Implementation of Multiplier less Architectures for Color Space Conversions on FPGA

Dr. Ahlam Fadhil Mahmood **Abdulkreem Mohammad Salih**

Computer Engineering Department
University of Mosul
ahlam.mahmood@gmail.com abdlat_1986@yahoo.com

Abstract

The divergence of computers, internet, and wide variety of interactive video devices, in most of the multimedia applications, all using different color representations, is forcing the digital designer today to convert between them. The objective is to have a converter, which will be useful for number of applications with a basic function of converting from one color space to another and the inverse on same architecture. This paper presents an efficient parallel multiplierless implementation for two color space converters (*RGB* to *YCbCr* and *YCbCr* to *RGB*). The proposed architecture is based on distributed arithmetic (*DA*) principles which has been implemented on the Xilinx *Spartan-3E XC3S500 FPGA* using fewer resources. The implementation approach exhibits better performances when compared with existing implementations, Modifications have been carried out in *DA* to reduce the hardware complexity with better performance in area, latency and throughput.
Keywords: Color Space Conversion; Distributed Arithmetic ; FPGA; Video, Processing; Image Processing .

تنفيذ معمارية بلا مضارب لتحويلات فضاء اللون على رقاقة البوابات القابلة للبرمجة حقليا

د.أحلام فاضل محمود
قسم هندسة حاسبات

عبد الكريم محمد صالح
قسم هندسة حاسبات

الملخص

أن انتشار الحاسبات والإنترنت والأنواع المختلفة مِنْ أدواتِ الفيديو التفاعليةِ، في أغلب التطبيقاتِ المتعددة الأوساطِ،وكُلّ منها يستخدم تمثيل مختلف للألوان يُجبرُ المصممَ الرقميَ على تنفيذ معمارية للتَحويل بينهما. لذلك كان هدف البحث تصميم محول للالوان، يَكُونُ مفيد لعددِ التطبيقاتِ وظيفته ألأساسية التَحويل من فضاء لون إلى آخر وبالعكس في نفس المعمارية. هذه الورقةِ تقدم معمارية متوازية بدون مضارب لتحوّلين لفضاءِ اللون (*RGB* to *YCbCr* and *YCbCr* to *RGB*). المعمارية المُقتَرَحة مستندة على مبدأ الحساب المُوَزَّع حيث طُبّقتْ على رقاقة البوابات القابلة للبرمجة حقليا *Spartan-3E XC3S500* بمكونات أقل. التطبيقُ المقترح أعطى نتائج أفضل مقارنة بالتطبيقاتِ الأخرى, التعديلات نفذت في الحساب المُوَزَّع لتخفيض تعقيد المكونات بالأداء الأفضل في المساحة وتقليل معدل الحصول على ألاخراج وكمية المعلومات التي يتم معالجتها.

Received: 2 – 8 - 2010 Accepted: 28– 11 - 2010

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. Each color space is optimized for a well-defined application area[1].

Different color spaces have historically evolved for different applications [2]. In each case a color space is chosen for application specific reasons and a certain choice is the best because it requires less storage, bandwidth or computation in analog or digital domains [3]. The three most popular color models are *RGB* color space (used in computer graphics); *YIQ*, *YUV* and *YCbCr* color space (used in image and video systems) and *CMYK* color space (used in color printing) [2,4]. All color spaces can be derived from the *RGB* information supplied by the devices such as cameras and scanners.

Any color model or color space is usually specified using three coordinates or parameters. These parameters describe the position of color within the color space being used [1]. Large number of high quality color images used in many multimedia applications, requires high capacity mass storage devices. *JPEG*, whose encoding starts with *RGB* to *YCbCr* conversion has become the most popular image compression technique and is the best example where such conversion is used [5,6]. Moreover this conversion is also needed in many video designs, digital coding of TV pictures, *HIDTV* and video digital libraries [7]. On the other hand, object tracking requires image segmentation to make a distinction between the target objects and the rest of the scene in a captured image. The tracking algorithm also starts with a color conversion stage [8].

Reconfigurable hardware devices in the form of Field Programmable Gate Arrays (*FPGAs*) have been proposed as viable system building blocks in the construction of high performance systems at an economical price. However, power budgets are becoming increasingly stringent and need higher attention in the early stages of the design cycle.

