FPGA Parallel Recognition Engine for Handwritten Arabic Words

Ashraf Suyyagh

Department of Computer Engineering, The University of Jordan,
Amman, 11942, Jordan
mrsuyyagh@gmail.com

Gheith Abandah

Department of Computer Engineering, The University of Jordan,
Amman, 11942, Jordan
abandah@ju.edu.jo
Tel: +962-79-5603-098
Fax: +962-6-530-0813

August 2013

Arabic optical character recognition (OCR) has been an active field of research for decades. Yet, it still has limited application domains mainly constrained to printed text. One reason of this limitation is the high time complexity of the current Arabic OCR algorithms. Accelerating the execution of these algorithms would open the door for applications in real-time assistive technologies, office automation, and portable devices. Modern programmable hardware devices like the field-programmable gate arrays (FPGAs) have numerous logic resources that allow parallel implementations of many algorithms. In this paper, we investigate implementing the feature extraction and classification stages of handwritten Arabic words on FPGAs. We study the performance and cost of four commonly-used feature extraction techniques and neural network classifiers on images from the IFN/ENIT database of handwritten Arabic words. The most efficient feature extraction technique and the best neural network found are implemented in parallel on an Altera FPGA. Multiple FPGA implementations with varying costs and performances are evaluated. The FPGA implementations are 20–200 times faster than the MATLAB software implementation. An implementation that only consumes about one quarter of the FPGA resources is 20 times faster than the software implementation and is less accurate by only 2.8%.

Keywords: Handwritten Arabic; OCR; FPGA; feature extraction; neural networks.

1. Introduction

Optical character recognition (OCR) is the electronic translation of printed or handwritten text into machine encoded electronic format. This allows for the text to be edited, searched, and stored more efficiently. OCR research for different languages reached different stages of development and maturity. Many researchers achieved near perfect accuracy results for printed text. This has led to OCR application in many consumer products and digitization projects [1].

Arabic is the native language of more than 440 million people and its alphabet is used in around 27 languages, including Arabic, Persian, Kurdish, Urdu, and Jawi [2]. Recognition rates for printed Arabic reach 99% accuracy for clean input on open vocabulary. Meanwhile, recent research results for handwritten Arabic text indicate that there is ample room for improvement [3]. In the latest International Conference on Document Analysis and Recognition (ICDAR 2011) Arabic handwriting recognition competition, the best system achieved only 92% and 85% recognition accuracies on two test sets of limited lexicon [4]. Consequently, only printed Arabic text has some commercial applications. Yet, in contrast to other languages, specifically the Latin languages, the application domain of Arabic OCR is still limited. For example, Latin OCR is used in real-time assistive technology devices, office automation, and portable applications [5]. One reason for this is the time-consuming algorithms used in Arabic OCR. Such complex algorithms are needed to address the cursive nature and peculiarities of the Arabic script.

Recent average times for recognizing a single handwritten Arabic word range from 0.1 to 17.8 seconds, where most systems report average times around 2.5 seconds per word [6]. Moreover, practical use of printed Arabic OCR software in large library digitization projects failed. This is due to low accuracy rates coupled with slow recognition speeds [7]. Therefore, for the expansion of Arabic OCR application, high-speed and highly-accurate algorithms are needed. However, there is often a tradeoff between OCR accuracy and speed [8]. Hardware programmable devices such as field programmable gate arrays (FPGAs) could serve as an acceleration platform for current and future complex algorithms.

FPGAs are configurable logic devices which have successfully been deployed in real-time image and video processing. They offer orders of magnitude of acceleration due to their potential for high-level of parallelism [9]. Furthermore, they have high computational density which could accommodate the implementation of resource-consuming and complex algorithms. Also they have low power consumption required in many embedded applications. Since OCR and image processing share many algorithms, FPGAs serve as a suitable Arabic OCR acceleration platform. This would allow the development of portable and low-power embedded solutions for Arabic OCR. Such solutions can be used in banking operations, mail services, office automation, and assistive devices, which will help millions of people [10].

In this paper, we investigate the feasibility of programmable hardware as a platform for recognizing handwritten Arabic words in limited vocabulary applications such as bank checks processing. The main aim of this study is to analyze some of the most commonly used feature extraction techniques, and to determine which is more efficient to implement on hardware. Efficiency is measured in terms of recognition accuracy and hardware resources cost. Moreover, we aim for an implementation which would deliver significant speed up over their software counterparts. In our analysis, we concentrate on the feature extraction and classification stages because they are the core of the OCR engines and are typically the most complex and time-consuming stages. Feature extraction has received the attention of many Arabic handwritten script researches. This is due to the fact that finding a set of features which can sufficiently describe and recognize Arabic words is important and not an easy task [11]. For the classification stage, we design and implement a neural network classifier tuned for our purposes.

This paper surveys the current literature regarding FPGA-based OCR systems in Section 2. Section 3 presents the methodology and tools used in this research. The used feature extraction techniques are described in Section 4. The sensitivity analysis, cost estimations, and efficiency analysis of the suggested feature extraction techniques are presented in Section 5. System design and implementation is described in Section 6, results analysis follows in Section 7, and the paper concludes with a summary and recommendations for future work in Section 8.

2. Literature Review

Only recently OCR implementations have emerged on hardware platforms. These hardware implementations have been modest to say the least. They only target some preprocessing steps or full systems at the character recognition level. Beg presented a Verilog model of an artificial neural network for isolated Arabic character recognition [12]. The letters are presented as 8×8-pixel grids to the network constituting 64 features. The binary pixel values don't need multiplication-intensive operations. Instead, the weights are added depending on the pixel value. Three-piece piecewise linear approximation is used to approximate the network transfer function. The recognition accuracy reported is 80.3%.

Al-Marakeby *et al.* use the hardware/software co-design approach to build an embedded system for recognizing Arabic characters [13]. They extract from the character image a varying number of Zernike moments, and they found that the recognition accuracy significantly improves when extracting more moments. A recognition accuracy of 99.6% is achieved when extracting 40 moments. The authors reported that large speed gain is achieved when the preprocessing stage is carried out using a SIMD processor instead of a single-processor, software implementation.

Printed and handwritten Farsi digits FPGA-based OCR systems are introduced in Refs. [14] and [15], respectively. In the former system, pixel density and projection are used on a set of 10×7 printed digits to achieve 100% accuracy. In the latter system, statistical information about pixel distribution in the upper, lower, right, and left halves of a 40×40 normalized binary handwritten digit image is used in the feature extraction stage. Additional features are extracted from the number of intersections of multiple crossing rays within the digit. Using an 11/16/10 neural network topology with log sigmoid transfer function, an accuracy of 96% is reported.

The preprocessing steps of line and character segmentation are developed on FPGAs for Jawi text in Refs. [16] and [17]. Both systems were based on histograms for line and character separation and achieve 97% and 98% accuracies, respectively.

FPGA-based systems to recognize the printed 26 letters of the English language are introduced in Refs. [18] and [19]. While the latter is based on simple template matching technique, the former uses nearest distance search algorithm. It applies associative memory to achieve 99.5% accuracy on 16×16-pixel characters. Srinivasan *et al.* analyzed and tuned a reference implementation of the OCR workload on a low-power, general-purpose processor and identified the hotspot functions that incur a large fraction of the overall response time [20]. Also they implemented an FPGA hardware accelerator for one of the hotspot functions to further reduce the execution time and power.

All systems surveyed above handle only character or digit recognition and classification. Input images are introduced in small window sizes. None of the systems tackle the complex problem of word-level classification or input images with large window sizes. Furthermore, they rarely report the hardware costs of such systems in terms or area, or maximum system speed ups or overall system efficiency. In our research, we investigate the recognition of complex cursive Arabic words introduced in large window sizes on FPGAs. We report the hardware cost, efficiency, and speedups over software solutions.

