International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 04 | Apr 2024 www.irjet.net p-ISSN: 2395-0072

Data Acquisition Using Camera Serial Interface

Diksha Sagar[1], Dr. Jeeru Dinesh Reddy[2]

[1]*PG Student, Dept. of Electronics and Communication Engineering, BMS College of Engineering, Bengaluru, India*
[2]*Professor, Dept. of Electronics and Communication Engineering, BMS College of Engineering, Bengaluru, India*

Abstract - *Reaching high integration, high speed, high resolution, and high reliability is the aim of image preprocessing systems. Image processing systems are widely employed in both the military and commercial industries. Image processing technology-based object detection has drawn a lot of interest in the military because of its noncontact capabilities, capacity to hide, and ability to avoid interference. In the business sector, it is widely used in industrial detection systems and machine vision. There are three main kinds of image processing systems that are used to implement digital image processing techniques. The three main chips that comprise each system are the FPGA (Field Programmable Gate Array), DSP (Digital Signal Processor chip), and ASIC (Application Specific Integrated Circuit). In this work, we created an image processing system based on FPGA. The system can take samples from the data stream.*

Key Words: FPGA, Image acquisition system, Image processing, Xilinx Vivado HLS, MATLAB, Verilog.

1. INTRODUCTION

Artificial intelligence, pattern recognition, and signal processing are all engaged in the study of picture collection and processing, which has been a popular area of research. This technology is mainly used in automotive electronics, consumer electronics, security monitoring, national defence, and other fields of 3D projection. The increasing popularity of digital image processing technology is inseparable from the perfecting of processing systems. In the image processing system, the key technology is real-time image acquisition and processing. Meanwhile, the speed and quality of image acquisition directly affect the system [10]. The advancement of large-scale integrated circuit fabrication technologies, particularly FPGA, and microelectronics has produced innovative concepts and techniques for enhancing the functionality of image processing systems in recent years. The image processing system based on FPGA is widely utilized in the image preprocessing area because of the vast amount of data and rapid processing speed required for low-level picture preprocessing. The need for video information has increased as a result of the advancements made in multimedia technologies in recent years. In any case, the significance of picture processing and capture is growing. 8-bit standard RGB (sRGB) pictures, which are commonly compressed using the JPEG standard, make up the great majority of images used in computer vision and image processing applications. The processes of almost all imaging applications support JPEG and sRGB picture formats. These days, the majority of cameras enable the saving of photos in RAW format, which is an unprocessed, minimally compressed picture format that captures the reaction from the camera sensor. More benefits of RAW over sRGB include a broader color gamut, a higher dynamic range (usually 12–14 bits), and a linear response to scene radiance. For several computer vision applications, including white balance, photometric stereo, picture restoration, and more, RAW is preferred. Photographers also prefer RAW because it gives them more versatility when manipulating images in post-processing. The serial process of compression of images starts with the conversion of an RGB image into YIQ if required. The resulting image is then transformed by DCT. In the quantization, unnecessary data about the image is eliminated from size and quality. Encoding of the image is done for protection by changing the names of the values of the quantized image by passing the image into the channel encoder. The image is involved in inverse quantization, which retrieves the lost data from the image. Passing through the inverse transformation phase forms the original image. Image compression is a technique that lowers the amount of data needed to communicate with an advanced image. And eliminating the redundant workers will provide this.

Operation of the JPEG Encoder core:

1.1 Color Space Transformation

The first operation of the JPEG Encoder core is converting the red, green, and blue pixel values to their corresponding Luminance and Chrominance (Y, Cb, and Cr) values. The RGB2YCBCR module is where this procedure is carried out. The operation is based on the following formulas:

$$Y = .299 * Red + .587 * Green + .114 * Blue$$

$$Cb = -.1687 * Red + -.3313 * Green + .5 * Blue + 128$$

$$Cr = .5 * Red + -.4187 * Green + -.0813 * Blue + 128$$

Fixed point multiplications are used to carry out these tasks. All of the constant values in the above 3x3 matrix are multiplied by 2^14 (16384). One clock cycle is used for the multiplications, and the next clock cycle is used to add the sum of the products. In order to obtain a quick

clock frequency during synthesis, this is done. Next, rather than really dividing the sums, the sums are divided by 2^14, which is accomplished by throwing away the 14 LSBs of the sum values. When rounding, the 13th LSB is examined, and if it is 1, 1 is added to the total.

