International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 9, No.1 (Jan-2020)
http://dx.doi.org/10.12785/ijcds/090102

FPGA Implementation of Enhanced JPEG Algorithm for Colored Images

Sa'ed Abed[1], Mariam AlKandari[1], Huda AlRasheedi[1] and Imtiaz Ahmad[1]

[1] *Department of Computer Engineering, Kuwait University, Kuwait*

Received 16 Jun. 2019, Revised 23 Sep. 2019, Accepted 30 Dec. 2019, Published 01 Dec. 2020

Abstract: Image quality and size have been growing very fast over the last decades requiring large storage space and high transmission rate. Image compression is an efficient method used to reduce the size of the image. JPEG algorithm has been considered as one of the famous techniques used for image compression. This paper proposed and implemented an optimized hardware solution called Hybrid Compression using Faster Color Conversion and Run Length (HC-FCC-RL) algorithm for JPEG algorithm based on FPGA to reduce the latency and accelerate the compression process. The paper also proposed a Fast Color Conversion with Approximation (FCCA) step to accelerate the conversion process from RGB to YC_bC_r. Using approximate techniques will reduce the number of resources used as well as the latency with some percentage error. In addition, the paper proposed a Parallel Run Length (P-RL) algorithm to speed up the design. The enhancement approach in this paper aimed to optimize the overall design of the JPEG algorithm targeting color images. To evaluate the performance of the proposed framework, the HC-FCC-RL architecture was implemented in Verilog and synthesized on FPGA board. The color conversion architecture uses 331 logic elements, a latency of 13.930 ns for converting the three colors component of blocks and a power dissipation of 861.03 mW. The percentage errors for the color conversion is 11.6%, 1.8% and 5.3% for Y, C_b, C_r components, respectively. The Run Length algorithm performs better than existing work by saving 48.36% in logic elements, 53.79% in latency and 62.86% in power dissipation. Thus, the proposed work demonstrated superior performance compared to current work in the literature.

Keywords: Image compression, FPGA, Color Conversion, JPEG Algorithm, Run Length Algorithm, Approximation Technique

1. INTRODUCTION

Joint Photographic Experts Group (JPEG) is one of the algorithms that helps in reducing the size of an image and excluding the redundancy in the image by compressing it [1, 2]. JPEG algorithm consists of six general steps. The first step is color conversion where the color is converted from red green and blue (RGB) color space to luminance chrominance (YC_bC_r) color space. In the second step, the image is divided into blocks. Third step is Forward Transform where Discrete Cosine Transform (DCT) is applied on each block. In the fourth step, the 2D array from DCT step is represented in another form using Quantization step. Zig-Zag reordering and Run Length algorithms are applied sequentially in the fifth step. The Last step uses Huffman coding algorithm which is used to compress the image farther. Fig. 1 shows the steps of compressing a colored image using JPEG algorithm.

Many technologies have been developed for improving the quality and the resolution of images. However, these images will take large space and consume huge power for processing them. Images are used in many applications such as mobile phones, computers, medicals, satellites and many other fields. Naturally, these images need to be transferred through the network. Transmission rate of these images will be extremely large, as it takes many Bytes. Therefore, image compression is an essential way to reduce the size of an image and hence reducing the transmission time.

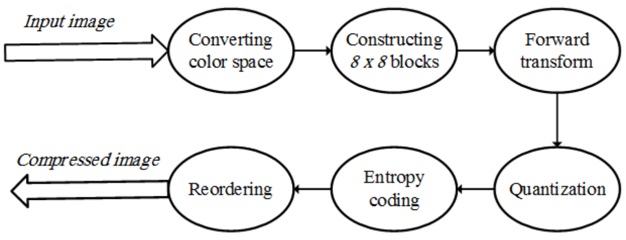

Figure 1. JPEG Algorithm Encoding Steps

Different software solutions have been proposed to perform JPEG algorithm, but these solutions have limitations in general purpose processor as they consume much power and add an overhead work on the process [3]. On the other hand, several hardware solutions are

E-mail: s.abed@ku.edu.kw, mariamalkandari@gmail.com, eng.hudaalrasheedi@gmail.com, imtiaz@eng.kuniv.edu.kw

proposed to solve the software limitations. Many researchers made effort in enhancing the algorithm by optimizing some part of the algorithm. Lack of optimized algorithm for color images motivates us to work on proposing a general algorithm that can be applied for both gray and color images. Moreover, most of the published work are optimizing one step of JPEG algorithm, while we are optimizing and improving two combined steps from JPEG algorithm. The work in this paper is an extension of [4] and is aimed to optimize two steps of the algorithm to enhance the overall design in general and mainly reduce the latency of the algorithm for colored images. These steps are converting color space and Run Length algorithm.

The first step in JPEG compression algorithm is color space conversion for compressing colored images from one color space to another to improve the design. The proposed solution, Fast Color Conversion with Approximation (FCCA) method contributes in reducing the latency of the design proposed in [5]. Approximation techniques are used in this paper in order to reduce the latency of color conversion step. Hence, accelerating JPEG algorithm since converting color space is a necessary step to start compressing the image.

The other optimization step is the Run Length algorithm which has big role in the compression process. It scans the whole array produced by the Zig-Zag algorithm to remove the redundancy. This algorithm has to scan a huge array to remove the redundancy and hence it takes a very long time. The proposed solution, Parallel Run Length (P-RL) in this paper is to divide the array into sub-arrays and perform Run Length algorithm in parallel, which will make the compression process faster.

