Hardware Implementation of Web Based Arabic Optical Character Recognition Units

Osama Al-Khaleel[a], Inad Aljarrah[a], Abdelrahman Idries[b], Khaldoon Mhaidat[a]
Department of Computer Engineering
Jordan University of Science and Technology
Irbid, Jordan 22110
Email[a]: {oda, inad, mhaidat}@just.edu.jo
Email[b]: aaidraisalsouqi11@cit.just.edu.jo

***Abstract*— Optical character recognition (OCR) is an important application in the field of pattern recognition. It extracts text from an image document and saves it in an editable form. Examples where OCR is used include library digitization and text searching in scanned documents. Web based applications are main tools for data processing over the net. However, implementing such applications in dedicated hardware systems would increase performance and reliability by many folds over software implementation. In this paper, we present a detailed hardware implementation of the features extraction and character matching units of an Arabic optical character recognition (AOCR) system. The hardware implementation of each of these two units is described in VerilogHDL and functionally tested using ISim from Xilinx. Furthermore, each implementation is synthesized using Xilinx ISE 13.1 targeting Xilinx Spartan6 FPGA family. Experimental results show significant speed up in the hardware implementations over software ones. We further, explore the possibility of accessing these systems over the Web. Thus, they are beneficial to wider range of people.**

***Index Terms*—AOCR, FPGAs, character matching, segmentation, hardware**

I. Introduction

Optical character recognition (OCR) is the method for converting a text document that is stored as an image into editable text. There are many applications that use optical character recognition such as searching scanned documents that are saved as images for a given text. Other applications include libraries digitization, reading filled forms of customers, and saving postal address off envelopes in text based files [1]. Due to its importance, OCR has been attracting researchers for the last four decades [1, 2]. For Arabic language, it is called Arabic optical character recognition (AOCR).

AOCR is harder to implement than OCR for other languages due to the continuous nature of the Arabic language writing. This continuity dictates that words must be segmented before each character is recognized.

OCR usually starts by dividing the words into characters. Each character is then recognized based on its feature descriptors. The text in the scanned document can be either handwritten or typewritten. In this work, only typewritten text will be considered.

Existing AOCR implementations are known to have long execution time because they are implemented in software and because they require intensive processing. For example, the segmentation process for the AOCR is repeated many times for each word to be divided into characters. Also, the feature descriptors are calculated for each character. Therefore, a hardware based implementation of AOCR is of great importance.

It can be easily figured out that extracting the feature descriptors and the character matching are the two main steps in AOCR. Thus, in this paper we propose a features extraction and a character matching units for Arabic characters recognition. The features that are under consideration are part of the features presented in [3] and [4]. Additional two features have been considered as will be shown in Section IV. The proposed character matching unit along with four different versions of the proposed features extraction unit have been investigated and implemented in Field Programmable Gate Array (FPGA).

Different web based OCR systems do exist. Examples of these are ABBYY FineReader [5], Free Online OCR [6], Free OCR [7], i2OCR [8], and OCRextrACT [9]. However, these systems are software based which dictates long processing time. Up to the knowledge of the authors, there is no hardware OCR implementation that is accessible through the web. Therefore, in this work we discuss the possibility of connecting the proposed hardware systems to the web as a high speed substitute for the software ones.

The rest of this paper is organized as follows: Section II presents the related work. Section III discusses the connectivity of hardware systems to the Web. Section IV and Section V discuss the proposed features extraction and character matching units respectively. Section VI talks about the experiments and gives some experimental results. Finally, the conclusion and the future work are presented in Section VII.

II. Related Work

Different approaches for Arabic optical character recognition have been proposed in literature. These

doi:10.4304/jetwi.6.2.210-219

approaches have been implemented in both software and hardware platforms. However, software implementations are known to be slow compared with the proper dedicated hardware ones. Actually, due to its complexity, few examples of hardware implementation are known to exist in literature.

The authors in [3] and [4] present a new method to segment and recognize typewritten Arabic characters. A gray scale image of the scanned Arabic text document is converted into binary image. Median filter is then applied on the binary image in order to remove the isolated pixels. The filtered image is divided into lines which are divided into words using segmentation algorithms. In the final stage they use character splitting and features extraction for character recognition.

An optical character recognition system for isolated Arabic characters is presented in [1]. It is implemented using the fixed-point digital signal processor (TI TMS320C6416T). Fuzzy art neural network is used to recognize the characters.

In [10], an automated recognition of handwritten Arabic characters is proposed. The approach is based on the geometrical features of the characters. Artificial neural network is adopted for characters classification.

One example of hardware implementation is presented in [11]. Altera FPGAs are targeted and an embedded design for Arabic optical character recognition is designed. The system consists of two parts: software part for image preprocessing and hardware part for image segmentation and character recognition.

Since there are similarities between Arabic and Persian characters, approaches for Persian character recognition might be applicable to Arabic character recognition. An FPGA implementation for Farsi handwritten digit recognition is proposed in [12]. The authors present a new method for feature extraction and a simple two layers Multi-Layer Perceptron (MLP) is used for digit classification.

One new algorithm for Persian handwritten characters recognition appears in [13]. The character is recognized by new decision tree using artificial neural network. A recognition accuracy of 98.72% has been achieved.

