JJEE Volume 9 | Number 3 | September 2023 | Pages 322-337

Jordan Journal of Electrical Engineering

ISSN (print): 2409-9600, ISSN (online): 2409-9619

Homepage: jjee.ttu.edu.jo

Rapid Prototyping of Image Contrast Enhancement Hardware Accelerator on FPGAs Using High-Level Synthesis Tools

Muhammad Bilal[1*], Wail Ismael Harasani[2], Liang Yang[3]

[1] Department of Electrical and Computer Engineering, King Abdulaziz University, Saudi Arabia
E-mail: meftekar@kau.edu.sa
[2] Department of Aerospace Engineering, King Abdulaziz University, Saudi Arabia
[3] Division of Energy and Sustainability, Cranfield University, Bedford, United Kingdom

Received: January 07, 2023 *Revised: February 25, 2023* *Accepted: March 02, 2023*

***Abstract*—**Rapid prototyping tools have become essential in the race to market. In this work, we have explored employing rapid prototyping approach to develop an intellectual property core for real-time contrast enhancement which is a commonly employed image processing task. Specifically, the task involves real-time contrast enhancement of video frames, which is used to repair washed out (overexposed) or darkened (underexposed) appearance. Such scenario is frequently encountered in video footage captured underwater. Since the imaging conditions are not known a priori, the lower and upper limits of the dynamic range of acquired luminance values need to be adaptively determined and mapped to the full range permitted by the allocated bitwidth so that the processed image has a high-contrast appearance. This paper describes a hardware implementation of this operation using contrast stretching algorithm with the help of Simulink high-level synthesis tool using rapid prototyping paradigm. The developed model can be directly used as a drop-in module in larger computer vision systems to enhance Simulink computer vision toolbox capabilities, which does not support this operation for direct FPGA implementation yet. The synthesized core consumes less than 1% of total FPGA slice logic resources while dissipating only 7 mW dynamic power. To this end, look-up table has been employed to implement the division operator which otherwise requires exorbitantly large number of logic resources. Moreover, an online algorithm has been proposed which avoids multiple memory accesses. The hardware module has been tested in a real-time video processing scenario at 100 MHz clock rate and depicts functional accuracy at par with the software while consuming lower logic resources than competitive designs. These results demonstrate that the appropriate use of modern rapid prototyping tools can be highly effective in reducing the development time without compromising the functional accuracy and resource utilization.

***Keywords*—**Rapid prototyping; High-level synthesis; Adaptive algorithm; FPGA; Hardware accelerator; Hardware-software co-design.

1. INTRODUCTION

Contrast adjustment is a vital pre-processing in a vast majority of widely used computer vision algorithms. The use and spread of image and video content have become ubiquitous with different operators capturing this data under vastly different lighting conditions. This make it necessary for the captured data to be processed to make it consistent in terms of dynamic range of the pixel luminance values in order for various image processing tasks to function correctly [1]. These include such common tasks as object detection [2], compression, scene segmentation [3], object character recognition, barcode decoding, and medical image enhancement [4-9] etc. The poor lighting conditions lead to an artifact known as 'contrast limitation'. This means that the luminance values of the pixels do not span their full dynamic

* Corresponding author

Article's DOI: 10.5455/jjee.204-1673105856

range, rendering them unfit for the above-mentioned tasks. For 8-bit images, this translates to the range, [0 255], with '0' and '255' denoting absolute dark and absolute bright, respectively. For contrast limited images, the pixels are either clustered towards 255 (washed out), zero (darkened), or the middle (flat). These cases have been depicted through an example in Figs. 1(a) to (f) with the help of histograms of luminance values plotted alongside each case. It can be noticed that only the last image Figs. 1(g) and (h) is visually pleasing because its pixels occupy the full dynamic range i.e. [0 255].

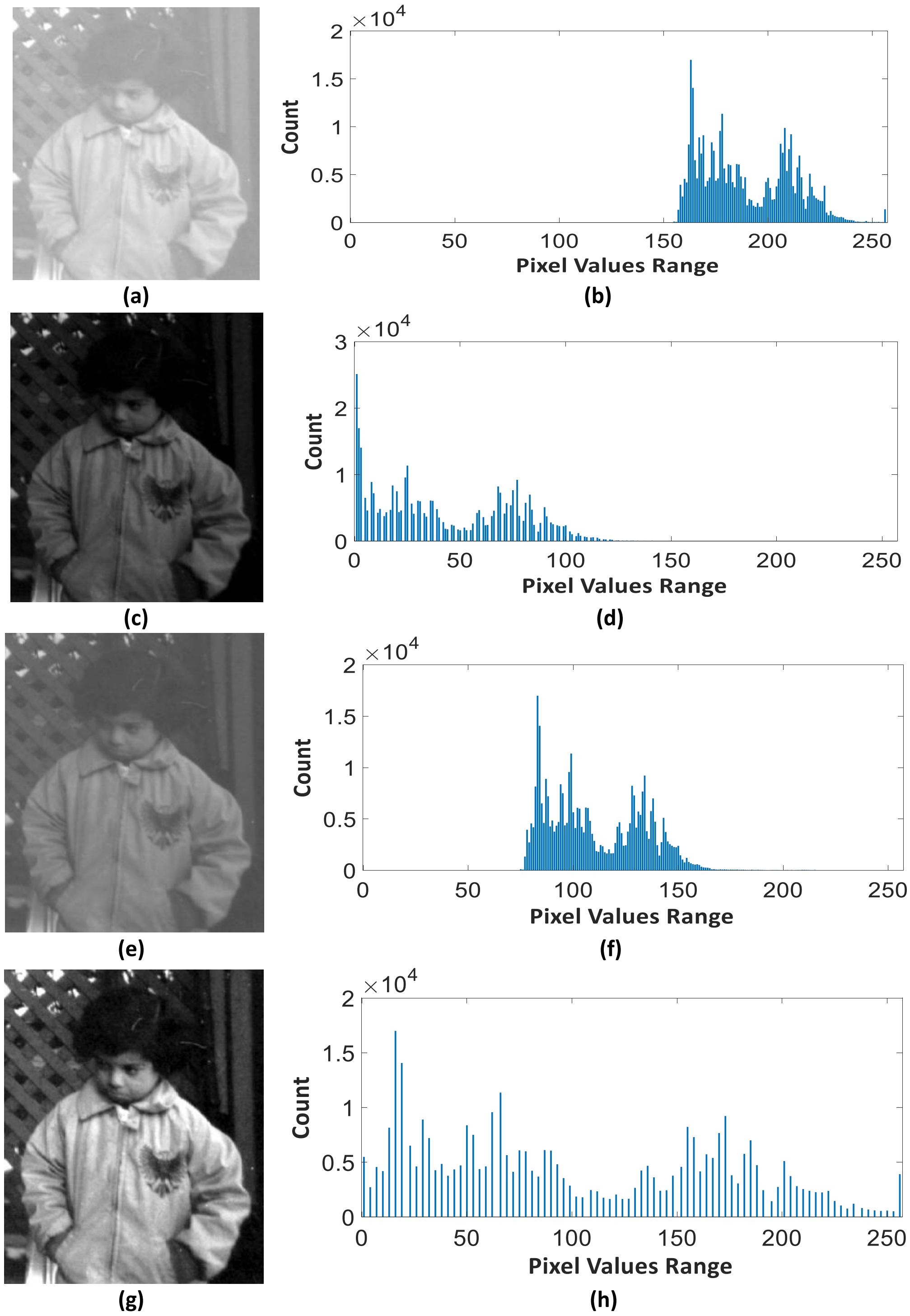


Fig. 1. Contrast stretching example: a) washed out image; b) histogram concentrated towards higher values; c) darkened image; d) histogram concentrated towards lower values; e) flat image; f) histogram concentrated in the middle; g) contrast-enhanced image; h) histogram with full dynamic range.

