Computers, Materials & Continua
DOI:10.32604/cmc.2022.024345
Article

Tech Science Press

An Efficient HW/SW Design for Text Extraction from Complex Color Image

Mohamed Amin Ben Atitallah[1,2,3,*], Rostom Kachouri[2], Ahmed Ben Atitallah[4] and Hassene Mnif[1]

[1]LETI (E.N.I.S.), University of Sfax, Sfax, Tunisia
[2]Laboratory of informatics, Gaspard-Monge, A3SI, ESIEE Paris, CNRS, Gustave Eiffel University, France
[3]National Engineering School of Gabes (ENIG), University of Gabes, Tunisia
[4]Department of Electrical Engineering, Jouf University, Sakaka, Aljouf, Saudi Arabia
*Corresponding Author: Mohamed Amin Ben Atitallah. Email: mohamed.amine@esiee.fr
Received: 14 October 2021; Accepted: 26 November 2021

Abstract: In the context of constructing an embedded system to help visually impaired people to interpret text, in this paper, an efficient High-level synthesis (HLS) Hardware/Software (HW/SW) design for text extraction using the Gamma Correction Method (GCM) is proposed. Indeed, the GCM is a common method used to extract text from a complex color image and video. The purpose of this work is to study the complexity of the GCM method on Xilinx ZCU102 FPGA board and to propose a HW implementation as Intellectual Property (IP) block of the critical blocks in this method using HLS flow with taking account the quality of the text extraction. This IP is integrated and connected to the ARM Cortex-A53 as coprocessor in HW/SW codesign context. The experimental results show that the HLS HW/SW implementation of the GCM method on ZCU102 FPGA board allows a reduction in processing time by about 89% compared to the SW implementation. This result is given for the same potency and strength of SW implementation for the text extraction.

Keywords: Text extraction; GCM; HW/SW codesign; FPGA; HLS flow

1 Introduction

With reference to the World Health Organization, around of the 1.3 billion people live in the world with a few forms of vision impairment [1]. Thus, constructing a personal text-to-speech system aids them to interpret text. In this context extracting text from image or video presents play an important role to develop such system that helps the society and people. For that a robust and accurate algorithm should be selected for text detection. This algorithm should categorize and retrieve all text from multimedia documents in real-time. Otherwise, it can be a vital drawback to extract and recognize the text characters due to their font style, alignment, orientation, textured background, complex colored, etc.

In literature, several methods [2–5] are proposed for text extraction from complex image. These methods use wavelet transform, thresholding, histogram technique, etc. The most important method developed for text extraction is the Gamma Correction Method (GCM) [6]. In fact, the GCM method outperforms the existing methods by suppress completely the background while conserving the features of the text [7]. In this method the Gray-Level Co-occurrence Matrix (GLCM) is calculated to determine the energy and the contrast. These textural features are used to characterize each gamma modified image and extract text from image. Unfortunately, the GCM method provides a high computationally complexity which presents a big issue in computer vision specially for the applications that need a real-time text extraction [8] such as text-to-speech system for helping the blind persons. However, many works in literature are concentrated on accelerating the proceedings of the GCM method by using software (SW) optimization or/and hardware (HW) implementation.

Indeed, Kachouri et al. [8] proposed an accelerated GCM (AGCM) which presents a software optimization of the GCM method. The AGCM proposes a sub-range of values where the chosen gamma should be found per the GCM method. But, the AGCM can affect the quality of the text extraction because the text features are determined from some gamma modified image only. Girisha et al. [9] proposed a Field Programmable Gate Array (FPGA) implementation of the GLCM for 8×8 image size. Each image pixel is presented by 4 bits. So, in this work, a 16×16 GLCM is computed for only a single direction ($\theta = 0°$). Akoushideh et al. [10] presented a HW architecture to implement four 256×256 GLCM matrix for four θ angles (0°, 45°, 90° and 135°) and their texture features. These matrixes are computed in parallel to reduce the execution time. But this architecture can support only 128×128 image size. Besides, the calculation of the texture features is realized based fixed-point operation which can generate an inaccurate result. Maroulis et al. [11] proposed a HW design to calculate sixteen 64×64 GLCMs in parallel and four texture features which are the entropy using a single core, the inverse difference moment, the correlation and the angular second moment. The inconvenient of this design that it uses the fixed-point operations to calculate the texture features instead of the floating-point operations. Further, it does not take in consideration the overlap between the diverse blocks of the image. Boudabous et al. [12] developed two hardware architectures to compute the contrast and energy texture features for GCM method. These architectures are integrated as custom instruction (CI) to the NIOS II softcore processor. Nevertheless, the performance of the developed system is far to work in real-time.

Recently, through the technology advances in system integration based on FPGA, we can see the system design as some functional block with at least one processor as the processing unit. The general idea is to process the complex computing tasks by the hardware part which allows to exploit the pipeline and the parallelism in algorithms, and to operate the software flexibility, resulting in a context of HW/SW codesign [13–15]. Thus, with increasing in the complexity of the FPGA design, it is needed to elevate the design space abstraction level from traditional approach based on Register-Transfer Level (RTL) to an effective approach allowed to decrease the FPGA design complexity. Therefore, in this day, the high-level approach based on software programming language such as C, C++, SystemC is became widely used with FPGA [16–18]. This permits to increase the designer productivity and reduce the run-time in the design flow.

However, the goal of our work is to study and implement in the HW/SW codesign context the GCM method. This method is used for text extraction from a complex color image. In fact, the HLS flow is used to design and implement the complex parts of the GCM method as an intellectual property (IPs) block. These blocks are connected as hardware coprocessor to the hardcore ARM Cortex-A53 and implemented on Xilinx ZCU102 FPGA board in a HW/SW codesign context to increase the reliability and the efficiency of the GCM method for text extraction.

Therefore, this paper presents in Section II the overview of GCM method. In Section III, the complexity study of the GCM method on ZCU102 FPGA board is detailed. In Section IV, the HLS implementation of the complex blocks in the GCM method is described. Section V details the HW/SW implementation of the GCM method. The performance evaluation of the proposed HW/SW GCM design is the subject of Section VI. In Section VII, conclusions are drawn.