3. Methodology

In this section, we present a general overview of the methodology undertaken in carrying out this research, the database of handwritten Arabic words used, the development environment and tools used, as well as the word classifier of choice.

3.1. *Overview*

Fig. 1 summarizes our research steps and methodology. The first step is adopting the IFN/ENIT database of handwritten Arabic words and selecting a subset of these words suitable for our purpose. We also preprocess these words to facilitate the next steps as detailed in Subsection 3.2.

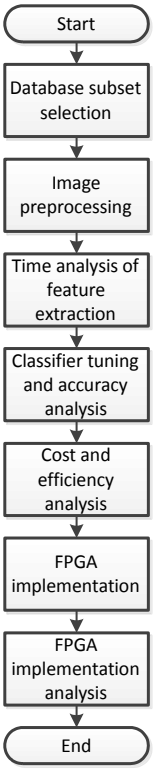

Fig. 1 Steps of this research

We have selected four feature extraction techniques for investigation. These four techniques are commonly-used techniques in Arabic handwritten OCR research. These techniques are discrete cosine transform (DCT), density, gradient, and Hu moments and their detailed description is in Section 4. These techniques are implemented in MATLAB and their execution time is evaluated; see Subsection 5.1.

We selected neural networks for handwritten Arabic word samples classification. Sensitivity analysis is carried on multiple neural network alternatives to reach a neural network that has satisfactory accuracy. The neural network classifier is evaluated with the four feature extraction techniques. The type of neural network used is described in Subsection 3.4 and the results of this analysis are in Subsection 5.2.

Hardware cost estimates in terms of logic resources consumed based on parallel implementations of the algorithms follows; see Subsection 5.3. This estimation is based on approaches proposed in the literature with slight modifications. Efficiency of the feature extraction techniques is found in terms of recognition accuracy and hardware cost; see Subsection 5.4.

The most efficient technique is adopted and implemented on Altera FPGA board. This board is described in Subsection 3.3. This implementation includes parallel density features extraction and neural network classifier that are adapted to fit on the limited hardware resources of the FPGA. This implementation is described in Section 6.

Finally, some design alternatives are investigated and evaluated. Section 7 presents the result of this evaluation and compares the FPGA implementation against its software counterpart in terms of time, accuracy, and hardware cost.

3.2. *The IFN/ENIT database and subset selection and preprocessing*

The IFN/ENIT database is a freely available database of handwritten Arabic words. It is developed to advance the research and development of Arabic handwritten word recognition systems. It was first introduced at the CIFED02 in 2002 [21]. The database is in Version 2.0 patch level 1e (v2.0p1e) and consists of 32,492 images of Arabic words handwritten by more than 1,000 writers. The word set is 937 Tunisian town/village names. The database is organized in five disjoint sets (from A to E)

for the purpose of training and testing. It has been used by more than 110 research groups in about 35 countries and is the reference database in the ICDAR Arabic handwriting recognition competition [4].

This database has more words than what is needed in our application target domains, such as bank checks processing. Therefore, we selected a subset of this database which includes all sample words that have a minimum of 75 occurrences each. This subset has 50 different words and a total of 8,699 word images.

All images are scaled to a fixed size of 64×256 pixels using the batch resize tool with the Lanczos filter of the software ACDSee Pro. All images are then thinned using MATLAB's utility of morphological operations on binary images. Scaling is needed to present the hardware with fixed-size data to simplify system design and implementation. Thinning is used as a normalization technique which erodes stroke thickness that varies from one writer to another.

3.3. *Development environment and FPGA board*

The 32-bit version of MATLAB R2009a Version 7.8.0.347 is our main software platform for algorithm analysis. Neural network code is generated using MATLAB's neural network pattern recognition tool. All software is installed on a fresh installation of a 64-bit Windows 7 Ultimate running on a PC. The processor is an Intel Core T6600 running at 2.2 GHz and equipped with 2-MB L2 cache. This PC has 4-GB DDR2 main memory with 800-MHz bus. Fixed-point arithmetic simulation is performed using MATLAB's fixed point tool. Further fixed-point simulations are performed using C code complied using the gcc compiler 4.4.3 on a fresh installation of Ubuntu 10.04 LTS. Ubuntu is installed in a virtual environment using VMWare Workstation 7.1.2. The image processing software ACDSee Pro 4 is used for batch renaming and resizing of the image files.

The target hardware platform is an Altera Cyclone II EP2C70F896C6 FPGA device in the Altera DE2-70 development and education board [22]. The EP2C70 FPGA is a low-cost, high-volume, high-performance FPGA for cost-sensitive applications. Table 1 lists the main features of the Altera EP2C70 FPGA.

Table 1 Altera EP2C70 FPGA features

Feature	Count
Logic elements (LE)	68,416
M4K RAM blocks (4 Kbits + 512 parity bits)	250
Embedded memory (Kbits)	1,125
18-bit×18-bit embedded multipliers	150
Phase-locked loops (PLL)	4
Maximum user I/O	622
Differential channels	262

All user hardware modules are developed in the Verilog hardware-description language using behavioural modeling on the Altera Quartus II Web Edition IDE 9.1 [23]. Waveform analysis and system debugging is carried out using SignalTap II logic analyzer, which is bundled with the IDE. Memory debugging, run-time loading, and memory reading is carried through the Quartus In-System Memory Content Editor. The overall system is designed using Altera SOPC Builder 9.1. Processor control code is develped using NIOS II IDE 9.1 SP1 using the C language.

3.4. *Neural network classifier*

The neural network used is a feed-forward, back-propagation network with three layers, input, hidden, and output. Only one hidden layer is used as it is sufficient for the pattern classification needs [24]. Fig. 2 shows a typical feed-forward neural network. The selected image samples are used for tuning the neural network classifier and testing the recognition accuracy. Default MATLAB neural network toolbox parameters are used to divide the samples into 70% for training, 15% for

validation, and 15% for testing. The indices of the test set are saved in order to retrieve the same test set for testing the hardware implementation. The output layer has 50 output nodes one for each of the 50 words. The number of hidden layer nodes is varied from 6 to 100 in steps of two and the recognition accuracy is recorded for each architecture step. The results are presented in Subsection 5.2.

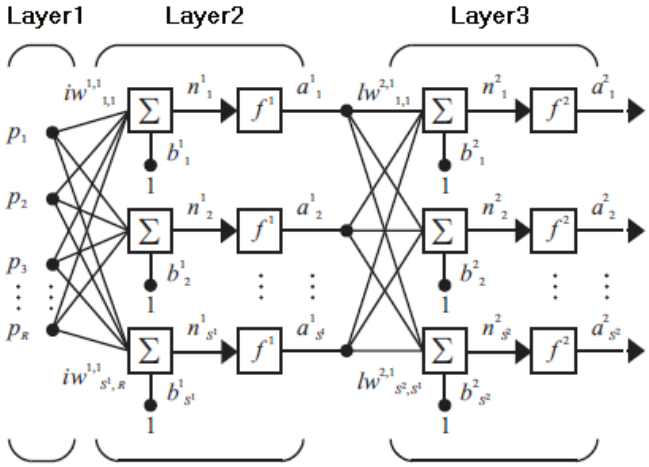

Fig. 2 Three-layer feed forward neural network [25]

The hyperbolic tangent sigmoid is used as the transfer function for the hidden layer, whereas the linear transfer function is used in the output layer. Moreover, three network training functions are investigated:

(1) Scaled conjugate gradient back-propagation (*trainscg*)
(2) Gradient descent with momentum and adaptive learning rate (*traingdx*)
(3) Gradient descent with adaptive learning rate back-propagation (*traingda*)

All training functions are used with their default settings. The simulations ran for a maximum of 50,000 epochs with a mean square error value set to 1.0×10^{-6}. Early stopping technique through the validation set is used to avoid over-fitting.

