Implementation of Reduced Area Image Enhancement Methods Using Verilog HDL

Kiruthiga E[1], Kiruthiga A[1], Sindhu S[2], Sakthi Arun S[3]

[1]*UG Student, Dept.of. E.C.E, Meenakshi Sundararajan Engineering College, Chennai, Tamil Nadu, India*
[2]*Assistant Professor, Dept. of. E.C.E, Meenakshi Sundararajan Engineering College,Chennai, Tamil Nadu, India*
[3]*VLSI Design Engineer of Missile Ingeniator, Tamil Nadu, India*

---***---

Abstract - *The demand of Image Processing methods traditionally implemented on a digital processing software such as MATLAB is increasing widely to get high performance.*

In this project we implemented four basic operations of Image Enhancement i.e. threshold, contrast, brightness, invert to manipulate the RGB values of every pixel of the image to improve the human interpretation of image.

To perform the above mentioned operations we have implemented Image Enhancement on FPGA (Field Programmable Gate Array) using Verilog HDL.

Implementation in HDL (Hardware Description Language) is quite different from implementation in MATLAB mainly because of the parallel nature of the HDLs. The system is implemented on FPGA, which is modern programmable logic device, i.e. we can program almost any digital function in it.

Key Words: **FPGA, Verilog, Image Enhancement**

1. INTRODUCTION

Image enhancement is to process an image in order that end result is greater suitable than original image and Image enhancement aims at improving the interpretability or perception of information in image for human viewers, or providing better input for other automated image processing techniques.

Image enhancement techniques help in improving the visibility of any portion or feature of the image suppressing the information in other portion or features.

There are many Hardware Description Languages (HDLs) available to help the engineers describe the circuit both logically and functionally so that they can simulate and properly calculate the performances with the help of personalized test environment and clock cycle.

Since the HDL syntax is always related to a hardware structure, the timing information of the potential hardware implementation is also available allowing specific speed optimizations. Above all, with the use of HDLs it means that we can enjoy hardware portability and on-the-fly reprogrammability. But here the bigger challenge is to implement the validated algorithms into a non-programming language as hardware description languages are. Also, the input and output RGB files need to be constructed accordingly to match the binary content permitted into the hardware simulators.

The main goal of any improvement methodology is simply too acquire a lot of appropriate result compared with the first as is from the purpose of read of a selected application

1.1 BLOCK DIAGRAM

The Verilog language has the power to browse or write files from a storage setting. This feature create it potential to significantly style the test benches to browse the test information from device, generate the stimulant signals to the Verilog check module and write back the results to the device. We need image but image file will be in .bmp file extension so this file is not readable to Xilinx Software. Moreover, here image convert to hex file using MATLAB.

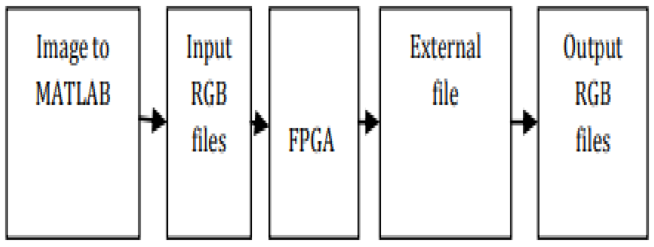


Fig 1. BLOCK DIAGRAM OF SYSTEM

The project is implemented on FPGA using Verilog, which is Hardware Description Language. The code written in Verilog describes the behavior of the desired hardware. The Hardware Description Language, Verilog HDL was developed to carry out readings and writings of files with ASCII characters and it does not allow to process bitmap or jpeg files. For that reason, it is necessary to represent binary information with ASCII characters in the hex format. Hex characters are quickly and easily converted to binary format by Verilog HDL. In order to resolve this problem it was defined a new image format to be used with the test bench described. The hex-file contain only information about RGB vector for each pixel of the input image and does not contain information about image dimensions or similar. The data from hex-file was applied as stimulus to the point operations blocks described in Verilog language. The result was obtained in another external file and we create an application described in Visual Studio to show the modified output image and to compare with the original input image.

In this project we use four basic operations of image enhancement threshold, contrast, brightness, invert to manipulate the RGB values of every pixel of the image to improve the human interpretation of image.