2 GCM Overview

The GCM [7] is an important method to extract text from a complex color image by removing the background and extracting the text. Indeed, the GCM method compute from the input image one hundred modified images by variation the γ value from 0.1 to 10.0 using 0.1 as increment step. In fact, as reported in Fig. 1, the pixel intensity of the modified image is depended on the gamma (γ) value. So if $\gamma > 1$ thus the gamma modified image becomes darker, but if $\gamma < 1$ thus the modified image becomes lighter than the original image.

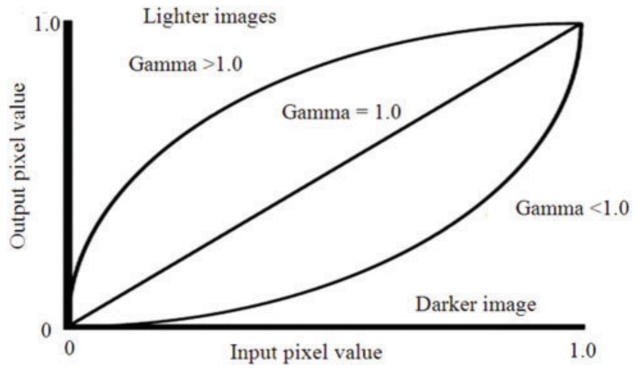

Figure 1: Gamma modified image

The GCM method calculates for each gamma modified image (γ value varies from 0.1 to 10.0 using 0.1 as increment step) four GLCM matrix, the textural features (energy and contrast) and the threshold value to extract the text from the image. Indeed, four GLCM for a given θ angle (θ = 0°, 45°, 90°, 135°) are computed to determine the textural features. In fact, Haralick et al. [19] proposed the GLCM as a second-order histogram statistic. However, the GLCM represents a best algorithm used for the texture analysis. It is calculated through counting all pairs of pixels in a gray level image having (i, j) coordinate for a specified distance (d) and angle (θ). Nevertheless, θ is included in the value range (θ = 0°, 45°, 90°, 135°). For each (i, j) pixel in the image, the direction of the four θ angles is determined as presented in Fig. 2. The GCM uses only the energy (Eq. (1)) and the contrast (Eq. (2)) as texture features from the 14 statistical GLCM features [20] proposed by Haralick et al. The energy describes the smoothness of the image. But the contrast indicates the spatial frequency of the image and a diverse moment of the GLCM. On the other hand, the image threshold value for the gamma modified image is calculated based on Otsu's method [21] which looks for the adequate threshold that decreases the variance within the class defined, as illustrated by Eq. (3), as a weighted addition of variance of the 2 classes. However, this method assumes that the threshold image holds 2 pixels classes (e.g: foreground and background). After, it computes the ideal threshold dissociating these 2 classes where their combined propagation is minimal.

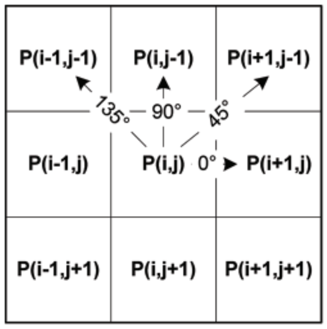

Figure 2: Four GLCM θ angle

$$\text{Energy} = \frac{1}{4}\sum_{\alpha}\left(\sum_{i=1}^{Ng}\sum_{j=1}^{Ng}\left(\frac{p(i,j)}{R}\right)^2\right) \tag{1}$$

$$\text{Contrast} = \frac{1}{4}\sum_{\alpha}\left(\sum_{n=0}^{Ng-1} n^2\left(\sum_{|i-j|=n}\left(\frac{p(i,j)}{R}\right)\right)\right) \tag{2}$$

where Ng is the possible number of gray level value, p(i, j) is the GLCM value, R is a normalization term and α is an angle value.

$$\sigma_w^2(\mathrm{T}) = \omega_1(\mathrm{T})\,\sigma_1^2(\mathrm{T}) + \omega_2(\mathrm{T})\,\sigma_2^2(\mathrm{T}) \tag{3}$$

where σ_1, σ_2 correspond to the inter-class variance. ω corresponds to the probability to be in class one or class two.

Therefore, as detailed in Fig. 3, the GCM applies three rules based on the energy, contrast and threshold to extract the best value of gamma with which it is possible to suppress from the modified image the largest amount of the background and generate the binarized image with the text features only.

Tab. 1 illustrates an example for computing the best gamma value for a test image presented in Fig. 4 (x) which selected from ICDAR Dataset [22]. In the beginning all gamma values have the same probability to be nominated per the GCM method for generating the optimal gamma for text extraction. However, to determine the best gamma for the image in Fig. 4, the rules in Fig. 3 are applied. Hence, the rule 3 is selected because for the $\gamma = 1$, the contrast is less than 1000 and the energy is less than 0.05. In this case, the best gamma value generated to extract the text from the image is equal to 5.5 as depicted in Fig. 4 (y) and Tab. 1. But the best threshold (T) selected to remove the background and convert the gray level image to the binary image is equal to 0.292969 as shown in Fig. 4 (z) and Tab. 1.

