Hw/Sw Design And Fpga Implementation Of The Gcm For An Efficient Text Extraction From Complex Images

Anis Boudabous, Mohamed Amine Ben Atitallah, Rostom Kachouri, Ahmed Ben Atitallah

Abstract: The Gamma Correction Method GCM is used as an important task in the text extraction process. In this paper, we propose a co-design implementation of the GCM using an FPGA board. This HW/SW implementation uses VHDL and C language to build a real embedded system validation. The implementation methodology performs an optimized run time of the GCM for text extraction in complex images. Our experiments are based on an FPGA HW/SW board. The obtained results showed that the proposed method can help to improve the performance of the GCM for text extraction by using the NiosII processor and hardware custom instructions. The proposed architecture enables to gain of 35% of the run time.

Index Terms: Text extraction, Gamma correction, HW/SW design, FPGA implementation, SOPC, NIOS-II processor, custom instructions.

1 INTRODUCTION

Recently, it has been noted that many interesting research studies that have focused on text mining in complex images, have been proposed. In our current research, we particularly deal with multi-scale algorithms or the Gamma Correction Method (GCM). To do so, we refer to some research studies, such as [1], which used the techniques of Liquid Cristal Display, [2] and [3] which presented image enhancement methods using GCM with three levels of threshold. [4] proposed a new adaptive gamma correction method. While [5] described an adaptive gamma correction for contrast enhancement in dark images. In [6] the authors illustrated an OCR correctness enhancement in manuscript images. On the other hand, [7] focused on contrast enhancement using a range-limited bi-histogram equalization with Adaptive Gamma Correction for color Images. In 2014, Devi et al. [8] presented a comparative study of different text extraction methods from images. A recent work by Sumathi et al [9] has proposed a method for the extraction of texts from complex images based on the GCM. Moreover, it has exhaustively described the different blocs existing in the extraction of the text from the image and proposed a new algorithm for the extraction of text from a complex image. This method proposes to eliminate the background of the image based on the gamma correction method. In fact, this method showed a lot of effectiveness in resolving this difficulty. Overall, the texts integrated in images can be divided into different categories. Some artificially covers the images and therefore, are considered as a legend. Furthermore, these texts are incorporated in the image which makes it more difficult to extract them.

Before defining the steps of the extraction of a text embedded in a complex image, it is important to know the basic properties of texts; geometry, color, movement, edge and compression. In fact, text extraction generates several problems, such as detection, location, identification, extraction, enhancement and recognition. In general, the first three steps can be confused, but in reality they are quite distinct. The detection step determines whether or not texts are present in a given image while localization is a process determining the location of a text in an image and generating frames around the text. This identification is performed to reduce the processing time of localization and memorize the original location of the text in the image. In fact, the extraction turns the text into a binary image by extracting it from the background of the initial support. As a result, the text will be improved to give a better resolution. The recognition process, using the technique of optical character recognition (OCR), transforms the printed text (analog) into a digital file consisting of ASCII characters instead of pixels (analog file). In this context, it is important to note that in our study, we are interested mainly in detecting, locating and extracting the text boxes from the scanned documents and colored images using the Gamma Correction Method (GCM). For instance, in figure 1, we show the basic steps of text extraction from images. Moreover, many related algorithms are used in color image processing, which are usually software implementations. On the other hand, the hardware feature will typically increase the processing speed. Then, through the technology advances in the system integration, we can propose the system design as some functional blocs with at least one processor (as a processing unit). The general idea behind this is to attribute complex computing tasks to the hardware (here FPGA). This attribution helps to exploit the parallelism and pipeline algorithms, and operate the software flexibility, which results in a context of “codesign”.

