ZYNQ-based HD EMG Prosthesis Controller

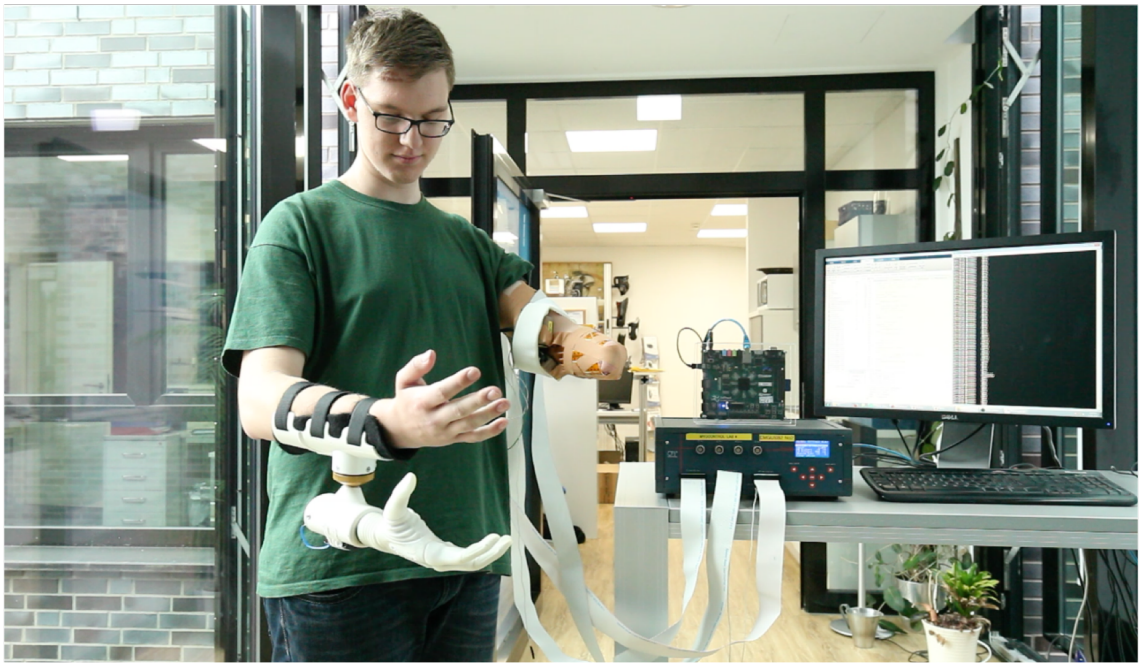
Georg Thombansen, Linus Witschen
thombang@mail.uni-paderborn.de, witschen@mail.uni-paderborn.de

Supervisor: Alexander Boschmann
alexander.boschmann@uni-paderborn.de

Team: XIL-54180

Computer Engineering Group
Paderborn University

June 30, 2016



https://youtu.be/RZGHkCOCRC4

Contents

1 Introduction

Upper-limp prostheses allow amputees to perform activities of daily living and regain a considerable amount of quality of life. To control active prostheses, electromyography (EMG), i.e. potentials of muscular activity, is used. The EMG signals are acquired from the skin surface and processed by a controller that uses a pattern recognition-based classifier to predict the amputee's intended movement. The output of the controller is send to a prosthesis which then performs the movement.

For commercially available prosthesis control, up to 4 EMG sensors are used. However, research has shown, that more sensors can improve the robustness of the control algorithm [3]. Modern EMG amplifiers can acquire up to 256 sensors [7], which can not be processed by a micro-controller in real-time as it would be necessary for an embedded prosthesis controller. A ZYNQ-based system that meets real-time requirements was developed earlier in our group [2] and published at ReConFig 2015. In this competition entry, we extended this system to work with real online EMG signals and a commercial hand prosthesis used by an amputee.

