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

أ.د. أحلام فاضل محمود
قسم هندسة حاسبات

ملخص

ان انتشار الحاسبات والإنترنت والأنواع المختلفة من أدوات الفيديو التفاعلية، في أغلب التطبيقات المتعددة الأوساط وقعت منها يستخدم تمثيل مختلف اللوان يجير المصمم الرقمي على تنفيذ معمارية للتحويل بينهما. لذلك كان هدف البحث تصميم محول لللوان يكون مفيد لعدد التطبيقات وظيفته الأساسية التحويل من فضاء لون إلى آخر (RGB to YCbCr and YCbCr to RGB) للمعمارية المقترحة مستندة على مبدأ الصلاحيات المروعة حيث اطلق على رقاقة البوابات القابلة للبرمجة Spartan-3E XC3S500 بشكل أقل التطبيق المقترح أعطى نتائج أفضل مقارنة بالتطبيقات الأخرى. التحديثات نفذت في الحساب المروعة لتخفيف عقد المكونات بالأداء الأفضل في المساحة وتقليل معدل الحصول على الإخراج وكمية المعلومات التي يتم عالجتها.

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, HDTV 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 et al 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 Xilinx 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 its 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.
3. CSC Using Distributed Arithmetic

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

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

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

\[
\begin{bmatrix}
    R \\
    G \\
    B
\end{bmatrix} =
\begin{bmatrix}
    1.164 & 0 & 1.59 \\
    1.164 & -0.392 & -0.813 \\
    1.164 & 2.017 & 0
\end{bmatrix}
\begin{bmatrix}
    Y \\
    Cb \\
    Cr
\end{bmatrix} +
\begin{bmatrix}
    -222.9 \\
    135.616 \\
    -276.8
\end{bmatrix} \ldots \ldots (2)
\]

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]:

\[ C_i = \sum_{k=0}^{N-1} A_{ik} B_k \]  

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

\[ B_k = \sum_{m=0}^{W-1} b_{k,m} 2^m \]  

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,

\[ C_i = \sum_{k=0}^{N-1} A_{ik} \sum_{m=0}^{W-1} b_{k,m} 2^m \]  

Interchanging the two summations gives:

\[ C_i = \sum_{m=0}^{W-1} \sum_{k=0}^{N-1} A_{ik} b_{k,m} 2^m \]  

Let us define \( Z_m \) as:

\[ Z_m = \sum_{k=0}^{N-1} A_{ik} b_{k,m} \]  

Then, (6) becomes

\[ C_i = \sum_{m=0}^{W-1} Z_m 2^m \]  

\( i = 0,1,2 \)

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 (\( b_{0,m}, b_{1,m}, \ldots, b_{(N-1),m} \)) is used as an address to retrieve the corresponding \( Z_m \) values. The ROM’s content is different and depends on the 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 \( (N = 4 \text{ and } W = 8) \), then (8) can be rewritten as [3]:

\[ C_i = \sum_{m=0}^{7} [Z_m^* 2^m + A_{i3}] \]  

where

\[ Z_m^* = \sum_{k=0}^{3} A_{ik} b_{k,m} \]  

It is worth mentioning that the size of the ROMs has been reduced to \( 2^3 \) as in table 1, where ROM tables gives the contents of each ROM.
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 ≤ n ≤ 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<n>A contain coefficients for RGB to YCbCr conversion while Block<n>B 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](image)

Table 1: Content of the ROM i (0 ≤ i ≤ 2)

<table>
<thead>
<tr>
<th>B&lt;i,0&gt;</th>
<th>B&lt;i,1&gt;</th>
<th>B&lt;i,2&gt;</th>
<th>The content of ROM&lt;i&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A11</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A11 + A12</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A00 + A12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A00 + A11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A00 + A11 + A12</td>
</tr>
<tr>
<td>Block A</td>
<td>Block B</td>
<td>Block C</td>
<td>Block D</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block A</th>
<th>Block B</th>
<th>Block C</th>
<th>Block D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The conversion from RGB to YCC (CSC)
<table>
<thead>
<tr>
<th></th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3: The content of blocks for (X1, X2, X3, X4) CSC
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 \((a_3 + 0.5)\), 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 accomplished to obtain the first output of all components after one clock. The entire image conversion can be carried out in \((\text{Latency} + (N \times M))\), where Throughput = \(1 + (N \times M)\) clock cycles, while using the Pipeline DA algorithm [3,4,10], the conversion can be carried out in \(8 + (N \times M)\) 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 265.534MHz.

