Analysis of myoelectric signals using a Field Programmable SoC

Bence J. Borbély*, Zoltán Kincses†, Zsolt Vörösházi‡, Zoltán Nagy*§, Péter Szolgay*§
*Faculty of Information Technology, Pázmány Péter Catholic University, Budapest, H-1083
†Institute of Informatics, University of Szeged, Szeged, H-6701
‡Dept. of Electrical Engineering and Information Systems, University of Pannonia, Veszprém, H-8200
§Cellular Sensory and Optical Wave Computing Laboratory, Hungarian Academy of Sciences, Budapest, H-1111

***Abstract*—A platform design for the analysis of human myoelectric signals (MES) is presented. Offline recorded multi-channel signals of forearm muscles are processed with a Field Programmable SoC in order to classify different movement patterns to control human-assisting electromechanical systems with multiple degrees of freedom (e.g. a prosthetic hand). Benchmark results of an ANSI C implementation are shown to assess the raw performance of the built-in ARM cores of the SoC. Possible computational bottlenecks are located based on the results and custom hardware implementations are shown to fully utilize the flexibility and performance of the used hardware platform.**

I. Introduction

The non-invasive measurement and analysis of human bioelectric signals has been an emerging field in the last decades. Electric signals measured at different skin surface locations have different characteristics. They can carry important features of an individual's current state of health via heart monitoring (electrocardiogram, ECG), they can drive Brain-Computer Interfaces if measured from the surface of the head (electroencephalogram, EEG) or can tell specific movement intents of patients with limb amputations if measured from the covering skin of residual muscles (myoelectric signal, MES), especially in the case of upper limb amputations.

In this study we focus on the processing and classification of MES data to utilize the flexibility and performance of a Field Programmable platform in a test environment for embedded prosthesis control. As a prototype system a widely recognized pattern recognition scheme was implemented to process four to eight forearm MES channels using time-domain signal features and an LDA classifier.

II. Methods

A. The pattern recognition method

The idea behind the standard pattern recognition based myoelectric control is to measure signals from multiple channels during different predefined isometric contractions of muscles (different states) and store specific features of these recordings as separate state descriptors (offline, supervised learning). After this stage an online stream of data from the same recording sites can be obtained and classified to categorize the actual signals into one of the trained classes. MES data is non-stationary and stochastic in nature therefore most of the related analyses apply processing windows to extract descriptive features of the signal. In the current implementation a 150 ms long processing window was used because it has been shown that this length enables optimal performance for this type of classifiers [1].

The spatial selectivity (the number of separable movement classes) in the system is highly determined by the number of separate recording channels. Previous studies justified that in the case of lower arm recordings four channels of MES are suitable to classify online measured data into one of six separate classes with high efficiency [2]. Based on these results we implemented a four-channel system as the basis of the test environment, but for testing reasons we extended it to have five, six and eight virtual channels to estimate performance in more complex recording environments. Because we had only four real channel recordings, six possible output classes were used in every case.

In real prosthetic applications overall latency and response time are critical factors of device acceptance which are determined by the processing window length and the amount of processing window shift (or sampling delay) during operation. Among these two factors window shift value can be varied to obtain different temporal resolutions, resulting that shorter shifts yield better response times at the cost of computational overhead.

1) Signal features: To characterize signal windows and to reduce data dimension the standard four element time-domain feature (TDF) set was calculated for each data window (150 ms) and channel in the performed simulations. These features were the mean absolute value (MAV), number of zero crossings (NZC), number of slope sign changes (NSSC) and the waveform length (WL) as described in previous studies [3], [4]. It is important to note that these features give only estimations of specific signal properties (e.g. NZC $\sim$ frequency) but it has been shown that they provide as good basis as frequency-domain features for classification of stationary signals for less computational cost and induce lower latency in the system [2].

