Implement Detail Enhancement Algorithm on FPGA for Real-Time and Energy-Efficient Embedded Systems

Le Thanh Tung†, Le Thanh Bang†, Pham Trong Thuy†, Nguyen Duc Hoan†, Vuong Dang Huy†
† Center of Engineering and Technologies, Viettel High Technologies Corporation, Hanoi, Vietnam

***Abstract*—FPGA (Field Programmable Gate Array) is a platform that allows carrying out many digital signal processing designs with numerous benefits in terms of speed, energy consumption and flexibility. The first part of this paper introduces a method enhancing the contrast of an input image while simultaneously compressing the dynamic range of that one. The second part focuses on converting that algorithm into a digital design to run successfully on Xilinx FPGA. We also estimate the outcome on Zybo Z7 board of Digilent and achieve an impressive result with 120 fps for input/output images of resolution 640x512. In addition, fixed-point numbers operations offer a high precision in comparison with floating-point numbers in MATLAB, which brings high reliability in deployment.**

***Index Terms*—FPGA, digital design, embedded system, edge-preserving filter, histogram projection, detail enhancement.**

I. Introduction

Digital Signal Processing (DSP) techniques have been applied in a range of different areas such as wireless communication, speech/image processing [1]. The important issues when building actual DSP systems that might be used in practice are related to the real-time ability and low-power demand [2]. Nowadays, there are a lot of development platforms which may meet those criteria. For instance, we can use GPUs for executing thousands of multiplications at the same time to speed-up computationally intensive tasks instead of CPUs. Besides, several firms provide DSP Integrated Circuits (ICs) accompanied by their own software development environments for compiling source code from some high level programming language or assembly to run on those chips. These DSP ICs are built to accelerate computational operations containing many additions and multiplications. Not only that, another programmable IC category which might be applicable in reality is FPGA. One of the advantages of FPGA compared to other IC categories is flexibility. We can use FPGA to implement DSP applications with our own interface standards without any constraint. Moreover, the stability is also a considerable perspective because with FPGA, we can estimate and modify our design to satisfy requirements including latency, pipelined stages, resources usage, etc.

A number of previous research have mentioned typical and popular tools for image enhancement, both in spatial and frequency domain, such as histogram equalization in [3], [4] and [5], histogram stretching in [6], wavelet transform in [7]. Tarek M. Bittibssi et.al [8] have been compared preceding papers and introduced the implementation of several traditional mechanisms on FPGA: median filtering, negative transformation, contrast stretching, etc. Most of these techniques produce output images which have the same gray levels as original images (e.g. 256 gray levels corresponding to 8-bit pixels). However, modern infrared cameras are able to snap a photo or record a video with a grayscale up to 14-bits associated with 16384 distinct gray levels, which makes enhancing the quality of images much more efficient. For example, if 14-bit images have low contrast meaning that it is too dark or too bright, we can take advantage of some processing mechanism to receive 8-bit output images but have better quality. Another reason to consider shrinking the dynamic range is that 8-bit images are good enough for human vision and 14-bit grayscale cameras have been become available today, particularly in military industries, since infrared image processing is crucial.

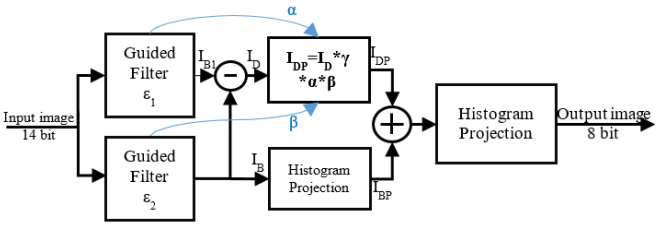

Fig. 1: The steps of detail enhancement algorithm

In this paper, we present a method for enhancing an input image by performing guided filters and histogram projection based on the method introduced in [9]. The input image containing 14-bit pixels is processed to produce the resulting image with 8-bit pixels but details are improved remarkably. The main purpose of this paper is to address the matter of transforming this algorithm from source code on MATLAB into a digital design which may operate correctly on FPGA with a minimum amount of resources.

