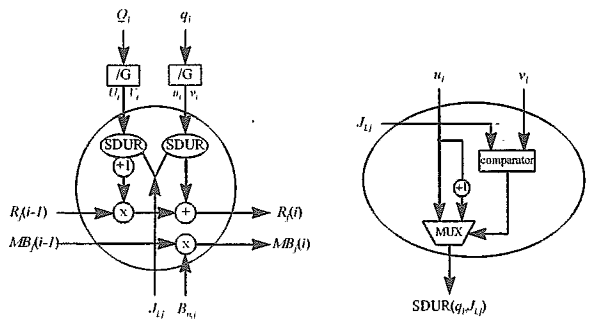


Fig. 6. Linear systolic array mapping.

The PEs in the same partition (in Fig. 5, each partition is identified by the dotted lines) can operate simultaneously. Therefore, to reduce the overall hardware cost, a linear systolic array architecture is employed as shown in Fig. 6. The detailed configuration of each processing element (PE), illustrated in Fig. 7, will be described in the following subsection.

As shown in Fig. 6, the proposed linear systolic array has M PEs. Each PE evaluates the value of $(1+SDUR(Q_i,J_{i,j}))\times R(i\text{-}1)+SDUR(q_i,J_{i,j})$ and then delivers the computation result forward to the adjacent PE. After M latency delay cycles elapsed, the systolic array will produce one weight address per cycle at the last PE. Therefore, the throughput rate is equal to one weight address per cycle.

For a typical case, the value of N is usually much greater than the value of M. N PEs will take $2\times N$ SDUR computation modules and N multipliers. For example, in the RGB color calibration application, there are three input variables (i.e., $M=3$). Let the generalization value G is set to 4, then we will get 64 non-zero terms for the fully connected addressing scheme, i.e., $N=64$. Only three PEs are needed for the realization of linear systolic array architecture regardless of what kind of addressing scheme is to be employed.



(a) configuration of a PE. (b) configuration of an *SDUR*.

Fig. 7. Detailed configuration of PE.

The SDUR operation can be carried out by one integer division $\frac{q_i}{G}$ and one increment or decrement operators. Let $SDUR(q_i,J_{i,j})=\left\lceil\frac{q_i-J_{i,j}}{G}\right\rceil$ and $0\leq J_{i,j}<G$, we can claim that the result of $SDUR(q_i,J_{i,j})$ operation, for $J_{i,j}=0, 1, \ldots, G\text{-}1$, is equal to either the quotient $\left[\frac{q_i}{G}\right]$ if the remainder R of $\frac{q_i}{G}$ is not greater than $J_{i,j}$ or $\left[\frac{q_i}{G}\right]+1$ if the remainder R is greater than $J_{i,j}$.

The *SDUR* operation can be carried out by one integer division $\frac{q_i}{G}$ and one increment operators. After analyzing the data sequence fed to a PE in Fig. 6, we find that the value of Q_i and q_i are constants during the evaluation session of PN_j's for a certain input vector X, and only the value of $J_{i,j}$, for $j=1$ to N, is varied. Therefore, by referring to the discussion of the above corollary, the operation of *SDUR* can be further accelerated and simplified by extracting the division operation from *SDUR* operator to shorten the computation time.

For each processing element PE, the corresponding division operation just needs to be activated once, and then the values u_i and v_i, which are corresponding to the quotient and remainder part respectively, are to be latched in the PE. Therefore, the operation performed in a *SDUR* operator not only becomes rather simple, but also takes less computation time. As shown in Fig. 7(b), the value of $SDUR(q_i,J_{i,j})$ can be easily determined, which equals to u_i, or u_i+1, by comparing the values of v_i and $J_{i,j}$: the value of *SDUR* is equal to either u_i when $v_i\leq J_{i,j}$, or u_i+1 when $v_i>J_{i,j}$.

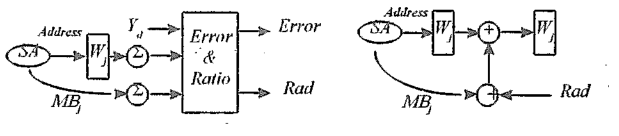
With the linear systolic array architecture, a cost-effective hardware implementation [10] of higher-order CMAC model is developed. Only M PEs are needed to compute all the weight cell addresses. The operations of a PE are rather simple: one multiplication, two SDUR operations, and one addition. The latency to generate the first address is equal to M operation cycles. After the first address is generated, there are N successive operation cycles for generating all the required weight cell addresses. In other words, the latency time for the first output vector is equal to $(M+N)$ operation cycles. The overall throughput of the linear systolic array is equal to $1/(N^*T_{cyc})$ for generating successive output vectors. The required resources on data path and performance for these two mappings are summarized in [11].