One of the methods reported in the literature to achieve this result programmatically is knows as 'Contrast Stretching' and can be mathematically represented as [1],

$$\hat{p} = (p - h_{low})\left(\frac{255}{h_{high} - h_{low}}\right) \tag{1}$$

where $\hat{p}$ is the output luminance value, while p is the input. h_{low} and h_{high} are the lower and upper limits of the image histogram. Thus, for a contrast-limited 8-bit image, the difference $h_{high} - h_{low}$ is less than 255 (Fig. 1), and after contrast stretching it is exactly 255. Moreover, it can be noticed from the three different cases of contrast limitation that the limits, h_{low} and h_{high}, can take on very different values. In the case of video streams, these values can change from one frame to another as well. Thus, a practical contrast stretching algorithm has to necessarily determine these two values before applying Eq. (1).

Various contrast enhancement methods have been reported in the literature. Contrast stretching and histogram equalization are the two most widely employed techniques among these. Lie et al. have described a method for image enhancement using histogram projection on non-overlapping blocks [10]. Histogram projection has been argued to be better than histogram equalization for block-based processing since it requires fewer memory resources. Arici et al. have improved upon the conventional histogram equalization technique to give more natural appearance to the processed images [11]. Recently, Kim and Kim have incorporated 1-D and 2-D histograms simultaneously to preserve the shape of the original luminance histogram while enhancing the dynamic range [12]. Abdoli et al. have provided another improvement over the classic histogram equalization method using Gaussian modeling of the homogenous regions in the input image [13]. Thus, a vast variety of image enhancement techniques rely on histogram-based approaches. Although functionally resulting in more visually pleasing results, these techniques inevitably require excessive memory accesses and are, hence, unsuitable for direct incorporation in low-power image processing pipeline hardware. Alareqi et al. have described an FPGA-based implementation of various image enhancement techniques, including contrast stretching, for biomedical applications [14]. They have utilized the high-level synthesis framework provided by Simulink toolboxes [15]. However, they have not provided details of their implemented circuit. Moreover, their framework only supports hardware co-simulation with desktop PC. Akkala et al. have considered the inverse problem of compressing the dynamic range of ultrasound images on FPGA [16]. Their supplied detail is also insufficient to reproduce their implementation or results. Li and Lilja have used stochastic arithmetic units for contrast stretching hardware [17]. However, this approach is only valid in the presence of strong noise. A general-purpose application has not been identified. Hanumantharaju et al. [18] have described a geometric mean-based hardware for image enhancement based on an earlier algorithm by Song and Qiao [19]. This design requires hardware square root and divider modules which consume too many logic resources. The paper does not shed light on how these requirements were met on a Xilinx Virtex-II FPGA with limited on-board resources. Khan et al. have described hardware implementation details of an Anisotropic Gaussian filter-based image enhancement algorithm [20]. This architecture uses local neighborhood processing to stretch the dynamic range of input pixels and requires several complicated arithmetic processing units and memory controllers for operation. The hardware architecture developed by Ho et al. in [21] uses local mean and variance values to adaptively adjust the contrast. However, the values calculated for local neighborhoods may not be globally valid and lead to artifacts. Furthermore, they have not described the implementation details of the contrast stretching equation especially the division operation which is a significant overhead. Luo et al. have described an algorithm for contrast enhancement of underwater images based on histogram stretching [22]. However, this algorithm is not conducive to hardware implementation due to the requirement of explicitly calculating the luminance histogram. Similarly, Kumar and Bhandari have proposed a fuzzy clustering-based algorithm for contrast enhancement which also requires explicit histogram estimation [23]. Lu et al. have suggested an adaptive algorithm for histogram equalization to address the problem of low contrast in infra-red images [24]. However, this algorithm also requires complex multi-scale convolution operations which requires multiple memory accesses per frame and is hence not suitable for deployment on real-time resource-constraint processing systems.

In conclusion, although a variety of different algorithms and corresponding software implementations have been put forward by various researchers, a detailed hardware implementation of the contrast enhancement operation with reproducible results is currently lacking in the contemporary literature. Furthermore, the reported works are mostly based on histogram-based techniques which utilize extensive memory resources for their operation and hence increase the power consumption as well as complexity of the circuit. To this end, we have developed a hardware accelerator for contrast stretching, which does not consume memory units and yet achieves performance at par with the software-based implementations such as those found in popular image processing toolboxes [25] and libraries [26]. The developed hardware IP core has been tested in practical scenario by incorporation into a video processing system implemented as a Hardware-Software (HW-SW) Co-design. To demonstrate its utility in a real-world application, it has been deployed to enhance underwater video footage captured on a surface vehicle developed for research purposes. Furthermore, the whole design framework is available for download as open-source software to facilitate practitioners and researchers in reproducing our work as well as extending the same [27].

2. THE PROPOSED HARDWARE DESIGN

The proposed hardware architecture for contrast stretching with adaptive lower and upper limits has been designed in Matlab/Simulink environment. This setup ensures flexible design platform as well as the facility to test the functionality using built-in software benchmark implementation of various image processing and computer vision algorithms. Moreover, Hardware Description Language (HDL) Coder toolbox allows rapid conversion of the developed model into synthesizable code for deployment on FPGA platforms as IP cores. Thus, the proposed solution has been developed as a drop-in module, which can be integrated within larger computer vision systems designed in Simulink for functionality testing as well as an IP core with standard interfaces for use in FPGA-based image processing pipelines.

Fig. 2 shows the top-level Simulink model of the proposed hardware design. The ‘ContrastStretch_HW’ block contains all the functionality to serially process an incoming pixel stream and adaptively adjust the contrast. The pixels enter this module through the 'pixelIn' signal in a raster scan order. The control signal bus, 'ctrlIn', contains the horizontal and vertical synchronization signals. These two ports combined form the same standard signals, which are commonly adopted by all commercial video cameras. In Xilinx Vivado development environment, AXI4-streaming bus protocol also uses the same signals. Thus, the hardware developed in Simulink HDL coder environment is directly compatible with Xilinx Vivado-based designs.

The 'ContrastStretch_SW' block is a Simulink intrinsic block which serves as the software benchmark for the developed hardware [15]. This block is a part of Simulink Computer Vision toolbox and is widely used by researchers in the field. In order to test the functionality, same video sequence is input to both software and hardware processing blocks and the results are monitored for any differences. Since testing is done offline, the 'Video Source' module reads a locally stored video sequence for processing. The hardware module, however, requires this video data to be fed serially to mimic the raster scan order used by actual video cameras. Thus, the software version has access to the whole frame data but the hardware version can only access the data pixel by pixel and all the required storage has to be done inside the hardware circuit. This is the reason that pixel-based processing is more hardware efficient than block or window-based processing since the latter requires memory units to be allocated for pixel storage. 'Frame To Pixels' block provides this necessary conversion to raster scan format so that the hardware can be tested in software environment.