2) LDA classifier: To partition the feature space into six subspaces (or classes) for pattern classification, linear discriminant analysis (LDA) was applied as described in [5]. The reason for using LDA is that it can reduce feature space dimensionality taking the separate subspaces into account. More specifically it finds those projection vectors in the complete feature space (in this case with dimension of (4 TDF $\times$ num. of channels)) which best separate the individual classes when the dataset is projected. After the projection vectors are calculated (num. of projection vectors $\ll$ num. of feature space dimensions) data points from the complete feature space are projected to get a more separable set of target classes having lower dimension (num. of projection vectors).

During online operation the actual recorded data is first transformed into the feature space (by calculating its time-domain features) followed by the projection to the same vectors obtained with the LDA algorithm. The classification takes place when these projected values are compared to the stored projections of the target classes and class labels are assigned to the data based on its distance (e.g. Euclidean geometry) from the stored class values.

B. Recorded and simulated data

Four channels of MES were recorded (F_s =1 kHz, resolution: 16 bit) from one subject during six different isometric muscle contraction classes following the method described in [6]. The recordings were performed independently from the processing system. The recording electrodes were placed on the forearm above the wrist flexors and extensors and on each side of the forearm, roughly at middle length. Separate data sets were recorded to train and test the classifier (with average length of 25 s for each class). Testing was performed using an appended array of test recordings in pseudo-random order as the input stream. For simulation reasons the measured MES data were extended to five, six and eight virtual channels using a perturbed version of the original recordings.

C. Algorithm

The practical realization of the computational steps as described previously is shown in Algorithm 1. It is important to note that this implementation is used for offline testing with previously measured training and test data, not for online streaming and processing of the input signals. However, the algorithmic design allows the extension of the system to have real-time functionality with only minor modifications.

Algorithm 1 Offline EMG classification

```
1: procedure EMGCLASSMAIN($N_{channels}, WinShift$)
2:     // Calculate and store the time-domain features of the training dataset
3:     PreprocessTrainingData($N_{channels}, WinShift$)

4:     // Calculate and store the LDA projection vectors which best separates the training dataset
5:     TrainLDAClassifier($preprocessedData$)

6:     // Assign class labels to the test signal windows based on the the separated training dataset
7:     ClassifyTestData($inputData, LDAdata$)
8: end procedure
```

The three main parts of the system were developed to allow easy separation of the main processing steps. *PreprocessTrainingData($N_{channels}, WinShift$)* calculates and stores all time-domain features of the training data based on the number of channels and the amount of processing window shift, decreasing the dimensionality at the first place. The second function, *TrainLDAClassifier($preprocessedData$)* calculates and stores the LDA projection vectors, which best separates the training dataset in the time-domain feature space. After these vectors are calculated, the training dataset is projected to reduce its dimension and prepare it for classification. The practical procedure of LDA vector calculation involves covariance and inverse matrix calculations, and determining eigenvalues and eigenvectors of matrices of size $N^2_{channels}$.

The last part, *ClassifyTestData($inputData, LDAdata$)* performs the classification of all input data window using LDA projection vectors calculated during classifier training.

III. Implementation

1) The Zynq-7000 platform: To implement the EMG processing system (Section II. A.) in hardware the Digilent Zedboard [7] was chosen, which is based on a Xilinx Zynq 7020 SoC architecture [8]. The Zynq 7000 family integrates the ARM Cortex-A9 dual core PS (Processing System) and the 28nm Xilinx Series-7 PL (Programmable Logic) fabric. The unique features of this system are the tight integration of the embedded microprocessor and the FPGA using standard AXI4 bus interfaces and the so-called processor centric approach, where the PS is initialized in the first step and the PL is configured in the second step during the startup sequence.

The Programmable Logic, which is based on the Xilinx's Artix FPGA family, is connected to the PS via several AXI4 interconnects; four 32-bit wide interfaces are dedicated to low latency access to the registers of the peripherals implemented in the PL. Four 64-bit wide high performance AXI4 buses are available for fast transfer of large amounts of data between the PL and the different memories. For tightly integrated co-processors, which should share data with the software part running on the PS, a specialized 64-bit wide coherent AXI4 bus connected to the snoop protocol of the L2 cache is also available.