1.2 IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps

1)Importing the image via image acquisition tools.

2)Analyzing and manipulating the image.

3)Output in which result can be altered image or report that is based on image analysis.

1.3 TYPES OF IMAGE ENHANCEMENT

Image enhancement techniques can be classified into two categories:

❖ **SPATIAL DOMAIN TECHNIQUE**: Spatial domain technique refers to enhancement of image based on operations performed directly on pixels of the image.

Types of spatial domain techniques:

➢ **Range operations**:

✓ Point operations.

✓ Neighborhood operations.

➢ Domain operations.

❖ **FREQUENCY DOMAIN TECHNIQUE:** It achieves enhancement through the use of mathematical transforms such as Fourier transforms and it is straightforward. The idea of blurring an image by reducing its high frequency components, or sharpening an image by increasing the magnitude of its high frequency components is intuitively easy to understand.

❖ **POINT OPERATION**

Point operations are nothing but simplest image filters, where the new value of a pixel is only determine by the original value of that single pixel alone. As mentioned, point operations perform modification of pixel values without changing the size, geometry or local structure of the image. The pixel value is given by **a = I (u, v)**, which depends exclusively on the **previous value** a = I (u, v) at the same position. This pixel value is independent from any other pixel value, including any of its neighbouring pixels. To map the original pixel values to the new values a function f (a) is used, a' ← f(a) I' (u, v) ← f (I (u, v)) for each image point with a spatial coordinate (u, v).

1.4 INTRODUCTION TO VERILOG

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip–flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits Verilog is like any other hardware description language. It permits the designers to design the designs in either Bottom-up or Top-down methodology.

- **Bottom-Up Design:** The traditional method of electronic design is bottom-up. Each design is performed at the gate-level using the standards gates. This design gives a way to design new structural, hierarchical design methods.
- **Top-Down Design:** It allows early testing, easy change of different technologies, and structured system design and offers many other benefits.
- Words that have special meaning in Verilog are called the Verilog keywords. For example **assign**, **case**, **while**, **wire, reg, and, or, nand,** and **module**. They should not be used as identifiers. Verilog keywords also include compiler directives, and system tasks and functions. The basic lexical tokens used by the Verilog HDL are similar to those in C Programming Language. Verilog is **case sensitive**. All the key words are in **lower case.**

2.HARDWARE

FPGA enables you to program product features, adapt to new standards, and reconfigure hardware for specific applications even after the product has been installed in the field hence the term '*field-programmable*'. Whereas '*gate arrays*' refer to two-dimensional array of logic gates present in its architecture. The system is supposed to be implemented on a BASYS 3 FPGA development board. It consists of three main parts:

- ✓ ***Configurable Logic Blocks*** — which implement logic functions.
- ✓ ***Programmable Interconnects*** — which implement routing.
- ✓ ***Programmable I/O Blocks*** — which connect with external components.

Logic blocks implement the logical functions required by the design and consist of various components such as transistor pairs, look-up tables (LUTs), flip flops, and multiplexers.

The Basys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. With its high-capacity FPGA (Xilinx part number XC7A35T1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.

2.1 FEATURES OF BASYS 3 FPGA BOARD

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. It features include:

- ✓ 33,280 logic cells in 5200 slices (each slice contains four
- ✓ 6-input LUTs and 8 flip-flops) 1,800 Kbits of fast block RAM
- ✓ Five clock management tiles, each with a phase-locked
- ✓ loop (PLL) 90 DSP slices
- ✓ Internal clock speeds exceeding 450MHz
- ✓ On-chip analog-to-digital converter (XADC)

2.2 PORT AND PERIPHERAL OF FPGA BOARD

The Basys 3 also offers an improved collection of ports and peripherals, including:

- ✓ 5 user pushbuttons
- ✓ 16 user LEDs
- ✓ 16 user switches
- ✓ Pmod for XADC signals
- ✓ Three Pmod ports
- ✓ 4-digit 7-segment display
- ✓ Serial Flash
- ✓ USB-UART Bridge
- ✓ 12-bit VGA output
- ✓ Digilent USB-JTAG port for
- ✓ FPGA programming and communication
- ✓ USB HID Host for mice, keyboards and memory sticks

2.3 WORKING

The Basys 3 works with Xilinx's new high-performance Vivado Design Suite. Vivado includes many new tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives. The System Edition includes an on-chip logic analyzer, high-level synthesis tool, other cutting-edge tools, and the free WebPACK version allows Basys 3 designs to be created at no additional cost.

