TÉCNICO LISBOA

Wearable IoT System for Monitoring People

Maria Inês Costa Frutuoso

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Rui António Policarpo Duarte
Prof. Horácio Cláudio De Campos Neto

Examination Committee

Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. Rui António Policarpo Duarte
Member of the Committee: Prof. António Manuel Raminhos Cordeiro Grilo

September 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank my supervisors Prof. Horácio Neto and Prof. Rui Duarte for their enthusiastic guidance over the last months. I am grateful for having the opportunity to learn with their expertise and insightful contributions.

I want to thank all who I have met throughout my academic adventure and with whom I have grown both personally and professionally.

I am thankful for the unconditional support of my loved ones, family and friends, along my journey.

Abstract

Wearable devices used for personal monitoring applications have been improved over the last decades. However, these devices are limited in terms of size, processing capability and power consumption. This thesis proposes an efficient hardware/software embedded system for monitoring bio-signals in real-time, including a heart rate calculator using Photoplethysmography (PPG) and an emotion classifier from Electroencephalography (EEG). The system is suitable for outpatient clinic applications requiring data transfers to external medical staff. The proposed solution contributes with an effective alternative to the traditional approach of processing bio-signals offline, by proposing a SoC-FPGA based system that is able to fully process the signals locally, at the node. Two sub-systems were developed targeting a Zynq 7010 device and integrating custom hardware IP cores that accelerate the processing of the most complex tasks. The PPG sub-system implements an autocorrelation peak detection algorithm to calculate heart rate values. The EEG sub-system consists of a KNN emotion classifier of preprocessed EEG features. The hardware/software solutions were compared to the software-only implementations executing in the Zynq's ARM processor, having obtained a speedup up to 40 times. The system consumes only 36% of the Zynq's resources and thus new functionalities may be added. The proposed system constitutes the foundation of more complex biometric systems, that may benefit from the combination of different reusable IP cores. This work overcomes the limitations of microcontrollers and general-purpose units, presenting a scalable and autonomous wearable solution with high processing capability and real-time response.

Keywords: Electroencephalography, Hardware/software co-design, Photoplethysmography, SoC FPGA, Wearable monitoring devices

Resumo

Os dispositivos *wearable* utilizados para monitorização da pessoa têm sido melhorados nas últimas décadas. Contudo, estes dispositivos estão limitados pelas dimensões, a capacidade de processamento e o consumo energético. Esta tese propõe um sistema *hardware*/*software* para a monitorização de bio-sinais em tempo-real, compreendendo uma calculadora de ritmo cardíaco utilizando a Fotopletismografia (PPG) e um classificador de emoções a partir da Electroencefalografia (EEG). O sistema é adequado para práticas em ambulatório que necessitem de transferir dados para equipas médicas remotas. A solução proposta é uma alternativa eficaz à abordagem tradicional de processamento de bio-sinais fora do dispositivo de aquisição, apresentando um sistema centrado numa plataforma SoC FPGA, que processa os bio-sinais localmente. Dois subsistemas foram desenvolvidos, projectados para o dispositivo Zynq 7010, integrando núcleos IP customizados em *hardware* para acelerar o processamento das tarefas computacionalmente mais complexas. O subsistema PPG implementa o algoritmo de detecção de picos de autocorrelação para calcular ritmos cardíacos. O subsistema EEG consiste num classificador KNN a partir de sinais EEG pré-processados. As soluções *hardware*/*software* foram comparadas com as implementações de *software*, executadas no processador ARM da Zynq, obtendo uma aceleração de até 40 vezes. O sistema ocupa 36% dos recursos disponíveis na Zynq, evidenciando que novas funcionalidades podem ser adicionadas. O sistema proposto constitui uma base para sistemas biométricos mais complexos, que beneficiem da combinação de diferentes núcleos IP reutilizáveis. Este trabalho supera as limitações de microcontroladores e unidades de processamento genéricas, apresentando uma solução wearable escalável, autónoma, com grande capacidade de processamento e resposta em tempo-real.