![Figure 3: The RGB to YCbCr waveform conversion](image)
Figure 4: The YCbCr to RGB waveform conversion

Table 4: Spartan3-E utilization summary for CSC

<table>
<thead>
<tr>
<th>Function</th>
<th>RGB to YCbCr &amp; YCbCr to RGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slices</td>
<td>259 out of 4656 5%</td>
</tr>
<tr>
<td>No. of Slice Flip Flops</td>
<td>239 out of 9312 2%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>460 out of 9312 4%</td>
</tr>
<tr>
<td>No. of bonded IOBs</td>
<td>50 out of 232 21%</td>
</tr>
<tr>
<td>Number of GCLKs</td>
<td>1 out of 24 4%</td>
</tr>
<tr>
<td>Maximum Frequency</td>
<td>265.534 MHz</td>
</tr>
</tbody>
</table>

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 ISE10.1 Place & Route on Spratan-3E, the resource utilizations are shown in Table 5.
Table 5: Spartan3-E utilization summary for CSC Core Generator

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

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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>F &amp; B</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F &amp; B</td>
</tr>
<tr>
<td>ROM (bytes)</td>
<td>624</td>
<td>N<em>M</em>3</td>
<td>-</td>
<td>312</td>
<td>-</td>
<td>-</td>
<td>624</td>
</tr>
<tr>
<td>Shift operations</td>
<td>21</td>
<td>1</td>
<td>-</td>
<td>21</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Adder</td>
<td>24</td>
<td>1</td>
<td>9</td>
<td>24</td>
<td>9</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>Multiplier</td>
<td>-</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>4</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>Shift register</td>
<td>-</td>
<td>1</td>
<td>40 (1bit)</td>
<td>-</td>
<td>30 (1bit)</td>
<td>3 Groups</td>
<td>-</td>
</tr>
<tr>
<td>Multiplierex</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Comparator</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Latency(Cycles)</td>
<td>8</td>
<td>8</td>
<td>11</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Average no. of cycles per conversion</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Power</td>
<td>low</td>
<td>medium</td>
<td>high</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>lower</td>
</tr>
<tr>
<td>Frequency(MHZ)</td>
<td>234</td>
<td>127</td>
<td>230.09</td>
<td>263</td>
<td>207.340</td>
<td>109</td>
<td>265.534</td>
</tr>
<tr>
<td>FPGA</td>
<td>XCV50 E-8</td>
<td>XCV2S300 E</td>
<td>XC3S500 E</td>
<td>XCV50 E-8</td>
<td>XC3S500 E</td>
<td>XCV2S300 E</td>
<td>XC3S500E</td>
</tr>
</tbody>
</table>

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

\[
RMS_{error} = \sqrt{\frac{1}{N \times M} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (I_{soft}(i,j) - I_{hard}(i,j))^2}
\]

...(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 image can be obtained fastly when using the proposed FPGA implementation, with a minimum error (due to the use of difference data representation in the two implement
<table>
<thead>
<tr>
<th>Proposed Hardware Run Time (ms)</th>
<th>Proposed Hardware Error</th>
<th>Proposed Hardware Error in R'G'B'</th>
<th>Proposed Hardware Error in Y'Cr'Cb'</th>
<th>Proposed Hardware Error in R'Y'CB'</th>
<th>Proposed Hardware Error in Y'CBC'Y'</th>
<th>Proposed Hardware Error in R'Y'CG'</th>
<th>Proposed Hardware Error in Y'CG'Y'</th>
<th>Proposed Hardware Error in R'Y'CG'</th>
<th>Proposed Hardware Error in Y'CG'Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.28</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
</tr>
<tr>
<td>0.28</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
</tr>
<tr>
<td>0.28</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
<td>0.151</td>
</tr>
</tbody>
</table>

Table 7: Software Hardware Implementations for YCCP CSC computations.
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: