Available online at www.sciencedirect.com
SCIENCE DIRECT®
Image and Vision Computing 23 (2005) 935–942
www.elsevier.com/locate/imavis

Accelerating colour space conversion on reconfigurable hardware

F. Bensaali*, A. Amira

School of Computer Science, Queen's University of Belfast, University Road, Belfast BT7 1NN, UK

Received 27 June 2004; received in revised form 25 March 2005; accepted 30 March 2005

Abstract

Colour space conversion is very important in many types of image processing applications including video compression. This operation consumes up to 40% of the entire processing power of a highly optimised decoder. Therefore, techniques which efficiently implement this conversion are desired. This paper presents two novel architectures for efficient implementation of a Colour Space Converter (CSC) suitable for Field Programmable Gate Array (FPGAs) and VLSI. The proposed architectures are based on Distributed Arithmetic (DA) ROM accumulator principles. The architectures have been implemented and verified using the Celoxica RC1000 FPGA development board. In addition, they are platform independent and have a low latency (eight cycles). The first architecture has a throughput of height, while the second one is fully pipelined and has a throughput of one and capable of sustained data rate of over 234 mega-conversions/s.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Colour space conversion; Field programmable gate array; Distributed arithmetic

1. Introduction

Colour is a visual sensation produced by the light in the visible region of the spectrum incident on the retina. Since the human visual system has three types of colour photoreceptor cone cells, three components are necessary and sufficient to describe a color [1].

Colour spaces (also called colour models or colour systems) is a method by which we can specify, create and visualize colour. There are many existing colour spaces and most of them represent each colour as a point in a three-dimensional coordinate system. Each colour space is optimised for a well-defined application area [3]. The three most popular colour models are RGB (used in colour printing). All of the colour spaces can be derived from the RGB information supplied by devices such as cameras and scanners.

Processing an image in the RGB colour space, with a set of RGB values for each pixel is not the most efficient method. To speed up some processing steps many broadcast, video and imaging standards use luminance and colour difference video signals, such as YCrCb, making a mechanism for converting between formats necessary. Several cores for RGB to YCrCb conversion can be found in the market, which have been designed for FPGA implementation, such as the cross proposed by Amphion Ltd [4], CAST.Inc [5] and ALMA.Tech [6]. Another such converter is presented in [7]. It has been implemented on a hybrid system which consists of the TriMedia processor augmented with FPGA-based reconfigurable functional units.

As part of an on going research project to develop a hardware accelerator for image and signal processing algorithms based on matrix computations at Queens University of Belfast [8–11], This paper proposes the use of FPGA as a low cost accelerator for two RGB to YCrCb Colour Space Convertion based architectures using DA principles, which is a bit level rearrangement of a multiply accumulate to hide the multiplications. The two proposed

* Corresponding author.
E-mail addresses: f.bensaali@qub.ac.uk (F. Bensaali), a.amira@qub.ac.uk (A. Amira).

0262-8856/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2005.03.006

architectures are based on serial and parallel manipulation of pixels.

The target hardware for the implementation and verification of the proposed architectures is Celoxica RC1000 PCI based FPGA development board equipped with a Xilinx XCV2000E Virtex FPGA [12,13]. The composition of the rest of the paper is as follows. A review for the conversion from R′G′B′ to Y′CrCb is given in Section 2. Sections 3 and 4 are concerned with the mathematical backgrounds and the descriptions of the two proposed architectures. Then the hardware implementations and the results and analysis are presented in Sections 5 and 6, respectively. Finally concluding remarks are given in Section 6.

2. Color space conversion: a review

As mentioned in the introduction, many color models have been proposed, each oriented towards supporting a specific task or solving a particular problem. Described below are the two colour systems selected for our study which are used in many image processing applications.

2.1. RGB colour space

RGB colour space is a simple and robust colour definition. RGB uses three numerical components to represent a colour. This colour space can be thought of as a three-dimensional coordinate system whose axes correspond to the three components, R or Red, G or Green, and B or Blue. RGB is the colour space that computer displays use. It corresponds most closely to the behavior of the human eye [1]. RGB is an additive colour system. The three primary colours red, green, and blue are added to form the desired colour. For a true colour image, the red, green, and blue components of a pixel are each with eight bits width. In total, it may have 16 million (2^{24}) possible colours. Each component has a range of 0–255, with all three 0s producing black and all three 255s producing white [1]. In the rest of this paper, the gamma-corrected RGB values are noted R′G′B′.

2.2. Y′CrCb colour space

Y′CrCb is a scaled and offset version of the YUV colour space based on luminance and chrominance, which correspond to brightness and colour. In this color space R′G′B′ is separated into a luminance part (Y′) and two chrominance parts (Cb and Cr). Y′ is defined to have a range of 16–235, Cb and Cr have a range of 16–240 [1,2].

