Flexible Biometric Online Speaker Verification System Implemented on FPGA Using Vector Floating Point Units

Enrique Cantó-Navarro, Mariano López-García, Rafael Ramos-Lara, and Raúl Sánchez-Reíllo, *Member, IEEE*

Abstract— **This paper presents the implementation on FPGA of an speaker verification system. The algorithm is executed by software over an embedded system that includes a MicroBlaze microprocessor connected to a Vector Floating-Point Unit (VFPU). The VFPU is designed to speed up the resolution of any vector floating-point operation involved in the verification algorithm, whereas the microprocessor manages the control of the process and executes the rest of operations. With a clock frequency of 40 MHz, the system is capable of executing in real-time the complete algorithm, processing a voice frame in 9.1 ms. The same verification process was carried out over two different systems: an ARM Cortex A8 microprocessor and configuring MicroBlaze with the scalar Floating-Point Unit provided by Xilinx. Experimental results show that when comparing our proposal against both systems, the number of clock cycles is reduced by a factor of x11.2 and x15.4, respectively. The main advantage provided by the VPFU is its flexibility, which allows quickly adapting the software to the potential changes produced in both the system and the user requirements. The algorithm was tested over a public database, which contains utterances of different users acquired under different environmental conditions, providing good recognition rates.**

Index Terms—**Biometrics, field programmable gate arrays, hardware-software co-design, speaker recognition, system-on-chip.**

I. Introduction

SPEAKER verification is one of the most widespread technologies used today in biometrics. This modality is supported by a solid signal processing theory developed for decades, and is based on robust algorithms that provide high recognition rates [1]. Moreover, the low price of microphones is a significant advantage against other types of biometrics requiring more expensive capture devices. Such a feature is fundamental for the rapid expansion of this technology in the low-cost consumer market. Nowadays, some smartphones already include applications that allow authorized users to unlock their phone by reproducing a passphrase, which replace the traditional methods based on key or PIN numbers. In addition, commercial transactions, or remote personal verification by phone, are also applications of speaker verification that could be currently used by a large segment of the population [1], [2].

Most speaker verification systems adopt an architecture that consists of two differentiated phases: enrolment and classification. During the enrolment phase, the system is trained with several utterances provided by a specific user. Utterances are processed in order to obtain a set of features that represent the specific physical structure of the individual's vocal tract. Although there are many approaches for extracting these features, the Mel-Frequency Cepstrum Coefficients (MFCC) have been the most widely used in speaker verification, as well as in speech recognition [3]-[5]. These coefficients have the ability to faithfully represent the distinguishing features of a signal of voice belonging to a particular user. In addition, they show a robust behaviour against background noise, providing high recognition accuracy even in text-independent scenarios. Finally, based on these coefficients, a model for each user is generated. In the classification phase, the user, whose identity should be verified, pronounces a new utterance. This utterance is processed and its feature vector, based on the MFCC coefficients, is extracted and compared against the model previously stored during the enrolment phase. The comparison returns a result, which is used to accept or deny the identity claimed by the user.

The classification phase, also known as the matching process, is based on algorithms capable of distinguishing the extracted features of any individual from the genuine user. Algorithms based on generative models, such as Hidden Markov Models or Gaussian Mixture Models, have traditionally been applied to speaker verification [6], [7]. The success of these statistical methods depends on the proper estimation of their parameters, which are calculated by

Enrique Cantó-Navarro is with the Department of Electronic Engineering, Electrical and Automatics, Universitat Rovira i Virgili, Tarragona, Spain (corresponding author phone: 34-977558522; fax: 34-977559605; email: enrique.canto@urv.cat).

Mariano López-García and Rafael Ramos-Lara are with the Department of Electronic Engineering, Universitat Politècnica de Catalunya, 08800 Vilanova i la Geltrú, Spain (email: mariano.lopez@upc.edu and rafa.ramos@upc.edu).

Raul Sánchez-Reíllo is with the Department of Electronic Technology, University Carlos III of Madrid, 28911 Spain (e-mail: rsreillo@ing.uc3m.es).

This work was supported by the Ministerio de Economía y Competitividad in the framework of the Programa Nacional de Proyectos de Investigación Fundamental, project TEC2012-38329-C02-02.

maximizing the likelihood of the data for a particular model. Such estimation provides good convergence properties, although it does not guarantee finding the global maximum [8], [9]. In contrast, Support Vector Machine (SVM) is a discriminative approach that is intended to estimate a decision surface directly, rather than modeling a probability distribution function [10]. Several versions of this algorithm, which use different kernels, have already been employed in speaker verification, providing outstanding matching results [11].

On other hand, due to the large number of computationally demanding operations performed during the verification process, the computational cost of these algorithms is usually high. Additionally, the platform in which the system is implemented, should be able to manage an important amount of data in real time. These issues are especially important in text-independent scenarios, where the verification process is achieved by pronouncing utterances of several seconds in length. Microprocessors of moderate economic cost have been proposed as a compromise solution for the implementation of biometric systems. The flexibility of a software implementation allows the rapid development of applications using fixed hardware architectures. However, the complexity of some algorithms leads to only high-performance microprocessors, included in personal computers, being appropriated for their processing in real time, at the expense of increasing the price and power consumption of the whole system. Hardware architectures based on Field Programmable Gate Arrays (FPGAs) are another alternative for implementing these algorithms. At a reasonably low-cost, these devices allow designing high-speed architectures useful for implementing biometric algorithms [12], [13]. There are also some publications addressing the implementation of speaker or speech recognition systems on FPGAs. Most of these publications proposed the implementation of only one stage, feature extraction or classification, for accelerating the total execution time [14]-[17]. Recently, the work of Ramos et al. [18], presented a more advanced implementation of a complete speaker verification system on an FPGA. Authors pointed out that their implementation was able to carry out the feature extraction and classification stages in a shorter time than the frame length (real-time processing). However, this performance was achieved at the expense of designing the system according to a specific sample frequency, a particular codification rate and transforming the original floating-point operations to a fixed-point format. Note that, any change introduced in any of the previous parameters involves redesigning the entire system.

These drawbacks can be partially overcome by adding a Floating-Point Unit (FPU) as part of the FPGA design. However, most of the proposed FPUs only include basic arithmetic operations (mul, add, sub and div) and are unable to process biometric algorithms in real time [19], [20]. An additional disadvantage, within the framework of biometrics, is that these math coprocessors only admit scalar numbers as input operands. Many functions used in speaker verification perform computations whose input operands are vectors of variable length. Although the FPU is able to resolve these computations, the time needed for their calculation could be significantly accelerated if a Vector Floating-Point Unit (VFPU) is used [21], [22]. When operating with vectors, the VFPU increases the throughput by reducing both the number of CPU fetches and the number of memory accesses.

This paper presents the implementation of a whole MFCC-SVM speaker verification system on an FPGA. The system consists of the Xilinx MicroBlaze general-purpose 32-bit microprocessor, and a VFPU that calculates any vector operation defined in floating-point format. The architecture of the VFPU is generic, so that it can be easily adapted to other soft-core microprocessors or FPGA families. Compared with a custom-hardware implementation, the main feature of the VFPU is its flexibility, which provides the possibility of easily introducing modifications in the algorithm or adding new processing stages. Besides, in the particular case of a speaker verification system, such flexibility allows designing the VFPU independently of the number of bits used for the codification of the input samples or the number of coefficients included in the feature vector. Furthermore, in applications in which samples of voice are affected by environmental conditions (background noise, distortion, etc.), or users have a remarkable common characteristic in their voice (due to age, prosodic features, etc.), changes in the parameters can be quickly introduced for adapting the system to such particular characteristics in order to improve the recognition rates. For instance, in [23] authors proposed a training process that includes simulated noisy data that provides an improvement higher than 23% compared with a classic training method. In [24], it is shown as due to the effect of aging, the recognition rates can be degraded by approximately 20% every 1-2 years. Finally, in [25] it is demonstrated as recognition performance based on cepstral features is improved by adding higher-level information, including prosodic and lexical features, which allows the equal error rate to be reduced by up to 19%.