The proposed design Hybrid Compression using Faster Color Conversion and Run Length (HC-FCC-RL) algorithm is implemented using Verilog based on Field Programmable Gate Array (FPGA) and analyzed using Quartus. In addition, MATLAB is used to convert the image into array of numbers. The proposed framework demonstrated superior image quality compared to original work in the literature. The proposed approximate color conversion (FCCA) uses 331 logic gates, a latency of 13.930 ns for converting the three colors component of *8x8* blocks and a power dissipation of 861.03 mW. The error rate of producing the YC_bC_r is 11.6% for Y component, 1.8% for C_b and 5.3% for C_r, respectively. The run length algorithm (P-RL) uses 3,468 logic gates, a latency of 24.312 ns and a power dissipation of 4,861.98 mW. The results are carried out on each color component at a time. The proposed framework demonstrated superior performance compared to current work in the literature.

The contribution of this paper can be summarized as:

- Proposing a faster compression for colored JPEG images.
- Reducing the latency of the first step in JPEG algorithm by applying approximation technique.
- Reducing the size of the design for the original color conversion step.
- Performing parallel Run Length algorithm.
- Implementing JPEG algorithm with all the modifications on FPGA board.
- Performing and comparing the results with the original design of JPEG algorithm.

The structure of the rest of the paper is as follows. Literature review of this work is presented in Section 2. The background of the JPEG algorithm and detail explanations of its step is provided is Section 3. The HC-FCC-RL methodology and the architecture of the design is presented in Section 4. The experimental results are shown in Section 5. Finally, we conclude the paper in Section 6 and provide some trends for future works.

2. RELATED WORK

JPEG compression is a standard that has been mostly used since many years. As the JPEG lossy compression is commonly used, it took the attention of many researchers. Numinous research has been published on the basic JPEG algorithm and how to improve it. JPEG lossy compression algorithm can be divided into four steps: Constructing *nxn* blocks and converting color space, Forward Transform, Quantizer and Symbol encoder. The general explanation of the algorithm is expressed in [1, 6]. Marcus discussed the general overview of the JPEG algorithm and showed some experimental results [1]. Lin in [6] gave an overview of the image compression in general, JPEG compression in specific, some details of the algorithm was missing such as the Huffman coding and he stated that there was some improvement that could be applied to the algorithm. Interestingly, he showed that changing the block size depending on the image could sometime be better than the standard block size that is *nxn*. Rawat *et al.* [7] analyzed and provided comparison of different image compression techniques such as Wavelet, JPEG/DCT, VQ, and Fractal.

Many researches had different improvements on JPEG algorithm and used different approaches to enhance it. The authors in [8] compared different enhancements on JPEG algorithm where most of the algorithms are already exists in software implementation, i.e. mozjpeg and libjpeg. Moreover, they compared these enhancements with a modern image compression algorithm named JPEG 2000. Furthermore, they gave an additional opportunity to enhance and optimize JPEG algorithm. The purpose of their research is to assist in deciding the best compression algorithm for a specific image. Alam *et al.* [9] improved one step of the JPEG algorithm by modifying the luminance quantization table for color image. Rippel *et al.* [10] carried out their research on a machine learning-based approach to lossy image compression which performs all existing codecs, while running in real-time.

Agostini *et al.* [5] enhanced the first step in compressing a colored image which is the conversion process of a colored image from RGB to YC_bC_r. This process must operate as fast as possible so that the compression process will not be delayed more. Thus, they tried to improve the design by pipelining it into five stages and improving the shifters used for the multiplication process.

Researchers in [11, 12, 13, 14] proposed fast solutions for color conversion targeting a general purpose processor. Kim *et al.* [11] had implemented an improved performance of color conversion using the concept of shared memory. They used OpenMP API which is used for parallel programming and divided the color conversion into multiple threads among different cores. Kim *et al.* in [12] again explained the importance of utilizing the multi-core processor for converting the color. They used OpenMP and TBB tools for parallel coding to perform the conversion on multicore and then compared parallel results with the serial results. Many processors had supported Single-Instruction-Multiple-Data (SIMD) and color conversion were implemented using SIMD. Color conversion has limitation when implemented in SIMD, thus Shahbahrami *et al.* in [13] proposed an extended sub-words and Matrix Register File (MRF) to increase the conversion further. Shen *et al.* in [14] enhanced the speed of the color conversion using Graphic Processing Units (GPU). They created a multiple thread for each color conversion kernel using CUDA which is tool used for developing parallelism applications in GPU.

Naumowicz *et al.* [15] illustrated the transistor level of color conversion. They used CMOS technology, studied the low level of the color conversion and applied migration method on the design. The migration method resulted in having low power and low area for color conversion.

Macieira *et al.* [16] provided a software-hardware co-design of two algorithms which are color conversion and thresholding. The architecture uses four color conversion blocks for parallelism. The system takes 12-bit word of RGB and converted into 32-bit floating point. They compared their design with a pure software solution and concluded that the hardware design is much faster than the software design.

Bensaali *et al.* [17] proposed a low power solution for color conversion using Distributed Arithmetic (DA). Color conversion equations have products that can be pre-computed. The pre-computed products are stored in ROM with 13-bit fixed point representation. They used an efficient way for rounding the output design of the architecture by adding 0.5 value to the result. Sapkal *et al.* [18] described the conversion of color space from RGB to YC_bC_r using PCI interface which is an FPGA based computing card. The proposed architecture in [19] used approximate equations of converting colored image from RGB to YC_bC_r. The authors multiplied and divided the three equations by 256 for rounding and scaling purpose and also proposed to add 0.5 to the result as an efficient way to round the final result.

