FPGA HW/SW Codesign Approach for Real-time Image Processing Using HLS

Conference Paper · May 2020

DOI: 10.1109/CCSSP49278.2020.9151686

6 authors, including:

Mohamed Salah Azzaz
Ecole Militaire Polytechnique
62 PUBLICATIONS **414** CITATIONS
SEE PROFILE

Abdelmadjid Maali
Ecole Militaire Polytechnique
25 PUBLICATIONS **106** CITATIONS
SEE PROFILE

Redouane Kaibou
Ecole Militaire Polytechnique
9 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Ibrahim Kakouche
Ecole Militaire Polytechnique
6 PUBLICATIONS **5** CITATIONS
SEE PROFILE

FPGA HW/SW Codesign Approach for Real-time Image Processing Using HLS

M. S. Azzaz
Ecole Militaire Polytechnique
Laboratoire SEN
Algiers, Algeria
ms.azzaz@gmail.com

A. Maali
Ecole Militaire Polytechnique
Laboratoire SEN
Algiers, Algeria
abdelmadjid.maali@esiee.fr

R. Kaibou
Ecole Militaire Polytechnique
Laboratoire SEN
Algiers, Algeria
kaibouredouane@gmail.com

I. Kakouche
Ecole Militaire Polytechnique
Laboratoire SEN
Algiers, Algeria
kakouhim@gmail.com

M. Saad
Ecole Militaire Polytechnique
Laboratoire SEN
Algiers, Algeria
saadtelecom0@gmail.com

H. Hamil
Mouloud Mammeri University
LAMPA Laboratory
Tizi-Ouzou, 15000, Algeria
hocine.hamil@ummto.dz

***Abstract*—Real-time constraint is one of the most common challenge found in many critical embedded applications, namely image and video processing. However, software tools such as Matlab and general purpose microprocessor are not suitable for deals with such as problems. Recently, FPGA reconfigurable circuit with its HLS software tools is become a very promising technology to be widely used in many applications encompassing all aspects and requirements of embedded system. This paper presents the implementation of morphological image operation including dilation, erosion and linear filtering. The implementation method is based on Hardware/Software (HW/SW) codesign approach using High Level Synthesis (HLS) tools, Xilinx Vivado 15.2 and SDK 15.2. Zedboard FPGA platform with Zynq device is used for this work in which is connected to a PC through Ethernet link. Captured image by Webcam is transmitted to the FPGA for processing and will be then returned to the PC to be displayed on the IHM interface, in real-time. Experimental results demonstrate the feasibility of the proposed approach and it can be extended for other applications.**

***Index Terms*—FPGA, HW/SW, Codesign, HLS, Xilinx, Image, Morphological**

I. Introduction

Digital image and video processing is actually considered as very dynamic area with critical applications often encountered in our daily lives, such as military and commercial drones, computer vision, surveillance of sensitive areas, identification in access control, automated inspection of the industry and many other areas [1]–[3]. Although some applications do not require a high data processing capacity, while other embedded applications require strong constraints such as real time, power, cost, and so on [4]. The processing complexity of these restrictive algorithms make their implementation on conventional general purpose processors not feasible and are best suited to hardware implementation [5]. The great advances in VLSI (*Very Large Scale Integrated*) technology make hardware implementation a very attractive alternative for many embedded applications [6], [7]. The intrinsic properties of these hardware circuits namely parallelism and pipeline will offer users the ability to implement a complex algorithms to overcome the data processing problems previously discussed especially real-time constraint.