II. DETAIL ENHANCEMENT ALGORITHM

The algorithm is divided into three steps. The purpose of the algorithm is to convert 14-bit infrared images with low contrast, poor detail, to 8-bit images but the details are clearer, higher contrast. The data processing steps are described as shown in Fig. 1.

Guided filter (GF) is proposed by Kaiming He in 2010 [10] and can be used to smooth images but preserving edges.

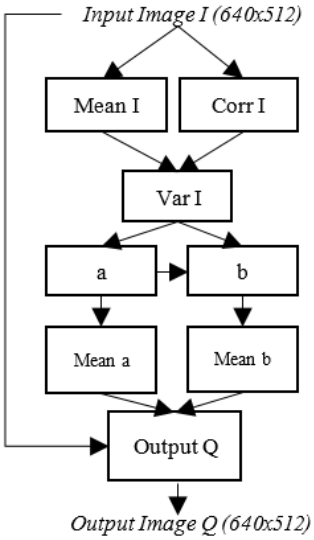

Fig. 2: The order of guided filtering procedure (The arrows in the figure indicate the relationship between interdependent quantities)

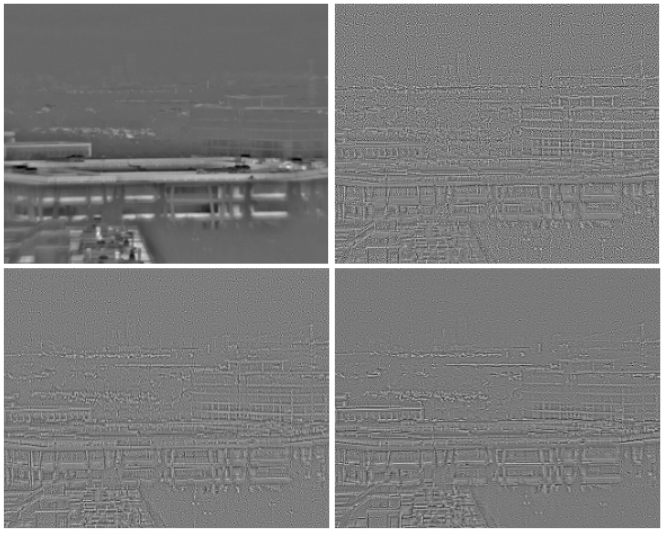

Fig. 3: Input image and I_D with $\varepsilon_2 = 100$, $\varepsilon_2 = 500$ and $\varepsilon_2 = 1000$ with $\varepsilon_1 = 50$. Small ε_2 values retain more details and noise than the big ones.

The sequence of this filter may be summarized as Fig. 2. To compute the mean value of an input image, we apply a kernel which has the size of 3×3 and this is the same as the two other steps of calculating $mean_a$ and $mean_b$ values. The correlation is also figured out by this filter except all pixels are squared from the original image. These are formulas needed to compute the output image Q accompanied by convolution operators:

$$a = var_{I.}/(var_I + \varepsilon) \tag{1}$$

$$b = mean_I - a_. * mean_I \tag{2}$$

$$Q = mean_{a_.} * I + mean_b. \tag{3}$$

We apply two guided filters corresponding to different parameters ε_1 and ε_2. The first guided filter with a small value ε_1 to reduce noise from the input image but still preserving small details and the output is denoted as I_{B1} in Fig. 1. The second one with the bigger value ε_2 will only maintain strong edges, which used to produce the base layer I_B. By subtracting I_B from I_{B1}, we obtain the detail layer I_D. The effects of $\varepsilon_2 = 100$, $\varepsilon_2 = 500$ and $\varepsilon_2 = 1000$ on I_D when ε_1 is fixed are demonstrated in Fig. 3. Besides, we consider the differences of several typical kernel sizes such as 3×3, 5×5 and 7×7. Fig. 4 shows the capability of detail extraction between those window sizes. In our implementation, we choose the kernel size 3×3 because a small kernel size with a large parameter ε_2 will create a satisfied detail layer I_D. Moreover, this will save a significant amount of resources on FPGA in deployment.