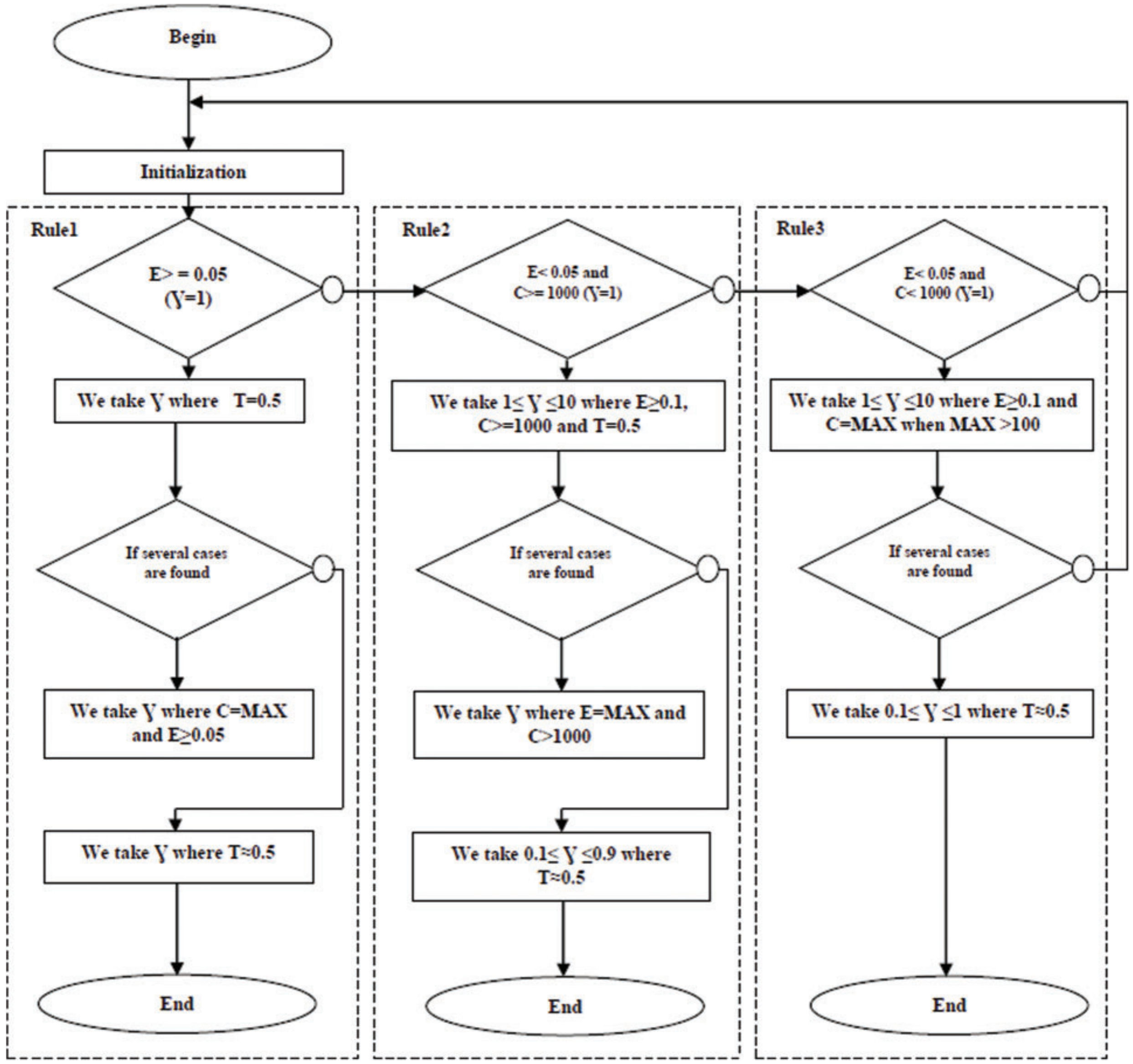
Begin
Initialization
Rule1
E> = 0.05 (Ɣ=1)
We take Ɣ where T=0.5
If several cases are found
We take Ɣ where C=MAX and E≥0.05
We take Ɣ where T≈0.5
End
Rule2
E< 0.05 and C>= 1000 (Ɣ=1)
We take 1≤ Ɣ ≤10 where E≥0.1, C>=1000 and T=0.5
If several cases are found
We take Ɣ where E=MAX and C>1000
We take 0.1≤ Ɣ ≤0.9 where T≈0.5
End
Rule3
E< 0.05 and C< 1000 (Ɣ=1)
We take 1≤ Ɣ ≤10 where E≥0.1 and C=MAX when MAX >100
If several cases are found
We take 0.1≤ Ɣ ≤1 where T≈0.5
End

Figure 3: The GCM rules

Table 1: Example of computing values for test image in Fig. 4

Gamma	Contrast	Energy	Threshold
0.1	12.8241	0.0145233	0.929688
0.2	43.2349	0.00858474	0.871094
...			
5.3	385.552	0.0853631	0.296875
5.4	382.212	0.0997864	0.296875

(Continued)

Table 1: Continued

Gamma	Contrast	Energy	Threshold
5.5	**377.857**	**0.11554**	**0.292969**
5.6	373.654	0.124139	0.292969
...			
9.9	224.941	0.381391	0.210938
10	1926.09	0.384658	0.207031

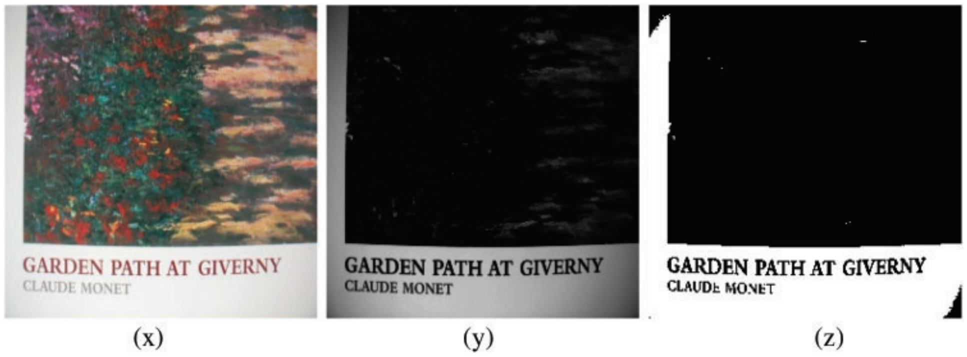

Figure 4: (x) Original image "Contrast = 336.098, Energy = 0.00111243" (y) Modified image "$\gamma = 5.5$" (z) Binarized image "T = 0.292969"

3 Complexity Study of the GCM Method

For the GCM complexity study, the ZCU102 FPGA board [23] is used. This board is based on the Zynq UltraScale + XCZU9EG Xilinx FPGA which contains quad-core ARM Cortex-A53 processor working at 677 MHz. The GCM C code is developed and executed in standalone mode under the ARM Cortex-A53 using the Xilinx Software Development Kit (SDK). In this mode, it should enable the "ff.h" library to manage the writing and the reading of the files from SD card. The execution time of the GCM method is given for the average of several images of size 256 $\times$ 256 pixels from ICDAR Dataset. This time is equal to 8.25 s which cannot be suitable for a real-time application. Therefore, it seems necessary to study the distribution of the CPU time according to the different processing constituting the GCM method in order to design a real-time system for text extraction from complex color image in HW/SW codesign context. The Fig. 5 gives a breakdown of the CPU time of the different blocks of the GCM method.