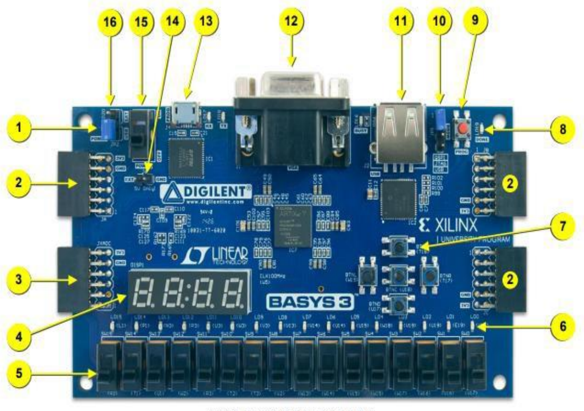


Figure 1. Basys 3 FPGA board with callouts.

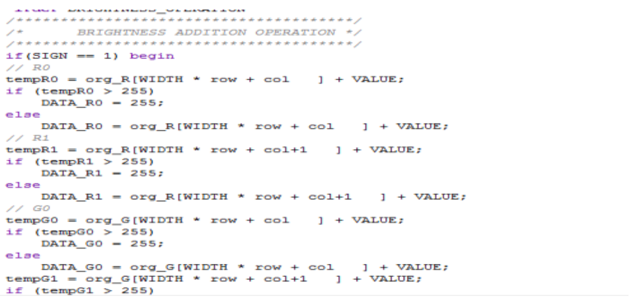
Fig2. BASYS 3 FPGA BOARD

TABLE-1 COMPONENT DESCRIPTION OF BASYS 3FPGA

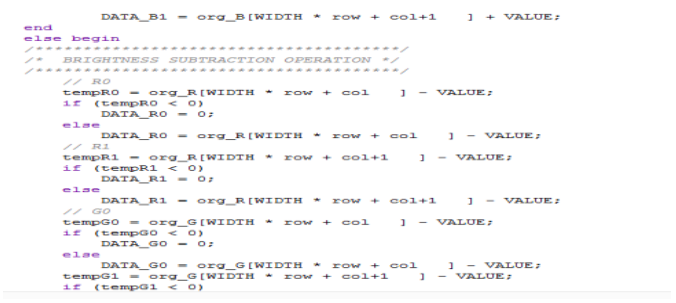
Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

2.4 IMAGE ENHANCEMENT METHODS

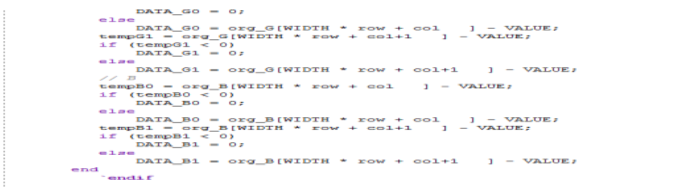
Brightness operation: - A dark region in an image may become brighter after the point operation and the common used point operation are increasing and decreasing of brightness. If an operator takes each pixel value and adds a constant number to it, then this point operation increases the brightness of the image and similar subtraction operation reduces the brightness.



```
/**************************************/
/*      BRIGHTNESS ADDITION OPERATION */
/**************************************/
if(SIGN == 1) begin
// R0
tempR0 = org_R[WIDTH * row + col    ] + VALUE;
if (tempR0 > 255)
    DATA_R0 = 255;
else
    DATA_R0 = org_R[WIDTH * row + col    ] + VALUE;
// R1
tempR1 = org_R[WIDTH * row + col+1   ] + VALUE;
if (tempR1 > 255)
    DATA_R1 = 255;
else
    DATA_R1 = org_R[WIDTH * row + col+1   ] + VALUE;
// G0
tempG0 = org_G[WIDTH * row + col    ] + VALUE;
if (tempG0 > 255)
    DATA_G0 = 255;
else
    DATA_G0 = org_G[WIDTH * row + col    ] + VALUE;
tempG1 = org_G[WIDTH * row + col+1   ] + VALUE;
if (tempG1 > 255)
```