Palavras-chave: Co-projecto *hardware/software*, Dispositivos *wearable* de monitorização, Electroencefalografia, Fotopletismografia, SoC FPGA

Contents

List of Tables

List of Figures

Listings

Acronyms

ABP	Arterial Blood Pressure
ADC	Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
avAE	Average Absolute Error
AXI	Advanced eXtensible Interface
BCI	Brain-computer Interfaces
BLE	Bluetooth Low Energy
BRAM	Block RAM
CLB	Configurable Logic Blocks
DDR	Double Data Rate
DMA	Direct Memory Access
DSP	Digital Signal Processor
ECG	Electrocardiogram
EEG	Electroencephalography
FF	Flip-flop
FPGA	Field-Programmable Gate Array
GCC	GNU Compiler Collections
GP	General Purpose
HLS	High-Level Synthesis
HP	High Performance
HR	Heart Rate
HW/SW	Hardware/Software

IC	Integrated Circuit
IP	Intellectual Property
IR	Infra-red
I²C	Inter-Integrated Circuit
KNN	K-Nearest Neighbours
LDA	Linear Discriminant Analysis
LUT	Look-up table
MAE	Mean Absolute Error
PCA	Principal Component Analysis
PL	Programmable Logic
PPG	Photoplethysmography
PPV	Positive Predictive Values
PS	Processing System
PTT	Pulse Transit Time
QDA	Quadratic Discriminant Analysis
RAM	Random-access Memory
RMSE	Root Mean Squared Error
RR	Respiratory Rate
RTL	Register Transfer Level
SCL	Serial Clock Line
SDA	Serial Data Line
SoC	System-on-Chip
STA	Static Timing Analysis
SVM	Support Vector Machine
UART	Universal Asynchronous Receiver/Transmitter
XADC	Xilinx Analog-To-Digital Converter
XSA	Xilinx Support Archive
ZYBO	ZYnq BOard

Chapter 1

Introduction

Over the last decades, wearable monitoring systems have been researched, developed and progressively enhanced to support healthcare needs, and fit for real-time bio-signals processing, including heart rate measurement and emotional state recognition. As a result, wearable devices are becoming more portable, user-friendly, accurate and reliable, which minimizes the disturbance to user's daily routine. Moreover, combined with access to wireless Internet, these devices are being used in remote subject monitoring. This thesis proposes a wearable solution that can assist different groups of people, as it can provide remote healthcare tracking, overcoming the state-of-the-art systems.

1.1 Motivation

Medical systems monitor valuable information from biometric signals, such as cardiac activity, physical movements and brain activity. More specifically, brain-computer interfaces (BCI) are developed to allow disabled people to control an external device using electrical signals from the brain, overcoming physical or neuromuscular limitations. As such, electroencephalography (EEG), a technique for monitoring brain activity, is widely used in these applications.

Wearable EEG devices have been given prominence in the last couple decades, although most of those systems are built for operation in a laboratory environment. That is, under controlled conditions with static elements, such as simulated ambience, luminosity or background noise [1]. Therefore, these applications do not prioritize portability or hardware simplicity, but rather high efficiency on signal acquisition and processing. "Beyond wearable" EEG, as suggested in [1], are closer to real life applications, reducing wires and becoming more comfortable and usable. However, such devices face the problem of battery autonomy. The commercialized EEG products available in the market last up to 9 hours. The inclusion of multiple sensors, such as optoelectronic sensors, thermometers and accelerometers, urges power efficiency during signal processing. Larger batteries provide longer autonomy despite of being impractical, while smaller ones grant more mobility but impose frequent charging to the user. Moreover, the high complexity of a monitoring system is often linked to larger devices that do not fit the wearable needs. The thesis proposes a solution that offers the capacity of handling complex bio-signals, while

guaranteeing portability.

1.2 Objectives and contributions

The novelty of this work is the usage of a System-on-Chip (SoC) Field-Programmable Gate Array (FPGA) to take advantage of high processing speed and reconfigurable logic. This kind of device is useful to create flexible and customized hardware solutions with high performance and low power consumption. It is intended to perform signal processing tasks locally and online, instead of transmitting the collected raw sensor data to be processed by an external server, as conventional systems do. By doing the computations locally, at the node, the required bandwidth and power consumption are minimized. Furthermore, this architecture offers parallel computation, which is suitable to handle multiple biometric signals at a time. Such functionalities overcome the limitations of conventional wearable solutions that use general-purpose CPU.