From Fig. 5, we can see that the complexity is highly localized in three processing blocks (GLCM, contrast and energy). Indeed, the GLCM needs 44% of CPU time. But the contrast and energy require 25% and 24% of CPU time, respectively. Thus, these results help us to make a hypothesis on the hardware and software distribution of the treatments: blocks which must be totally or partially implemented in HW and blocks likely to be implemented in software. In fact, the GLCM block must be fully implemented in HW because of its memory intensive access. Besides, we can profit by the parallelism provided by the HW to compute the four GLCM matrix for $\theta = 0^\circ$, 45°, 90° and 135° and the textural features (contrast and energy) associated with each GLCM matrix in parallel. However, the contrast and energy are calculated based on the GLCM matrix using Eqs. (1) and (2), respectively. The results are obtained by these equations are in floating-point which cannot be implemented in

HW. Therefore, to solve this problem, the sum operations in the contrast and energy equations are performed in HW for reducing the time needed to access to the values of the GLCM matrix. But the division by the parameter R is carried out in SW in order to not lost in the accuracy of the results given by the contrast and energy. Further, the computation of the one hundred modified images is realized by using floating-point operation. For that, the SW implementation is preferred to perform the gamma estimation algorithm by variation the γ value from 0.1 to 10.0 using 0.1 as increment step. On the other hand, the otsu algorithm is based on several conditions. For this, we believe that the HW implementation cannot accelerate this algorithm. Accordingly, we suggest to implement it in SW.

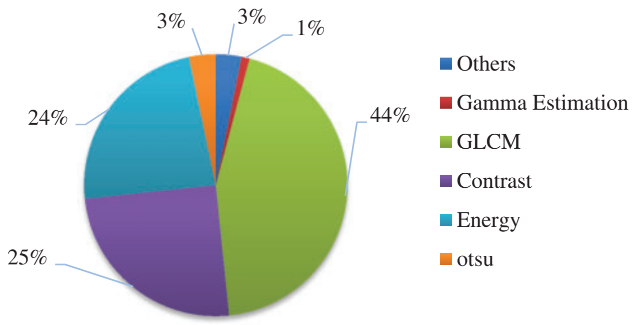

Figure 5: The average CPU execution time of different blocks of the GCM method

4 HLS Architecture of the GLCM Coprocessor

Since their creation, the FPGAs have not stopped evolving. It is used to implement several applications in various field such as image and video processing [24,25], IoT system [26], neural networks [27], etc. Recently, the High-level synthesis (HLS) flow is used with FPGA to decrease the design cycle time and increase the flexibility and the productivity of design by performing the design space exploration from a given behavioural description based on a high-level programming language (Like: SystemC, C/C++, etc.) without having to actually adapt it. For that, the HLS is emerged as an efficient and powerful tool to speed up the process of designing a hardware architecture. In this context, several HLS tools are elaborated like Xilinx Vivado HLS tool which permits through the directives to control how the design can be implemented, e.g., the arrays can be implemented as registers or memories, the loops can be pipelined or not unrolled or fully/partially unrolled. Thus, this tool allows in fraction of time to explore the design space and generate an optimized hardware design with taken an account a trade-off between hardware cost and execution time.

Thereby, in this work, the GLCM coprocessor is designed using the Vivado HLS 18.1 tool as IP block. However, as reported in Fig. 6, the GLCM coprocessor collects the image pixels through a circular buffer and computes the GLCM matrix for four θ angles ($\theta = 0^\circ, 45^\circ, 90^\circ, 135^\circ$) with distance ($d = 1$) between two pixels and the textural features (contrast and energy) associated for each θ angle. In fact, the developed coprocessor contains two inputs: vector 1 and vector 2. Indeed, the vector 1 contains a line of image. But the vector 2 contains the next line. 8 bits are used to store pixel in each vectors. Consequently, the Ng is equal to 256 and the size of the GLCM matrix is 256×256 coefficients. The GLCM coprocessor receives in parallel two pixels from vector 1 and 2. Each pixel received is stored in internal circular buffer to reduce the DDR external access and increase the data bandwidth. Fig. 7 illustrates the principle of the circular buffer unit which contains the centrale pixel (black.background) and its neighboring pixel (gray.background) for the four θ angles ($\theta = 0^\circ, 45^\circ, 90^\circ, 135^\circ$). Once, the five pixels are ready Fig. 7A, the central pixel and the neighboring pixel for a specific direction are sent to calculate the associate GLCM matrix in parallel. After that, as illustrated in Fig. 7B, the central pixel is glided by one pixel to the right and a new neighboring pixel are introduced into the circular buffer unit. Once the four GLCM matrix are calculated, the computation of the preliminary values of the contrast and energy are started in parallel. In the end, the GLCM coprocessor provides as output two values used to calculate the contrast and the energy.