Researchers in [19, 20, 21] used the same approximation concept for converting color space in order to accelerate the conversion process. They converted the floating point numbers in the equations of converting color space into integers as a fast conversion method. Chernov *et al.* [19] used this approximation technique to convert color space from RGB to HSV (hue, saturation and value) and vice versa. They did not use lookup table technique unlike Jiang *et al.* [20] where they converted the floating point numbers into integers and transformed multiplications to lookup tables and additions. They also used pipelining technology for farther acceleration. Liu *et al.* [21] replaced all the floating point multiplication into fixed point shifters and adders as a fast method to convert color space from YC_bC_r to HSV. The authors in [22] proposed FPGA design for the color space conversion which is used in Video Graphics Array (VGA) standard. They used the standard equation for YC_bC_r to RGB transformation but since the equation needs fractional arithmetic they simplified the design by multiply both sides of the standard equation by a constant which is equal to 256. In addition, since the interface of the SRAM supports only 16 bits and each pixel has 24 bit, so 8 bit should be discarded. Thus, the authors selected 5 most significant bits (MSB) of R, 6 MSB of G and 5 MSB of B since human eye is more sensitive the green color.

Because Run Length algorithm performs big part of the compression process by removing the redundancy, Tu *et al.* [23] made an enhancement on the Run-Length coding algorithm. This algorithm usually encodes (RUN/LEVEL) pairs all together at the same time where RUN represents the number of zeroes and LEVEL is the value of the following nonzero coefficient. The enhancement was made so that the algorithm will deal with a small number of symbols and encoding RUN and LEVEL separately. Gupta *et al.* [24] followed another approach to improve Run-Length coding algorithm by using three symbols (RUNLENGTH, SIZE, AMPLITUDE) instead of two (RUN/LEVEL) which is the standard of Run-Length algorithm. Their research was done for only gray scale images. Akhtar *et al.* [25] applied Run-Length coding algorithm for biomedical imaging. They improved the Run-Length encoding scheme by removing the unintended redundancy by using an ordered pair only when a zero occurs. Therefore, the Run-Length will not make a pair for any ASCII character that does not have zeroes preceding it. This will reduce the number of pairs as well as the output size of the Run-Length process. The same authors in [26] had applied the same enhancement technique on the same enhancement on Run-Length algorithm but for Space Research Program at Institute of Space Technology (IST). Vohra *et al.* in [27] focused on compressing the test data, which is stream of data that represents the defects in the digital circuit. They proposed three techniques for the compression that are based on run length algorithms. These techniques are 10 Coded, Selective CCPRL and Optimal selective count compatible run length (OSCCPRL). The 10 Coded (10C) technique divided the test data into sub blocks and considered the compatibility checks between the adjacent blocks by adding prefix and sub-prefix bit. In the other hand, the selective CCPRL chose either to encode by CCPRL scheme or not. OSCCPRL technique used a combination of the first and second technique. The data is divided into levels and the inter level encoding is done by 10C and the intra level is done by SCCPRL.

After reviewing the previous work carried out by the researchers, it can be concluded that the idea proposed which was pipelining the design that converts the colored image from RGB to YC_bC_r [3] is close of what we are trying to propose. However, we are looking forward to optimizing the design more by adding an approximation concept that is explained in the methodology section and find a better way to express the equations of converting color space without affecting the quality of the images.

Run length algorithm is performing big part of the compression process by removing the redundancy. Many researchers are trying to optimize this algorithm to speed up the compression time of a gray scale image. Their proposed work was to optimize the algorithm either by encoding RUN and LEVEL separately [23] or by encoding three symbols (RUNLENGTH, SIZE, AMPLITUDE) instead of two, in addition the research was done only for the gray scale images [24]. However, in this paper, we are going to divide the array produced by the previous step into sub-arrays and scan these sub-arrays using Run Length algorithm at the same time which will make the compression process faster. Our design is targeting colored images unlike the previous researches where they targeted only gray scale images.

3. BACKGROUND

JPEG compression algorithm is one method to solve huge image size and slow transmission issues of an image. JPEG algorithm can be described in the following steps:

- Converting color space: A colored image is converted from red, green and blue (RGB) color space to luminance chrominance (YC_bC_r).
- Constructing *8x8* blocks: The image is divided into *8x8*pixels blocks.
- Forward transform: Discrete Cosine Transform (DCT) is used for each *8x8* blocks in this step.
- Quantization: Each coefficient in the DCT output is divided by its corresponding quantized value from a standardized quantization table.
- Reordering: Zig-Zag reordering and Run Length algorithms are applied sequentially. Therefore, a two dimensional array is going to be reordered into one dimensional array so that the redundant zeroes are discarded from the one dimensional array.
- Entropy coding: Huffman coding algorithm is going to be used at this step in order to reduce the redundancy more.

In this section, color conversion and Run length algorithms are going to be described in details in order to understand HC-FCC-RL design. In addition, DCT will also be explained since it is the critical path and the main step in JPEG algorithm.

A. Color Conversion

Color space conversion is the first operation that must be accomplished before starting the compression process for a colored image. The conversion is necessarily to convert a colored image from RGB representation that is the most popular color space used in digital images to luminance chrominance (YC_bC_r) representation where luminance (Y) indicates intensity of the image and chrominance (C_bC_r) indicates the blue and red colors information, respectively. The reason behind this conversation is that an image in RGP representation is not suitable for image compression applications and the RGB color space is a highly correlated color space [28]. Fig. 2 shows an image that is converted from RGB color space to (YC_bC_r) color space.