The proposed system intends to measure a person's heart rate using photoplethysmography (PPG) and to assess emotional state via EEG. PPG is a technique widely used in smartwatches, that uses a light source pointed to the skin to detect blood volume changes in a vascular tissue, which is captured by a sensor and translated to an electric signal. Blood variations are synchronous to heart beat, so heart rate can be computed after processing this signal. With EEG, the electric activity of the brain is recorded using electrodes placed on the head, resulting in a set of signals with multiple frequencies. Emotion recognition is achieved after processing EEG signal, extracting relevant features (or patterns) from it and building a classifier that identifies emotions accurately. The biometric system demonstrates the use of two independent sub-systems, each one dedicated to the respective bio-signal. The PPG subsystem will handle PPG samples since their acquisition by an optoelectronic sensor, with the purpose of determining instant heart rate values. The EEG sub-system will consist of a machine learning classifier that takes as input preprocessed EEG signals and returns the predicted emotion.

The main goal is to take advantage of SoC FPGA to conceive a real-time monitoring system for bio-signals. One contribution of this work is to develop dedicated hardware to process the bio-signals collected by the sensors. This will be achieved by designing reconfigurable logic accelerators, which contain preconfigured functions, as Intellectual Property (IP) cores. These blocks are intended to accelerate the processing of specific bio-signals. The processing tasks are distributed between software-only instructions and the custom IP cores, constituting a hybrid Hardware/Software (HW/SW) architecture. The most complex tasks are handled by the IP cores, and the remaining ones are implemented in embedded software run by the processor. An objective of the thesis is to run the processing tasks in shorter times, when compared to the software-only implementations. The concept of the proposed system architecture is sketched in Figure 1.1. This includes an abstract representation of the Zynq-7010 SoC. Two main components can be distinguished: the Processing System (PS) – corresponding to the dual-core processor – and the Programmable Logic (PL) – related to the FPGA fabric. The IP cores, included in the PL, are connected to the PS by Advanced eXtensible Interface (AXI) buses. Moreover, this work aims to find the optimal design of the system, such that the hardware components necessary for its

implementation fit the resources available in the targeted platform.

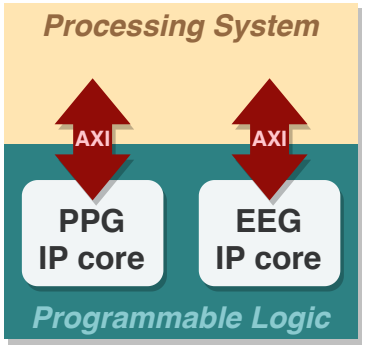


Figure 1.1: Proposed system architecture.

The system based on a SoC FPGA will be demonstrated using the ZYnq BOard (ZYBO) [2]. This board includes a Zynq-7010 All Programmable SoC[1], which is composed of a dual-core ARM Cortex-A9 processor[2]. This dedicated hardware offers programmable logic solutions, reducing design complexity and optimizing performance-per-watt ratio, being advantageous to this application.

1.3 Outline

The thesis is organized as follows. Chapter 2 provides an overview of the PPG and EEG techniques and presents the state-of-the-art algorithms for processing the bio-signals. Moreover, the chapter reviews FPGA-based medical devices similar to the proposed by this thesis. Chapter 3 introduces the biometric sub-systems. It is described the process behind the selection of the algorithm for handling PPG signals and the classifier to identify emotions from EEG. The processing steps involved in both sub-systems are detailed and analysed. The chapter closes with the high-level architecture of the biometric system. The development of the hardware modules that accelerate the processing tasks is addressed in Chapter 4. The integration of those modules with embedded software is described in Chapter 5. The acceleration results obtained by the sub-systems are provided. Chapter 6 summarizes the conclusions of this work and suggests some directions for future work.

[1] Zynq-7000 SoC family webpage: `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`; accessed on 22nd May 2020.

[2] Cortex-A9 webpage: `https://developer.arm.com/ip-products/processors/cortex-a/cortex-a9`; accessed on 22nd May 2020.

Chapter 2

Background on biometric signals processing and HW/SW co-design

This chapter exposes and clarifies theoretical concepts used throughout this report, including an overview of relevant related work. The underlying framework of the thesis includes the biometric techniques EEG and PPG, but also the application of FPGA in biomedical prototypes. When analysing related researches, emphasis is placed on those supported by source code written in C, a low-level language suitable for embedded design.