2) ANSI C implementation: The algorithm described in Section II. C. was implemented in ANSI C on a laptop computer having an Intel Core i5-540M CPU running at 2.53 GHz. The extracted time-domain features were $MAV, NZC, NSSC, WL$. Self-written implementations were used for all numerical methods. Inverse matrix calculation was performed based on Gauss-Jordan elimination. Because LDA vector calculation needs only eigenvectors, but not eigenvalues accurately, eigenvalues were only estimated using the QR iteration with limited number of steps. The eigenvectors were then accurately calculated applying the Inverse Iteration to the estimated eigenvalues.

The development system was running Ubuntu Linux 12.04 LTS operating system and the *gcc* compiler was used to generate executables. To compile the source onto the ARM cores of the Zynq processor, gcc's cross compiler version (arm-linux-gnueabi-gcc) was used. For optimal performance the -O3 compiler option was applied in both situations.

3) The implemented architecture on FPGA: The proposed architecture contains five main parts implemented on two different places: on the hard-processor system (PS) fabric, and on the PL (FPGA fabric) of the Zynq SoC. The *ARM Processor Core* and the *Memory Controller* are located in the PS part of the SoC, while the *Vector Processor*, the *Preprocessor* and the *Sensor Interface* are implemented on the PL part of the Zynq SoC. The high-level steps of the algorithm (described in Section II. C.) are executed by the *ARM Processor Core*, while the vector operations are performed by the *Vector Processor*. These vector operations are required in the classification part of the algorithm. The *Preprocessor* part is responsible for

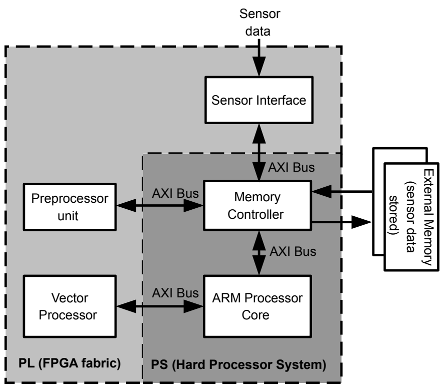

Fig. 1. The overall system implemented on the Xilinx Zynq AP SoC

the calculation of the time-domain properties. The sensors attached to the system are connected through the *Sensor Interface*. Incoming data received from these sensors are stored in the external DDR3 memory. The control of this external memory and the load / store operations are handled by the *Memory Controller*. All of the units communicate to each other using AXI-4 Interconnect. The block diagram of the proposed architecture is depicted in Figure 1.

4) The Preprocessor unit: Time-domain properties of the measured signal is computed by using specialized processing elements. Using these properties the actual EMG interval can be characterized by less data than using the whole data window (as described previously in Section II. A.). This unit contains eight main components that utilize data of the same processing window. The components of the Preprocessor unit are:

The *MAV* (Mean-of-Absolute Value) *Unit* calculates the average of the summed-absolute values inside a processing window.

The *MAVS* (Mean of Absolute Value Slope) *Unit* utilizes the former results carried out by the *MAV Unit*, and it will simply make the difference of the average of absolute values on each successive window. Therefore this value can be calculated as a difference of two consecutive *MAV* values [3]

The *RMS* (Root Mean Square) *Unit* is responsible for computing the average (mean) of squared data which are located within a processing window.

The *WAMP Unit* is responsible for calculating the Willison amplitude. This component counts all amplitude changes of incoming signals within a processing window, which are higher a given threshold level [9].

The *NZC* (Number of Zero Crossings) *Unit* determines all possible zero-crossings on an incoming signal, at these points the difference between the values with opposite signs are larger than a pre-defined threshold. In these cases, the threshold is necessary to eliminate false zero-crossings, which may arise from environmental noise [4].

The *NSSC* (Number of Slope Sign Changes) *Unit* will determine the number of direction of changes, in which cases the first or the last changes among three consecutive values are larger than a predefined limit [4]. This limiting factor comes from the filtering out of external environmental noise.

The *VAR* (Variance) *Unit* gives the variance within a processing window, which is considered to be proportional to the force produced by the muscle [9].

