Image Processing to manipulate RGB values using Verilog.

Preprint · December 2021

DOI: 10.13140/RG.2.2.35935.79528/1

2 authors, including:

Ashutosh Lembhe
VIT University
3 PUBLICATIONS **6** CITATIONS

SEE PROFILE

International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 04 | Apr 2022 www.irjet.net p-ISSN: 2395-0072

Image Processing to Manipulate RGB Values Using Verilog

Ashutosh Lembhe[1], Sanjana Vernekar[2]

[1]*UG student, School of Electrical Engineering Vellore Institute of Technology,Chennai. India*
[2]*UG student, School of Electrical Engineering Vellore Institute of Technology,Chennai. India*

Abstract - *Image processing is a widely used application in today's world. Be it a photo taken from phone or camera. There is a process that goes into it to make the image stabilized. Python is mostly used for image processing because of its easy coding and libraries used for different operations. We have implemented the image processing in the Verilog code. In this paper we have showed how we converted an image in a hex file using MATLAB and then used the hex file in Verilog coding to perform some image processing operations. We have Xilinx to write our code and perform the operations. There will be an output image after the code is compiled and run. We will be performing basic three operation for an image that will be in a hex file as an input. The Three basic operations are inverting of the image, threshold operation of the image, image brightness reduction. This paper also aims to show how to process an image using Verilog from reading an input bitmap image in Verilog, processing and writing the processed result to an output bitmap image in Verilog.*

Key Words: **Image Processing, Verilog HDL, FPGA, RGB Values, Image Manipulation, Colour processing.**

1.INTRODUCTION

Image processing is a technique for applying operations on an image in order to improve it or extract relevant information from it. It's a sort of signal processing in which the input is an image and the output is either that image or its characteristics/features. Image processing basically includes the following three steps: 1. Importing Image hex file via image acquisition tool. 2. Analysing and manipulating the image. 3. Output in which image can be a altered image or report that is based on image analysis. By using Verilog hardware descriptive language we can directly code at hardware level and enjoy its hardware portability advantage. Verilog allows us to read hex file for manipulation of image easily. By using Verilog hardware engineer can easily analyse the circuit synthesis by using one of the Xilinx's property to create circuit with use of Verilog code. Because Verilog syntax is constantly linked to a hardware structure, timing information about a hypothetical hardware implementation is also available, allowing for specialised speed enhancements.

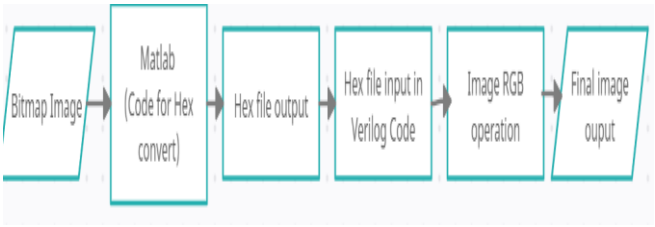

Fig -1: Image Processing Block Diagram

2. MATLAB CODING FOR HEX FILE

Images cannot be read directly by Verilog. The image must be transformed from bitmap to hexadecimal format before it can be read in Verilog. To convert a bitmap image to a hex file, we use MATLAB the input image needs to be converted into 768x512 pixels, and the image.hex file contains the bitmap image's R, G, and B data. Below is the photo of code used for converting bitmap image into a hex file.

```
%Embedded SyStems Project (Ashutosh Lembhe,Sanjana vernerkar)
close all;
clear all;
clc;

b=imread('input.jpg'); % 24-bit BMP image RGB888

k=1;
for i=512:-1:1 % image is written from the last row to the first row
for j=1:768
a(k)=b(i,j,1);
a(k+1)=b(i,j,2);
a(k+2)=b(i,j,3);
k=k+3;
end
end
fid = fopen('input.hex', 'wt');
fprintf(fid, '%x\n', a);
disp('Text file write done');disp(' ');
fclose(fid);
```

Fig -2: Matlab Hex code file.

The image is named as input for MATLAB code to identify it in the same folder. The image is send through some iterations in for loop. Notice the loop is from 512 to 786 same as the pixels of the image. In the iterations the images RGB data is extracted and a file name input.hex is given out by using the commad fopen. The input hex file only contains RGB vectors for each pixel of the input image. The next section covers the Verilog part of the research.

3.VERILOG CODING FOR IMAGE READING OPERATION.

Image processing operations are now implemented in Verilog. The Verilog coding is divided into two parts, first is implementation where parameters for operations such as brightness, invert, threshold are declared. The second is simulation part where the above operations are defined and coded in details. Xilinx allows the use to simulate on virtual board in its environment. The board chosen for virtual simulation is VIRTEX 6 FPGA board. We will be performing three types of operations on the image. Brightness operation, Inverting operation and Threshold operation. Below is the image of operation definition file.

```
1  /************************* ***********************************************/
2  /************************* Definition file ******************************/
3  /************************* ***********************************************/
4  `define INPUTFILENAME       "input.hex" // Input file name
5  `define OUTPUTFILENAME      "output.bmp"     // Output file name
6
7  // Choose the operation of code by delete // in the beginning of the selected line
8  `define BRIGHTNESS_OPERATION
9  `define INVERT_OPERATION
10 `define THRESHOLD_OPERATION
11
```

Fig -3: Parameters file from where operations will be defined

To read the image hexadecimal data file, Verilog uses the command $readmemh. After reading the image.hex file, the RGB picture data is kept in memory and read out for processing. Below is the image of image read module. The image width and height are stored in their respective variables. An array is also created for storing RGB data after going through series of operations. Two arrays for each color has been created. One for storing even data and another for storing odd data. In the "parameter.v" file, the image processing operation is chosen. Simply switch the comment line to change the processing operation.

```
4  `include "parameter.v"                // Include definition file
5  module image_read
6  #(
7    parameter WIDTH    = 768,              // Image width
8              HEIGHT   = 512,              // Image height
9              INFILE   = "input.hex",  // image file
10             START_UP_DELAY = 100,        // Delay during start up time
11             HSYNC_DELAY = 160,           // Delay between HSYNC pulses
12             VALUE= 100,                  // value for Brightness operation
13             THRESHOLD= 90,               // Threshold value for Threshold operation
14             SIGN=0                       // Sign value using for brightness operati
15                                          // SIGN = 0: Brightness subtraction
16                                          // SIGN = 1: Brightness addition
17 )
18 (
19    input HCLK,                           // clock
20    input HRESETn,                        // Reset (active low)
21    output VSYNC,                      // Vertical synchronous pulse
22    // This signal is often a way to indicate that one entire image is transmitted.
23    // Just create and is not used, will be used once a video or many images are tran
24    output reg HSYNC,                     // Horizontal synchronous pulse
25    // An HSYNC indicates that one line of the image is transmitted.
26    // Used to be a horizontal synchronous signals for writing bmp file.
27     output reg [7:0]  DATA_R0,           // 8 bit Red data (even)
28     output reg [7:0]  DATA_G0,           // 8 bit Green data (even)
29     output reg [7:0]  DATA_B0,           // 8 bit Blue data (even)
30     output reg [7:0]  DATA_R1,           // 8 bit Red  data (odd)
31     output reg [7:0]  DATA_G1,           // 8 bit Green data (odd)
32     output reg [7:0]  DATA_B1,           // 8 bit Blue data (odd)
33    // Process and transmit 2 pixels in parallel to make the process faster, you can
34    output           ctrl_done           // Done flag
```

Fig -4: Image Read Module.

3.1 Brightness Operation Verilog Code.

This module changes the brightness of the image by adding or removing a fixed value from the pixel value. The following code adds and subtracts a constant value from the pixel values of a picture. In this, two types of operations are coded. One is brightness addition and other is brightness subtraction.

```
243       /*************************************/
244       /*    BRIGHTNESS ADDITION OPERATION */
245       /*************************************/
246       if(SIGN == 1) begin
247       // R0
248       tempR0 = org_R[WIDTH * row + col    ] + VALUE;
249       if (tempR0 > 255)
250          DATA_R0 = 255;
251       else
252          DATA_R0 = org_R[WIDTH * row + col    ] + VALUE;
253       // R1
254       tempR1 = org_R[WIDTH * row + col+1    ] + VALUE;
255       if (tempR1 > 255)
256          DATA_R1 = 255;
257       else
258          DATA_R1 = org_R[WIDTH * row + col+1    ] + VALUE;
259       // G0
260       tempG0 = org_G[WIDTH * row + col    ] + VALUE;
261       if (tempG0 > 255)
262          DATA_G0 = 255;
263       else
264          DATA_G0 = org_G[WIDTH * row + col    ] + VALUE;
265       tempG1 = org_G[WIDTH * row + col+1    ] + VALUE;
266       if (tempG1 > 255)
267          DATA_G1 = 255;
268       else
269          DATA_G1 = org_G[WIDTH * row + col+1    ] + VALUE;
270       // B
271       tempB0 = org_B[WIDTH * row + col    ] + VALUE;
272       if (tempB0 > 255)
273          DATA_B0 = 255;
274       else
275          DATA_B0 = org_B[WIDTH * row + col    ] + VALUE;
276       tempB1 = org_B[WIDTH * row + col+1    ] + VALUE;
277       if (tempB1 > 255)
278          DATA_B1 = 255;
279       else
280          DATA_B1 = org_B[WIDTH * row + col+1    ] + VALUE;
281    end
```

Fig -5: Brightness Addition Verilog code.

When sign variable is declared one in image read module the operation performed on the image is brightness addition. The temp variables perform an calculations using the above formulas and check of the value exceeds the limit of 255, if yes then it assigns the RGB data the value of 255.

```
283     /*************************************/
284     /* BRIGHTNESS SUBTRACTION OPERATION */
285     /*************************************/
286        // R0
287        tempR0 = org_R[WIDTH * row + col    ] - VALUE;
288        if (tempR0 < 0)
289           DATA_R0 = 0;
290        else
291           DATA_R0 = org_R[WIDTH * row + col    ] - VALUE;
292        // R1
293        tempR1 = org_R[WIDTH * row + col+1    ] - VALUE;
294        if (tempR1 < 0)
295           DATA_R1 = 0;
296        else
297           DATA_R1 = org_R[WIDTH * row + col+1    ] - VALUE;
298        // G0
299        tempG0 = org_G[WIDTH * row + col    ] - VALUE;
300        if (tempG0 < 0)
301           DATA_G0 = 0;
302        else
303           DATA_G0 = org_G[WIDTH * row + col    ] - VALUE;
304        tempG1 = org_G[WIDTH * row + col+1    ] - VALUE;
305        if (tempG1 < 0)
306           DATA_G1 = 0;
307        else
308           DATA_G1 = org_G[WIDTH * row + col+1    ] - VALUE;
309        // B
310        tempB0 = org_B[WIDTH * row + col    ] - VALUE;
311        if (tempB0 < 0)
312           DATA_B0 = 0;
313        else
314           DATA_B0 = org_B[WIDTH * row + col    ] - VALUE;
315        tempB1 = org_B[WIDTH * row + col+1    ] - VALUE;
316        if (tempB1 < 0)
317           DATA_B1 = 0;
318        else
319           DATA_B1 = org_B[WIDTH * row + col+1    ] - VALUE;
320     end
```

Fig -6: Brightness subtraction Verilog Code

If the variable sign is assigned 0 then the operational code will be as shown in fig 6. If the value calculated from the formula comes to be less than 0 then the variables for for RGB data are assigned to 0. The formula involves using the width, row and col values.

3.2 Inverting Operation Verilog Code.

This module inverts an image by inverting the bits of the grayscale pixel value. The RGB pixel values must be equalised to convert a coloured image to a grayscale image, which is done by averaging the three colour components.

```
323     /*************************************/
324     /*    INVERT OPERATION          */
325     /*************************************/
326     `ifdef INVERT_OPERATION
327        value2 = (org_B[WIDTH * row + col    ] + org_R[WIDTH * row + col    ] +org_G[WIDTH * row + col    ])/3;
328        DATA_R0=255-value2;
329        DATA_G0=255-value2;
330        DATA_B0=255-value2;
331        value4 = (org_B[WIDTH * row + col+1    ] + org_R[WIDTH * row + col+1    ] +org_G[WIDTH * row + col+1    ])/3;
332        DATA_R1=255-value4;
333        DATA_G1=255-value4;
334        DATA_B1=255-value4;
335     `endif
```

Fig -7: Inverting Operation Verilog code.

From the fig it is clearly shown that the value2 and value4 variables are used to store the average values after calculating the average value using RGB. Value2 is used for calculation of even data RGB values by subtracting it from 255 and storing it in the respective variables. Value4 variable is used for calculating odd data of RGB values. Note that the formula for calculating value4 and value2 is slightly different.

3.4 Threshold Operation Verilog Code.

This Verilog code is used to perform the threshold operation, which consists of setting the pixel value over a threshold value to 255 and the pixel value below the threshold value to 0. To perform the threshold operation, we can use the Verilog testing code shown below. As shown in the code above both lower and upper limit is set. The if condition checks if the value calculated through the formula given is above threshold or not, if yes then all odd and even data values are set to 255.

```
336     /*************************************/
337     /********THRESHOLD OPERATION  ********/
338     /*************************************/
339     `ifdef THRESHOLD_OPERATION
340
341     value = (org_R[WIDTH * row + col    ]+org_G[WIDTH * row + col    ]+org_B[WIDTH * row + col    ])/3;
342     if(value > THRESHOLD) begin
343        DATA_R0=255;
344        DATA_G0=255;
345        DATA_B0=255;
346     end
347     else begin
348        DATA_R0=0;
349        DATA_G0=0;
350        DATA_B0=0;
351     end
352     value1 = (org_R[WIDTH * row + col+1    ]+org_G[WIDTH * row + col+1    ]+org_B[WIDTH * row + col+1    ])/3;
353     if(value1 > THRESHOLD) begin
354        DATA_R1=255;
355        DATA_G1=255;
356        DATA_B1=255;
357     end
358     else begin
359        DATA_R1=0;
360        DATA_G1=0;
361        DATA_B1=0;
362     end
363     `endif
364
365   end
366 end
367
368 endmodule
```

Fig -8: Threshold Operation Verilog Code.

3.2 Verilog Coding for Image Write Operation.

The "parameter.v" file which was mentioned earlier also specifies the input and output file directories and names. After processing the image, the processed data must be written to an output image for verification purposes.

```
4   module image_write
5   #(parameter WIDTH    = 768,                    // Image width
6             HEIGHT   = 512,                    // Image height
7             INFILE   = "output.bmp",           // Output image
8             BMP_HEADER_NUM = 54                // Header for bmp image
9   )
10  (
11     input HCLK,                                // Clock
12     input HRESETn,                             // Reset active low
13     input hsync,                               // Hsync pulse
14      input [7:0]  DATA_WRITE_R0,               // Red 8-bit data (odd)
15      input [7:0]  DATA_WRITE_G0,               // Green 8-bit data (odd)
16      input [7:0]  DATA_WRITE_B0,               // Blue 8-bit data (odd)
17      input [7:0]  DATA_WRITE_R1,               // Red 8-bit data (even)
18      input [7:0]  DATA_WRITE_G1,               // Green 8-bit data (even)
19      input [7:0]  DATA_WRITE_B1,               // Blue 8-bit data (even)
20     output   reg    Write_Done
21  );
22  integer BMP_header [0 : BMP_HEADER_NUM - 1];   // BMP header
23  reg [7:0] out_BMP  [0 : WIDTH*HEIGHT*3 - 1];  // Temporary memory for image
24  reg [18:0] data_count;                         // Counting data
25  wire done;                                     // done flag
26  // counting variables
27  integer i;
28  integer k, l, m;
29  integer fd;
```

Fig -9: Image write module in Verilog.

The output file image is stored in the variable infile. An array is also been created for temporary image storage.

```
30  //-----------------------------------------------------------//
31  //-------Header data for bmp image-----------------------------//
32  //-----------------------------------------------------------//
33  // Windows BMP files begin with a 54-byte header:
34
35  initial begin
36     BMP_header[ 0] = 66;BMP_header[28] =24;
37     BMP_header[ 1] = 77;BMP_header[29] = 0;
38     BMP_header[ 2] = 54;BMP_header[30] = 0;
39     BMP_header[ 3] =  0;BMP_header[31] = 0;
40     BMP_header[ 4] = 18;BMP_header[32] = 0;
41     BMP_header[ 5] =  0;BMP_header[33] = 0;
42     BMP_header[ 6] =  0;BMP_header[34] = 0;
43     BMP_header[ 7] =  0;BMP_header[35] = 0;
44     BMP_header[ 8] =  0;BMP_header[36] = 0;
45     BMP_header[ 9] =  0;BMP_header[37] = 0;
46     BMP_header[10] = 54;BMP_header[38] = 0;
47     BMP_header[11] =  0;BMP_header[39] = 0;
48     BMP_header[12] =  0;BMP_header[40] = 0;
49     BMP_header[13] =  0;BMP_header[41] = 0;
50     BMP_header[14] = 40;BMP_header[42] = 0;
51     BMP_header[15] =  0;BMP_header[43] = 0;
52     BMP_header[16] =  0;BMP_header[44] = 0;
53     BMP_header[17] =  0;BMP_header[45] = 0;
54     BMP_header[18] =  0;BMP_header[46] = 0;
55     BMP_header[19] =  3;BMP_header[47] = 0;
56     BMP_header[20] =  0;BMP_header[48] = 0;
57     BMP_header[21] =  0;BMP_header[49] = 0;
58     BMP_header[22] =  0;BMP_header[50] = 0;
59     BMP_header[23] =  2;BMP_header[51] = 0;
60     BMP_header[24] =  0;BMP_header[52] = 0;
61     BMP_header[25] =  0;BMP_header[53] = 0;
62     BMP_header[26] =  1;
63     BMP_header[27] =  0;
```

Fig -10: Header data for BMP image.

The above is the code for a 54 byte header in Verilog. The bitmap image's header info is extremely significant. The written picture may not be shown appropriately if there is no header data.

```
// row and column counting for temporary memory of image
always@(posedge HCLK, negedge HRESETn) begin
    if(!HRESETn) begin
        l <= 0;
        m <= 0;
    end else begin
        if(hsync) begin
            if(m == WIDTH/2-1) begin
                m <= 0;
                l <= l + 1; // count to obtain row index of the out_BMP temporary memory to save image data
            end else begin
                m <= m + 1; // count to obtain column index of the out_BMP temporary memory to save image data
            end
        end
    end
end
```

Fig -11: Verilog Code for storing of temporary memory of image.

The above iterative loop is used for counting rows and column index for temporary memory to save image data.

4. RESULTS

Here are the simulation results produced by applying the operations outlined in Verilog HDL to an input image. The image given as an input is of the size 768x512. This image will first be given as an input to MATLAB. Then its hex file is generated which will be given as an input to the Verilog coding in Xilinx.

Fig -12: Original Input Bitmap image

Run the simulation, close the simulation and open the output image for checking the result. Followings are the output images which are processed by the selected operations in parameter.v.

Fig -13: Reduced Brightness Image output

We can see that the brightness has been reduced significantly. The code shown in fig 6 has done its role and subtracted the brightness from its original image and lowered it.

Fig-14: Inverted Image output.

For inverting operation the value must have been subtracted from 255 to reach this contrast level for the output image.

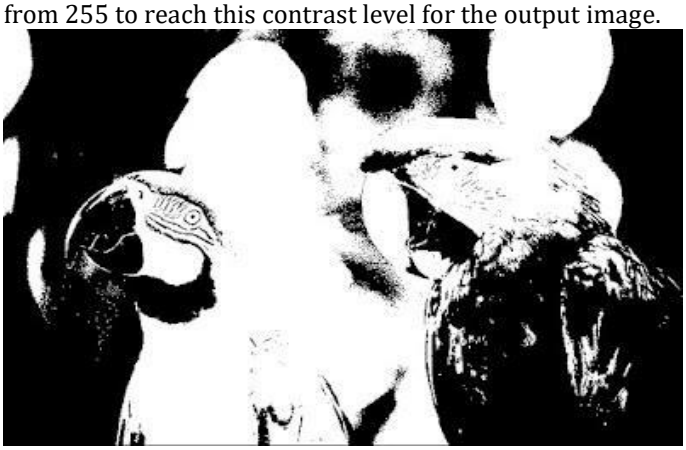

Fig -15: Threshold Image output.