- Anis Boudabous, College of Computer and Information Sciences, Jofu University, Kingdom of Saudi Arabia, and LETI (E.N.I.S.), University of Sfax, Tunisia.
- Mohamed Amine Ben Atitallah, National Engineering School of Gabes, University of Gabes, and A3SI, ESIEE Paris, CNRS, Paris-Est University, France.
- Ahmed Ben Atitallah College of Engineering, Jofu University, Kingdom of Saudi Arabia, and LETI (E.N.I.S.), University of Sfax, Tunisia.
- Rostom Kachouri, Laboratory of Informatics, Gaspard-Monge, A3SI, ESIEE Paris, CNRS, Paris-Est University, France.
- aboudabbous@iu.edu.sa; benatitallahmohamedamin@yahoo.fr; rostom.kachouri@esiee.fr; abenatitallah@ju.edu.sa.
In this work, we are interested particularly in the design of a System on Programmable Chip or SOPC using the EP2S60F672C3 FPGA board. We propose in this work a Custom Instructions integrated in our HW/SW system. This implementation will be made to speed up the overall execution time of the algorithm relating to the Gamma Correction Method (GCM). Therefore, this paper is organized as follows: Section II presents the method and theory of Gamma Correction Method (GCM). Section III deals with a preliminary complexity study of GCM. Then, in section IV, we propose a SOPC system based on the NiosII processor. This section also reports on a software implementation and profiling which was performed using the NiosII processor in a SOPC context. In section V, we explain and detail the hardware implementation of contrast and energy. The HW/SW implementation of the Gamma Correction Algorithm using the Altera NiosII development board is the subject of section VI. Finally, the conclusions are drawn in the last section.

2 THE GAMMA CORRECTION METHOD
The Gamma Correction Method (GCM) is a well-known technique applied in color images and particularly in text extraction. More precisely, it eliminates the background and focuses mainly on the text region. As a consequence, the correlation between the existing input (x) and output (y) can be represented by the following equation (1):

\[ y = x^\gamma \]  

(1)

Inside the perfect case, \( y = 1 \), the input and output values are the same. Therefore, if the gamma values deflect from ‘1’, we detect a major intensity difference in the input and output values. While if gamma comes close to zero, the output pixels turn dark. On the other hand, if gamma tends to infinity, the pixels become brighter. In fact, the diagram below, (figure 2), indicates the different stages of GCM designed for text extraction from complex images. We begin by generating the adapted (or modified) image from the original one for the different gamma values (\( \gamma = 0.1 \ldots 10 \)) using the following equation:

\[ pm(i) = \left( 0.5 + \left( \frac{i}{\text{GRAYLEVEL}-1} \right)^\gamma \right) \times \text{GRAYLEVEL} - 1 \]  

(2)

pm(i): is performed to modify all pixels in the original image and create the adapted or modified image, for each gamma value.

The GRAY LEVEL represents the pixel density. After that, for each image, we define the image threshold by producing four Gray Level Co-occurrence matrices, GLCM, (for 0, 45, 90 and 135 degrees). In fact, these co-occurrence matrices calculate the occurrence probability \( P(i, j, \delta, \theta) \) of the pair values of pixels situated at some distance within the test image. As for the occurrence probability, it defines the number of times, anywhere, a color intensity i of one pixel appears at a distance equal to 1 (\( \delta = 1 \)) from another pixel with color intensity j. We consider this, along with all the orientation values (\( \theta = 0, 45, 90 \) and 135 degree). The characteristics extracted from the co-occurrence matrices contain the homogeneity information, the linear dependencies between the gray levels, the contrast and the image complexity. The matrices obtained according to the four directions are calculated as shown below:

\[ P(i, j, \delta, 0) = \left\{ \begin{array}{c} \{ (k, l), (m, n) \} \in (N \times M)^2 \\ \text{with} \ (k - m = \delta, |l - n| = \delta, l_{kl} = i, l_{mn} = j) \end{array} \right\} \]

\[ P(i, j, \delta, 45) = \left\{ \begin{array}{c} \{ (k, l), (m, n) \} \in (N \times M)^2 \\ \text{with} \ (k - m = \delta, l - n = -\delta, l_{kl} = i, l_{mn} = j) \end{array} \right\} \]

\[ P(i, j, \delta, 90) = \left\{ \begin{array}{c} \{ (k, l), (m, n) \} \in (N \times M)^2 \\ \text{with} \ (|k - m| = \delta, l - n = 0, l_{kl} = i, l_{mn} = j) \end{array} \right\} \]

\[ P(i, j, \delta, 135) = \left\{ \begin{array}{c} \{ (k, l), (m, n) \} \in (N \times M)^2 \\ \text{with} \ (k - m = \delta, l - n = \delta, l_{kl} = i, l_{mn} = j) \end{array} \right\} \]

where:

\( (k, l) \) are the coordinates of color intensity i \( \in [0, n_{\text{max}}-1] \).