1.2 Discrete Cosine Transform

The next step after calculating the Y, Cb, and Cr values is performing the Discrete Cosine Transform (DCT). This is commonly referred to as a 2D DCT. The actual formula is the following:

DY = T * Y * inv(T)

T is the DCT matrix. Y is the matrix of Y values for the 8x8 image block. DY is the resultant matrix after the 2D DCT. The DCT needs to be performed separately on the Y, Cb, and Cr values for each block. The DCT of the Y values is performed in the y_dct module. The cb_dct and cr_dct modules contain the DCT of the Cb and Cr values. Since the cb_dct and cr_dct modules are nearly identical, I will just discuss the y_dct module in this section.

Now you may have noticed that I have not centered the Y, Cb, and Cr values on 0 in the previous stage. To do that, I would have subtracted 128 from the final Y value, and not added the 128 to the final Cb and Cr values. To perform the DCT, the values of Y, Cb, and Cr need to be centered around 0 and in the range –128 to 127. However, I perform a few tricks in the DCT module that allow me to keep the Y, Cb, and Cr values in the range from 0-255. I do this because it makes the implementation of the DCT easier.

The DCT matrix, or T as I call it, is multiplied by the constant value 16384 or 2^14. The rows of the T matrix are orthonormal (the entries in each row add up to 0), except for the first row. Because the rows 2-8 are orthonormal, it does not matter that I have not centered the Y values on 0. I perform the multiplication of the T rows by the Y columns of data, and the extra 128 in each of the Y values is cancelled out by the orthonormal T rows. The first row, however, is not orthonormal - it has a constant value of .3536, or 5793 after it is multiplied by 2^14. Since I have not centered Y by 0, the extra 128 in each value will result in an extra 128*8*5793 = 5932032 in the final sum. So to make up for the fact that I have not centered the Y values on 0, I subtract 5932032 from the result of the first row of T multiplied by each of the 8 columns of the Y matrix. If you think about this, it means I have to perform a total of 8 subtractions for an 8x8 matrix of Y values. If I had subtracted 128 from each Y value before the DCT module, I would have needed to perform a total of 64 subtractions.

After multiplying the T matrix by the Y matrix, the resulting matrix is multiplied by the inverse of the T matrix. In order to maximize the clock frequency for the design, this procedure is carried out in the code. The result is the code may look overly confusing, but I tried many different schemes before settling on the one used in the code. I would simulate the code, make sure it functioned, and then synthesize to see what clock speed I could get. I kept doing this until I eventually found the optimal clock speed, which was about 300 MHz. I targeted a Xilinx Virtex 5 FPGA to achieve this speed.

1.3 Quantization

The next step is fairly straightforward. The module y_quantizer comes next for the Y values. The Cb and Cr values go through the cb_quantizer and cr_quantizer modules. The 64 quantization values are stored in the parameters Q1_1 through Q8_8. I used finals values of 1 for my core, but you could change these values to any quantization you want. I simulated different quantization values during testing, and I settled on values of 1, corresponding to Q = 100, because this stressed my code the most and I was trying to break the core in my final testing. The core did not break, it worked, but I left the quantization values as they were.

As in previous stages, I avoid performing actual division as this would be an unnecessary and burdensome calculation to perform. I create other parameters QQ1_1 through QQ8_8, and each value is 4096 divided by Q1_1 through Q8_8. For example, QQ1_1 = 4096 / Q1_1. This division is performed when the code is compiled, so it doesn't require division in the FPGA.