'Vector2RGB' is another block necessary for the simulation of the hardware block in Simulink environment [15]. By default, Xilinx AXI4 streaming protocol has 4 channels per pixel i.e. one each for the three color channels and one for transparency. Normally transparency is not used in many applications such as the contrast stretching. Thus, the 'Vector2RGB' simply adds the forth transparency channel with a dummy '255' value to signify no transparency. These two conversions are reversed at the output side by 'Pixels To Frame' and 'RGB2Vector' blocks respectively before displaying the results. Furthermore, contrast stretching is usually done for only the luminance aspect of the image. Thus, in this work, we have only considered contrast stretching of this single channel. The Simulink intrinsic 'RBG to intensity' block performs this color to luminance (intensity) conversion for the software module. The hardware module has its own conversion block as an internal component as shown in Fig. 3. After Color Space Conversion (CSC) from RGB to intensity (luminance), the pixel data is processed by a user-defined function block which implements the main contrast stretching algorithm as dictated by Eq. (1) and has been described next.

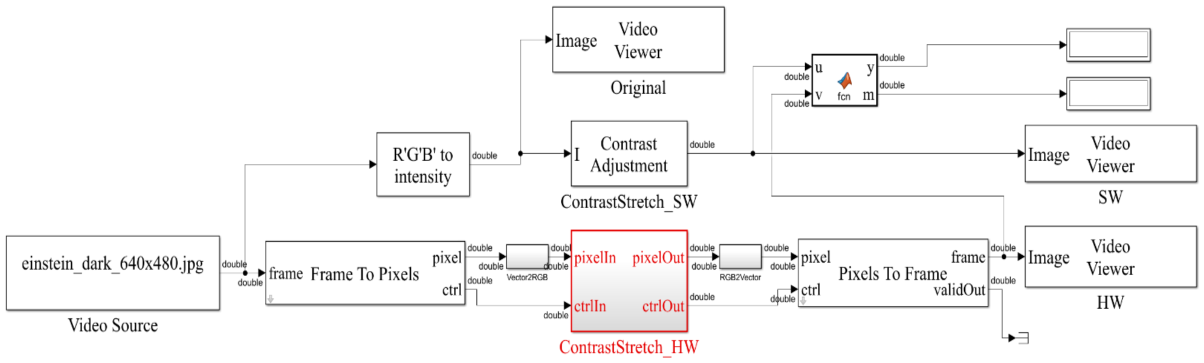


Fig. 2. Top-level Simulink model for development and testing of the proposed hardware for contrast stretching algorithm with adaptive limits.

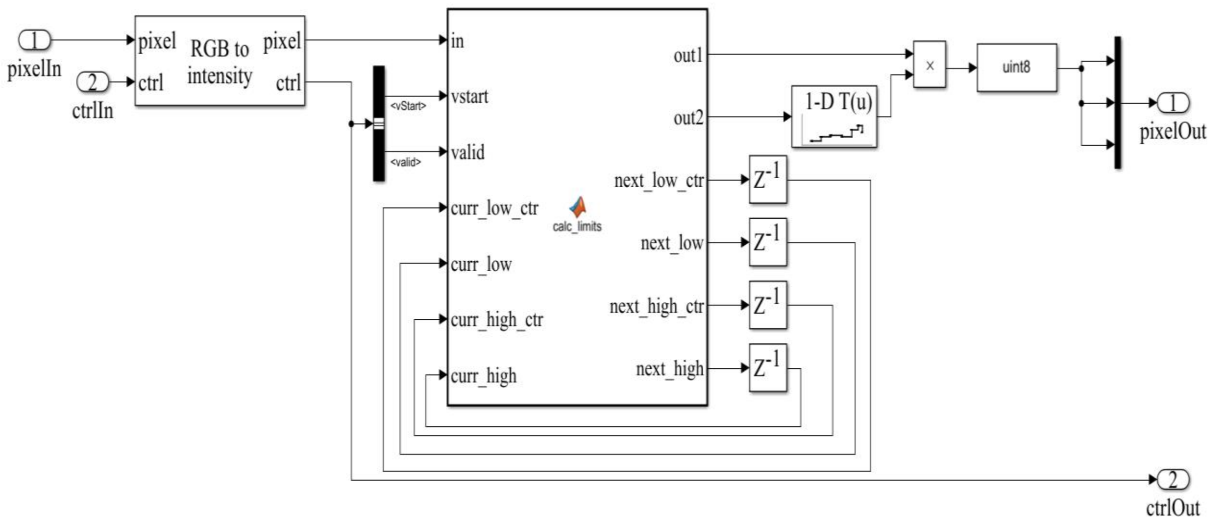


Fig. 3. Internal circuitry of 'ContrastStretch_HW' module.

2.1. Internal Circuit Design of Contrast Stretching Module with Adaptive Limits

As mentioned earlier, the main problem in adaptively adjusting the contrast of an input video stream is the determination of the limits, h_{low} and h_{high}, which can differ with respect to time. The corresponding software benchmark block determines these limits by analyzing the histogram (Fig. 1) of the whole input image/frame. For hardware implementation, however, each pixel is processed only once and not stored for later retrieval. This is important to keep the image processing pipeline simple. Inclusion of memory access units not only consumes more resources, it also slows down the operation and consumes more power. To tackle this problem, in this work, we have exploited the temporal redundancy in the frame data to advantage. Specifically, video frames are captured at a rate of no less than 10 frames per second (fps) in most commercial cameras. Although some higher end systems use up to 120 fps, 10 ~ 30 fps is the common range. While higher numbers are useful to capture the motion of fast objects, the relative motion between adjacent frames is limited to only a few pixels. This means that the pixels take on very similar values and have very low entropy. This fact is used in video compression systems as well where small arithmetic difference between pixel values from adjacent frames is exploited to yield low bit rate for transmission and storage. The same phenomenon can be used for contrast stretching by determining the lower and upper limits for one frame and using it to process the next frame. Since the pixel values do not change drastically between temporally close frames, their histograms also depict the same behavior. Thus, an 'online' algorithm for contrast stretching has been developed where a pixel in the current frame is processed only once; to calculate the lower and upper limits for the next frame and contrast adjustment using the limits that were calculated for the previous frame. This scheme, shown visually in Fig. 4, ensures that no memory units are employed while functional correctness is preserved thanks to the low entropy of visually similar adjacent frames. Thus, the limits calculated for 'Frame 0' are used for 'Frame 1'. Similarly, 'Frame 1' uses the values calculated for 'Frame 0' to adjust its own contrast while calculating the new limits for 'Frame 2' and so on. The corresponding online algorithm to dynamically adjust the contrast by calculating the limits has been described in Algorithm 1.

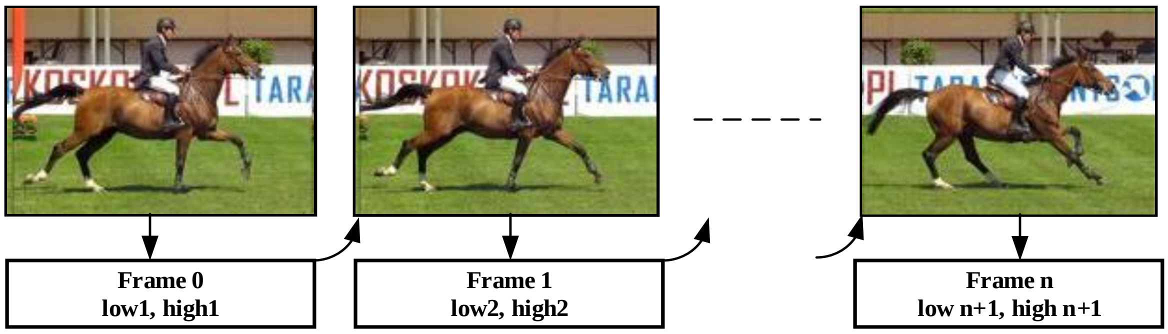


Fig. 4. Online algorithm for adaptive determination of lower and upper limits for contrast stretching by exploiting visual similarity (low entropy) of temporally close frames.