Recently, number of existing architectures for Color Space Conversion (*CSC*) and their hardware implementations proposed. In [3], three ways to implement the *RGB* to *YCbCr CSC* were described. The work presented in [4] for concerned with the implementation of *CSC* operation on using RISC Processor based on bit-plan algorithm. In [9], an *FPGA* based functional unit and associated instruction approach were presented for the implementation of *CSC* operation on the Xilinx *FPGA*. F. Bensaali etal Suggest a power modeling of color space *FPGA* converter [10]. In addition, the results achieved in terms of maximizing their *FPGA* resources were demonstrated on a commercial *CSC* engine design using two Xilinx Virtex-II Pro *FPGAs* and presented in [11]. Y.Yang in [12] ,suggest a new fast software algorithm for *YCbCr* to *RGB* conversion based on shift and addition operations to take place of the float-point multiplication operation. The aim of this paper is to develop power efficient architecture based on Distributed Arithmetic (*DA*), ideally suited for multiplierless implementations of an *RGB* to *YCbCr* and *YCbCr* to *RGB CSC* core on *Spartan-3E FPGAs*. The features of the conversion matrices have been exploited to develop the mathematical model in order to reduce the ROM size, the area consumed by the design, and to speed up the computation procedure by minimizing the number of the required shift operations.

The rest of the paper is as follows: Section 2 gives a brief overview of color spaces transformations and the necessity of conversion. Section 3 & 4 are concerned with the mathematical backgrounds and the descriptions of the proposed architectures based *DA* techniques respectively. In section 5, the *FPGA* based architecture is proposed, and the Xillinx Color Space Core Generator explained in section 6. The obtained results with *VHDL* and Matlab implementations are compared in section 7. Finally in section 8 the conclusions and future modifications are indicated.

2. Color Space Transformation

Different color spaces have historically evolved for different applications. A certain choice is better because it required less storage, bandwidth or computation in analog or digital domains. [3] The objective is to have all inputs be converted to a common color space before algorithms and processes are executed. Converters are useful for number of applications including image processing and filtering. The converter basic function is to convert from one color space to another. This paper describes one such conversion.

2.1 RGB Color Space

The Red , Green, and Blue color space is widely used in computer graphics. Red, Green, and Blue are the three primary colors and are represented by three dimensional Cartesian coordinate system [2]. It is an additive color space where each component has a range of 0 to 255, with all three 0s for producing a black color and all three 255 for producing a white color [9]. Though being the simplest and robust color space , *RGB* has few disadvantages. It has high correlation between it's components (R, G, and B) . It is psychologically non intuitive and another problem is the perceptual non uniformity. So *RGB* is not very efficient when dealing with real world images and thus processing an image in *RGB* color space is usually not the most efficient method [3].

2.2 YCbCr Color Space

YCbCr is a family of color spaces used in video systems. *Y* is the luma component and *Cb* and *Cr* are the blue and red chroma components [5]. *YCbCr* Color Space was developed as part of the Recommendation ITU-R BT.601 for worldwide digital component video standard and then used in television transmissions. Here the *RGB* color space is separated into a luminance part (*Y*) and two chrominance parts (*Cb* and *Cr*)[9].

If the color information is stored in the intensity and color format, some of the processing steps can be made faster. As a result, *Cb* and *Cr* provide the hue and saturation information of the color and *Y* provides the brightness information of the color. Because eye is less sensitive to *Cb* and *Cr*, engineers did not need to transmit *Cb* and *Cr* at the same rate as *Y*. Thus after such conversion, less storage and bandwidth is needed, resulting in a reduced-cost design .

2.3 Converting from RGB to YCbCr Color Space

Decomposing an *RGB* color image into one luminance image and two chrominance images is the method that has been used in the most commercial applications such as face detections, *JPEG* and *MPEG* imaging standards [5]. This basically is *YCbCr* color space. *Y* has the range of 16 to 235 and *Cb* and *Cr* have the range of 16 of 240 [6]. Decomposing *RGB* color Space to *YCbCr* (as shown in Figure 1) is suitable because of the non-correlation among the spaces of *YCbCr*, so each space can be analyzed separately. Moreover *Cr* and *Cb* spaces can be compressed more heavily than *Y* space to get better compression ratio [5]. The *JPEG* & *MPEG* encoders always start with *RGB* to *YCbCr* conversion and the last stage of decoder is the *YCbCr* to *RGB* conversion unit.