The input values are multiplied by their corresponding parameter values, QQ1_1 through QQ8_8. Then, the bottom 12 bits are chopped off the product. This gets rid of the 4096, or 2^12, that was used to create the parameters QQ1_1 through QQ8_8. The final values are rounded based on the value in the 11th LSB.

1.4 Huffman Encoding

The module y_huff performs the Huffman encoding of the quantized Y values coming out of the y_quantizer module. The modules cb_huff and cr_huff perform the Huffman encoding for the Cb and Cr values. The module yd_q_h combines the y_dct, y_quantizer, and y_huff modules. The values from y_quantizer are transposed (rows swapped with columns) as they are input to the y_huff module. This is done so that the inputs of each 8x8 block to the top module, jpeg_top, can be written in the traditional left to right order. Peforming the DCT requires matrix multiplication, and the rows of the T matrix are multiplied by the columns of the Y matrix. So the Y values would need to be entered in a column format, from top to bottom, to implement this. Instead, the Y values can be entered in the traditional row format, from left to right, and then by transposing the values as they pass between the y_quantizer and y_huff modules, the proper organization of Y values is regained.

The Huffman table can be changed by changing the values in this module – the specific lines of code with the Huffman table are lines 1407-1930. However, the core does not allow the Huffman table to be changed on the fly. You will have to recompile the code to change the Huffman table. You should create a full Huffman table, even if you have a small image file and do not expect to use all of the Huffman codes. The calculations in this core may differ slightly from how you do your calculations, and if you use a Huffman table without all of the possible values defined, the core may need a Huffman code that is not stored in the RAM, and the result will be an incorrect bitstream output.

The DC component is calculated first, then the AC components are calculated in zigzag order. The output from the y_huff module is a 32-bits signal containing the Huffman codes and amplitudes for the Y values.

2. PROPOSED METHODOLOGY

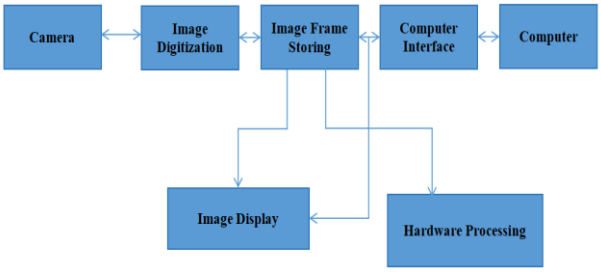

Fig 1. Block Diagram of the Entire Camera Serial Interface[10]

The procedure for taking and processing a picture is shown in the block diagram. The process starts with the image capture phase, which gathers visual data. The acquired image is next subjected to Digital Conversion, which converts it into a format that may be altered digitally. The digitally converted image finds temporary refuge in Temporary Storage before progressing to the Computer Interface. This interface facilitates communication between the computer system and imaging devices. The crucial stage of image processing comes next, where the digital picture is altered by applying adjustments, improvements, or other changes. At this juncture, two divergent paths emerge:

1) Display Output: In this case, consumers may see the finished product by viewing the processed image on a screen.

2) Hardware Processing: As an alternative, further hardware-level adjustments or examinations of the image may be performed.

The workflow from taking a picture to showing the processed result is represented by this paradigm, where each block stands for a critical stage in the entire digital image processing process.

Fig 2. Block Diagram

1) Raw Image:

The starting point of our journey is the Raw Image. This represents the unprocessed visual data captured by a camera or imaging device. It's akin to the untouched canvas awaiting transformation.

2) Compression:

Our next stop is the Compression stage. Here, the raw image undergoes a crucial process. Imagine it as a digital wardrobe organizer—compressing bulky files without compromising quality. Compression reduces the file size, making it more manageable for storage and transmission.

3) Memory:

Finally, our transformed image finds its home in Memory. Think of this as a digital attic—a place where data resides temporarily. Whether it's RAM, hard drives, or cloud storage, memory holds our compressed image until further action is taken.