\( (m, n) \) are the coordinates of color intensity j \( \in [0, n_{\text{max}}-1] \).

This enabled us to calculate the energy and contrast values [3][4].
2.1 Energy and Contrast
The energy value (E), which is also defined as uniformity or ASM (Angular Second Moment), describes the smoothness of the image. It performs the uniformity of texture which is in fact the pixel pair duplications. The energy is then given by equation (3):

$$E = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2$$

(3)

$p(i,j)$: GLCM pixel

$N_g$: the maximum of gray level

The contrast (C) estimates the local gray level deviation (equation 4) of which indicates the spatial frequency of the image and a diverse moment of GLCM.

$$C = \sum_{i=1}^{N_g} h^2 \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)$$

(4)

The gray level image will then be transformed into a binary image by means of Otsu’s technique.

2.2 Otsu’s technique
Roughly speaking, Otsu’s technique [3] looks for the adequate threshold (denoted by T) that decreases the variance within the class, (the intra-class variance), distinguished as a weighted addition of the two class variances, as described in equation (5):

$$\sigma_k^2(T) = \omega_1(T)\sigma_1^2(T) + \omega_2(T)\sigma_2^2(T)$$

(5)

$\sigma_1, \sigma_2$: corresponds to the inter-class variance

$\omega$: corresponds to the probability to be in classes 1 or 2.

Moreover, Otsu proved that reducing the intra-class variance is a matching concept to capitalize an inter-class variance specified by equation (6):

$$\sigma_k^2(T) = \sigma_1^2(T) - \omega_1(T)\omega_2(T)[\mu_1(T) - \mu_2(T)]^2$$

(6)

$\mu$: corresponds to the mean value for separation,

$\omega_1(T)$ corresponds to the class probability, which is calculated in equation 7 using the image histogram:

$$\omega_1(T) = \sum_i^h p(i)$$

(7)

$p(i)$: density of the probability,

Where the class mean is $\mu_1(T)$, as presented in equation (8):

$$\mu_1(T) = \sum_i^h p(i)x(i))/\omega_1$$

(8)

$x(i)$: is a vector from (0) to (maxium of gray level -1)

This process will be repeated 100 times (MGLCM, contrast, energy, and threshold) for all the gamma values. After that, we guess the optimum value of Gamma based on the contrast, energy and threshold. Finally, the corrected and the binary images are created. In the next section, we will discuss in details how to deduce the most favorable Gamma value.

2.3 Gamma estimation
To approximate the Gamma value, we apply several rules extracted from paper [3]. This expected value of gamma was applied in the image input to generate a corrected image. In conclusion, Otsu’s algorithm was used to evaluate the threshold and apply the value needed to produce the output image. Figure 3 shows a diagram illustrating the GCM rules.

3 PRELIMINARY STUDY OF GCM
Our previous research has focused on the complexity of this efficient algorithm for text extraction from complex images based on the GCM [10]. Initially, our C/C++ algorithm study of gamma correction was made on a personal computer i3, 2 GHz. We calculated an execution time equal to 2s for an image of 221 × 228 pixels. Then, the performance of our C/C++ algorithm was compared to C code written using the OpenCV library. This later is provided in the framework of cooperation with the computer lab Gaspard Monge, ESIEE Paris. In fact, a comparison between the execution time of our C code and the C code using OpenCV has been cited in the previous work [10]. We therefore obtained an important optimization of the execution time (50%) using our C code. Our code gave the same value of gamma compared to the code using an OpenCV. Finally, the basic purpose was to develop a C/C++ algorithm regardless of the OpenCV library. This new version of this algorithm makes it possible to implement the gamma correction method on an FPGA board in the context of HW/SW design. To validate the Gamma correction algorithm (or Gamma estimation), we used a standard image having a size of 221 × 228 pixels. The implementation of the different stages of the GCM (see figure 2) was determined for each gamma value, contrast, energy and threshold. Table 1 provides an overview of the values generated using a test image (figure 14 (a1), size 221 × 228 pixels). By means of these results, we can estimate the optimal gamma value using the rules 1, 2 or 3 (see figure 3). Therefore, in the case of the image in figure 14 (a1), Rule 2 has been applied for $\gamma = 1$ (original image) then, the energy value is <0.05 and the contrast value> = 1000. This rule helps to resolve the optimum value of gamma ($\gamma = 10$) which helps to remove the image background (figure 14 (b1)) and therefore the selection of the threshold value (T = 0.402344) which is used to convert the gray scale image into a binary image (figure 14 (c1)). However, in the image presented in figure 14 (a2), Rule 3 has been applied for $\gamma = 1$ (original image); where the energy value <0.05 and the contrast value<1000 therefore $\gamma = 6$ (figure 14 (b1)). $T=0.292969$ (figure 14 (c1)). This new version of algorithm can be implemented in a HW/SW design based on “System on Programmable Chip” (SOPC) in order to improve its performance. In the following sections, we will define and discuss the SOPC system design.

