AUTOMATIC DESIGN OF SHIFT-AND-ADD BASED COLOR SPACE CONVERTER USING A GENETIC ALGORITHM

Ching-Yi Chen1, Ching-Han Chen2, Chih-Hao Ma2 and Po-Yi Wu2

1Department of Information and Telecommunications Engineering, Ming Chuan University, Taoyuan, Taiwan
2Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan
E-mail: chingyi@mail.mcu.edu.tw

ICETI 2012-P1002_SCI
No. 13-CSME-83, E.I.C. Accession 3541

ABSTRACT
The main purpose of this paper is to investigate a novel design method using a genetic algorithm (GA) to automatically evolve the multiplierless CSC circuit architecture. In order to demonstrate the effectiveness of the described design method, several test images are adopted respectively to perform RGB to YCbCr color conversion experiment. The experimental results represent that the performance of the implemented hardware architecture is good when carrying out color space conversion from RGB to YCbCr. It also has the advantage of being high-speed, low-complexity, and low-area.

Keywords: color space converter; genetic algorithm; FPGA.

CONCEPTION AUTOMATIQUE POUR CONVERTISEUR D’ESPACE BASÉE SUR LA TECHNIQUE DE CODE COULEUR “SHIFT-AND-ADD” À L’AIDE D’UN ALGORITHME GÉNÉTIQUE

RÉSUMÉ
Le but principal de cette recherche est d’étudier une méthode de conception innovatrice utilisant un algorithme génétique (GA) pour automatiquement développer une architecture de circuit sans multiplicateur. Pour démontrer l’efficacité de la méthode, plusieurs tests d’images ont été adoptés pour mener les expériences de conversion de RGB à YCbCr. Les résultats expérimentaux montrent que la performance de l’architecture du matériel est bonne pour le convertisseur d’espace RGB à YCbCr, et possède aussi l’avantage de la rapidité, le peu de complexité, l’espace réduit.

Mots-clés : convertisseur d’espace avec couleur ; algorithme génétique ; FPGA.
1. INTRODUCTION

Color space conversion has come to play a very important role in video processing and transmission technology; generally speaking, it is not practical to try to transmit images in RGB color space because it requires quite high bandwidths. The human eye is more sensitive on brightness change than on color change. Hence, if luminance-chrominance color space is used for color image transmission, data storage amount and bandwidth can be reduced, and the price is on the negligence of very tiny or almost unrecognized color change information. YCbCr is a hardware-oriented color model. When video processing is done in YCbCr color space, we can get very high compression ratio and transmission rate, hence, in most image or video compression applications (for example, JPEG and MPEG, etc.), we will usually need to use the transformation between RGB and YCbCr color space.

In many digital signal processing (DSP) algorithms, multiplier coefficients are either floating-point numbers, 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 [1]. Floating-point multiplication in either a general-purpose processor or in a custom hardware implementation is generally an expensive operation in terms of latency or hardware area. 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 equation contains massive floating-point operations, even if we use directly the floating-point intellectual property provided by Altera Corporation [2] for the circuit design, the cost of hardware resource needed will still be far higher the cost of the hardware architecture needed for processing integer operation; next, in the conversion equation, massive multiplication operation 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.

The achievement of automatic evolution of electronic circuit through evolutionary algorithms (EAs) is a new and expected research field. As early as 1966, Neumann [3] had proposed the concept of hardware evolution. In the beginning of 1990, the fast development in programmable logic device product triggered research fever based on the use of reconfigurable hardware as the realization platform of hardware circuit. Some researches started to try the association of soft computing and reconfigurable hardware to perform the circuit design, and this is the beginning of the development of EHW theory. The initial objective of such a technique was to be used as a new design method for digital circuits, and the researchers used biological evolutionary rules to let the system design autonomously the digital circuits to be realized on reconfigurable hardware platform. Through evolutionary computation methods, an expected research field of hardware circuit automatic evolution can be achieved. In 1992, De Garis [4] proposed the first circuit design concept that associates machine learning, artificial intelligence and hardware planning. Later, Koza et al. [5], Thompson [6] and Sipper et al. [7] also developed the theory and method of hardware evolution one after another; such a design procedure, applying biological evolution technology to do hardware circuit design and synthesis, is called Evolvable Hardware (EHW) [8–10].