Image Compression:

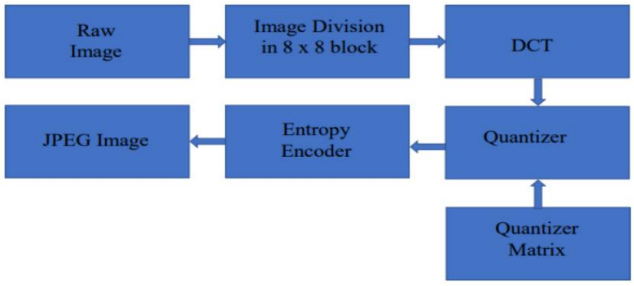

Fig 3. Block Diagram of Compression

2.1 Raw Image:

The procedure starts with a raw picture, which is an array of two-dimensional pixel values that represent the color and intensity information of the image.

2.2 Image Division in 8x8 Blocks:

The raw image is divided into non-overlapping 8x8 blocks. These blocks are independent units for processing and analysis.

2.3 Discrete Cosine Transform (DCT):

Each 8x8 block is then subjected to a mathematical transformation called the Discrete Cosine Transform (DCT). The DCT converts the spatial information of the block into frequency information. It represents the image data in terms of different frequency components.

2.4 Quantization:

After the DCT, the resulting frequency coefficients are quantized. Quantization reduces the precision of the coefficients by dividing them by a set of quantization values. This process discards high-frequency components and preserves the low-frequency components, allowing for significant compression.

2.5 Entropy Encoding:

The next step is to compress the quantized coefficients using an entropy encoding technique (usually Huffman coding). By using the statistical characteristics of the data, entropy encoding assigns longer codes to less frequent coefficients and shorter codes to more often occurring ones. This further reduces the overall size of the encoded data.

2.6 JPEG Image:

Finally, the entropy-encoded data is combined with the necessary header information to form the compressed JPEG image file. The resulting JPEG image consists of a series of compressed blocks, each containing the encoded and quantized coefficients, along with the necessary information for decoding and displaying the image.

Benefits of Image Compression:

It enables a reliable cost of savings that is included with the sending of less data on the network of switched telephones, in which the cost of a call is normally dependent on its duration. It is not only to decrease the requirements of storage but also to decrease the entire time of execution. It decreases the chances of the error's transmission. It enables a level of security against monitoring unlawful activities.

JPEG Process:

1. Dividing the image into 8 by 8 blocks is the first stage in the compression process.

2. While the breaking is done, we must apply the DCT to each image.

3. Therefore, by quantization each block is get compressed.

4. The array of compressed blocks that constitute the image gets stored by drastically reduced amount of space.

Inputs:

The top-level module, jpeg_top, manages a JPEG encoding core with minimal inputs: clock, enable, and reset lines. When the initial pixel's data is ready and stays high during each 8x8 pixel block's 64-clock cycle input, the enable signal is triggered. Following this input phase, there is no more data received for the minimum 33 clock cycles that the enable signal remains high. The enable signal is then quickly lowered for one clock cycle and then raised once again for the subsequent 8x8 block input. Red, green, and blue pixel values are stored on the 24-bit data bus, with blue stored in bits [23:16], green in bits [15:8], and red in bits [7:0]. At the first clock cycle of the last 8x8 block, the end_of_file_signal gets high, signaling that all of the bits in this final block must be produced. The bitstream is managed via the 32-bit output bus.

Outputs:

The JPEG bitstream is output via the 32-bit bus signal JPEG_bitstream. The first eight bits are located at [31:24], followed by the following eight bits at [23:16], and so on. When the data_ready signal is high, the data in the JPEG bitstream is legitimate. To signal genuine data, data_ready will only be high for a single clock cycle. The signal eof_data_partial_ready will be high for one clock cycle when the additional bits are in the signal JPEG_bitstream on the last block of data if the last bits do not fill the 32-bit bus. The end_of_file_bitstream_count 5-bit signal indicates how many additional bits there are.