A novel Persian digits optical character recognition algorithm that is implemented in FPGA chip is presented in [14]. The approach mainly depends on the horizontal and vertical projections of digits. In this algorithm the image is converted into black and white image where the black color pixels (pixels with value of 0) represent the digit and the white color pixels (pixels with value of 1) represent the background. The features vectors are calculated for each digit. A feature vector contains the maximum number of black pixels in the horizontal and the vertical projections, the total number of the black pixels, and the number of the black pixels in the top left quarter of the image. In the recognition stage, the feature vector is extracted for the input digit and it is compared with all feature vectors that have been stored in a data base.

Much work has been done in the area of implementing part of server-client systems in hardware. One example is presented in [15] where a web server is designed using Xilinx MicroBlaze soft processor. The authors also explain a way where a client can connect to the web server running on the MicroBlaze soft processor.

Another web server design example by Altera is found in [16]. The design is based on Nios II development board.

In [17], the authors propose a system in which an image is captured by an image recorder and then it is sent to a PC. The PC establishes a communication link with an FPGA board and the image is transferred to the board for processing. An edge detection algorithm is implemented on the FPGA. The edge detection algorithm is applied on the image and then the processed image is transferred back to the PC for displaying. Different techniques are adopted for the implementation of the edge detection algorithm. In the first one, MicroBlaze is only used to run a software version of the edge detection algorithm. In the second implementation, MicroBlaze is used as the processor and QUKU as the hardware accelerator. QUKU is a coarse grained dynamically reconfigurable PE overlay for FPGAs. In the third implementation, MicroBlaze is used as the processor and a 4x4 coarse grained PE array as the hardware accelerator. Finally, in the fourth implementation, MicroBlaze is used as the processor and the IP based Sobel and Laplace circuits as the hardware accelerators. All of the communications between the PC and the FPGA board are carried out through an Ethernet link.

An FPGA-based web server that is targeted for biological alignment has been designed and implemented in [18]. In their approach a host application receives queries from users and submits them to the FPGAs coprocessors. Queries are initially stored in a MySQL database and are examined by the host application in order to figure out the proper FPGA configuration that is to be used. In order to serve many requests, many FPGA chips can be used and several processes would be run on each FPGA simultaneously. Upon finishing the processing, the host application collects the results and stores them in the database with a unique ID similar to the one that has been assigned to the incoming query. The user can access these results using their ID via a web interface.

The authors of [19] propose a system in which FPGA is configured via an Internet connection. The client uploads some data along with the FPGA configuration to a web server. An application running on the server performs some processing on the data sent by the client and the FPGA is then configured with the user configuration.

[20] and [21] propose a teaching environment in which the students can remotely access hardware resources for the purpose of performing experiments in the field of electronics and FPGAs respectively. The students are able to communicate with these hardware components using the web browser through a web server.

Authors in [22] develop an embedded web server using FPGAs. A client side application can communicate with the web server using HTTP protocol. The messages sent

by the client are displayed on a LCD on the server side. The web server is designed using MicroBlaze soft processor. A light MicroBlaze based embedded web server that hosts static web pages is introduced in [23]. The server enables HTTP file transfer and IO communication. The proposed system has been integrated in a network of wireless sensor nodes.

Novel web server architecture and its implementation using FPGAs are proposed in [24]. The request type that the authors adopt for the web clients requests in the system is the HTTP GET.

III. Connectivity to the Web

Different scenarios can be adopted for connecting hardware accelerators to the Web in order to maximize the utilization of these accelerators by exposing them to wide area of clients. In any of these scenarios, the client sends queries through the Web to the web server which guarantees the proper response to be sent back to the client. However, it can be figured out from Section II that in the server side different approaches may be used to implement the hardware accelerator system. One approach is to have a separate web server communicate with the hardware accelerator that is implemented in the FPGA. In this approach the web server receives the requests from the clients and delvers them to the hardware accelerator. The accelerator does the required processing and returns back the results to the server which sends it back to the client. Another approach is to have both the web server and the accelerator embedded on the same FPGA. Although this approach may complicate the design process and may require more hardware resources, it definitely speeds up the whole process as the need for separate web server is avoided. The clients send the request directly to the FPGA board which is connected to the web through an Ethernet port. The data is accepted by the embedded web server and directed to the accelerator within the same chip. The accelerator performs the processing and directs back the results to the embedded web server also internally within the chip. The embedded web server then sends back the results to the client. Usually, an embedded web server is designed by running web server over a soft processor like the MicroBlaze processor from Xilinx. In this case, the hardware accelerator would be hardcoded using other resources in the FPGA. However, with an external web server, the hardware accelerator is implemented using both ways. So it is either implemented in software and we let the software to run over the embedded soft processor or it is hardcoded using other resources in the FPGA chip.

Although all previously mentioned implementation approaches are feasible as these approaches or similar ones have been proposed in literature like those presented in [15-24], we will suggest to go for the approach where the web server is not embedded in the FPGA as this approach is much simpler to implement. However, some tuning and modifications are required in order to have the system function properly. Therefore in the following discussion we will present the details of how the proposed hardware systems can be accessed through the web.

In the proposed system, the queries consist of scanned text documents. On the web server side an application would carry out some preprocessing on the scanned text document so that it is ready to be forwarded to the hardware of the features extraction and character matching units for text recognition.