4. Feature Extraction Techniques

Four feature extraction techniques for possible development on programmable hardware devices are considered. Namely, 2D forward discrete cosine transform, density, gradient, and Hu moments. The main rationale behind choosing these commonly-used techniques is that they offer simplicity, good recognition accuracy, and room for parallelization. The latter advantage is needed to reduce recognition time. Additionally, we prefer these techniques as their computational cores have been used one way or another in other application domains on FPGAs. Some of the other feature extraction techniques which we excluded from our investigation are based on structural features such as loops, concavities, ascenders, descenders, and intersections to name a few. The detection of such perpetual features is often unreliable due to wide variation in handwriting styles [26]. Furthermore, some of these techniques need a baseline estimation step which adds to the solution complexity [27]. Aside from the complexity of their extraction, more than one structural feature needs to be combined in one feature vector to reach good recognition accuracy [1]. On the other hand, DCT, as an example, gives good recognition results even when used alone [24]. Also, the suggested algorithms are not restricted to holistic approaches but also are used in segmentation-based methods at the character level [1]. This makes our analysis a more general analysis. Each of the suggested four techniques is discussed in one of the following subsections.

4.1. *Two dimensional forward discrete cosine transform*

The *discrete cosine transform* (DCT) expresses a signal, image, or a function in terms of a sum of sinusoids with different frequencies and amplitudes. It is similar to the Fourier transform in that it transforms the signal or image from the time or

spatial domain to the frequency domain. When applied to two-dimensional (2D) signals such as images, DCT is referred to as 2D-DCT. However, we will refer to this sort of transform simply as DCT from now on. This transform helps separate the image into spectral sub-bands of differing importance. The DCT reduces redundancy and focuses the energy of the image in a very limited frequency set yielding a small number of features. Fig. 3 illustrates how DCT concentrates the energy in the sample image.

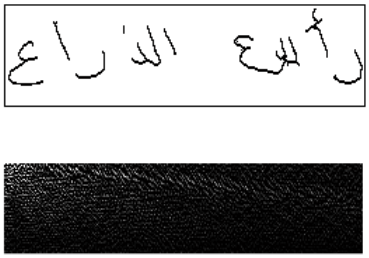

Fig. 3 The DCT image resulting from the top image

To elaborate further, for a binary image *A* with width *W* and height *H* (in pixels), the corresponding output DCT is stored in image *B* with the same width and height. *A(x, y)* represents the pixel value at row *x* and column *y*, where $A(x, y) \in \{0, 1\}$. *B(p, q)* is similar to *A(x, y)* but corresponds to the output image. For a *H*×*W* image, DCT can be found by:

$$B(p,q) = \alpha_p \alpha_q \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} A(x,y) \cos\left(\frac{\pi(2x+1)p}{2W}\right) \cos\left(\frac{\pi(2y+1)q}{2H}\right),$$

$$0 \le p \le W-1, \quad 0 \le q \le H-1,$$

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{W}}, & p = 0 \\ \sqrt{\frac{2}{W}}, & p \neq 0 \end{cases}, \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{H}}, & q = 0 \\ \sqrt{\frac{2}{H}}, & q \neq 0 \end{cases} \tag{1}$$

Efficient computation of DCT is possible through the row-column decomposition (RCD) technique [28]. In RCD, one dimensional DCT is applied to all rows in the first pass, and then it is reapplied on the columns in the second pass as follow.

$$B_y(p) = \alpha_p \sum_{x=0}^{W-1} A_y(x) \cos\left(\frac{\pi(2x+1)p}{2W}\right),$$

$$B(p,q) = \alpha_q \sum_{y=0}^{H-1} B_y(p) \cos\left(\frac{\pi(2y+1)q}{2H}\right) \tag{2}$$

Even though the DCT coefficients are computed for every pixel in the image with a total of 64×256 coefficients for our images, it has been shown that the first 20 features preserve 99% of the image energy. Selecting more features than 35 yields no improvement in the recognition accuracy [24].

Based on this, a feature set of 36 DCT coefficients has been chosen in our tests. This considerably reduces the number of inputs to the neural network, and thus reduces the number of required multiplications in each neuron. This allows for a feasible implementation. DCT features extraction is based on zigzag ordering, two orderings are usually used as shown in Fig. 4. The first ordering is adopted.

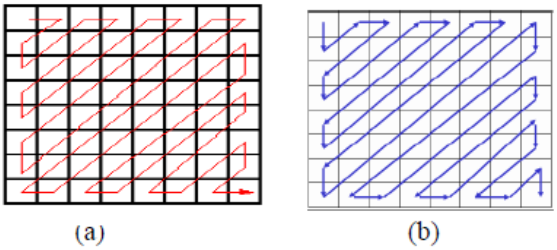

Fig. 4 The two zigzag DCT feature orderings [24]

4.2. *Density features*

The density technique is based on dividing the input word image into several $m \times n$ regions in which the number of black pixels is each region is found and used as a feature. We adopt window sizes that are multiples of 8 pixels to reduce hardware complexity in regards to memory alignment. Also coarse regions are adopted because it is highly unlikely that the Arabic letters would fall in the same small region for all writers. We adopted four region sizes: 32×32, 16×64, 32×64, and 32×128, resulting in 44 density features. Fig. 5 shows the density regions used and their order. Since some regions are made up of smaller regions, the density of the larger regions can be aggregated from smaller ones.

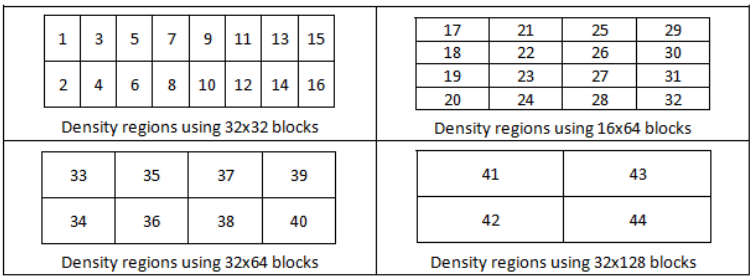

Fig. 5 The four density feature subsets when coarse regions are used on the 64×256 image

4.3. *Gradient features*

This feature extraction technique is adapted from the work in Ref. 29], yet it is slightly modified such that it is simpler to implement in hardware. This technique applies four scale-invariant 3×3 masks to the image; these masks are shown in Fig. 6. This would result in decomposing the input word image into four separate output images where each one only contains the pixels that match the corresponding mask.

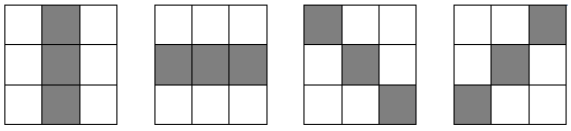

Fig. 6 The set of four gradient masks used to extract the gradient features

For each gradient mask, a 64×256 blank image is created. As the mask scans the input image from left to right and from top to bottom, any match between the input image and the mask triggers copying the mask to the created image. Fig. 7 shows the four extracted images of a sample image. The extracted four images are divided into eight 64×32 regions and the number of black pixels in each region is found. This provides a feature vector of length 4×8 =32 features.