3. RESULTS

A commonly used image contains redundant information because of neighbouring pixels, which are correlated and contain redundant information. The main objective of image compression is to remove redundancies from an image by removing them as much as possible while keeping the resolution and visual quality of the compressed image as close as possible to the original image. Compression is further divided into predictive and transform coding. Transform coding means that a large amount of information is transferred into a very small number of blocks. One of the best examples of a transformed coding technique is the wavelet transform. Predictive means, based on the training set (neighbours), reduce some redundancies. Context-based compression algorithms are used as predictive techniques.

Using Verilog code, the suggested block diagram was put into practice in hardware. The Xilinx Vivado 2018.2 version tool was used to simulate the hardware, and an RTL schematic for the design was produced. The Xilinx Vivado 2018.2 version tool was also utilized for the synthesis of the design blocks. A C specification may be converted into a register transfer level (RTL) implementation using the Xilinx Vivado High-Level Synthesis (HLS) tool, which can then be

synthesized into a Xilinx field programmable gate array (FPGA). With the FPGA, you may create C specifications in C, C++, SystemC, or as an API C kernel for the Open Computing Language (OpenCL). Its massively parallel design offers advantages over traditional processors in terms of power, cost, and performance. HLS makes it possible by aiming for an FPGA as the execution fabric. This allows the implementation of computationally intensive software algorithms into actual products, not just functionality demonstrators.

3.1 Software Simulation Results

Huffman Image Compression :

1) Original Image:

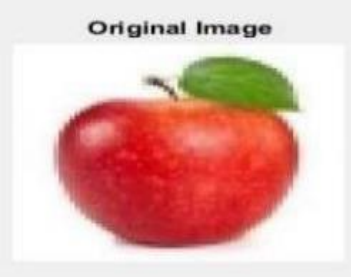

Fig 4. Original Image

The original image's details are as follows:

Image size : 1.45KB

Dimensions : 225x225

- Width : 225pixels
- Height : 225pixels

Horizontal Resolution : Nil

Vertical Resolution : Nil

Bit Depth : Nil

Focal Length : Nil

2) Compressed Image:

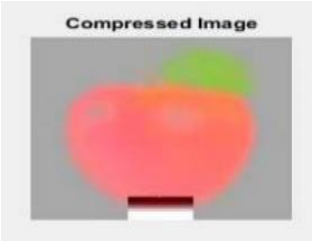

Fig 4. Compressed Image

The compressed image's details are as follows:

Image size : 0.90KB

Dimensions : 64x64

- Width : 64pixels
- Height : 64pixels

Horizontal Resolution : 96 dpi

Vertical Resolution : 96 dpi

Bit Depth : 24

Focal Length : 35mm

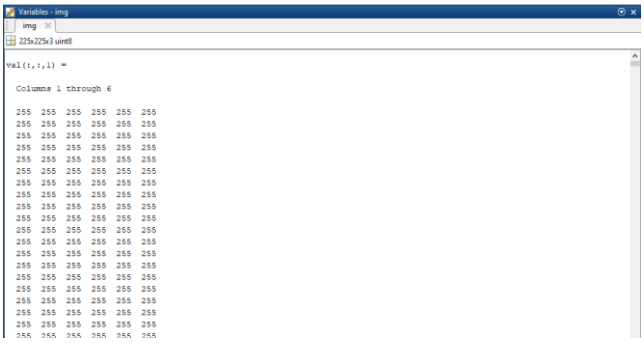

Fig 6. MATLAB Memory Window for RGB Image

A similar simulation is done for the coloured image in Fig. 7. using the same Huffman image compression code. It is clear from the comparison photos that the 1.45 kb original image has been shrunk to 0.90 kb.