As shown by Fig. 1, using a Web interface, the user uploads the scanned text document to a Web server. The hardware system is attached to the server. A server side application preprocesses the scanned document. During the preprocessing stage, the image of the text document is converted from gray scale into binary scale. It is then

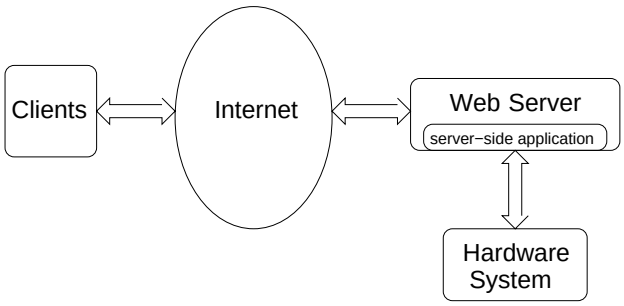

Figure 1. A block diagram to show how the hardware system is connected to the Web.

segmented into lines where each line consists of 26 rows of pixels and then into characters that are 16 columns by 26 rows. The preprocessing application sends 16 columns by 26 rows (that represent the first character in the scanned text document) to the system in a row by row fashion and waits for an acknowledgment from the hardware system. On its side, the hardware system recognizes the character presented in these 16 columns of pixels and sends the acknowledgment to the server application. The hardware system uses its features extraction units to compute the features vector for the entered data. The calculated features vector is compared concurrently with the pre-computed features vector of All Arabic characters using multiple character matching units. The matching results from all of the character matching units are fed into a priority encoder in order to decide the matched character. The highest priority is given to the first character in the Arabic alphabet which is the “Alif”. The second highest priority is given to the “Baa” which is the second character in the Arabic alphabet and so on. Afterward, the system returns back the recognized character to the server-side application. The system keeps this matching process until all characters in the document are recognized and the server-side application sends back the resulted text to the client through the web server.

In this study, standalone characters will be under consideration. The scanned text document is assumed to include only separate characters that are not connected to form words. Therefore, this work is character-level recognition not word-level recognition.

IV. Features Extracting NNIT

In this section we present the features extraction unit which has been originally proposed in [25] (an earlier conference version of this paper).

Arabic optical character recognition (AOCR) takes an image that represents a document of Arabic text. Part of the proposed work in this paper is the design of the character features extraction of an AOCR. Therefore, the input is presumed to be an image of a segmented Arabic character. Many features of each character are extracted from the input image. These features can be compared with the pre-calculated features that are usually stored in a database. The features adopted here are similar to those presented in [3] with the feature extraction process being implemented in a hardware platform instead of the software platform used in [3]. The hardware implementation is built on FPGAs.

For the feature descriptors of a character to be extracted from an image, preprocessing operations like converting the colored or gray scale image into binary image are performed. Binary images have enough information for character recognition, so dealing with colored or gray scale images would add up unnecessary processing overhead. Besides that, processing a binary image is much faster since pixels are represented by a single bit. The gray to binary conversion process is attained by simply replacing each pixel value by either 1 or 0 based on a particular threshold value. The threshold value is usually calculated based on the histogram distribution of the given image. For example, in the gray scale images, each pixel is represented by 8 bits. Therefore, the maximum pixel value is 255 and the threshold value is set to a value between 0 and 255. In the simplest case, the value is set to 128. Thus, if the most significant bit in a pixel value is 1 then the pixel value is set to 1; otherwise the pixel value is set to 0. An example to illustrate the gray scale to binary image conversion for this case is shown in Fig. 2.

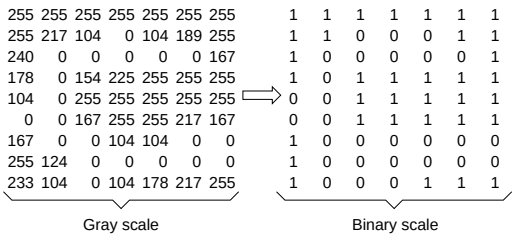

Figure 3. Gray scale to binary image conversion example.

As soon as the image is converted to binary, it is set for processing over the hardware. The image is transferred into the FPGA and the pixel values are saved in the distributed RAM elements which are storage units in the FPGA.

The dimensions of the character image are 26 rows by 16 columns, which sums to an area of 416 pixels. Rows of the image are entered to the system serially. Since there are 26 rows, a total of 26 clock cycles are required for the image to be stored in the distributed RAM elements. After the data entry process is finished, the rows of the image are fed internally to the feature descriptors calculation modules in order to compute the wanted features. The outputs of these modules are compared with the stored features in order to identify the character. The entire process is depicted in Fig. 3 which represents the features extraction unit.

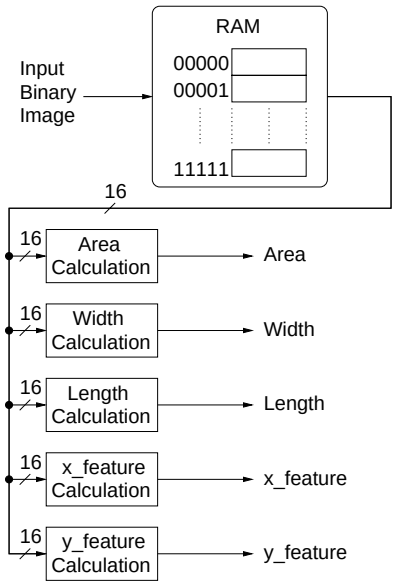

Figure 2. Features extraction unit.