Algorithm 1. Online adaptive contrast adjustment of video frames

```
Input: Video frames : $V_1$, $V_2$, $V_3$, …. $V_n$
Initialize: $h_{low}$ = 0, $h_{high}$ =255
for $i$=$1$ to $n$ do
        if(($\#PixelCount < h_{low}) = 0.01 * \#TotalPixelCount$)
                $h_{low}$ = $h_{low}$ + 1
        if(($\#PixelCount > h_{high}) = 0.01 * \#TotalPixelCount$
                $h_{high}$ = $h_{high}$ -1
        for each pixel p in the frame do
                $\hat{p} = (p - h_{low})\left(\frac{255}{h_{high}-h_{low}}\right)$
end
```

In order to calculate the limits for contrast stretching, the span of the image histogram is analyzed (Fig. 1). In practice, a few pixels always take on extremely low and extremely high values while the bulk forms the main histogram, which may or may not span the full dynamic range. Thus, theoretically, the histogram spans the full dynamic range due to these 'outlier' pixels. To counter this problem, researchers in [1] have suggested to discard these pixels at both extremes by a determined percentage. Simulink software block adopts the same strategy. Thus, the lower limit, h_{low}, is determined as follows,

$$h_{low} = x \ni p(X < x) = p1 \tag{2}$$

where 'X' is the set of all pixel luminance values in the input frame, and '$p1$' is a pre-determined constant. The probabilities are estimated from the histogram of the luminance values. Thus, for $p1$ = 0.01, the lower limit is set to discard 1% of pixels from the lower end of the histogram (outliers). Similarly, h_{high}, is determined as follows,

$$h_{high} = x \ni p(X > x) = p2 \tag{3}$$

Thus, for $p2$ = 0.01, the upper limit discards 1% of outlier pixels from the upper end of the histogram. The values of $p1$ and $p2$ are generally fixed a priori. We have set these values to be 0.01 for both. The corresponding software block is set to use the same values for functionality comparisons. As mentioned earlier, in order to save precious memory resources, the proposed hardware model does not populate a full histogram for the frames being processed. Instead, the lower and upper limits are determined using the following simplified formulation,

$$p(X < x) = p1 = 0.01 \tag{4}$$

$$\frac{\#PixelCount < h_{low}}{\#TotalPixelCount} = 0.01 \tag{5}$$

$$(\#PixelCount < h_{low}) = 0.01 * \#TotalPixelCount \quad (6)$$

The value of h_{low} is initialized to be 0 at the start. Since the frame size is fixed as well as the value of $p1$, the hardware only needs to count the number of pixels with luminance values less than the current value of h_{low} to determine whether Eq. (6) holds true or not. This can be achieved through a counter and a comparator. If at the end of the frame, this equation holds, then no change to the limit is made. If, on the other hand, the number of pixels on the left-hand side (LHS) of Eq. (6) is less than the right-hand-side (RHS) constant, the limit is increased by a step, usually one. In the case of LHS being more than RHS, the limit is decreased by the same step. Thus, h_{low}, is adaptively adjusted every frame to ensure that it closely follows the actual lower limit of the true histogram of the image by only allowing a few outliers (e.g. 1%) below this limit. A similar formulation for the upper limit can be expressed as follows,

$$(\#PixelCount > h_{high}) = 0.01 * \#TotalPixelCount \quad (7)$$

h_{high} is initialized to be 255 at the start.

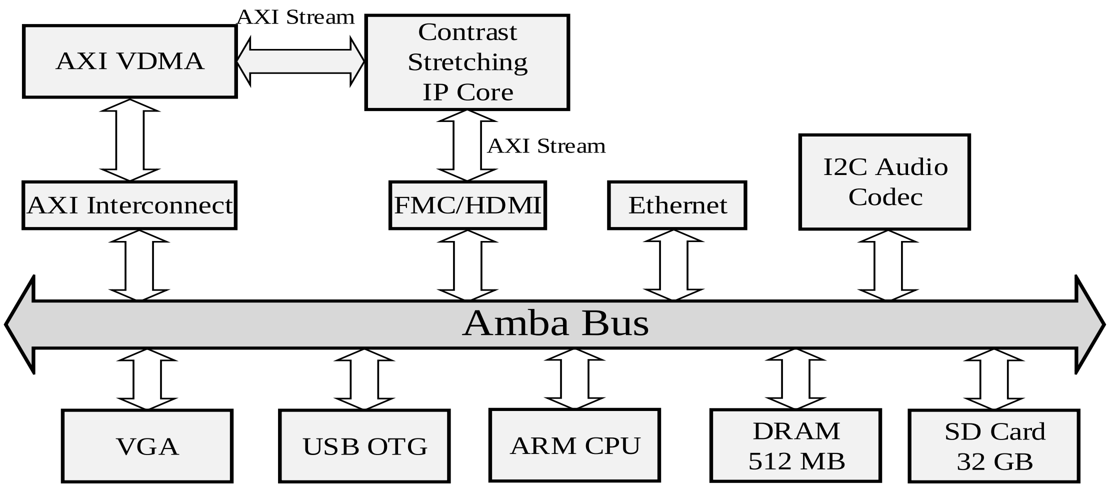
Thus, the hardware needs to store only the upper and lower limits in the register along with two separate counters. These four registers have been shown in Fig. 3. The comparison and update mechanism has been coded in the user-defined function, 'calc_limits'. This function uses 'vstart' and 'valid' signals to count the pixels according to the conditions in Eqs. (6) and (7). The first of these signals signifies the start of the frame while the second asserts when valid pixel values are present. Since, h_{high} and h_{low} always lie within the range [0 255], 8-bit registers have been allocated to each of these. For the counter, 20-bit registers have been allocated. This allows counting to '1048576' which allows processing VGA resolution frames (640 × 480). For larger frames, more bit-widths are needed to be allocated accordingly.

The actual contrast stretching operation, Eq. (1), has also been implemented in the same user-defined function i.e. 'calc_limits'. As discussed in the previous section, this operation involves a division operator which demands a lot of logic resources if implemented directly in the hardware. To circumvent this problem, we have implemented the division operation as a Look-Up Table (LUT). In Eq. (1), the value '$h_{high-}h_{low}$' is in the numerator. This signal has the dynamic range [0 255]. Thus, an LUT has been incorporated, which stores the pre-computed reciprocal values for this range. In Fig. 3, the signal 'out2' is input to the LUT to implement the division operation of Eq. (1). Signal 'out1' is the remaining part of Eq. (1). The two signals are then multiplied to give the final result after conversion to an 8-bit integer. The interim signals use fixed-point format to preserve the fidelity. Thus, the LUT values are stored as 16-bit operands with 15 bits allocated for the fractional part.