### Table 1

<table>
<thead>
<tr>
<th>Gamma Threshold</th>
<th>Contrast</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>15.8656</td>
<td>0.011908</td>
</tr>
<tr>
<td>0.925781</td>
<td>55.4663</td>
<td>0.013286</td>
</tr>
<tr>
<td>0.2</td>
<td>56.5976</td>
<td>0.014153</td>
</tr>
<tr>
<td>0.863281</td>
<td>2046.98</td>
<td>0.142772</td>
</tr>
<tr>
<td>5.3</td>
<td>2056.77</td>
<td>0.150066</td>
</tr>
<tr>
<td>0.445313</td>
<td>2078.57</td>
<td>0.156321</td>
</tr>
<tr>
<td>5.5</td>
<td>2089.88</td>
<td>0.159688</td>
</tr>
<tr>
<td>0.441406</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.445313</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.9</td>
<td>2487.94</td>
<td>0.267063</td>
</tr>
<tr>
<td>0.40625</td>
<td>2495.65</td>
<td>0.268467</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NIOSII PROCESSOR

In general, a SOPC or an embedded system is designed to perform one or more functions that are defined in the specifications. For the software part, the C/C++ language is generally used. For the hardware part, the VHDL or the Verilog language are usually operated as they help with description in the RTL (Register Transfer Level). An important stage of validation and testing of the good functionality of the described circuit is then performed. The Quartus II software was proposed by the Altera (actually Intel-FPGA) Company [11] for the complete management of the FPGA design flow. The IDE Quartus II (Integrated Design Entry) integrates a SOPC Builder tool which is used to build the SOPC system besides, this tool integrates various I/O devices such as the NiosII processor, the SRAM and SDRAM controller, a DMA controller (Direct Memory Access). Figure 4 illustrates the different devices connected to the NiosII processor through the Avalon bus for a typical SOPC system [12].

Similarly, we can integrate our component in the design as an external IP block (camera interface, VGA, hardware accelerator ...). We can also integrate many devices as many as we like but they are just limited by the number of pins and logic cells of the FPGA. At the final step of the SOPC system, Quartus II generates the project by integrating all IP modules. After making a synthesis, we get the FPGA programming file corresponding to our SOPC design. On the other hand, we have a software development kit which includes the entire C files (.c and .h) to control the peripheral I/O Altera. The co-design advantage appears here, with the opportunity to develop some parts of the design via a hardware or a software. The NiosII embedded processor is a software firmcore processor which is exclusively devoted to the Altera family. In fact, the embedded software processor NiosII is configurable and upgradeable, enabling developers to have a flexible and robust SOPC solution. Moreover, the NiosII processor is a totally synchronous RISC processor which has a Harvard internal architecture and a maximum of six pipeline stages with a 32-bit bus width. Its performance is 30 to 80 MIPS (Million Instructions per Second). However, it is possible to accelerate some processes by adding custom instructions to the NiosII processor, as described in VHDL. In this way, it is possible to realize the operator overload or simply extend the instruction set. From figure 5, it is clear that we can add two types of instruction: Combinatorial (one cycle) or sequential (multi cycle) [12], to the Arithmetic Logic Unit (ALU) of the NiosII processor. To evaluate the speed performance of the gamma correction algorithm on an embedded design, we have implemented a SOPC system based on the NiosII embedded processor, which is shown in figure 6. This system consists of the NiosII processor, the Avalon bus and some devices (memory controller, UART, timer ...). The NiosII processor is the heart of the system. It is clocked at 100 MHz, and multiplication and division are wired. Moreover, the size of the instruction cache and data is 64KB. This processor is connected to various devices through the Avalon bus. The interface of this bus is automatically generated by an Altera NiosII generation tool (SOPC Builder). This embedded system is implemented on a Stratix II FPGA EP2S60F672C3, which provides 48352 ALUTs, 310 KB of embedded memory, 288 DSP blocs (9bits x 9bits), 6 PLLs and 493 inputs/outputs.