2.1 Electroencephalography

Electroencephalography is a non-invasive technique for probing electric activity of the human brain neurons, by attaching electrodes on the scalp that detect voltage fluctuations upon ion flow [3]. Five major frequency bands can be identified in brain waves, depending on the neural activity – delta (1-3 Hz), theta (4-7 Hz), alpha (8-13 Hz), beta (14-30 Hz) and gamma (31-50 Hz) [3] –, whose frequency pattern changes may denote a response to an external stimulus, or some brain disorder [4]. Activities such as sleeping, exercising or meditation can also be detected in brain waves.

The positioning of electrodes is crucial for accurate signal acquisition, given the scope of the application. The standardization is set by the International 10/20 System [5], represented in Figure 2.1, although extended versions have been proposed [6]. There are two EEG montage types: referential and bipolar. Referential considers a common reference electrode, from which the potential difference of all electrodes is computed. In bipolar montage, each electrode is associated with a reference.

A common application of EEG is emotion classification, which maps and recognizes patterns on features of EEG signals from different known emotions. Russell [7] defined arousal as the metric for awareness or unawareness during an activity, and valence as the metric for pleasure or displeasure. Both quantities are described a 2D plane, where arousal is in the horizontal axis and valence in the vertical axis. The resulting emotion in each quartile is a combination of the two. The model is represented in Figure 2.2. According to Davidson et al. [9], positive valence activates the left side of the

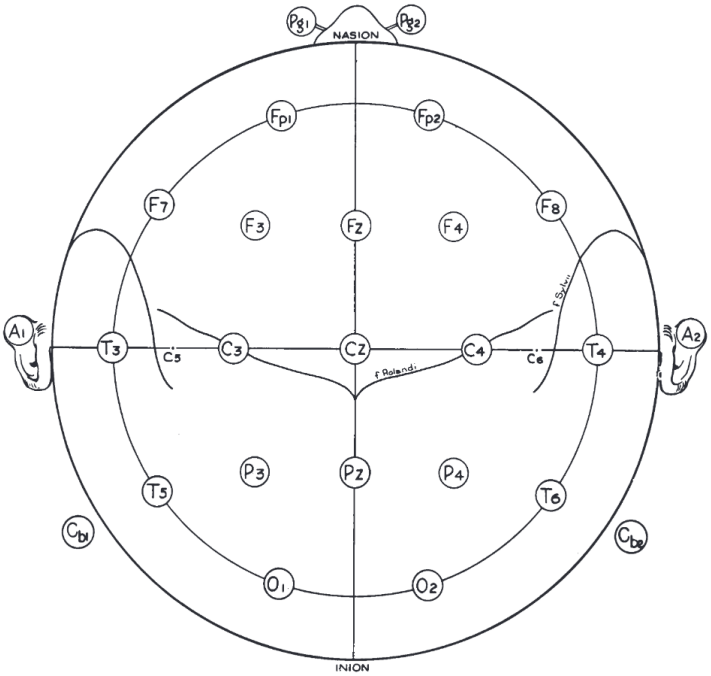


Figure 2.1: Electrode-positioning standard by International 10/20 System [5]. 'Pg' stands for pharyngeal area, 'Fp' for fronto polar, 'F' for frontal, 'T' for temporal, 'C' for central, 'P' for parietal, 'O' for occipital and 'Cb' for cerebellar.

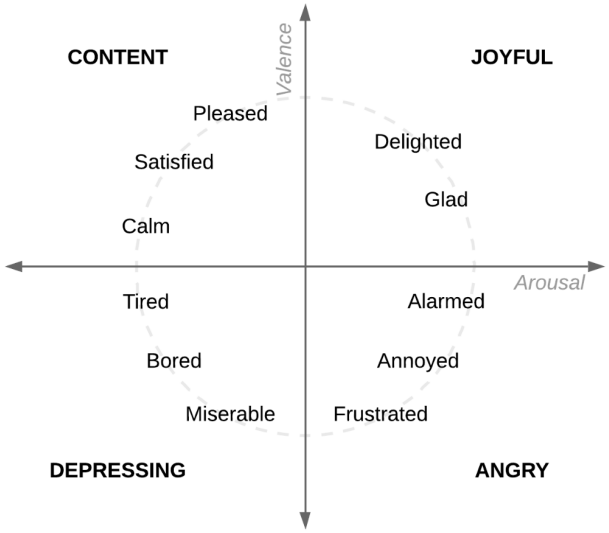


Figure 2.2: Russell's [7] Valence-Arousal model [8].

anterior temporal region, while negative valence stimulates the right side of frontal and anterior temporal regions. Regarding the relation between emotion dimensions and the power of major frequency bands, as per Kandel et al. [10], alpha is visible on parietal and occipital regions and associated to low arousal and high valence state; beta is prominent in frontal area during high arousal activities; delta and theta are related to low arousal.

