14th International Conference
MIXED DESIGN
MIXDES 2007
Ciechocinek, POLAND
21 – 23 June 2007

HARDWARE-SOFTWARE CODESIGN OF A FINGERPRINT ALIGNMENT PROCESSOR

M. FONS, F. FONS, E. CANTÓ
UNIVERSITAT ROVIRA I VIRGILI, SPAIN

KEYWORDS: Hardware-software codesign, Embedded systems, Fingerprints

ABSTRACT: Within the biometrics field, the development of an automatic personal authentication system is nowadays an open research problem. Most of the difficulties rely on the complexity and the computational power needed to implement an algorithm reliable enough to guarantee the validity of the recognition system even when only low-quality biometric information is available from the users. Apart from the inherent complexity of the biometric algorithm, there exist many additional requirements for the application: high-security level, real-time characteristics and low-cost. All these factors originate a technical challenge. A compromise between the final application performance and the amount of resources needed to implement the system exists. Following this direction, the design of a fingerprint alignment processor developed by means of hardware-software codesign techniques is presented in this paper. The performance achieved with this processor is compared against the performance reached with an only software-oriented solution. The acceleration factor achieved by the hardware proves the feasibility of real-time applications, which is not guaranteed when developing the same algorithm under purely-software platforms.

MOTIVATION

The purpose of an AFAS (automated fingerprint authentication system) is to accurately verify the identity of an individual by means of his/her genuine fingerprint characteristics. By comparing a query fingerprint with an originally enrolled fingerprint acting as template, the system attempts to certify that the user (query print) is either who claims to be (template print) or, on the contrary, is an impostor. Four are the main stages involved in any fingerprint-based personal authentication system:

- Fingerprint acquisition. By means of electronic sensors a digital image of the user's fingerprint is obtained.
- Feature extraction. Those relevant and distinctive characteristics available in the finger impression are identified and recorded.
- Fingerprint alignment. Given a template and a query fingerprints, their extracted features are used to align both images.
- Feature matching. Once the alignment is done, the comparison of the overlapped area between both prints permits to determine the similarity level of both impressions, and finally deduce if they come or not from the same individual.

Two are the main characteristics used to align and match fingerprints:

- the field orientation map of the fingerprint, determined by the global ridge-valley orientation flow present in the finger epidermis;
- and the spatial distribution of those local characteristic points, called minutiae and mainly based on the ridge endings and the ridge bifurcations of the skin.

Given two fingerprint impressions (template and query) to be compared, it is possible to extract from them those characteristic features, as shown in fig. 1.

An efficient alignment process for the template and query images is needed before determining the similarity score between both prints. The performance of an AFAS heavily relies on the fingerprint image quality. In poor quality images, the minutiae information may not be reliable due to missing or spurious minutiae points obtained in the feature extraction stage. Therefore, the alignment based on minutiae points may be erroneous. However, the field orientation map presents a higher robustness level against noisy and low-quality fingerprints [1]. For this reason, in this paper a technique that uses the extracted field orientation map to align fingerprint images is proposed.

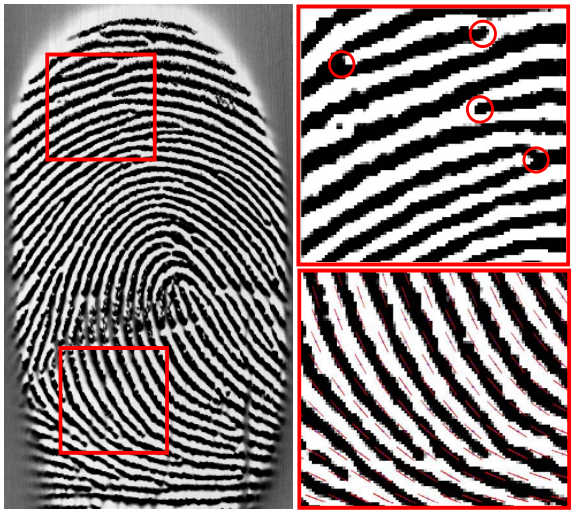

Fig. 1. Fingerprint characteristics

In the following sections the design of the fingerprint alignment processor is presented. First, the alignment process is described in detail. Next, the design of the application is exposed: the hardware-software partitioning, the hardware-software interface and the synthesis of the hardware accelerator are covered. Finally, the performance reached by the application is compared against the performance obtained when the same application is developed under a purely-software platform. The paper ends with some concluding remarks.