In this work, the development board used for experimental design is the EP2S60F672C3 map (figure 7) which was proposed by the Altera firm [11]. The board integrates an FPGA circuit associated with external devices, such as (JTAG, SRAM, SDRAM ...). Besides, the Altera NiosII development kit based on the Stratix II FPGA provides a complete development environment, including the hardware and the software needed for the system design. Based on the NiosII embedded processor family and the Stratix II EP2S60 component, the kit will be ideal for the design and validation of a broad range of embedded applications. The results of the synthesis of our system using Quartus II are shown in table II. To estimate the complexity of the gamma correction method, we used the standard image size 221x228 pixels. The execution of our gamma correction C/C++ algorithm on the embedded system NiosII provides 18 minutes as a run time. Therefore, it is necessary to study the distribution or partition of the CPU run time according to the different blocs that constitute the gamma correction algorithm to detect which blocs will be realized as a hardware accelerator.
blocks of the gamma correction algorithm for some images of 221x228 pixels. We also noticed that complexity is highly located in the three processing blocks: GLCM (35% of CPU time), contrast (37% CPU time) and energy (13% of CPU time).

Fig. 8: The average CPU run-time of different blocks of the gamma correction algorithm

5 HARDWARE IMPLEMENTATION OF CONTRAST AND ENERGY

In this section, we propose a hardware implementation as a policy of ALU equations of energy and contrast data that are illustrated as follows:

\[
\text{Energy} = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2}{N_g-1} \\
\text{Contrast} = \frac{\sum_{n=0}^{N_g-1} n^2 \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)}{N_g-1} 
\]

\(N_g\): Maximum gray level=256.
\(p(i,j)\): the pixels of GLCM.
\(n\): \((i-j)^2\)

In order to implement the equations of contrast and energy, combinatorial instructions are used. They consist of different I/O signals to communicate with the Arithmetic Logic Unit and the NIOSII processor. The signals, dataa and datab inputs as well as the output result, are implemented using 32 bits. An appropriate declaration and good instancing of contrast and energy instructions give you a good real time connection between the hardware and the software part. The execution and the validation use the SOPC Builder tool. Figures 9 and 10 show the architecture developed for the hardware implementation of the contrast and energy equations, respectively. In fact, we have developed an architecture for the contrast (figure 9) and the calculation of the \((i-j)^2 \times P(i,j)\) and \(((i-j-1)^2 \times P(i,j+1))\) of two pixels of the matrix GLCM then perform their summation. For this reason, we use in our architecture dataa and datab as two input signals. DataA signal contains the concatenated values of \(i\) and \(j\), which are 8 bits each. On the other hand, datab signal contains the concatenated values of \(P(i,j)\) and \(P(i,j+1)\). These values are 16 bits each while the final result was obtained by combining 32 bits. The same architecture is applied in the case of energy (figure 10) to calculate \(P(i,j)^2\) of two pixels of the matrix GLCM. Indeed, DataA and DataB contain \(P(i,j)\) and \(P(i,j+1)\), respectively. The final result is obtained by adding them. The implementation
results for the contrast and energy instructions on Stratix II FPGA EP2S60 are given in table III. These results show a low use of hardware resources on FPGA for the contrast and energy architectures.

![Hardware Architecture of the Contrast Instructions](image9.png)

**Fig. 9: Hardware architecture of the contrast instructions**

![Hardware Architecture of the Energy Instructions](image10.png)

**Fig. 10: Hardware architecture of the energy instructions**

**TABLE 3**

<table>
<thead>
<tr>
<th>Hardware Implementation Results of Contrast and Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contrast</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>ALUTs</td>
</tr>
<tr>
<td>I/O</td>
</tr>
<tr>
<td>Blocs DSPs</td>
</tr>
</tbody>
</table>

6 HW/SW IMPLEMENTATION OF THE GAMMA CORRECTION ALGORITHM

In this section, we will evaluate the gamma correction algorithm performance in a HW / SW context. Indeed, the assessment will be made before and after the integration of the hardware instructions of the contrast and energy in the ALU of the NIOSII processor as Custom instructions. In fact, figure 11 illustrates the NIOSII embedded system after the integration of the contrast and energy instructions. The results of the implementation of our embedded system NIOSII FPGA II Stratix EP2S60 are given in table IV. These results also show that the implementation of various devices constituting this system leaves enough resources in the FPGA to add other IP blocks. The percentage measures and the use rate of the processor, depending on the different processing block gamma correction algorithms, are illustrated in figures 12 and 13. We represent the software solution and the HW / SW solution for an image of size $106 \times 160$ pixels and $221 \times 228$ pixels.

