



Evolving binary-weights neural network using hybrid optimization algorithm for color space conversion

C.Y. Chen*

Department of Information and Telecommunications Engineering, Ming Chuan University, Taoyuan, Taiwan, ROC.

Received 22 May 2014; accepted 11 May 2015

KEYWORDS

Artificial neural networks;
Neuroevolution;
Evolutionary algorithms;
PSO;
Color space converter.

Abstract. Artificial Neural Networks (ANNs) are applied to many complex real-world problems, ranging from image recognition to autonomous robot control. However, to design a neural network that can implement special task, it is necessary to select an appropriate biological neuron model, meanwhile, good learning algorithm should be adopted to achieve the expected goal. Neuroevolution is a form of machine learning that uses Evolutionary Algorithms (EAs) to train ANNs. EAs, for the learning algorithm used by neural networks, can provide alternative and complementary solution, which can avoid the frequently happened issues of “getting stuck in local minimum” during the iteration process made by gradient-based learning algorithms. In this paper, a method using Hybrid PSO-based Learning Algorithm (HPLA) to evolve the connection weights and network parameters of Binary-Weights Neural Network (BWNN) will be introduced. The extracted knowledge from trained BWNN can then be used to construct high-speed shift-and-add based Color Space Converter (CSC) hardware architecture. The experimental results in this research also show that the performance of implemented hardware architecture is good at RGB to YCbCr color space converting, and it also has the advantages of high-speed and low-complexity.

© 2015 Sharif University of Technology. All rights reserved.

1. Introduction

Color spaces is a method by which different colors can be specified, created and visualized. There are many existing color spaces and most of them represent each color as a point in a three dimensional coordinate system. Different color spaces have historically evolved for different applications. RGB, YUV and YCbCr are common color spaces in use. YUV and YCbCr are adopted in video codec and transmission (for example, JPEG, MPEG, etc.), while RGB is adopted for display, so the conversion between them is unavoidable [1].

In many Digital Signal Processing (DSP) algorithms, multiplier coefficients are floating-point num-

bers, rational numbers, or real numbers. However, when implementing DSP algorithms, fixed-point coefficient operations are often preferred over floating-point due to lower complexity and power consumption [2]. When we are to realize Color Space Converter (CSC) from RGB to YCbCr, we will meet two dilemmas in the hardware circuit design; first, the conversion formula contains many floating-point operations; next, in the conversion formula, many multiplication operations will be met. If multiplier is used in the hardware architecture to realize the multiplication operation function, then massive operation time and hardware area will have to be consumed. In order to solve bad efficiency problem of traditional floating-point based CSC architecture, when it is used for color conversion, there are lots of different solutions proposed by scholars in the academy [1,3,4]. The LUT method is one of the high efficient methods especially for embedded

*. Tel.: +886-3-350-7001#3737;
Fax: +886-3-359-3879
E-mail address: chingyi@mail.mcu.edu.tw

systems [3]. In Ref. [4], Li et al. have proposed a method using Genetic Algorithm (GA) to evolve automatically shift-and-add based CSC. Such method is simple and effective, and it can also get high-speed CSC architecture. However, in its design process, only image quality is considered instead of considering the hardware resource consumed, hence, the number of adder used will be too much.

ANNs have been fruitfully used in a variety of fields, including economics, finance, meteorology, and engineering. Constructing neural networks involves difficult optimization problems, such as finding a network architecture appropriate for the application at hand, and finding an optimal set of weight values for the network to solve the problems [5]. The training process of ANNs consists of two tasks, the first one is the selection of an appropriate architecture for the problem, and the second is the adjustment of the connection weights of the network. However, ANNs have their own limitations. They are easy to fall into local minimum and sometimes hard to adjust the architecture. When there are many local minima, a neural network using a hill-climbing algorithm for optimization may stall on a different minimum on each run of the network.

Neuroevolution is a method for modifying neural network weights, topologies in order to learn a specific task. Various evolutionary computation methods, such as GA, Genetic Programming (GP) and Evolution Strategies (ES) etc. could be applied to evolve a neural network on its topological architecture, connection weights and activation functions, or even groups of learning rules [6]. With the development of computer technology, these methods have already achieved huge progress and exhibited a merging tendency [7,8]. GA has been used to solve each of these optimization problems. In weight optimization, the set of weights is represented as a chromosome, and a genetic search

is applied to the encoded representation to find a set of weights that best fits the training data [5]. Koza provides an alternative method to representing ANNs [9], under the framework of GP, which enables modification of not only the weights, but also the architecture for a neural network. In [10], the authors use ES to learn the weights of the neural network instead of learning the method of the network.

Among lots of computing techniques, Particle Swarm Optimization (PSO) has characteristics, such as less parameters setting, robustness and high convergence speed. It is inspired by insect swarms and has proven to be a competitor to GA when it comes to optimization problems. Successful applications of PSO to some optimization problems, such as function minimization and ANNs design [11-13]. However, the main problem with PSO is that it prematurely converges [14-15] to stable point, which is not necessarily minimum.