2.3. Converting from R′G′B′ to Y′CrCb

Decomposing an R′G′B′ colour image into one luminance image and two chrominance images is the method that has been used in most commercial applications such has face detection [14,15], as well as the JPEG and MPEG imaging standards [16–18].

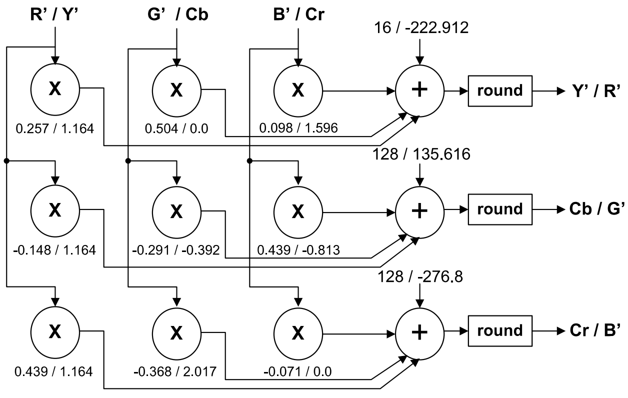


Fig. 1. General block diagram for R′G′B′↔Y′CrCb CSC.

The calculation of Y′CrCb colour components from R′G′B′ components consumes up to 40% of the processing power in a highly optimized decoder [16]. Accelerating this operation would be useful for the acceleration of the whole process. A colour in the R′G′B′ colour space is converted to the Y′CrCb colour space using the following equation

$$\begin{pmatrix} Y' \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0.257 & 0.504 & 0.098 & 16 \\ 0.439 & -0.368 & -0.071 & 128 \\ -0.148 & -0.291 & 0.439 & 128 \end{pmatrix} \begin{pmatrix} R' \\ G' \\ B' \\ 1 \end{pmatrix} \tag{1}$$

while the inverse conversion can be carried out using the following equation:

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} 1.164 & 1.596 & 0.0 & -222.912 \\ 1.164 & -0.813 & -0.392 & 135.616 \\ 1.164 & 0.0 & 2.017 & -276.8 \end{pmatrix} \begin{pmatrix} Y' \\ Cr \\ Cb \\ 1 \end{pmatrix} \tag{2}$$

Fig. 1 shows the direct mapping of Eqs. (1) and (2).

3. Proposed architecture based serial manipulation approach

3.1. Mathematical background

Since colour space conversion can be expressed as a Matrix-Vector (MV) multiplication, a novel algorithm based DA is presented in this section.

DA distributes arithmetic operations rather than grouping them as multipliers do. Conventional DA, called ROM-based DA, decomposes the variable input of the inner product to bit level in order to generate precomputed data.

ROM-based DA uses a ROM table to store the precomputed data, which makes it regular and efficient in the use of the silicon area, in a VLSI implementation. The advantage of a DA-based ROM approach is its efficiency of implementation. The basic operations required are a sequence of ROMs, addition, subtraction and shift operations of the input data sequence [19]. Examples for the use of DA can be found in these Refs. [19–21].

Consider the matrix-vector product given by the following equation

$$C_i = \sum_{k=0}^{N-1} A_{ik} B_k \tag{3}$$

where $\{A_{ik}\}$'s are L-bits constants and $\{B_k\}$'s are written in the unsigned binary representation as shown in Eq. (4)

$$B_k = \sum_{m=0}^{W-1} b_{k,m} 2^m \tag{4}$$

where $b_{k,l}$ is the mth bit of B_k, which is zero or one, W is the word-length used which represents the resolution for each colour component of a pixel.

Substituting (4) in (3)

$$C_i \sum_{k=0}^{N-1} A_{ik} \left(\sum_{m=0}^{W-1} b_{k,m} 2^m \right) = \sum_{m=0}^{W-1} \left(\sum_{k=0}^{N-1} A_{ik} (b_{k,m} 2^m) \right) \tag{5}$$

define

$$Z_m = \sum_{k=0}^{N-1} A_{ik} b_{k,m} \tag{6}$$

therefore, C_i can be computed as:

$$C_i = \sum_{m=0}^{W-1} Z_m 2^m \tag{7}$$

The idea is that since the term Z_m depends on the $b_{k,m}$ values and has only 2^N possible values, it is possible to precompute and store them in ROMs. An input set of N bits $(b_{0,m}, b_{1,m}, \ldots b_{(N-1),m})$ is used as an address to retrieve the corresponding Z_m values. The ROM's content is different and depends on the constant matrix A coefficients. These intermediate results are accumulated in W clock cycles to produce C_i coefficients.