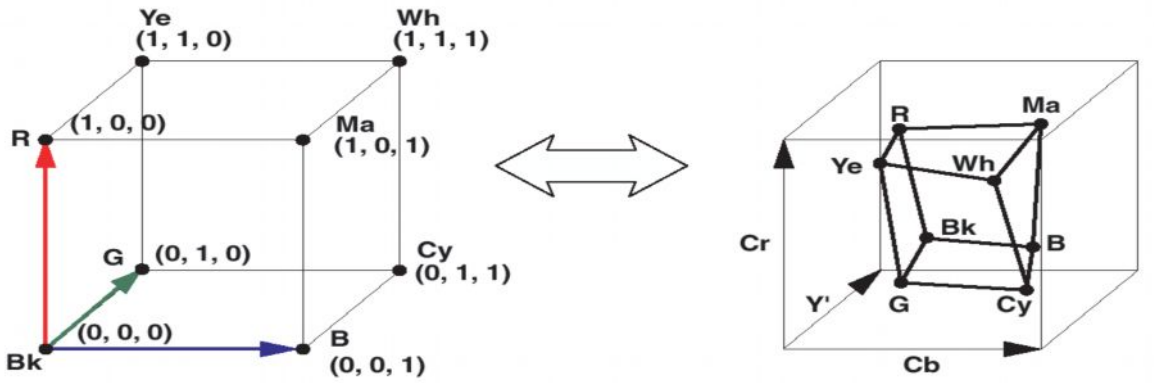

Figure 1: RGB and YCrCb Color Representations

3. CSC Using Distributed Arithmetic

A color in the *RGB* color space can be converted to the *YCbCr* color space using the following equation:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} \quad \ldots(1)$$

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

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.164 & 0 & 1.59 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0 \end{pmatrix} * \begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} + \begin{pmatrix} -222.9 \\ 135.616 \\ -276.8 \end{pmatrix} \quad \ldots..(2)$$

Direct *CSC* requires nine multiplications and nine additions per conversion. The transform coefficients may take floating or fixed-point representation as they are real signed numbers.

There are several conventional methods including direct method and lookup table method, The direct method is floating-point library routine followed by quantification according to matrices in (1) and (2). Obviously, the advantage is free of extra memory consumption while its disadvantage is the abundant floating-point multiplications. As *YCbCr* to *RGB* conversion often have to be done on fixed-point DSPs. For these embedded systems, the floating-point operations are converted to fixed-point shift and addition operations [12].

The Look-up Table (*LUT*) method is one of the high efficient methods especially for embedded systems. So other methods are often compared with LUT to examine their efficiency [3,4,10].

Distributed Arithmetic (*DA*) is a lookup table-based technique that is conductive to *VLSI* implementation and has been used to implement various signal processing algorithms

including *CSC* and *DCT* [5]. *CSC* can be implemented using *DA* approach as described in the following:

Consider the matrix-vector product between the *RGB* vector and a conversion matrix, given by [3,4]:

$$C_i = \sum_{k=0}^{N-1} A_{ik} B_k \quad \text{.....(3)}$$

where $\{A_{ik}\}$'s are L-bits constants, N is the number of coefficients $A_{i\,k}$ which equal to 3 in this case, and $\{Bk\}$'s are written in the unsigned binary representation as shown in the following equation :

$$B_k = \sum_{m=0}^{W-1} b_{k,m}\, 2^m \quad \text{.....(4)}$$

where $b_{k,m}$ is the m^{th} bit of B_k , which is zero or one, *W* is the word-length used which represents the resolution for each color component of a pixel. Substituting (4)in(3) yields,

$$C_i = \sum_{k=0}^{N-1} A_{ik} \sum_{m=0}^{W-1} b_{k,m} 2^m \quad \text{.....(5)}$$

Interchanging the two summations gives:

$$C_i = \sum_{m=0}^{W-1} \sum_{k=0}^{N-1} A_{ik} b_{k,m} 2^m \quad \text{.....(6)}$$

Let us define Z_m as:

$$Z_m = \sum_{k=0}^{N-1} A_{ik} b_{k,m} \quad \text{.....(7)}$$

Then, (6) becomes

$$C_i = \sum_{m=0}^{W-1} Z_{im} 2^m \quad \text{.....(8)}$$

i= 0,1,2

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 ($b_{0,m}$, $b_{1,m}$, . . . $b_{(N-1),m}$) is used as an address to retrieve the corresponding Z_m values. The ROM's content is different and depends on the number of shifts for coefficients of matrix A.

Since all the components are in the range of 0 to 255, 8 bits are enough to represent them. In the proposed application (*N* = 4 and *W* = 8), then (8) can be rewritten as [3]:

$$C_i = \sum_{m=0}^{7} [Z_l^* 2^m + A_{i3}] \quad \text{.....(9)}$$

where

$$Z_m^* = \sum_{k=0}^{2} A_{ik} b_{k,m} \quad \text{.....(10)}$$

It is worth mentioning that the size of the ROMs has been reduced to 2^3as in table 1, where ROM tables gives the contents of each ROM.

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

$B_{0,m}$	$B_{1,m}$	$B_{2,m}$	The content of 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	$A_{i0}+ A_{i1}+ A_{i2}$

4. The Proposed Architecture

A key objective of this research is to implement a core which performs two different color conversions (*RGB*↔*YCrCb*) on *Sparatan-3E FPGA*. 624 bytes ROMS are needed (312 bytes for each conversion). Figure 2 shows the proposed core and its internal architecture .