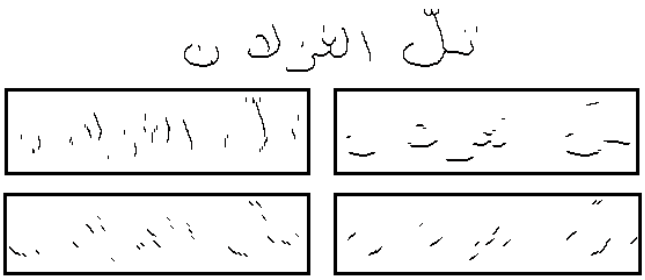

Fig. 7 Four extracted images after applying the gradient masks on the word Tal Al-Ghuzlan تل الغزلان

4.4. *Hu moments*

Moments are image mathematical descriptors which can compactly capture certain properties of the image. For example, the first through fourth-order geometric moments correspond to the area, variance, skewness, and kurtosis of an image. Thus providing a powerful statistical feature set for the identification and classification of images. Hu moments, which are seven in total, are in fact derived from the centralized moments up to order three [30]. They are invariant towards rotation, translation, and scaling. The centralized moments are themselves derived from the geometric moments. Hu moments are superior to centralized moments in that they are invariant to rotation and scaling whereas centralized moments are only invariant to translation. For the set of moment equations, the reader is referred to Ref. 30]. The input image is divided into eight 64×32 regions and the seven Hu moments are found for each region. This provides a feature vector of length 7×8 =56 features.

5. Analysis of the Classifier and Feature Extraction

In this section, we present the analysis carried out for the classifier and the four feature extraction techniques. Mainly, we compare the four techniques in terms of time, accuracy, hardware cost of suggested parallel implementations, and efficiency. Efficiency analysis is used to determine the best candidate for hardware implementation.

5.1. *Time analysis*

The suggested feature extraction techniques are coded, whenever possible, using MATLAB's built-in optimized functions to achieve high speed. Every technique is run 100 times on the entire 8,699 samples. MATLAB's built-in tic/toc functions are used for time measurements. The times spent in reading the image and performing feature extraction are recorded. The net time spent performing feature extraction is averaged over all iterations and the results are shown in Fig. 8. The DCT is the fastest technique.

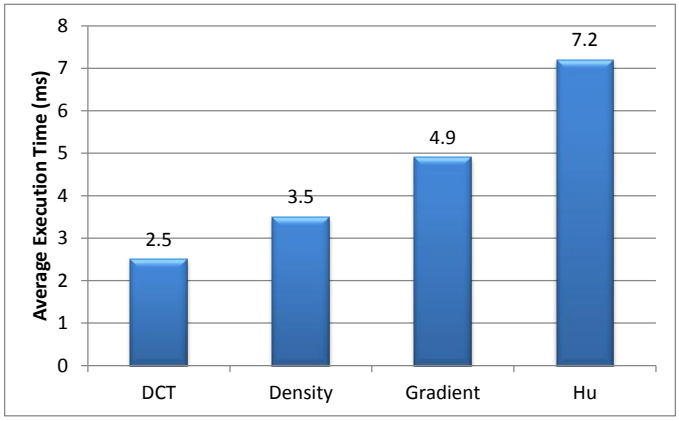

Fig. 8 Average execution time for the four feature extraction techniques

5.2. *Accuracy analysis*

As described in Section 3, the features extracted from each technique for all images are used as inputs to train, validate, and test the neural network. Fig. 9 shows the recognition accuracies of the four techniques as functions of the number of neurons in the hidden layer. The figure also shows the accuracies achieved using the three training functions.

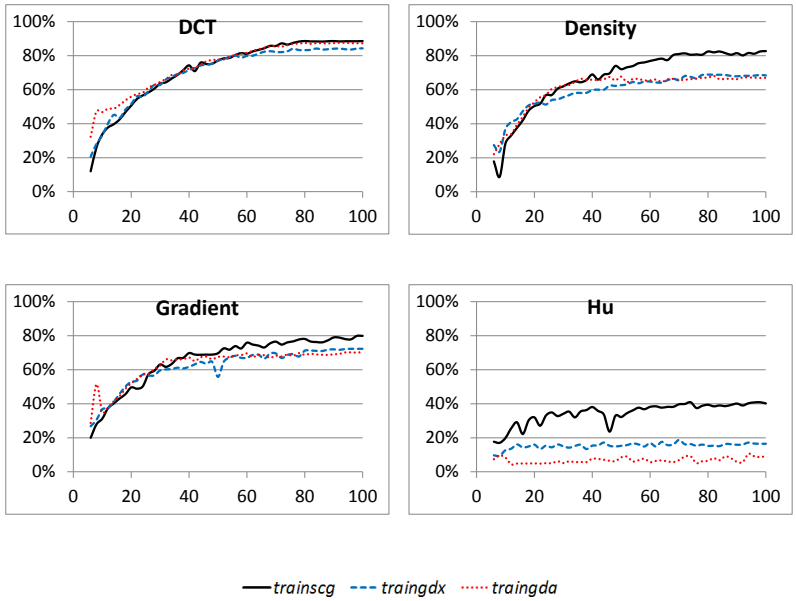

Fig. 9 Recognition accuracy *vs.* number of hidden layer neurons for the four techniques using three training functions

The best results are obtained when the *trainscg* training function is used. Fig. 9 also shows that all feature extraction techniques offer no significant accuracy improvement beyond 80 hidden layer nodes. Close examination of these results shows that DCT gives the highest accuracy, followed by density, followed by gradient, and Hu moments give significantly lower accuracy. To emphasize this observation, Fig. 10 shows the recognition accuracies of the four feature extraction techniques using *trainscg* and 80 hidden-layer neurons.

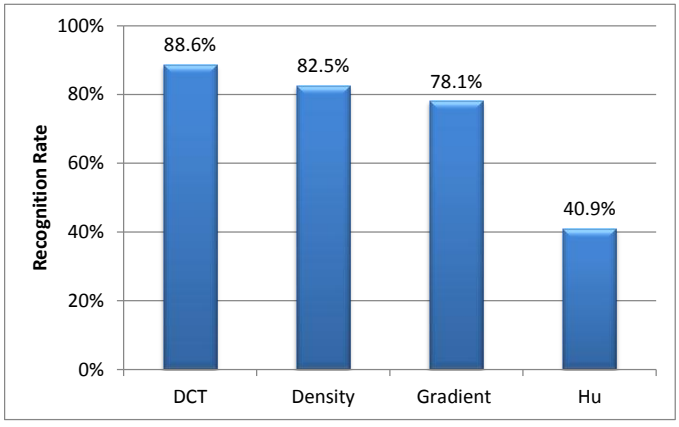

Fig. 10 Recognition accuracies of each of the four feature extraction techniques

5.3. *Cost analysis*

In this cost analysis of the four feature extraction techniques, we estimate the FPGA resources utilized. This estimation is restricted to the computational cores of these techniques in their direct implementation. The estimated implementations are assumed parallel hardware implementations to achieve speed advantages. This estimation is based on the hardware costs of some basic blocks on Cyclone II device that are shown in Table 2. These costs are the actual reported costs after building each block from Altera Megafunction library using the MegaWizard of the Quartus II IDE in the balanced mode. Megafunctions are ready-made, parameterized, pre-tested blocks of intellectual property that are optimized to make efficient use of the architecture of the device.

Table 2 The cost of some basic blocks on Cyclone II device

Function	IP core name	Cost in logic elements
16×16 2D forward DCT	CAST's DCT	1,240
16×16 multiply-accumulator unit	ALTMULT_ACCUM	400
16×16 integer multiplier	LPM_MULT	336
32×32 integer multiplier	LPM_MULT	624
M-bit accumulator	ALTACCUMULATE	M
M-bit integer adder/subtractor (No overflow)	LPM_ADD_SUB	M
32×32 floating-point multiplier	ALTFP_MUL	962

The estimations detailed below are based on fixed-point implementations. We avoided using floating-point numbers due to their high cost relative to their fixed-point counterparts, as demonstrated in Table 2. It is assumed that the memory interface cost for the four techniques is the same. Furthermore, we assume that each module's controller cost is far less than the computational core cost. Therefore, no cost estimation for the controller or module interfacing is carried out.