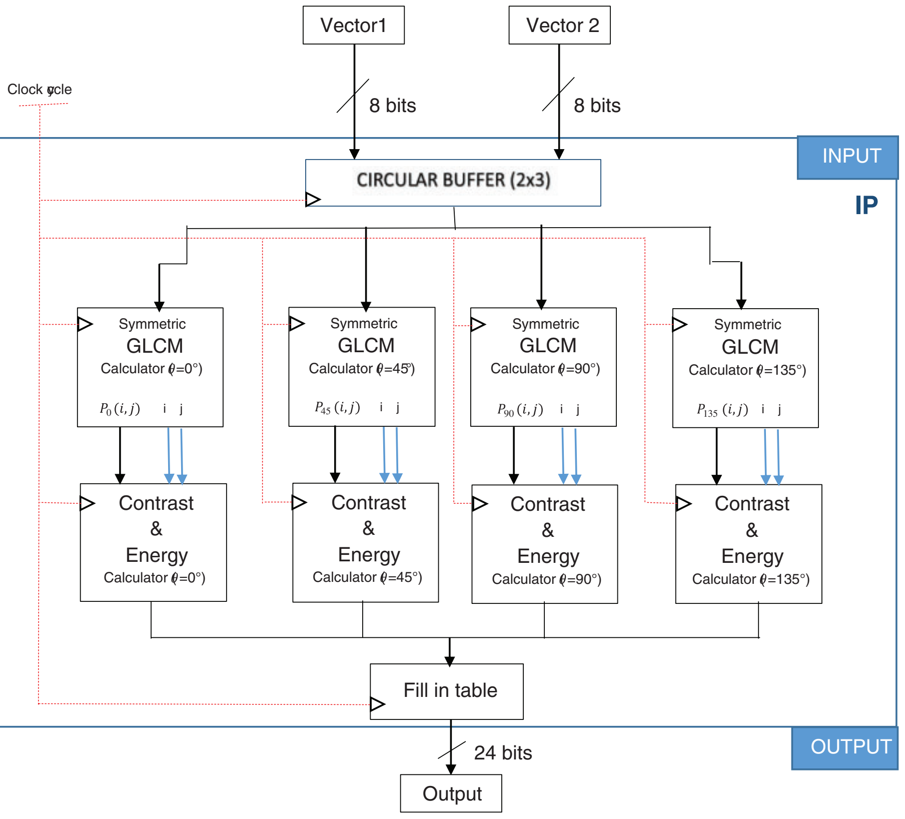
Vector1
Vector 2
Clock cycle
8 bits
8 bits
INPUT
IP
CIRCULAR BUFFER (2x3)
Symmetric GLCM Calculator (θ=0°) $P_0(i,j)$ i j
Symmetric GLCM Calculator (θ=45°) $P_{45}(i,j)$ i j
Symmetric GLCM Calculator (θ=90°) $P_{90}(i,j)$ i j
Symmetric GLCM Calculator (θ=135°) $P_{135}(i,j)$ i j
Contrast & Energy Calculator (θ=0°)
Contrast & Energy Calculator (θ=45°)
Contrast & Energy Calculator (θ=90°)
Contrast & Energy Calculator (θ=135°)
Fill in table
24 bits
OUTPUT
Output

Figure 6: HLS hardware design of the GLCM coprocessor

The C programming language is used to develop the specification of the GLCM coprocessor. The instruction in the C code is written with Vivado HLS friendly syntax. The optimization steps are applied to the C code by adding incrementally several directives. Thus, three solutions are generated to implement the GLCM coprocessor. Tab. 2 and Fig. 8 illustrate the comparison of these solutions in terms of hardware cost and the number of clock cycles, respectively.

However, the solution 1 are generated with Vivado HLS 18.1 tool without adding any directives. According to Tab. 2, this solution consumes 1% of Look-up-Table (LUT), 0.3% of Flip-Flop (FF), 13% of BRAM blocks and 0.3% of DSP blocks of the Zynq XCZU9EG Xilinx FPGA. But the number of clock cycles is equal to 459324, as reported in Fig. 8 to compute four GLCM matrix and the preliminary values of the contrast and energy for 256 $\times$ 256 pixels image size. To decrease the number of clock cycles, the solution 2 is generated by applying the PIPLINE directive in the loop iterations. The PIPLINE is used with an interval equal to 1 to reduce the time latency. This solution allows to decrease the number of clock cycles by 57% as depicted in Fig. 8 with an increase in number of LUTs, FFs and DSP blocks by 88%, 89% and 93%, respectively (Tab. 2), relative to the solution 1. From this result we can see an important increase in the hardware cost for solution 2. Thus, we have to integrate the ALLOCATION and RESSOURCE directives to the GLCM C code to reduce the hardware cost. In fact, the ALLOCATION directive is added to process the multiplication operations which allows to share the hardware resources between several operations. Furthermore, the RESOURCE directive is used to implement the GLCM matrix by a specific memory blocks (BRAMs). This optimization allowed to design the solution 3 which gives a decrease in number of LUTs and FFs by 3.4% and 3%, respectively compared to solution 2 as illustrated in Tab. 2 with the same number of clock cycles as proved from Fig. 8. Afterward, the solution 3 is used for HW/SW implementation of the GCM method.

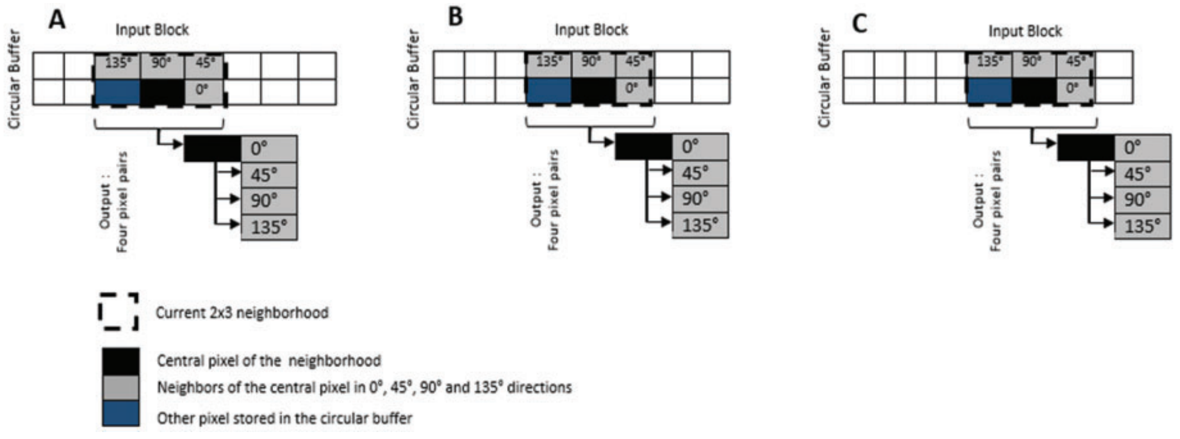
A
B
C
Input Block
Circular Buffer
135°
90°
45°
0°
Output : Four pixel pairs
Current 2x3 neighborhood
Central pixel of the neighborhood
Neighbors of the central pixel in 0°, 45°, 90° and 135° directions
Other pixel stored in the circular buffer

Figure 7: Circular buffer unit

Table 2: Comparison of the hardware cost for HLS solution

Architectures	LUTs	FF	BRAM18k	DSP48E
#Solution 1	2694	1944	240	8
#Solution 2	23929	19200	240	116
#Solution 3	23110	18640	240	116