3.2. Case study: converting from $R'G'B' \leftrightarrow Y'CrCb$

The CSC core implements the following mathematical formula to convert from one space to another

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ 1 \end{pmatrix} \tag{8}$$

where C_i $(0 \le i \le 2)$ and B_i $(0 \le i \le 3)$ represent the input and output colour components, respectively. Since all the components are in the range of 0–255, eight bits are enough to represent them. In our application ($N=4$ and $W=8$), C_i can be computed as

$$C_i = \sum_{m=0}^{7} Z_m 2^m \tag{9}$$

where

$$Z_m = \sum_{k=0}^{3} A_{ik} b_{k,m} \tag{10}$$

$$C_i = \begin{array}{cccccc} k=0 & k=1 & k=2 & k=3 & & \\ (A_{i0} \times b_{0,0} + & A_{i1} \times b_{1,0} + & A_{i2} \times b_{2,0} + & A_{i3} \times b_{3,0}) & \times 2^0 + & m=0 \\ (A_{i0} \times b_{0,1} + & A_{i1} \times b_{1,1} + & A_{i2} b_{2,1} + & A_{i3} \times b_{3,1}) & \times 2^1 + & m=1 \\ (A_{i0} \times b_{0,2} + & A_{i1} \times b_{1,2} + & A_{i2} \times b_{2,2} + & A_{i3} \times b_{3,2}) & \times 2^2 + & m=2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (A_{i0} \times b_{0,7} + & A_{i1} \times b_{1,7} + & A_{i2} \times b_{2,7} + & A_{i3} \times b_{3,7}) & \times 2^7 + & m=7 \end{array} \tag{11}$$

Three ROMs (one for each matrix A row) with the size of $2^N = 2^4 = 16$ are needed in order to store the precompute 2^4 possible partial products values. Since the last element of the vector B is equal to 1:

$$b_{3,m} = \begin{cases} 1 & \text{for } m = 0 \\ 0 & \text{for } m \neq 0 \end{cases} \tag{12}$$

Eq. (9) can be rewritten as

$$C_i = \sum_{m=0}^{7} Z_l^* 2^m + A_{i3} \tag{13}$$

where

$$Z_m^* = \sum_{k=0}^{2} A_{ik} b_{k,m} \tag{14}$$

$$\begin{array}{rccccc} & k=0 & k=1 & k=2 & & \\ & (A_{i0} \times b_{0,0} + & A_{i1} \times b_{1,0} + & A_{i2} \times b_{2,0}) & \times 2^0 + & m=0 \\ & (A_{i0} \times b_{0,1} + & A_{i1} \times b_{1,1} + & A_{i2} \times b_{2,1}) & \times 2^1 + & m=1 \\ C_i = & (A_{i0} \times b_{0,2} + & A_{i1} \times b_{1,2} + & A_{i2} \times b_{2,2}) & \times 2^2 + & m=2 \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ & A_{i,0} \times b_{0,7} & A_{i1} \times b_{1,7} & A_{i2} \times b_{2,7} & \times 2^7 + & m=7 \\ & A_{i3} & & & & \end{array} \tag{15}$$

It is worth mentioning that the size of the ROMs has been reduced to 2^3. Table 1 gives the content of each ROM.

3.3. Proposed architecture

Since our objective is to implement a core which performs two different colour conversions ($R'G'B' \leftrightarrow Y'CrCb$), six ROMS are needed (three for each

Table 1
Content of the ROM i ($0 \le i \le 2$)

$b_{0,m}$	$b_{1,m}$	$b_{2,m}$	The content of the ROM i
0	0	0	0
0	0	1	A_{i2}
0	1	0	A_{i1}
0	1	1	$A_{i1}+A_{i2}$
1	0	0	A_{i0}
1	0	1	$A_{i0}+A_{i2}$
1	1	0	$A_{i0}+A_{i1}$
1	1	1	$A_{i0}+A_{i1}+A_{i2}$

conversion). Figs. 2 and 3 show the proposed core pins and its internal architecture, respectively.

The pins description is given in Table 2.

The proposed architecture consists of three identical Processing Elements (PEs) and two memory blocks. Each PE comprises a parallel ACCumulator (ACC) and a right shifter and each memory block consists of three ROMs with the size of 2^3 each (see Fig. 4). The ROM's content is different and depends on the matrix A coefficients, which depend on the conversion type.

It is worth mentioning that our architecture is scalable, however , it can be used to perform n conversions by adding every time $3\times n$ ROMs in order to store the matrix conversion coefficients and keeping always the same PEs. An $N\times M$ image can be converted using the proposed architecture by setting the inputs every eight clock cycles

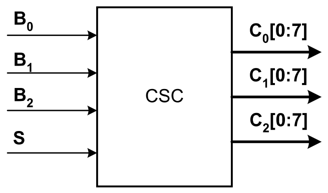


Fig. 2. Symbol of the CSC core.