After that, we forward the output I_B through a module performing compressing from 16384 to 256 gray levels. This operation is described as Fig. 5. Specifically, we obtain a histogram from the 14-bit input image which has passed through the guided filter and then choose an appropriate threshold to decide whether the value at each bin of that histogram is zero or one. The next step is to compute cumulative summation for each gray level after thresholding and all values of this cumulative summation series are normalized by a factor which equals to the number of bins assigned value 1 in the earlier step so that coefficients after this procedure are between 0 and 1. The pseudo-code of this algorithm may be summarized as:

Algorithm 1 Histogram Projection

Step 1: *Hist = histogram (input image)*
Step 2: *HE = Hist* $\geq$ *Thresh*
Step 3: *HE* $= \frac{cumsum(HE)}{sum(HE)}$; *range = sum(HE)*
Step 4:
if *range* ≥ 255 **then**
    $Y = 255 \times HE$
else
    $Y = \frac{255 - range}{2} + range \times HE$
end if
We can see that Y is an array or a vector consist of 16384 values. In order to acquire the output image, replace pixels from the input image by the value at corrresponding index of Y
Step 5:
For i=1 to length(*Hist*) **do**
    Image Out$(ImageOut == (i-1)) = Y(i)$
End

From the formula (1), we can see that a is large when var_I is much bigger than ε and this will happen at edges of objects in the input image. We use $mean_a$ values from two filtering

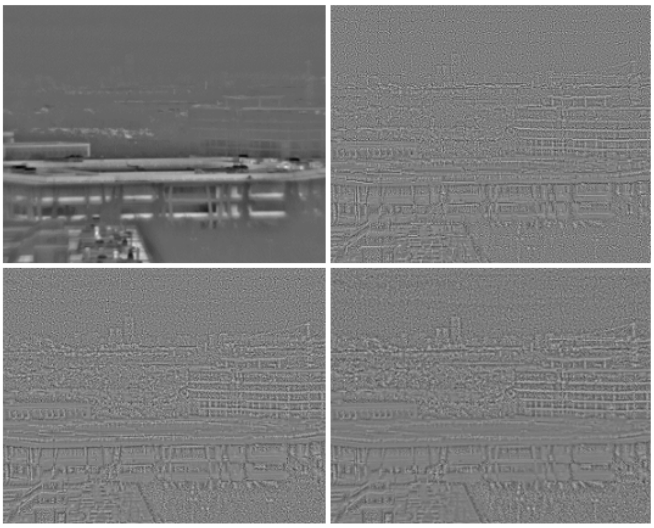

Fig. 4: Input image and I_D with different kernel sizes of guided filter (3×3, 5×5 and 7×7).

processes and denote them as α, β in Fig. 1. We use these coefficients in computing I_{DP}, in which γ is a configurable parameter. The result of $\alpha \times \beta$ is called a gain mask. We demonstrate different gain masks corresponding to $\varepsilon_2 = 100$, $\varepsilon_2 = 500$ and $\varepsilon_2 = 1000$ when $\varepsilon_1 = 50$ in Fig. 6. Next, we combine I_{BP} and I_{DP} by an addition operator. After adding them together, we have an image with pixel values which might be greater than 255. Again, we apply histogram projection one more time to compress the output image into range $0, \ldots, 255$ and then receive the final result which is an 8-bit output image with a better quality compared to the original image.

III. PROPOSED THREE-STEP ALGORITHM IMPLEMENTATION ON FPGA

Over the last section, we understand that two main tasks of the detail enhancement algorithm are guided filters and histogram projection. Firstly, we start studying about how to perform a filter on FPGA. There are a number of different methods, however, we have selected the one which is nearly similar to the principle presented in [11]. Fig. 7 and Fig. 8 is the outline of our design. As you can see, to perform convolution with a kernel of size 3×3, we need a line buffer (Fig. 7) to store input pixels. Every clock cycle, a new pixel enters this line buffer and each pixel which has been put into previously is shifted to the right one step or moved down the next line. We pull out a window of 3×3 pixels from these lines to calculate the mean value of them as indicated in Fig.8, where adders and multipliers have one clock cycle latency. It is obvious that we may duplicate this kind of architecture for other filters in our design. In fact, as described in section II, we have four separate mean filters which used to compute the mean and correlation values of an input image, as well as $mean_a$ and $mean_b$ in the next step. By taking advantage

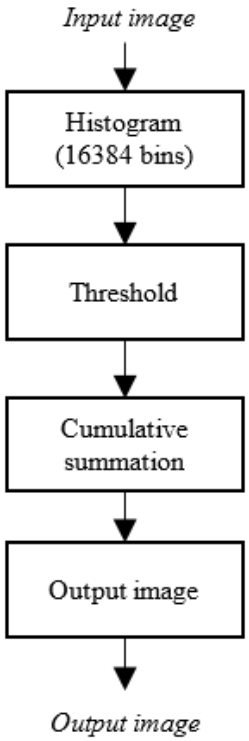

Fig. 5: The steps of histogram projection.

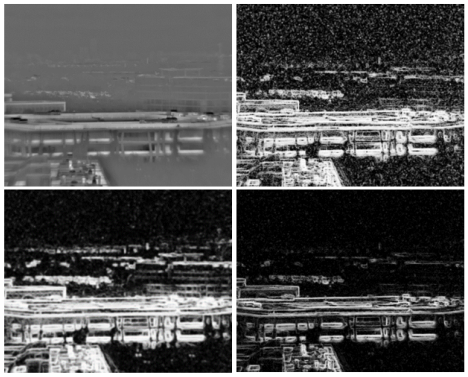

Fig. 6: Different gain masks with different parameters $\varepsilon_2 = 100$, $\varepsilon_2 = 500$ and $\varepsilon_2 = 1000$. The bright region means that gain mask value is big.

of a well-designed filter, we may avoid time-consuming in debugging and concentrate on optimizing fixed-point numbers in order to acquire an acceptable precision but with the modest amount of resources. To avoid wasting logic resources on FPGA, we only build one block for computing value var_I of both guided filters, and then design two separate calculation blocks for a and b values with different parameters ε. In fact, ε_1, ε_2 and γ are configurable parameters and may be modified by users, who integrate our IP core into their overall system, so we do not make those values fixed in our design.

Secondly, we build a module for histogram projection.

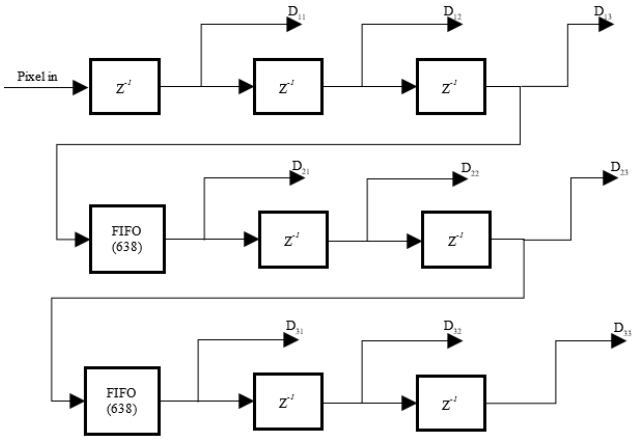

Fig. 7: Line buffer for convolution with kernel 3×3.

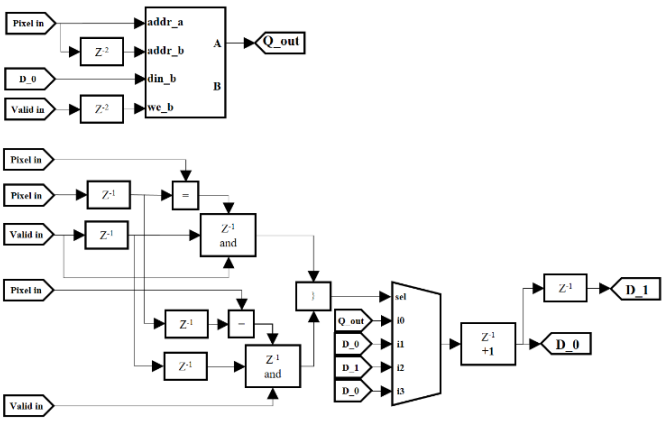

Fig. 9: The diagram of histogram calculation block.