In this study, a method using neuroevolution strategy to realize multiplierless CSC architecture will be introduced. The complementary characteristics between PSO and GA are used to construct high efficiency hybrid evolutionary learning algorithm, which is then used to train BWNN (shown in Figure 1). After BWNN training is finished, the extracted knowledge from neural network can then be used to construct high-speed shift-and-add based CSC circuit hardware architecture. In the proposed hybrid learning algorithm, the elite strategy has been adopted to retain some particles with superior fitness in the population but sort out particles with bad fitness; moreover, the ones with better fitness are selected to reorganize and disturb their encoding strings through crossover and mutation procedures so as to replace the particles sorted out in the population. Through this method, diversification of particles in the population can be enhanced, and the premature convergence of PSO can be avoided too.

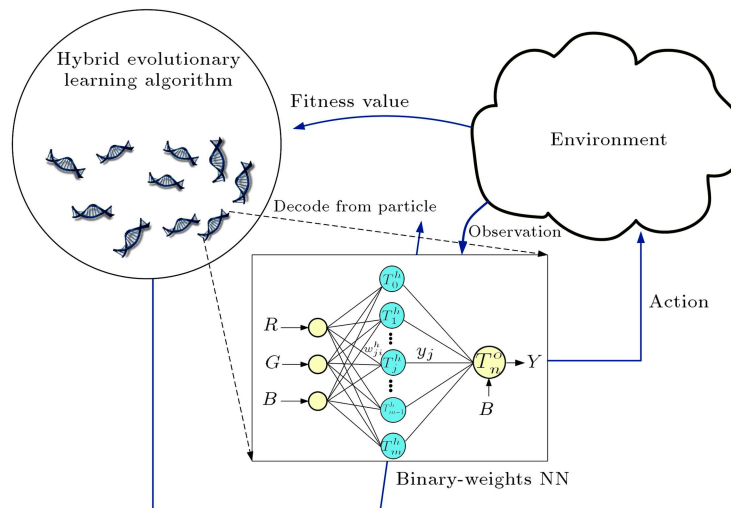


Figure 1. Evolving connection weights in a binary-weights neural network.

The composition of the rest of the paper is as follows. A review for the ANNs and an introduction for BWNN are given in Section 2. Section 3 is concerned with the color conversion formulas and the description of the proposed method. Then the multiplierless CSC implementation and the results and analysis are presented in Section 4. Finally, Section 5 concludes the paper.

2. Binary-weights neural network

ANNs are parallel computing machines that learn from examples and by interaction instead of by programming. They are inspired by biology, and composed of elements with functionality similar to a biological neuron. The ANNs can be made in many different ways and can try to mimic the brain in many different ways. Basically, ANNs are widely used in functional approximation and pattern classification applications due to their capability for modeling complex and highly nonlinear functions.

A multilayer neural network consists of a layer of input units, one or more layers of hidden units, and one output layer of units. Each connection between nodes has a weight associated with it. The learning process in multilayer neural network is usually implemented by training data set, because the learning process is achieved by iteratively adjusting the connection weights. The error calculations used to train a multilayer neural network are very important. Like least square, the sum-of-squared error is calculated by looking at the squared difference between what the network’s output for training data and the target value. Formally the equation is the same as one-half the traditional least squares error. For a given set of input vectors $\{x^t\}$, $t = 1, 2, \dots, N$ and desired vectors $\{d^t\}$, the error function is as follows:

$$E(w) = \frac{1}{2} \sum_{t=1}^N (y(x^t, w) - d^t)^2, \quad (1)$$

where N is the total number of training cases, $y(x^t, w)$ is the observed output for the t th training case, and the d^t is the target output for that case.

BWNN introduced in this paper is of three-layer architecture, which is, respectively, input layer, hidden layer and output layer, as shown in Figure 2. The weights of BWNN are only formed by binary digits of 0 and 1. Input layer receives the input data, and then the data is processed by hidden layer and sent to output layer for final result output.

The BWNN can be represented mathematically by the following equations:

$$\text{net}_j^h(\underline{x}) = \sum_{i=1}^p w_{ji}^h \cdot x_i, \quad j = 0, 1, \dots, m, \quad w_{ji}^h \in \{0, 1\} \quad (2)$$

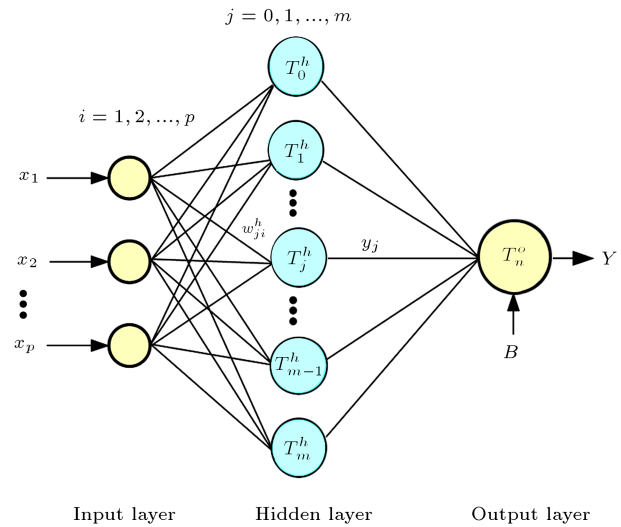


Figure 2. The proposed architecture of BWNN.

$$y_j(\underline{x}) = f(\text{net}_j^h(\underline{x})) = f\left(\sum_{i=1}^p w_{ji}^h \cdot x_i\right) = \left(\sum_{i=1}^p w_{ji}^h \cdot x_i\right) \ll j, \quad (3)$$

$$Y(\underline{x}) = f\left(\sum_j y_j(\underline{x})\right) = \left(\left(\sum_{j=0}^m \sum_{i=1}^p w_{ji}^h \cdot x_i\right) \ll j\right) \gg n + B, \quad (4)$$

where p and m are two constants set by the user, w_{ji} denotes a weight from the input to the hidden layer, and B is a bias.