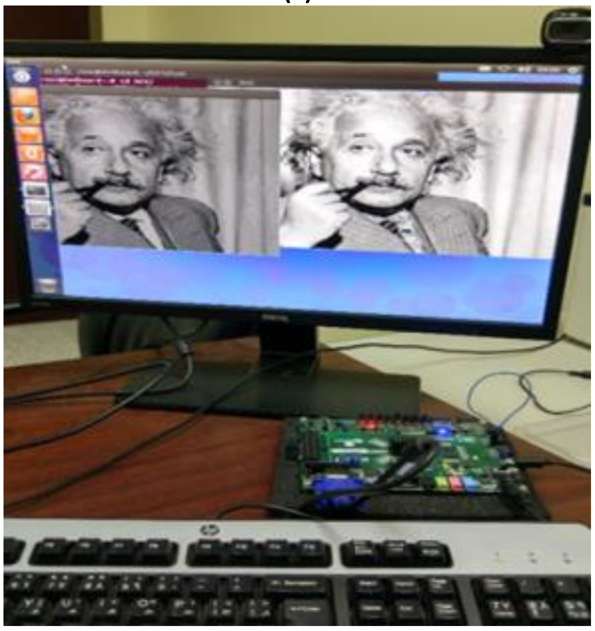
2.2. IP Core Generation and Incorporation in HW-SW Co-Design

The 'ContrastStretch' module in the top module is selected for HDL generation using HDL coder after testing for functionality and comparison with the software module. This tool provides the necessary options to select the Xilinx specific interfaces such as the AXI4 streaming protocol. It is worth mentioning here that if the LUT-based division operation is not implemented, then the Xilinx synthesis tools have to be instructed to use DSP blocks for this purpose as discussed in the next section.

The generated IP core is tested for use in the practical environment by insertion into a HW-SW co-design. In this work, Xilinx Vivado tool has been used for design entry, synthesis and ultimately programming the FPGA platform. We have employed Xilinx Zedboard for this purpose. It has Xilinx Zynq-7000 AP SoC XC7Z020-CLG484 FPGA with a clock speed of 100 MHz. The on-chip ARM microprocessor provides the necessary software support through Ubuntu with OpenCV computer vision library. This HW-SW co-design has all the necessary peripherals to implement a video processing system as shown in Fig. 5. The contrast stretching IP core can use either direct video stream from HDMI input or frames stored on main RAM accessible via video DMA through AXI4 streaming protocol. Alternatively, a webcam can also be used as the source of video stream. The dataflow through the Vivado project has been shown in Fig. 6. The contrast stretching IP core can be optionally accessed through AXI-lite interface to change user settings as well. Our design, however, adaptively sets the limits based on the frame characteristics. Thus, there are no user-settable options.



(a)



(b)

Fig. 5. HW-SW co-design: a) block diagram; b) implementation on Zedboard to test the developed contrast stretching IP core.

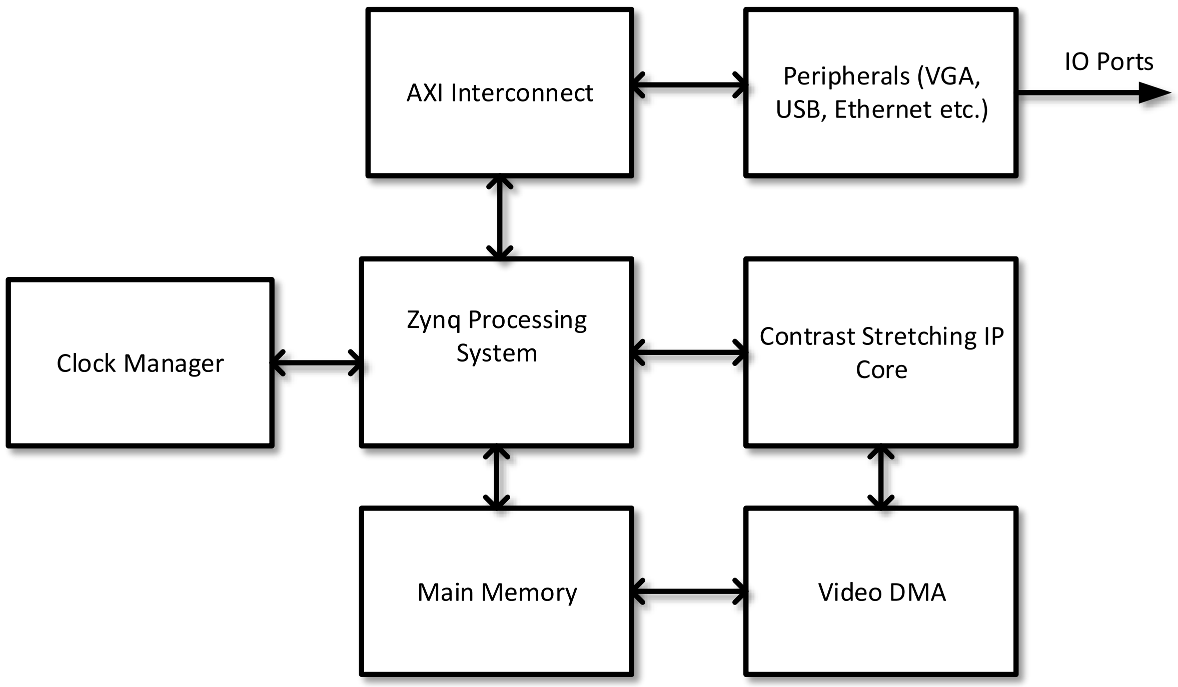


Fig. 6. HW-SW co-design components' integration in the Vivado project.

3. RESULTS AND DISCUSSION

The summary results of synthesizing the complete HW-SW co-design (Fig. 6) on Zedboard using Vivado are illustrated in Fig. 7. It can be noticed that full system consumes only 16% of the available slice LUT resources while only minimal registers, on-chip memory (BRAM) and specialized arithmetic units (DSP) have been used. Thus, there are enough logic resources remaining to implement a larger computer vision system. This is also evident from the FPGA layout reported by Vivado 'place and route' tool and shown in Fig. 8. It can be seen that most of the silicon real-estate on FPGA device is vacant.

Table 1 contains the detailed FPGA synthesis report of the developed hardware IP core for contrast stretching along with the complete system as described in the previous section. It can be noticed that the proposed design consumes around 4% of the total Slice LUT used by the full system which is less than 1% of the whole FPGA. Moreover, the CSC part of the IP consumes the bulk of these resources. Thus, 3 DSP blocks are consumed by this module, and only one by the circuit implemented for Eq. (1). Same is true for the registers. The CSC module uses 75% of all the registers consumed by the whole IP. It may be reiterated that CSC is an intrinsic Simulink block and does not allow any changes to be made inside. The bulk of the logic resources, registers and memory units (BRAM) are utilized by the Processing Side (PS) and its interfaces. It also consumes the most dynamic power as estimated by the Vivado tool i.e. 1.77 W. In contrast, the developed IP core only dissipates 7 mW while operating at 100 MHz. At this clock rate, it is able to process Full HD video frames (1920 × 1080) at 48 fps. The extremely low resource utilization and associated low power dissipation has been made possible mainly due to the LUT-based implementation of the divider module and adoption of online algorithm for adaptive adjustment of limits. A divider is generally synthesized using multiple DSP blocks and consumes considerable dynamic power as well. Moreover, the online algorithm (Fig. 4) eliminates the need for irregular memory accesses and processes each pixel only once. This is made possible by realizing and exploiting the temporal redundancy of the adjacent frames. In comparison, the adaptive contrast enhancement technique proposed by [28] consumes multiple times more logic resources as well as BRAM and DSP units. Moreover, their design implemented through Xilinx System Generator is only capable of running at 60 MHz. In [29], the authors have reported the FPGA implementation of various image enhancement techniques for automatic vehicle plate detection. Their corresponding implementation of contrast stretching hardware also consumes almost double the LUT resources than our proposed design. Moreover, this design consumes a lot of on-chip memory resources, as well. The reported dynamic power consumption is also not suitable for insertion into real-time image processing pipelines. In [30], the authors have proposed FPGA implementation of an advanced adaptive contrast enhancement algorithm. While this design achieves operating speed up to 350 frames per seconds, it consumes exorbitantly large number of resources as seen from Table 1. These implementation results demonstrate that implementation of complicated contrast enhancement algorithms inevitably require excessively large number of on-chip resources which prohibits inclusion of such circuits in real-time image processing pipeline of low-complexity vision systems. The proposed solution, on the other hand, provides a drop-in module for inclusion in computer vision systems developed using Simulink high-level synthesis tools and requires fewer logic resources.