The proposed architecture consists of eight identical Processing Elements ($PE_n(1 \le n \le 8)$) . Each PE_n comprises three sub processes element and it contains two memory blocks (A and B) and one parallel signed integer adders. $Block_{nA}$ contain coefficients for *RGB* to *YCbCr* conversion while $Block_{nB}$ used for *YCbCr* to *RGB* conversion, chip selection pin is assigned to convert between the two transformations. As shown in Figure 2 , each ROM store 2^3 coefficients. ROM contents depends on the conversion type as illustrated in Tables 2 and 3.

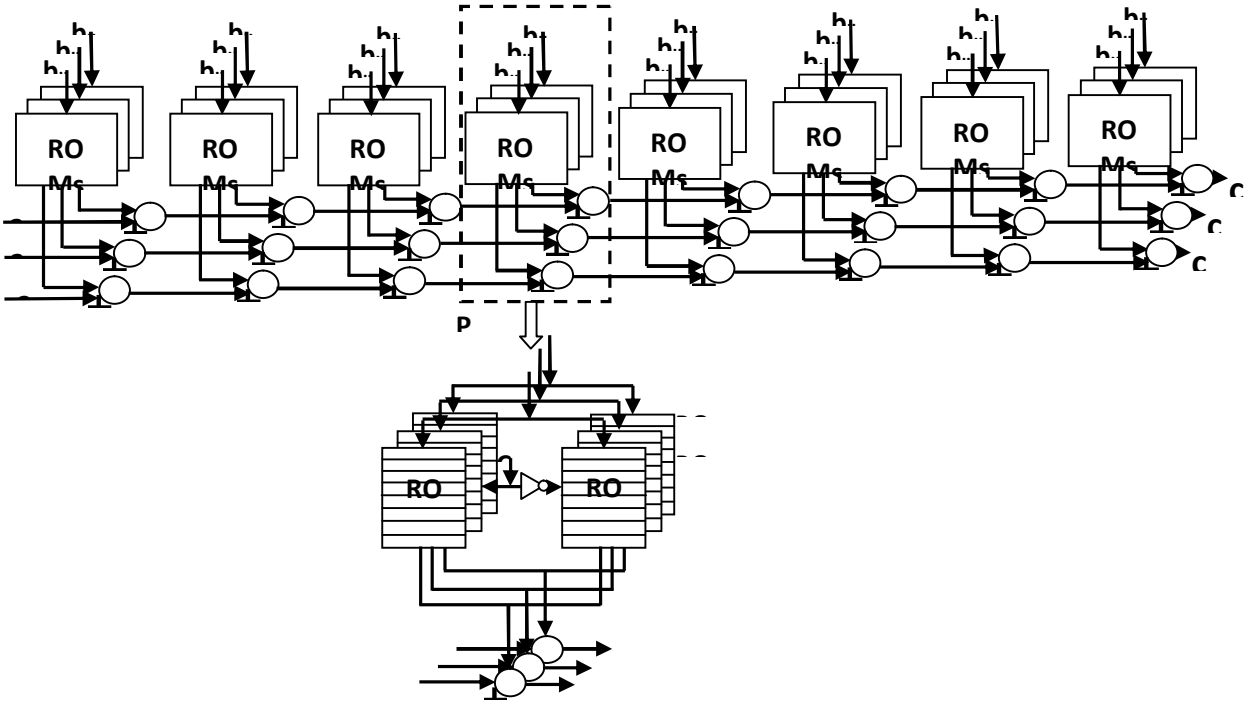

Figure 2: The Proposed CSC

Table 2 : The Content of $Block_A$ ROMs for (RGB to YCbCr) CSC

$R_{ij,0,1}$	$G_{ij,0,1}$	$B_{ij,0,1}$	$Block_{1A}$			$Block_{2A}$			$Block_{3A}$			$Block_{...A}$		$Block_{8A}$		
			ROM_1	ROM_2	ROM_3	ROM_1	ROM_2	ROM_3	ROM_1	ROM_2	ROM_3			ROM_1	ROM_2	ROM_3
0	0	0	0	0	0	0	0	0	0	0	0	...	...	0	0	0
0	0	1	3.13	14.04	-2.27	6.27	28.09	-4.55	12.54	56.19	-9.08	...	...	401.4	1798.14	-290.81
0	1	0	16.12	-9.31	-11.77	32.25	-18.62	-23.55	62.51	-37.24	-47.10	...	...	2064.38	-1191.93	-1507.32
0	1	1	19.26	4.73	-14.04	38.52	9.47	-28.09	77.05	18.94	-56.19	...	...	2465.79	606.20	-1798.14
1	0	0	8.22	-4.73	14.04	16.44	-9.47	28.09	32.89	-18.94	56.19	...	...	1052.67	-606.20	1798.14
1	0	1	11.36	9.31	11.77	22.72	18.62	23.55	45.44	37.24	47.10	...	...	1454.08	1191.93	1507.32
1	1	0	24.35	-14.04	2.27	48.7	-28.09	4.54	97.40	-56.19	9.08	...	...	3117.05	-1798.14	290.81
1	1	1	27.48	0	0	54.9	0	0	109.9	0	0	...	...	3518.46	0	0