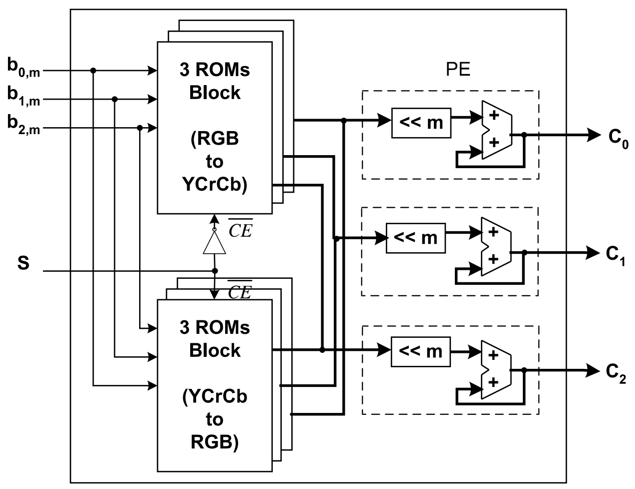


Fig. 3. Serial CSC based DA architecture.

Table 2
Pins description

Name	Dir	Description
B_0	I	First input colour space component
B_1	I	Second input colour space component
B_2	I	Third input colour space component
C_0	O	First output colour space component
C_1	O	Second output colour space component
C_2	O	Third output colour space component
S	I	Colour space conversion type selection

using the R′G′B′; components of a new pixel (Y′CrCb for the inverse conversion) Fig. 5.

4. Proposed architecture based parallel manipulation approach

4.1. Mathematical background

Consider an $N\times M$ image (N: image height, M: image width).

Let represent each image pixel by b_{ijk}, ($0\le i\le N-1$, $0\le j\le M-1, 0\le k\le 2$), where

$$\begin{cases} b_{ij0} = R_{ij}/ \text{ the red component of the pixel in row } i \text{ and column } j \\ b_{ij1} = G_{ij}/ \text{ the green component of the pixel in row } i \text{ and column } j \\ b_{ij2} = B_{ij}/ \text{ the blue component of the pixel in row } i \text{ and column } j \end{cases} \tag{16}$$

The image can be converted using the following mathematical formula

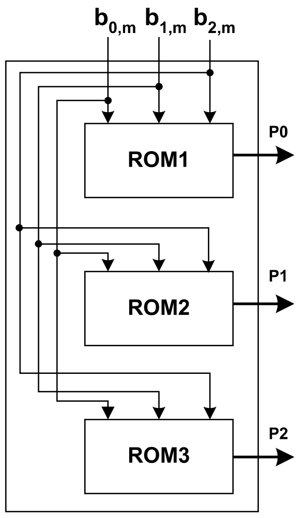


Fig. 4. Memory block structure.

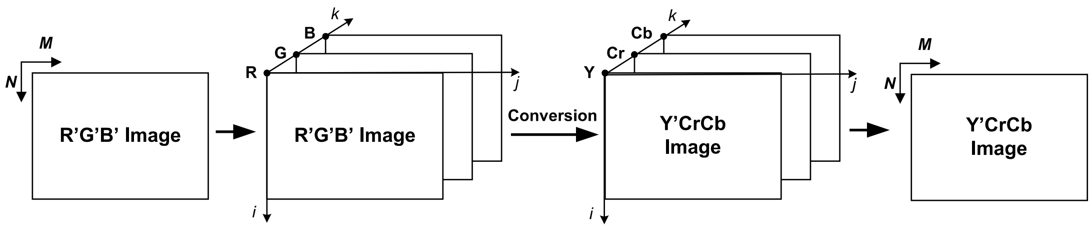


Fig. 5. R′G′B′ to Y′CrCb conversion.