2.4. On-chip learning module

The difference of learning scheme between CMAC and back-propagation neural network [12-14] is the use of local approximation technique in CMAC rather than global approximation in back-propagation neural network. Since only a small subset of weights are updated during a training cycle, local approximation technique has the advantages of faster learning speed .

The learning of CMAC model denotes the procedure of modifying the value of weights; the association vector is not adjusted in the learning process. In CMAC model, all weights are initialized to zero. The M.S.E. is adopted as the energy function during learning process, and the maximum gradient method [12-15] is used to calculate the adjustment amount of each weight.

Fig. 8 shows the operation flow of CMAC learning; the learning module accepts the addresses, weights, and corresponding receptive field function values as inputs in each cycle. The learning rule is list as follows:



(a) Operation flow of phase 1. (b) Operation flow of phase 2.

Fig. 8. Operation flow of learning for CMAC model.

$$w_j = w_j + \beta \cdot \frac{(Y_d - Y) \cdot MB_j}{\sum_{j=1}^{N} MB_j} \tag{8}$$

where w_j means the jth weight, MB_j means the jth receptive field function value (i.e. the coefficient of corresponding weight), Y_d is the desired output, $Y = \sum_{j=1}^{N} w_j \cdot MB_j$, and N denotes the number of non-zero terms a training pattern excites (N = 64 in our design). Let $R_{ad} = \beta \cdot (Y_d - Y) \big/ \sum_{j=1}^{64} MB_j$; since the value of R_{ad} is always a constant during a certain training pattern is trained, we just need to compute the value of R_{ad} once in the learning period of a certain pattern. For simplicity, the learning module is divided into two phases.

In the first phase, we compute the value of R_{ad}. Because a division operation will be performed in the first phase, three extra clock cycles are needed for the division operation after 64 clock cycles since the first set of input data is received. Thus, it spends 67 clock cycles in first phase. In the second phase, each weight w_j will be added with the value of R_{ad} * MB_j. During this phase, 64 clock cycles are needed.

Totally, the learning module spends 131 clock cycles to train a certain training pattern. In our simulation, 455 training patterns are given, and each training pattern is trained in 200 iterations, so about 2.38 seconds (200 * 455 * 131 * 100 * 2 ns) is needed for the whole training process if no extra delay between two adjacent training patterns is inserted. However, the training process by software simulation will take 60 seconds in SPARC 10.

3. Synthesis result of higher-order CMAC

To implement the hardware architecture of higher-order CMAC neural network model, we adopt the high-level synthesis methodology to design the circuits. All the modules are described in VHDL hardware description language and then synthesized by the Viewlogic EDA tool. The processing speed of synthesized chip is about 46 ns. The critical delay comes from the propagation latency of a combinational divider. Table 1 summaries the hardware costs and its performance.

Table 1. Synthesis report of higher-order CMAC chip.

	Cell	Gate Count	Min. Delay	Max. Delay
ACC_GEN	1470	4325	6.3	46
ADR_GEN	1842	4500	4.5	36
OPT_GEN	1050	2575	3.1	21
LRN_GEN	440	6150	0.0	20
PROTOTYPE	4802	16550	0.0	46

4. Experimental results

In color correction experiment, we set B-spline function order, n, to 4 since it generates smoother result. Besides, the learning rate, β, is set to 1.0, the quantization level, Q, is set to 8, and 455 training patterns are given to train three CMACs for correcting R, G, and B. In our simulation, the scanner used is the Umax uc300 color scanner, and the printer used is the FARGO PrimeraPro thermal color printer.

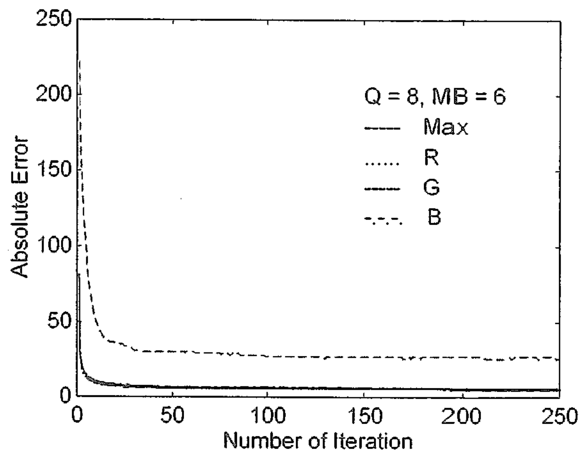
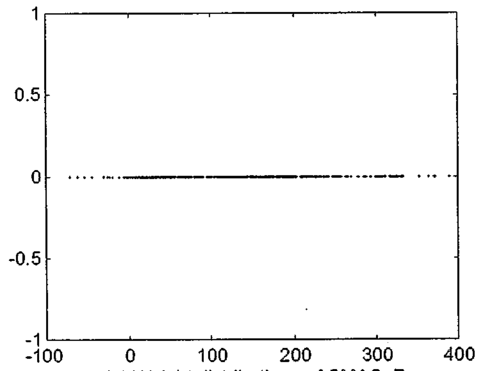
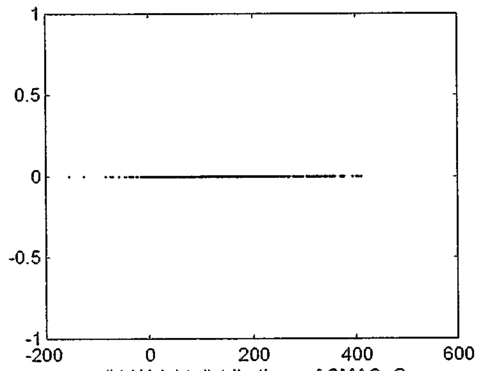