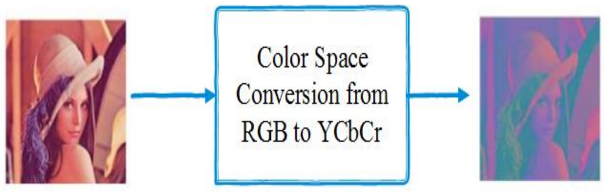

Figure 2. Color Conversion Block Diagram

The conversion process is done using the standard formulas presented in [5, 28]. For each operation of these formulas, a suitable hardware is needed to perform the addition or/and multiplication, for example adders, multipliers or/and shifters.

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B + 16 \quad (1)$$

$$C_b = -0.169 \times R - 0.331 \times G + 0.5 \times B + 128 \quad (2)$$

$$C_r = 0.5 \times R - 0.419 \times G - 0.081 \times B + 128 \quad (3)$$

B. Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is a well-known mathematical model that has been used in different computer fields like image processing, video processing, face recognition and many other applications. DCT is an orthogonal and reversible transformation used to convert the image from the spatial domain to frequency domain [29]. It is a very useful method in JPEG algorithm, as it converts the image into a domain that makes the compressing process easier for the encoder. The transformation is done to decompose the image into three categories of frequencies which are low, middle and high frequencies [29]. The DCT will discard the redundancy in the image and produce an image that has an important data in the low frequency category and the less important data in a higher frequency [30]. The DCT step will help the encoding process to discard the higher frequency component, because the human eye is less sensitive to the high frequency components. Equation (4) presents the two dimensions DCT (2D-DCT) [30].

$$DCT(u,v) = c(u)c(v)\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)\left[\cos\frac{(2x+1)u}{2n}\right]\left[\cos\frac{(2y+1)v}{2n}\right]$$

The summation limit is from 0 to n. However, since the image will be divided into *8x8* sub images, then n will be 8 in this case. The term f(x, y) is the pixel components, and $c(u)c(v)$ is expressed as shown in (5) [30]:

$$c(u), c(v) = \left\{ \begin{matrix} \frac{1}{\sqrt{n}} \quad for\ u = v = 0 \\ \sqrt{\frac{2}{n}}\ for\ u = v\ \in \{1,2, \ldots n-1\} \end{matrix} \right\} \tag{5}$$

There are many versions of DCT and different algorithms are used to implement DCT. The one dimension DCT (1D-DCT) is calculated by (6).

$$DCT(u) = c(u) \sum_{x\,=\,0}^{n-1} f(x) \left[\cos\frac{(2x+1)u}{2n}\right] \tag{6}$$

From (6), cosine part is pre-calculated and will be multiplied by each pixel. One famous algorithm for calculating the 1D-DCT that is used in many JPEG implementations is proposed in [2]. This algorithm is chosen for calculating the 1D-DCT because it has less number of operations. Fig. 3 shows the block diagram of the 2D-DCT.

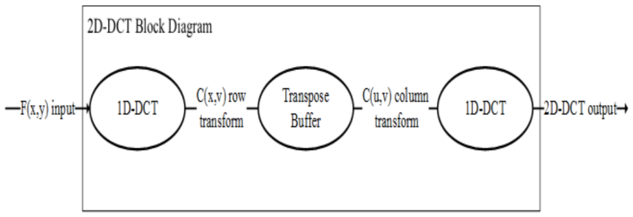

Figure 3. Block Diagram of the DCT Model

The block diagram consists of three blocks; 1D-DCT transformation block, transpose buffer and 1D-DCT transformation block. The first 1D-DCT will take the *8x8* block input and apply the DCT calculation in the row-wise and the output will be stored in the Transpose buffer. The Transpose buffer will transpose the first 1D-DCT coefficient. The second DCT will apply the calculation on column-wise and produce the transformation output.

C. Run Length

After reordering *8x8* blocks into one dimensional array using Zig-Zag reordering algorithm, it is time for Run Length algorithm to do part of the compression process by scanning the whole array produced by the Zig-Zag algorithm to remove the redundancy and making a pairs of (A, B) where A is the number of zeroes preceding the nonzero number B. Fig. 4 shows an example of applying Run Length algorithm on a simple array. In reality, the array is going to be more than 100 times the array presented in the example.

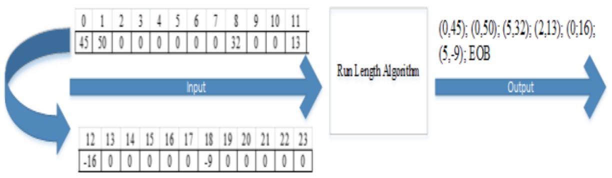

Figure 4. Example of Run Length Algorithm

If the array is having multiple of zeroes consequently until the end of the array, then the scanning process is finished. The output will be ended with a special coded value End of Blocks (EOB) [6].

4. PROPOSED METHODOLOGY

This section describes our detailed HC-FCC-RL architecture for the two main steps in the JPEG algorithm. The proposed solution for the color conversion (FCCA) and the parallel Run length algorithm (P-RL) are explained in the following subsections.

A. FCCA Step

In this step, we are going to enhance the design proposed by [4] by reducing the latency and accelerating the conversion process. The high level block diagram of color conversion step is shown in Fig. 5.