Black and white is obtained after threshold operations. The threshold value obtained here was around 120 after the code was executed.

4. CONCLUSION.

The results obtained were promising. Verilog has the ability to manipulate RGB values without the need of any external library. With help of iterative loops and mathematical formulas the image processing operations were implemented successfully. It can be concluded that Verilog can also be used for image processing operations and can be directly used for coding at hardware level.

REFERENCES

[1] John C. Russ - "Image Processing Handbook (sixthedition)", CRC Press, pp. 270-331, 2011.

[2] Raman Maini, H. Aggarwal - "A Comprehensive Review of Image enhancement Techniques", Journal of Computing, vol. 2, issue 3, ISSN 2151-9617, pp. 269-300, 2010.

[3] Wilhelm Burger, Mark J. Burge -"Principles of Digital Image Processing – Fundamental Techniques", Undergraduate Topics in Computer ScienceK. Elissa, "Title of paper if known," unpublished.

[4] Daggu Venkateshwar Rao, Shruti Patil, Naveen Anne Babu and V. Muthukumar - "Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based HardwareDescriptive Languages", International Journal of Theoretical and Applied Computer Sciences, Volume 1, Number 1, pp. 9–34, 2006

[5] Wilhelm Burger, Mark J. Burge - "Digital Image Processing – An Algorithmic Introduction Using Java", e-ISBN 978-3-540-30941-3, Springer, 2008

[6] Priyanka S. Chikkali, K. Prabhushetty - "FPGA based Image Edge Detector and Segmentation", International Journal of Advanced Engineering Sciences and Technologies, vol. no. 9, issue no.2, pp187- 192, ISSN 2230-7818, 2011

[7] Raman Maini, H. Aggarwal - "A Comprehensive Review of Image enhancement Techniques", Journal of Computing, vol. 2, issue 3, ISSN 2151-9617, pp. 269-300, 2010

[8] Nick Efford - "Digital Image Processing – A Practical Introduction Using Java", pp. 103-132, 2000.

[9] Iuliana CHIUCHISAN, Marius CERLINCA, Alin-Dan POTORAC, Adrian GRAUR "Stefan cel Mare" University of Suceava str.Universitatii nr.13, RO-720229 Suceava 20

[10] A. Zuloaga, J.L. Martin, U. Bidarte, J.A. Ezquerra - "VHDL test bench for digital image processing systems using a new image format", ECSI, 2007.

[11] Xilinx. Inc.Xilinx 4 Software Manuals : Libraries Guide. 20002

[12] R.C.Gonzalez and R.E.Woods. "Digital Image Processing", prentice hall, ISBN 0-13-094659, pp. 1-142, 2002.

[13] A. DeHon, "The Density Advantage of Reconfigurable Computing," IEEE Computer, vol. 33, pp. 41-49, 2000

[14] Varsha S.Surwase and S.N.Pawar "Vlsi implementation of image processing algorithms on fpga", IJEEE volume 3,number 3 (2010), pp. 139-145.

[15] S Patel, VV Dwivedi, YP Kosta, "A Parametric Characterization and Comparative Study of Okumura and Hata Propagation-loss-prediction Models for Wireless Environment", International Journal of Electronic Engineering Research, Vol. 2, No. 4

[16] Sagar Patel, Rahul Patel and Jaymin Bhalani, "Performance Analysis & Implementation of different Modulation Techniques in Almouti MIMO Scheme with Rayleigh Channel", International Conference on Recent Trends in computing and Communication Engineering, April 2013

[17] Jigar Kumar, Arpita Patel and Sagar Patel, "Multiuser – MIMO Broadcast Channel Techniques", InternationalResearch Journal of Engineering and Technology, Vol. 3, No. 2, Feb. 2016.