Table 3 : The Content of $Block_B$ ROMs for (YCbCr to RGB) CSC

$Y_{ij0,1}$	$Cr_{ij0,1}$	$Cb_{ij0,1}$	$Block_{1B}$			$Block_{2B}$			$Block_{3B}$			$Block_{...B}$	$Block_{8B}$		
			ROM_1	ROM_2	ROM_3	ROM_1	ROM_2	ROM_3	ROM_1	ROM_2	ROM_3		ROM_1	ROM_2	ROM_3
0	0	0	0	0	0	0	0	0	0	0	...	...	0	0	0
0	0	1	0	-3.13	16.13	0	-6.27	32.37	0	-12.55	64.54	...	0	-401.40	2065.40
0	1	0	12.768	-6.50	0	25.53	-13	0	51.07	-26.01	0	...	1634.30	-832.51	0
0	1	1	12.768	-9.64	16.13	25.53	-19.28	32.37	51.07	-38.56	64.54	...	1634.30	-1233.92	2065.40
1	0	0	9.312	9.31	9.31	18.62	18.62	18.62	37.24	37.24	37.24	...	1191.93	1191.93	1191.93
1	0	1	9.312	6.17	25.44	18.62	12.35	50.89	37.24	24.70	101.79	...	1191.93	790.52	3257.34
1	1	0	22.08	2.80	9.31	44.16	5.61	18.62	88.32	11.23	37.24	...	2826.24	359.42	1191.93
1	1	1	22.08	-0.32	25.44	44.16	-0.65	50.89	88.32	-1.31	101.79	..	2826.24	-41.98	3257.34

The precomputed partial products are stored in the ROMs using 13 bits fixed point representation. The inputs and outputs of the two architectures are presented using 8 bits and the outputs are rounded. The initial value for each accumulator is set in advance to (a_{i3} + 0.5), where ($0 \leq i \leq 2$).

The architecture operates in a parallel manner ; during the first clock all bits of *RGB* or *YCbCr* are applied to ROMs blocks and they are processed as address of *LUT* to calculate the multiplication result. Then parallel accumulation of results will be a ccomplished to obtain the first output of all components after one clock. The entire image conversion can be carried out in (Latency + (N × M)), where Throughput = 1 + (N × M) clock cycles, while using the Pipeline *DA* algorithm [3,4,10], the conversion can be carried out in 8 + (N × M) clock cycles, i.e., after one clock all three color components give one output in each cycle.

CSC FPGA Implementation

The CSC architecture presented in this paper has been synthesized using Xilinx *ISE9*.2i. The target device is *Spartan-3E XC3S500 FPGA*. The architecture has been captured in *VHDL* and the resulting hardware simulations have been performed in Modelsim as shown in Figures 3 and 4 . The synthesis results are presented in Figure Table 4. Uses 259 Slices and reaches an operating frequency of 265.534*MHz*.