$$\begin{pmatrix} \begin{pmatrix} c_{000} \\ c_{001} \\ c_{002} \end{pmatrix} & \cdots & \begin{pmatrix} c_{0(M-1)0} \\ c_{0(M-1)1} \\ c_{0(M-1)2} \end{pmatrix} \\ \begin{pmatrix} c_{100} \\ c_{101} \\ c_{102} \end{pmatrix} & \cdots & \begin{pmatrix} c_{1(M-1)0} \\ c_{1(M-1)1} \\ c_{1(M-1)2} \end{pmatrix} \\ \vdots & \cdots & \vdots \\ \begin{pmatrix} c_{(N-1)00} \\ c_{(N-1)01} \\ c_{(N-1)02} \end{pmatrix} & \cdots & \begin{pmatrix} c_{(N-1)(M-1)0} \\ c_{(N-1)(M-1)1} \\ c_{(N-1)(M-1)2} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \end{pmatrix} \otimes \begin{pmatrix} \begin{pmatrix} b_{000} \\ b_{001} \\ b_{002} \\ 1 \end{pmatrix} & \cdots & \begin{pmatrix} b_{0(M-1)0} \\ b_{0(M-1)1} \\ b_{0(M-1)2} \\ 1 \end{pmatrix} \\ \begin{pmatrix} b_{100} \\ b_{101} \\ b_{102} \\ 1 \end{pmatrix} & \cdots & \begin{pmatrix} b_{1(M-1)0} \\ b_{1(M-1)1} \\ b_{1(M-1)2} \\ 1 \end{pmatrix} \\ \vdots & \cdots & \vdots \\ \begin{pmatrix} b_{(N-1)00} \\ b_{(N-1)01} \\ b_{(N-1)02} \\ 1 \end{pmatrix} & \cdots & \begin{pmatrix} b_{(N-1)(M-1)0} \\ b_{(N-1)(M-1)1} \\ b_{(N-1)(M-1)2} \\ 1 \end{pmatrix} \end{pmatrix} \tag{17}$$

where the operation $\otimes$ can be defined as follows: each vector

$$\begin{pmatrix} c_{ij0} \\ c_{ij1} \\ c_{ij2} \end{pmatrix}$$

is the result of the product

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \end{pmatrix} \begin{pmatrix} b_{ij0} \\ b_{ij1} \\ b_{ij2} \\ 1 \end{pmatrix},$$

where c_{ijk} represent the output image colour space components and

$$A = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & a_{22} & A_{23} \end{pmatrix}$$

represents one of the constant matrices in Eqs. (1) and (2).

The c_{ijk} elements (the output image colour space components) can be computed using the following equation

$$c_{ijk} = \sum_{m=0}^{3} A_{km} b_{ijm} \quad (0 \le i \le N-1,\ 0 \le j \le M-1,\ 0 \le k \le 2) \tag{18}$$

where $\{A_{km}\}$'s are L-bits constants and $\{b_{ijm}\}$'s are written in the unsigned binary representation as shown in Eq. (17):

$$b_{ijm} = \sum_{l=0}^{W-1} b_{ijm,l} 2^l \quad (0 \le i \le N-1,\ 0 \le j \le M-1,\ 0 \le m \le 2) \tag{19}$$

Using the same development in Section 4.1, Eq. (16) can be rewritten as

$$c_{ijk} = \sum_{l=0}^{7} Z_l^* 2^l + A_{k3} \tag{20}$$

where

$$Z_l^* = \sum_{m=0}^{2} A_{km} b_{ijm,l} \tag{21}$$

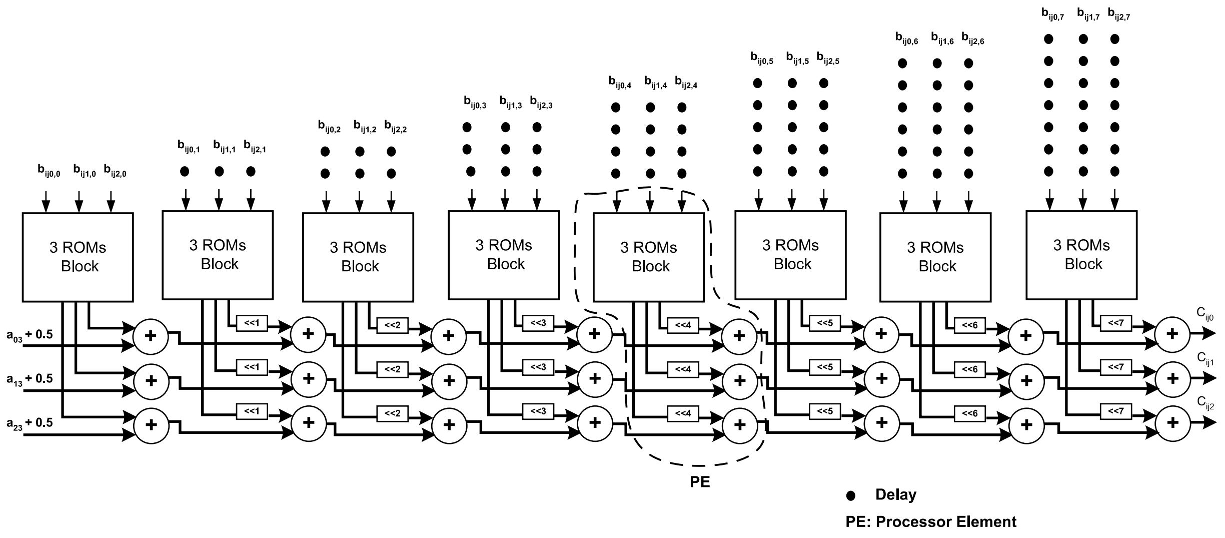


Fig. 6. Proposed parallel architecture based on DA principles clock cycles, where (3×4) is the constant matrix A size.