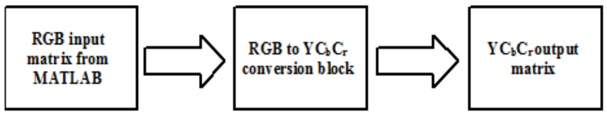
RGB input matrix from MATLAB
RGB to YC$_b$C$_r$ conversion block
YC$_b$C$_r$ output matrix

Figure 5. Color Conversion Block Diagram

The architecture designed by [4] consists of five stages pipeline, where in the first two stages the multiplication is performed using shifters and adders. The authors also applied parallelism by duplicating stages 1 and 2 three times in order to perform the multiplication process of the three components (R, G and B) for one equation at the same time. Stage 3 was designed to add the multiplication results of R and G components and stage 4 was designed to add the results from stage 3 with B component resulting from the multiplication. The final stage used an adder called *INC* to round the final result. After each pipeline stage, numbers of registers have been used to store the results of the stage. The latency of this architecture is five clock cycles and using at most 6 adders and 12 shifters at the same time. Note that the design in [5] used basic equations. In this paper, an approximation technique is applied to accelerate the conversion process. Based on approximation techniques used in [19, 20, 21], we found that it is possible to round the decimal points in (1), (2) and (3) into one decimal point digit. The new three equations for converting color space from RGB to YC_bC_r is shown as follows:

$$Y = 0.3 \times R + 0.6 \times G + 0.1 \times B + 16 \tag{7}$$

$$C_b = -0.2 \times R - 0.3 \times G + 0.5 \times B + 128 \tag{8}$$

$$C_r = 0.5 \times R - 0.4 \times G - 0.1 \times B + 128 \tag{9}$$

FCCA architecture reduced the number of resources used in the design as well as the latency. Tables I, II and III illustrate shifting amount needed for each number from the (7), (8) and (9) to perform the multiplication part from the equations.

TABLE I. SHIFTS AT EACH SHIFTER FOR COMPONENT R

Color Component	value	Binary value	BS1	BS2	BS3	BS4
Y	0.3	0.01001101	r[2]	r[5]	r[6]	r[8]
Cb	0.2	0.00110011	r[3]	r[4]	r[7]	r[8]
Cr	0.5	0.10000000	r[1]	-	-	-

TABLE II. SHIFTS AT EACH SHIFTER FOR COMPONENT G

Color Component	value	Binary value	BS5	BS6	BS7	BS8
Y	0.6	0.10011010	g[1]	g[4]	g[5]	g[7]
Cb	0.3	0.01001101	g[2]	g[5]	g[6]	g[8]
Cr	0.4	0.01100110	g[2]	g[3]	g[6]	g[7]

TABLE III. SHIFTS AT EACH SHIFTER FOR COMPONENT B

Color Component	value	Binary value	BS9	BS 10	BS 11	BS12
Y	0.1	0.00011010	b[4]	b[5]	r[7]	-
Cb	0.5	0.10000000	b[1]	-	-	-
Cr	0.1	0.00011010	b[4]	b[5]	b[7]	-

It can be noticed from Tables I, II and III that we can reduce the number of shifters from 12-barrel shifter to only 10 barrel shifters. Since shifter 4 is shifting an 8-bits number 8 times that means the result is directly zero eight bits, so shifter 4 can be excluded from the design as there is no need to make more effort and take more time to shift eight bits number 8 times. Moreover, shifter 12 is not performing any shifting therefore the result is directly zero eight bits. The number of adders used in the first stage also reduced from 6 adders to only 4 adders since 2 shifters have been deleted from the design. This also caused reduction in the total number of registers needed in the design. The other extra stage, stage 5 is needed to add either 16 or 128 to the final result from stage 4. A multiplexer is needed to make a choice of either adding 16 to the result if calculating *Y* component or adding 128 to the result if calculating C_b or C_r components.

TABLE IV. MULTIPLEXER CHOICES AND RESULTS

Choice "S"	Result "R"
0	16
1	128

Table IV shows the possible selection of the multiplexer and the result from it. In the last stage, the adder INC will round the output by adding the carry to the number in order to get an 8-bit value at the end. Rounding and approximation techniques used in this paper will cause a significant amount of error. The approximate parallel architecture is shown in Fig. 6.