Fig. 9. Absolute error of R, G, and B with respect to different MB.

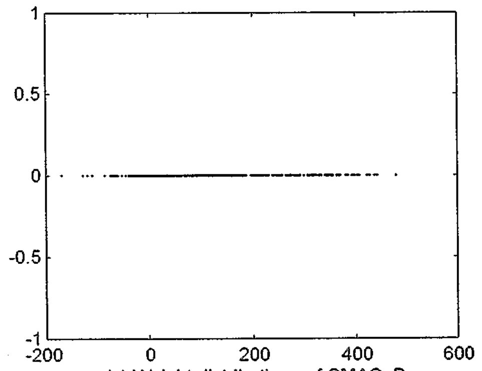
There are two factors which will affect the results. One is how many bits we represent the fraction in physical hardware, another is how many bits are afford to represent the value of weight cells. Fig. 9 illustrates



(a) Weight distributions of CMAC_R



(b) Weight distributions of CMAC_G

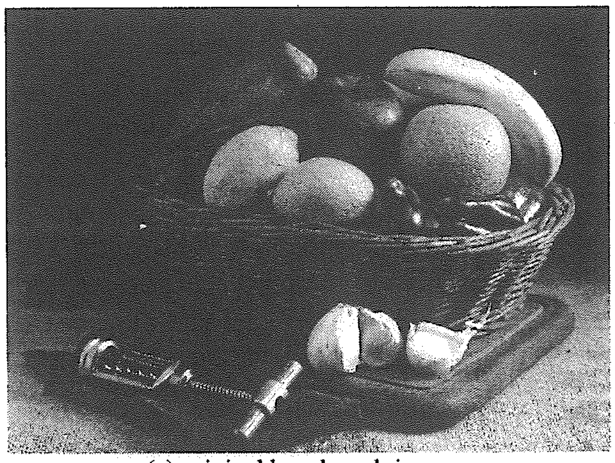


(c) Weight distributions of CMAC_B

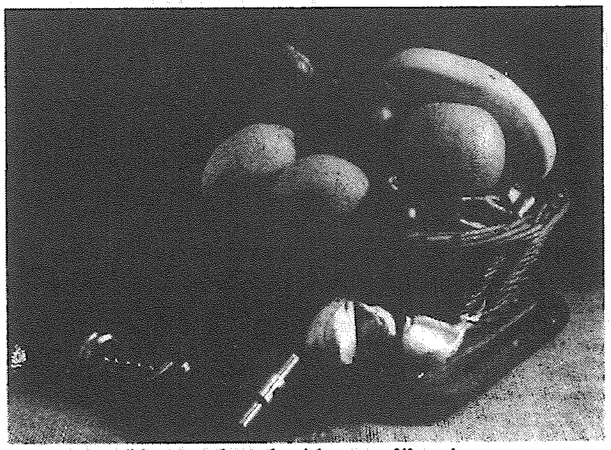
Fig. 10. Weight distributions of color correction.

the error curves for adopting 6-bit receptive field values (MB=6). We observed that the differences among the corrected images for different MB values (MB=6~11) is tiny. Therefore, we adopt 6-bit fraction for less circuit cost.