$$\begin{array}{lllllll} & m=0 & m=1 & m=1 & & & \\ & (A_{k0}\times b_{ij0,0}+ & A_{k1}\times b_{ij1,0}+ & A_{k2}\times b_{ij2,0}) & \times 2^{0}+ & l=0 & \\ & (A_{k0}\times b_{ij0,1}+ & A_{k1}\times b_{ij1,1}+ & A_{k2}\times b_{ij2,1}) & \times 2^{1}+ & l=1 & \\ c_{ijk}= & (A_{k0}\times b_{ij0,2}+ & A_{k1}\times b_{ij1,2}+ & A_{k2}\times b_{ij2,2}) & \times 2^{2}+ & l=2 & \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \\ & (A_{k0}\times b_{ij0,7}+ & A_{k1}\times b_{ij1,7}+ & A_{k2}\times b_{ij2,7}+ & \times 2^{7}+ & l=7 & (22)\\ & A_{k3} & & & & & \\ & PP_{ijk0}+ & PP_{ijk1}+ & PP_{ijk2}+ & PP_{ijk3}+ & & \\ = & PP_{ijk4}+ & PP_{ijk5}+ & PP_{ijk6}+ & PP_{ijk7}+ & & \\ & A_{k3} & & & & & \end{array}$$

Likewise the first proposed architecture, the ROM's content is different and depends on the matrix A coefficients, which depend on the conversion type.

4.2. Proposed architecture

Eq. (18) can be mapped into the proposed architecture as shown in Fig. 6.

The architecture consists of eight identical PE$_n$s ($0 \leq n \leq 7$). Each PE$_n$ comprises three parallel signed integer adders, three n right shifters and one ROMs block, which have the structure as shown in Fig. 4. It is worth noting that the architecture has a latency of W and a throughput rate equal to 1. The entire image conversion can be carried out in (Latency + (NM)Throughput) = 8 + (NM) clock cycles, while using the standard algorithm (Fig. 7), the conversion can be carried out in ($3\times4\times N\times M$) clock cycles, where (3×4) is the constant matrix A size.

Fig. 8 shows the functional analysis diagram of the proposed architecture.

5. Hardware implementation

The two proposed architectures based on DA technique have been implemented and verified using the Celoxica RC1000-PP FPGA development board. The RC1000-PP board used is a standard PCI bus card equipped with theVirtex-E2000 FPGA chip (package: bg560, speed grade 6). Tables 3 and 4 give the content of the ROMs used for R′G′B′ to Y′CrCb and Y′CrCb to R′G′B′ conversions for both architectures, respectively.

The second proposed architecture can be used for the inverse conversion (Y′CrCb to R′G′B′) by:

- Duplicating the ROMS using the same implementation approach used for the first architecture (with a selector signal which allows the user to choose the appropriate converter); or
- Setting the contents of the ROMs in advance, depending on the desired conversion.

```
for i ←— 1 to L do                 // scanning image rows
  for j ←— 1 to M do               // scanning image columns
     for k ←— 1 to 3 do            // scanning the three RGB valus of a pixel
        for k ←— 1 to 3 do         // scanning columns of the constant conversion matrix

                  $c_{ijk}$ += $a_{km}$ x $b_{ijm}$

        end for
     end for
  end for
end for
```

Fig. 7. Pseudo code for the standard algorithm.

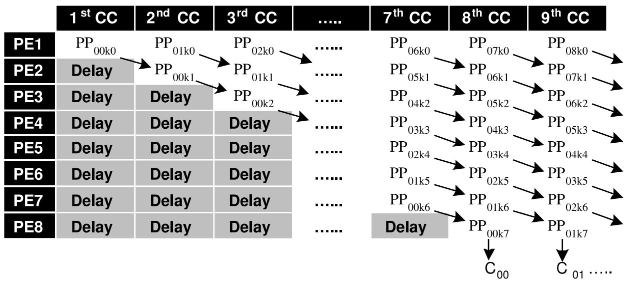


Fig. 8. Functional analysis diagram.