ALIGNMENT

The alignment technique exposed in this work is based on the fingerprint field orientation map. The template and query fingerprints are tessellated into square cells of N x N pixels, and for each cell the local field orientation is computed as suggested in [2]. The resultant field orientation is codified in the range [0°-180°).

Once the field orientation matrices for the template and query fingerprints are built, the next step consists in finding the best alignment between both prints. Given a template of t_x x t_y cells and a query fingerprint of q_x x q_y cells, both orientation matrices are correlated. The alignment process tries to find an overlapped region between both images that accomplishes the requirement (3) while minimizing the objective function shown in (6).

$$Template\ F.O.\ Matrix \equiv T(u,v) \quad \forall u \in 1...t_x, v \in 1...t_y$$
$$Query\ F.O.\ Matrix \equiv Q(u,v) \quad \forall u \in 1...q_x, v \in 1...q_y$$
$$Overlapped\ F.O.\ Matrix \equiv O(i,j) = T(i,j) \cap Q(i,j) \tag{1}$$

$$\forall (i,j) \in O, \Delta\phi(i,j) = min\left(\left\|\phi_T(i,j) - \phi_Q(i,j)\right\|, 180 - \left\|\phi_T(i,j) - \phi_Q(i,j)\right\|\right) \tag{2}$$

$$\sum_{(i,j)}^{O} 1 \geq Threshold_{OV} \tag{3}$$

$$\overline{\Delta\phi_O} = \frac{\sum_{(i,j)}^{O} \Delta\phi(i,j)}{\sum_{(i,j)}^{O} 1} \tag{4}$$

$$\sigma_O = \sqrt{\frac{\sum_{(i,j)}^{O} \left(\Delta\phi(i,j) - \overline{\Delta\phi_O}\right)^2}{\sum_{(i,j)}^{O} 1}} \tag{5}$$

$$f_O = \overline{\Delta\phi_O} + \sigma_O \tag{6}$$

The template field orientation matrix is moved over the query field orientation matrix, totally or partially as depicted in fig. 2. For each relative position, the number of overlapped cells and the average and standard deviation of the difference between both field orientations in the overlapped regions are computed.

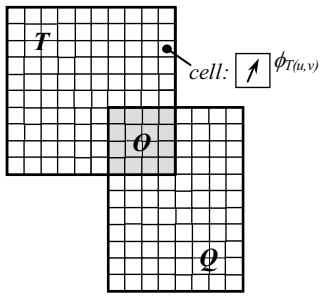

Fig. 2. Alignment analysis

In order for the algorithm to be able to properly align fingerprints even when the original template and query present different orientations, it is possible to rotate the query field orientation matrix θ degrees prior to computing the alignment between both matrices. The field orientation at cell (i',j') on the rotated query matrix can be found by rotating an angle θ the field orientation of its corresponding cell (i,j) in the original query matrix:

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} \cos\theta & -sen\,\theta \\ sen\,\theta & \cos\theta \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \tag{7}$$

$$\phi'_{(i',j')} = \phi_{(i,j)} + \theta \tag{8}$$

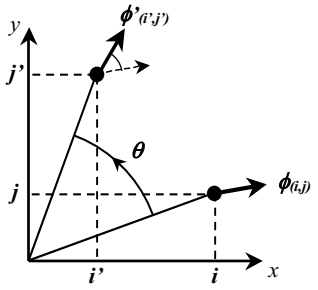

Fig. 3. Rotation process of the field orientation matrix

$Threshold_{OV}$ is a configurable parameter to adjust the minimum amount of overlapped cells required to assure the good alignment. Furthermore, equation (2) has been modified as in (9) in order for the alignment algorithm to tolerate those elastic deformations inherent to fingerprints [3]. Each template cell is compared against not only its corresponding aligned query cell, but against a query neighbourhood kernel of size K' x K' (K'= 2K + 1) centred on its aligned query cell.