3. Using HPLA-based BWNN for CSC design

3.1. RGB to YCbCr conversion

The Red, Green and Blue (RGB) color space is widely used throughout computer graphics. The image in RGB color space is not suitable for image compression applications, because the image in RGB color space is highly correlated. In the YCbCr color space, image data consists of three components: luminance (Y), blueness (Cb), and redness (Cr). The first component, luminance, represents the intensity of the image, while the Cb and Cr components called chrominance indicate how much blue and red is used, respectively. The YCbCr color space is a scaled and an offset version of the YUV color space. Y is defined to have a range of 16-235; Cb and Cr are defined to have a nominal range of 16-240 [16]. The conversion from RGB to YCbCr is given by:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}. \quad (5)$$

3.2. Hybrid PSO-based Learning Algorithm (HPLA)

3.2.1. PSO basics

PSO is an evolutionary computation technique developed by Kenney and Eberhart in 1995 [17]. The method has been developed through a simulation of simplified social models. PSO is based on swarms, such as fish schooling and bird flocking. According to the research results for bird flocking, birds find food by flocking (not by each individual). It must also have a fitness evaluation function that takes the particle's position and assigns to it a fitness value. The position with the highest fitness value in the entire run is called the global best (G). Each particle also keeps track of its highest fitness value. The location of this value is called its personal best (P_i). The basic algorithm involves casting a population of particles over the searching space, remembering the best (most fit) solution encountered. At each iteration, every particle adjusts its velocity vector, based on its momentum and the influence of both its best solution and the best solution of its neighbors, and then computes a new point to examine. The studies shows that the PSO has more chance to "fly" into the better solution areas more quickly, so it can discover a reasonable quality solution much faster than other evolutionary algorithms. The original PSO formulate is described by the following equations [17]:

$$V_{i,n}(t+1) = \zeta \cdot V_{i,n}(t) + r_1 \cdot \text{rand}_1 \cdot (P_{i,n}(t) - L_{i,n}(t)) + r_2 \cdot \text{rand}_2 \cdot (G_n(t) - L_{i,n}(t)), \quad (6)$$

$$L_{i,n}(t+1) = L_{i,n}(t) + V_{i,n}(t+1), \quad (7)$$

where n is the dimensional number, i denotes the i th particle in the population, V is the velocity vector, L is the position vector and ζ is the inertia factor. r_1 and r_2 are the cognitive and social learning rates, respectively.

3.2.2. Real-coded GA

A genetic or evolutionary algorithm applies the principles of evolution found in nature to the problem of finding an optimal solution to a solver problem. GA was initially introduced by John Holland in seventies as a special technique for function optimization [18]. It is the most widely known type of evolutionary computation methods [4,19,20]. The basic operators used in GA consist of selection, crossover, and mutation. GA parameters (such as population size, crossover probability

(P_c), and mutation probability (P_m) etc.) interact in complex ways. There is evidence showing that the probabilities of crossover and mutation are critical to the success of GA [21]. Traditionally, determining what probabilities of crossover and mutation should be used is usually done by means of experimental methods.

An individual or solution to the problem to be solved is represented by a list of parameters called chromosome or genome. Genes and chromosomes are the basic building blocks of the GA. The conventional standard GA encodes the optimization parameters into binary code string. The strings are combinations of 0 s and 1 s, which represent the value of a number. During each generation, the chromosomes are evaluated by a fitness function, which is a measure for the quality of the solution. To create the next generation, the individuals with higher fitness value will have higher probability of being selected as candidates for the next generation. In addition, new chromosomes (called offspring) are formed by either merging two chromosomes from the current generation using the crossover operator and modifying a chromosome using the mutation operator.

To reduce computing time, several researchers have tried to use real-coded values instead of binary bit strings for implementing chromosome encoding. The role and behavior of genetic operators in real-coded GA are fundamentally different from those in binary encodings, although motivation of the operators and the framework of GA are similar. It has been confirmed to have better performance than either binary or gray encoding for constrained optimization problems [22–24]. The crossover and mutation operator of real-coded GA are described as follows.

Crossover operators. A crossover is a process where new individuals are created from the information contained within the parents. It is the main genetic operator. The crossover operator of a real-coded GA is constructed by borrowing the concept of linear combination of strings from the area of convex set theory, where the weighted average of two strings (chromosomes) is calculated as follows [25–26]:

$$x'_i = \text{Uniform}(0, 1) \cdot x_i + (1 - \text{Uniform}(0, 1)) \cdot x_{i+1}, \quad (8)$$

$$x'_{i+1} = (1 - \text{Uniform}(0, 1)) \cdot x_i + \text{Uniform}(0, 1) \cdot x_{i+1}. \quad (9)$$

Mutation operators. Random changes are generated to the population in mutation. Mutation can create a new genetic material in the population to maintain the population diversity. This prevents the population from converging towards a local optimum. For a point x_k , the mutated solution is as follows:

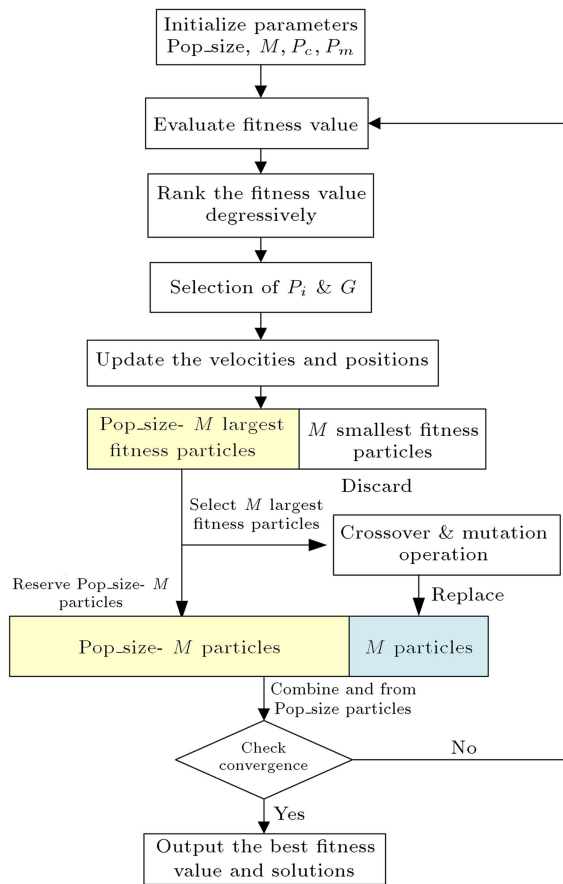


Figure 3. The flow chart of HPLA.

$$x'_k = x_k + \text{rand} \times N(0, 1). \tag{10}$$

3.2.3. Implementation of HPLA algorithm

In the introduced hybrid learning algorithm, when all the particles follow Eqs. (6) and (7) to finish velocities and positions update, we will then retain particles with better fitness in the population, then through crossover and mutation operators by using Eqs. (8)-(10), a lot of new particle strings are then generated so as to replace particles sorted out due to bad fitness. By doing so, the size of population is then retained unchanged. Figure 3 shows the flow chart of HPLA.

3.3. HPLA-based BWNN for CSC design

Figure 4 is the architectural diagram showing the use of HPLA-based BWNN to realize multiplierless CSC. Taking RGB to Y as example, the input value of BWNN should be (R,G,B), and the output value should be Y. If the initial solution of PSO is {χ₁, χ₂, χ₃, n}, then after finishing the training of BWNN, the parameters set R = {w_{ji}, n} extracted from BWNN can then be used to construct multiplierless CSC architecture expected to be obtained. Similarly, as long as the same procedures are followed, hardware architectures such as RGB to Cb and RGB to Cr can be obtained too.

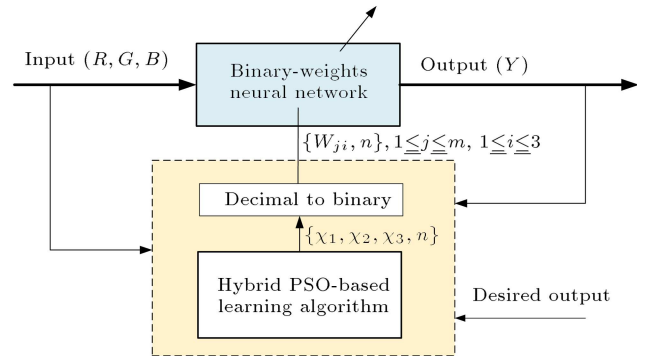


Figure 4. Multiplierless CSC design using HPLA-based BWNN method.

3.3.1. Encoding strategy of particles

In our proposed learning algorithm, each particle is a string encoded by four real number values. When particle string is sent to fitness function to do fitness value evaluation, the first three real number values in particle string will all be converted into binary strings of fixed length to be used as weights for connecting input layer and hidden layer in BWNN. The last real number value is then the number of bits to shift right for the output value of the output neuron. Figure 5 shows the example of the encoding of a particle.

3.3.2. Fitness function evaluation

When too many adders are used, more hardware resources will be consumed during circuit synthesis, hence, in this study, two indexes such as image quality and hardware resource consumed will be considered at the same time, and the designed fitness function is as shown in Eq. (11):

$$\Psi = \left(\alpha \times \sum_{i=1}^N |f_{i,\text{desired}} - f_{i,\text{obtained}}| + \beta \times N_{\text{Add}} \right)^{-1}, \tag{11}$$

where N denotes the total number of training patterns, N_{ADD} denotes the number of adders, parameters α and β are constant between 0 and 1, and f_{i,obtained} is calculated using Eq. (5).

4. Performance evaluation

In this research, the required parameter set setting of the HPLA-based BWNN is as follows: Pop_size = 40, M = 10, maxgen =100, P_c = 0.8, P_m = 0.1, α = 0.005, β = 1, m = 9, and the bias of BWNN B ∈ {16, 128}. The search space of χ₁, χ₂, and χ₃ is in the range of [0, 1023], and the search space of n is in the range of [0, 31]. The training data set used by HPLA-based BWNN consists of 729 data patterns which are the sub-sampled data in the RGB color space. The designed architectures were described in Verilog-HDL and synthesized for Altera Cyclone II EP2C70 FPGA, with the aid of the tool Quartus II.