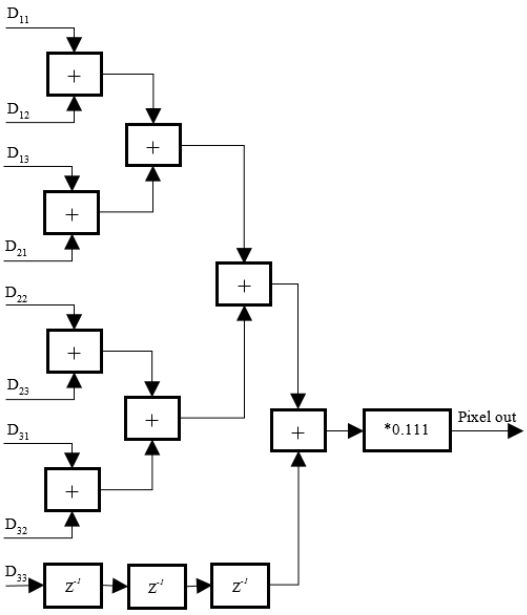

Fig. 8: Pipelined additions and multiplication.

Fig. 10: Waveform created from the diagram in Fig. 9. This figure shows that our design can resolve situations that many pixels which have the same value appear in continuous clock cycles.

The histogram of an input image may be determined and saved to a Block RAM while input pixels are coming in, as the random access characteristic of RAM is suitable for this work. Nevertheless, we also need to pay attention that it takes three clock cycles to calculate for each input pixel including three operations: read the current value from RAM, increase by one and write the added value back. On one side, there are a number of circumstances in this process that new coming pixels are equal to the previous ones which are in progress of adding or writing their new values to the memory, thus we need to check and handle those situations. We summarized our solution in Fig. 9, which will give us the corresponding waveform in Fig. 10. On the other side, Fig. 11. illustrates necessary memory blocks to store information while implementing this algorithm. The reason why we use two distinct Block RAMs is that cumulative summation is a time-consuming activity because with 16384 bins of the histogram, we have to spend approximately that number of clock cycles to read values from the memory, compare those with a threshold value then calculate and write results of them to another Block RAM, which stores only cumulative summation series. In this period, we also have to clear each memory cell of BRAM containing the histogram to zero right after handling of that cell is completed. Therefore, we utilize two Block RAMs alternating each other to avoid conflict in case new pixels appear during those steps. As a result, we can deploy our detail enhancement algorithm on FPGA without any delay.

Up until this point, we have explained almost all of major architectures in our design. Another outstanding idea which we applied is that we use the histogram of the previous frame to compute the output of the current frame. As described above, assuming that we wait for cumulative summation finished, we need to buffer the current frame and stop receiving new coming pixels. Because of the large latency of this operation, we use the histogram of the last frame to execute histogram projection for the ongoing one. We have come up with this

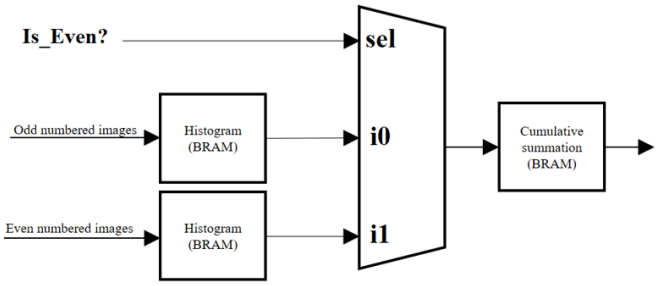

Fig. 11: The diagram of histogram selection.