This improves the robustness of the alignment algorithm against noisy images, fingerprint deformations and low-quality prints. Therefore, the alignment algorithm finds the best alignment between the template and the query field orientation matrices taking into account any translation, rotation and elastic deformation of the acquired fingerprints. As it can be deduced, the amount of relative alignments to be studied increases exponentially with the size of the field orientation matrices as well as with the number of rotation angles to be considered in the alignment process.

$$\forall (i,j) \in O, \Delta\phi(i,j) = \min_{\substack{K_i = -K...0...+K \\ K_j = -K...0...+K}} \left(\left\|\phi_T(i,j) - \phi_Q(i+K_i, j+K_j)\right\|, 180 - \left\|\phi_T(i,j) - \phi_Q(i+K_i, j+K_j)\right\|\right) \tag{9}$$

So it becomes a brute force alignment algorithm where all relative positions template-query accomplishing certain requirements are correlated. The flow diagram used to find the best alignment between the template and the query prints is shown in fig. 4.

```
Inputs: template & query F. O. matrices
Initialization of f_objective to Max. Value
for θ = …-30°,-20°,-10°,0°,+10°,+20°,+30°…
{
  Compute rotated query F. O. matrix
  for x = …-3,-2,-1,0,+1,+2,+3…
  {
    for y = …-3,-2,-1,0,+1,+2,+3…
    {
      Translate template F. O. matrix over
      rotated query F. O. matrix
      if (Overlap cells(T,Q(x,y,θ)) > Thd_OV)
      {
        Compute f(T,Q(x,y,θ))
        if (f(T,Q(x,y,θ)) < f_objective)
        {
          f_objective = f(T,Q(x,y,θ))
          X_objective = x
          Y_objective = y
          θ_objective = θ
        }
      }
    }
  }
}
if (f_objective < Max. Value)
{
  [Alignment parameters]
  Return(f_objective,X_objective,Y_objective,θ_objective)
}
else
{
  [No alignment found]
  Return (-1)
}
```

Fig. 4. Alignment flow

At the end of the processing, either the best alignment is found or a negative response is given. In case of positive alignment, the outputs of the algorithm are the relative rotation (θ) and translation offsets (X,Y) to be applied to the original prints in order to align them. These outputs, as well as the objective function and the number of overlapped cells would be used as reliable inputs for the next matching stage.

HW-SW PARTITIONING

A benchmark of the automatic personal recognition systems available today points that most of them rely on purely software solutions running under personal computers or microprocessor platforms [4]. However, this architecture can result inappropriate when the complexity of the application increases and real-time characteristics are required for the functionality.

The advances made in VLSI offer the hardware-software codesign methodology as a challenging alternative solution. An embedded system featuring a software block (MCU, DSP) and a hardware block (FPGA, ASIC) highlights much more advantages than traditional solutions. The introduction of the field programmable logic devices into the system allows the partitioning of any application into hardware and software tasks: those complex and time-consuming computational tasks can be accelerated by synthesizing them into the hardware core blocks (FPGA), while the rest of less computationally expensive tasks can be kept under the control of the software block (MCU). Therefore, a solution based on hardware-software codesign is proposed in this work. In the suggested architecture the MCU becomes the master scheduler of the alignment process, responsible for controlling and monitoring the application flow, while the FPGA works as a slave block where specific coprocessors can be synthesized in order to accelerate those critical tasks. The final purpose of this is to reduce the execution time of the application in order to reach real-time performance. Therefore, to do this, first of all it is needed to identify which are those critical tasks that should be accelerated by hardware. A first software implementation of the complete algorithm is performed and those time-consuming tasks are selected.