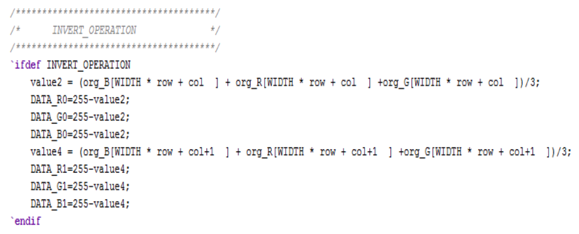
```
            DATA_B1 = org_B[WIDTH * row + col+1    ] + VALUE;
    end
    else begin
    /**************************************/
    /*  BRIGHTNESS SUBTRACTION OPERATION */
    /**************************************/
        // R0
        tempR0 = org_R[WIDTH * row + col    ] - VALUE;
        if (tempR0 < 0)
            DATA_R0 = 0;
        else
            DATA_R0 = org_R[WIDTH * row + col    ] - VALUE;
        // R1
        tempR1 = org_R[WIDTH * row + col+1    ] - VALUE;
        if (tempR1 < 0)
            DATA_R1 = 0;
        else
            DATA_R1 = org_R[WIDTH * row + col+1    ] - VALUE;
        // G0
        tempG0 = org_G[WIDTH * row + col    ] - VALUE;
        if (tempG0 < 0)
            DATA_G0 = 0;
        else
            DATA_G0 = org_G[WIDTH * row + col    ] - VALUE;
        tempG1 = org_G[WIDTH * row + col+1    ] - VALUE;
        if (tempG1 < 0)
```



```
            DATA_G0 = 0;
        else
            DATA_G0 = org_G[WIDTH * row + col    ] - VALUE;
        tempG1 = org_G[WIDTH * row + col+1    ] - VALUE;
        if (tempG1 < 0)
            DATA_G1 = 0;
        else
            DATA_G1 = org_G[WIDTH * row + col+1    ] - VALUE;
        // B
        tempB0 = org_B[WIDTH * row + col    ] - VALUE;
        if (tempB0 < 0)
            DATA_B0 = 0;
        else
            DATA_B0 = org_B[WIDTH * row + col    ] - VALUE;
        tempB1 = org_B[WIDTH * row + col+1    ] - VALUE;
        if (tempB1 < 0)
            DATA_B1 = 0;
        else
            DATA_B1 = org_B[WIDTH * row + col+1    ] - VALUE;
     end
        `endif
```

INVERT OPERATION:

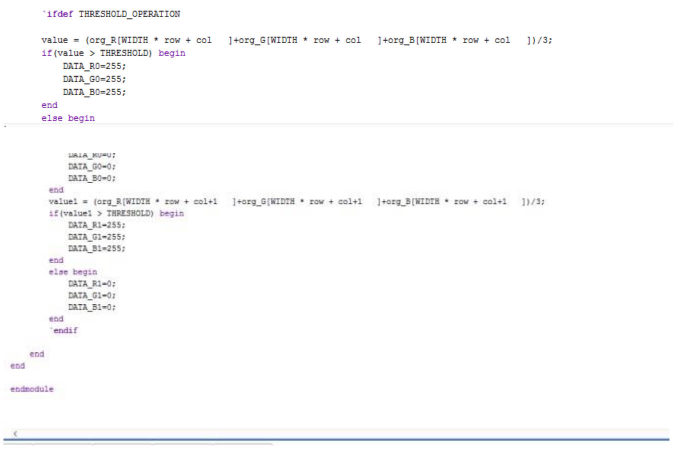
Inverting an intensity image is a simple point operation that reverses the ordering of pixel values (by multiplying with−1) and adds a constant value to map the result to the admissible range again.RGB pixel values must be equalized and it is done by taking the average of the three color components.



```
/**************************************/
/*     INVERT_OPERATION              */
/**************************************/
`ifdef INVERT_OPERATION
   value2 = (org_B[WIDTH * row + col  ] + org_R[WIDTH * row + col  ] +org_G[WIDTH * row + col  ])/3;
   DATA_R0=255-value2;
   DATA_G0=255-value2;
   DATA_B0=255-value2;
   value4 = (org_B[WIDTH * row + col+1  ] + org_R[WIDTH * row + col+1  ] +org_G[WIDTH * row + col+1  ])/3;
   DATA_R1=255-value4;
   DATA_G1=255-value4;
   DATA_B1=255-value4;
`endif
```