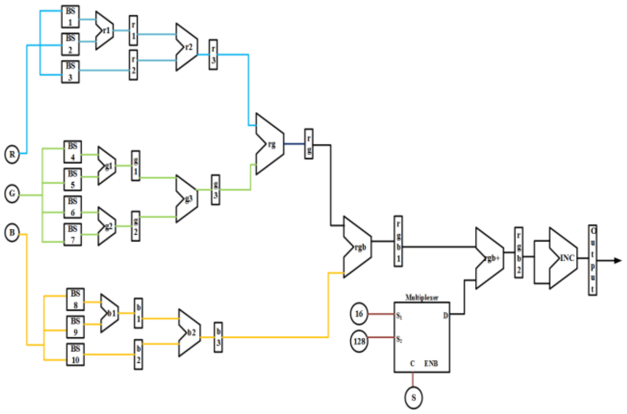

Figure 6. Approximate Parallel Architecture of Color Conversion

B. *P-RL Algorithm*

In JPEG algorithm, Run Length algorithm is used to count the consecutive zero before the non-zero coefficient. This is done by scanning 64 array length to produce the pairs of (A, B) where A is the number of zeroes preceding the nonzero number B. This paper proposed a Parallel Run Length (P-RL) algorithm to fasten the Run Length-encoding step. The idea is to divide the 64-element array into two sub-arrays, and apply the Run Length algorithm on these sub-arrays. The zigzagged array represented by z[i] is divided into two equal arrays; z_1[0..31] and z_2[32..64]. The Run Length algorithm encodes each array. Meanwhile, the Run Length algorithm will count the number of zero preceding the nonzero number then we should consider the number of ending zeroes in the first array which is EOB1, because these zeroes belongs to the second array (the first nonzero number). However, in this design we are going to approximate this approach by discarding the ending zeroes from the first array. In the original design in [24], they also discard EOB of 64 element array where these zeroes belong to the next 64 element array since the image was divided into blocks. The flowchart of P-RL algorithm is presented in Fig. 7. It can be noted that although the proposed algorithm used 2 arrays, the P-RL algorithm can be generalized for any number of sub array sizes 4, 8, 16, etc. In fact, it will fasten the compression step since the algorithm will be applied for each sub array in parallel. However, since we used an approximation method by discarding the EOB at each array, it could affect the accuracy of the image since the decompression will lose some zeros.

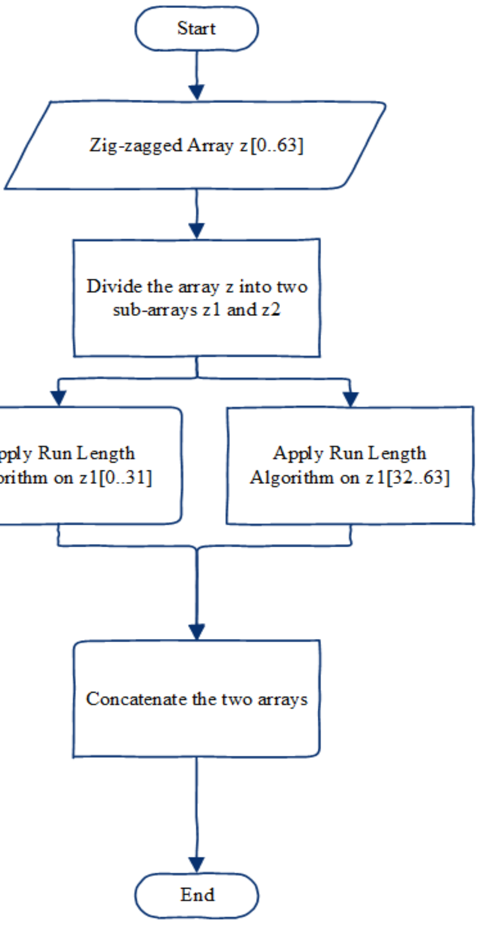
Start
Zig-zagged Array z[0..63]
Divide the array z into two sub-arrays z1 and z2
Apply Run Length Algorithm on z1[0..31]
Apply Run Length Algorithm on z1[32..63]
Concatenate the two arrays
End

Figure 7. P-RL Algorithm

5. Experimental Results

In this section, the experimental results were evaluated after implementing both designs the color space conversion and Run Length algorithm using Verilog, downloaded and synthesized on FPGA Cyclone II family, and analyzed using Quartus.

A. Color Conversion

The color conversion architecture was implemented using Verilog. The main components used in the design are:

- Three 8-bit x16384 word RAM: one RAM for each color component (R,G,B)
- 10 Shifters
- Ripple Carry Adder-Subtractor units
- 8-bit registers
- Multiplexer
- Increment unit: for rounding the result
- Control Unit

The color component was acquired from the MATLAB functions. Each color component coefficient of *8x8* blocks are saved as .mif and used as initialization of the RAM in Quartus. Each color component is processed at one clock cycle. The output of Y, C_r and C_b is available after 6 clock cycle. Table V shows a comparative synthesis result of the FCCA design and the original design in [5].

TABLE V. SYNTHESIS RESULT OF COLOR CONVERSION IN THE PROPOSED DESIGN AND THE ORIGINAL DESIGN

Result	Original Design	Proposed Design
Latency (ns)	14.604	13.930
Frequency (MHz)	68.47	71.79
Estimated total logic elements	360	331
Total Thermal Power Dissipation (mW)	1944.48	861.03
Total Memory bits	393216	344064

From Table V, the FCCA design was able to contribute in improving most design aspects from latency improvement to total memory bits. FCCA design accelerated the color conversion by 4.62%. The logic element saves compared to the original design by 8.06% less resource. FCCA design achieves a low power dissipation compared to the original design in [5] by decrease of 55.72%.

The simulation result is done using waveform. The waveform result was compared with MATLAB function for converting the RGB to YC_bC_r. MATLAB result is approximately the same but with some error rate generated from the approximation method proposed in this architecture. Table VI shows the result of each component from MATLAB, original design in [5] and FCCA design. It can be noticed from Table 6 that YC_bC_r components are slightly differ from MATLAB results since in FCCA design we applied an approximation method for the converting equations. Even the accuracy rate will decrease due to the extra bits being set, the rate in the accuracy reduction is small since the bits being set are from the LSBs and their effect in reducing the accuracy of the result is minimal.

TABLE VI. YC_BC_R RESULT OF MATLAB, ORIGINAL DESIGN AND THE PROPOSED DESIGN

Color Component	MATLAB	Original Design	Proposed Design
Y	155	159	173
Cb	111	239	109
Cr	168	49	177

Table VII illustrates the error rate percentage of the original [3] and FCCA design compared to MATLAB solution. It can be observed that the overall error rate in FCCA design is less than the design proposed in [5].