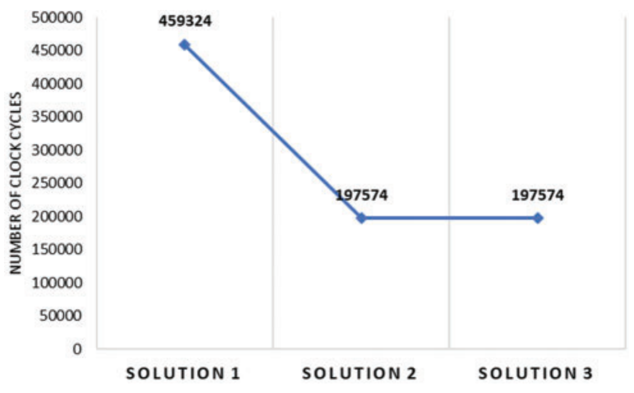

Figure 8: Comparison of the number of clock cycles for HLS solutions

5 HW/SW Implementation of the GCM Method

The ZCU102 board is selected for the HW/SW implementation of the GCM method. In this board, the Zynq XCZU9EG FPGA consists by two parts: the Processing System (PS) part and the Programmable Logic (PL) part. The PS part is based on the ARM Cortex-A53 processor. But the PL part contains 274k LUTs, 2520 DSPs block and 32.1 Mb memory. This architecture is completed by industry standard AXI (Advanced eXtensible Interface) interface protocol which provided by ARM as part of the Advanced Microcontroller Bus Architecture (AMBA) standard. The AXI interface affords a high bandwidth, low latency connection between PS and PL parts. However, there are two main AXI4-interfaces: AXI4-Stream and AXI4-Lite. In fact, the AXI4-Stream provides a high-speed streaming data by using point-to-point streaming data without indicating any addresses. But the AXI4-Lite is a traditional low throughput memory communication used for example from/to status and control registers.

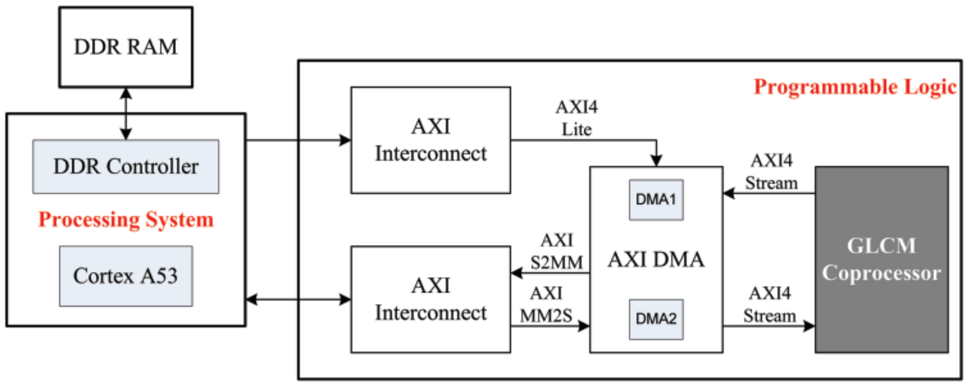

Figure 9: HLS HW/SW GCM design

Fig. 9 describes the HW/SW design to implement the GCM algorithm. In fact, the SW part is developed by C/C++ programming language which is used to control the data transfer between the DDR memory and the GLCM coprocessor, to generate the gamma modified images, to compute the finale value of the energy and the contrast and to determine the threshold value through the otsu algorithm for extracting text from color image. Further, the SW part is compiled for the standalone mode using the SDK tool to generate the executable file (.elf) and performed by the PS part through the ARM Cortex-A53 processor at 677 MHz. On the other hand, the HW part is used to implement the GLCM algorithm on PL part and perform it at 100 MHz. The connection between the PS and PL parts is realized through the AXI4-Stream interface by using Direct Memory Access (DMA) to increase the data bandwidth. Thus, two DMAs (DMA1 and DMA2) are connected to the GLCM coprocessor for parallel data transfer with DDR memory. Indeed, DMA1 is configurated in write/read mode. But DMA2 is configurated in read mode only. But the AXI4-Lite is used to configure the DMA by indicating the address from where it starts to read data and the data length. These interfaces (AXI4-Stream and AXI4-Lite) are created using the Vivado HLS tool when generating the GLCM coprocessor which is exported as IP core to the Vivado design tool 18.1 to produce the HW/SW GCM design.

Nevertheless, in the beginning, the color image is stored in the DDR memory. Then, the ARM Cortex-A53 initiates through AXI4-Lite interface the DMA to transfer two image lines in parallel from DDR memory to GLCM coprocessor using DMA1 and DMA2. Once the GLCM coprocessor receives pixels, it starts to compute in parallel the GLCM matrix for four θ angles (θ = 0°, 45°, 90°, 135°) and the textural features (contrast and energy) associated for each θ angle as shown in Fig. 6. When, the GLCM coprocessor finish to process data, the DMA1 send the preliminary values of the contrast and energy to the PS part where the ARM Cortex-A53 processor performs a floating-point division by the parameter R as indicated by Eqs. (1) and (2) to obtain the final values of the energy and contrast, respectively. Then, it calculates the threshold (T) using otsu algorithm and determines the optimum value of gamma which allows to eliminate the background and extract the text from complex color image.

The HLS HW/SW GCM design is synthesized and implemented by Vivado design tool 18.1. The implementation results shows that this design uses 51308 (18.72%) of LUTs, 45206 (8.24%) of FFs, 123 (13.49%) of BRAMs and 118 (4.68%) of DSPs blocks of the Zynq XCZU9EG FPGA.

6 Performance Evaluation

The performance evaluation of the HW/SW GCM design is carried out on the ZCU102 board. The performance is evaluated and compared to the GCM SW implementation in terms of the efficiency of correctly detected characters in the image and the execution time. However, in experimental evaluation, the execution time is measured by the PS timer. Besides, several images from ICDAR Dataset are selected. The size of this image is 256 × 256 pixels. Furthermore, the F-measure is computed based on Eq. (4) to analysis the performance of the text extraction. In fact, F-measure is the harmonic mean of recall (Eq. (5)) and precision (Eq. (6)) rates [28].

$$\mathrm{F-measure} = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{4}$$

$$Recall = \frac{Number\ of\ correctly\ detected\ text\ regions}{Number\ of\ ground\ truth\ text\ regions} \tag{5}$$

$$Precision = \frac{Number\ of\ \ correctly\ detected\ text\ regions}{Number\ of\ detected\ text\ regions} \tag{6}$$

From Fig. 10, we can conclude that the HW/SW implementation of the GCM method reduces dramatically the execution time by 89% compared to the SW implementation. This result is provided for the same performance for text extraction which is justified by the value of the F-Measure for HW/SW and SW implementation of the GCM method as presented in Fig. 11.