Table 3 summarizes the estimated number of logic elements needed for parallel implementations of the four techniques. The *logic element* (LE) is the smallest unit of logic in the Cyclone II architecture. The LE has a four-input look-up table and an output flip-flop. It is capable of implementing any four-input logic function or act as a full adder. In all the suggested implementations, the image is read only one time. To further speed up feature extraction, the module extracts the required feature values using parallel feature extraction units. The following paragraphs summarize these implementations and their estimated costs. For more details, see Ref. [31].

Table 3 Hardware cost estimates for the four feature extraction techniques

Technique	Logic elements (LE)	Normalized cost
DCT	40,832	11.0
Density	3,713	1.0
Gradient	6,016	1.6
Hu moments	13,572[a]	3.7

[a]Raw moments cost only

We assume that DCT is extracted efficiently in the row-column decomposition approach described in Subsection 4.1. For each of the 64 rows, row y is read and all its $B_y(p)$ values are computed for $p = 0 \ldots 7$. The $B_y(p)$ values are saved and used in the second stage to compute the $B(p,q)$ values. We further assume that all cosine values are hardcoded using enough LEs. The inputs to the first stage are the row pixels $A(x,y)$ and p. We use 16-bit fixed-point numbers for the cosine terms and generate each term of the $B_y(p)$ summation using 16 LEs. The input of each LE is one binary pixel and a 3-bit p field. In the second

stage, a 16×16 multiplier is used to multiply the $B_y(p)$ values by the cosine values. The multi-input addition in both stages is done using compressor trees [32]. The cost of this implementation in LEs is the sum of the costs: buffering one row (256×1), first-stage cosine terms (256×16), first-stage multi-input adder (4,080), first-stage α_p multiplier (336), buffering the $B_y(p)$ values (64×8×16), second-stage cosine values (64×16), second-stage cosine term multipliers (64×336), second-stage multi-input adder (1,008), and second stage constant α_q multiplier (336); a total of 40,832 LE.

The density features are extracted as described in Subsection 6.1. The cost of this implementation in LEs is the sum of the costs: buffering one region (32×32×1), summing the number of pixels of the 32-bit block rows (32×57), region rows adder (197), buffering the region sums and region upper half sums (16×(10+9)), Features 17–32 adders and subtractors ((8+8)×13), Features 33–40 adders (8×13), and Features 41–44 adders (4×13); a total of 3,713 LE.

The suggested implementation for gradient features extraction loads one image row at a time in a three-row buffer where a new row replaces the oldest row as the scan progresses. The four gradient masks are applied on the row in parallel to generate four gradient rows. Since each gradient pixel is a function of nine pixels, three logic elements are needed for extracting one gradient pixel. The pixels in each of the eight 32-pixel segments of the gradient rows are summed and accumulated for the 64 rows of each region. The cost of this implementation in LEs is the sum of the costs: buffering three rows (3×256×1), gradient extraction masks (4×256×3), summing 32-pixel segments (4×8×57), accumulating segment sums (4×8×11)); a total of 6,016 LE.

In the proposed Hu moment features extraction, one 64×32 region is loaded at a time and its ten raw moments $M_{ij} = \sum_y \sum_x y^j x^i A(x,y),\ \forall i+j \leq 3$ are computed in parallel. This process is repeated for the 8 regions to extract 8×10 raw moments. The raw moments are computed similar to the DCT implementation. Partial sums are found for each 32-pixel row and are accumulated for the 64 rows. The $y^j x^i A(x,y)$ terms are hardcoded using LE cells. The inputs to each cell is one pixel $A(x,y)$ and the current row y. We use w-bit integers for these terms and generate each term using w×2 LEs. The width w depends on the largest term in M_{ij} and increases as the moment order $(i+j)$ increases. The cost of computing each raw moment is the sum of the costs: term cells (32×w×2), 32-term adder (31×w+42), and accumulator (w+10). We have estimated that the cost of buffering one region (64×32) and extracting the 10 raw moments is 13,572 LE. The Hu moments are derived from the normalized central moments and the latter are derived from the raw moments. We don't bother to estimate the cost of this derivation because Hu moments accuracy is low and the cost of extracting the raw moments is already greater than the cost of the density or gradient features. Moreover, Hu moments' advantage over the raw moments is their invariance to translation, scaling, and rotation which is not useful for the used fixed-size images.

5.4. *Efficiency analysis*

Efficiency is derived here by dividing the recognition accuracy of the proposed technique over the normalized cost. Fig. 11 shows the efficiency of the four feature extraction techniques normalized to the cost of the density feature extraction technique. It is clear that the density is superior to the other techniques. Therefore our FPGA implementation is based on the density feature extraction technique.

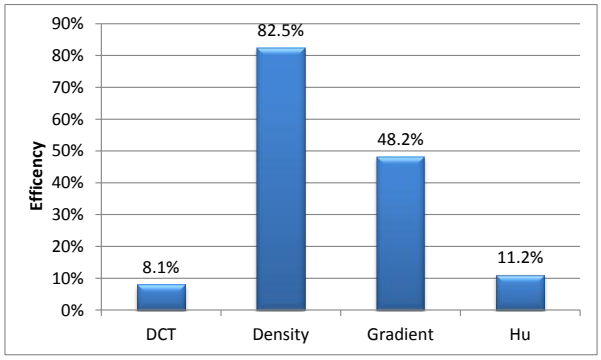

Fig. 11 Accuracy/cost efficiency

6. FPGA Implementation

A software hardware co-design approach is adopted in our implementation to simplify the design process. Altera's NIOS II/f processor core is used. It is responsible for managing image transfer, reading the neural network output, and calculating the overall recognition rate of the test set. It also acts as the main controller of the system; it monitors progress of each recognition stage and initiates subsequent stages. Fig. 12 illustrates the overall system design.

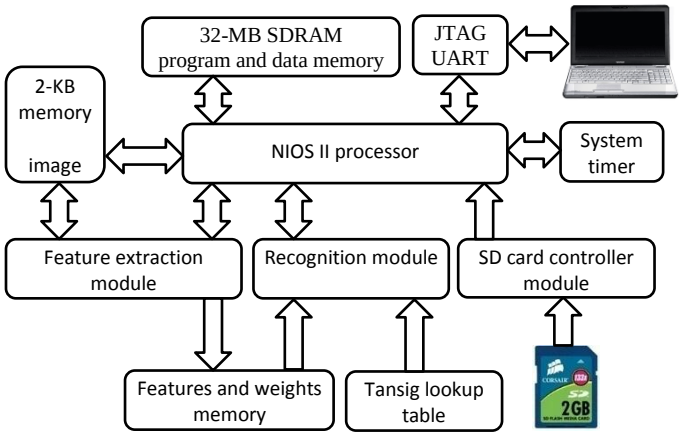

Fig. 12 System block diagram

In this design, the test image set is stored on an external secure digital (SD) card. The JTAG UART is used to configure the FPGA, download the compiled C user code and libraries of the NIOS II/f processor to the SDRAM, as well as interface between the hardware and the IDE terminal for monitoring and debugging purposes. The system starts by searching for and mounting the SD card, reading the FAT tables, and loading the first image to the memory image storage. Since the images are binary, a total of 2 KB of data is transferred for the fixed image sizes of 64×256 bits. The processor signals the feature extraction module to start by setting the start bit inside the module's control register. This register is internally checked every clock cycle and is reset by the module when it finishes feature extraction of the image. The resulting feature vector is stored in the features and weights memory. When the start bit of the feature extraction module is polled as zero, the processor starts the recognition module. The final output vector of the recognition module is internally stored in the module and is read by the processor. The IFN/ENIT database documentation requires that the zip code of the Tunisian town/village name is reported rather than the name itself [21]. So the neural network output vector is used to look-up the zip code. The retrieved zip code is compared against the actual zip code of the image to find the recognition accuracy. To simplify this process, image file names are named such that the four initial characters of the file name equal the zip code.