As shown in fig. 4, the algorithm has been split into several sequential tasks. In the starting point both field orientation matrices are saved and available into the system memory. The original query matrix can be rotated by means of (7) and (8). Once the template and the original or rotated query matrices are ready, the next step consists in pre-aligning template and query matrices by moving the template matrix the translation offsets X and Y over the query. For each relative position, the amount of overlapped cells template-query is calculated. If the number of overlapped cells is above a specified threshold, the alignment analysis is performed. During the alignment process, the average and standard deviation of the orientation difference between both fingerprints taking into account all overlapped cells is computed. In order to accelerate the computation of the objective function, some optimizations are done as shown in (13):

$$\sum_{(i,j)}^{O} 1 = cells\ \ in\ overlap\ \ \ region\ \ = \sum n_i = N \tag{10}$$

$$\overline{\Delta\phi_O} = \frac{\sum_{(i,j)}^{O} \Delta\phi(i,j)}{\sum_{(i,j)}^{O} 1} = \frac{\sum_{(i,j)}^{O} \Delta\phi(i,j)}{N} \tag{11}$$

$$\sigma_O^2 = \frac{\sum_{(i,j)}^{O} \left(\Delta\phi(i,j) - \overline{\Delta\phi_O} \right)^2}{\sum_{(i,j)}^{O} 1} = \frac{\sum_{(i,j)}^{O} \Delta\phi^2(i,j) - N \cdot \overline{\Delta\phi_O}^2}{N} \tag{12}$$

$$f_O = \overline{\Delta\phi_O} + \sigma_O = \frac{\sum_{(i,j)}^{O} \Delta\phi(i,j)}{N} + \sqrt{\frac{\sum_{(i,j)}^{O} \Delta\phi^2(i,j)}{N} - \left(\frac{\sum_{(i,j)}^{O} \Delta\phi(i,j)}{N} \right)^2} \tag{13}$$

Two accumulators are used to compute $\sum\Delta\phi$ and $\sum\Delta\phi^2$. Therefore the calculation of the average and the standard deviation can be done simultaneously in the same loop and it is not needed to calculate the objective function in 2 sequential loops (the first one to calculate the average and a second loop to obtain from it the standard deviation). Moreover, and due to the fact that the fingerprint can present some low-quality regions, where it is difficult to extract a reliable field orientation, those poor cells are codified to the value 255. This value is clearly out of the expected range [0-180) so, 255 is used as an indicator flag in order for the alignment processor disregard the computation of those unreliable cells.

The algorithm tries to find the best alignment. The best alignment is the one that minimizes the objective function f_o while accomplishing the requirement of having a minimum amount of overlapped cells. If the overlapped area between both matrices is bigger than the threshold, the application calculates the average and standard deviation for the overlapped region. If the objective function weight is below the current weight result, the result is updated with the calculated weight. Thus, the result is always updated with the best found value. After computing the alignment between the query and the template for one relative position, the loop continues till all possible translations of the template matrix over the query are computed. Once computed all overlap possibilities for query orientation θ, the query matrix is then rotated again and the loop is repeated until all rotations are processed. At the end, the result points to the best alignment or, in case of not having found any good alignment, the system notifies the negative matching result.

Table 1 shows the execution time of each repetitive task when implementing the algorithm under a personal computer platform based on Intel Core2 Duo MCU @1,83GHz. The total execution time when comparing two matrices of 32x32 cells does not meet the real-time characteristics requested to the alignment algorithm. From table 1 it can be deduced that task 2 has clearly a higher computational cost than the others, so this task is pointed as the critical one that must be implemented by hardware and transferred to a FPGA device.

TABLE 1. Tasks execution times

Task	*Timing*
1) Query matrix rotation	*234 μs*
2) Template matrix translation and Query-Template overlap analysis	*1238 μs*
3) Objective function computation	*< 1 μs*
Total execution time (32x32 cells typ.)	*637500 μs*

HW-SW INTERFACE

The physical platform selected to implement the alignment algorithm is based on the EXCALIBUR system-on-chip family device from Altera. It embeds a 32-bit ARM MCU, a 1-Mgates FPGA and about 40-Kbytes of SRAM memory (split into single and dual port memory) in a single device. Furthermore, an external SDRAM memory is added to the embedded system to expand the total memory resources.