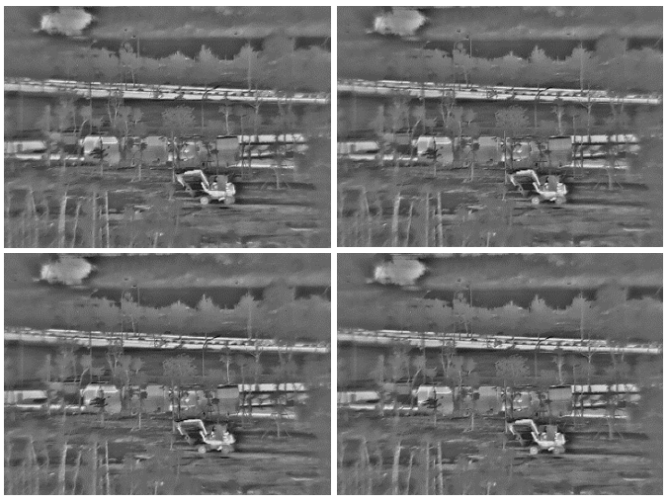

Fig. 12: Continuous frames at the second histogram projection step.

idea and investigated effects that might happen before finally agreed this approach. Provided that our system is running at speed 60 or even up to 120 fps, the difference of the histograms between continuous frames is trivial, as you can see an example in Fig.12-14. As a result, every operation in our design is pipelined and the processing speed depends on the speed of input pixels, as well as the maximum clock frequency that our IP core may achieve without any timing error.

IV. RESULTS

We have implemented this detail enhancement algorithm and packaged into an IP Core compatible with AXI video streaming interface standard. We experimented our design on Zybo Z7 board of Digilent and in Vivado 2018.3 environment. According to acquired reports after running synthesis and implementation steps, the statistics about different kinds of resources that our IP Core occupies are listed in Table.1. Our design can achieve the maximum clock frequency up to 125 MHz, which may process one frame of size 640×512 in approximately 2.62 millisecond, provided that input pixels are pushed into continuously every clock cycle. Due to the fully-pipelined architecture, while performing calculation on the current frame, our IP core may receive and process the next

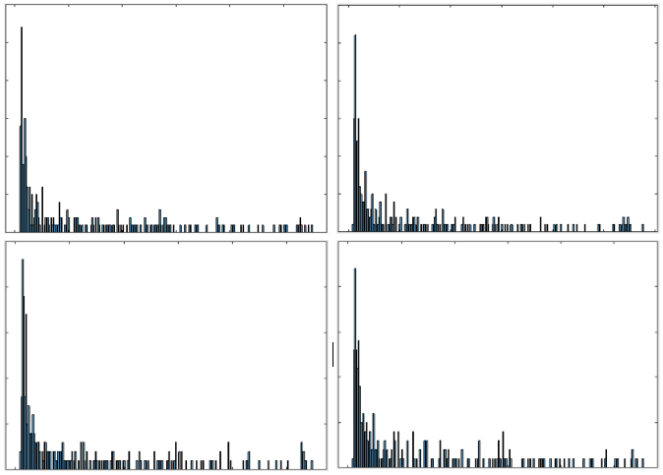

Fig. 13: The histograms of continuous frames corresponding to Fig. 12.

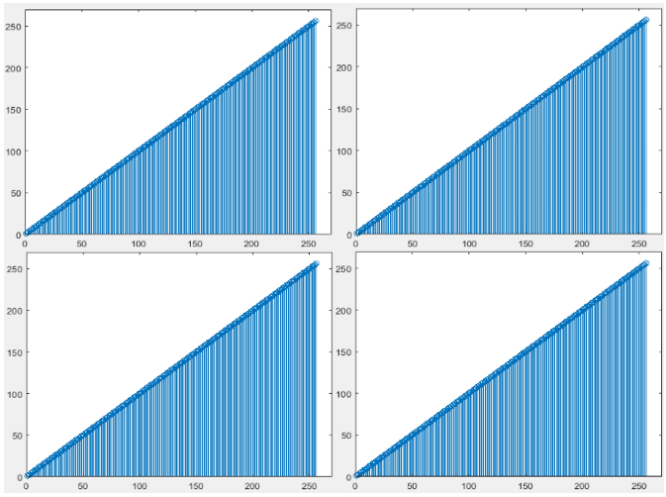

Fig. 14: The cumulative summation series of continuous frames corresponding to Fig. 12.

frame when it comes, so we do not need to wait for the current one finished completely. Therefore, our design may operate in systems that require a high frame rate.