The properties of FPGA (*Field Programmable Gate Array*) make them a very desired technology to implement digital image and video processing. This alternative is proved its performance as an material accelerator in various fields [8]–[10]. However, the development time and human resources for a hardware implementation are more important than those of a software implementation. This due to the complexity of implementation by using *Hardware Description Language* (Verilog and VHDL), which allows designers to develop at various levels of abstraction. Technological advances towards Xilinx family of FPGA 7 series such as ZedBoard and hardware development time constraints have prompted FPGA chip designers to introduce new software design tools that operate at a high synthesis level [11], [12]. These tools facilitate to the users the hardware implementation and often incorporate a C/C++ language. HLS (*High Level Synthesis*) Vivado [13]–[15] is one of the most used tools which can directly transform a C description to a hardware IP, described on Verilog or VHDL (RTL: *Register-Transfer Level*). The performance of generated HDL code can be compared to hand-coded RTL in terms of processing time and resources.

Given the importance of digital video processing and their significant implementation on reconfigurable FPGA device to achieve the requirements of embedded systems, this work addresses the implementation of image and video processing algorithms like morphological and convolution operation on FPGA using co-design approach and HLS Vivado tools, in which the objective is to show the feasibility of designing critical embedded applications on FPGA platform by using HW/SW co-design approach and high-level synthesis tools that allow to have a good compromise in terms of performance and development time.

The rest of the paper is organized as follow: Section II

presents the the modern Xilinx technology Zynq SoC. Image and video processing implementation results are given in Section III. Section IV concludes this work.

II. Zynq FPGA device and Vivado tools

A. The new Xilinx technology Zynq SoC

Currently, with the great advances in the embedded system design, a new kind of devices are provided by some leader of FPGA designer namely Xilnx. Xilinx has presented the new FPGA Zynq, whereas Altera has produced Altera SoC series. Both technologies have the same structure and performance. This work is focused on the Xilinx Zynq device. Figure 1 provides a very ideal platform for the implementation of System on Chip (SoC): Xilinx markets the device as "All-Programmable SoC" (APSoC), which captures perfectly his performances in terms of size and cost. The SoC include different components namely memories, A/D and D/A converters, Processors, I/O blocks, etc. Figure 1 illustrates a high-level model of the Zynq architecture. Zynq consists of two main parts: a Processing System (PS) that is built around a dual-core ARM Cortex-A9 processor and a Programmable Logic (PL) equivalent to hardware accelerator (FPGA). The connection between the PL and the PS are several, programmable and allow fast communication of transmitted data. While, in the old FPGA technology the separation between the processor and the PL limit these connections and the speed of data transmission.

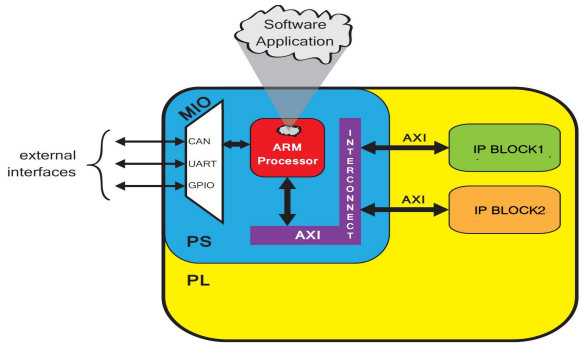

Fig. 1. An overview of the Zynq SoC.

There exist different kinds of programming the FPGA zynq device: programming the Processing System (Software), programming the Programmable Logic (Hardware) or co-design HW/SW programming (PL/PS). Connections between PL and PS parts are made using industry-standard connections AXI (*Advanced eXtensible Interface*) [16].

The image and video processing in this work is done on the Zedboard platform, because its performance and it can serves as a unique platform for Embedded system design as well as hardware/software co-design.The ZedBoard shown in Fig. 2 is a low-cost development platform, which has a Zynq XC7Z020 device. It is a co-production between Xilinx, Avnet (distributor) and Digilent (the card manufacturer). Although, suitable as a development platform for the industry, the ZedBoard is also aimed at students, academics and amateurs, with specific materials to meet new users of Zynq, and addressing the curve.