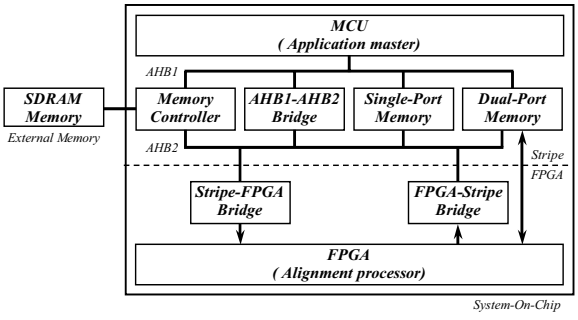

Fig. 5. Embedded system used in this work

In order to implement the interface between both cores (MCU and FPGA), there exist two AHB (AMBA High Performance Bus) buses. Moreover, the internal Dual-Port SRAM memory connects both cores so this memory can be used as an alternative way to exchange information during the application.

The application has been partitioned in order to have both cores working concurrently. For this reason, task 1 has been selected to be computed by the software block, whereas tasks 2 and 3 have been implemented by hardware. Given two field orientation matrices, the hardware coprocessor is responsible for analyzing all possible translations of the template with regard to the query and determining the best alignment. In parallel, while the FPGA is computing the alignment of both images, the MCU can calculate the rotated query to be analyzed in the next loop.

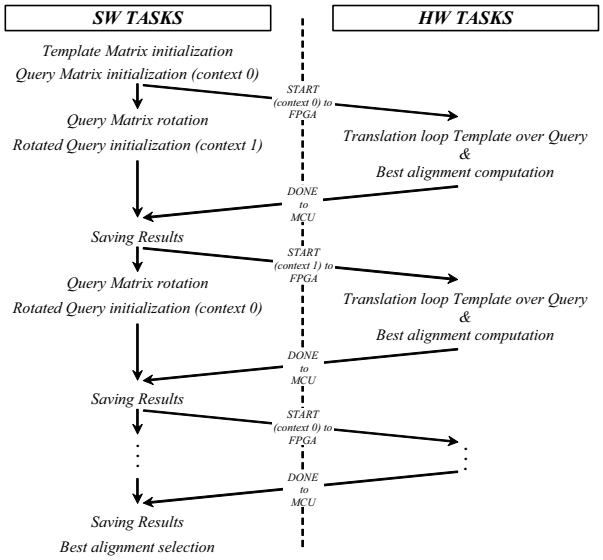

Fig. 6. Hw-sw partitioning

Two specific memory blocks have been synthesized in the FPGA in order to store the template and query field orientation matrices respectively. The MCU is responsible for initializing both memories. To allow the MCU to initialize the next rotated query matrix to be analyzed while the FPGA is running with the previous query, the query memory capability has been increased to allow 2 matrices to be stored simultaneously. The query memory has been spread into 2 contexts, and the MCU is responsible for indicating to the FPGA processor which context must be computed each time. While FPGA processes context 0, MCU initializes context 1, and vice versa. Figure 6 shows the final application flow for the alignment algorithm taking into account the hardware-software partitioning. For each rotation angle of the query map, the MCU computes the rotated map, and the hardware processor is in charge of determining the best alignment between the rotated query and the reference template map. In each iteration the MCU receives from the FPGA the alignment scores as well as the translation offsets of the best alignment found. Once all the rotation angles for the query fingerprint are analyzed, the best alignment among all combinations becomes the alignment result.

HW DESIGN

The block diagram of the alignment processor synthesized on the FPGA is shown in fig. 8. The hardware processor is responsible for finding the best alignment between two field orientation maps. During the alignment process, all possible overlap combinations between the template and the query matrices having a shared surface equal or bigger than a threshold of 16x16 cells are studied, as represented in fig. 7.

The scheduling of the application tasks is intended in order to accelerate as much as possible the computing.