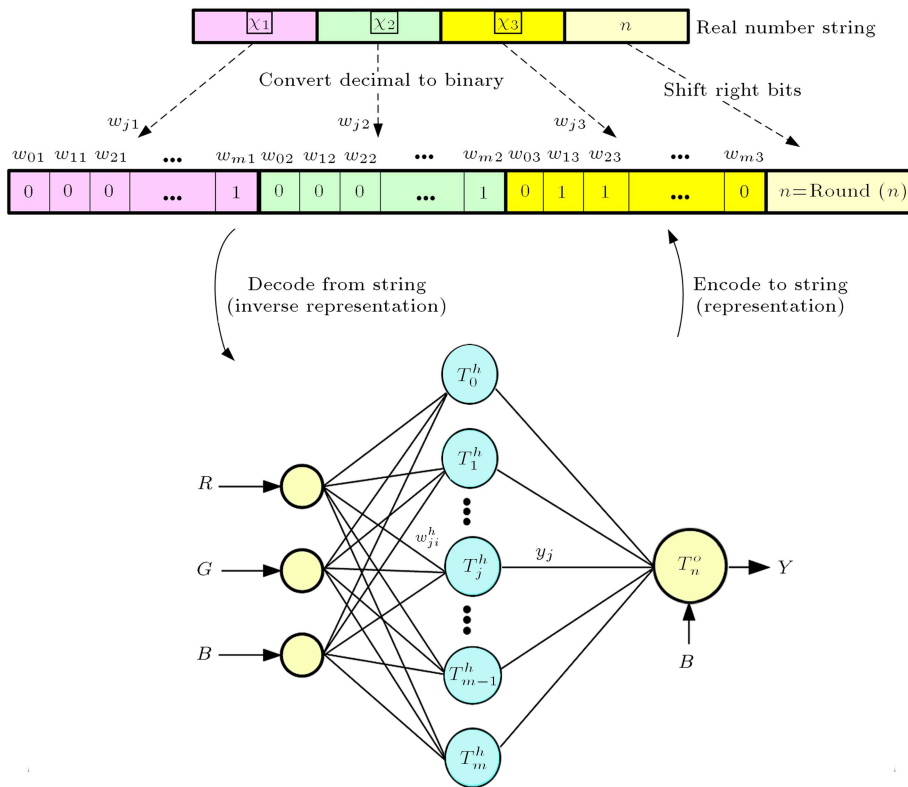


Figure 5. A coded particle string.

When BWNN that is responsible for the implementation of color space conversion, such as RGB to Y, RGB to *Cb*, and RGB to *Cr*, has been finished with training, the knowledge and related parameters extracted from black boxes are shown respectively as in Eqs. (12)-(14):

$$x_i = \begin{bmatrix} R \\ -G \\ -B \end{bmatrix}, \quad n = 11, B = 128 \quad (14)$$

Eq. (15) shows the multiplierless CSC architecture from BWNN framework introduced in this research:

RGB to Y:

$$w_{ji}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix},$$

$$x_i = \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \quad n = 11, B = 16 \quad (12)$$

$$Y = \left(((R \ll 9) + (G \ll 2) + (G \ll 3) + (G \ll 4) + (G \ll 5) + (G \ll 6) + (G \ll 7) + (G \ll 8) + (G \ll 9) + (B \ll 5) + (B \ll 6) + (B \ll 7)) \gg 11 \right) + 16,$$

RGB to *Cb*:

$$w_{ji}^T = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$x_i = \begin{bmatrix} -R \\ -G \\ B \end{bmatrix}, \quad n = 11, B = 128 \quad (13)$$

$$Cb = \left((-R \ll 2) - (R \ll 3) - (R \ll 5) - (R \ll 8) - (G \ll 4) - (G \ll 6) - (G \ll 9) + (B \ll 2) + (B \ll 7) + (B \ll 8) + (B \ll 9) \gg 11 \right) + 128,$$

RGB to *Cr*:

$$w_{ji}^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

$$Cr = \left(((R) + (R \ll 4) + (R \ll 7) + (R \ll 8) + (R \ll 9) - (G \ll 8) - (G \ll 9) - (B \ll 4) - (B \ll 7)) \gg 11 \right) + 128. \quad (15)$$

In order to enhance the processing efficiency of the mentioned CSC architecture on color space conversion, in this study, how to perform pipelined design on the electronic circuit architecture, described in Eq. (15), will be further described too. The concept of pipelined design has been widely applied in all kinds of engineering fields; for example, the production line of the automobile plant is operated based on the concept similar to pipeline processing. For the hardware circuit, the application of pipeline architecture is similar to the production line in the automobile plant, and the data in massive processing is similar to the objects in the production line of a plant. Moreover, several functional units with special functions, for example, sub-circuits, such as adder and subtractor, are used to perform data processing. The processed data are coordinated and integrated through register, and then are sent to the next stage for subsequent processing. According to Eq. (15), the circuit representation can be designed into pipeline architecture, which is shown in Figures 6-8. All these color conversion circuits are divided into five stages, and the sub-circuit in each stage is formed by signed adder; each stage

Table 1. Comparison of the time needed for processing 512×512 size color image using Nios II C/C++ language.

Method	Time cost (s)	Clock frequency (MHz)
Traditional CSC	54	100
Proposed CSC	1.2	100

is set up with pipeline register at the data output location of the path to store temporarily the processed data.