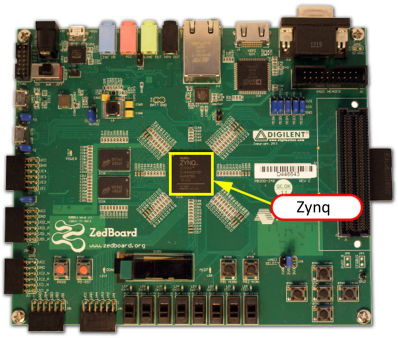

Fig. 2. The Zedboard platform.

According to Xilinx, the Artix-7 FPGA on the Zynq 7020 device has about 85k Logic Cells, 53,200 Look-Up Tables, 106,400 Flip-Flops, 220 Programmable DSP Slices and 560 KB Block RAM

B. FPGA Zyng design tools

In order to develop applications on FPGAs, software tools are provided by Xilinx, offering several functionalities, making it possible to exploit the platform's peripherals, and programming the latter in order to achieve the desired operation by the application:

Vivado IDE (*Integrated Development Environment*) is an integrated development environment for the design of the hardware architecture of the system (processor, memory, peripherals, external devices, bus connections, etc.).

SDK (*Software Development Kit*) is a development environment for the software part of the system, using the C and C++ programming languages. It supports all IPs provided by Xilinx. SDK provides tools for compiling, debugging and running the software application.

HLS (*High Level Synthesis*) is a rapid prototyping tool, which allows the transformation of a code written in C, C++ or SystemC into a Register Transfer Level (RTL) code (Fig. 3), which can then be synthesized and implemented on the programmable logic a Xilinx FPGA or Zynq device.

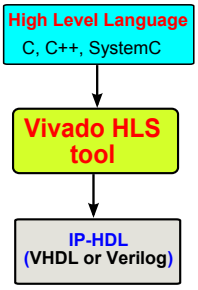

Fig. 3. Vivado HLS tool.

IP blocks designed using HLS are intended to be implemented in the PL (Programmable Logic) part of the Zynq

target device. There may be many of these modules in a Zynq system design, and the important task of this process is to interact them appropriately with the rest of the design (using specific connections). The Vivado HLS tool provides both a graphical user interface (GUI) and Command Line Interface (CLI), which can be used separately or in combination with each other, depending on the user preferences.

System Generator is also used for fast prototyping as a DSP design tool from Xilinx. This tool allows to use Simulink Mathworks model-based design environment for implementing on FPGA. This is approach is based on the configuration the system to be implemented by using of blockset on Simulink Matlab, which basically try to convert C or Matlab code into a digital circuit.

C. Design Flow for Zynq SoC

The design flow for Zynq SoC is shown and summarised in Fig. 4.

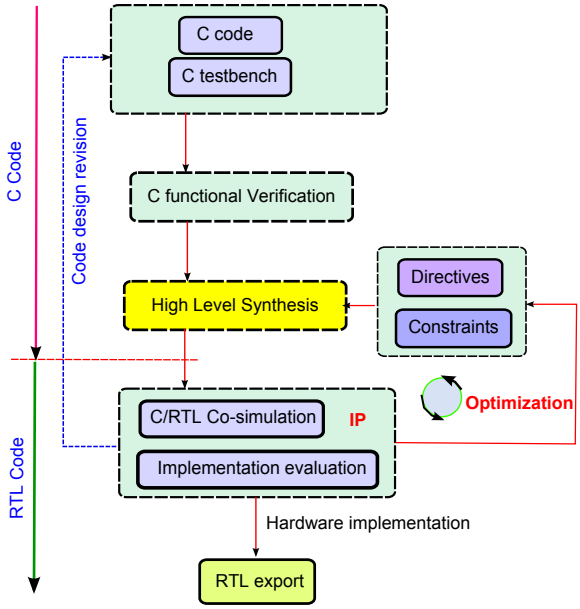

Fig. 4. Vivado HLS design flow.

HLS Process inputs

The description input for HLS is C/C++/SystemC language. This description is associated with a testbench, which can verify the correct operation of the main program.

Functional Verification

The C code design with its C testbench are compiled to generate a set of output vectors which are known to be correct.