The paper is organized as follows. Section II describes the basic theory about the algorithm presented based on MFCC and SVM. Section III presents the internal structure of the VFPU and the floating-point operations being implemented. Section IV shows the experimental results, and finally Section V shows the conclusions.

II. ONLINE SPEAKER VERIFICATION ALGORITHM

The architecture of the proposed speaker verification system is well-documented, so it will be briefly reviewed here [18]. Specifically, the block feature extraction is based on the MFCC coefficients, whereas the SVM algorithm is the basis for designing the block creation model and classification.

A. MFCC-based feature extraction

The MFCC are the essential elements used for obtaining the feature vector x_m. During the feature extraction, the speech signal is segmented into frames of 25 ms in length. The frame m is the basic unit from which the feature vector is obtained. Any frame is overlapped by 15 ms with its previous one, being 10 ms the frame advance.

TABLE I
FUNCTIONS USED TO CALCULATE THE FEATURE VECTOR ASSOCIATED WITH FRAME m USING A NUMBER OF SAMPLES *V*=200.

Description	Mathematical Expression
1.- Average value	$\bar{S} = \frac{1}{V}\sum_{n=0}^{V-1} x_m(n)$
2.- Signal Normalization	$\bar{x}_m(n) = x_m(n) - \bar{S}, \quad 0 \le n \le V-1$
3.- Energy	$C_o(m) = \ln\left[\sum_{n=0}^{V-1}\left(\bar{x}_m(n)\right)^2\right]$
4.- Pre-emphasis filter	$y(n) = \bar{x}_m(n) - a \cdot \bar{x}_m(n-1), \ 0 \le n \le V-1, \ a = 0.97$
5- Filtering with Hamming window	$z(n) = y(n) \cdot w(n)$, being: $w(n) = \begin{cases} 0.54 - 0.46 \cdot \cos\left(\frac{2 \cdot \pi \cdot n}{V-1}\right), & \text{if } 0 \le n \le V-1 \\ 0, & \text{otherwise} \end{cases}$
6.- Zero padding	$x(n) = \begin{cases} z(n), & \text{if } 0 \le n \le V-1 \\ 0, & \text{if } V \le n \le 255 \end{cases}$
7.- Fast Fourier Transform (FFT)	$X(k) = \sum_{n=0}^{L-1} x(n) \cdot e^{-j \cdot \frac{2 \cdot \pi \cdot k \cdot n}{L}}, \ 0 \le k \le L-1 \ (L = 256)$
8.- Bank of Mel filters	$S(k) = \lvert X(k) \rvert \cdot Mel(f), \ 1 \le k \le 26$ $Mel(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right)$
9.- Napierian logarithm of the absolute value	$f(k) = \ln\lvert S(k) \rvert, \quad 1 \le k \le 26$
10.- Discrete cosine transform	$C_n(m) = \sqrt{\frac{2}{K}} \cdot \sum_{k=1}^{K} f(k) \cdot \cos\left[n \cdot (k - 0.5) \cdot \frac{\pi}{K}\right]$, being $K = 26$ and $1 \le n \le 12$
11.- Delta coefficients	$\Delta C_n(m) = \frac{\sum_{t=-M}^{M} t \cdot C_n(m+t)}{\sum_{t=-M}^{M} t^2}, \ 0 \le n \le 12 \text{ and } M = 2$

Usually, the optimal number of parameters that form the feature vector is 26. The first one, $C_0(m)$, is the Napierian logarithm of the energy localized in a temporal window; the following twelve, $C_1(m),\ldots,C_{12}(m)$, are directly based on the Mel-frequency Cepstrum Coefficients, which represent the spectral envelope of the signal of voice [3]-[5]; and the last thirteen are known as differential or delta coefficients and are denoted as $\Delta C_0(m),\ldots, \Delta C_{12}(m)$. Such delta coefficients represent the variation of the MFCC between adjacent frames. Table I summarizes the complete sequence of functions and operations involved in the calculation of the feature vector. Functions are listed in sequential order regarding their execution (the final parameters are outlined in bold). Note that the first parameter is directly obtained in step 3. The MFCC coefficients are calculated in step 10, after applying several functions over the original frame *m*. Finally, delta coefficients are obtained in step 11 with a delay of 2 frames. As can be seen, to compute these delta coefficients it is necessary to calculate before the MFCC of previous frames *m-2*, *m-1,* and subsequent frames *m+1* and *m+2*, respectively.

B. Model creation

The aim of the training stage is to create a model for each user, which contains the main characteristics that represent the user's voice. The data employed to build such a model include several feature vectors of this user (genuine), as well as other feature vectors belonging to different people (impostors). The model is obtained off-line, using a training algorithm that runs in a desktop computer, and in this particular case, employing the public database BANCA [26]. Since the classification stage is implemented by an SVM algorithm, the model consists of a set of *Q* support vectors $y_j(i)$ ($i=0..25, j=0..Q-1$) and their associated parameters (ρ, γ, P_j). Note that the size of $y_j(i)$ (26 elements) coincides with the number of parameters that form the feature vector x_m. LIBSVM or Torch are specific libraries suitable to perform the training process of classifiers based on SVM models [27], [28]. The training algorithm is executed several times, changing the number of feature vectors, and adjusting a characteristic threshold called ρ. The purpose is to find the optimal number *Q* of support vectors $y_j(i)$ that lead to the best classification results. In this particular case, we found that such optimal number *Q* is 3,636. Each of these support vectors $y_j(i)$ has an associated constant P_j (Lagrange coefficient) provided as a result of the training algorithm, along with an additional parameter γ common to all support vectors. The optimal values obtained for γ and ρ are 0.4 and -0.44, respectively. Besides, the total number of feature vectors used in this off-line training process is 8,000, including genuine and impostor users Thus, the model λ for a specific user can be described as:

$$\lambda = \left(\gamma, y_j(i), P_j, \rho\right) \ j = 0..Q-1 \text{ and } i = 0..25 \tag{1}$$

C. SVM-based classification

The aim of the SVM classifier is to figure out whether the feature vector matches or not the user's model. Such a comparison returns a binary result, which assigns 1 when the match is positive, and 0 when is negative. Note that this comparison is performed for each frame *m*.

SVM maps the input data into a higher dimension feature space, in which a hyperplane, that separates and maximizes the margin between classes, is found. Such a transformation, from an initial space to another of a higher dimension, is achieved by means of a function kernel. One of the most common kernels used in identification is the radial basis function. This function, adapted to the context of speaker verification, is represented by the following expression:

$$\alpha(m) = \sum_{j=0}^{Q-1} P_j \cdot e^{-\gamma \sum_{i=0}^{25}\left(x_m(i) - y_j(i)\right)^2} \tag{2}$$

being $x_m(i)$ the feature vector of frame *m* described by:

$$x_m(i) = \begin{cases} C_i(m), & 0 \le i \le 12 \\ \Delta C_{i-13}(m), & 13 \le i \le 25 \end{cases} \tag{3}$$

The calculation of expression (2) is, by far, the most time-consuming process that involves the most intensive computations. The result $\alpha(m)$, along with the threshold parameter ρ obtained during the training stage, is employed for determining the assignation value $\beta(m)$:

$$\beta(m) = \begin{cases} 1, & \text{if } \alpha(m) \le \rho \ \text{(positive match)} \\ 0, & \text{otherwise (negative match)} \end{cases} \tag{4}$$

Finally, after analyzing all the frames in the utterance, if the percentage (denoted as Matching) of feature vectors belonging to the user's model overcomes a threshold, the identity claimed by the user is confirmed as genuine (T refers to the total number of frames):

$$Matching = \frac{\sum_{m=1}^{T} \beta(m)}{T} \cdot 100 \quad (5)$$

III. VECTOR FLOATING-POINT UNIT ARCHITECTURE

A. VFPU description

Fig. 1 and Fig. 2 show the internal architecture of the VFPU presented in this paper. Its main features can be summarized as follows:

1) The VFPU executes computations on vectors of arbitrary size using operands of single precision (32-bit) defined by the standard IEEE-754. Using this format the total compatibility between the data shared by the microprocessor and the VFPU is ensured. Although a design based on half-precision (16-bit) would consume lower hardware resources, in such case a block for data conversion should be included to guarantee the compatibility between both formats, which may introduce for some stages a penalty on the execution time. Furthermore, using half-precision the computations performed in some stages may lead to produce underflow (overflow) errors, which should be conveniently managed to avoid their potential effect on the recognition process.