THRESHOLD OPERATION:

Threshold operations are particularly interesting for segmentation in the process of isolating an object of interest from its background. Threshold an image means transforming all pixels in two values only. Set the pixel above a threshold value to 255 and below it to 0.



```
`ifdef THRESHOLD_OPERATION

value = (org_R[WIDTH * row + col   ]+org_G[WIDTH * row + col   ]+org_B[WIDTH * row + col   ])/3;
if(value > THRESHOLD) begin
    DATA_R0=255;
    DATA_G0=255;
    DATA_B0=255;
end
else begin

     DATA_R0=0;
     DATA_G0=0;
     DATA_B0=0;
  end
  value1 = (org_R[WIDTH * row + col+1   ]+org_G[WIDTH * row + col+1   ]+org_B[WIDTH * row + col+1   ])/3;
  if(value1 > THRESHOLD) begin
     DATA_R1=255;
     DATA_G1=255;
     DATA_B1=255;
  end
  else begin
     DATA_R1=0;
     DATA_G1=0;
     DATA_B1=0;
  end
  `endif

    end
end

endmodule
```

2.5 5 ADVANTAGES

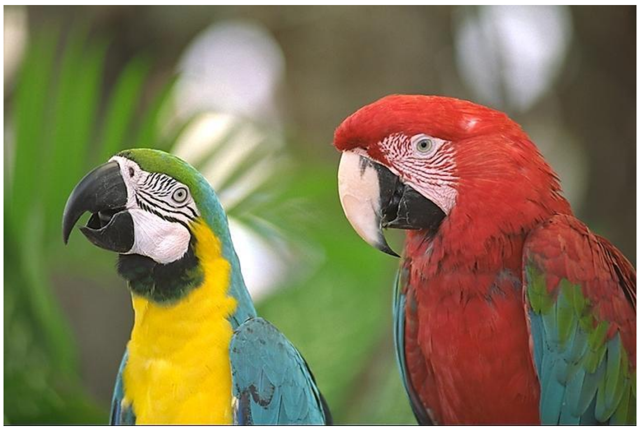
1)No processing or fixing chemicals are needed to take and process digital images.
2)Unrecognizable features can be made prominent.
3)Images can be smoothened.
4)It allows robots to have vision.
5)It allows industries to remove defective products from the production line**.**
6) It requires very small time; starting from design process to functional chip.
7)No physical manufacturing steps are involved in it.

2.6 RESULTS

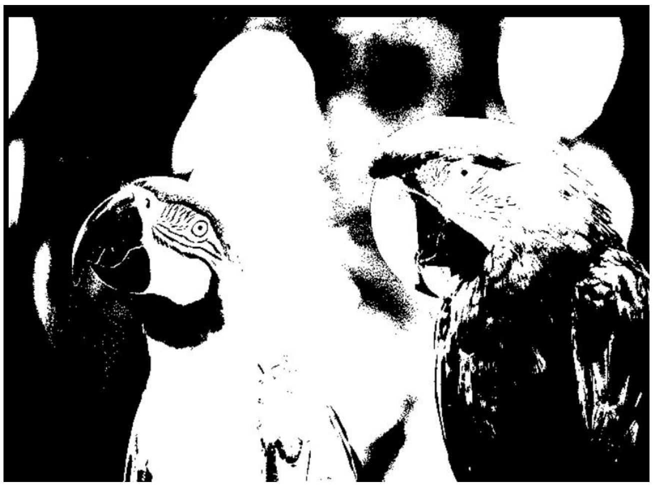
The simulation results obtained after applying the operations described using Verilog HDL to an input image are shown here