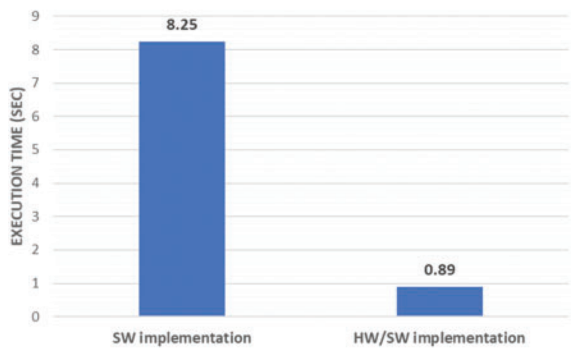

Figure 10: Comparaison of execution time between SW and HW/SW implementation of the GCM method

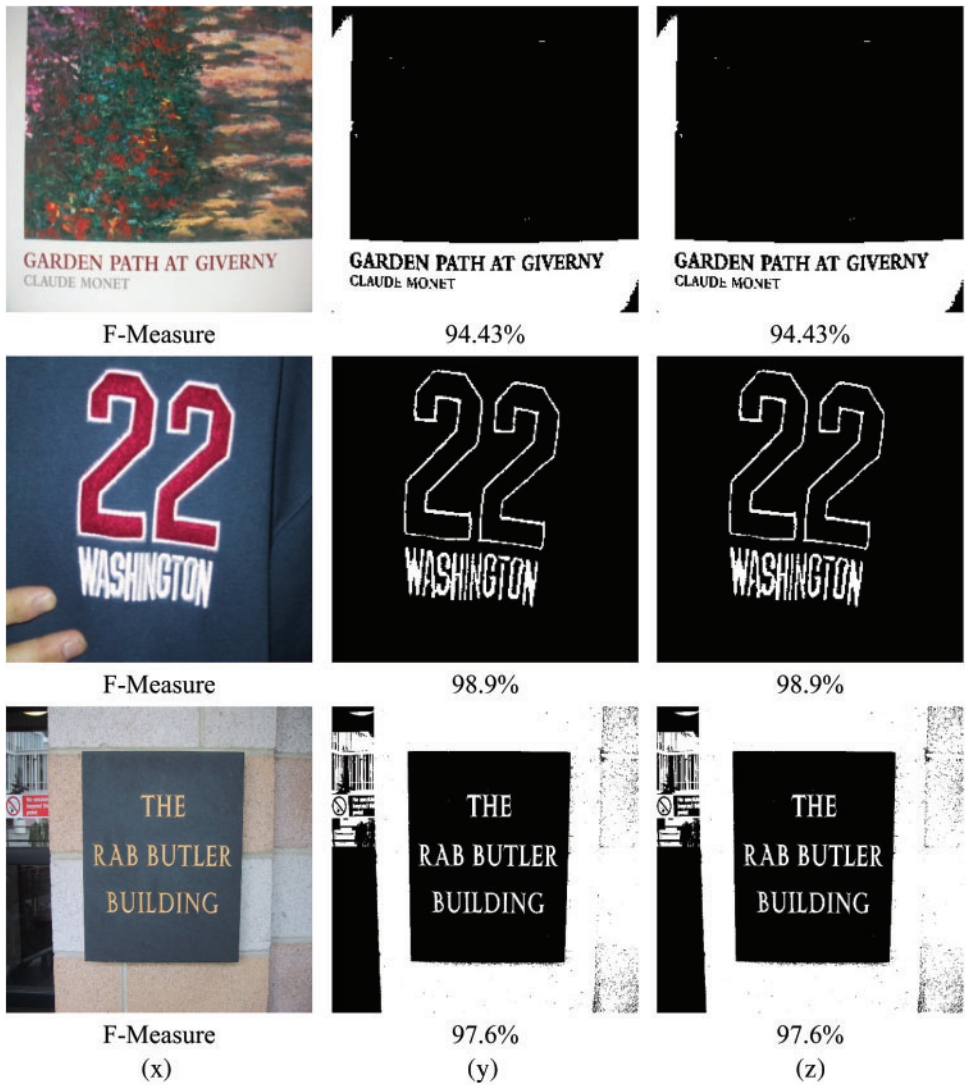

Figure 11: (x) Original image, (y) Text extraction with SW implemetation, (z) Text extraction with HW/SW implementation

Tab. 3 presents the comparison of our HW/SW GCM design with works proposed in literature. All works use image from ICDAR dataset to evaluate their proposition, However, we can notice that our design is more performant and can consume less energy than [8] which proposes some optimization in the GCM method to determine the best gamma value from some modified image. This affects the efficiency of the text extraction. Also, our design is more performant than [12] which add custom instruction to NIOS II processor to compute contrast and energy on hardware.

Table 3: Comparison of the proposed GCM design

Ref	Execution time (ms)	Design specification
[8]	478 ms	Intel Core (TM) i3 @3.07 GHz
[12]	702 s	NIOS II + C. Instruction @100 MHz
Our Design	890 ms	ARM Cortex-A53 @667 MHz + GLCM coprocessor@100 MHz

7 Conclusion

In this paper, the HW/SW codesign implementation of the GCM algorithm is proposed. The HLS flow is used to design an efficient HW architecture for the critical block in the GCM by adding incrementally to the C code some directives (such as PIPELINE, ALLOCATION, RESSOURCE) through Xilinx Vivado HLS tool. This block is implemented in the PL part as IP block and integrated with the ARM Cortex-A53 processor in HW/SW codesign context. The data transfer between the DDR memory and the PL part is carried out by using the AXI4-stream through the DMA to increase the data bandwidth. The evaluation of the proposed design is realized on ZCU102 FPGA board. The experimental results show that the HLS HW/SW GCM design speed up the execution time by 89% relative to the SW implementation with same value of the F-Measure for correctly text extraction. However, the designed system can be integrated into the text-to-speech system to help visually impaired people to interpret text.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. Jacobsen, "World health organization," 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visualimpairment.