If the division is made according to the level of chromosome representation, the design approach of EHW can be divided into logic gate-level EHW and function-level EHW. In the former case, EAs were used to select automatically basic logic gates such as AND gate, OR gate or XOR gate to generate the digital circuit to be realized; and in the latter case, some functional modules such as multiplexer, adder and multiplier were used as the basic building blocks. Aguirre et al. [11, 12] had introduced a Boolean function design method using multiplexer and Genetic Programming (GP); in the function-level EHW platform proposed by Higuchi et al. [13], the application of EHW was enhanced to a higher level, and so the actual application need was better met. Lots of researchers had followed this basis to develop some important applications, for example, low power consumption circuit [14], lossless image compression circuit [15], multiplier block synthesis [16], and digital image filter [17, 18], etc.
The main concept of the EHW is to convert the circuit architecture into gene encoding that can be operated through evolutionary computation, meanwhile, in the iteration process, the structure of the digital circuit is adjusted dynamically so as to search the optimal circuit that meets the condition limitation of the problem. Biological evolution theory thought that biological evolution was based on three factors such as genetics, natural selection and mutation. Evolutionary computation is a random-search method to simulate the survival competitions among species; the researchers can use a simple encoding scheme to represent all kinds of complex circuit structure, meanwhile, through genetics operation and survival mechanism of the fittest, the search on optimal solutions from all the individuals in solution space is then guided. In the EHW the circuit combinations of all functions that can be formed by a reconfigurable hardware platform is the solution space to be searched, and the encoding value of each individual in the space represents certain possible hardware architecture. The use of the search method on evolutionary computation can, in the iteration process, continuously change the device type of hardware circuit and the connection relation among devices, then in the solution space, one hardware architecture that can satisfy spec requirements such as function, dimension, operation speed or power consumption is found.

In the proposed method, we use the genetic algorithm (GA) to automatically evolve the efficient hardware architecture of the CSC. The architecture employs only a few shift and addition operations to replace the complex floating-point multiplications. The target hardware for the implementation and verification of the proposed approach is Altera DE2-70 board equipped with a Cyclone II 2C70 FPGA chip. Through our proposed design procedure, the EHW can not only process the design of gate-level digital circuits or the realization of Boolean functions, but can also be further applied to the design of video processing circuit architecture such as CSC, etc.

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

2. GENETIC ALGORITHM

EAs are general-purposed stochastic search methods. These algorithms are based on the evolution of a population toward the solution of a certain problem. EAs can be used successfully in many applications requiring the optimization of a certain multi-dimensional function. The population of possible solutions evolves from one generation to the next, ultimately arriving at a satisfactory solution to the problem. In recent years, EAs have increasingly been used in solving many difficult optimization problems. Evolutionary method includes the following techniques: GA, Evolutionary Programming (EP), Evolution Strategies (ES), Ant Colony Optimization [19], and Particle Swarm Optimization [20], etc. EAs start with an arbitrarily initialized population of coded individuals, each of which represents a search point in the space of potential solution. The advantage of each individual is evaluated by a fitness function which is defined from the objective function of the optimization problem. Then, the population evolves toward increasingly better regions of the search space by means of both random and probabilistic biological operations.

A genetic or EA applies the principles of evolution found in nature to the problem of finding an optimal solution to a solver problem. GA was developed by Holland [21]. Although this algorithm emerged simultaneously with two other streams known as ES and EP, GA is today the most widely known type of evolutionary algorithms [22, 23]. The diversity of applications by utilizing GA in search and optimization is also quite impressive. The basic operators used in GA consist of selection, crossover, and mutation. A GA operates through a simple cycle of stages:
Step (1) Define a genetic representation of the system, and create a “population” of strings (i.e. chromosomes).

Step (2) Evaluate the fitness of each string in the population.

Step (3) Create a new population by repeating the following steps until the new population is complete:

1. Selection: Select parent strings from a population according to their fitness (the better fitness, the bigger chance to be selected).
2. Crossover: With a crossover probability, cross over the parents to form new offsprings.
3. Mutation: With a mutation probability, mutate new offspring at each locus (position in string).
4. Accepting: Place the new offspring in a new population.

Each cycle in GA produces a new generation of possible solutions for a given problem. In the first phase, an initial population, describing representatives of the potential solution, is created to initiate the search process. GA typically represents solutions as binary strings. The elements of the population are encoded into bit-strings, called chromosomes. In binary encoding, every chromosome is a string of bits, 0 or 1. The elements of the population are encoded into bit-strings, called chromosomes.