2) Computations can be performed with vectors stored in external memory, scalar numbers provided by the microprocessor, or any combination of them. Likewise, the result of any computation can be placed on an external memory, or read by the microprocessor.

3) The internal architecture of the VFPU is designed to optimize vector computations based on the execution of a set of basic floating-point operations. In this way, these computations can be performed avoiding unnecessary accesses to external memory, which are used to store temporary results.

The data-path, described in Fig. 1, basically consists of four blocks. The *Bus Interface* connects the VFPU to the microprocessor through the system bus. This *Interface* is in charge of managing the writing in registers S0 to S7 of both scalar operands and memory addresses, in which the vectors are located. The *Memory FIFO* reads these vectors directly from external RAM, and writes in the same memory the resulting vector obtained after finishing a set of operations. The *Register File* contains 16 registers of 32-bits (R0 to R15), which store the result provided by the FPU. These registers can be used as new operands in subsequent operations. The FPU is designed in order to perform the operations described in Table II. As Fig. 2 shows, its internal design includes a specific block capable of performing the exponential function, which is the basis of the kernel employed by the SVM classifier. The addition of this block, as part of the VFPU, is necessary to solve the algorithm in real-time, since the SVM is the most time-consuming process. Furthermore, the table also presents the throughput and the latency (in clock cycles, T_{CLK}) for each operation. As the FPU is internally segmented into several stages, the number of results per second available at its output depends on the type of operations launched by the control unit. For instance, although the latency of an exponential function is $37 \cdot T_{CLK}$, when such an operation is consecutively executed more than once the second and subsequent results are obtained in $29 \cdot T_{CLK}$ (throughput of 34.5 kFLOP/MHz). The rest of the operations behave in a similar way, so that as indicated in Table II, the maximum throughput provided by the VFPU is 1 MFLOP/MHz.

TABLE II
LATENCY AND THROUGHPUT FOR OPERATIONS PERFORMED BY THE FPU

Operation	*Latency*	*Throughput*
Addition, subtraction, int. to float, multiplication	$4 \cdot T_{CLK}$	1 MFLOP/MHz ($1 \cdot T_{CLK}$)
Division, square root	$33 \cdot T_{CLK}$	38.5 kFLOP/MHz ($26 \cdot T_{CLK}$)
Absolute value, negation	$1 \cdot T_{CLK}$	1 MFLOP/MHz ($1 \cdot T_{CLK}$)
Exponential, logarithm	$37 \cdot T_{CLK}$	34.5 kFLOP/MHz ($29 \cdot T_{CLK}$)

The de-normalization block (Denorm) turns the IEEE-754 format into a fixed point more suited to carry out any operation. Once the process of calculation is finished, the normalization (Norm) and rounding (Round) blocks perform the opposite operation providing the result in the original format.

B. Architecture for maximum speed processing

The design of the VFPU should be performed in order to ensure the capacity of the system to work in real time, so that the feature extraction and matching processing of frame m should be performed before a new frame ($m+1$) is ready to be

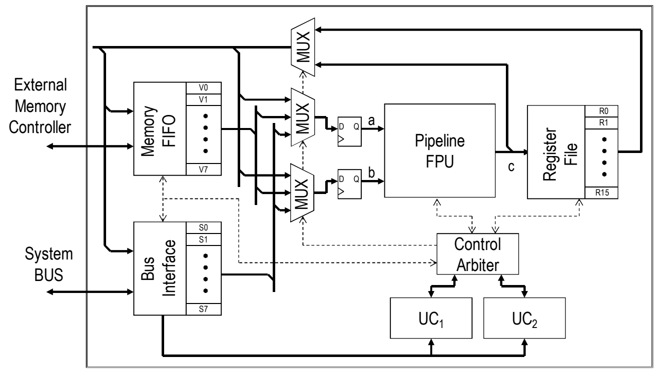

Fig. 1. Datapath description for the VFPU.

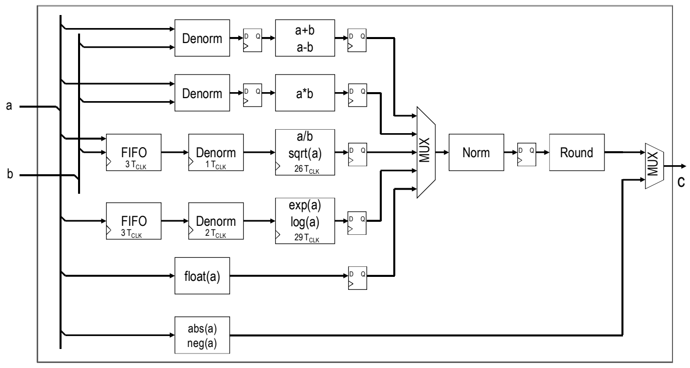

Fig. 2. FPU internal architecture.

processed (in our particular case this time is 10 ms, which is the frame advance). The aim of this section is to highlight the computational features of the architecture proposed in section III.A. For that purpose, a simple example based on the processing function represented in (6) is chosen:

$$\alpha(m) = \sum_{j=0}^{3635} e^{\sum_{i=0}^{25}\left(x_m(i)-y_j(i)\right)^2} \qquad (6)$$

Note that this function is very similar to the kernel used by the SVM classifier shown in (2), but setting γ and P_j at -1 and 1, respectively, and fixing the number of support vectors to 3,636. It should be pointed out that, as it is described in section II.B, the actual values used in the experimental results for γ, ρ and P_j are different and are obtained by applying the training algorithm. For the sake of simplicity, let us suppose that initially the resolution of (6) is performed using only one control unit (UC). The program that should be executed is represented in Fig. 3.

Clearly, in such a program two loops can be identified: the external loop, indexed by j and related to the number of support vectors; and the internal loop managed by index i, whose upper limit is determined by the size of the feature vector. Note that such a code is highly vectorizable, that is, elements of the input (output) vectors $x_m(i)$ and $y_j(i)$ are read (written) in sequential order and the same sequence of operations is repeated over all the elements of a vector. As it will be shown in section IV, this property is very important to achieve high acceleration factors by the VFPU. Thus, the microprocessor manages the execution process and solves the non-vectorizable code, whereas the VFPU is in charge of performing all the vectorizable operations involved in (6). Fig. 4 shows as such operations are launched sequentially, according to their particular latency represented in Table II. This simple structure works properly, but it has two important drawbacks:

- The UC only launches a new operation if both the input operands and the block used for implementing the operation are available. As a consequence, additions, subtractions and multiplications employed to calculate the exponent of (6) take 4·T_{CLK}. This time is far from the maximum throughput achievable by these operations (1 MFLOP/MHz).
- Note that once the exponential function is launched, the processing of a new exponent could be started (the exponential result is not required for its evaluation). However, since

```
while (T<Num_Frames) {
  R12=0;
  for (j=0; j<3636; j=j+1) {
     VFPU_UC {
        R4=0;
        Loop (i=0; i<26: i++) {
           R0 ← xm(i) – yj (i);      // xm(i) – yj (i);
           R0 ← R0 · R0;             // [xm(i) – yj (i)]²
           R4 ← R4 + R0;             // Σ[xm(i) – yj (i)]²
        }
     R8 ← expj (R4);                 // exp {Σ[xm(i) – yj (i)]²}
     R12 ← R12+R8;                   // Σ exp {Σ[xm(i) – yj (i)]²}
     }
  }
}                                    Code executed by the VFPU
```

Fig. 3. Code executed by the VFPU for solving expression (6).

operations are executed sequentially, the UC must wait until the result of the exponential is accumulated in register R12.