The precomputed partial products are stored in the ROMs using 13 bits fixed point representation (eight bits for integer part and five bits for fractional part). 13-bit arithmetic is used inside the architecture. The input sand output soft hetwo architectures are presented using eight bits and the outputs are rounded. Rounding usually looks at the decimal value and if it is greater than or equal to 0.5, then the result is increased by one. This implies a condition of verifying followed by another arithmetic operation. A more efficient way to round a number is to add 0.5 to the result and truncate the decimal value. This technique has been applied in our proposed architecture. The initial value for each PE's ACC (for the serial architecture) and for the first PEs adder (for the parallel architecture) is set in advance to (A_{i3}+0.5), where ($0\leq i\leq 2$). The MACs and parallel signed adders have been implemented using Xilinxs CoreGen utility, which contains many efficient designs that can often save time for a programmer [22]. The shifters and ROMs initialisation have been implemented using VHDL. All design components have been connected together using Handel-C, a C-like language supporting parallelism, flexible data size and compilation of high-level programs directly into FPGA hardware [23].

6. Result and analysis

In order to make a fair and consistent comparison with the existing FPGA based CSCs, the XCV50E-8 FPGA device has been targeted. Table 5 illustrates the performances obtained for the proposed architecture in terms of area consumed and speed which can be achieved.

The proposed architecture based serial manipulation approach shows significant improvements in comparison with the existing implementatins [4–6], which perform the R′G′B′ to Y′CrCb conversion, in terms of the area consumed and the maximum running clock frequency, while the second architecture outperforms the existing ones in term of the number of conversions per second.

It is worth nothing that the existing cores used the same number of bit representation for the input and output data used in our design.

In addition a software based approach to perform the same operation has been implemented on a 2.0 GHz Intel®Pentium 4 processor with 1 GB of SDR RAM using Borland C++ Builder V6.0.

Table 6 illustrates the hardware/software implementations comparison in terms of the RMS error-due to the use of difference data representation in the two implementations,

$$\left(\mathrm{RMS}_{\mathrm{Error}} = \sqrt{1/(NM)\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(I_{\mathrm{soft}}(i,j) - I_{\mathrm{hard}}(i,j))^2}\right)$$

and the computation time, when using the second proposed DA architecture. Table 6 shows the test results for two different images (Baboon image (512×512) and Pepper image (256×256)). It can be seen that the same converted image can be obtained fastly when using the FPGA implementation, with a minimum error (due to the use of difference data representation in the two implementations).

7. Conclusion

Processing an image in the RGB color space, with a set of RGB values for each pixel is not the most efficient method. To speed up some processing steps many broadcast, video and imaging standards use luminance and color difference

Table 3
The content of the ROMs (R′G′B′ to Y′CrCb)

$R'_m/R'_{ij0,l}$	$G'_m/G'_{ij1,l}$	$B'_m/B'_{ij2,l}$	ROM1	ROM2	ROM3
0	0	0	0	0	0
0	0	1	0.098	−0.071	0.439
0	1	0	0.504	−0.368	−0.291
0	1	1	0.602	−0.439	0.148
1	0	0	0.257	0.439	−0.148
1	0	1	0.355	0.368	0.291
1	1	0	0.761	0.071	−0.439
1	1	1	0.859	0	0

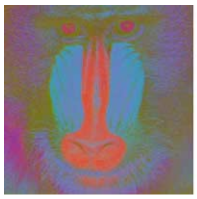
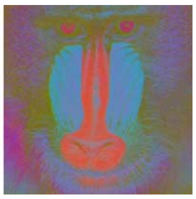
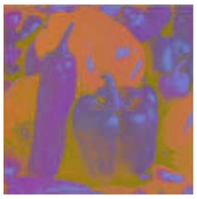
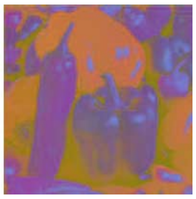
Table 4
The content of the ROMs (Y′CrCb to R′G′B′)

$Y'_m/Y'_{ij0,l}$	$Cr_m/Cr_{ij1,l}$	$Cb_m/Cb_{ij2,l}$	ROM1	ROM2	ROM3
0	0	0	0	0	0
0	0	1	0	−0.392	0
0	1	0	1.596	−0.813	1.596
0	1	1	1.596	−1.025	1.596
1	0	0	1.164	1.164	1.164
1	0	1	1.164	0.772	1.164
1	1	0	2.76	0.351	2.76
1	1	1	2.76	−0.041	2.76

Table 5
Performance comparison with existing CSC cores

Design parameters	Slices	Speed (MHz)	Throughput (vector/clock cycle)
Proposed architecture (1)	70	128	8
Proposed architecture (2)	193	234	1
CAST.Inc [5]	222	112	1
ALMA.Tech [6]	222	105	1
Amphion Ltd [4]	204	90	1

Table 6
Software/hardware implementations for RGB to YCrCb CSC comparisons

Original image	Software implementation	Hardware implementation	RMS error		Computation time (ms)	
					Software	Hardware
			Y Cr Cb	0.487 0.630 0.461	126	1.2
			Y Cr Cb	0.684 0.830 0.396	43	0.28

video signals, such as YCrCb, making a mechanism for converting between formats necessary. $R'G'B' \leftrightarrow Y'CrCb$ conversions require enormous computing power. However, novel, scalable and efficient architectures based on DA principles have been reported in this paper. The implementation result shows the effectiveness of the DA approach. The performance in terms of the area used and the maximum running frequency of the proposed architectures has been assessed and has shown that the proposed systems requires less area and can be run with a higher frequency when compared with existing systems.