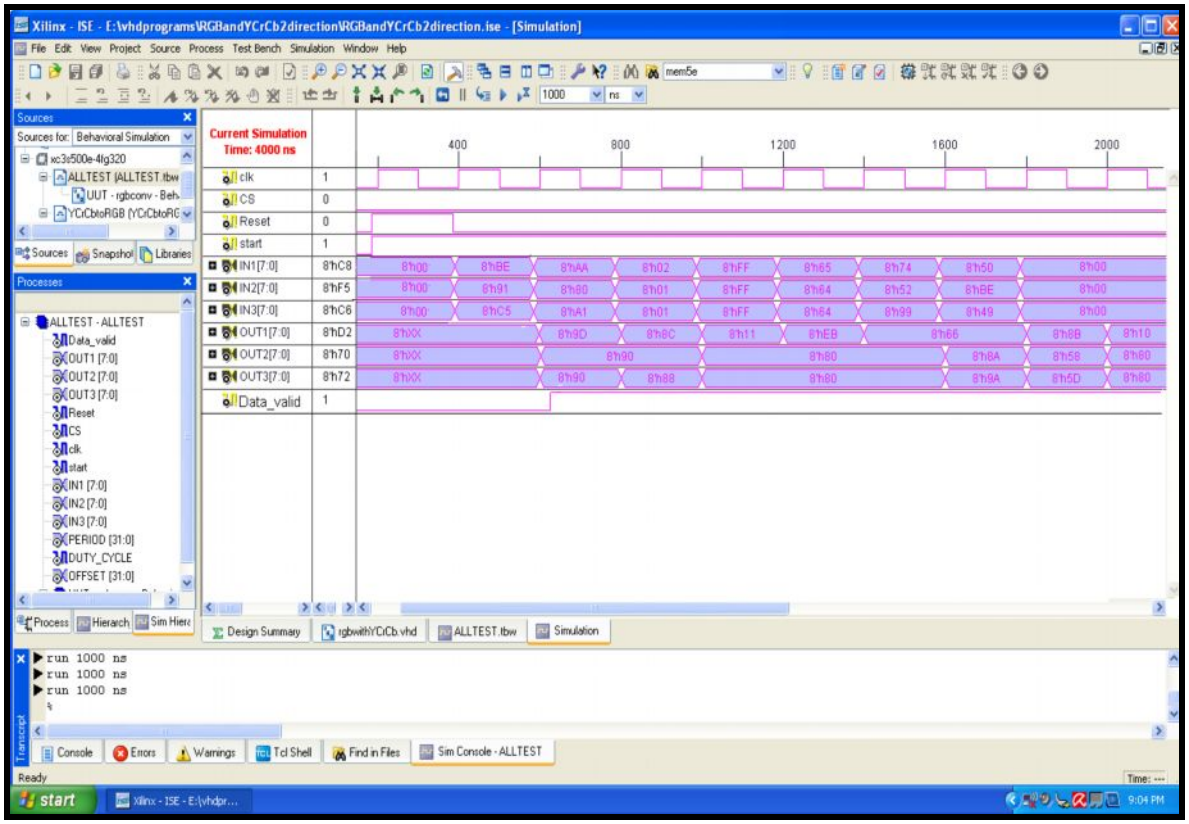

Figure 3: The RGB to YCbCr waveform conversion

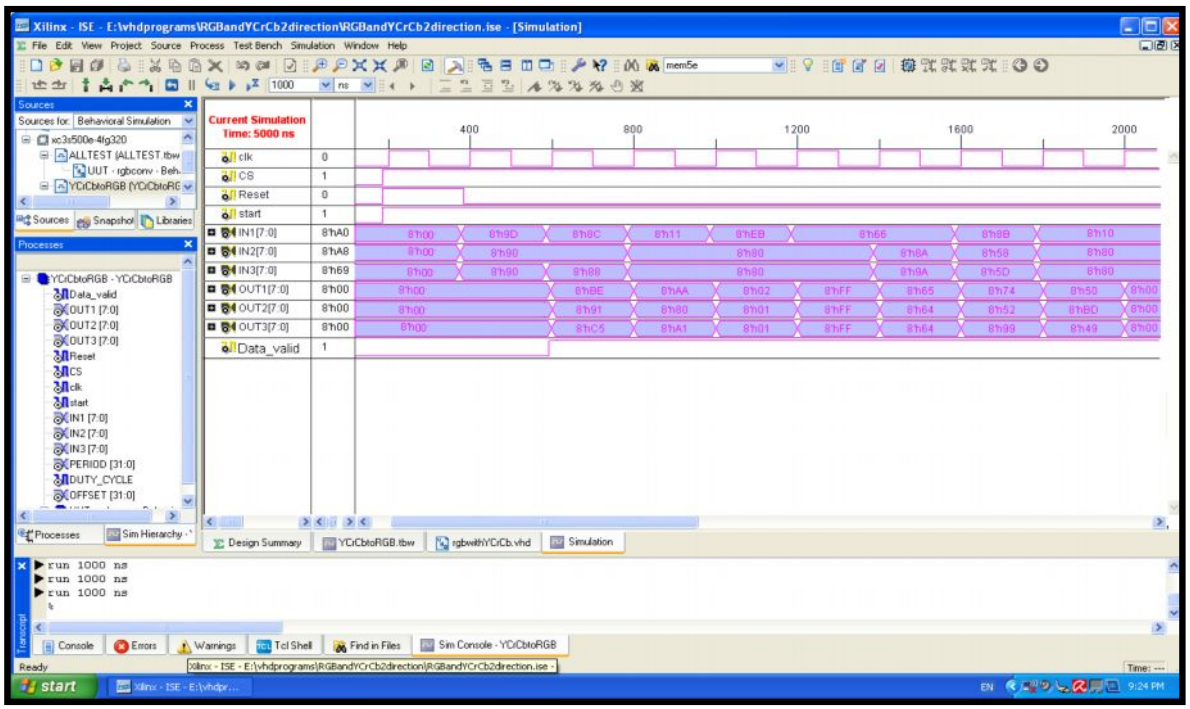

Figure 4: The YCbCr to RGB waveform conversion

Table 4: Spartan3-E utilization summary for CSC

Function	RGB to YCbCr & YCbCr to RGB
No. of Slices	259 out of 4656 5%
No. of Slice Flip Flops	239 out of 9312 2%
No. of 4 input LUTs	460 out of 9312 4%
No. of bonded IOBs	50 out of 232 21%
Number of GCLKs	1 out of 24 4%
Maximum Frequency	265.534 MHz