The feature descriptors for each character are unique and can distinguish the characters from each other. The feature descriptors that have been used in [3] and are being implemented using hardware in this work are: character area, character length, and character width. Two more features are added in this work which are: the *x_feature* and the *y_feature*. A complete description of each of these features and their hardware implementation will be presented next.

The area of each character denotes the sum of the pixel values of the character. Each input image consists of the pixel values of the character and the pixel values of the background. The image is binary and the pixel values of the background are all zeros. Whereas the pixel values of the character are ones. Therefore, the area of the character is simply evaluated by summing all the pixel values of the image since the 0's values do not contribute to the result. Fig. 4. shows the hardware module to accomplish the calculations. The pixel values in each row are summed together and to the sum of the pixel values from the previous row. The content of the register is initially cleared to 0 so that the sum of the pixel values of the first row is added to 0. This accumulation of the sum of the pixel values is repeated for 26 times until all rows are added.

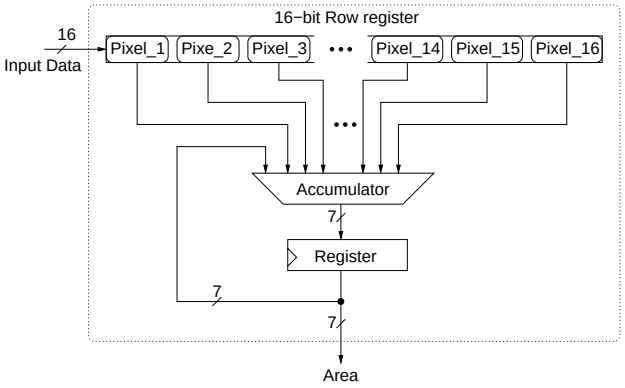

Figure 4. Area calculation module.

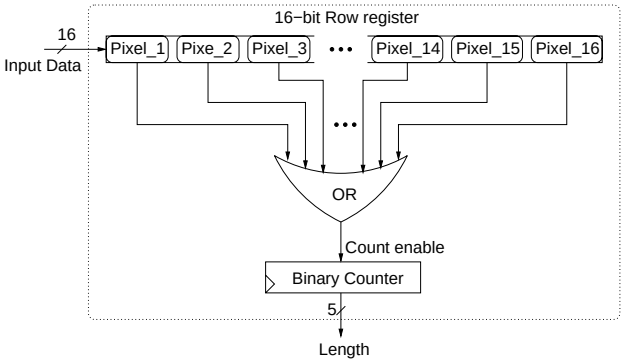

Figure 5. Length calculation module.

The length of a character is calculated by counting the number of rows in the binary image where the row has at least one pixel of value 1. As shown in Fig. 5, the length calculation module is very simple. The 16-bit register holds the input row while the OR gate checks for the occurrence of 1 among the pixel values in that row. If the output of the OR gate is 1 then the count enable of the counter is enabled, which results in it counting up by one. After each clock cycle a row is shifted into the 16-bit register. Thus, 26 cycles are required to shift in all rows.

On the other hand, the width of a character is the number of the columns that have at least one pixel value of 1 in the binary image. One possible way to calculate the width is to use a method similar to that used for computing the length with the columns being entered sequentially instead of the rows. However, this would require reading the image twice and so will impose an extra delay. Another alternative is to calculate the width during the row by row reading of the binary image. The idea is based on the fact that if a pixel value in the row is 1 then the corresponding column which contains that pixel is counted for the final value of the width. Therefore, as shown by Fig. 6, during reading the rows, the pixel values are tracked.

If a pixel value during reading any row happens to be 1 then its corresponding flip flop is set to 1 since the flip

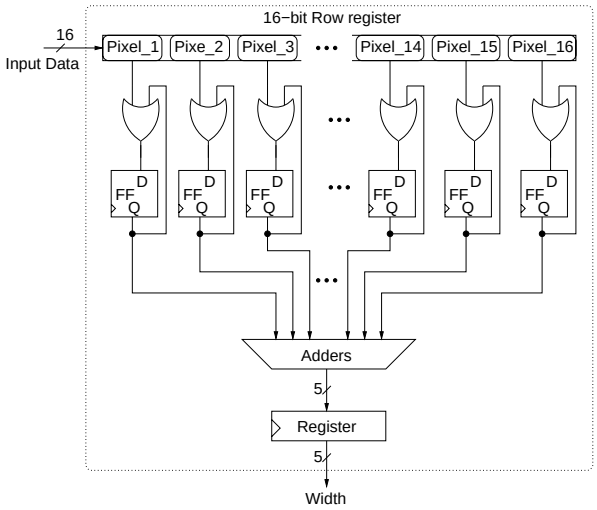

Figure 6. Width calculation module.

flops are initially reset. The feedback from the output of the flip flop guarantee's that any change on the pixel value to 0 would not affect the flip flop output. After processing all rows the flip flops outputs reflect the number of the columns that amount for the width value. The outputs of the flip flops are added in the final stage to compute the value of the width.