6.1. *Feature extraction module*

The image is divided into 16 regions each is 32×32 bits. One region is loaded at a time to extract the density features 1–16 shown in Fig. 5. The remaining features 17–44 are derived from features 1–16. The starting address of the regions is internally stored inside the feature extraction module. The module reads the 32 rows of each region through its memory interface and buffers them. The initial region address is set, memory transfer signals are asserted, and the data is transferred and buffered. The address is adjusted to point to the next row within a region by increments of 32 bytes.

After the end of one region transfer, the pixels of each row are summed up in parallel using 32 multi-input adders as shown in Fig. 13. Then the 32 partial sums are aggregated using another multi-input addition stage. This stage generates two results for each Region *i*: the sum of Rows 0–15 (*Upper[i]*) and the sum of Rows 0–31 (*Feature[i]*). These two results are buffered. The next region address is then loaded and the same computation is repeated for the remaining 15 regions.

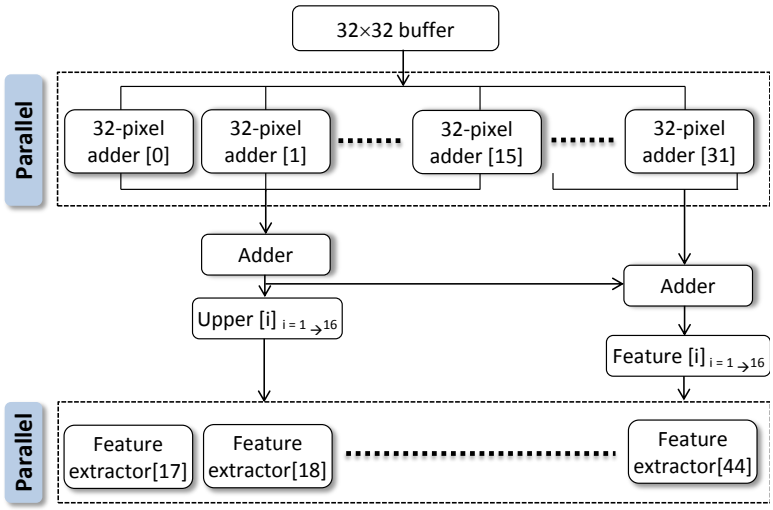

Fig. 13 Feature extraction module

Features 17–44 are computed in parallel from the buffered regions results. Each of the Features 33–40 is computed by adding the corresponding two horizontally-adjacent features of the range 1–16. Each of the Features 40–44 is computed by adding the corresponding two horizontally-adjacent features of the range 33–40. The odd-numbered features in the range 17–32 are found by summing two horizontally-adjacent *Upper* results; whereas the even-numbered features are found by subtracting the odd-numbered feature from the corresponding feature in the range 33–40. Finally, all features are saved in the features and weights memory.

6.2. *Neural network recognition module*

Our analysis concluded that a neural network with 44/80/50 topology gives superior accuracy. Due to the limited resources of the FPGA, a sequential network is built where a single neuron function assumes the role of all the network's neurons. This is typical neural network implementation for large networks on FPGAs. However, all arithmetic computations within this neuron function are performed in parallel. The multiplications of the neuron inputs and their weights in addition to the multi-input addition are performed in parallel as shown in Fig. 14. The neuron function performs only 44 parallel multiplications for the hidden layer neurons and performs 80 parallel multiplications for the output layer neurons. The embedded FPGA multipliers are utilized for input normalization and weight multiplication. The input normalization is only performed on the inputs of the hidden layer.

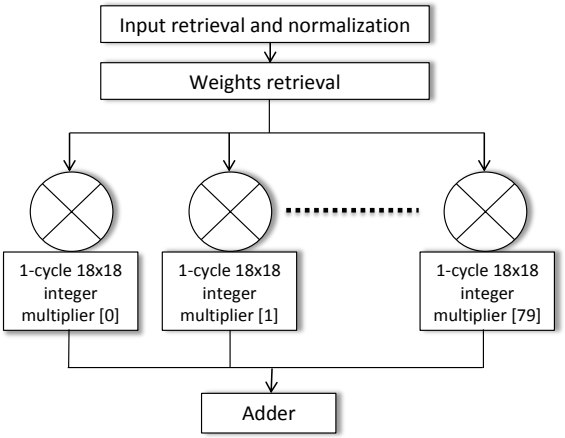

Fig. 14 The implementation of the neuron function

The neural network feature inputs are normalized to [-1 1] as they are retrieved from the memory. The signed fixed-point format is used to allow using the embedded 18×18 integer multipliers. We estimated that implementing the neuron function without using the embedded multipliers would consume 73% of the target FPGA logic elements, which is clearly resource inefficient approach.

Initially, the MATLAB *mapminmax* normalization function was considered to normalize the integer feature values:

$$X_{NRM} = (Y_{MAX} - Y_{MIN}) \times \frac{X_{IN} - X_{MIN}}{X_{MAX}} - 1, \text{ where } Y_{MAX} = 1, \; Y_{MIN} = -1 \qquad (3)$$

As $X_{MIN} = 0$ for all features, this function reduces to:

$$X_{NRM} = X_{IN} \times \frac{2}{X_{MAX}} - 1 \qquad (4)$$

However, this function requires readjusting to get fixed-point format. A similar function that directly gives fixed-point values is:

$$X_{NRM} = X_{IN} \times \left(\frac{2^{N+1}}{X_{MAX}}\right) - 2^N \qquad (5)$$

where N is the number of the fixed-point fraction digits. As our implementation uses 9-bit fractions, the above equation becomes:

$$X_{NRM} = X_{IN} \times \left(\frac{1{,}024}{X_{MAX}}\right) - 512 \qquad (6)$$

However, to reduce the rounding error in the $(1{,}024/X_{MAX})$ scaling factor, the equation is adjusted to:

$$X_{NRM} = \left(X_{IN} \times \text{Round}\left(\frac{1{,}024}{X_{MAX}} \times 2^M\right)\right) \gg M - 512 \qquad (7)$$

In this equation, the integer feature X_{IN} is multiplied by a scaling factor, shifted right M bits to offset the 2^M factor, and minuend by 512. For 18-bit multiplication operands and taking into consideration the minimum X_{MAX} of all features, we found that the largest M value that can be used without multiplication overflow is 14. So the final normalization equation used is:

$$X_{NRM} = \left(X_{IN} \times \text{Round}\left(\frac{1{,}024}{X_{MAX}} \times 16{,}384\right)\right) \gg 14 - 512 \qquad (8)$$

All the scaling factors are pre-computed in advance as 18-bit numbers and stored internally inside the module. The shifting operation is not needed since we extract the correct bits from the 36-bit multiplier output.

The hidden layer neurons use the hyperbolic tangent sigmoid transfer *tansig* function. We implement this function using a look-up table to save logic resources. The symmetry of the *tansig* function allows reducing the table size by half. Moreover, we observed that the values from 0 to 3.875 nearly capture all the nonlinearity of the function while values larger than 3.875 can be mapped to 1. The table has 16-bit signed fixed-point values in increments of 1/512, resulting in a table size of 3,970 bytes.