Finally, the role of the *WL* (Waveform Length) *Unit* is to calculate the length of the waveform, which is a characteristic feature of signal complexity [3]. As it can be seen, more time-domain feature are calculated in the Preprocessor Unit than in the C implementation. The reason for this is the higher computational efficiency of the custom PL implementation compared to the ARM cores and to allow better flexibility of the system for later testing conditions. The implementation and simulation process of Preprocessor unit was completed in Xilinx ISE Design Suite 14.2 [8]. Both the general (6-input LUTs, D Flip-Flops, Slices) and the dedicated (DSP48 multiplier slices, 36Kbit Block RAMs) resource requirements, and the maximal reachable clock frequency of the implemented Preprocessor unit were investigated. Moreover, it has been examined how the number of parallel analysis windows changes if the size of sampling windows increases.

The resource utilization (Table I) shows that increasing the size of sampling windows will only moderately increase the resource requirements of the Preprocessor unit. Because a sampling window typically contains N samples (between 50 and 250 elements) in practice, the general and dedicated resource utilization of the Preprocessor unit was investigated between 50 and 300 samples, where sample number increased by 50 in each step. The available resources and the device utilizations for Xilinx Zynq 7020 AP SoC are summarized below on Table I.

The maximum reachable operating frequency ($\sim$ 140 MHz) was also measured by using the Static Timing Analyzer tool in the Xilinx ISE Design Suite. This means that the Preprocessor unit is capable of processing one sample in each analyzing window within 7.28 ns. The incoming sensor data are sampled at 1 KHz for each channel, therefore a sample should be stored in the external memory in every 1 ms. Processing of each analysis window requires at the maximum of 2 x N clock cycles, because the MAVS Unit computes the MAV value from the actual and the previous processing windows. The number of real-time processable channels are in the range of 1389 and 231 depending on the sampling period which is usually in the range of 50 ms to 300 ms.

5) The Vector Processor: As described previously in the proposed EMG processing system, a substantial part of the algorithm is the classification phase, where double precision floating-point vector-, and matrix operations are required. Unfortunately, in one hand, the built-in Neon SIMD engine in the ARM Cortex-A9 Core does not support double precision vector floating-point operations. On the other hand, scalar floating-point computing performance is not high enough to perform the required operators at acceptable speed. The Vector Processor [10] can be built-up from a scratch-pad memory, several vector registers, and a floating-point adder and multiplier (because the majority of the required operations are multiplications and additions). The matrices are stored in the scratch-pad memory, where the high-speed memory access by the ARM Processor Core is critical. The Vector Processor is capable of computing simple addition, multiplication and multiply-addition operations. Moreover, an addition and a multiplication operation can be computed in parallel when separate result registers are used. The length of the vectors is limited by the depth of the vector registers, and they can be configured on-the-fly to adapt to the requirements of the classification algorithm. The schematic block diagram of the Vector Processor can be seen in Figure 2.

TABLE I. THE GENERAL AND DEDICATED RESOURCE REQUIREMENTS OF THE PREPROCESSOR UNIT

N (samples)	General Resource Requirement			Dedicated Resource Requirement	
	6-input LUTs (max: 53200)	*Flip-Flops (max: 106400)*	*Slices (max: 13300)*	*DSP48 Slice (max: 220)*	*36Kb BlockRAM (max: 140)*
50	1120 / 5320	1477	399	19	0
100	1132	1495	368	19	0
150	1161	1548	371	21	0
200	1149	1547	377	21	0
250	1175	1548	384	21	0
300	1163	1566	376	21	0