The *x_feature* is a value that is calculated based on the *x*-coordinate for all pixels in the image. Any pixel in the binary image can be referenced by one point (*x*-coordinate , *y*-coordinate) in the two dimensional plane and according to its position in the image. It should be pointed out that the *x*-coordinate and the *y*-coordinate of the top left pixel in the binary image is 1 and 1 respectively and the *x*-coordinate increases horizontally from left to right. Whereas, the *y*-coordinate increases vertically from top to bottom.

As shown by Equation 1, *x_feature* is calculated by adding up the *x*-coordinate for all pixels that have value equals to 1 and subtracting the amount $k \times A$ from the total sum.

$$x\_feature = \left(\sum_{i=1}^{26} \sum_{j=1}^{16} x_j \times V_{ij} \right) - k \times A \qquad (1)$$

where:

x_j is the *x*-coordinate of the (i,j) pixel.

$V_{ij} \in \{0,1\}$ is the pixel value of the (i,j) pixel.

A is the area of the character.

k is the number of columns that are to the left of the beginning of the character pixels in the image (All of these columns have their pixel values equal to 0).

The term $k \times A$ is subtracted from the total sum so that the effect of *x*-coordinate of the first character pixel, from left, on the calculation of the *x_feature* happens as if the *x*-coordinate of the pixel is 1. This would propagate to all other pixels in the columns which results in having the *x_feature* independent from the actual position of the character in the image.

An exact hardware implementation of Equation 1 is shown in Fig. 7. However, since V_{ij} is either 0 or 1, the multipliers can be replaced with multiplexers to reduce the total delay and area of the design as will be shown in Section V.

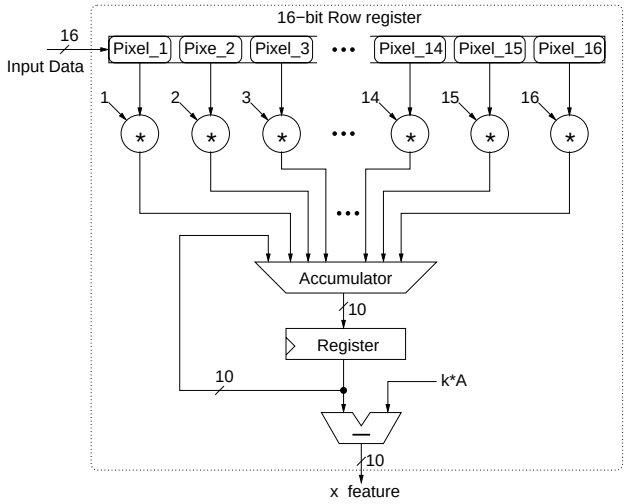

Figure 7. *x_feature* calculation module.

On the other hand, *y_feature* is calculated by adding up the *y*-coordinate for all pixels that have value equals to 1 and subtracting the amount $l \times A$ from the total sum as given by Equation 2.

$$y\_feature = \left(\sum_{j=1}^{16} \sum_{i=1}^{26} y_i \times V_{ij} \right) - l \times A \qquad (2)$$

where:

y_i is the *y*-coordinate of the (i,j) pixel.

$V_{ij} \in \{0,1\}$ is the pixel value of the (i,j) pixel.

A is the area of the character.

l is the number of rows that are above the beginning of the character pixels in the image (All of these rows have their pixel values equal to 0).

The term $l \times A$ is subtracted from the total sum so that the effect of *y*-coordinate of the first character pixel, from top, on the calculation of the *y_feature* happens as if the *y*-coordinate of the pixel is 1. This would propagate to all other pixels in the rows which also results in having the *y_feature* independent from the actual position of the character in the image.

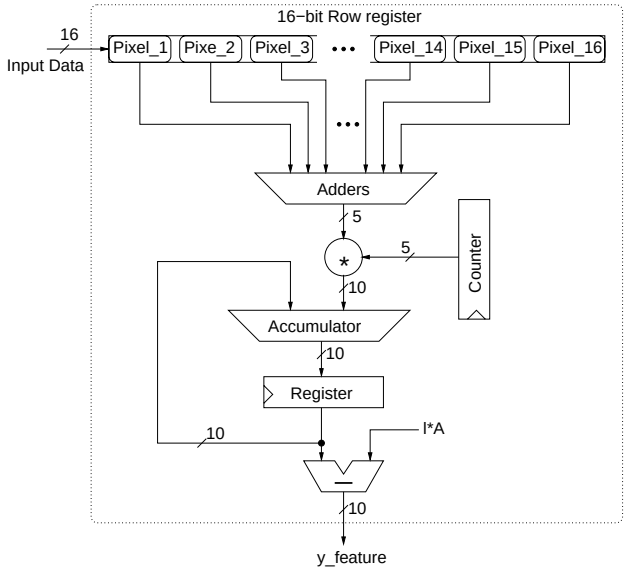

Figure 8. *y_feature* calculation module.

Since the image is being read row by row and in order to avoid reading the image twice, Equation 2 can be reordered to yield a more suitable formula. This can be carried out based on the fact that all the pixels in the same row have the same *y*-coordinate. The reordering process produces Equation 3 which can be exactly implemented in hardware as shown in Fig. 8.

$$y\_feature = \left(\sum_{i=1}^{26} \left(y_i \times \sum_{j=1}^{16} V_{ij} \right) \right) - l \times A \qquad (3)$$

An example to demonstrate the features calculation is shown in Fig. 9. In this example, the area of the character is 9, the length of the character is 4, the width of the character is 4, the *x_feature* is 23, and the *y_feature* is 21.