Four variations of the recognition module are implemented and tested. Each has the ability to transfer the feature and weight values in different chunk sizes. The Avalon memory-mapped interface width was varied from 32 (Default) to 64, 512, and 1,024 bits. The features and weights memory bus widths are adjusted to match these sizes accordingly. Additionally, the number of normalization multipliers is increased accordingly. The first 3 bus widths were actually implemented. However, the 1,024-bit bus width was not implemented because we ran out of embedded multipliers. However, the performance of this variation is extrapolated from the three other implementations.

7. Results and Discussion

In this section, we present the results of the FPGA-based handwritten Arabic word OCR implementation. We elaborate on system speed up gains, resource utilization, and recognition accuracy.

7.1. *Speed up gains*

Hardware system timing is measured using the SignalTap II logic analyzer IP core. While the feature extraction module runs at 100-MHz clock, the Neural Network runs at 45 MHz. The time to extract the features and recognize one sample word is constant over all word images. Fig. 15 shows this time for the software MATLAB implementation and the four FPGA implementations in microseconds. This figure indicates that the hardware implementations are 20–200 times faster than the software implementation.

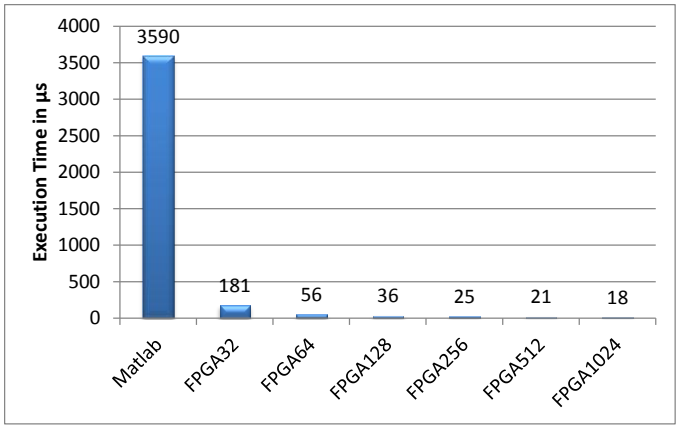

Fig. 15 Total system execution time (feature extraction, normalization, and recognition)

7.2. *Hardware resource costs*

Fig. 16 shows the implementation cost in terms of used logic elements out of the 68,416 available logic elements. The FPGA32 implementation consumes 16,971 logic elements, around 25% of the available logic elements. The recognition, feature extraction, and NIOS II processor take 21% of the logic elements while the other supporting and interface modules take 4%.

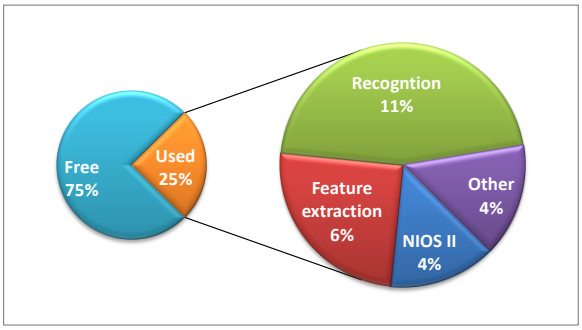

Fig. 16 Implementation cost (logic resources)

Fig. 17 shows the implementation cost in terms of used memory resources out of the 1,125 Kbits available embedded memory. The features and weights memory, *tansig* look up table, processor cache, and image buffer consume 260 Kbits, around 23% of the available embedded memory.

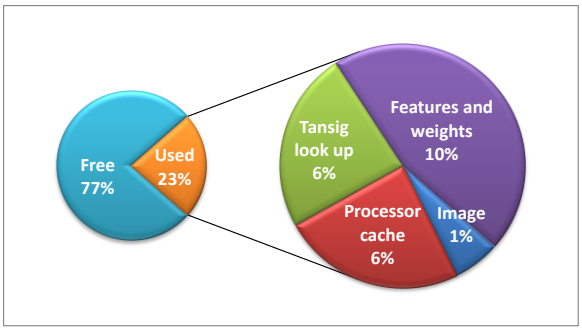

Fig. 17 Implementation cost (on-chip memory resources)

7.3. *Recognition accuracy*

In Subsection 5.2, we have shown that the density feature extraction technique has 82.5% recognition accuracy. This result is obtained using MATLAB implementation with floating-point numbers on a test set of 1,304 word images (15% of the selected samples). However, our FPGA implementation is based on 16-bit signed fixed-point format that has 9-bit fraction field. This number format is less precise than the floating-point format used in MATLAB. Moreover, we have approximations in the normalization process. Fig. 18 shows the recognition accuracies of the software and FPGA implementations on the 1,304-word test set. To analyze the effects of these approximations on accuracy, we made two different implementations for the neural network input normalization stage. The first is implemented in C code and executed by the NIOS II/f processor. This software normalization implements Eq. (4) similar to what is used in MATLAB. The second hardware implementation implements Eq. (8) as described in Subsection 6.2. The results shown in this figure suggests that the FPGA hardware implementation is 2.8% less accurate than the MATLAB implementation. About 1.5% of this drop is due to the approximation in the normalization stage and 1.3% is due to using fixed-point numbers.

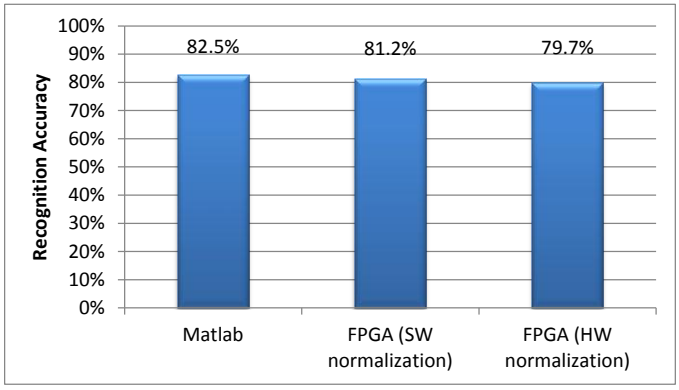

Fig. 18 Recognition accuracies for MATLAB and FPGA implementations

8. Conclusions and Future Work

In this paper, we investigated the feasibility of using programmable hardware devices, namely FPGAs, to recognize handwritten Arabic words. We described a parallel FPGA implementation that offers high speed and good accuracy compared with software implementations. This implementation consumes only about one quarter of the target FPGA resources. In this investigation, we concentrated on four feature extraction techniques that are commonly used in recognizing handwritten Arabic words: DCT, density, gradient, and Hu moments. We analyzed the accuracy of these four techniques and conducted sensitivity analysis to determine the neural network topology used in the classification stage. This analysis is based on recognizing samples of a 50-word subset of the IFN/ENIT database. Hardware cost estimates in terms of logic resources

based on parallel implementations of these techniques were found. And we adopted the density technique because it has the highest efficiency in terms of recognition accuracy and hardware cost.

The FPGA implementation employs an implementation of the soft processor NIOS II/f as the main system controller. It is also responsible of image transfer to the on-chip memory from the storage device. The feature extraction and the recognition stages are implemented as separate modules each with its own interface to memory for faster access and module independence. Multiple design alternatives and arithmetic approximations were investigated and analyzed.

The main contribution of this research is that it is the first of its kind to implement an FPGA recognition system of handwritten Arabic words using the holistic approach. We demonstrated speedup gains of 20–200 over the same algorithms when executed in software on a Core 2 Duo computer running at 2.2 GHz. This is encouraging as high speedup factors are expected should such a system is implemented in assistive technology or handheld devices.

This work can be extended by analyzing other feature extraction techniques as well as using combined features from various feature extraction techniques to achieve higher classification accuracy. Other classifier types could also be investigated such as hidden Markov models and recurrent neural networks. This research used preprocessed images as the input to the system; several preprocessing techniques could be investigated, such as baseline estimation, thinning, and scaling. We only investigated the simple holistic approach. The more general segmentation-based model would serve as logical upgrade to the system for the use with unconstrained vocabulary. Moreover, one could make use of the free remaining FPGA space by adding extra modules. One such addition is a speech synthesis core. This could serve as a prototype for future assistive technology devices for the Arabic language.