The performance of the strings, often called fitness, is then evaluated with the help of some functions, representing the constraints of the problem. Depending on the fitness of the chromosomes, they are selected for a subsequent genetic manipulation process. It should be noted that the selection process is mainly responsible for assuring survival of the best-fit individuals. After the selection of the population strings is over, the genetic manipulation process consisting of two steps is carried out. In the first step, the crossover operation that recombines the bits (genes) of every selected chromosomes pairs is executed. The second step in the genetic manipulation process is termed mutation, where the genes at one or more randomly selected positions of the chromosomes are altered. It randomly alters each gene with a small probability. The mutation process helps to overcome trapping at local minima. The offsprings produced by the genetic manipulation process are the next population to be evaluated.

The pseudo-code of the standard GA is given in Fig. 1.

3. BASIC CONCEPT OF GA-BASED EVOLUTIONARY CSC DESIGN

3.1. Color Space 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 the quantum of blue and red respectively. This color space is based on the space YUV, that is used in the PAL European TV standard, where component Cb and Cr are shifted scales of U and V components [24].

The conversion from RGB to YCbCr is given by

\[
\begin{align*}
Y & = 0.257 \cdot R + 0.504 \cdot G + 0.098 \cdot B + 16 \\
Cb & = -0.148 \cdot R - 0.291 \cdot G + 0.439 \cdot B + 128 \\
Cr & = 0.439 \cdot R - 0.368 \cdot G - 0.071 \cdot B + 128
\end{align*}
\]
**Procedure GA:**

{  
    \( t=0; \)
    
    Randomly generate initial population \( \text{POP}(t) \);
    Evaluate \( \text{POP}(t) \) to obtain its fitness values;

    \textbf{while} (not done) {
        
        Select \( \text{POP}(t+1) \) from \( \text{POP}(t) \);
        Crossover \( \text{POP}(t+1) \) based on probability \( r_c \);
        Mutate \( \text{POP}(t+1) \) based on probability \( r_m \);
        Evaluate \( \text{POP}(t+1) \);
        \( t=t+1; \)
    }

    \textbf{until} (termination is met)

}  

Fig. 1. The pseudo-code of the GA.

Figure 2 shows a direct mapping of the above three equations. This conversion is performed as digital multiplication and addition utilizing floating-point multipliers and summation circuits as shown in Fig. 2.

**3.2. Evolvable Hardware Framework for Multiplierless CSC Design**

The evolution process is used to develop the optimal design configuration of the CSC based on shift-and-add architecture. In this process, an initial design configuration represented by a set of chromosomes is selected randomly. The performance of the configuration, often called fitness, is then evaluated with the help of a fitness function, representing the constraints of the design problem. The process is repeated for all other configurations in the initial population and a new population is generated. After a number of iterations, the best fit design configuration is obtained. The genetic algorithm described above uses mutation and crossover operations to generate the new population.
The structure of a population, composed of \( P \) chromosomes and “\( 3 \times (m + 1) + (n + 1) \)” genes in each chromosome.

### 3.2.1. Encoding of the Chromosomes

An individual, or solution to the problem to be solved, is represented by a list of parameters, called chromosome or genome. The most common way of representing a chromosome is performed using fixed-length binary string which is a combination of 0s and 1s representing a number in binary form. The initial population is a collection of randomly generated individuals encoded to binary strings and representing different solutions to the optimization problem. In the mentioned method, we have used fixed-length binary string to form and realized the color conversion circuit configuration. The structure of the entire encoding string can be divided into four sections: the first three sections are used respectively to generate the shift-and-add operation circuits of three input color values of R, G, and B, and the final section is used to decide the right-shift operation values of all the summations. The structure of a population, chromosome and gene are illustrated in Fig. 3.

Figure 4 describes the coding method of binary string chromosome and the corresponding circuit configuration. In the string section that is labeled “Shift/add of R”, the bit values of the locations of index 1, index \( (m - 2) \) and index \( (m - 1) \) are “1”, and the rest of bits are “0”, which means the realization of an operation circuit of \((R \text{ shl} 1) + (R \text{ shl} (m - 2)) + (R \text{ shl} (m - 1))\). Similarly, in the string section that is labeled “Shift/add of G”, only the bit values of locations of index 1 and index \( m \) are “1”; in the word string section that is labeled “Shift/add of B”, the bit values of the locations of index 1, index 2 and index \( m \) are “1”, which means respectively the operation circuit structure of \((G \text{ shl} 1) + (G \text{ shl} m)\) and \((B \text{ shl} 1) + (B \text{ shl} 2) + (B \text{ shl} m)\). From the output summation result of the corresponding circuit of the above three sections, through the right-shift number (to convert the binary string in the section into decimal number) obtained from the encoding value of the fourth section, we can then perform shift operation and add it with the constant item to get the final output result of color conversion.