3.2 Hardware RTL Schematics, Simulation and Synthesis Results

1) RTL Schematic of the Proposed Block Diagram

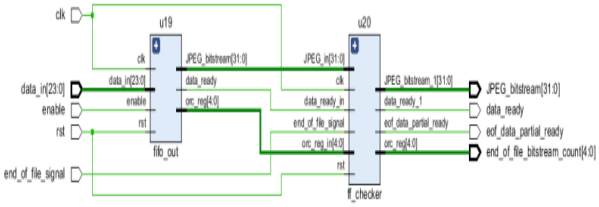

Fig 7. RTL Schematic of The Proposed Block Diagram

2) Simulation Result

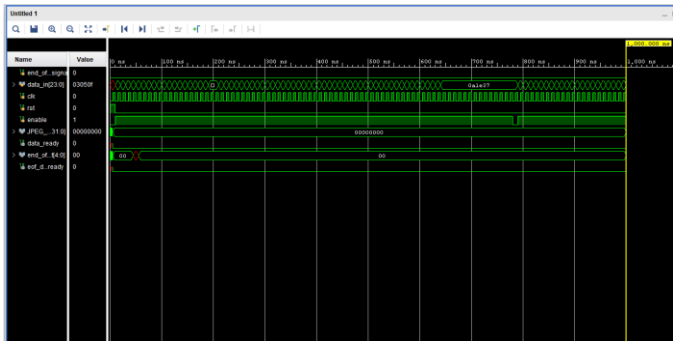

Fig 8. Simulation Result

3) Synthesis Result

The design blocks were synthesised using the Xilinx Vivado 2018.2 Hlx tool. A resource utilisation table is also given below.

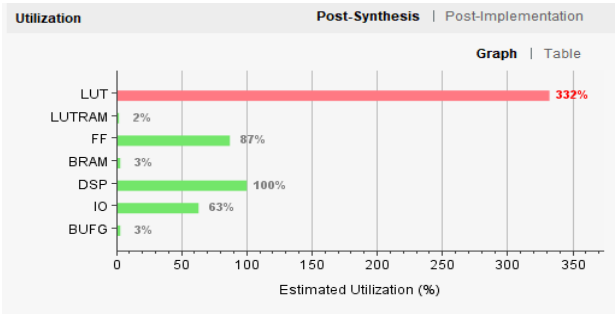

Fig 10. Resource Utilisation Graph

Table -1

Resource	Estimation	Available	Utilization%
LUT	69138	20800	332.39
LUTRAM	156	96100	1.63
FF	36115	41600	86.81
BRAM	1.50	50	3.00
DSP	90	90	100.00
IO	67	106	63.21
BUFG	1	32	3.13

Fig 11. Resource Utilisation Table

4. CONCLUSIONS

We proposed an efficient method to process JPEG images by converting them to grayscale and then to PNG using MATLAB. Image compression reduces memory usage. Identifying data patterns is crucial for effective compression. Implementing image processing on an FPGA involves systematic steps, including camera selection, FPGA choice, and HDL programming. The FPGA synthesis and implementation processes, along with configuration and testing, are crucial for ensuring the successful integration of the designed system.

REFERENCES

[1] Xin Liu, Ling Li, " FPGA-based Three- dimensional endoscope system using a single CCD camera," *IEEE*, 2015.

[2] B.A.S.A. Thilakaratne, W.A.S. Wijesinghe, "FPGA Based Camera Interface For Real Time Video Processing," 2015.

[3] Sanjay Kumar Gupta, "An Algorithm For Image Compression Using Huffman Coding Techniques," *IJARSE*, Vol. No. 5, Issue No. 07, July 2016.

[4] Shammi Rahangdale, Paul Keijzer, P.Kruit, "MBSEM Image Acquisition and Image Processing in Lab View FPGA," *IWSSIP*, 2016.

[5] Ravi B. Humane, Prof. A. R. Askhedkar, " Sensors Interfacing on Re-configurable Platform using FPGA in IoT environment, "IJIRSET, Vol. 5, Issue 9, Sepetember 2016.