References

[1] B. Payette, Color space converter: $R'G'B'$ to $Y'CrCb$, Xilinx Aplication Note, XAPP637, V1.0, September, 2002.

[2] C. Poynton, A Technical Introduction to Digital Video, Wiley, New York, 1996.

[3] R.C. Gonzalez, R.E. Woods, Digital Image Processing, second ed., Prentice-Hall, Englewood Cliffs, NJ, 2002.

[4] Data sheet (www.amphion.com), Color Space Converters, Amphion semiconductor Ltd, DS6400 V1.1, April, 2002.

[5] Application Note (www.cast-inc.com), CSC Color Space Converter, CAST Inc., April, 2002.

[6] Datasheet (www.alma-tech.com), High Performance Color Space Converter, ALMA Technologies, May 2002.

[7] M. Sima, S. Vassiliadis, S.D. Cotofana, J.T.J. van Eijndhoven, Color space conversion for MPEG decoding on FPGA-augmented TriMedia processor The 14th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP'03), Netherlands, June (2003) pp. 250–259.

[8] F. Bensaali, A. Amira, Design and efficient FPGA implementation of an RGB to YCrCb color space converter using distributed arithmetic, Proceedings of the International Conference on Field Programmable Logic (FPL), Lecture Notes in Computer Science, to be published by Springer, August, 2004.

[9] A. Amira, A Custom Coprocessor for Matrix Algorithm, PhD thesis, Queen's University of Belfast, 2001.

[10] F. Bensaali, A. Amira, I.S. Uzun, A. Ahmedsaid, An FPGA implementation of 3D affine transformations The 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS'03), Sharjah, UAE, December (2003).

[11] F. Bensaali, A. Amira, I.S. Uzun, A. Ahmedsaid, Efficient implementation of large parallel matrix product for DOTs The International Conference on Computer, Communication and Control Technologies (CCCT'03), Florida, USA, July (2003).

[12] Datasheet, (www.celoxica.com), RC1000 Reconfigurable hardware development platform, Celocixa Ltd, 2001.

[13] URL:www.xilinx.com.

[14] A. Albiol, L. Torres, E.J. Delp, An unsupervised color image segmentation algorithm for face detection applications Proceedings of the International Conference on Image Processing, October vol. 2 (2001) pp. 681–684.

[15] P. Kuchi, P. Gabbur, P.S. Bhat, S. David, Human face detection and tracking using skin color modelling and connected component operators, The IETE Journal of Research, Special issue on Visual Media Processing May (2002).

[16] M. Bartkowiak, Optimisations of color transformation for real time video decoding Digital Signal Processing for Multimedia Communications and Services, EURASIP ECMCS 2001, Budapest, September (2001).

[17] J.L. Mitchell, W.B. Pennebaker, MPEG Video Compression Standard, Chapman and Hall, London, 1996.

[18] J. Bracamonte, P. Standelmann, M. Ansorge, F. Pellandini, A multiplierless implementation scheme for the JPEG image coding algorithm IEEE Nordic Signal Processing Symposium, Kolmarden, Sweden, June 13–15 (2000).

[19] A. Amira, An FPGA based parameterisable system for discrete Hartley transforms implementation Proceedings of the International Conference on Image Processing (ICIP), Barcelona, Spain, September (2003).

[20] H. Ohlsson, L. Wanhammer, Maximally fast numerically equivalent state-space recursive digital filters using distributed arithmetic Proceedings of the IEEE Symposium in Nordic Signal Processing (NORSIG2000), Kolmarden, Sweden, June (2000) pp. 295–298.

[21] O. Gustafsson, L. Wanhammar, Implementation of a digital beamformer in an FPGA using distributed arithmetic Proceedings of the IEEE Symposium in Nordic Signal Processing (NORSIG2000) Kolmarden, Sweden, June (2000) pp. 295–298.

[22] Application Note, Xilinx CoreGen, Handel-C, AN 58 v1.0, 2001.

[23] Manual, (www.celoxica.com) Handel-C Language Reference Manual, Celocixa Ltd, 2003.