Here, the calculation time needed for several different CSC architectures proposed in this paper in processing single image will be compared so as to understand the performance of the proposed methods in performing color conversion. The related comparison results are as shown in Table 1. When we are executing traditional floating-point based CSC software programs in Nios II processor, the color conversion processing time needed for single 512 × 512 size image was 54 s. However, if the proposed CSC architecture is used instead to implement the same conversion task, the time needed will be only 1.2 s. Therefore, it is clear that the CSC architecture realized in this study has a very good performance, which is sufficient to deal with the need of real-time image processing. In addition to that, as compared to the CSC architecture used and proposed by Li et al. in 2013 [4], which used 47 adders, the proposed CSC architecture only needs 32 adders. This shows significant difference in the consumption of hardware resource in both cases.

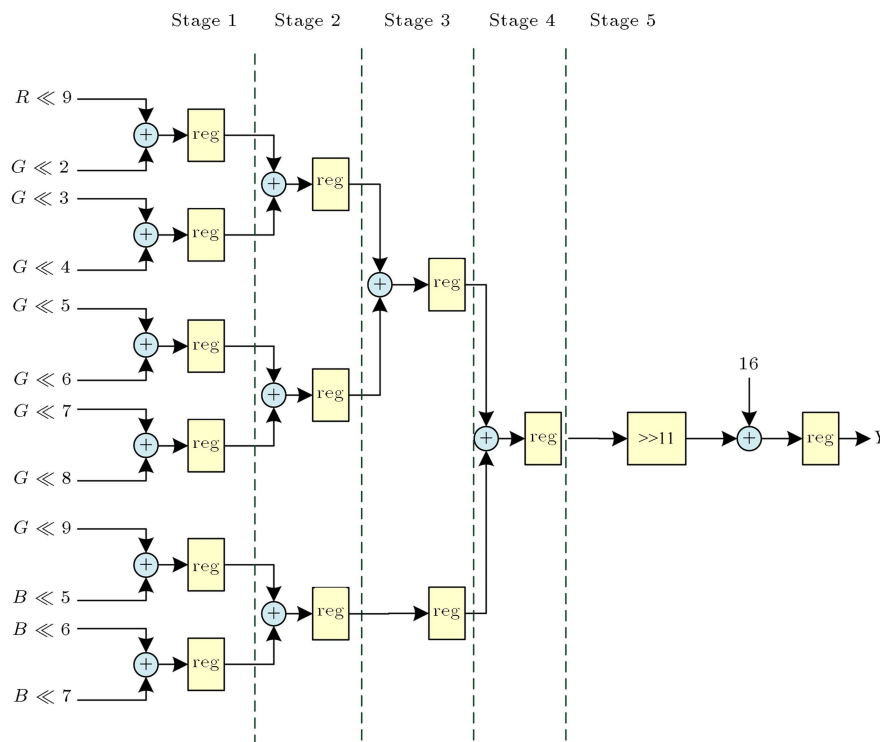


Figure 6. The pipelined architecture for RGB to Y.

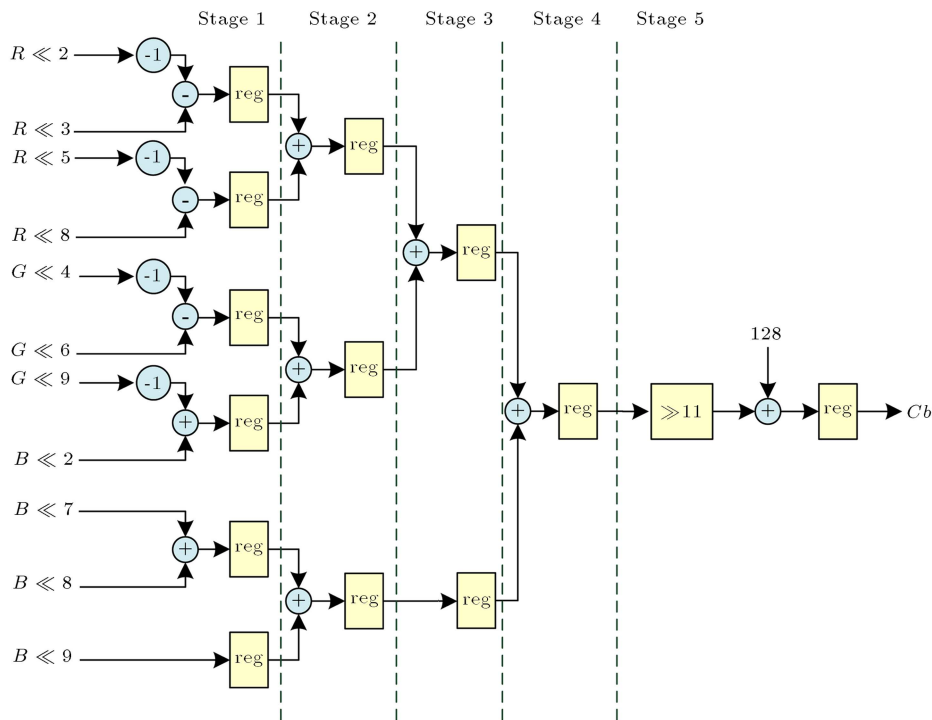


Figure 7. The pipelined architecture for RGB to C_b .