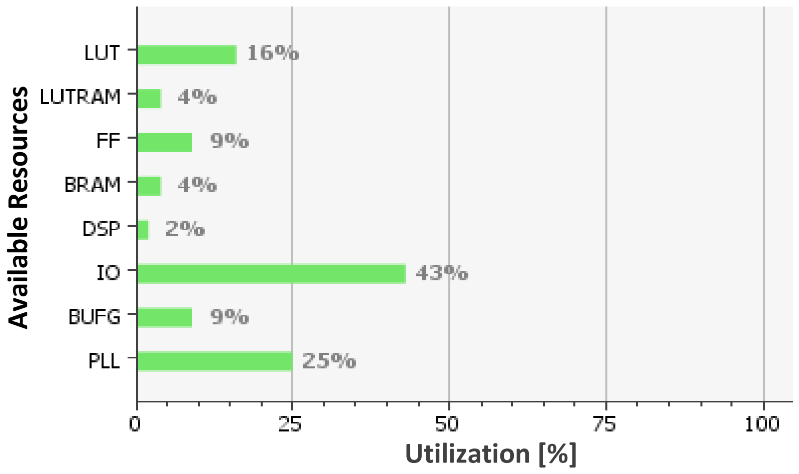


Fig. 7. Design utilization as a percentage of total available resources on FPGA platform.

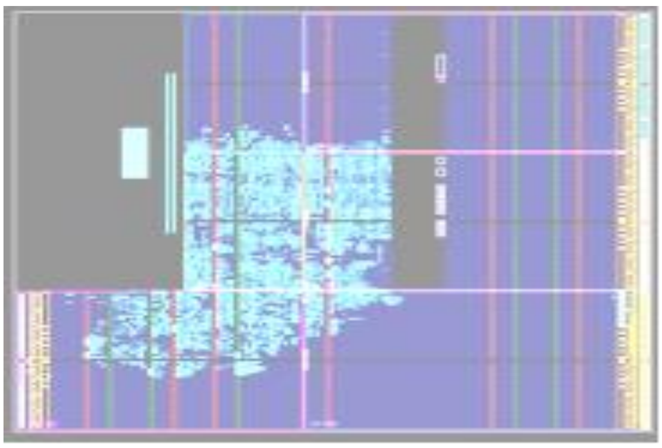


Fig. 8. FPGA layout after 'place and route' phase of implementation.

Table 1. FPGA synthesis results for the proposed contrast stretching IP core.

Component	FPGA	Slice LUT	Slice registers	BRAM	DSP	Dynamic Power [W]
Proposed Design Full System	Xilinx XC7Z020	8355	9118	6	4	1.768
AXI Interconnect	-	1182	1399	1.5	0	0.006
AXI VDMA	-	2231	2732	0	0	0.01
Contrast Stretching IP Core	-	336	304	0	4	0.007
CSC converter	-	210	228	0	3	-
Adaptive Limits	-	34	76	0	0	-
Eq. (1)	-	92	0	0	1	-
Ref. [28]	Xilinx Spartan III	3342	-	4	17	
Ref. [29]	Xilinx Virtex V	756	-	800	-	0.101
Ref. [30]	Xilinx Virtex IV	4766	440	16	-	-

Moreover, despite using approximate arithmetic circuit for division operation, the functional accuracy is also very close to its software counterpart, as can be noticed from Fig. 9. The results of the hardware model, despite using fixed-point and approximate arithmetic processing elements, are perceptually identical to those of the full precision software model. Even in the worst-case scenario with extremely low contrast ration in the input image, the maximum deviation from the software model is 10 per pixel, with average error being only 3. Fig. 10 shows the result of applying the proposed online contrast stretching algorithm to a video frame captured via an underwater observation camera.