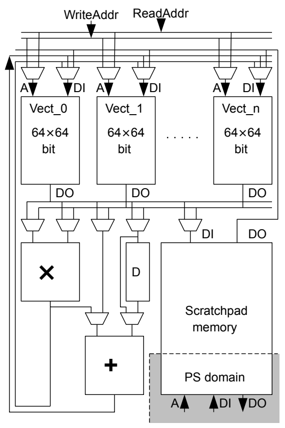

Fig. 2. The Vector Processor unit

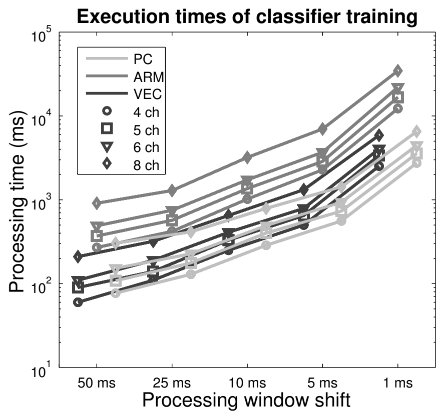

Fig. 3. Off-line runtime results of the C implementation

IV. RESULTS AND CONCLUSION

The system using the proposed hardware blocks (*Preprocessor unit* and *Vector Processor unit*) were implemented. Tests with various channel numbers and processing window shift values were performed on the development PC described in Section III. 2. and on the Zedboard itself. In addition, computing times were estimated on a system where all vector operations would be performed with the described *Vector Processor*. 20 different test conditions were analyzed using 4 different channel numbers (4, 5, 6, 8) and 5 different processing window shift values (50 ms, 25 ms, 10 ms, 5 ms, 1 ms). Computing times on the three different platforms are summarized in Figure 3. The results show that the processing times increase exponentially when window shift value and the number of analyzed channels are increased. In addition, approximately one order of magnitude speed-up can be reached on the ARM processors based on the running times. It is promising that using only the *Vector Processor unit* the Zynq platform is capable for the same performance as the Intel Core i5 processor, consuming one order of magnitude less power (the measured power consumption of the whole Zedboard was $\sim$4 W during operation).

The final system that uses both the ARM cores and custom hardware elements implemented in the FPGA fabric provides real-time measurement and processing of bio-signals in a high performance embedded environment.

ACKNOWLEDGMENT

This research project was supported by the OTKA Grant No. K84267. The publication and research has been supported by the European Union and Hungary and co-financed by the European Social Fund through the project TÁMOP-4.2.1.B-11/2/KMR-2011-0002 and TÁMOP-4.2.2.C-11/1/KONV-2012-0004.

REFERENCES

[1] L. H. Smith, L. J. Hargrove, B. a. Lock, and T. a. Kuiken, "Determining the optimal window length for pattern recognition-based myoelectric control: balancing the competing effects of classification error and controller delay." *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 19, no. 2, pp. 186–92, Apr. 2011.

[2] L. J. Hargrove, K. Englehart, and B. Hudgins, "A comparison of surface and intramuscular myoelectric signal classification." *IEEE Trans. Biomed. Eng.*, vol. 54, no. 5, pp. 847–53, May 2007.

[3] B. Hudgins, P. Parker, and R. N. Scott, "A new strategy for multifunction myoelectric control." *IEEE Trans. Biomed. Eng.*, vol. 40, no. 1, pp. 82–94, Jan. 1993.

[4] K. Englehart and B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control." *IEEE Trans. Biomed. Eng.*, vol. 50, no. 7, pp. 848–54, Jul. 2003.

[5] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[6] B. Borbély, "Design and simulation of a processing system for myoelectric data," in Hungarian, Pázmány Péter Catholic University, 2012.

[7] "Digilent Zedboard (official webpage - 2013)." [Online]. Available: http://www.zedboard.org

[8] "Xilinx official webpage (2013)." [Online]. Available: http://www.xilinx.com

[9] M. Zecca, S. Micera, M. C. Carrozza, and P. Dario, "Control of multifunctional prosthetic hands by processing the electromyographic signal." *Critical reviews in biomedical engineering*, vol. 30, no. 4-6, pp. 459–85, Jan. 2002. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/12739757

[10] Z. Nagy, A. Kiss, A. Zarándy, T. Zsedrovits, B. Vanek, and T. Péni, "Volume and power optimized high-performance system for UAV collision avoidance," in *IEEE International Symposium on Circuit and Systems, ISCAS 2012*, Seoul, 2012.