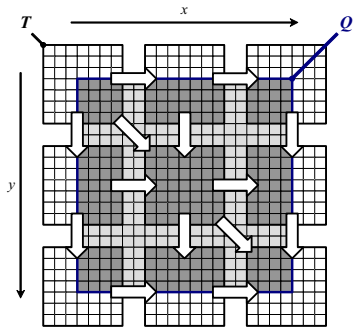

Fig. 7. Template matrix translation over query matrix

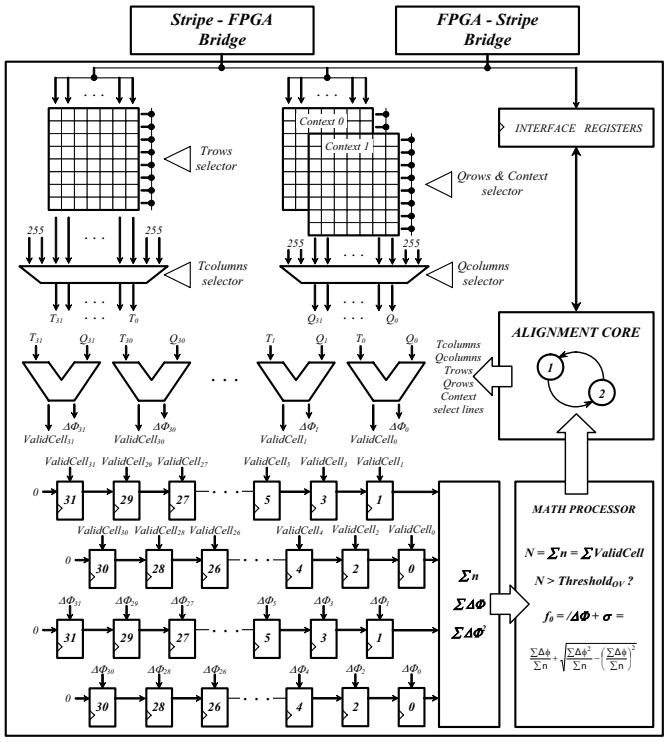

Fig. 8. Block diagram

Two levels of tasks scheduling are possible:

- Application level: both cores are working in parallel in order to speed up the alignment. The query memory is a single memory from MCU point of view, but it is split into 2 memory blocks from the point of view of the FPGA. With this, the query memory can be divided into 2 contexts allowing then the MCU to initialize one context while the processor is working with the other, thus making possible both cores to work in parallel. If several orientation angles for the query print need to be analyzed, while the FPGA processes the alignment for query orientation θ in context 0, the MCU can upload context 1 with the next rotated query to be analyzed (θ+Δθ), and vice versa.
- FPGA level: internally into the FPGA, the different steps in which the alignment algorithm is split are organized in pipeline strategy, as shown in fig. 9.

The alignment processor is able to process a complete row in parallel. This is done by widening the data buses into the FPGA. The maximum matrix wide is 32 cells, and 2 multiplexers are connected downstream the memories to select the proper columns in each step. These 2 multiplexers, together with the memories address bus permit the selection of the desired rows and columns at each time, so the translation of the query and template matrices is possible.

Any image is processed row by row. For each row, the processing consists in calculating the difference between the field orientation of the template and query cells. For each template cell, a query neighbourhood kernel 5x5 is taken into consideration, so each template cell is compared against 25 query cells, and the smallest field orientation difference is selected as the result. The processing of one row is done in 25+16 clocks. In each system clock one comparison is done, so 25 clocks are needed in order to find the best alignment between one template cell and its corresponding 25 query cells. After this period, a total of 32 partial results are obtained (corresponding to the complete matrix width), but from here it is needed to obtain $\sum n_i$, $\sum \Delta\phi_i$ and $\sum \Delta\phi_i^2$. This is done by splitting those 32 partial data into 2 rows (2x16) and shifting them through 2 specific accumulator processors. Therefore 16 additional clocks are needed to obtain the partial results of one row. This process is repeated in pipeline form for all overlapped rows under analysis until the end of the overlapped area.