6. Xilinx Color Space Core Generator:

The Xilinx CORE Generator system generates and delivers parameterizable cores optimized for Xilinx FPGAs. It is used to design high-density Xilinx FPGA devices and achieves high performance results, while at the same time, reducing the design time. The CORE Generator is included in ISE Xilinx Foundation, with a variety of cores memories and storage elements, math functions , DSP functions, and a variety of basic elements. Elements. The *RGB* to *YCbCr* and the *YCbCr* to *RGB* cores are generated by the Xilinx Core Generator10.1 [1,9,13] and configured for 8-bit input data, 8-bit output data. After Xilinx *ISE*10.1Place & Route on *Spratan-3E*, the resource utilizations are shown in Table 5.

Table 5: Spartan3-E utilization summary for CSC Core Generator

function	RGB to YCbCr	YCbCr to RGB
No. of Slices	199 out of 4656 4%	151 out of 4656 3%
No. of Slice Flip Flops	370 out of 9312 3%	245 out of 9312 2%
No. of 4 input LUTs	202 out of 9312 2%	165 out of 9312 1%
Number used as logic	165	154
Number used as Shift registers	37	11
Number of MULT18X18SIOs	4 out of 20 20%	4 out of 20 20%
Number of GCLKs	1 out of 24 4%	1 out of 24 4%
Maximum Frequency	230.97 MHz	207.340 Hz

7. Results and Comparison

A comparison of *CSC* architecture proposed in this paper with the simple *DA* architecture presented by Bensaali [3,10] shows that the proposed one processes input pixels in parallel instead of pipeline treatment, thus omitting all shift operations. Bensaali's design [3,10] requires eight cycles per pixel, while the proposed core needs one clock only. The proposed architecture also performs better than the CAST design [14] as the latter requires fixed five cycles per pixel and occupies 303 slices in Xilinx Spartan FPGA device. The area occupied by the proposed *CSC* design is around 259 slices for the same device as illustrated in Table 6 while a single clock cycle is utilized.

Table 6: The Performance of various CSC architectures.

Reference	[3]	[4]	[9]	[10]	[13]	[14]	Proposed
Function	F& B	F	F	F	B	F	F & B
ROM (bytes)	624	N*M*3	-	312	-	-	624
Shift operations	21	1	-	21	-	-	-
Adder	24	1	9	24	9	3	24
Multiplier	-	1	4	-	4	9	-
Shift register	-	1	$40_{(1bit)}$	-	$30_{(1bit)}$	3 Groups	-
Multipliexer	-	1	6	-	6	-	-
Comparator	-	1	-	-	-	-	-
Latency(Cycles)	8	8	11	8	7	5	1
Average no. of cycles per conversion	8	4	1	1	1	1	1
Power	low	medium	high	low	high	high	lower
Frequency(MHZ)	234	127	230.09	263	207.340	109	265.534
FPGA	XCV50 E-8	XC2S300 E	XC3S500 E	XCV50 E-8	XC3S500 E	XC2S300 E	XC3S500E

Note: F =Forward Color conversion (RGB to YCbCr)
B=Backward Color Conversion (YCbCr to RGB)

The architecture for color space conversion in *RGB* to *YCbCr* domain proposed by M. Bilal [4] exploits the similarity in bit-planes of a natural image to bring the algorithmic efficiency of Distributed Arithmetic (*DA*) approach . The new hardware is very simple(fewer resources) as compared to that.
The architecture for color space conversion in *RGB* to *YCbCr* domain proposed by M. Bilal [4] exploits the similarity in bit-planes of a natural image to bring the algorithmic efficiency of Distributed Arithmetic (*DA*) approach . The new hardware is very simple(fewer resources) as compared to that.

Table 7 illustrates the hardware/software implementations comparison in terms of the RMS error due to the use of difference data representation in the two implementations

$$RMS_{Error} = \sqrt{1/(N*M)\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(I_{soft}(i,j) - I_{hard}(i,j))^2)}$$

….(11)

where N*M image size

$I_{soft}(i,j)$ is the pixel of software image

$I_{hard}(i,j)$ is the pixel of hardware image

Table 7 shows the test results for three different images (Baboon image (512×512), Pepper (256×256) and Sailboat image (256×256)). It can be seen that the same converted im age can be tiobtained fastly when using the proposed *FPGA* implementation, with a minimum error (due to the use of difference data representation in the two implement