High-Level Synthesis

The next step consists of introducing constraints and directives necessary for the RTL description of the circuit. Before starting the HLS process, the user can configure his preferred HDL language and also the simulator. Once the process of the HLS is completed, HLS tool generate an IP with a set of output files, including design files in RTL language.

C/RTL Cosimulation

Once HLS has generated the RTL IP, the next step concern also the design verification phase via C/RTL cosimulation in Vivado HLS. The inputs introduced in the C testbench code will be act directly on the RTL code and the simulator is used then to check the outputs of the design.

Evaluation of Implementation

After the generating the IP design and simulate it using C/RTL cosimulation, the design will be then constrained (HLS constraints and directives) and implemented on the FPGA target. This step of implementation will be then evaluated using some metrics namely required resources (LUT, FF, DSP, BRAM, etc.), Latency, Maximum frequency, power consumption and so on.

Optimization

This step consists of a feedback between the obtained results of the design in terms of performances and the C code design. So, if the the solution present a non sufficient performances, an optimization phase can be considered by introducing some refinement and review on the C code of the original description. In order to get a best solution, multiple HLS iterations may be undertaken using modified directives and constraints.

RTL Export

Once the the best solution of the design has been validated by multiple HLS iterations (optimization), to get the desired performances, this design will be integrated as an IP into a another system. This integration can be achieved by using directly the generated RTL sources (VHDL or Verilog) or by packaging the design to be integrated on the library of other xilinx tools within Vivado IDE and System Generator.

III. Image and Video Processing on Zynq SoC

Through this section, the different algorithms implemented on co-design will be shown and compared them with other purely software implementation . The synthesis results on the Zedboard platform will be also given by specifying the both hardware and software architectures. The proposed solution will be then realized by real time video processing application on Zynq SoC. The steps taken to achieve this experimental goal are as follows:

- Simulation and design of the IP core architecture created for these algorithms.
- Performance evaluation and analysis of IP core synthesis results.
- Integration of the IP core into the overall system architecture and comparison with an embedded processor implementation.
- Application of the proposed architecture to real-time video processing.

A. Convolution algorithm

The convolution theorem that applies to a digital image I, is given by the Eq. 1. The output is a new modified filtered image.

$$g(x,y) = (I * h)[x,y] = \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} h[i,j] \cdot I[x-i, y-j] \quad (1)$$

where $g(x,y)$ is the filtered image, $I(x,y)$ is the original image and $h(x,y)$ is the filter kernel. To perform a two dimensions convolution will require:

- Define the filter core;
- Drag the kernel over the entire image so that the center of the nucleus coincides with the pixel to be processed;
- Multiply the values of the pixels under the kernel by the corresponding values (weight) and sum the total;
- Copy the new value to the same place in a new image.

The flowchart of the adopted 2D convolution algorithm in this design is shown in Fig. 5.

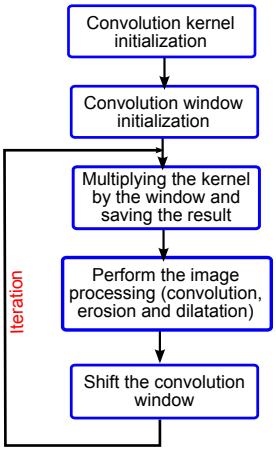

Fig. 5. Flowchart of the adopted 2D convolution algorithm.

The convolution operation requires, for an iteration, a certain number of pixels for it to be performed (convolution window), these pixels will be received by the hardware structure, which will handle the convolution, through the part software, for this, an input and a output of type "Streaming" are provided for this structure. These two ports will respectively handle the input and output of the pixel stream. Once the stream received, buffers, which have the same size of the window, will be positioned in the latter to perform the operation on the first pixel, as shown in Fig 6.