References

1. Khorsheed, M. (2002). Off-line Arabic character recognition — a review. *Pattern Anal. Appl. 5*(1), 31–45.
2. Lewis, M. P. (2009). *Ethnologue: Languages of the world*. Dallas, TX: SIL International.
3. Lorigo, L. M., & Govindaraju, V. (2006). Offline Arabic handwriting recognition: a survey. *IEEE Trans. Pattern Anal. Mach. Intell. 28*(5), 712–724.
4. Märgner, V., & El-Abed, H. (2011). ICDAR 2011 Arabic handwriting recognition competition. In: Proc. 11th Int'l Conf. Document Analysis and Recognition, Beijing, pp. 1444–1448.
5. Impedovo, S., Ottaviano, L., & Occhinegro, S. (1991). Optical character recognition — a survey. *Int'l J. Pattern Recognit. Artif. Intell. 5*(1), 1–24.
6. Märgner V., & El-Abed, H. (2009). ICDAR 2009 Arabic handwriting recognition competition. In: Proc. 10th Int'l Conf. Document Analysis and Recognition, Barcelona, pp. 1383–1387.
7. Abdullah, S., & Marsidi, S. (2008). Digitization of Arabic materials in IIUM library: Challenges and problems. In: Proc. World Congress of Muslim Librarian & Information Scientists, Malaysia, pp. 25–27.
8. Rice, S. (1996). *Measuring the Accuracy of Page-Reading Systems*. PhD Dissertation, University of Nevada, Las Vegas.
9. Pellerin, D., & Thibault, S. (2005). *Practical FPGA Programing in C*. Upper Saddle River, NJ: Prentice Hall Press.
10. Elbeheri, G., Mahfoudhi, A., & Everat, J. (2009). Perspectives from the Arab World. In: Proc. Int'l Dyslexia Association Perspectives on Language and Literacy, pp. 9–12.
11. Abandah, G., & Malas, T. (2010). Feature selection for recognizing handwritten Arabic letters. *Dirasat Eng. Sci. J. 37*(2), 242–256.
12. Beg, A. (2008). An efficient realization of an OCR system using HDL. In: Proc. 2008 World Congress in Computer Science, Computer Engineering, and Applied Computing, Las Vegas.
13. Al-Marakeby, A., Kimura, F., Zaki, M., & Rashid, A. (2013). Design of an embedded Arabic optical character recognition. *J. Signal Processing Systems. 70*(3), 249–258.
14. Toosizadeh, N., & Eshgi, M. (2005). Design and implementation of a new Persian digits OCR algorithm on FPGA chips. In: Proc. 13th Conf. European Signal Processing (EUSIPCO2005), Antalya, Turkey.
15. Moradi, M., Pourmina, M., & Razzazi, F. (2010). A new method of FPGA implementation of Farsi handwritten digit recognition. *European J. of Scientific Research. 39*(3), 309–315.

16. Razak, Z., Zulkiflee, K., Salleh, R., Yaacob, M., & Tamil, M. (2007). A real-time line segmentation algorithm for an offline overlapped handwritten Jawi character recognition chip. *Malaysian J. Comput. Sci. 20*(2), 171–182.
17. Razak, Z., Zulkiflee, K., Salleh, R., Noor, N. M., & Yaacob, M. (2009). Off-line handwritten Jawi character segmentation using histogram normalization and sliding window approach for hardware implementation. *Malaysian J. Comput. Sci. 22*(1), 34–43.
18. Ahmadi, A., Abedin, M. D., Kamimura, K., Shirakawa, Y., Mattausch, H. J., & Koide, T. (2004). Associative memory based hardware design for an OCR system and prototyping with FPGA. In: Proc. Third COE Int'l Workshop on Nanoelectronics for Tera-bit Information Processing, Hiroshima, Japan, pp. 40–41.
19. Dang'ana, M. (2008). A simplified OCR system hardware implementation in FPGA (2008). McGill University.
20. Srinivasan, S., Zhao, L., Sun, L., Fang, Z., Li, P., Wang, T., Iyer, R., Illikkal, R., & Li, D. (2010). Performance characterization and acceleration of optical character recognition on handheld platforms. In: Proc. IEEE Int'l Symp. Workload Characterization (IISWC), Atlanta, pp. 2–4.
21. Pechwitz, M., Maddouri, S. S., Märgner, V., Ellouze, N., & Amiri, H. (2002). IfN/ENIT – database of handwritten Arabic words. In: Proc. Colloque Int'l Francophone sur l'Ecrit et le Document (CIFED), pp. 127–136.
22. Altera Website. DE2-70 Development and Education Board. http://www.altera.com/education/univ/materials/boards/de2-70/unv-de2-70-board.html. Accessed 4 Mar 2013.
23. Altera Website. Quartus II Web Edition Software. http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html. Accessed 4 Mar 2013.
24. Al-Khateeb, J., Ren, H., Jiang, J., Jianmin, I., Stan, S., & El-Abed, H. (2008). Word-based handwritten Arabic scripts recognition using DCT features and neural network classifier. In: Proc. 5th Int'l Multi-Conf. Systems, Signals and Devices, Amman, pp. 1–5.
25. Beale, M., Hagan, M., & Demuth, H. (2010). *Neural Network Toolbox User's Guide*. Natick, MA: MathWorks.
26. Madhvanath, S., & Govindaraju, V. (2001). The role of holistic paradigms in handwritten word recognition. *IEEE Trans. Pattern Anal. Mach. Intell. 23*(2), 149–164.
27. Noaparast, K., & Broumandnia, A. (2009). Persian handwritten word recognition using Zernike and Fourier–Mellin moments. In: Proc. 5th Int'l Conf. Sciences of Electronic Technologies of Information and Telecommunications, Tunisia.
28. Cho, N. I., & Lee, S. U. (1991). Fast algorithm and implementation of 2-D discrete cosine transform. *IEEE Trans. Circuits Syst. 38*(3), 297–305.
29. Ebrahimpour, R., Davoudi Vahid, R., & Nezhad, B. M. (2011). Decision templates with gradient based features for Farsi handwritten word recognition. *Int'l J. Hybrid Information Technology. 4*(1), 1–12.
30. Hu, M. (1962). Visual pattern recognition by moment invariants. *IRE Trans. Information Theory. 8*(2), 179–187.
31. Suyyagh, A. (2011). *Investigating Synthesis of Efficient Handwritten Arabic Word Recognition Engines on FPGAs*. MSc Thesis, The University of Jordan.
32. Parandeh-Afshar, H., Neogy, A., Brisk, P., & Ienne, P. (2011). Compressor tree synthesis on commercial high-performance FPGAs. *ACM Trans. Reconfigurable Technol. Syst. 4*(4), 39.

Ashraf Suyyagh received his BSc and MSc degrees in Computer Engineering from the University of Jordan in 2007 and 2011, respectively. He is currently a PhD candidate in Electrical Engineering at McGill University. His research interests are in the areas of cyber-physical, embedded and multi-sensory systems as well as reconfigurable computing.

Gheith Abandah has received MSE and PhD in Computer Science and Eng. from the University of Michigan in 1995 and 1998. He has been responsible for several funded projects in the areas of Arabic text recognition and electronic voting. He has more than 15 years of industrial experience in localization, military electronics, product and project development and deployment. He is an associate professor with the University of Jordan and chair of the Computer Eng. Dept. He was the general chair for the IEEE AEECT 2011 conference and chair of IEEE–Jordan Section.