The efficiency of this structure can be readily improved by introducing some modifications in the software program oriented to mitigate the first drawback. The idea, shown in Fig. 5, is very simple and allows the throughput to be maximized when calculating the exponent value. Now, the outer loop is partially unrolled to compute groups of four support vectors at the same time. For instance, in the example of Fig. 5, the VFPU operates with support vectors ($j+i$) ($i\in$ [0:3], being j=4·k and $k\in$ [0:908]). Thus, the UC would launch four times the same operation in four consecutive clock cycles; one for each of the four support vectors processed in parallel. As Fig. 4 shows, and taking into account latencies presented in Table II, using the initial pseudocode any support vector can be processed in 353·T_{CLK}. However, as indicated in Fig. 5, when introducing the proposed software modification, four support vectors can be processed in 440·T_{CLK}, which is equivalent to processing each one in only 110·T_{CLK}. Note that the second and subsequent exponentials are calculated in 29·T_{CLK}, according to the throughput of this operation.

On the other hand, the speed of the memory controller, along with the amount of data to be read, may limit the performance of the VFPU. The memory controller has been designed to read new data in 1·T_{CLK}. Since vectors $x_m(i)$ and $y_j(i)$ have 26 elements in each one, the exponent evaluation involves reading 208 data ([26·$x_m(i)$ + 26·$y_j(i)$]·4 support vectors). The execution time needed to process (6) is not only affected by the memory bandwidth, but also by the improvement obtained when processing groups of four support

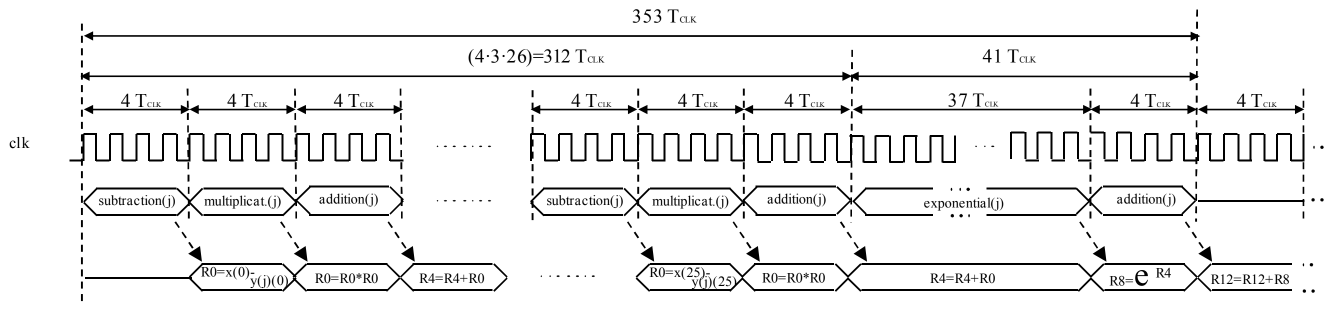

Fig. 4. Operations and results for a VFPU that includes only one control unit.

vectors in parallel. Then, taking into account these considerations, this time could be approximated by the following expression:

$$T_{UC} = \frac{3636}{4} \cdot \left[N_{INIT} + \max(208,440) \right] T_{CLK} = \frac{3636 \cdot 449 \cdot T_{CLK}}{4} \quad (7)$$

where *max(a,b)* is a function that returns the maximum value. N_{INIT} represents the initial latency (not included in Fig. 4 and Fig. 5) necessary for the initialization of the UC ($5 \cdot T_{CLK}$) and those registers ($4 \cdot T_{CLK}$) that work as accumulators ($N_{INIT} = 9 \cdot T_{CLK}$).

The second drawback, related to the wait-states cycles introduced by the UC, could be eliminated including two control units (UC1 and UC2) and using pipeline techniques at function level. Using this new structure, the UC1 controls the calculation of the exponent, whereas the UC2 manages the evaluation of the exponential. Since both control units try to access the FPU, an arbiter is needed to manage the permissions (see Fig. 1). Fig. 6 shows as the control units launch operations following a specific sequence, so that during the calculation of the j_{th} exponent, the exponential of the previous one $(j\text{-}1)_{th}$, is also processed in parallel. Using only one control unit (Fig. 4), the exponent and exponential function (including the accumulation) are solved in $312 \cdot T_{CLK}$ and $41 \cdot T_{CLK}$, respectively. However, if both operations are launched in parallel, the execution time is mainly dominated by the calculation of the exponent, which is the longer operation. In addition, if this hardware structure is combined by programming the VFPU in such a way that groups of four support vectors are processed at the same time (loop unrolling), the resulting throughput is substantially increased. This improvement is represented in Fig. 7, which shows as operations are launched in a specific order by the control units and their impact on execution time. This design of the VFPU has some interesting features:

- Note that the operations (subtraction, multiplication and addition) launched by the UC1 are executed in pipeline (throughput 1 MFLOP/MHz), so that their execution time is equal to the number of operations managed by such a control unit ($N_{UC1} = 312 \cdot T_{CLK}$).
- The operations launched by UC2 interrupt the pipeline created by UC1, since UC2 takes the control of the bus arbiter and stops the operations controlled by UC1. The delay introduced by this interruption adds an additional execution time, which is equal to the number of operations managed by UC2 ($N_{UC2} = 8 \cdot T_{CLK}$).
- As Fig. 2 shows, the normalization and rounding blocks are shared by all operations except by *abs(a)* and *neg(a)*. Thus, when a new result is available at the output of the exponential, the UC1 is forced to delay $1 \cdot T_{CLK}$ the launching of a new operation. This clock cycle is the time needed by the block Norm of Fig. 2 to normalize any value. Therefore, the

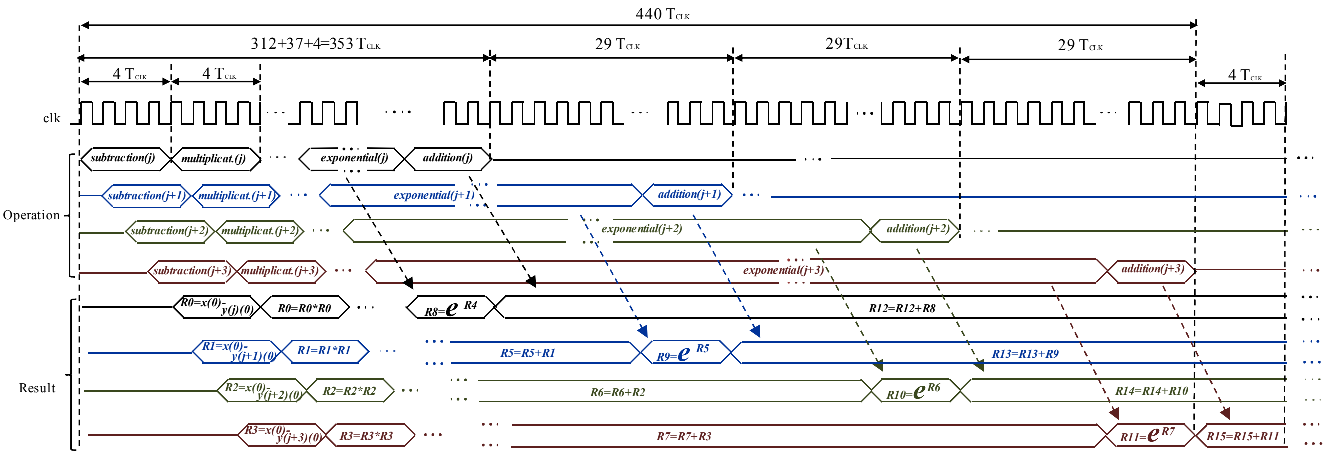

Fig. 5. Increasing the thoughput by software. Groups of four support vectors are managed by the VFPU (only one control unit).