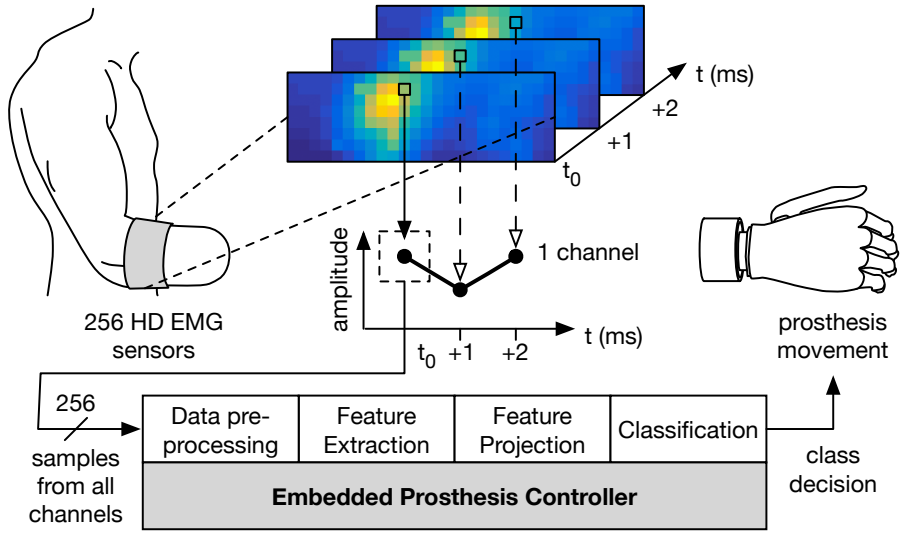


Figure 1: ZYNQ-based prosthesis controller: HD EMG input signals, four processing steps on ZYNQ System-on-Chip. Class decision as controller output.

The system consisting of the DAQ device, signal processing chain in the controller and prosthesis can be seen in Figure 1. The signal processing chain consists mainly of four steps: data pre-processing, feature extraction, feature projection and classification. In the first step, the data pre-processing, EMG signals are acquired from a DAQ device and stored in data windows. These data windows, that hold a history of EMG signals, are used to extract characteristic features for the later classification step. First, however, the dimensionality of data is reduced by projecting the data into a dimension space that has a lesser dimension than the original space. The dimensionality-reduced data are then used by the pattern recognition-based classifier to predict a movement. The predicted

movement is then passed on to a commercial prosthesis that performs the output of the controller. The time from acquisition of EMG signals to a classified movement decision is often defined as controller delay [5]. Figure 2 and Equation 1 show how the controller delay D corresponds to the processing time T_d. In order to achieve responsive control of artificial hands, the controller delay has to be minimized. In our setup, a minimal controller delay of $D = 79$ ms can be achieved if the processing time for each window is $T_d < 1$ ms.

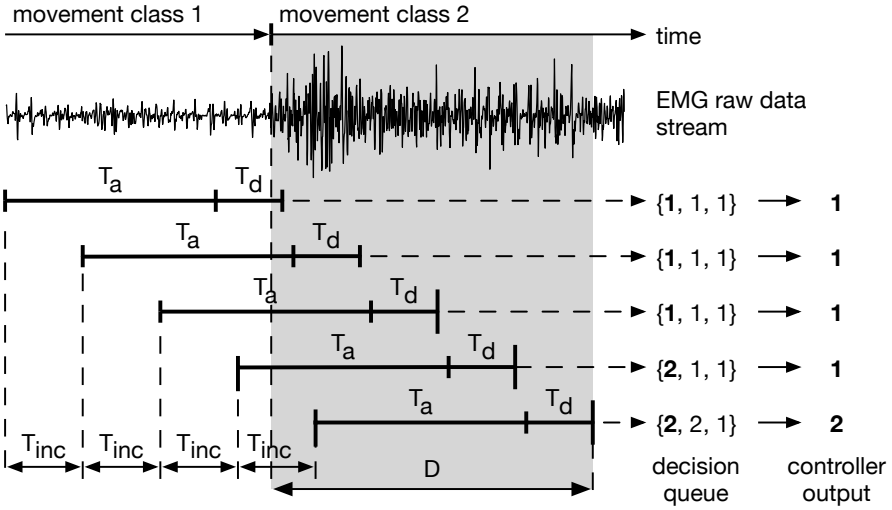


Figure 2: Controller delay D for prosthesis control with overlapping windows of window size T_a and window shift T_{inc}. T_a is the processing time for each window. The controller output is obtained after propagating through a majority vote decision queue with depth n.

$$D = \frac{1}{2} \cdot T_a + \left(\frac{n+1}{2}\right) \cdot T_{inc} + T_d \tag{1}$$

Figure 3 shows the amount of time spent in the steps: feature extraction, feature projection, classification and for communication. The graph indicates that the feature extraction phase grows significantly faster compared to the other steps. In this step, the well-established time-domain feature set [6] is extracted. The set consists of: mean average value (MAV), length of the waveform (WFL), the amount of zero-crossigns (ZC) and slope-sign-changes (SSC). The following formulas show how the features are calculated. The calculations are independent of each other and can be executed in parallel using a hardware accelerator.