THRESHOLD OPERATION

INPUT IMAGE



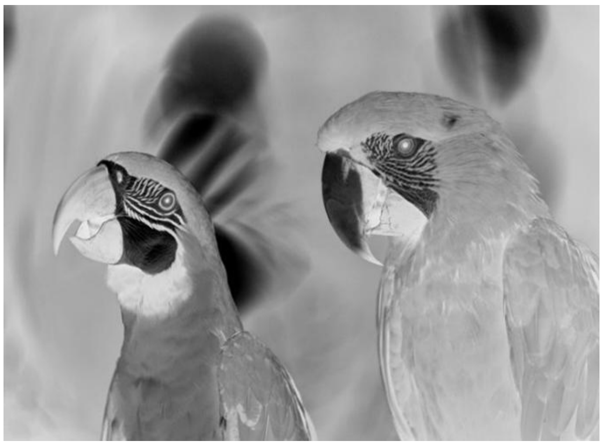
OUTPUT IMAGE



INVERSION OPERATION

INPUT IMAGE

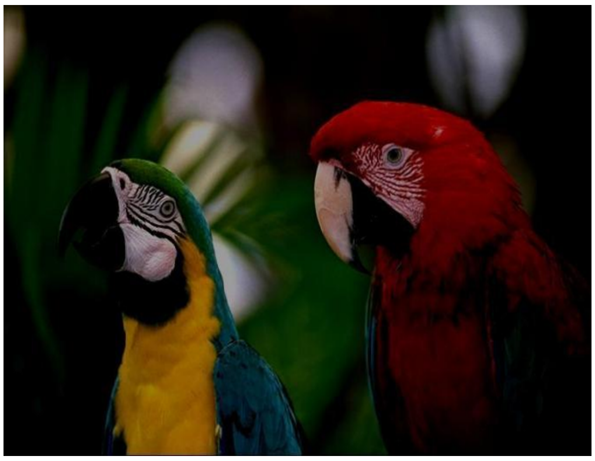
OUTPUT IMAGE



BRIGTNESS OPERATION

INPUT IMAGE

OUTPUT IMAGE



3. CONCLUSIONS

Image Processing has been designed and implemented using Verilog HDL and simulated using ISim from Xilinx ISE Design Suite 14.3 and synthesized using Xilinx XST with lot of image processing creeping into the PDA and other hand held equipment, there is an urge for providing a specialized Image Processor. This work gives an insight into the most generalized architecture that can be customized for other image processing applications too. Though, the most fundamental point operations on the image are discussed, the idea may be carried forward for designing filtering applications also. The major challenge in this work is to choose a proper FPGA for prototyping, since the memory buffer needs enormous memory, the crucial aspect is to choose such an FPGA which has enough RAM, FIFO resources.

REFERENCES

[1] A Comprehensive Survey on Implementation of Image Processing Algorithms using FPGA Madhava Prabhu S, and Seema Verma, 5th IEEE International Conference on Recent Advances and Innovations in Engineering- ICRAIE 2020 (IEEE Record#51050)

[2]Implementation of An Automatic Image Enhancement Algorithm for Contrast Stretching on FPGA Zhaochao Shi1 , Hui Wu ,Weiming Mao ,Jing Wang, Chao Zhang1, People's Republic of China , 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC) | 978-1-

7281-5244-8/20/$31.00 ©2020 IEEE | /DOI: 10.1109/ITAIC49862.2020.9338896

[3] CORDIC-Based High-Speed VLSI Architecture of Transform Model Estimation for Real-Time Imaging Anirban Chakraborty , Member, IEEE, and Ayan Banerjee IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 29, NO. 1, JANUARY 2021

[4] An Extremely Pipelined FPGA Implementation of a Lossy Hyperspectral Image Compression Algorithm Daniel Báscones , Carlos González , and Daniel Mozos, IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, VOL. 58, NO. 10, OCTOBER 2020

[5] FPGA-based Edge Inferencing for Fall Detection Kishore, Christopher Paolini, Mahasweta Sarkar, 2020 IEEE Global Humanitarian Technology Conference (GHTC)

[6] FPGA Design of Real-Time MDFD System Using High Level Synthesis CHULIANG WEI1,2, RULIN CHEN1,2, AND QIN XIN 3, Received May 31, 2019, accepted June 14, 2019, date of publication June 21,2019, date of current version July 11, 2019.

[7] FPGA Implementation of Image Enhancement using Verilog HDL Prof. Narayan A. Badiger[1] , Miss.Jayamma Muragod[2] , Miss.Poornima Pattar[2] ,Miss. Priyanka Gundlur[2] , Miss.Savita Bhadr, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 06 | June 2020 www