In terms of visual representation, our algorithm significantly improves image quality and can well support observations as shown in Fig. 15.

We also consider the effect of guided filters on the output image. In the Fig. 16, we compare the output in two cases, either only histogram projection or complete detail enhancement algorithm from the input image.

V. CONCLUSION

In conclusion, detail enhancement algorithm presented in this paper not only improves the contrast of infrared images significantly but also achieves real-time capability when deployed on FPGAs. Using histogram information of the

TABLE I: Statistics of resources for detail enhancement IP core

Stage	Slice LUTs (53200)		Block RAM (140)		DSPs (220)		Slice Registers (106400)	
	No	**%**	**No**	**%**	**No**	**%**	**No**	**%**
Post synchronization	12782	24.03	31	22.14	48	21.82	18818	17.69
Post implementation	10079	18.95	31	22.14	48	21.82	16533	15.54

Fig. 15: The input and output of detail enhancement algorithm. (left: original 14-bit images, right: output 8-bit images).

Fig. 16: Only histogram projection (left) and full detail enhancement (right).Detail enhancement with guided filters give us the better quality image.

previous frame to compute the output for the current frame both eliminates delay due to waiting and makes our design fully-pipelined. Finally, the amount of logic resources needed to integrate our IP core, as well as the maximum clock frequency which is achievable, is absolutely feasible when implemented in practical embedded systems.

REFERENCES

[1] V. D. Nguyen, V. L. Pham, V. X. Hoang, H. D. Han, H. T. Nguyen and T. H. Nguyen, "Implementation of an OFDM system based on the TMS320C6416 DSP," *International Conference on Advanced Technologies for Communications*, Hai Phong, pp. 74-77, 2009.

[2] T. H. Nguyen, T. H. Nguyen, T. Yoon, W. Jung, D. Yoo and S. Ro, "An ICI Suppression Analysis Testbed for Harbor Unmanned Ground Vehicle Deployment," in *IEEE Access*, vol. 7, pp. 107757-107768, 2019.

[3] Abduallah M. Alsuwailem, and Saleh Alshebeili , "A new approach for Real Time Histogram Equalization using FPGA," *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, 2005.

[4] Sachdeva, Nitin, and Tarun Sachdeva, "An FPGA based Real-time Histogram Equalization Circuit for Image Enhancement," *IJECT vol. 1, no. 1, December 2010, pp. 63-67*.

[5] Sundaram Sowmya, and Roy P. Paily, "FPGA implementation of image enhancement algorithms," *2011 International Conference on Communications and Signal Processing*.

[6] Priyanka Saini, Adesh Kumar, and Neha Singh, "FPGA Implementation of 2D and 3D Image Enhancement Chip in HDL Environment," *IJCA vol. 62, no. 21, January 2013*.

[7] Sangjin Kim, Wonseok Kang, Eunsung Lee, and Joonki Paik, "Wavelet-Domain Color Image Enhancement Using Filtered Directional Bases and Frequence-Adaptive Shrinkage," *IEEE Transactions on Consumer Electronics*, 1063-1070, June 2010.

[8] Tarek M. Bittibssi, Gouda I. Salama, Yehia Z. Mehaseb, and Adel E. Henawy, "Image Enhancement Algorithms using FPGA," *International Journal of Computer Science and Communication Networks*, Vol 2(4), 536-542.

[9] Ning Liu, and Dongxue Zhao, "Detail enhancement for high-dynamic-range infrared images based on guided image filter," *Infrared Physics & Technology*, Volume 77, July 2016, Elsevier.

[10] Kaiming He, Jian Sun, and Xiaoou Tang, "Guided Image Filtering," *IEEE Transactions on Pattern analysis and Machine intelligence*, vol. 35, 2013.

[11] Abdullah Al-Dujaili, and Suhaib A. Fahmy, "High Throughput 2D Spatial Image Filters on FPGAs," *arXiv 2017, arXiv:1710.05154*.