The values of the buffers will be then multiplied by the convolution mask, pixel by pixel, and the result will be stored in another window, having the same size as the mask, as shown in Fig. 7. the operation that will be chosen, the IP proceeds to a certain number of methods to return to the output the value of the appropriate pixel:

- *Convolution*: We perform the pixel-by-pixel sum of the resulting window, and the value of the sum will be returned.

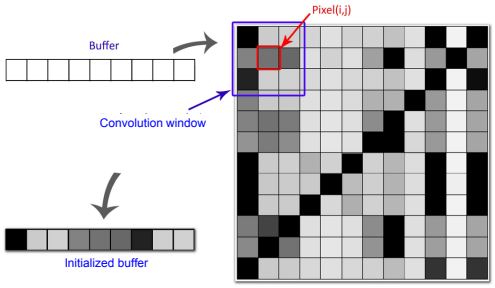

Fig. 6. Process of filling the buffer.

- *Erosion*: extract the maximum value of the resulting window and it will be returned.
- *Dilation*: extract the minimum value of the resulting window and it will be returned.

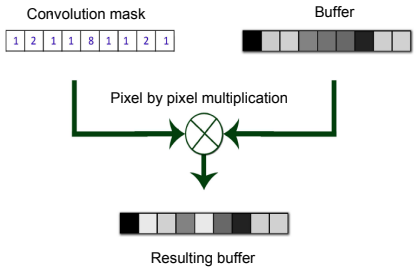

Fig. 7. Process of multiplying the buffer by the mask.

B. HLS Vivado design and simulation

The convolution algorithm was described using C language and the HLS vivado tool has created its RTL (VHDL) IP core, which is shown in Fig. 8.

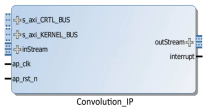

Fig. 8. Designed IP convolution on HLS vivado.

The **inStream** and **outStream** ports are respectively for receiving and sending the pixel stream. The **CRTL BUS** port is for the selection of the operation to be performed. The **KERNEL BUS** port is for receiving the convolution mask to perform. The simulation of this IP under Vivado simulator is represented in Fig. 9.

Simulation results under Vivado simulator of the designed IP convolution core showed the correctness of its outputs. The FPGA synthesis results of this IP are depicted in Fig. 10 and 11.

According to the synthesis results of the generated IP core, it can be seen that the estimated clock for the latter is 8.34, and the expected latency for the IP is 691449 clock cycles, which

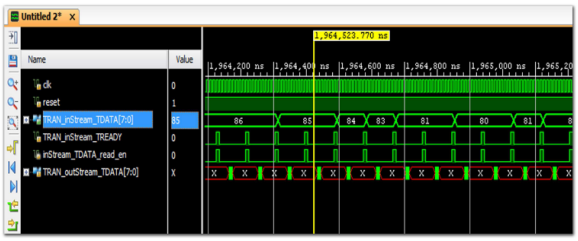

Fig. 9. Vivado simulation result of the designed IP convolution.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.34	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
691448	691448	691449	691449	none

Detail

Fig. 10. Timing and latency synthesis results of the designed convolution IP on Zynq.

is equivalent to an execution time of 7 μs at a frequency of 100 MHz, which represents a very fast execution time. This time constraint is very required in real-time applications. On the other hand, the generated IP requires low FPGA resources in terms of DSP, BRAM, FF and LUT, as shown in Fig. 11, which makes it possible to integrate it into the global architecture of any system requiring 2D convolution. Thus we will implement it later in the hardware architecture of our video processing system under Vivado IDE.

C. Proposed hardware architecture

The proposed image and video processing on Zynq SoC architecture under Xilinx Vivado tool is illustrated on Fig. 12.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	9	0	802
FIFO	-	-	-	-
Instance	2	-	184	214
Memory	3	-	0	0
Multiplexer	-	-	-	329
Register	-	-	874	-
Total	5	9	1058	1345
Available	280	220	106400	53200
Utilization (%)	1	4	~0	2

Fig. 11. Resources synthesis results of the designed convolution IP on Zynq.