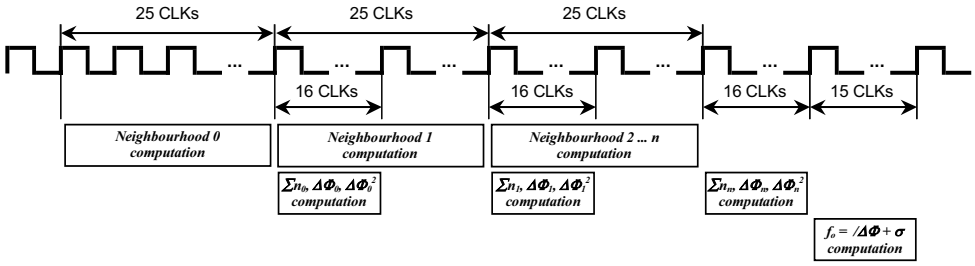

Fig. 9. Overlapped region processing pipeline

Once the complete image is processed, the total accumulation results $\sum n_i$, $\sum \Delta\phi_i$ and $\sum \Delta\phi_i^2$ corresponding to the overlapped image are obtained. After this, 15 additional system clocks are needed to compute the objective function f_O from the accumulator values $\sum n_i$, $\sum \Delta\phi_i$ and $\sum \Delta\phi_i^2$. The computation is repeated for any different overlap between the template and the query matrices by translating the template image a new offset over the query fingerprint. But only the results of the best alignment found are recorded by the FPGA at the end of the process.

All this process is managed by the hardware core, responsible for reading the START command and the configuration information coming from the MCU, and executing the rest of the computing. At the end of the process, the alignment results ($f_{objective}$, $\sum n_{objective}$, $X_{objective}$,$Y_{objective}$) are stored in specific interface registers and a DONE flag is set to the MCU.

CONCLUSIONS

The development of an automatic fingerprint alignment processor is suggested in this work. The comparison of the field orientation map extracted from a template fingerprint with the orientation map extracted from a query fingerprint permits to deduce if both finger impressions present a minimum overlap or not. In case there is no overlap between both, it is assumed that both prints are generated from different fingers. However, if some overlap exists, a deep analysis will proceed in order to decide if both prints come from the same user. A brute force analysis of the field orientation maps is done to find the best alignment of both prints: all possible overlaps between both images sharing a minimum and programmable area are studied and among them, the best overlap is selected. Apart from the high computational power demanded to the algorithm, an application with on-line characteristics is requested.

Once defined the alignment algorithm, a purely-software implementation under an embedded system is done in order to see which the limitations are. Using a 32-bit ARM processor running at 50 MHz, the alignment algorithm takes about 21791ms when typical sizes (256x256 pixels) for the input images are considered and no rotation is taken into account. This results in too much time for on-line applications. The hardware-software codesign of the algorithm is then proposed in order to reach real-time performance. The right partitioning of the application allows the efficient implementation of the system, from which it is possible to reach the timing requirements using an additional and small FPGA device.

Table 2 shows the final timing performance reached by the application when comparing it against a purely-software solution, as well as the additional hardware resources needed to accomplish the targets. It has been proven that in those applications where with only software-based solutions timing performance can not be reached, the hardware-software codesign techniques can be used as an alternative proposal. An efficient hardware-software partitioning of the algorithm has allowed the development of this kind of high-performance applications at low cost.

TABLE 2. System requirements and performance

Resources usage		
MCU	*f_{clk} (MHz)*	*50*
FPGA	*# logic cells*	*10601*
	# flip flops	*1844*
	# mem.bits	*139264*
	f_{clk} (MHz)	*50*
Application timing		
Task	*Hw*	*Sw*
1) Query matrix rotation	*-*	*7,8 ms*
2) Template matrix translation & Query-Template overlap analysis	*16,3 μs*	*36,1 ms*
3) Objective function comp.	*0,3 μs*	*3,2 μs*
Total execution time (only sw)	*21791,1 ms*	
Total execution time (hw-sw)	*41,6 ms*	

REFERENCES

[1] A. Ross, J. Reisman, A. Jain, "Fingerprint Matching Using Feature Space Correlation", Post-ECCV Workshop on Biometric Authentication, LNCS 2359, 2002, pp. 48-57

[2] L. Hong, Y. Wan, A. Jain, "Fingerprint Image Enhancement: Algorithm and Performance Evaluation", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, 1998, pp. 777-789

[3] D. Maltoni, D. Maio, A. K. Jain, S. Prabhakar, "Handbook of Fingerprint Recognition", Springer, 2003

[4] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, A. K. Jain, "FVC2004: Third Fingerprint Verification Competition", Proceedings of ICBA 2004, LNCS 3072, pp. 1-7