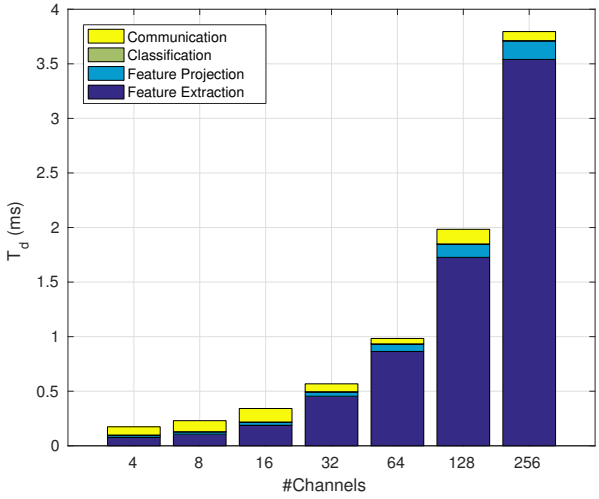


Figure 3: Calculation time of four steps during classification processing in software for varying numbers of up to 256 channels.

$$MAV = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{2}$$

$$WFL = \sum_{n=1}^{N-1} |x_{n+1} - x_n| \tag{3}$$

$$ZC = \sum_{n=1}^{N-1} \Big(sgn\left(x_n \cdot x_{n+1}\right) \cap |x_n| \geq threshold\Big) \tag{4}$$

$$SSC = \sum_{n=2}^{N-1} \Big(sgn\big((x_n - x_{n-1}) \cdot (x_n - x_{n+1})\big)\Big) \tag{5}$$

$$sgn(x) = \left\{ \begin{array}{ll} 1, & \text{if } x \geq threshold \\ 0, & \text{otherwise} \end{array} \right\}$$

2 Design

In this section, we give a short overview of the entire system, before describing the main processing part implemented on ZYNQ SoC in more detail. Figure 4 shows the prototype with all components used in a test setup, where our test subject controls a commercial hand prosthesis with muscular activity measured on his lower left residual limb. Myoelectric potentials are measured with a sensor array of 192 electrodes. These high density (HD EMG) signals are acquired, amplified and filtered by third party hardware and sent to PC via a USB connection. The PC is connected via TCP/IP to the central processing unit,

a Zedboard housing the ZYNQ SoC. Here, three steps are performed: data pre-processing, feature extraction, and pattern recognition-based classification of hand movement classes. Given classes are sent back to the PC, which controls a Otto Bock Michelangelo artificial hand over Bluetooth connection. Important to note is the modularity of the setup. Our ZYNQ-based prosthesis controller only relies on TCP/IP connection to receive data at the input and write back calculated movement classes as controller output. Different myoelectric data sources can be used as well as different devices for visualization of movement classes.