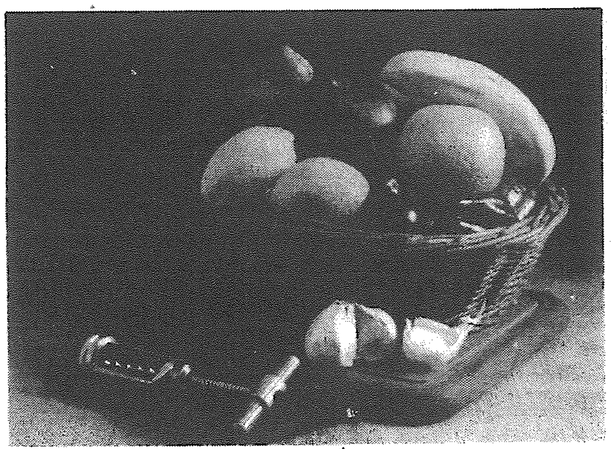
As to the effect of the second factor, it shows that for getting good results, 9-bit weight memory (with one sign bit) is required at least. The weight distributions in color correction, illustrated in Fig. 10, show that many values of weight cells exceed 255. In our design, weights are represented in a 10-bit integer with one sign bit.



(a) original benchmark image.



(b) reproduced without calibration.



(c) calibrated with a 4th-order CMAC.

Fig. 11. Calibration effect of the developed color reproduction prototype chip.

The source image is shown in Fig. 11(a). Fig. 11(b) is the corresponding image printed without color correction. Obviously, color distortion phenomena is very serious. The reproduced picture, as shown in Fig. 11(c), presents rich and vivid colors for better visual effects. On the other hand, the picture in Fig. 11(b), which is reproduced without applying any color calibration scheme, exhibits poor and drab colors.

5. Conclusion

According to experimental results, higher-order CMAC model exactly exhibits excellent performance on color correction. In fact, besides color correction, it can be employed in other fields such as adaptive filter, estimation, and control, where it is desired to learn not only a function, but also the function derivatives.

We also developed a cost-effective hardware architecture to realize higher-order CMAC model. A linear systolic array architecture is derived so that we can efficiently generate all required weight cell addresses in a related hyper-cube window. Only M (number of input dimensions) PEs are needed for the computations of N weight cell addresses. The throughput of such a systolic architecture is equal to $1/(N^*T_{acc})$, i.e., it takes N computation cycles to generate two adjacent output vector.

In addition to the weight cell address generation mechanism, a fast computation unit for the evaluation of B-spline receptive fields is also constructed. The lattice computation hierarchy reduces the amount of operations to n(n-1)/2, where n is the order of B-spline function. No stack storage mechanism is required to complete the evaluation of B-spline recurrence formulation. The learning module is added such that higher-order CMAC model can work faster than software does. In our application, it spends 2.38 seconds to perform a complete training by hardware approach. However, it spends 60 seconds to perform the same process by software approach in SPARC 10.

References

[1] C. W. Chang and P. R. Chang, "Neural plant inverse control approach to color error reduction for scanner and printer," 1993 Int'l Conf. on Neural Networks, San Francisco, CA, March 28 - April 1, 1993.

[2] P.R. Chang, "Color correction for scanner and printer based on B-Spline CMAC neural network," Technical Report, National Chiao Tung University, 1993.

[3] J.S. Albus, "A theory of cerebellar function," *Mathematical Biosciences*, Vol. 10, pp. 25-61, 1971.

[4] J.S. Albus, "Mechanisms of planning and problem solving in the brain," *Mathematical Biosciences*. Vol. 45, pp. 247-293, 1979.

[5] J.S. Albus, "A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 97, pp. 220-227, 1975.

[6] J.S. Albus, "Data storage in the CMAC," *Journal of Dynamic Systems, Measurement, and Control*, Vol.97, pp. 228-233, 1975.

[7] J.S. Albus, *Theoretical and Experimental Aspects of Cerebellar Model*, Ph. D. Dissertation, Dep. Biomed. Eng., Univ. Maryland, Dec. 1972.

[8] S.H. Lane, D.A. Handelman, and J.J. Gelfand, "Theory and development of higher-order CMAC neural networks," *IEEE control systems*, pp. 22-30, 1992.

[9] J.S. Ker, et. al., "Enhancement of the weight cell utilization for CMAC neural networks: Architecture Design and Hardware Implementation," Proc. Of MICRO-NEURO'94, Torino, Italy.

[10] J.S. Ker, "Design and implementation of higher-order CMAC neural network chip for color correction," Technical Report, National Cheng Kung University, 1995.

[11] Jar-Shone Ker, Yau-Hwang Kuo, and Bin-Da Liu, "Hardware Realization of Higher-Order CMAC Model for Color Calibration," Proc. of IEEE 1995 Intl. Conf. on Neural Networks (ICNN'95), pp.1656-1661, Nov. 1995.

[12] J.A. Freeman, D.M. Skapura, *Neural Networks Algorithms , Application, and Programming Techniques*. 1991.

[13] J.M. Zurada, *Artifical Neural System*, West Publishing Company, 1992.

[14] Simmon Haykin, *Neural Networks*, MACMILLAN, New York, 1994.

[15] T. Kohonen, *Self-Organization and Associative Memory*. New York: Springer-Verlag, 1988.