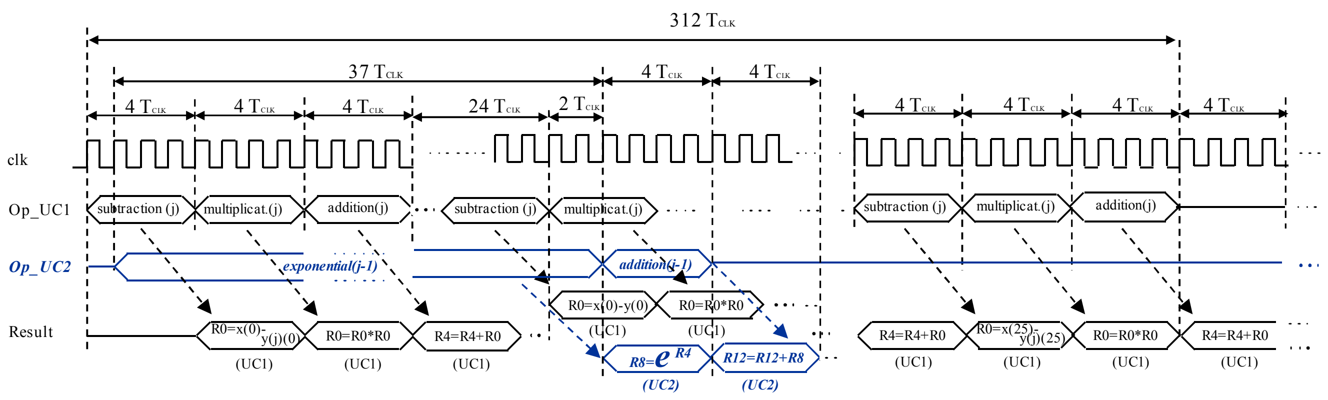

Fig. 6. Operations and results for a VFPU with 2 control units (here, the technique of using groups of four support vectors is not employed).

delay added to the execution time is equal to $4\cdot T_{CLK}$, which is the number of exponential functions managed by UC2.

In order to obtain a general expression, let Q and F be the number of support vectors and the size of a feature vector, respectively, and let N_{INIT} be the initial configuration delay defined previously. The time needed to solve (6) using a VFPU with two control units and resolving groups of four support vectors in parallel could be approximated by:

$$T_{2UC} = \frac{Q}{4}\left[N_{INIT} + \max(4\cdot 2\cdot F, N_{UC1} + N_{UC2} + 4)\right]T_{CLK} = \frac{Q}{4}\left[N_{INIT} + 4\cdot\max(2\cdot F, 3\cdot F + 3)\right]T_{CLK} \tag{8}$$

Note that like expression (7), this execution time T_{2UC} does not depend on the speed of the memory controller, since the addition of $N_{UC1}=(4\cdot 3\cdot F)$ plus $N_{UC2}=(4\cdot 2)$ is higher than $4\cdot 2\cdot F$ (the number of cycles devoted to read data from external memory). Analyzing expression (8) and substituting F and Q by 26 and 3636, respectively, it is easy to conclude that the total number of clock cycles needed to calculate (6) is $302{,}697\cdot T_{CLK}$. Consequently, the average throughput provided when processing these computations is about 0.96 MFLOP/MHz, very close to the maximum theoretical value of 1 MFLOP/MHz provided by the VFPU.

A simple way of increasing the computational capability of the VFPU is augmenting the number of lanes that form the architecture. Lanes are implemented by creating N identical copies of the FPU included in Fig. 1. Thus, from a theoretical point of view, the execution time is reduced by N, since the system is able to process in parallel N groups of four support vectors $y_j(i)$. However, this reduction is also achieved at the expense of increasing the total area by N. Additionally, the more lanes are included, the more important is the computational capability of the VFPU, but also the more significant are the limitations introduced by the memory controller. If the system includes N lanes related to N identical FPU, expression (8) is modified as follows:

$$T_{NLanes} = \frac{Q}{N\cdot 4}\left[N_{INIT} + \max(N\cdot 4\cdot 2\cdot F, N_{UC1} + N_{UC2} + 4)\right]\cdot T_{CLK} = \frac{Q}{N\cdot 4}\left[N_{INIT} + 4\cdot\max(N\cdot 2\cdot F, 3\cdot F + 3)\right]\cdot T_{CLK} \tag{9}$$

Note that time reduction is not proportional to the number of lanes N. In fact, depending on both, the size of the feature vector F and the number of lanes N, this time T_{NLanes} could be limited by either the amount of memory accesses ($N\cdot 4\cdot 2\cdot F$) or by the number of cycles needed by both control units $4\cdot(3\cdot F+3)$ to solve the operations. Thus, the addition of new lanes does not always provide the expected benefits in terms of computational capability.

However, in situations in which the memory access is the most restrictive term in (9), there are some modifications that can be added in the design of the VFPU. Usually, these modifications involve a trade-off between resource utilization and performance. For instance, the vector $x_m(i)$, which is identical for all support vectors $y_j(i)$ (j=0..3665), could be read only once in order to save memory accesses. Such a vector could be initially stored in an internal circular shift register (CSR) and used when required by the operations involved in (6). Thus, when including a CSR as part of the VFPU, expression (9) would be modified as follows:

$$T^{CSR}_{NLanes} = \left\{\left[N_{INIT} + F\right] + \frac{Q}{N\cdot 4}\left[N_{INIT} + 4\cdot\max(N\cdot F, 3\cdot F + 3)\right]\right\}\cdot T_{CLK} \cong \frac{Q}{N\cdot 4}\left[N_{INIT} + 4\cdot\max(N\cdot F, 3F + 3)\right]\cdot T_{CLK} \tag{10}$$

where (N_{INT}+F) represents the time needed to read $x_m(i)$, which could be neglected when compared with the rest of terms of (10). Unlike expression (9), when N=2 the execution time is now limited by the number of operations launched by both control units, rather than by the memory accesses. After performing some preliminary designs, we realized that when

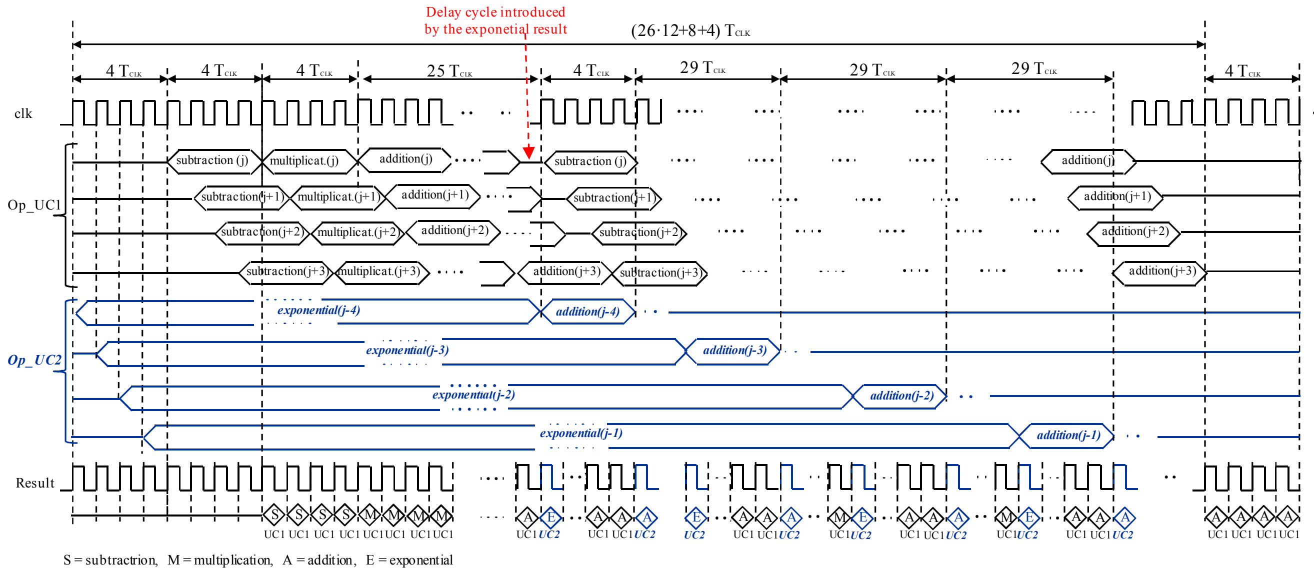

Fig. 7. Operations and results using 2 control units. Groups of four support vectors are processed at the same time.