### 3.2.2. Fitness Function Calculation

The fitness of an evolved circuit is a measure of how well the circuit matches the design specification. In this paper, the fitness function \( FV \) for each individual is computed by

\[
FV = \frac{1}{\sum |\text{Color}_{\text{desired output}} - \text{Color}_{\text{obtained output}}|}
\]  

where the fitness value is calculated by accumulating the difference between the sum of the desired output and the circuit output, and the desired output is calculated using Eq. (1).
Mapping a binary string chromosome into the design domain to create a color space conversion hardware circuit, where “L” and “R” denote the left-shift (shl) and right-shift (shr) operators, respectively.

Fig. 5. An example of two-points crossover between two chromosomes.

3.2.3. Crossover and Mutation Operators

Crossover involves combining different parts of two parents to make two different offsprings. In the crossover operator, new chromosomes are created by exchanging information among the chromosomes of the current generation in the mating pool. There are many different types of crossover operators for ordered chromosomes. The two-point crossover operation is illustrated in Fig. 5.

Crossover operators are mainly responsible for the search of new and better offsprings, but they do not have the ability to generate any new genetic material. As in nature, mutation is used for changing chromosomes randomly and introducing new genetic material into the population. Figure 6 shows the genetic manipulation process, where the genes at one or more randomly selected positions of the chromosomes are altered. It randomly alters each gene with a small probability.
Finally, the proposed GA-based evolutionary CSC design process is summarized in the following steps:

Step (1) Represent the problem variable domain as a chromosome of a fixed length, and define a fitness function to measure the performance, or fitness, of an individual in the problem domain.

Step (2) Set the number of individuals ($P$), number of generations (maxgen), probabilities of crossover ($r_c$) and mutation ($r_m$), and then generate randomly an initial diversified population.

Step (3) Evaluate the fitness of each design configuration (i.e. individual) in the population using the problem’s fitness measure.

Step (4) Create new individuals for the population by applying the following genetic operations with specified probabilities:

1. Selection: Copy the selected individuals to the new population. The better the fitness of an individual,
2. Crossover: Perform crossover on the selected parents at probability $r_c$ to form new offsprings.

3. Mutation: Create one new offspring individual for the new population by randomly mutating a randomly chosen part of one selected individual.

Step (5) Repeat Steps (3)–(4) until a stop criterion is satisfied or a predefined number of iterations is completed.

4. IMPLEMENTATION RESULTS AND DISCUSSIONS

In this study, we have set up the parameters of GA to be $P = 80$, $\text{maxgen} = 200$, $r_c = 0.7$, and $r_m = 0.1$; and we have set up $m = 9$, $n = 4$ for binary string chromosome, hence, the encoding length of chromosome should be $3 \times (m + 1) + (n + 1) = 35$ bits. The simulation environment used for the experiment is the PC with built-in Intel(R) Core(TM) i5-650 CPU (4 M Cache, 3.20 GHz), and memory of 4.00 GB RAM. The designed architectures were described in Verilog-HDL and synthesized for Altera FPGAs of the family EP2C70F896C6, with the aid of the tool Quartus II 9.1.

Equation (3) shows hardware architecture configuration through the automatic evolution from GA-based evolvable hardware framework introduced in this research, and Fig. 7 shows the corresponding block diagram:
Table 1. Software/hardware implementations for RGB to YCbCr CSC comparisons.

<table>
<thead>
<tr>
<th>Test image</th>
<th>Software Implementation (using Eq.(1))</th>
<th>Hardware Implementation (using Eq.(3))</th>
<th>RMS error</th>
<th>Computation Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>PC (Eq.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nios II (100Mhz)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hardware (Eq.3)</td>
</tr>
<tr>
<td>Lake</td>
<td><img src="image1.png" alt="Lake Image" /></td>
<td><img src="image2.png" alt="Lake Image" /></td>
<td>0.32</td>
<td>20.7m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.21m</td>
</tr>
<tr>
<td>Baboon</td>
<td><img src="image3.png" alt="Baboon Image" /></td>
<td><img src="image4.png" alt="Baboon Image" /></td>
<td>0.53</td>
<td>21.6m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.21m</td>
</tr>
<tr>
<td>Pepper</td>
<td><img src="image5.png" alt="Pepper Image" /></td>
<td><img src="image6.png" alt="Pepper Image" /></td>
<td>0.61</td>
<td>20.6m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.21m</td>
</tr>
</tbody>
</table>