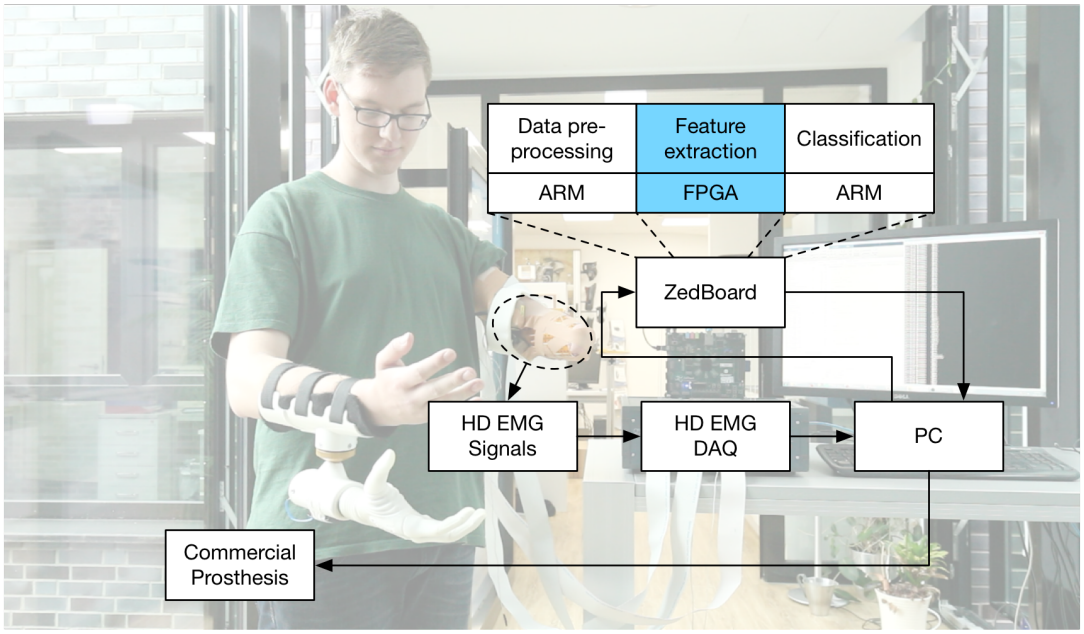


Figure 4: The complete system setup with highlighted components.

Now, we describe the main processing unit in more detail. The system architecture utilizes both, the ARM processor as well as the FPGA of the ZYNQ SoC. The ARM processor runs Linux with ReconOS as a operating system layer [1]. ReconOS is developed at University of Paderborn. It helps designing for heterogenous architectures such as the ZYNQ by providing interfaces for the communication between the ARM processor, the FPGA and memory, which makes it suitable for our application. ReconOS provides and manages the interfaces and offers simple inter-thread communication via message boxes. Following the ReconOS design flow, the system was first implemented entirely in software. This system was then profiled and the most time-consuming parts were then accelerated in hardware using Vivado HLS. In the following sections, the hardware and software parts of the developed architecture are described.

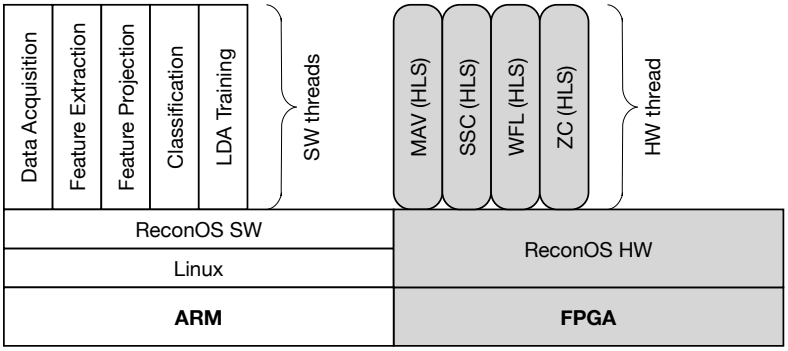


Figure 5: ZYNQ architecture: ARM processor with Linux and ReconOS operating system layer and its software threads. FPGA with ReconOS hardware modules and its feature extraction hardware thread with HLS accelerated feature extractors.

2.1 Hardware

In our application, feature extraction was identified as the most time-consuming step, as depicted in Figure 3. It was accelerated through an FPGA core generated using Vivado HLS. As the extracted features are independent of each other, multiple feature extractions can be processed in parallel to achieve speed-ups. For our prototype, four well-established time-domain feature extractors were implemented. However, the modularity of the design allows the implementation of different feature extractors. All feature extractors are connected to a ReconOS hardware thread with two FIFOs each, one to receive input samples and one to send back calculated features. Figure 5 shows how the feature extractors are connected through a feature extraction manager to the ReconOS hardware thread. The figure also depicts ReconOS and its memory and communication interfaces.

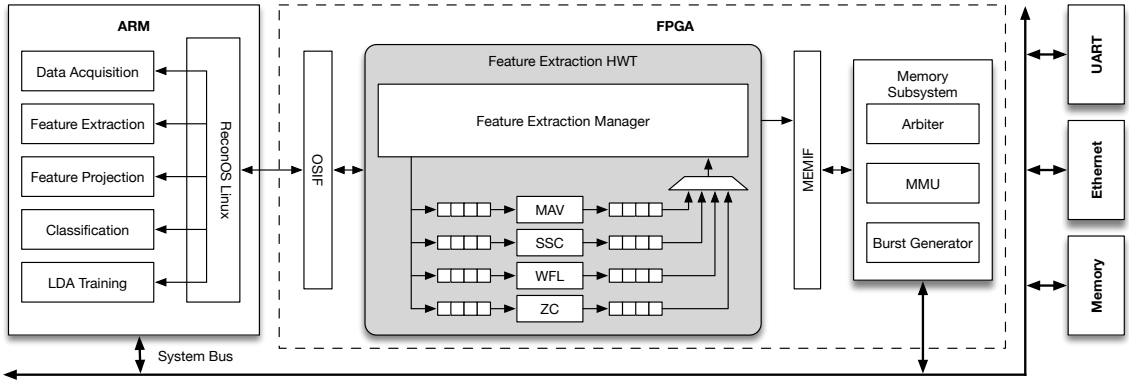


Figure 6: System archtiecture with ReconOS, software threads on ARM and accelerated HLS hardware threads on FPGA.