TABLE III
NORMALIZED AREA, TIME AND THROUGHPUT REGARDING THE SIMPLEST SOLUTION (*N*=1, 1 UC AND WITHOUT INCLUDING CSR). THE FREQUENCY IS 40 MHZ.

F=26 Q=3636	*System*	*AREA*	*TIME*	*THROUGHPUT (MFLOP/MHZ)*
N=1	1 UC	1	1 (10.20 ms)	0.71 (71%)
	2 UC	1.10	0.74 (7.57 ms)	0.96 (96%)
	2 UC + CSR	1.18	0.74 (7.57 ms)	0.96 (96%)
N=2	1 UC	1.80	0.50 (5.10 ms)	1.43 (71%)
	2 UC	1.90	0.48 (4.83 ms)	1.51 (75%)
	2 UC + CSR	2.06	0.37 (3.78 ms)	1.92 (96%)
N=4	1 UC	3.40	0.47 (4.78 ms)	1.51 (38%)
	2 UC	3.50	0.47 (4.78 ms)	1.51 (38%)
	2 UC + CSR	3.82	0.24 (2.41 ms)	3.01 (75%)

including the CSR the area is increased about 0.1·*N*. Table III shows the trade-off between area and performance for different values of *N* and using one or two control units (1 UC or 2 UCs). Results are normalized regarding the simplest design, which is based on one lane and one UC (first row of table III). Note that the maximum value tested for *N* is four, since with a higher value the execution time would be limited by the memory accesses. In the design based on two UCs and one lane, the inclusion of a CSR only increases the area, but does not give any additional advantage in terms of speed. However, the use of two UCs is interesting, since it reduces the time by 25.8% and only increases the area by 10%. For two lanes, the fastest solution is achieved including two UCs and a CSR. The CSR increases the area about 8.4%, with regard to the design based on two UCs, but also it reduces the time by 22.9%. Likewise, when *N*=4 the addition of a second control unit does not reduce the resolution time. However, it is observed as when the CSR is added, the increase of area is about 9%, but a significant improvement is obtained in the execution time (48.9%). As it will be shown in next section, the optimal solution is obtained including only one lane and two UCs. This will be the structure employed in the experimental results, since such implementation is able to process frames in real-time using the minimum area and providing the maximum throughput.

IV. EXPERIMENTAL RESULTS

In order to experimentally prove the advantages of our proposal, a XC3S2000 Spartan 3 FPGA has been selected for its implementation. The system includes a MicroBlaze microprocessor that executes by software the whole speaker verification algorithm. The VFPU, designed from scratch in VHDL, is connected to the microprocessor through the system bus and solves any vector floating-point computation. Square root and division are operations whose design is based on a radix-2 restoring algorithm. The logarithm and exponential functions are developed following a CORDIC algorithm. The rest of operations are implemented by combinational circuits.

The clock frequency used to obtain the experimental results is 40 MHz. Program and data are located in a 2MB SRAM external memory. This memory is connected to both the microprocessor and the VFPU, which have direct access to read and write data. Moreover, other peripherals such as timers, UARTs, input-output ports, etc., are also implemented as part of the embedded system.

TABLE IV
AREA AND MAXIMUM CLOCK FREQUENCY F_{MAX}. PERCENTAGE (%) AGAINST TOTAL NUMBER OF RESOURCES IN THE FPGA

Subsystem	LUT (Lookup table)	FF (Flip-Flops)	CLB Slices	MULT 18x18	F_{max} (MHz)
Microblaze + FPU	3,051 (7%)	1,799 (4%)	1,683 (8%)	7 (17%)	92.2
VFPU	10,611 (25%)	7,522 (18%)	7,649 (37%)	20 (50%)	42.7
Rest of peripherals	2,076 (6%)	1,090 (2%)	1,452 (7%)	0 (0%)	45.4
Embedded System	**15,738 (38%)**	**10,441 (25%)**	**10,784 (52%)**	**27 (67%)**	**42.4**

Table IV shows the resources of the FPGA required for the implementation of the whole system and the maximum clock frequency reported by the synthesis tool.

A. Recognition results

Fig. 8 shows the DET (Detection Error Tradeoff) curve obtained for the BANCA public database for different trials that combine gender, female (F) or male (M), and environmental conditions (controlled (C), adverse (A) and degraded (D)) under which the utterances have been acquired. This curve represents the False Match Rate (FMR, erroneous classification of a genuine user as impostor) versus the False Non-Match Rate (FNMR, erroneous classification of an impostor user as genuine).

The Equal Error Rate (ERR) is defined as the point of the DET curve where FMR and FNMR are equal. This parameter is usually accepted as a measure of quality of a biometric algorithm. As expected, the best ERR (7%) is given for the database with utterances acquired under controlled conditions. In contrast, the worst results, which correspond to utterances

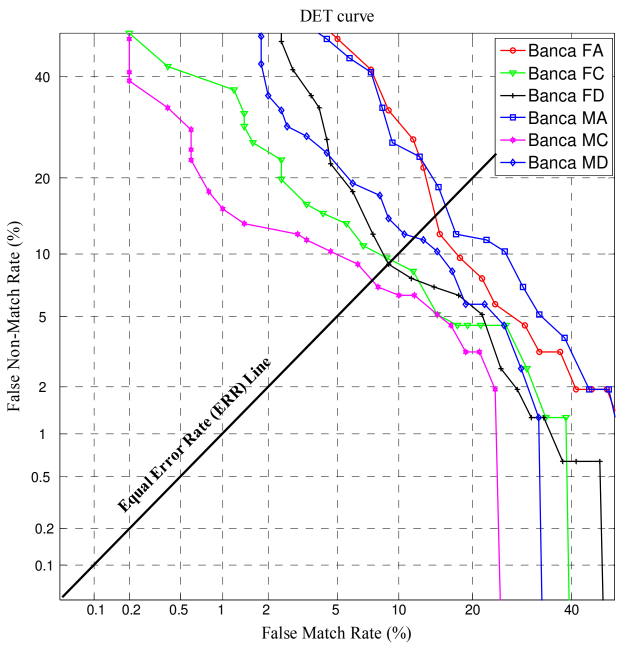

Fig. 8. DET curves for BANCA database (ρ=-0.44, γ=0.4).

obtained in adverse and degraded conditions, present a ERR ranged between 15% and 17%, respectively.

B. Speed Processing

In order to compare the performance of the proposed VFPU in terms of speed, the speaker verification algorithm was executed on two additional systems: an ARM Cortex-A8 microprocessor clocked at 720 MHz and the MicroBlaze microprocessor configured with its own FPU designed by Xilinx. The results for ARM are given for two different implementations. In the first one, a standard compilation was performed over a code based on Single Instruction Single Data (SISD) operations. In the second one, the code was rewritten including NEON instructions, which were programmed by means of intrinsics to increase performance by using vectorization [29]. The latter implementation is usually faster, since NEON performs Single Instruction Multiple Data (SIMD) processing on several floating-point lanes. Such dedicated instructions are used to load (store) vector data between the registers and the external memory. Note that with these four implementations, it is easy to compare the performance obtained by non-vectorized (µBlaze+FPU and ARM-SISD) versus vectorized implementations (µBlaze+VFPU and ARM-SIMD).

Table V shows the execution time for such four implementations including the feature extraction and matching stages. The table also presents the specific execution time for each function described in Table I. Such results are also presented in clock cycles, so that they can be particularized for the operating frequency of a faster FPGA featured with a higher degree of speed.