Y = (((R) + (R shl1) + (R shl4) + (R shl9) + G + (G shl1) + (G shl2) + (G shl3) + (G shl4) + (G shl5) + (G shl6) + (G shl7) + (G shl8) + (G shl9) + (B) + (B shl1) + (B shl2) + (B shl3) + (B shl6) + (B shl7)) shr11) + 16

Cb = (((-(R) - (R shl3) - (R shl5) - (R shl8) - (G) - (G shl5) - (G shl6) - (G shl9) + (B) + (B shl1) + (B shl2) + (B shl3) + (B shl7) + (B shl8) + (B shl9)) shr11) + 128

Cr = (((R shl1) + (R shl2) + (R shl7) + (R shl8) + (R shl9) - (G shl4) - (G shl5) - (G shl6) - (G shl7) - (G shl9) - (B shl4) - (B shl7)) shr11) + 128

The Altera Nios II processor is a 32 bit soft core processor provided by the Altera, it is the most widely used soft processor in the FPGA industry. When we use the Nios II IDE to run CSC software programs on the Nios II processor (system frequency: 100 MHz), it takes 54.3 seconds for conversion by applying Eq. (1) onto a 512 × 512 size image to do color space conversion, whereas by applying Eq. (3) to do the same conversion task, it only takes 1.51 seconds. As a result, the shift-and-add based CSC has a distinct advantage over floating-point multiplier based CSC in high-speed characteristic. Some other simulation results and related performance investigation content are listed in Table 1.

In Table 1, we have adopted images of three different colors, namely, Lake image (512 × 512), Baboon image (512 × 512), and Pepper image (512 × 512) as the test data, meanwhile, performance indexes such as
computation time, root-mean-square (RMS) error for the hardware (defined in Eq. (3) and software (defined in Eq. (1)) are compared respectively. From the table, we can clearly see that when hardware and software are implementing color space conversion of the image, the RMS error between both of them is very small; however, the execution speed of hardware is far faster than that of the software. In addition, our designed multiplierless CSC, in the hardware circuit realization, also has the advantage of requiring lesser hardware area resource. The architecture only needs to use 476 logic elements of Cyclone II EP2C70F FPGA, as compared to the architecture in Fig. 2 that requires 2127 logic elements, thus lots of hardware area resource can be saved. The calculation of RMS error is as follows:

\[
\text{RMS error} = \left( \frac{1}{MN} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (I_{\text{standard}}(i,j) - I_{\text{hardware}}(i,j))^2 \right)^{1/2}
\]

(4)

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

5. CONCLUSIONS

Human’s vision is more sensitive on luminance, however, the RGB color system cannot process the luminance of images. Moreover, higher bandwidth is needed when the image formed by RGB is transmitted, and the storage will also occupy more memory space. In order to process the video or image format more effectively, many communication techniques or broadcasting communication system will need to use color space transformation mechanism between RGB and YCbCr so as to separate luminance apart from color information and to achieve the objectives of bandwidth saving and the enhancement of image compression ratio. However, since the conversion from RGB to YCbCr will take massive computation, it is commonly seen in the literature to use hardware architecture to enhance the conversion efficiency. It is hoped that the application need of real time processing can be achieved through low-complexity, low-area, high speed and high performance circuit architecture.

In this paper, a method using GA to make automatic evolution of the hardware architecture of RGB to YCbCr CSC is introduced. The converter hardware architecture uses tiny amounts of shift and addition operations to realize the operation function of floating-point multiplications with large computing amount, hence, it has advantages such as easy turning into hardware, low hardware area and high-speed, etc. According to the experiment results, for the color space converter we realized in Cyclone II EP2C70F, the FPGA chip needs only 4.21 ms to complete the conversion of a 512 × 512 color image. In the speed, the efficiency is much higher than that of software; and only the resource of 476 logic elements of FPGA chip needs to be used, the consumed hardware area resource is pretty small, hence, it has industrialized practical value.

ACKNOWLEDGEMENT

This research was supported by the National Science Council of the Republic of China under Contract No. NSC 100-2221-E-130-019.

REFERENCES