![Architecture of the NIOSII Embedded System Using Custom Instructions](image11.png)

**Fig. 11: Architecture of the NIOSII embedded system using Custom instructions**

**TABLE 4**

<table>
<thead>
<tr>
<th>HW/SW Implementation Results of the NIOSII Embedded System Using Custom Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA resources</td>
</tr>
<tr>
<td>ALUTs</td>
</tr>
<tr>
<td>I/O</td>
</tr>
<tr>
<td>RAM Blocks</td>
</tr>
<tr>
<td>DSP Blocks</td>
</tr>
<tr>
<td>PLL</td>
</tr>
<tr>
<td>Fmax</td>
</tr>
</tbody>
</table>

According to figures 12 and 13, we notice a significant optimization in the execution time for the contrast (about 19%) and for low energy (about 1%). This is due to the energy equation which is less complex than the contrast one. The NIOSII processor is configured to perform multiplication by hardware in one clock cycle. In all cases, the time allotted to contrast and energy remains high. This time is due to memory accesses that are important. Finally, we notice from figures 12 and 13 that the HW/SW solution (using custom
instructions for the contrast and energy) allowed 35% of optimization in run time compared to the SW solution. This is true for the image sizes 106 × 160 pixels and 221 × 228 pixels. In conclusion, these two architectures allow a speed gain about 35%. This is due to the halving of the iteration number at the software level. In addition, this is due to the minimization of the complexity of calculation when we apply arithmetic operations in hardware while the contrast and energy instructions have taken only 1% of the FPGA area.

7 CONCLUSION
In this paper, we have proposed an acceleration method of the GCM algorithm based on the HW/SW co-design platform. This implementation is based on an embedded NiosII FPGA board, which is a methodology that takes into account the dependency between GCM components. The obtained results of HW/SW implementation are interesting. Indeed, since the whole run time of the method fell by 35% and preserved the same image quality. In fact, the algorithm was applied to multiple images taken from the ICDAR database. However, this model has a deficiency caused by is due to the memory occupation in the different stages of the GCM. Therefore, the integration of the GLCM helps to ameliorate the hardware implementation opportunity by adding a new IP (intellectual property) in the design. This idea is very interesting not only for our case study, but also for other similar algorithms.

8 ACKNOWLEDGMENT
We would like to thank our colleagues from ESIEE Paris, who provided us with insights and expertise that greatly assisted us to conduct this research work and then, improve it.

REFERENCES
[2]. Xu Guan, Su Jian, Pan Hongda, Zhang Zhiguo, Gong Haibin,
[6]. ElHarraj and N. Raisouini “OCR accuracy improvement on document images through a novel pre-processing approach” Signal & Image Processing : An International Journal (SIPIJ) Vol.6, No.4, August 2015
[12]. https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-niosii-2s60.html
Fig. 3: The GCM rules

Fig. 5: Custom instructions added to the NIOSII CUP
Fig. 12. CPU runtime using the SW solution (a) and HW/SW (b) using image size 106 × 160 pixels

Fig. 13: CPU runtime using the SW solution (a) and HW/SW (b) using image size 221×228 pixels

<table>
<thead>
<tr>
<th>(a1)</th>
<th>(b1)</th>
<th>(c1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contrast= 2495.65, Energy= 0.268467</td>
<td>γ=10</td>
<td>T=0.402344</td>
</tr>
</tbody>
</table>
Contrast = 745.576, Energy = 0.00184619, $\gamma = 6$, $T = 0.292969$

Contrast = 820.318, Energy = 0.00102407, $\gamma = 5.4$, $T = 0.261719$

**Fig. 14.** (a) original image (b) gamma correction effect (c) Output image