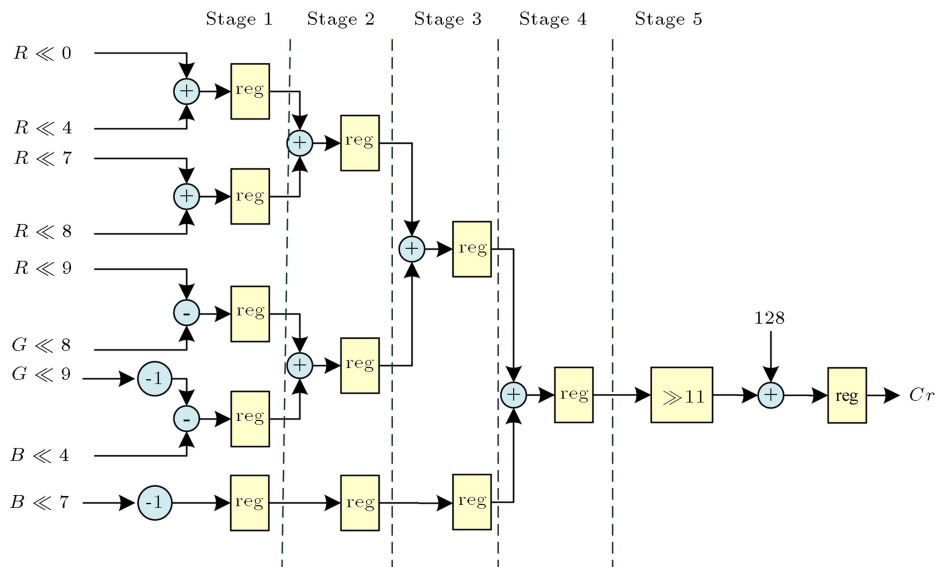




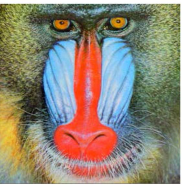
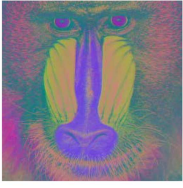

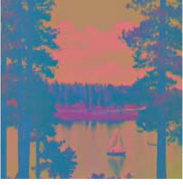
Figure 8. The pipelined architecture for RGB to C_r .

When we further use Verilog-HDL to describe circuit architectures, such as Figures 6-8, and download them to Cyclone II FPGA chip, the performance data of hardware circuit is then as shown in Table 2. From the data listed in the table, it can be seen that the hardware architecture realized in this research can have a maximum operating frequency of 306 MHz in Cyclone II FPGA chip, which shows better performance as compared to the CSC circuit proposed by Bensaali and Amira [27], CAST. Inc. [28], ALMA. Tech. [29], and Amphion Ltd. [30], hence, it can be proved that the

Table 2. Performance comparison with existing CSC hardware architecture.

Design parameters	Speed (Mhz)
Proposed CSC core	306
Bensaali and Amira [27]	234
CAST. Inc. [28]	112
ALMA. Tech. [29]	105
Amphion Ltd. [30]	90

Table 3. A comparison between color space conversion result and related performance for test images.

Test image	Conversion results	RMSE	Computation time	
			Nios II	Proposed CSC core
		Y 0.3441 Cb 0.7222 Cr 0.1623	1.2 s	0.856 ms
		Y 0.7803 Cb 0.5348 Cr 0.3991	1.2 s	0.856 ms
		Y 0.4793 Cb 0.5483 Cr 0.2877	1.2 s	0.856 ms

above CSC circuit architecture has, in fact, excellent characteristic and practical value.

Next, three 512×512 color images which are frequently used for image quality test in the academy were used to perform the experiment so as to compare the execution speed and image quality, which are as shown in Table 3. From Table 3, it is clear that when hardware method is used to realize CSC, the processing speed will be far faster than that realized by Nios II C/C++ software design method; in other words, the color conversion task of the image can be indeed processed in real time. In addition, RMSE index in the table shows that the proposed CSC can retain good image quality while doing color space conversion, and the error between it and the result obtained from Eq. (5) is relatively small (which is about in the range of 0.2~0.8), hence, in real application, almost no influence will be seen. The calculation of RMSE is shown in Eq. (16):

RMSE =

$$\sqrt{\frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (f_{\text{Template}}(i, j) - f_{\text{obtained}}(i, j))^2}, \quad (16)$$

where M and N are the number of rows and columns of the image.

5. Conclusion

In this paper, a method using HPLA-based BWNN to design high-speed shift-and-add based CSC is intro-

duced. In the learning strategy, we have introduced two consideration factors of RMSE and adder number at the same time, hence, the realized circuit can, under the minimal hardware consumption, still retain a very good color conversion quality. In addition, the proposed CSC architecture is pretty simple and has very high execution speed; hence, it is very suitable to be integrated into all kinds of image application systems that need real time processing for taking care of the task of fast color space conversion.

Acknowledgments

This research was supported by the National Science Council of the Republic of China under Contract No. NSC 102-2221-E-130 -021.