0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	0
0	1	1	0	0	0
0	0	0	1	1	0
0	0	0	0	0	0

Figure 9. An example to demonstrate the features calculation.

Due to the fact that the features extraction unit and its modules are designed such that the binary image is read once, the distributed RAM elements can be removed from the design without affecting the final results. This would not have been possible if the image is to be read more than one time. During the FPGA implementation phase, both options have been investigated as presented in Section V.

V. CHARACTER MATCHING UNIT

The character matching unit accepts the features vector that is attained by the features extraction unit along with the features vector that is pre-computed and stored in the system. In this paper, we will assume that the computed and the stored feature vectors are *F* and *G* respectively and are defined as follows:

$F=\{A_n, W_n, L_n, X_n, Y_n\}$.

$G=\{A_s, W_s, L_s, X_s, Y_s\}$.

where A_n, W_n, L_n, X_n, and Y_n are the computed *area*, *width*, *length*, *x_feature*, and *y_feature* respectively and A_s, W_s, L_s, X_s, and Y_s are the stored *area*, *width*, *length*, *x_feature*, and *y_feature* respectively.

These two vectors are compared for the purpose of character matching. The comparison process is based on calculating the dot product of the two vectors and dividing by the product of the magnitude of the two vectors. The outcome of these calculations is called *matching_value* as defined by Equations 4, 5, 6, and 7.

$$matching\_value=\frac{F\cdot G}{|F||G|} \quad (4)$$

$$F\cdot G=A_nA_s+W_nW_s+L_nL_s+X_nX_s+Y_nY_s \quad (5)$$

$$|F|=\sqrt{A_n^2+W_n^2+L_n^2+X_n^2+Y_n^2} \quad (6)$$

$$|G|=\sqrt{A_s^2+W_s^2+L_s^2+X_s^2+Y_s^2} \quad (7)$$

The character matching unit is depicted in Fig. 10. If the *matching_value* is less than a threshold value then a matching is achieved and the process is restarted for a new matching. Otherwise; the computed feature vector is compared with the other stored feature vectors until a matching is achieved. As an example, assume:

The computed features vector is F={21,7,5,21,19}

The pre-stored one is G={20,8,7,18,20} then

$F\cdot G$=(21×20)+(7×8)+(5×7)+(21×18)+(19×20)=1269

$|F|=\sqrt{21^2+7^2+5^2+21^2+19^2}\cong 36.3$

$|G|=\sqrt{20^2+8^2+7^2+18^2+20^2}\cong 35.2$

$matching\_value=\frac{1269}{36.3\times 35.2}=0.993$

It should be pointed out that the comparison process is started with the stored feature vector of first letter in Arabic language which is the "Alif", then the second, third and so on. In order to achieve higher performance, during the implementation phase, we let the computed vector be concurrently compared with all stored features vectors using multiple character matching units. The $(matching\_value)^2$ (Equation 8) computation is implemented as it is more convenient for hardware than the (*matching_value*) itself. This would not affect the matching process as the square of (*matching_value*) is proportional to (*matching_value*). Thus, the decision can be made based on (*threshold*) and calculating the square root is avoided. Moreover, to simplify the implementation of the division operation, we compute $100\times(matching\_value)^2$ and we compare with 100×(*threshould*) which is two decimal digits. The pseudo code to compute the value $100\times(matching\_value)^2$ is depicted in Algorithm 1.

$$(matching\_value)^2=\frac{(F\cdot G)^2}{|F|^2|G|^2} \quad (8)$$

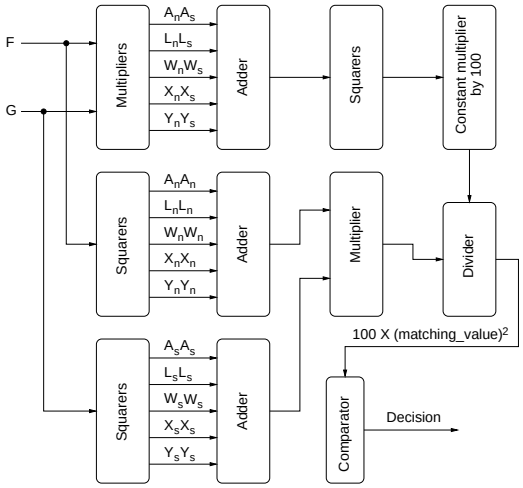

Figure 10. Character matching unit.