TABLE VII. THE ERROR RATE OF YCBCR COMPARED TO MATLAB RESULT

Color Component	Percentage error in the original design	Percentage error in the Proposed design
Y	2.6%	11.6%
Cb	115.3%	1.8%
Cr	70.8%	5.3%

B. Parallel Run Length Algorithm

The main component was coded using Verilog. Table VIII shows a comparative synthesis result of the P-RL algorithm design and the Run length algorithm in the original design [25].

TABLE VIII. SYNTHESIS RESULT OF RUN LENGTH IN THE PROPOSED DESIGN AND THE ORIGINAL DESIGN

Result	Original Design	Proposed Design
Latency (ns)	52.619	24.312
Frequency (MHz)	19	41.13
Estimated total logic elements	6,716	3,468
Total Thermal Power Dissipation (mW)	13,092.01	4,861.98

It can be observed from Table VIII that the latency has been reduced by 53.79% and the logic element is saved by 48.36%. P-RL design achieves a low power dissipation compared to the original design in [25] by decrease of 62.86%.

The area decreased because we have reduced the size of run and level output array since the 64 element array input is almost filled with zeroes and partially filled with non-zero numbers. Therefore, the number of bits used for the level is also reduced to half and hence the power is reduced as well.

6. CONCLUSION

This paper proposed fast design architecture for converting the color space from RGB to YC_bC_r. FCCA solution reduced the latency and improved the color conversion step by using approximation method. Approximation helped in reducing the number of resources from 12 shifters to 10 shifters and from 6 adders to 4 adders. FCCA architecture was implemented and simulated in Quartus. The simulation results were compared with a MATLAB function for converting the color from RGB to YC_bC_r, and the results were approximately equal with an error rate of 11.6% for Y component, 1.8% for C_b and 5.3% for C_r. The error rate has low impact on the image quality. The synthesis result showed a latency of 13.930 ns which was 4.62% faster than the other designs in literature. Total logic elements were 331 which is 8.06% less than the compared design. The power dissipation of this architecture was also reduced by 55.72% too.

In this paper, a parallel run length algorithm P-RL was also implemented in Verilog and simulated in Quartus. P-RL design was able to reduce the latency by 53.79% and save the logic gates by 48.36% compared to other designs. The percentage of the power dissipation is also reduced by 62.86%.

The work done in this paper can be extended to include an implementation of faster 2D-DCT architecture to contribute in enhancing the overall design of the JPEG compression. In addition, for the Run length algorithm, since we are enhancing the latency of the design and modifying the design to target colored images, then Run Length algorithm should be applied for three the components (Y, C_b and C_r). For each of three components, there are many arrays of 64 elements, so providing three parallel Run Length algorithm for each component is another future trend.

REFERENCES

[1] Marcus, M. , "JPEG Image compression," Dartmouth Math Department, Dartmouth College, US, 2014.

[2] Kovac, M., Ranganathan, N., "JAGUAR: A fully pipelined VLSI architecture for JPEG image compression standard," Proceedings of the IEEE, 1995, 83, (2), pp. 247-258.

[3] Agostini, L. V., Bampi, S., Silva, I. S. , "High throughput architecture of JPEG compressor for color images targeting FPGAs," 13th IEEE International Conference on Electronics, Circuits and Systems (ICECS'06), Nice, France, 2006, pp. 180-183.

[4] Sa'ed Abed, Mariam AlKandari, Huda AlRasheedi and Imtiaz Ahmad, "Enhanced JPEG Algorithm for Colored Images Based on FPGA Implementation," In Proc. of the 8th IEEE International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO'19), Bahrain, April 15-17, 2019.

[5] Agostini, L. V., Silva, I. S., Bampi, S. , "Parallel color space converters for JPEG image compression," Microelectronics Reliability, 2004, 44, (4), pp. 697-703.

[6] Lin, P. Y. , "Basic image compression algorithm and introduction to JPEG standard," National Taiwan University, 2009, Taipei, Taiwan, ROC, pp. 1-15.

[7] Rawat, S., Verma, A. K. , "Survey paper on image compression techniques," International Research Journal of Engineering and Technology (IRJET),2017, 4, (2), pp. 842-846.

[8] Richter, T. , "JPEG on STEROIDS: Common optimization techniques for JPEG image compression," IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 2016, pp. 61-65.

[9] Alam, L., Dhar, P. K., Hasan, M., Bhuyan, M. G., Daiyan, G. M. , "An improved JPEG image compression algorithm by modifying luminance quantization table," International Journal of Computer Science and Network Security (IJCSNS), 2017, 17, (1), pp. 200-208.

[10] Rippel, O., Bourdev, L. , "Real-Time adaptive image compression," Proceedings of the 34th International Conference on Machine Learning (ICML 2017), arXiv preprint arXiv:1705.05823, Sydney, Australia, 2017, pp. 1-16 .

[11] Kim, C. G., Beak, B. J. , "Fast JPEG color space conversion on shared memory," International Conference on Information Science and Applications (ICISA), Suwon, South Korea, 2013, pp. 1-3.

[12] Kim, C. G., Seo, Y. H. , "Parallel JPEG color conversion on multi-core processor," International Journal of Multimedia and Ubiquitous Engineering, 2016, 11, (2), pp. 9-16.

[13] Shahbahrami, A., Juurlink, B., Vassiliadis, S. , "Accelerating color space conversion using extended subwords and the matrix register file," Eighth IEEE International Symposium on Multimedia (ISM'06), San Diego, CA, USA, 2006, pp. 37-46.