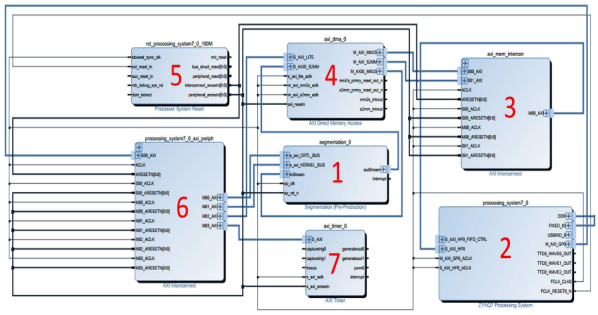

Fig. 12. Proposed hardware image architecture for video processing on Zynq SoC.

As shown in Fig. 12, the IP includes two ports of type "**Stream**", it must work with another IP, type DMA (Direct Memory Access), the latter has the advantage of being directly linked to the RAM of the FPGA platform, it will handle the communication of the incoming and outgoing pixel stream of our IP convolution in a fast way. The proposed architecture also includes:

1) Block [1]: The convolution IP block is the designed HLS architecture,
2) Block [2]: The Zynq7 Processing System block is the block containing the embedded ARM processor, which will be used for the software architecture,
3) Block [3]: The AXI Interconnect0 block will handle the interconnections between the different blocks of the hardware architecture.
4) Block [4]: The AXI DMA block will read and write the pixel stream between the IP core and the RAM.
5) Block [5]: The Processor System Reset block will be responsible for the reset of the entire system and global and local interrupts.
6) Block [6]: The AXI Interconnect1 block which will take care of the interconnections between the different blocks of the hardware architecture.
7) Block [7]: The Axi Timer block will be responsible for measuring the time taken at runtime.

The synthesis results after placement and route of the co-design architecture proposed on FPGA Zynq type are shown in Fig. 13. These results demonstrate that the proposed co-design approach for image and video processing through a convolution design require a low FPGA resources in terms of LUT (9 %) BRAM (3 %) and FF (5 %).

In order to evaluate the design in terms of time, a comparison between the software approach and the codesign approach is summarized in Table I for morphological processing (*convolution*) and for different known kernel mask: *erosion*, *dilatation*, *impulse*, *Stamping* and *contour*.

The Table I shows the difference of the implemented algorithms between co-design and software approach, in terms of execution time. This difference is due essentially to the number of clock cycles performed required during the execution of the implemented algorithms, in fact an hardware approach can

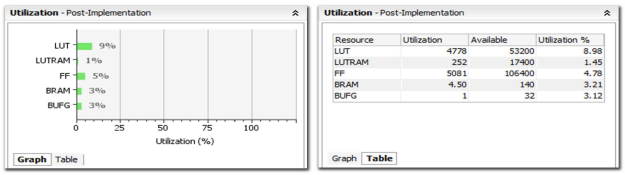

Fig. 13. Synthesis results of the proposed hardware image architecture for video processing on Zynq SoC.

TABLE I
TIME PROCESSING COMPARISON

Operation		Run time (s)	
		Soft. (sw)	codesign (hw/sw)
Morphological proc.	Erosion	0.024350	0.007337
	Dilatation	0.023998	0.007337
Convolution	Impulse	0.023765	0.007339
	Stamping	0.027787	0.007337
	Contour	0.027789	0.007338
Mean time		0.025929	0.007337

execute all the iterations in parallel, which is not the case for the software that works sequentially, which generates a greater number of cycles. So, in this case the co-design time processing represents 3.54 of the software time processing. The FPGA implementation results on image processing are shown in Fig. 14 and 15.

Fig. 14. Image FPGA Morphological processing implementation results.

Fig. 15. Image FPGA convolution processing implementation results.