The extractors themselves were accelerated using Vivado HLS. Computation is implemented in C++ and synthesized with Vivado HLS to ready-to-use hardware components described in VHDL. This allows for fast implementation of different feature extractors

without in-depth knowledge of VHDL hardware design. Additionally, the graphical tool offers better ways to validate the functional correctness compared to classic hardware design and gives estimates about execution times. Vivado's *hls::stream* is used to connect HLS processing cores to the ReconOS hardware thread and provides an interface to read from the input FIFO and write to the output FIFO of each feature extraction core. The cores are implemented in such a way that samples are continuously read, whenever available at the input FIFO. One feature is written back, whenever an entire window of samples is processed. As the input data is given to every feature extractor, multiple features can be calculated in parallel.

2.2 Software

The software system on the ZYNQ SoC comprises of the following tasks: data preprocessing, feature projection and classification. Profiling results have shown that processing these steps in software shows sufficient performance. In the initial system, also the feature extraction step was part of the software system. To accelerate the processing, however, the C++ code of the feature extractors was used in Vivado HLS to generate the FPGA cores of the feature extractors.

A MATLAB program running on the communication PC provides an interface to enable the communication between the ZYNQ and the peripherals, i.e. the DAQ device and the prosthesis. In order to test our ZYNQ-based prosthesis controller without access to expensive peripherals, we provide a test client that compiles for a Linux host system. It creates a TCP/IP connection and sends first pre-recorded training, and then test data to the ZYNQ processing system. Refer to the README document, if you want to try the system.

All software components are implemented as software threads in C++ and compiled using the Xilinx *arm-gnueabi-g++* cross compiler.

2.3 Design Reuse

The implemented system design offers a great reusability through its modular concept. Each component in the system can be replaced. The system allows to connect different DAQ devices or prostheses. Furthermore, each step of the signal processing chain can be replaced, i.e. different feature extractors or classification algorithms can easily be implemented. The current system uses a multi-class Linear Discriminant Analysis as described in [4]. This classifier could for example be replaced by a Support Vector Machine.
However, this may lead to different features that have to be extracted. These feature may be complex, which makes the development of an FPGA core for acceleration more difficult and the development process more time consuming. The design can be simplified and the required time can be reduced by using Vivado HLS to generate accelerated FPGA cores.

3 Results

To test the performance of the time critical path of the system, pre-recorded training and test data were sent to the ZYNQ from a Linux host PC over TCP/IP. Figure 7 shows the processing times for classifying unknown sample data to a pre-trained movement class. Calculation times are shown for three different architectures and varying channel counts from 4 to 256 channels. The objective of minimal controller delay of 79 ms and the corresponding maximal calculation time of 1 ms is highlighted. A software-only solution can only classify up to 64 channels in the desired time frame. Replacing feature extractors with HLS generated hardware cores, features can be extracted in parallel, leading to a significant speedup. Now, up to 192 channels can be processed with the desired controller delay. For comparison, feature extractors were also implemented in VHDL by hand. They result in further speedup with the cost of additional design time.

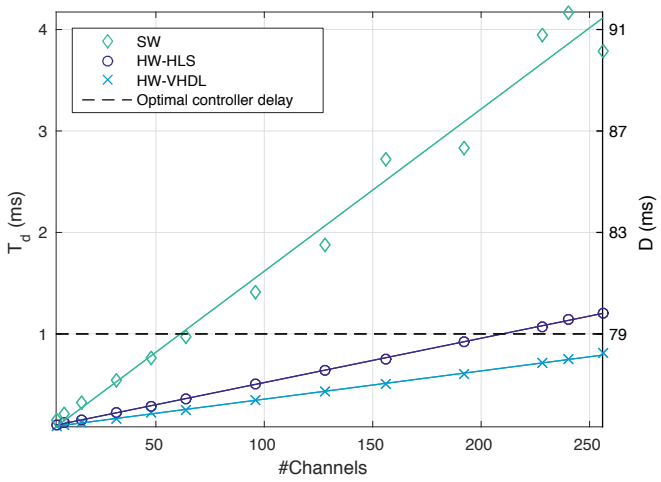


Figure 7: Processing time of pattern recognition-based classification of movement classes for three different architectures is depicted on the left. Software-only, feature extraction accelerated with HLS and hand-written hardware accelerators. The processing times are shown for varying numbers of up to 256 channels. The right axes shows corresponding controller delay. Window size = 150 ms.