As mentioned earlier, a complete frame processing should be carried out in less than 10 ms (frame advance). Only the embedded system designed with the VFPU (9.1 ms) and the ARM-SIMD execution using NEON instructions (5.71 ms) are capable of executing the whole speaker verification algorithm in time. However, it is important to point out that ARM achieves this result using a clock frequency 18 times higher than the proposed VFPU. Therefore, if both systems use the same frequency, our proposal would be about 11.29 times faster than the ARM-SIMD execution. Furthermore, it is noteworthy that the NEON architecture consists of four lanes, unlike the actual VFPU implementation that is performed including only one lane. If the VFPU was built with 2 lanes, applying (9) the frame matching stage would be processed in 4.83 ms with a clock of 40 MHz, which is faster than the ARM-SIMD implementation that takes 5.55 ms (fifth column of Table V).

When configuring MicroBlaze with the scalar FPU supplied by Xilinx, the execution time is approximately 140 ms, which compared with our proposal leads to an average acceleration (fourth column of Table V) of x15.4. The acceleration provided for each stage of the feature extraction and matching stage is different. As shown in the example of section III.B, such acceleration mainly depends on the degree of compliance of the following five factors:

a) Use of exponential or logarithm functions. These operations are solved by a specific block included in the VFPU, which provides a faster resolution when compared with the standard math library.

b) Use of function level pipelining techniques for solving in parallel several operations. This technique can be applied since the VFPU is designed with two control units.

c) Balance between the number of vectorizable and non-vectorizable computations involved in a specific code.

d) Loop unrolling to process groups of up to four operations at the same time.

e) Usually, the configuration time of the VFPU is considered negligible when compared with the computational time. However, when short-vectors are processed this

TABLE V
DEGREE OF COMPLIANCE FOR: A) USE OF EXP. OR LOG. FUNCTIONS, B) FUNCTION LEVEL PIPELINING, C) VECTORIZABLE CODE, D) LOOP UNROLLING AND E) NEGLIGIBLE CONFIGURATION TIME.
EXECUTION TIME, IN TERMS OF NUMBER OF CLOCK CYCLES T_{CLK} AND ms, FOR EACH STAGE OF THE SPEAKER VERIFICATION ALGORITHM.

Stage	Degree of compliance for a), b), c), d) and e): L=low, M=medium, H=high	µBlaze + VFPU (40 MHz)	µBlaze + FPU of Xilinx (40 MHz)	Acceleration µBlaze+VFPU vs µBlaze+FPU	ARM-SIMD (NEON) Cortex A8 (720 MHz)	ARM-SISD Cortex A8 (720 MHz)	Acceleration ARM-SIMD vs ARM-SISD
(1) Average value, normalization, energy and emphasis	*L, L, H, H, M*	*1,454·T_{CLK} (36.35 µs)*	*9,570·T_{CLK} (239.25 µs)*	*6.58*	*2,967·T_{CLK} (4.12 µs)*	*14,248·T_{CLK} (19.79 µs)*	*4.80*
(2) Filtering by Hamming window	*L, L, H, H, M*	*1,382·T_{CLK} (34.55 µs)*	*10,386·T_{CLK} (259.65 µs)*	*7.52*	*2,512·T_{CLK} (3.49 µs)*	*13,665·T_{CLK} (18.98 µs)*	*5.44*
(3) Zero padding and FFT	*L, L, M, H, M*	*36,562·T_{CLK} (914.05 µs)*	*142,898·T_{CLK} (3,572.45 µs)*	*3.91*	*79,453·T_{CLK} (110.35 µs)*	*171,403·T_{CLK} (238.06 µs)*	*2.16*
(4) Filter Channels (Mel filters)	*L, M, M, H, M*	*10,641·T_{CLK} (266.03 µs)*	*52,769·T_{CLK} (1,319.23 µs)*	*4.96*	*11,821·T_{CLK} (16.42 µs)*	*21,952·T_{CLK} (30.49 µs)*	*1.86*
(5) Logarithm and absolute value	*H, H, H, H, H*	*895·T_{CLK} (22.38 µs)*	*18,283·T_{CLK} (457.08 µs)*	*20.43*	*12.791·T_{CLK} (17.77 µs)*	*13,622·T_{CLK} (18.92 µs)*	*1.06*
(6) Discrete cosine transform	*L, L, H, H, M*	*1,079·T_{CLK} (26.98 µs)*	*9,666·T_{CLK} (241.65 µs)*	*8.96*	*3,538·T_{CLK} (4.91 µs)*	*17,632·T_{CLK} (24.49 µs)*	*4.98*
(7) Delta coefficients	*L, L, H, L, L*	*510·T_{CLK} (12.75 µs)*	*1,020·T_{CLK} (25.50 µs)*	*2.0*	*1,776·T_{CLK} (2.47 µs)*	*2,219·T_{CLK} (3.08 µs)*	*1.25*
Frame Extraction *(1)+(2)+(3)+(4)+(5)+(6)+(7)*		52,523·T_{CLK} (1.31 ms)	244,592·T_{CLK} (6.11 ms)	4.66	114,858·T_{CLK} (159.53 µs)	254,741·T_{CLK} (353.81 µs)	2.22
Frame Matching	*H, H, H, H, H*	311,594·T_{CLK} (7.79 ms)	5,375,773·T_{CLK} (134.39 ms)	17.25	3,994,088·T_{CLK} (5.55 ms)	7,137,820·T_{CLK} (9.91 ms)	1.79
Processing of a complete frame		**3.64·10^5·T_{CLK} (9.10 ms)**	**56.20·10^5·T_{CLK} (140.51 ms)**	**15.44**	**41.09·10^5·T_{CLK} (5.71 ms)**	**73.93·10^5·T_{CLK} (10.27 ms)**	**1.80**

simplification could be false. In such cases, the code is hardly accelerated because configuration and computation time have similar values.

Table V shows the degree of compliance (low, medium or high) of these five factors for each stage involved in the whole algorithm. Note that those stages with higher degrees of compliance provide higher acceleration factors. For instance, the frame matching and the logarithm stage are accelerated by x17.25 and x22.43, respectively, since they meet all factors mentioned before. In contrast, the stage devoted to calculate the delta coefficients is only accelerated by x2, since although its code is vectorizable, the configuration time of the VFPU is not negligible, pipelining techniques cannot be applied and exponential functions are not utilized. In general, stages included in the frame extraction step provide lower accelerations, since they only meet some of the five factors described previously.

In [18], the authors presented a custom-hardware implementation of a speaker verification system. Extrapolating their results for a frequency of 40 MHz, a frame would be processed in 5.8 ms. Other similar proposals are presented in [30] and [31] leading to different results. However, if they are compared with the VFPU implementation there are some drawbacks that should be pointed out:

- The hardware design presented in [18] is based on fixed-point arithmetic. Authors achieved high accurate results using a variable word length, whose dimension is adjusted to obtain similar results as those produced in floating-point arithmetic. Thus, any change in the feature vector (adding for example the second derivative of the MFCC coefficients), or in the number of bits used in the codification of the input samples, involves redesigning the overall system. Due to the flexibility of our implementation any of these changes only require a simple modification that should be performed on the software program. Such flexibility in not offered by any of the custom-hardware designs presented in [18], [30] or [31]. Additionally, the design described in [18] is performed using specific tools (Xilinx Core Generator) that are only valid for a particular FPGA vendor. In contrast, as mentioned before, the architecture of the VFPU is generic, so that it can easily be implemented in any FPGA.
- Inherently, floating-point computations have associated a large dynamic range, which is especially important when processing extremely large data sets or data sets where the range may be unpredictable. This characteristic is very suited when dealing with intensive computations such as the kernel function used by the SVM classifier.
- The software code is usually written using float-type variables, since by default many functions (logarithm, exponential, trigonometric, square root etc.) are defined and implemented in floating-point arithmetic. Thus, such arithmetic could be coded directly into hardware operations represented in this format. However, fixed-point arithmetic requires an additional effort, since the original program should be transformed. Further, when performing operations in fixed-point arithmetic there is a risk of producing an overflow, underflow or round-off error. Particularly, this could happen if the database or the size of the input samples change.