[14] Shen, G., Zhu, L., Li, S., Shum, H. Y., Zhang, Y. Q. , "Accelerating video decoding using GPU," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03), Hong Kong, China, 2003, 4, pp. IV-772-5.

[15] Naumowicz, M., Melosik, M., Katarzynski, P., Handkiewicz, A. , "Automation of CMOS technology migration illustrated by RGB to YCrCb analogue converter," Opto-Electronics Review, 2013, 21, (3), pp. 326-331.

[16] Macieira, R. M., Cambuim, L., Souza, L. L., Oliveira, L. A., Rios, M., Barros, E. , "The design of an image converting and thresholding hardware accelerator," Brazilian Symposium on Computing Systems Engineering (SBESC), Manaus, Brazil, 2014, pp. 103-108.

[17] Bensaali, F., Amira, A., Chandrasekaran, S. , "Power modeling and efficient FPGA implementation of color space conversion," 13th IEEE International Conference on Electronics, Circuits and Systems (ICECS'06), Nice, France, 2006, pp. 164-167.

[18] Sapkal, A.M., Munot, M., Joshi, M.A. , "R'G'B'to Y'CbCr color space conversion using FPGA," IET International Conference onWireless, Mobile and Multimedia Networks, Beijing, China, 2008, pp. 255-258.

[19] Chernov, V., Alander, J., Bochko, V. , "Integerbased accurate conversion between RGB and HSV color spaces," Computers & Electrical Engineering, 2015, 46, pp. 328-337.

[20] Jiang, H., Li, H. Liu, T., Zhang, P., Lu, J. , "A fast method for RGB to YCrCb conversion based on FPGA," 3rd International Conference on Computer Science and Network Technology (ICCSNT), Dalian, China, 2013, pp. 588-591.

[21] Liu, Z. G., Du, S. Y., Yang, Y., Ji, X. H. , "A fast algorithm for color space conversion and rounding error analysis based on fixed-point digital signal processors," Computers & Electrical Engineering, 2014, 40, (4), pp. 1405-1414.

[22] Zhang, X., Li, X., Yang, W., Li, R. , "FPGAbased color space conversion system design and implementation," IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 2016, pp.1-4.

[23] Tu, C., Liang, J., Tran, T. D. , "Adaptive runlength coding," IEEE Signal Processing Letters, 2003, 3, (10), pp. 61-64 .

[24] Gupta, A., Srivastava, M. C., Pandey, S. D., Bhandari, V. , "Modified runlength coding for improved JPEG performance," International Conference on Information and Communication Technology (ICICT'07), 2007, pp. 235-237.

[25] Akhtar, M. B., Qamar, I. , "Open source algorithm for storage area and temporally optimized run length coding for image compression technology used in biomedical imaging," International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan, 2012, pp. 16-21.

[26] Akhtar, M. B., Qureshi, A. M., Qamar, I. , "Optimized run length coding for jpeg image compression used in space research program of IST," International Conference on Computer Networks and Information Technology (ICCNIT), Abbottabad, Pakistan, 2011, pp. 81-85.

[27] Vohra, H., Singh, A. , "Optimal selective count compatible runlength encoding for SOC test data compression," Journal of Electronic Testing, 2016, 32, (6), pp. 735-747.

[28] Li, S. A., Chen, C. Y., Chen, C. H. , "Design of a shift-and-add based hardware accelerator for color space conversion," Journal of Real-Time Image Processing, 2015, 10, (2), pp. 193-206.

[29] Dahmouni, A., Aharrane, N., El Moutaouakil, K., Satori, K. , "Face recognition using local Binary probabilistic pattern (LBPP) and 2DDCT frequency decomposition," 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV), Beni Mellal, Morocco, 2016, pp. 73-77.

[30] Aggrawal, E., Kumar, N. , "High throughput pipelined 2D Discrete cosine transform for video compression," International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, 2014, pp. 702-705.

Sa’ed Abed received his B.Sc. and M.Sc. in Computer Engineering from Jordan University of Science and Technology in 1994 and 1996, respectively. In 2008, he received his Ph.D. in Computer Engineering from Concordia University, Canada. Currently, he is an associate professor in the Department of Computer Engineering at Kuwait University. His research interests include VLSI Design, formal methods and Image Processing.

Imtiaz Ahmad has a B.Eng in electrical engineering, from University of Engineering & Technology, Lahore, Pakistan in 1984, M. Eng. in electrical engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia in 1988, and Ph.D. in computer engineering from Syracuse University, Syracuse, New York,USA, in 1992. His research interests are in design automation of digital systems, parallel and distributed computing. He is currently a professor of the Computer Engineering Department at Kuwait University, Kuwait since 1994.

Mariam AlKandari received the B.S. degree in computer engineering from College of Engineering and Petroleum, Kuwait University, Kuwait, in 2015, and currently the M.Sc. degree in computer engineering from College of Engineering and Petroleum, Kuwait University, Kuwait. She is currently a research assistant in renewable energy program, energy and building center at Kuwait Institute for Scientific Research. Her current research interests include machine learning techniques, artificial Intelligence, optical networks, renewable energy technologies and energy efficient systems.

Huda AlRasheedi received the B.S. degree in computer engineering from College of Engineering and Petroleum, Kuwait University, Kuwait, in 2015, and currently the M.Sc. degree in computer engineering from College of Engineering and Petroleum, Kuwait University, Kuwait. She is currently working as a Contract Engineer for various trends including information technology’s trends at commercial and planning department in Kuwait Oil Company. Her current research interests include machine learning techniques and communication networks mainly optical networks.