**I J C T A**, 9(19) 2016, pp. 9191-9197
© International Science Press

# Image Processing using FPGAs: A Framework

**M. V. Ganeswara Rao, Rajesh K Panakala and A. Mallikarjuna Prasad**

**ABSTRACT**

In technologically advanced world, automation of various processes is very essential in many applications such as security, production, medical, remote sensing etc. In past, computing power and algorithm computational complexity is became barrier to use image processing as a solution in many applications. Recently, many researchers have proposed complex image processing algorithms, which are computationally efficient and similarly high performance computational platforms such as DSP and FPGAs, are also developed. The flexible and concurrency support hardware architecture of FPGAs encourage implementation of image processing algorithms on FPGAs. This paper presents a framework to implement image processing algorithm on FPGAs and digital image colour model conversion using FPGA as a case study. Two sets of images are applied to hardware architecture and results are compared with software implementation.

***Keywords:*** Image Processing, Field Programmable Gate Array (FPGA), Digital Signal processor (DSP)

## I. INTRODUCTION

In recent times, image processing has become central tool for many applications including high performance and large data processing applications such as face detection, face recognition etc. under numerous conditions (lighting variations, orientation, expression etc). Until FPGAs were introduced, Digital Signal Processors are the only computational platforms to implement high performance image processing. Some of the advantages are lager flexibility and less time to prototype make the FPGAs alternate for DSPs [1-3]. The key deference between DSPs and FPGA is their hardware architecture, FPGA does not have fixed hardware architecture and it can be configured according to user requirements using hardware descriptive languages such as VHDL, Verilog HDL etc., In contrast, DSPs have fixed architecture with memory, controller, data path and instructions executed sequentially according software program [4-5].

Image processing algorithms has to process huge amount of pixel data at high speed and for this type of applications FPGAs are exceedingly suitable, because of their parallel processing capability. However, this is not possible with DSPs because of their processing power and data transfer with fixed width data bus such as 8bit, 16bit, 32bit, 64bit etc. Many researchers also reported that the pipeline techniques and more chip area make FPGAs power efficient than DSPs in image processing applications. There are many approaches to implement image processing applications on FPGAs. This paper presents a simple approach, which uses MATLAB and Xilinx core generator to implement image processing algorithms on FPGA [6-7].

This paper has 5 sections, section 2 introduces overall framework for image processing using FPGAs, which includes conversion of .jpg image into .coe file, generation of Block RAM (BRAM) using Xilinx

[*] Department of Electronics and Communication Engineering, Shri Vishnu Engineering College for women, Bhimavaram, India, *E-mail: mgr_ganesh@svecw.edu.in*

[**] Department of Electronics and Communication Engineering, PVP Siddhartha Institute of Technology, Vijayawada, India, *E-mail: rkpanakala@gmail.com*

[***] Department of Electronics and Communication Engineering, JNTU College of Engineering, Kakinada, India, *E-mail: a_malli65@yahoo.com*

core generator, loading .coe into BRAM, interfacing BRAM to processing blocks, writing processed image into .dat file and finally converting .date file to 2D image. In the section 3, colour model conversion from RGB to YCrCb is presented. Results and conclusions are discussed in section 4 and 5 respectively.

## II. FRAMEWORK

Images processing with FPGA provide solutions in many applications. A proposed framework to process images using FPGAs is presented in Figure 1. In stage I mage acquired by the image acquisition hardware is converted into *.coe file to load into BRAM in next stage. The Xilinx core generator is used to generate BRAM and loads the previously generated *.coe file is into BRAM in stage II. After adding BRAM core to project, application hardware can interface with BRAM and pixel data can accessed as illustrated stage III. In this stage, a free running counter acts as address generator for BRAM, and processed pixel data are stored in *.data file. In the final stage IV, *.data file created can be accessed in MATLAB environment to convert back two dimensional image. More details are provided in next sub sections.

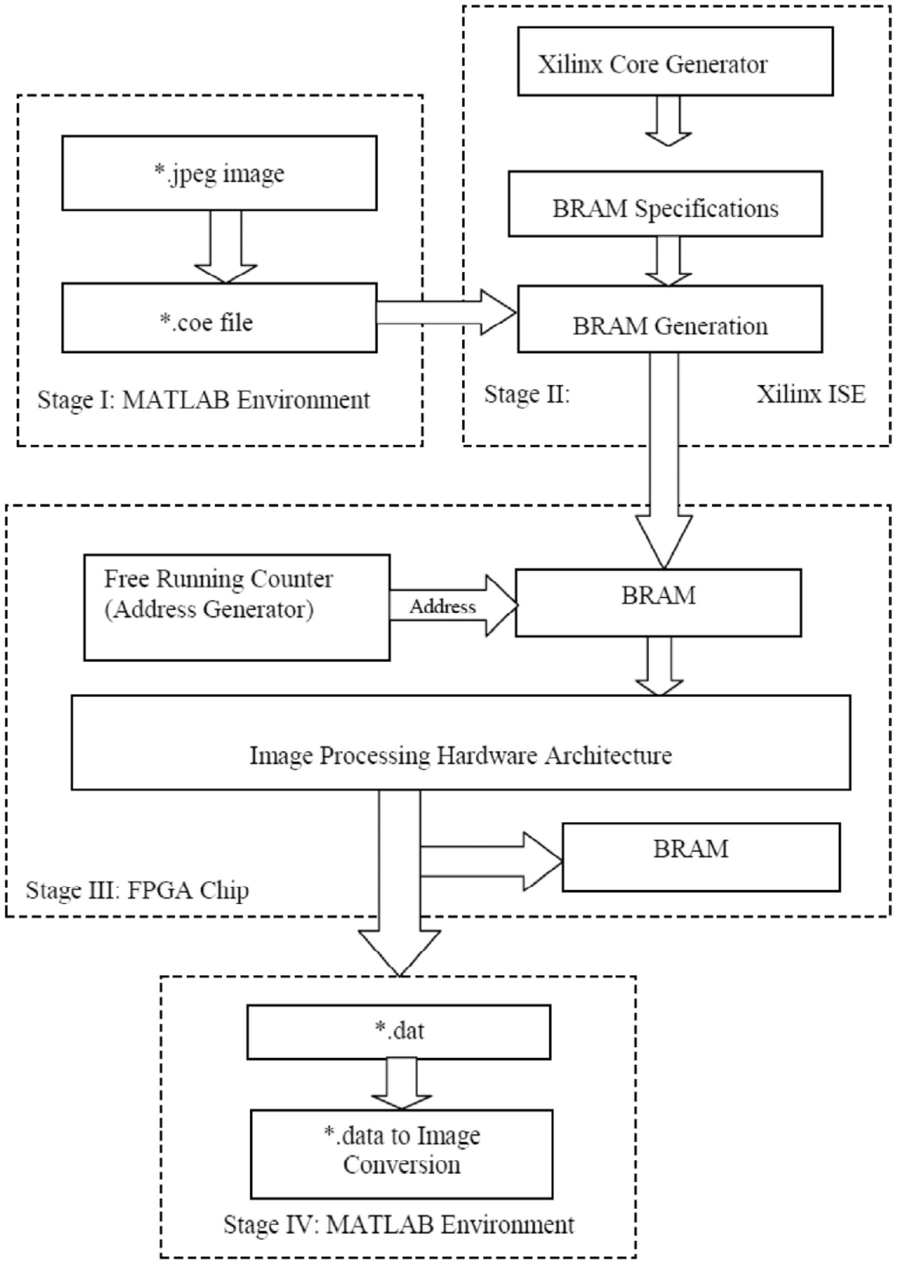

*.jpeg image
*.coe file
Stage I: MATLAB Environment
Xilinx Core Generator
BRAM Specifications
BRAM Generation
Stage II:
Xilinx ISE
Free Running Counter (Address Generator)
Address
BRAM
Image Processing Hardware Architecture
BRAM
Stage III: FPGA Chip
*.dat
*.data to Image Conversion
Stage IV: MATLAB Environment

**Figure 1: Overview of image loading in BRAM**

## 2.1. Conversion of .jpg image to coe file

In many applications, image(s) must be loaded in memory to be accessed by the rest of the image processing hardware. In any memory, data stored is stored in one dimensional, but image to be processed is 2 dimensional. So it is need to convert 2D image data into 1D data and then load them in memory. The pseudo code used to generate 1D image data form image is shown in Figure 2. This code used to convert a 2D RGB image into 1D hex coefficient *.coe file, which is used to load into BRAM. A sample *.coe file generated in shown in figure 3., which has an array of 24 bit pixel data (8 bits for each of the RGB).

```
img = imread(' image location');
[M N P] = size(img)
s=fopen('file location of *.coe', 'w+');
fprintf(s,'%s\n',';  VGA Memory Map ');
fprintf(s,'%s\n',';  .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', M, N);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
cnt = 0;
for r=1:M
            for c=1:N
                        cnt = cnt + 1;
                        R = img(r,c,1);
                        G = img(r,c,2);
                        B = img(r,c,3);
                        Rb = dec2bin(R,8);
                        Gb = dec2bin(G,8);
                        Bb = dec2bin(B,8);
                        Outbyte = [ R G B ];
                        fprintf(s,'%X',R);
                        fprintf(s,'%X',G);
                        fprintf(s,'%X',B);
          if((r== M) && (c == N))
                        fprintf(s,'%c',';');
          else
                        fprintf(s,'%c\n',',');
                        end;end;end
fclose(s);
```

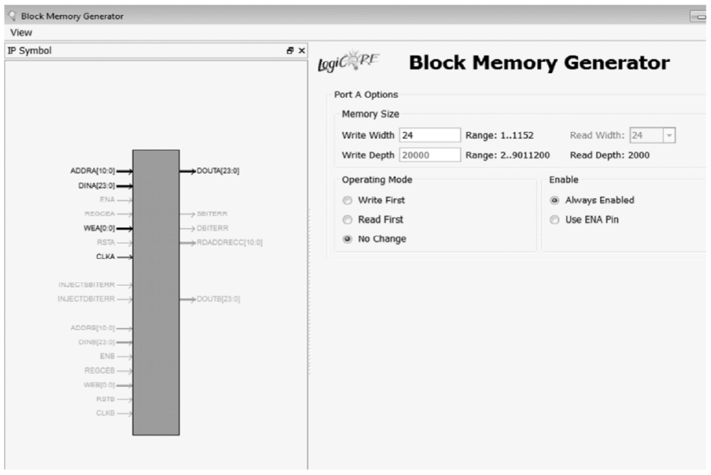
**Figure 2: Pseudo code to convert .jpg image into *.coe**

```
; VGA Memory Map ; .COE file with hex coefficients ; Height: 106, Width:
160memory_initialization_radix=16;memory_initialization_vector=957C7F,957C7F,977C81
,977C81,977C83,977C83,987D82,987D82,9A8083,9A8083,9A8083,9C8285,9A7F84,9C81
86,9A8083,9B8184,9B8285,9B8285,9C8285,9D8386,9D8487,9B8285,9C8285,9F8588,9E8
588,9E8588,9F8588,9F8588,9F8689,A0878A,A1878A,A1878A,A18788,A28889,A1878A,
A2888B,A48789,A28889,A3898C,A38A8D,A6898D,A5888C,A7888E,A78A8E,A78A8E,
A68C8D,A68C8F,A68C8F,A68C8F,A78D90,A78D90,A78D90,A78D90,A78D90,A78D90,
A78D90,A78D90,A88E91,AA8D91,.............................................
```

**Figure 3: Sample .Coe file**

## 2.2. BRAM Generation and Image loading

A block RAM is generated using Xilinx Core generator (see figure 4.) with specifications mentioned in the table 1. This BRAM is loaded with *.coe file created in the previous section.



Port A Options
Memory Size
Write Width 24 Range: 1..1152 Read Width: 24
Write Depth 20000 Range: 2..9011200 Read Depth: 2000
Operating Mode
Write First
Read First
[✓ No Change]
Enable
[✓ Always Enabled]
Use ENA Pin

**Figure 4: Configuring BRAM**

## 2.3. Interface BRAM

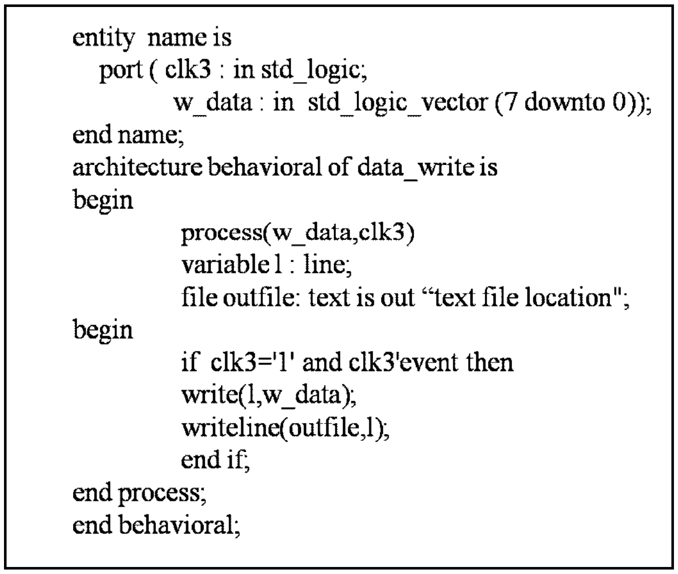
After successfully generating the BRAM with desired specifications, it must be integrated into image processing architecture. An address generation control block will generate sequential address for BROM to read the pixel data from memory. Further these pixels are processed by FPGA hardware and produce processed pixel data. Which will be write into another *.dat file to load the processed image.

## 2.4. Writing to .dat file

The processed pixel data are written into *.dat file to convert image back to 2D digital image. The Pseudo code writes the processed data into *.data file is shown in Figure 6. This code writes 8 bit data into one dimensional array in hex decimal format.

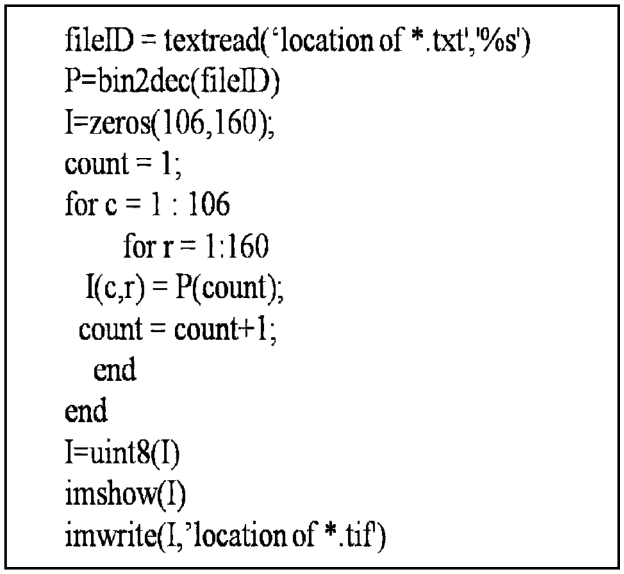
**Table 1**
**Specifications of BRAM**

| *Specification* | *Value* |
|---|---|
| Memory width | 24 bit |
| Address lines | 10 |
| Memory depth (size) | 1MB |
| Enable | Always |
| Port | Single |



```
entity  name is
   port ( clk3 : in std_logic;
             w_data : in  std_logic_vector (7 downto 0));
end name;
architecture behavioral of data_write is
begin
          process(w_data,clk3)
          variable l : line;
          file outfile: text is out “text file location";
begin
          if  clk3='1' and clk3'event then
          write(l,w_data);
          writeline(outfile,l);
          end if;
end process;
end behavioral;
```

**Figure 6: Writing image data to file**



```
fileID = textread(‘location of *.txt','%s')
P=bin2dec(fileID)
I=zeros(106,160);
count = 1;
for c = 1 : 106
      for r = 1:160
  I(c,r) = P(count);
  count = count+1;
    end
end
I=uint8(I)
imshow(I)
imwrite(I,’location of *.tif')
```

**Figure 7: Writing to .dat file**

## 2.5. Converting .dat file to .tif image

The final step in this framework is to convert the one dimensional image data file (*.data file) created in the previous section into two dimensional image with required format such as *.tif. The pseudo code which generates a 2D image with the dimensions of 106X160 is shown in figure 7.

## III. COLOUR MODEL CONVERTION

Numerous colour spaces such as RGB, YCrCb, YIQ CMY, YIS, CIE XYZ, HIS, HIV etc. are utilized to represent colour depending on application. For example image acquisition and display systems use RGB colour model and similarly YIQ and CMY are used in TV broadcasting and Image printing. In this section more emphasis is given to RGB and YCrCb colour models.

## 3.1. RGB colour model

This colour model is a device based colour model, different devices recognize and detect colour in many ways since a deferent physical element detects R B G components differently. In this additive colour model R, G and B components are added to from various colours in the colour space. The RGB colour space represented by a cube is shown Figure 8.

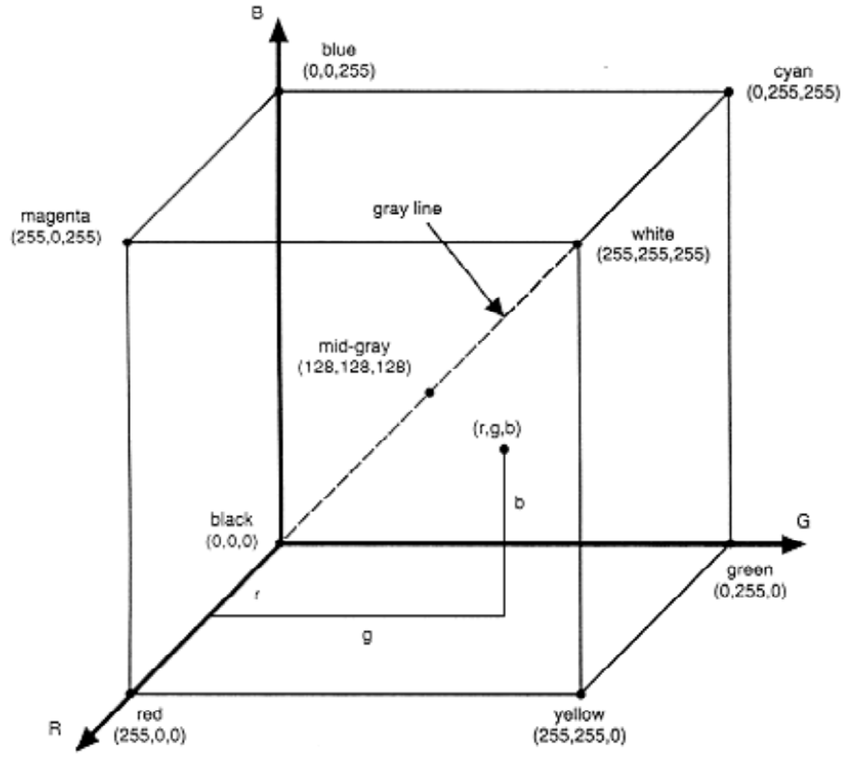


**Figure 8: RGB Colour space**

## 3.2. YCrCb colour model

In this model, Y represents luminance and Cr and Cb are non negation chrominance values, The chrominance values provide easy ways to segment skin fraction of image compared to RGB & HSV model. In this colour space Y, Cr and Cb components are uncorrelated unlike RGB, which is very useful in many Image processing algorithms.

## 3.3. Convertion from RGB to YCrCb

The conversion from RGB to YCrCb has its advantage, Medical research is tells that human eye has dissimilar sensitivity to brightness and colour. The eye has millions of rods (around 120 million), which are more sensitive than cones (around 7 millions). The rods are not very sensitive to colour component, whereas cones are much sensitive to colour component. The fact is that, it is not necessary to keep all information of colour components and thus these colour components are sub-sampled in many applications such as JPEG compression. So it is very useful to convert high correlated RGB colour space to an uncorrelated YCrCb colour space. The formula for converting from RGB to YCrCb model is given below.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.184 & 0.614 & 0.062 \\ -0.101 & -0.339 & 0.439 \\ 0.439 & -0.399 & -0.040 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

## IV. RESULTS

The RGB to YCrCb colour space conversion is successfully implemented using proposed frame work and two set of images are applied to hardware. Figure 8.(a) to (d) shows the RGB representation, Luminance component (Y), Chroma Red component(Cr) and Chroma Blue components (Cb) of the test image 1 respectively. Similarly Figure 9.(a) to (d) shows the RGB representation, Luminance component (Y), Chroma Red component(Cr) and Chroma Blue components (Cb) of the test image 2. The gray level profiles of two test images are shown in figure 10. & figure 11. The experimental results show that, in YCrCb colour brightness and Chroma are highly uncorrelated, It is also observed that results obtained from FPGA implementation are on par with software implementation. The performance parameter MSE, PSNR and SSI are calculated for output images with reference software implementation are shown in table 1.

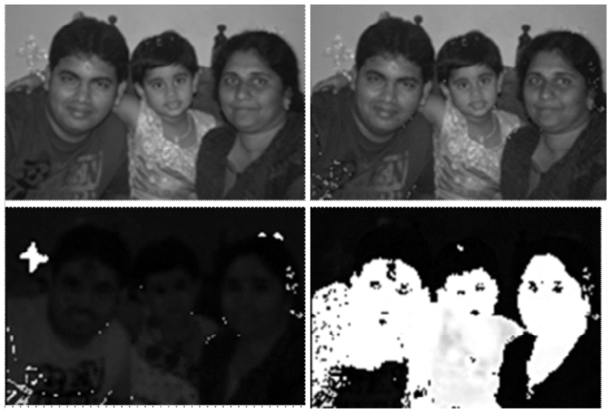


**Figure 8: (a) RGB image of example 1 (b) Y Component (c) Cb Component (d) Cr Component**

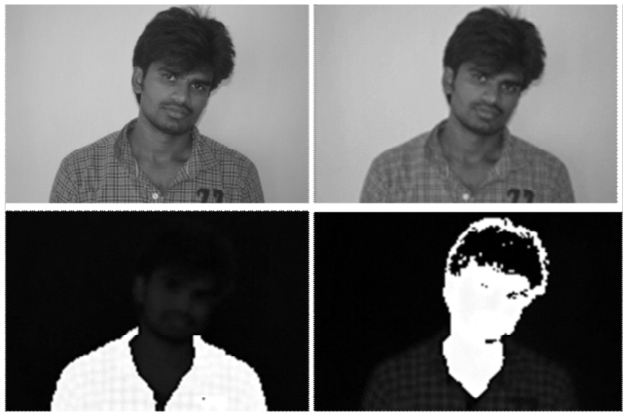


**Figure 9: (a) RGB image of example 2 (b) Y Component (c) Cb Component (d) Cr Component**

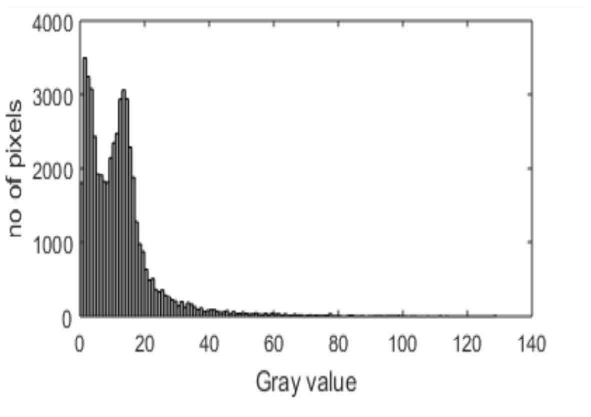


**Figure 10: Histogram of example 1**

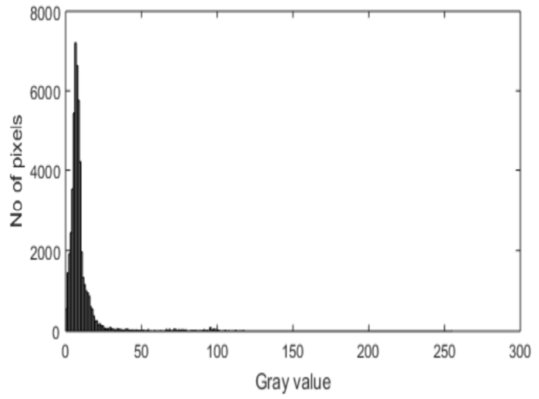


**Figure 11: Histogram of example 2**

**Table 2**
**Performance Parameters**

| | *Example 1* | | | *Example 2* | | |
|---|---|---|---|---|---|---|
| | *Y* | *Cr* | *Cb* | *Y* | *Cr* | *Cb* |
| MSE | 250.3635 | 1.6128e+04 | 1.5684e+04 | 235.6021 | 1.6063e+04 | 1.5193e+04 |
| PSNR | 24.1451 | 6.0550 | 6.1763 | 24.4090 | 6.0724 | 6.3143 |
| SSIM | 0.7433 | 0.1782 | 0.1773 | 0.8121 | 0.1406 | 0.1073 |

## V. CONCLUSIONS

The image processing using FPGA is one of the demanding solutions for many real-time applications. In this paper we have presented a simple image processing frame work, which is not required any additional hardware and software to image acquisition and display. Based on proposed framework, hardware architecture to convert RGB colour to YCrCb colour has been developed and implemented on Xilinx Spartan 3E FPGA. Two sets of image data are applied to hardware, It is observed that results are as expected compared with software implementations. Various performance measures are obtained from results that are shown in table 2.

## REFERENCES

[1] Seunghun Jin, Dogkyun Kim thuy Tuong Nguyen et.al “Design and Implementation of a Pipelined datapath for High-Speed Face Detection Using FPGA,” IEEE Trans. Industrial informatics vol. 8, no 1, pp 158-166, Feb. 2012.

[2] V. P. Javier, R. M. José, S. R. Rafael and T. S. Gerardo, “Acceleration with FPGA for blocks and subblocks edge pattern classification in DCT domain images,” 2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, Montevideo, 2014, pp. 707-712.

[3] M. V. G. Rao, P. R. Kumar and A. M. Prasad, “Implementation of real time image processing system with FPGA and DSP,” 2016 International Conference on Microelectronics, Computing and Communications (MicroCom), Durgapur, India, 2016, pp. 1-4.

[4] S. Mars, A. El Mourabit, A. Moussa, Z. Asrih and I. El Hajjouji, “High-level performance estimation of image processing design using FPGA,” 2016 International Conference on Electrical and Information Technologies (ICEIT), Tangiers, Morocco, 2016, pp. 543-546.

[5] P. K. Dash, S. Pujari and S. Nayak, “Implementation of edge detection using FPGA & model based approach,” Information Communication and Embedded Systems (ICICES), 2014 International Conference on, Chennai, 2014, pp. 1-6.

[6] Melanie Po Leen O,”hardware implementation for Face Detection on Xilinx Virtex-II FPGA using Reversible Component Transformation Colour Space, “ IEEE Proc. Electronic Design, Test and Applications,2005 pp: 41-46

[7] S. C. Chan, H. O. Ngai and K. L. Ho, “A programmable image processing system using FPGA,” Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS ’94, London, 1994, pp. 125-128 vol. 2.