[2] C. Yi and Y. Tian, "Text string detection from natural scenes by structure-based partition and grouping," *IEEE Transactions on Image Processing*, vol. 20, no. 9, pp. 2594–2605, 2011.

[3] Y. Pan, X. Hou and C. Liu, "A hybrid approach to detect and localize texts in natural scene images," *IEEE Transactions on Image Processing*, vol. 20, no. 3, pp. 800–813, 2011.

[4] T. Q. Phan, P. Shivakumara and C. L. Tan, "Detecting text in the real world," in *Proc. of the ACM Conf. on Multimedia*, New York, NY, USA, pp. 765–768, 2012.

[5] C. Yi and Y. Tian, "Text extraction from scene images by character appearance and structure modeling," *Computer Vision and Image Understanding*, vol. 117, no. 2, pp. 182–194, 2013.

[6] C. P. Sumathi and G. G. Devi, "Automatic text extraction from complex-colored images using gamma correction method," *Journal of Computer Science*, vol. 10, no. 4, pp. 705–715, 2014.

[7] G. G. Devi and C. P. Sumathi, "Text extraction from images using gamma correction method and different text extraction methods—A comparative analysis," in *Proc. of the IEEE Conf. on Information Communication and Embedded Systems*, Chennai, India, pp. 1–5, 2014.

[8] R. Kachouri, C. M. Armas and M. Akil, "Gamma correction acceleration for real-time text extraction from complex colored images," in *Proc. of the IEEE Conf. on Image Processing*, Quebec City, QC, Canada, pp. 527–531, 2015.

[9] A. B. Girisha, M. C. Chandrashekhar and M. Z. Kurian, “FPGA implementation of GLCM,” *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 2, no. 6, pp. 2618–2621, 2013.

[10] A. R. Akoushideh, A. Shahbahrami and B. M. N. Maybodi, “High performance implementation of texture features extraction algorithms using FPGA architecture,” *Journal Real-Time Image Processing*, vol. 9, no. 1, pp. 141–157, 2014.

[11] D. Maroulis, D. K. Iakovidis and D. Bariamis, “FPGA-based system for real-time video texture analysis,” *Journal of Signal Processing Systems*, vol. 53, pp. 419, 2008.

[12] A. Boudabous, M. A. Ben Atitallah, R. Kachouri and A. Ben Atitallah, “HW/SW design and FPGA implementation of The GCM for an efficient text extraction from complex images,” *International Journal of Scientific & Technology Research*, vol. 9, no. 3, pp. 6572–6581, 2020.

[13] A. Ben Atitallah, M. Kammoun and R. Ben Atitallah, “An optimized FPGA design of inverse quantization and transform for HEVC decoding blocks and validation in an SW/HW environment,” *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 28, no. 3, pp. 1656–1672, 2020.

[14] T. M. Alanazi, A. Ben Atitallah and I. Abid, “An optimized SW/HW AVMF design based on high-level synthesis flow for color images,” *CMC-Computers, Materials & Continua*, vol. 68, no. 3, pp. 2925–2943, 2021.

[15] A. Boudabous, L. Khriji, A. Ben Atitallah, P. Kadionik and N. Masmoudi, “Efficient architecture and implementation of vector median filter in co-design context,” *Radioengineering-Prague*, vol. 16, no. 3, pp. 113–119, 2007.

[16] A. Ben Atitallah, M. Kammoun, K. M. A. Ali and R. Ben Atitallah, “An FPGA comparative study of high-level and low-level combined designs for HEVC intra, inverse quantization, and IDCT/IDST 2D modules,” *International Journal of Circuit Theory and Applications*, vol. 48, no. 8, pp. 1274–1290, 2020.

[17] M. Kammoun, A. Ben Atitallah, K. M. Ali and R. Ben Atitallah, “Case study of an HEVC decoder application using high-level synthesis: Intra prediction, dequantization, and inverse transform blocks,” *Journal of Electronic Imaging*, vol. 28, no. 3, pp. 1, 2019.

[18] A. B. Atitallah and I. Abid, “An efficient FPGA implementation of AVMF filter using high-level synthesis,” in *Proc. of the IEEE Conf. on Sciences and Techniques of Automatic Control and Computer Engineering*, Monastir, Tunisia, pp. 82–85, 2020.

[19] R. M. Haralick, K. Shanmugam and I. Dinstein, “Textural features for image classification,” *IEEE Transactions on Systems Man and Cybernetics*, vol. 3, no. 6, pp. 610–621, 1973.

[20] L. H. Siew, R. M. Hodgson and E. J. Wood, “Texture measures for carpet wear assessment,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 1, pp. 92–105, 1988.

[21] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[22] ICDAR Dataset, “Text in scene images,” 2011. [Online]. Available: http://robustreading.opendfki.de/wiki/SceneText.

[23] Xilinx, “Xilinx Zynq UltraScale + MPSoC ZCU102 evaluation kit,” 2021. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#hardware.

[24] M. Kammoun, A. Ben Atitallah, R. Ben Atitallah and N. Masmoudi, “Design exploration of efficient implementation on SOC heterogeneous platform: HEVC intra prediction application,” *International Journal of Circuit Theory and Applications*, vol. 45, no. 12, pp. 2243–2259, 2017.

[25] A. Ben Atitallah, H. Loukil, P. Kadionik and N. Masmoudi, “Advanced design of TQ/IQT component for H.264/AVC based on SoPC validation,” *WSEAS Transactions on Circuits and Systems*, vol. 11, no. 7, pp. 211–223, 2012.

[26] Y. Chen, J. He, X. Zhang, C. Hao and D. Chen, “Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs,” in *Proc. of the ACM Symp. on Field-Programmable Gate Arrays*, New York, NY, USA, pp. 73–82, 2019.

[27] X. Zhang, A. Ramachandran, C. Zhuge, D. He, W. Zho *et al.,* "Machine learning on FPGAs to face the IoT revolution," in *Proc. of the IEEE Conf. on Computer-Aided Design*, Irvine, CA, USA, pp. 894–901, 2017.
[28] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong *et al.,* "robust reading competitions," in *Proc. of the IEEE Conf. on Document Analysis and Recognition*, Edinburgh, UK, pp. 682–687, 2003.