Algorithm 1: Computing $100\times(matching\_value)^2$

Input: A_n, W_n, L_n, X_n, Y_n
Output: D
/* A_n, A_s:new and stored area */
/* W_n, W_s:new and stored width */
/* L_n, L_s:new and stored length */
/* X_n, X_s:new and stored x_feature */
/* Y_n, Y_s:new and stored y_feature */
/* D:Decision */
/* Th:Threshold */
begin
  $Counter \leftarrow 0$;
  $D \leftarrow NO$;
  while D=NO **do**
    Read new $(A_s, W_s, L_s, X_s, Y_s)$;
    $T_1 \leftarrow sum(A_nA_s, W_nW_s, L_nL_s, X_nX_s, Y_nY_s)$;
    $T_2 \leftarrow (A_n^2 + W_n^2 + L_n^2 + X_n^2 + Y_n^2)$;
    $T_3 \leftarrow (A_s^2 + W_s^2 + L_s^2 + X_s^2 + Y_s^2)$;
    $T_4 \leftarrow 100 \times T_1 \times T_1$;
    $T_5 \leftarrow T_2 \times T_3$;
    while $T_4 > T_5$ **do**
      $T_4 \leftarrow T_4 - T_5$;
      $Counter++$;
    if $Counter > Th$ **then**
      $D \leftarrow YES$;
      Report matched character;
      Restart for new character matching;
    else
      Continue;

V. Experimental Results

The proposed features extraction and character matching units have been described and functionally tested using VerilogHDL and ISim from Xilinx. The features extraction unit has been tested for all Arabic characters and the outputs from the unit are listed in Table I. The font type is bold times new roman and the font size is 11.

In fact, four different versions of the features extraction unit have been tested and implemented. In the first version, multipliers are used in the *x_feature* calculation module and the unit is implemented with distributed RAM elements. In the second version, MUXs are used in the *x_feature* calculation module instead of the multipliers and the unit is implemented with distributed RAM elements. In the third version, multipliers are used in the *x_feature* calculation module and the unit is implemented without distributed RAM elements. Finally, In the fourth version, MUXs are used in the *x_feature* calculation module instead of the multipliers and the unit is implemented without distributed RAM elements.

TABLE I
FEATURES FOR EACH ARABIC CHARACTERS BASED ON THE PROPOSED FEATURES EXTRACTION UNIT.

Char.	*area*	*length*	*width*	*x_feature*	*y_feature*
ا	14	9	2	20	70
ب	32	8	10	182	150
ت	32	7	10	181	162
ث	37	9	10	207	236
ج	45	11	9	195	254
ح	40	11	9	169	224
خ	45	14	9	190	394
د	15	7	4	45	72
ذ	21	10	4	58	145
ر	19	8	7	97	105
ز	24	11	7	123	191
س	42	8	14	319	209
ش	52	12	14	435	406
ص	55	9	16	507	293
ض	60	12	16	563	523
ط	42	9	9	227	256
ظ	47	10	9	262	308
ع	43	12	8	167	298
غ	49	15	8	190	481
ف	44	9	12	337	338
ق	45	11	9	276	350
ك	42	10	8	222	279
ل	33	13	7	153	278
م	24	9	6	80	102
ن	31	11	8	153	235
ه	18	6	4	42	69
و	24	8	7	123	118
ي	36	9	8	178	199

All these versions have been synthesized and implemented using Xilinx ISE 13.1 targeting Xilinx Spartan-6 FPGA family (xc6slx150t-3fgg676). The experimental results for the four different implementations that have been investigated in this work are listed in Table II.

It is clear from Table II that the fourth version requires the least number of slice LUTs. On the other hand, the best delay and the lowest total power dissipation are achieved by the third version. It is also clear that the highest throughput is achieved by the first and the third versions since they have the highest frequency.

TABLE II
THE IMPLEMENTATION RESULTS FOR THE FOUR DIFFERENT VERSIONS THAT HAVE BEEN INVESTIGATED.

	With distributed RAM		Without distributed RAM	
	using multiplier	using MUXs	using multiplier	using MUXs
Number of Slice registers	91	94	75	78
Number of Slice LUTs	253	223	242	214
Execution time(*ns*)	547	555	279	283
Maximum Frequency (MHz)	96.9	95.5	96.9	95.5
Power (*mW*)	3.81	4.21	3.76	4.21

In addition, the character matching unit has been synthesized and implemented using the same environment (Xilinx ISE 13.1 targeting Xilinx Spartan-6 FPGA family (xc6slx150t-3fgg676)) and the results are reported in Table III.

For comparison purposes between the hardware and the software implementations, the execution time spent by the proposed character matching unit has been compared with the execution time required by the software implementation. The hardware implementation of the proposed character matching unit has been carried out using FPGAs while the the software implementation uses

TABLE III
THE IMPLEMENTATION RESULTS FOR THE CHARACTER MATCHING UNIT.

Parameter	**Result**
Number of Slice registers	1596
Number of Slice LUTs	56864
Maximum Frequency (MHz)	160.7

C++. The code of the character matching unit has been run on Pentium 4 processor with 3 GHz clock frequency and 1 GB RAM and the results are listed in Table IV. It is clear that the hardware implementation is much faster than the software implementation.

V. Conclusion and Future Work

Arabic character features extraction and character matching units have been proposed. The features extraction unit calculates the feature descriptors for an Arabic character. While, the character matching unit compares the calculated feature descriptor with pre stored ones for character matching. The feature descriptors include the area, the width, and the length of the character. Both of the proposed units can be employed in any AOCR system. The connectivity of these units to the Web has been discussed. The character matching unit along with four versions of the features extraction unit have been investigated and implemented in FPGA using VerilogHDL and Xilinx ISE 13.1 CAD tools. Experimental results are promising. As a next step, more features will be considered such as the centroid of the character and the number of its objects. In addition, an integrated system that segments the words and the characters and recognizes them will be developed.