A closer look at the feature extraction part of the classification chain is given in Figure 8. On average, the speedup from software to HLS-generated hardware for a single feature extractor is 21 %. The significant speedup for the combined feature extraction of 74 % originates from processing all features in parallel.

As depicted in Figure 4, our system was validated with an amputee test person. After a training session, to train the pattern recognition classifier the subject was asked to perform movements in real time. The controller was able to classify the movements, sent the movement classes back to PC, which controlled a commercial prosthesis. The capability of the controller to be used for daily activities was validated with a cup stacking experiment, where cups were grabbed and placed on top of each other.

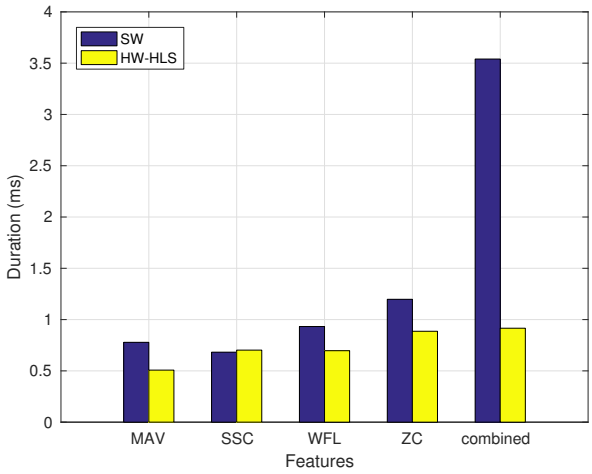


Figure 8: Execution times for feature extraction of 256 channels. Depicted are the times of four different time domain feature extractors alone and all four combined. A software implementation (left) is compared to accelerated hardware with HLS (right). Window size = 150 ms.

4 Conclusion

In this work, we extended a ZYNQ-based HD EMG prosthesis controller. The entire controller including pattern recognition-based training and classification is implemented on a ZYNQ SoC. The most time-consuming parts are accelerated with programmable hardware generated by Vivado HLS. Our results have shown that classification with feature extractors accelerated by Vivado HLS give significant speedups compared to a software-only solution. 192 HD EMG channels can be processed during classification in less than 1 ms to minimize the controller delay. HLS offers synthesis of accelerated FPGA cores in a quick design process. High level code such as C++ is easily adaptable for use with HLS synthesis to generate hardware description. Extending the system to work with real HD EMG signals, a modern artificial hand and an amputee as test subject, we were able to validate our ZYNQ-based controller in a real-world experiment. The subject was not only able to intuitively control different degrees of freedom of the prosthesis, he also achieved solving real-world problems as seen in the cup stacking experiment.

The prosthesis controller prototype implemented a well-established pattern recognition-based classifier to proof its functionality. In this work, its real-life applicability has been shown through the processing of real online HD EMG signals from an amputee and the control of a real prosthesis. In future work, we plan to implement more advanced pattern recognition-based feature extractors using Vivado HLS.

References

[1] Andreas Agne, Markus Happe, Andreas Keller, Enno Lubbers, Bernhard Plattner, Marco Platzner, and Christian Plessl. ReconOS: An Operating System Approach for Reconfigurable Computing. *IEEE Micro*, 34(1):60–71, 2014.

[2] Alexander Boschmann, Andreas Agne, Linus Witschen, Georg Thombansen, Florian Kraus, and Marco Platzner. Fpga-based acceleration of high density myoelectric signal processing. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8. IEEE, 2015.

[3] Alexander Boschmann and Marco Platzner. Reducing the limb position effect in pattern recognition based myoelectric control using a high density electrode array. In *IEEE ISSNIP Biosignals and Biorobotics Conference (BRC)*, 2013.

[4] Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification (Second Edition)*. Wiley-Interscience, 2001.

[5] Todd Farrell and Richard F Weir. Analysis window induced controller delay for multifunctional prostheses. Myoelectric Symposium, 2008.

[6] Bernard S Hudgins, Philip A Parker, and Robert N Scott. A New Strategy for Multifunction Myoelectric Control. *IEEE Trans. Biomed. Eng.*, 40(1):82–94, 1993.

[7] OT Bioelettronica. *Quattrocento Multichannel Data Sheet*, 2015.