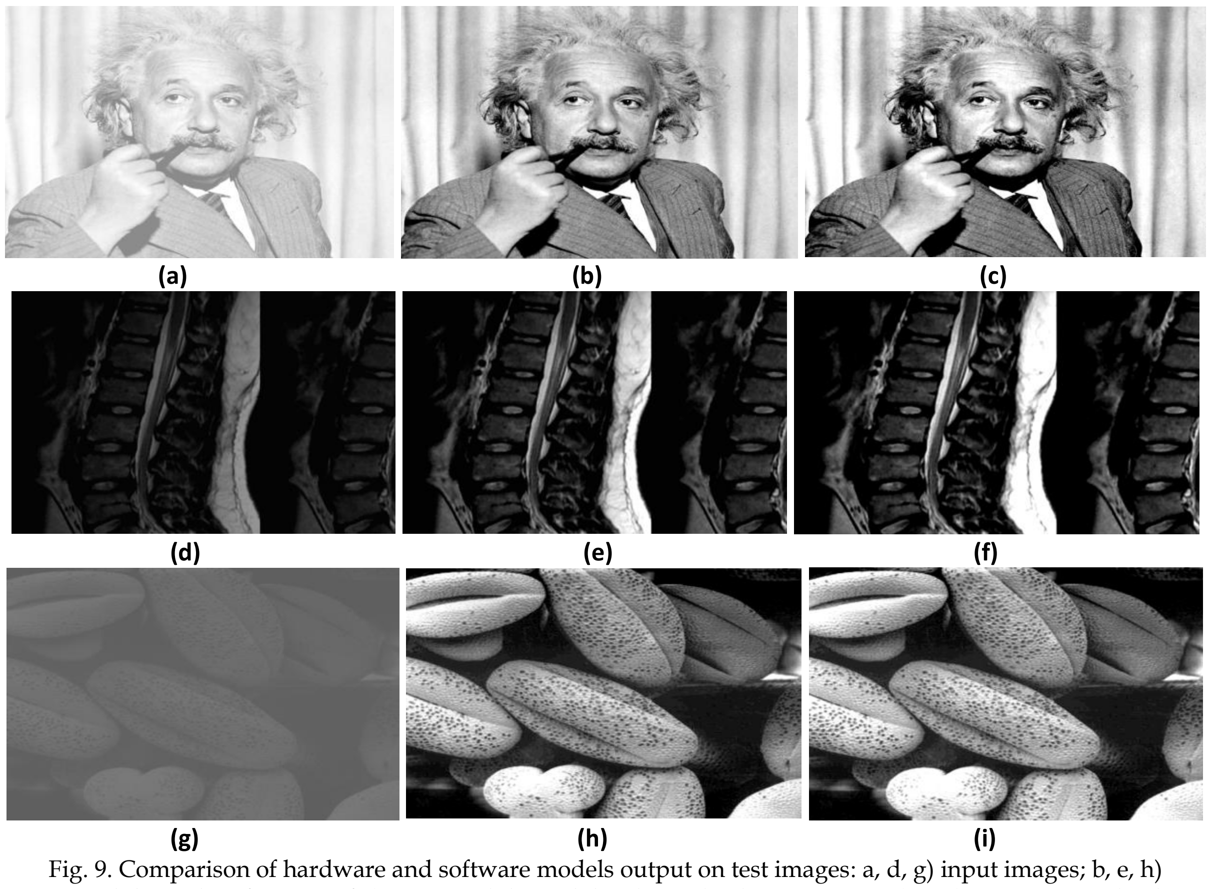
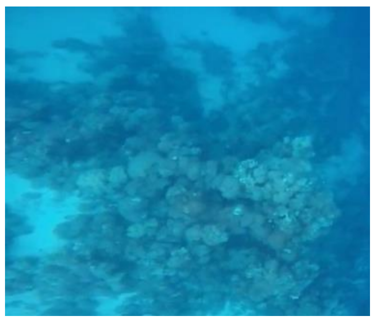
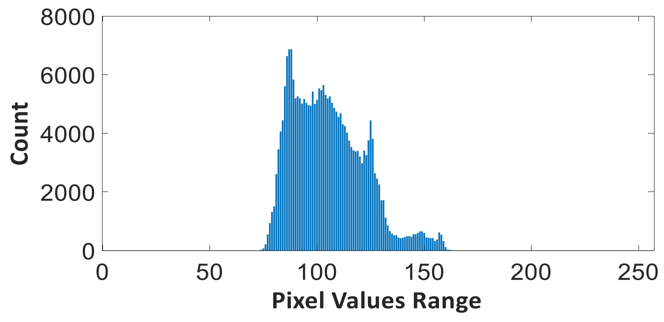
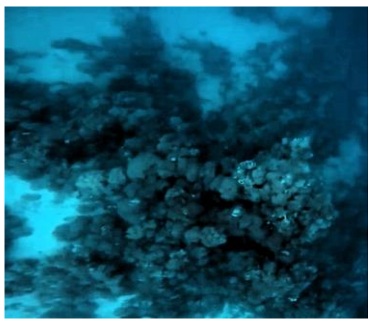


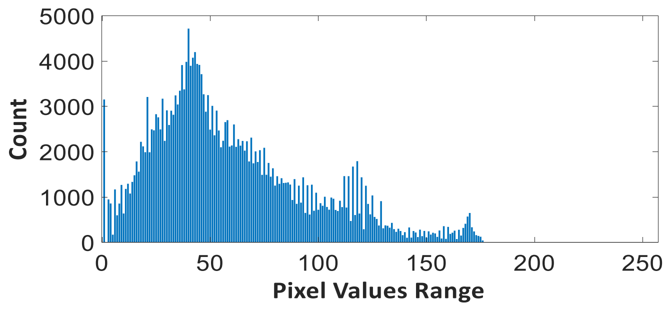
Fig. 9. Comparison of hardware and software models output on test images: a, d, g) input images; b, e, h) processed through Software; c, f, i) processed through hardware leading to average(max.) error per pixel values of 1.075(4), 1.4(4) and 3.1(10), respectively on the tested images.


(a)

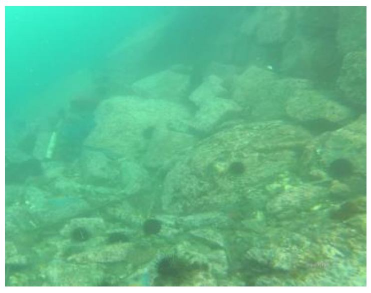


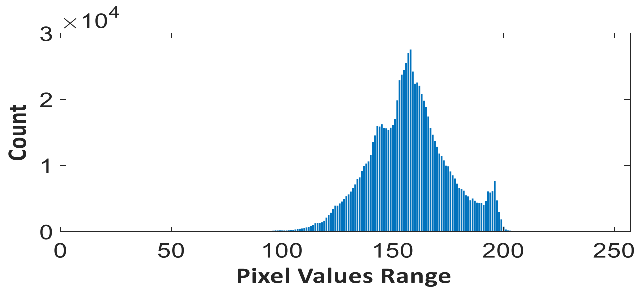
(b)


(c)

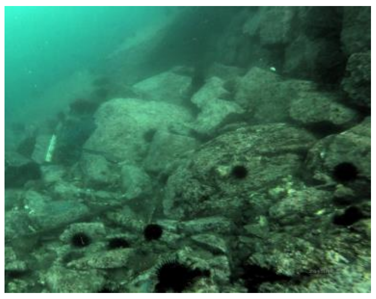


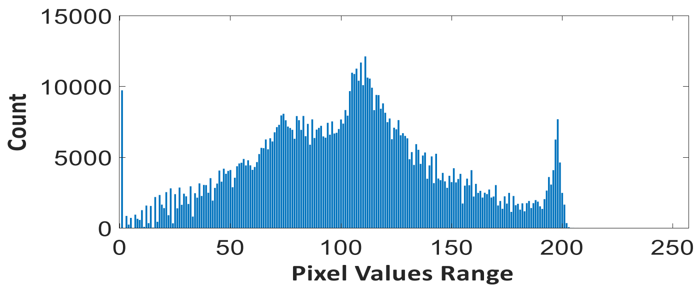
(d)


(e)



(f)


(g)



(h)

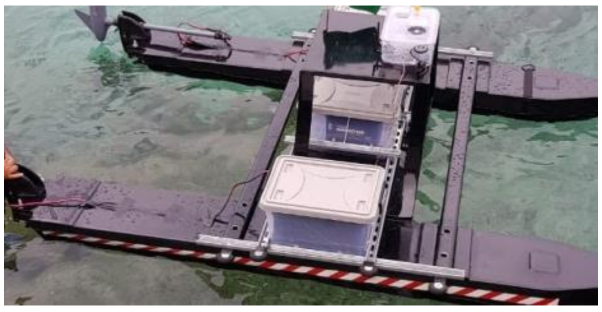

(i)

Fig. 10. Visual results of contrast enhancement of underwater video frames using the proposed online algorithm: a) input frame 1; b) input frame 1 histogram; c) processed frame 1; d) processed frame 1 histogram; e) input frame 2; f) input frame 2 histogram; g) processed frame 2; h) processed frame 2 histogram; i) underwater video capturing system on-board explorer boat.

The research platform boat shown in Fig. 10(i) has been especially developed to explore the underwater coral structures near the coast of Jeddah, Saudi Arabia in Red Sea using downward looking submerged camera. As seen in Figs. 10(a) and (c), the sample video frames suffer from contrast limited exposure. This can be observed as narrow histograms of the corresponding illumination values shown in Figs. 10(e) and (g) respectively for each sample. Figs. 10(b) and (d) show the result of processing the contrast limited frames using the proposed algorithm. As depicted by their corresponding visual appearances as well as the spread-out histograms (Figs. 10(f) and (h)), the contrast has been adequately enhanced.