Moreover, expression (8) is quite consistent with the results shown in Table V. Note that since $\gamma \neq -1$ and $P_j \neq 1$, then N_{UC2} is equal to 16 (eight new multiplications are added). This theoretical expression, calculated with a particular frequency of f_{CLK}=40MHz, leads to an execution time of about 7.75 ms, which represents an approximated error of 0.5% against the real value of 7.79 ms. Such error is mainly due to the communication delays produced by the configuration time of the VFPU.

V. Conclusion

This paper presented the design and implementation on FPGA of a complete biometric algorithm for speaker verification. The feature extraction stage is based on the calculation of the Mel-Frequency Cepstrum Coefficients, whereas the classification is performed by means of a SVM model. The paper also describes a generic architecture of VFPU that solves all the vector floating-point computations involved in the algorithm. Additionally, the architecture provides a high flexibility, which allows quickly adapting the parameters of the algorithm to different conditions related with the acquisition of samples or the particular features of a group of users. Its design includes two control units that maximize the throughput and allow the entire algorithm to be solved in real time. The performance of the VFPU was compared with two systems of similar features: the FPU provided by Xilinx and the ARM Cortex A8 microprocessor. Experimental results show as each frame is processed by the VFPU in $3.64 \cdot 10^5$ clock cycles, which represents an acceleration factor of x11.2 and x15.4 when compared with systems based on an ARM-NEON microprocessor and the FPU of Xilinx, respectively.

REFERENCES

[1] J.P. Campbell Jr, “Speaker recognition: A tutorial,” Proceedings of the IEEE, vol. 85, no. 9, pp. 1437–1462, 1997.

[2] D.A. Reynolds, “An overview of automatic speaker recognition technology,” IEEE International Conference in Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 4072-4075, 2002.

[3] Steven B. Davis and Paul Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” IEEE Transactions on Acoustics Speech, and Signal Processing, vol. ASSP-28, No 4, August 1980.

[4] Jia Lei and Xu Bo, “Including detailed information feature in MFCC for large vocabulary continuous speech recognition,” IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’02). Vol. 1, pp. I805-I808, 2002.

[5] Childers, D. G., Skinner, D. P.: The Cepstrum: A Guide to Processing. Proceedings of the IEEE, Vol. 65, no. 10, pp. 1428-1443, October 1977.

[6] Munteanu, D.-P.; Toma, S.-A. “Automatic speaker verification experiments using HMM,” 8th International Conference on Communications (COMM), pp. 107-110, 2010.

[7] Reynolds, D.A.; Rose, R.C., “Robust text-independent speaker identification using Gaussian mixture speaker models,” IEEE Transactions on Speech and Audio Processing, Vol. 3, Issue: 1, pp. 72-3, 1995.

[8] A. Ganapathiraju, J. E. Hamker, and J. Picone, “Applications of support vector machines to speech recognition,” IEEE Transactions on Signal Processing, vol. 52, no. 8, pp. 2348-2355, Aug. 2004.

[9] Vincent Wan and Steve Renals, “Speaker Verification Using Sequence Discriminant Support Vector Machines,” IEEE Transactions on Speech and Audio Processing, Vol. 13, Issue: 2, pp. 203 – 210, March 2005.

[10] Burges, C.J.C., “A Tutorial on Support Vector Machines for Pattern Recognition,” Kluwer Academic Publishers, Data Mining and Knowledge Discovery, vol. 2, pp. 121-167, 1998.

[11] Wan, V., Campbell W. M., “Support Vector Machines for Speaker Verification and Identification,” Proceedings of the 2000 IEEE Signal Processing Society Workshop Neural Networks for Signal Processing X. vol.2, pp. 775-784, 2000.

[12] F. Fons, M. Fons and E. Cantó, "Fingerprint Image Processing Acceleration Through Run-Time Reconfiguration Hardware", IEEE Transactions on Circuits and Systems II, Vol. 57, Issue 12, December 2010.

[13] M. López, J. Daugman and E. Cantó, "Hardware-Software Co-design of an iris recognition algorithm", IET Information and Security, Vol. 5, Issue 1, pp. 60-68, April 2011.

[14] Choi, W-Y., Ahn, D., Burn Pan, S., Chung, K., Chung, Y., Chung, S-H. “SVM-Based Speaker Verification System for Match-on-Card and its Hardware Implementation”, ETRI Journal, vol. 28, no. 3, pp. 320-328, June 2006.

[15] Manikandan, J.; Venkataramani, B.; Avanthi, V., “FPGA Implementation of Support Vector Machine Based Isolated Digit Recognition System,” 22nd International Conference on VLSI Design, pp. 347-352, 2009.

[16] Ngoc-Vinh Vu; Whittington, J.; Hua Ye; Devlin, J., “Implementation of the MFCC front-end for low-cost speech recognition systems,” IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2334-2337, 2010.

[17] Phaklen EhKan, Timothy Allen, and Steven F. Quigley, “FPGA Implementation for GMM-Based Speaker Identification,” International Journal of Reconfigurable Computing, Vol. 2011, 2011.

[18] R. Ramos-Lara, M. López-García, E. Cantó-Navarro and L. Puente-Rodriguez, “Real-Time speaker verification system implemented on reconfigurable hardware,” Journal of Signal Processing Systems, vol. 71, no. 2, pp. 89-103, May 2013.

[19] P. Karlström, A. Ehliar & D. Liu, “High-performance, low-latency field-programmable gate array-based floating-point adder and multiplier units in a Virtex 4,” IET Comput. Digit. Tech., 2008, Vol. 2, nº 4, pp. 305-313, Jul. 2008.

[20] Y. Chong & S. Parameswaran, "Configurable Multimode Embedded Floating-Point Units for FPGAs," IEEE Transactions on Very Large Scale Integration Systems, vol.19, no.11, pp.2033-2044, Nov. 2011.

[21] S. Chen, R. Venkatesan & P. Gillard, “Implementation of Vector Floating-Point Processing Unit on FPGAs for High Performance Computing,” Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 881-886, May 2008.

[22] J. Kathiara & M. Leeser, "An Autonomous Vector/Scalar Floating Point Coprocessor for FPGAs," IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.33-36, May 2011.

[23] Ming Ji, T. J. Hazen, J. R. Glass and D. A. Reynolds, “Robust Speaker Recognition in Noisy Conditions,” IEEE Transactions on Audio, Speech and Language Processing, Vol. 15, Issue 5, pp. 1711-1723, 2007.

[24] Yuri Matveev, “The Problem of Voice Template Aging in Speaker Recognition Systems,” Speech and Computer, Lecture Notes in Computer Science, Vol. 8113, pp. 345-353, 2013.

[25] L. Ferrer, E. Shriberg, S. Kajarekar and K Sonmez, “Parameterization of Prosodic Feature Distributions for SVM Modeling in Speaker Recognition,” International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. IV-233 – IV-236, 2007.

[26] Samy Bengio, Frederic Bimbot, Miroslav Hamouz, Johnny Mariethoz, Jiri Matas, Kieron Messer, Fabienne Poree, Belen Ruiz - The BANCA Database and Evaluation Protocol, Lecture Notes in Computer Science Volume: 2688, Publisher: Springer, Pages: 625-638, 2003

[27] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[28] http://www.torch.ch/introduction.php

[29] ARM public limited company, “Introducing Neon, Development article”, 2009. http://infocenter.arm.com/help/index.jsp

[30] J. Manikandan, B. Venkataramani and V. Avanthi, “FPGA implementation of Support Vector Machine based on Insolated Digit Recognition System,” 22nd International Conference on VLSI Design, pp. 347-352, 2009.

[31] Mohit Shah, Lifeng Miao, Chaitali Chakrabarti, and Andreas Spanias, “A speech emotion recognition framework based on latent dirichlet allocation: algorithm and FPGA implementation,” IEEE international Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2553-2557, 2013.