Table 7 : Software/ Hardware implementations for RGB↔YCrCb CSC comparisons

Original Image	Software RGB to YCbCr Implementation	Hardware RGB to YCbCr Implementation	Hardware YCbCr to RGB Implementation	RMS Error In Ref[3]		Proposed RMS Error		Computation time (ms)		
								software	Hardware	
									In Ref[3]	proposed
				Y	0.487	Y	0.1825	126	1.12	0.987
				Cb	0.630	Cb	0.1999			
				Cr	0.461	Cr	0.1888			
				Y	0.684	Y	0.1681	43	0.28	0.246
				Cb	0.830	Cb	0.2286			
				Cr	0.396	Cr	0.2021			
				Y		Y	0.1773	43	0.28	0.246
				Cb		Cb	0.1814			
				Cr		Cr	0.1612			

8. Conclusions

A parallel architecture for forward and backward Color Space Conversion has been proposed . It combines the area efficiency of distributed arithmetic with the efficient parallel based algorithm to save redundant computations. This approach is important from power consumption point of view since only one clock cycle is needed to first obtain output converted data. The proposed architecture has been compared as a IP functional unit in a *Spartan-3E FPGA*. The results have been verified through implementation on Xilinx *Spartan FPGA* with maximum throughput 265.5M sample/sec and minimum RMS error. Power reduction has been also realized by minimizing the number of arithmetic operations. This type of architecture is especially useful for mobile devices where power consumption is critical and directly related to the complexity of hardware and clock frequency.

Future works may include the use of such CSC core for real time video processing and extend it to other types of color space conversion on Programmable Gate Arrays.

References:

[1] B. Payette, “Color Space Converter: RGB to YCbCr,” *Xilinx Application Note,* XAPP637, V1.0, September 2002.

[2] R.C. Gonzalez and R.E. Woods, “Digital Image Processing,” Second Edition, Printice Hall Inc, 2002.

[3] F.Bensaali, A.Amira and A.Bouridane. ″Design and Implementation of Efficient Architectures for Color Space Conversion″ , ICGST-GVIP Journal, Volume 5, Issue1, December 2004.

[4] M. Bilal and Sh. Masud ," Efficient Color Space Conversion using Custom Instruction in a RISC Processor", IEEE International Symposium on Circuits and Systems, 2007,pp. 1109-1112.

[5] N. Tiwari and S. C.Reddy, "Performance Measurement of a Fully Pipelined JPEG" Codec on Emulation Platform", Advance Computing Conference (IACC), 2010 IEEE 2nd International 978-1-4244-4791-6/10 , pp. 167 - 171.

[6] T.S. Wong, C. A. Bouman, Fellow, IEEE, I. Pollak, and Z. Fan, "A Document Image Model and Estimation Algorithm for Optimized JPEG Decompression", IEEE Transactions on Image Processing, Vol. 18, No. 11, Nov. 2009.

[7] T. Saidani , D. Dia, W. Elhamzi, M. Atri and R. Tourki, "Hardware Co-simulation For Video Processing Using Xilinx System Generator", Proceedings of the World Congress on Engineering 2009 Vol I WCE 2009, July 1 - 3, 2009, London, U.K.

[8] C. T. Johnston, K. T. Gribbon and D. G. Bailey," FPGA based Remote Object Tracking for Real-time Control", 1st International Conference on Sensing Technology Nov. 21-23, 2005 Palmerston North, New Zealand.

[9]Xilinx," RGB to YCrCb Color-Space Converter v1.0 ", Product Specification, DS657 March 24, 2008.

[10] F. Bensaali, A. Amira and S.r Chandrasekaran," Power Modeling and Efficient FPGA Implementation of Color Space Conversion", 13th IEEE International Conference on Electronics, Circuits and Systems, Vol. ICECS apos;06. , Issue , 10-13 Dec. 2006 pp:164 - 167.

[11] J. Galindo , E. Peskin , B. Larson and G. Roylance, " Leveraging Firmware in Multichip Systems to Maximize FPGA Resources: An Application of Self-Partial Reconfiguration", 2008 International Conference on Reconfigurable Computing and FPGAs, Print ISBN: 978-1-4244-3748-1, pp:139 -144, Location: Cancun.

[12] Y.Yang, P. Yuhua, and L. Zhaoguang," A Fast Algorithm for YCbCr to RGB Conversion", IEEE Transactions on Consumer Electronics, Vol. 53, No. 4, pp:1490 - 1493 Nov. 2007.
[13] Xilinx,"YCrCb to RGB Color-Space Converter v1.0", Product Specification, DS659 March 24, 2008.
[14] CSC color conversion core, CAST Inc. http://www.castinc.com/cores/csc/csc-xilinx.shtml (Oct 06, 2006).

The work was carried out at the college of Engg. University of Mosul