[6] WANG Rong-yang, Lu Bo, Qian Zhen-hue, "Real-time Mechanical Parking Equipment Image Acquisition and Preprocessing Based on FPGA," *ICCA*, 2016.

[7] S.M. Dominguez-Nicolas, P. Argiielles-Lucho, P.Wiederhold, "FPGA based image acquisition and graphic interface for hardness tests by indentation," *ISSN*, 2016.

[8] Ms. Sonal R. Lad1, Prof. P.C. Bhaskar2, "Acquisition Board Design Based on Arm and FPGA for Image Data," *IJIREEICE*, Vol. 4, issue 6, June 2016.

[9] Naga Raju Boyal, Vijay Kumar Jindel, Bala Venkateswarlu Avvaru2, Sreelekha Kande3 and Ramanjappa Thogata, "Design and Development of FPGA Based Image Acquisition System," *ISSN*, 2017.

[10] Rui Lu1, Xiaohui Liu1, Xiaodan Wang2, Jin Pan1, Kuangyi Sun1 and Hellen Waynes, "The Design of FPGA-based Digital Image Processing System and Research on Algorithms," ISSN: 2233-7857 Vol. 10, No.2, *IJFGCN*, 2017.

[11] Rang M.H. Nguyen1, Michael S. Brown, "RAW Image Reconstruction Using a Self-contained sRGB-JPEG Image with Small Memory Overhead," *Springer*, 2017.

[12] Himanshu Shekhar1, Hitesh Pant2, Ritanshu Tyagi3, Abhigyan Singh4, "Huffman Coding Based Lossless Image Compression, "*IJARIIE-ISSN(O)*, Vol. No. 4, Issue 05, 2018.

[13] Hao Wang, Zhi Weng, Yan Li, "Design of high-speed image acquisition system based on FPGA," *IEEE*, 2018.

[14] Si Yong Fu, "Design of High Speed Data Acqusition System for Linear Array CCD Based on FPGA," *ICMIR*, 2019.

[15] V.R.Balaji1, J.Sathiya Priya, J.R.Dinesh Kumar, "FPGA Implementation of Image Acquisition in Marine Environment," *ISSN*: 0973-2667 Vol. No. 13, Number 2, 2019.

[16] Rohit Raj1, Navneet Singh2, "A Review-FPGA based Architectures for Image Capturing Consequently Processing and Display using VGA Monitor," IRJET Volume 07, issue 01, Jan 2020.

[17] Prashant Kharat, Vaibhav Mapari, Dr. Amol Bhatkar, "Digital Camera Deactivation By Using IR Based Image Processing Technique," *IJSRST* Vol. 5, issue 6, 2020.

[18] Sharan Reddy Ayiluril, Sampath Kumar Yelchuri2, Vusa Laxumudu3, Gorrepati Praveeen Sajan4, Arvapalli Yaswanth Pavan Kumar5, Kulraj Kaur6, "JPEG Image Compression Using MATLAB," www.irjmrts.com, Vol. No. 3, Issue No. 05, May 2021.

[19] Aleksander Mielczarek 1, Dariusz Radoslaw Makowski 1, Christopher Gerth 2, Bernd Steffen 2, Michele Caselle 3 and Lorenzo Rota 33,4, "Real-Time Data Acquisition and Processing System for MHz Repetition Rate Image Sensors," *MDPI*, 2021.

[20] Eduardo Magdaleno 1, Manuel Rodriguez Valido 1, David Hernandez 2,3, Maria Balaguer 4, Basilio Ruiz Cobo 2,3 and David Diaz 1, "FPGA Implementation of Image Ordering and Parking Algorithm for TuMag Camera," *MDPI*, 2021.

[21] https://www.xilinx.com/support/documentation-navigation/design hubs/dh0012-vivado-high-level-synthesis-hub.html.