TABLE IV
THE EXECUTION TIME NEEDED BY THE CHARACTER MATCHING UNIT AND THE SOFTWARE IMPLEMENTATION

Approach	Worst case execution time
Software implementation	2.2 μs
Proposed unit in FPGAs	0.616 μs

References

[1] H. Almohri, J. S. Gray, and H. Alnajjar, "A real-time DSP-based optical character recognition system for isolated arabic characters using the TI TMS320C6416T," *The 2008 IAJC-IJME International Conference*, November 2008.

[2] M. Kavianifar and A. Amin, "Preprocessing and structural feature extraction for a multi-fonts arabic/persian OCR," in *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, September 1999, pp. 213–216.

[3] I. Aljarrah, O. Al-Khaleel, K. Mhaidat, M. Alrefai, A. Alzu'bi, and M. Rabab'ah, "Automated system for arabic optical character recognition with lookup dictionary," *Journal of Emerging Technologies in Web Intelligence*, vol. 4, no. 4, 2012.

[4] I. A. Aljarrah, O. D. Al-Khaleel, K. Mhaidat, M. Alrefai, A. Alzu'bi, and M. Rabab'ah, "Automated system for arabic optical character recognition," in *Proceedings of the 3rd International Conference on Information and Communication Systems*, ser. ICICS '12. plus 0.5em minus 0.4emNew York, NY, USA: ACM, 2012, pp. 5:1–5:6.

[5] ABBYY, "Abbyy finereader online," http://finereader.abbyyonline.com/en.

[6] OnlineOCRService, "Free online ocr," http://www.onlineocr.net/.

[7] FreeOCR, "Free ocr," http://www.free-ocr.com/.

[8] i2OCR, "i2ocr," http://www.i2ocr.com/.

[9] OCRextrACT, "Ocrextract," http://www.ocr-extract.com/.

[10] M. M. Fahmy and S. A. Ali, "Automatic recognition of handwritten arabic characters using their geometrical features," *Studies in Informatics and Control*, vol. 10, no. 2, 2001.

[11] A. Al-Marakeby, F. Kimura, M. Zaki, and A. Rashid, "Design of an embedded arabic optical character recognition," *Journal of Signal Processing Systems*, vol. 70, pp. 249–258, 2013.

[12] M. Moradi, M. A. Pourmina, and F. Razzazi, "FPGA-based farsi handwritten digit recognition system," *International Journal of Simulation Systems, Science and Technology*, vol. 11, no. 2, 2010.

[13] M. Rajabi, N. Nematbakhsh, and S. A. Monadjemi, "Article: A new decision tree for recognition of persian handwritten characters," *International Journal of Computer Applications*, vol. 44, no. 6, pp. 52–58, April 2012, published by Foundation of Computer Science, New York, USA.

[14] N. Toosizadeh and M. Eshghi, "Design and implementation of a new persian digits ocr algorithm on fpga chips," in *European Signal Processing (EUSIPCO2005), 13th Conference*, Antalya, Turkey, September 2005.

[15] M. Muggli, M. Ouellette, and S. Thammanur, "Embedded system example: Web server design using microblaze soft processor," Xilinx, Application notes 433, August 2004.

[16] ALTERA, "Web server design example," http://www.altera.com/support/examples/nios2/exm-micro\s\do5(t)utorial.html.

[17] S. Shukla, N. W. Bergmann, and J. Becker, "A web server based edge detector implementation in fpga," in *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, 2008, pp. 441–446.

[18] Y. Liu, K. Benkrid, A. Benkrid, and S. Kasap, "An fpga-based web server for high performance biological sequence alignment," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, 2009, pp. 361–368.

[19] O. Cret, Z. Baruch, and K. Pusztai, "Fpgaw: Fpga configuration over the internet," in *Proceedings of the 3rd International Conference on Mathematical and Computational Applications (ICMCA'2002)*. plus 0.5em minus 0.4emKonya, Turkey: Selɜuk University, Septemper 2002.

[20] A. F. Herrero, I. Elguezábal, and M. L. Vallejo, "A web-based environment providing remote access to fpga platforms for teaching digital hardware design." in *e-Learning*, 2008, pp. 161–165.

[21] A. Fernández-Herrero, P. Ituero, M. López-Vallejo, and F. G. Redondo, "Web-based integrated environment for self-learning electronics: the analog and digital laboratory at home," in *Fourth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, 2011, pp. 124–129.

[22] R. S. Soni and D. Asati, "Development of embedded web server configured on fpga using soft-core processor and web client on fpga using soft-core processor and web client on pc," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 1, no. 5, pp. 295–298, June 2012.

[23] P. B. Reddy, K.Soundararajan, and M. Prasad, "Implementation of light weight internet controlled web server in embedded systems," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 2, no. 1, pp. 183–188, 2013.

[24] J. Yu, Y. Zhu, L. Xia, M. Qiu, Y. Fu, and G. Rong, "Grounding high efficiency cloud computing architecture: Hw-sw co-design and implementation of a stand-alone web server on fpga," in *Applications of Digital Information and Web Technologies (ICADIWT), 2011 Fourth International Conference on the*, 2011, pp. 124–129.

[25] O. Al-Khaleel, A. Idries, K. Mhaidat, and I. Aljarrah, "FPGA-based features extraction unit for arabic characters," in *Proceedings of The International Conference on Information and Communication Systems (ICICS 2013)*, Irbid, Jordan, April 2013.