4. CONCLUSIONS

This paper has described an adaptive contrast enhancement algorithm and its corresponding hardware implementation on FPGA. The proposed hardware is not only very suitable for inclusion in an image processing pipeline for real-time applications due to its low power dissipation and processing speed, but it also consumes very few logic resources. Furthermore, the contrast enhancement module has been developed as a Simulink block which makes it a valuable resource for inclusion in larger computer vision system since Vision HDL toolbox does not provide this functionality as a drop-in synthesizable module at the moment. This work has demonstrated the real-time contrast stretching operation with minimal resources on a Xilinx Zynq FPGA which may not be suitable for energy-constraint systems due to its higher static power consumption. Thus, for the future work, it has been planned to port the design onto commercially available low-power devices.

Acknowledgement: The authors acknowledge the fund from the British Council

REFERENCES

[1] R. Gonzalez, R. Woods, *Digital Image Processing*, Upper Saddle River, N.J.: Prentice Hall, 2008.

[2] H. Al-Zoubi, M. Al-khassaweneh, I. Altawil, "An image processing approach for marble classification," *Jordan Journal of Electrical Engineering,* vol. 1, no. 2, pp. 73-81, 2015.

[3] A. Alqudah, S. Qazan, H. Alquran, I. Qasmieh, A. Alqudah, "COVID-19 detection from X-ray images using different artificial intelligence hybrid models," *Jordan Journal of Electrical Engineering,* vol. 6, no. 2, pp. 168-178, 2020.

[4] C. Huang, M. Nguyen, "X-ray enhancement based on component attenuation, contrast adjustment, and image fusion," *IEEE Transactions on Image Processing,* vol. 28, no. 1, pp. 127-141, 2019.

[5] J. Liu, C. Zhou, P. Chen, C. Kang, "An efficient contrast enhancement method for remote sensing images," *IEEE Geoscience and Remote Sensing Letters,* vol. 14, no. 10, pp. 1715-1719, 2017.

[6] B. Xu, Y. Zhuang, H. Tang, L. Zhang, "Object-based multilevel contrast stretching method for image enhancement," *IEEE Transactions on Consumer Electronics,* vol. 56, no. 3, pp. 1746-1754, 2010.

[7] V. Syrris, S. Ferri, D. Ehrlich, M. Pesaresi, "Image enhancement and feature extraction based on low-resolution satellite data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 8, no. 5, pp. 1986-1995, 2015.

[8] K. Hee-Chul, K. Byong-Heon, C. Myung-Ryul, "An image interpolator with image improvement for LCD controller," *IEEE Transactions on Consumer Electronics,* vol. 47, no. 2, pp. 263-271, 2001.

[9] M. Suthar, H. Asghari, B. Jalali, "Feature enhancement in visually impaired images," *IEEE Access,* vol. 6, pp. 1407-1415, 2018.

[10] B. Liu, W. Jin, Y. Chen, C. Liu, L. Li, "Contrast enhancement using non-overlapped sub-blocks and local histogram projection," *IEEE Transactions on Consumer Electronics,* vol. 57, no. 2, pp. 583-588, 2011.

[11] T. Arici, S. Dikbas, Y. Altunbasak, "A histogram modification framework and its application for image contrast enhancement," *IEEE Transactions on Image Processing,* vol. 18, no. 9, pp. 1921-1935, 2009.

[12] D. Kim, C. Kim, "Contrast enhancement using combined 1-D and 2-D histogram-based techniques," *IEEE Signal Processing Letters,* vol. 24, no. 6, pp. 804-808, 2017.

[13] M. Abdoli, H. Sarikhani, M. Ghanbari, P. Brault, "Gaussian mixture model-based contrast enhancement," *IET Image Processing,* vol. 9, no. 7, pp. 569-577, 2015.

[14] M. Alareqi, R. Elgouri, M. Tarhda, K. Mateur, A. Zemmouri, A. Mezouari, L. Hlou, "Design and FPGA implementation of real-time hardware co-simulation for image enhancement in biomedical applications," *in 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems*, pp. 1-6, 2017.

[15] *Simulink Vision HDL Toolbox*, 2018. <https://www.mathworks.com/products/vision-hdl.html>

[16] V. Akkala, P. Rajalakshmi, P. Kumar, U. Desai, "FPGA based ultrasound backend system with image enhancement technique," *in 5th ISSNIP-IEEE Biosignals and Biorobotics Conference (2014): Biosignals and Robotics for Better and Safer Living*, pp. 1-5, 2014.

[17] P. Li, D. Lilja, "Using stochastic computing to implement digital image processing algorithms," *in 2011 IEEE 29th International Conference on Computer Design*, pp. 154-161, 2011.

[18] M. Hanumantharaju, M. Ravishankar, D. Rameshbabu, S. Ramachandran, "A novel FPGA implementation of adaptive color image enhancement based on HSV color space," *in 2011 3rd International Conference on Electronics Computer Technology*, vol. 2, pp. 160-163, 2011.

[19] G. Song, X. Qiao, "Color image enhancement based on luminance and saturation components," *Proceedings of 1st International Congress on Image and Signal Processing*, vol. 3, pp. 307-310, 2008.

[20] T. Khan, D. Bailey, M. Khan, Y. Kong, "Efficient hardware implementation for fingerprint image enhancement using anisotropic gaussian filter," *IEEE Transactions on Image Processing,* vol. 26, no. 5, pp. 2116-2126, 2017.

[21] J. Yun Ho, K. Jae Seok, H. Bong Soo, K. Moon Gi, "Design of real-time image enhancement preprocessor for CMOS image sensor," *IEEE Transactions on Consumer Electronics,* vol. 46, no. 1, pp. 68-75, 2000.

[22] W. Luo, S. Duan, J. Zheng, "Underwater image restoration and enhancement based on a fusion algorithm with color balance, contrast optimization, and histogram stretching," *IEEE Access,* vol. 9, pp. 31792-31804, 2021.

[23] R. Kumar, A. Bhandari, "Fuzzified contrast enhancement for nearly invisible images," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 32, no. 5, pp. 2802-2813, 2022.

[24] H. Lu, Z. Liu, X. Pan, "An adaptive detail equalization for infrared image enhancement based on multi-scale convolution," *IEEE Access,* vol. 8, pp. 156763-156773, 2020.

[25] *Matlab Image Processing Toolbox*. https://www.mathworks.com/products/image.html?adobe_mc_ref=https%3A%2F%2Fwww.google.com.sa%2F>

[26] *The OpenCV Reference Manual (2.4.9.0 ed.)*. <http://opencv.org/>

[27] *Real-time Contrast Stretching IP Core*, 2021. <https://github.com/4mbilal/XilFPGAdev/tree/main/Custom_IP_Cores/MathworksHDLCoder/ContrastStretching >

[28] C. Lu, H. Hsu, L. Wang, "A new contrast enhancement technique implemented on FPGA for real time image processing," *in 2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 542-545, 2009.

[29] R. Shandilya, R. Sharma, "FPGA implementation of image enhancement technique for automatic vehicles number plate detection," *in 2017 International Conference on Trends in Electronics and Informatics*, pp. 1010-1017, 2017.

[30] B. Unal, A. Akoglu, "Resource efficient real-time processing of contrast limited adaptive histogram equalization," *in 2016 26th International Conference on Field Programmable Logic and Applications*, pp. 1-8, 2016.