IV. CONCLUSION

High-performance image and video processing have challenging demands on many real-time embedded applications. In contrary, the software implementation, hardware implementation is usually required as an material accelerator to provide high processing speed. However, hardware approach require a long prototyping time. Therefore, co-design approach presents a good compromise between performances and prototyping time. In this work, a codesign approach is adopted, using Vivado HLS to design an IP convolution for real-time image and video processing. The proposed codesign approach demonstrates its performances in terms of time processing compared to software one. The FPGA implementation results required also a low resources which leads to add other image or video processing blocks.

REFERENCES

[1] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai et al.,"Edge-based Live Video Analytics for Drones," IEEE Internet Computing, 2019.
[2] A. Heredia and G. Barros-Gavilanes, "Video processing inside embedded devices using SSD-Mobilenet to count mobility actors," In 2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI), pp. 1–6, 2019.
[3] H. Kavalionak, C. Gennaro, G. Amato, C. Vairo, C. Perciante, C. Meghini and F. Falchi, "Distributed video surveillance using smart cameras," Journal of Grid Computing, 17(1), pp. 59–77, 2019.
[4] Z. Fang, D. Hong and R. K. Gupta, "Serving deep neural networks at the cloud edge for vision applications on mobile platforms," In Proceedings of the 10th ACM Multimedia Systems Conference pp. 36–47, 2019.
[5] J. Hosseinzadeh and M. Hosseinzadeh, "A comprehensive survey on evaluation of lightweight symmetric ciphers: hardware and software implementation," Advances in Computer Science: an International Journal, 5(4), pp. 31-41, 2016.
[6] P. Dewan, R. Vig, N. Shukla and B. K. Das, "Novel VLSI Architectures for Image Segmentation and Edge Detection Algorithm," International Journal of Computer Applications, 149(10), 2016.
[7] Y. H. Shiau, Y. T. Kuo, P. Y. Chen and F. Y. Hsu, "VLSI Design of an Efficient Flicker-Free Video Defogging Method for Real-Time Applications," IEEE Transactions on Circuits and Systems for Video Technology, 29(1), 238–251, 2017.
[8] O. Reiche, M. A. Özkan, F. Hannig, J. Teich and M. Schmid, "Loop parallelization techniques for fpga accelerator synthesis,". Journal of Signal Processing Systems, 90(1), pp. 3-27, 2018.
[9] F. Siddiqui, S. Amiri, U. I. Minhas, T. Deng, R. Woods, K. Rafferty and D. Crookes, "FPGA-Based Processor Acceleration for Image Processing Applications," Journal of Imaging, 5(1), 16, 2019.
[10] S. van der Vlugt, H. A. Ara, R. de Jong, M. Hendriks, R. G. Marin, M. Geilen and D. Goswami, "Modeling and analysis of FPGA accelerators for real-time streaming video processing in the healthcare domain," Journal of Signal Processing Systems, 91(1), pp. 75-91, 2019.
[11] K. Khaled, C. Osama, M. Osama, H. Magdy, H. Mahmoud, Y. Hossam et al., "Interfacing USRP Kit with Zynq-7000 Evaluation Kit," In 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST) (pp. 1-4), 2019.
[12] S. Liu, F. Lau and B. C. Schafer, "Accelerating FPGA Prototyping through Predictive Model-Based HLS Design Space Exploration," In Proceedings of the 56th Annual Design Automation Conference 2019 (p. 97), 2019.
[13] A.Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," In Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA), pages 33–36. ACM, 2011.
[14] Vivado Xilinx, "UltraFast Design Methodology Guide for the Vivado Design Suite," User manual, 2015.
[15] F. Aydin, H. F. Ugurdag, V. E. Levent, A. E. Guzel, N. F. R. Annafianto, M. A. Ozkan et al. "Rapid Design of Real-Time Image Fusion on FPGA using HLS and Other Techniques," In 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA) (pp. 1-6), 2018.
[16] L. H., Crockett, R. A. Elliot and M. A. Enderwitz, "The zynq book tutorials for zybo and zedboard," Strathclyde Academic Media, 2015.