References

1. Yang, Y., Peng, Y. and Liu, Z. "A fast algorithm for YCbCr to RGB conversion", *IEEE Trans. Consumer Electron.*, **53**(4), pp. 1490-1493 (2007).
2. Gustafsson, O. and Qureshi, F. "Addition aware quantization for low complexity and high precision constant multiplication", *IEEE Signal Process. Lett.*, **17**(2), pp. 173-176 (2010).
3. Webb, J.L.H. "Efficient table access form reversible variable-length decoding", *IEEE Trans. Circuits Syst. Video Technol.*, **11**(8), pp. 981-985 (2001).
4. Li, S.A., Chen, C.Y. and Chen, C.H. "Design of a shift-and-add based hardware accelerator for color space conversion", *J. Real Time Image Process* (2013). doi: 10.1007/s11554-013-0324-7

5. Zhang, B.T. and Muhlenbein, H. “Evolving optimal neural networks using genetic algorithms with Occam’s razor”, *Complex Syst.*, **7**(3), pp. 199-220 (1993).
6. Li, S., Yuan, J., Yue, X. and Luo, J. “The binary-weights neural network for robot control”, *Proceedings of 2010 3rd RAS & EMBS*, pp. 765-770 (2010).
7. Palmes, P.P., Hayasaka, T.C. and Usui, S. “Mutation-based genetic neural network”, *IEEE Trans. Neural Networks*, **16**(3), pp. 0587-600 (2005).
8. Yao, X. and Liu, Y. “A new evolutionary system for evolving artificial neural networks”, *IEEE Trans. Neural Networks*, **8**(3), pp. 694-713 (1997).
9. Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Neural Selection*, MIT Press (1992).
10. Berlanga, A., Isasi, P., Sanchis, A. and Molina, J.M. “Neural networks robot controller trained with evolution strategies”, *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 413-419 (1999).
11. Ratnaweera, A., Saman, K. and Watson, H.C. “Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients”, *IEEE Trans. Evol. Comput.*, **8**(3), pp. 240-255 (2004).
12. Juang, C.F. “A hybrid genetic algorithm and particle swarm optimization for recurrent network design”, *IEEE Trans. Syst. Man Cybernet.*, **32**, pp. 997-1006 (2004).
13. Da, Y. and Ge, X.R. “An improved PSO-based ANN with simulated annealing technique”, *Neurocomput. Lett.*, **63**, pp. 527-533 (2005).
14. Van den Bergh, F. and Engelbrecht, A.P. “A cooperative approach to particle swarm optimization”, *IEEE Trans. Evol. Comput.*, **8**(3), pp. 225-239 (2004).
15. Premalatha, K. and Natarajan, A.M. “Hybrid PSO and GA models for document clustering”, *Int. J. Advance. Soft Comput. Appl.*, **2**(3), pp. 1-19 (2010).
16. Hsu, R.L., Abdel-Mottaleb, M. and Jain, A.K. “Face detection in color images”, *IEEE Trans. Pattern Anal. Mach. Intell.*, **24**(5), pp. 696-707 (2002).
17. Eberhart, R. and Kennedy, J. “Particle swarm optimization”, *Proceedings of IEEE International Conference on Neural Network*, pp. 1942-1948 (1995).
18. Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, USA (1975).
19. Back, T., *Evolutionary Algorithm: Comparisons of Approaches*, Chapman and Hall, Cambridge, UK (1994).
20. Page, A.J. and Naughton, T.J. “Framework for task scheduling in heterogeneous distributed computing using genetic algorithms”, *Artif. Intell. Rev.*, **24**(3-4), pp. 415-429 (2005).
21. Ochoa, G., Harvey, I. and Buxton, H. “On recombination and optimal mutation rates”, *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 488-495 (1999).
22. Lee, K.Y. and Mohamed, P.S. “A Real-coded genetic algorithm involving a hybrid crossover method for power plant control system design”, *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 1069-1074 (2002).
23. Yalcinoz, T. and Altun, H. “Power economic dispatch using a hybrid genetic algorithm”, *IEEE Power Eng. Rev.*, **21**(3), pp. 59-60 (2001).
24. Ramos, R.M., Saldanha, R.R., Takahashi, R.H.C. and Moreira, F.J.S. “The real-biased multiobjective genetic algorithm and its application to the design of wire antennas”, *IEEE Trans. Magn.*, **39**(3), pp. 1329-1332 (2003).
25. Kaelo, P. and Ali, M.M. “Integrated crossover rules in real coded genetic algorithms”, *Eur. J. Oper. Res.*, **176**(1), pp. 60-76 (2007).
26. Ripon, K.S.N., Kwong, S. and Man, K.F. “A real-coding jumping gene genetic algorithm (RJGGA) for multiobjective optimization”, *Inform. Sciences*, **177**(2), pp. 632-654 (2007).
27. Bensaali, F. and Amira, A. “Design and efficient FPGA implementation of an RGB to YCrCb color space converter using distributed arithmetic”, *Int. J. Graph. Vis. Image Process.*, **5**(1), pp. 37-47 (2004).
28. Application Note: CSC color space converter, CAST, Inc., <http://www.cast-inc.com> (2002).
29. Datasheet: High performance color space converter, ALMA Technologies, <http://www.alma-tech.com> (2002).
30. Datasheet: Color space converters, Amphion semiconductor Ltd, DS6400 V1.1, <http://www.amphion.com> (2002).

Biography

Ching-Yi Chen received his PhD degree from Tamkang University, New Taipei City, Taiwan, in 2006. He joined the Department of Information and Telecommunications Engineering at Ming Chuan University in 2007 as an Assistant Professor. Currently, he is an Associate Professor at the department. His main research interests include swarm intelligence, pattern recognition, and embedded systems.