2.1.1 Signal processing

Processing the EEG signal comprises several steps, namely noise reduction, signal enhancement, feature extraction and classification. During the acquisition via the electrode, the recorded signal is at-

tenuated by skin tissues and bones [10], but also subject to noise caused by muscular activities, eye movements, eye blinks and cardiac signals [11] [12], as represented in Figure 2.3. In fact, normal EEG signal amplitude ranges microvolts, although a single neuron promotes voltage changes of millivolts [10]. Therefore, in order to remove this noise, the EEG signal is preprocessed and its quality improved [11] [13]. Most of the possible methods that perform this use spatial or temporal filters, or both [11]. Spatial filters combine the recorded signals into a single one with higher Signal-to-Noise Ratio (SNR) [14]. Some examples, supported by references containing additional details, include Independent Component Analysis (ICA) [15], Common Average Referencing (CAR) [16], Surface Laplacian (SL) [17], Principal Component Analysis (PCA) [18] and Common Spatial Patterns (CSP) [19]. On the other hand, temporal filters attenuate specific frequencies of the signal and select those bands carrying relevant information. This can be achieved by applying discrete or fast Fourier transforms, finite impulse response (FIR) filters or infinite impulse response (IIR) filters [20]. After signal preprocessing, features are extracted, that is, patterns are identified in order to reduce dimensional space without losing essential information [21]. There are several extraction methods, such as Wavelet Transformations (WT) [21], Wavelet Packet Decomposition (WPD) [22] and Autoregressive (AR) [23], but also ICA and PCA can be applied in this step. Finally, classification is performed by, for example, Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Neural Network (NN) or k-Nearest Neighbour (KNN) [13]. Also, some of these classifiers have been extended to adaptive versions [14].

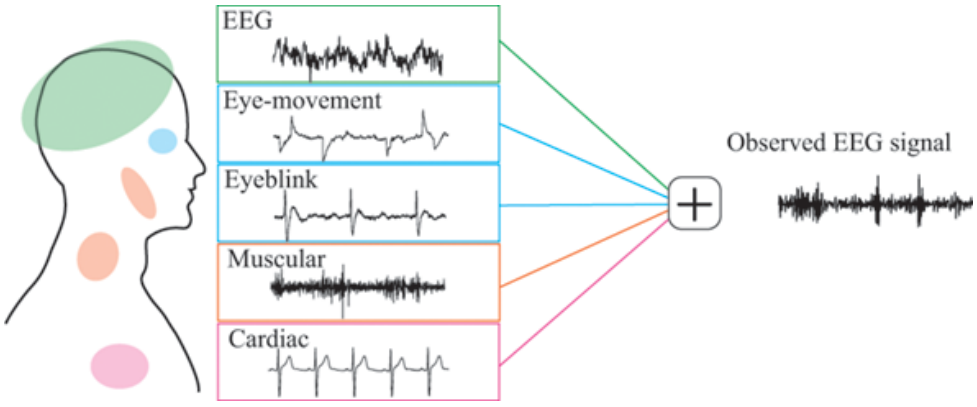


Figure 2.3: Different noise sources affecting a typical EEG waveform [24].

2.1.2 Related work

EEG has been used in several academic works with different purposes, namely emotion recognition [12], seizure and traumatic brain injury detection [25], but also identification of alcoholism, depression, dementia, epilepsy, Alzheimer and Parkinson [4]. In this section, the scope of the analysis confines to emotion recognition.

Lin et al. [26] studied the emotional responses from brain activity to stimuli of different music. Brain signals from 26 subjects were recorded by 24 electrodes, namely FP1, FP2, F7, F8, F3, F4, FT7, FT8, FC3, FC4, T7, T8, P7, P8, C3, C4, TP7, TP8, CP3, CP4, P3, P4, O1 and O2. The classification was performed by a support vector machine (SVM) into four emotional states, according to 2D emotion

model: 'joy', 'pleasure', 'angry' and 'sadness'. The classification accuracy obtained was 82.37%.

Vijayan et al. [27] proposed a subject independent algorithm involving wavelet transform, Shannon entropy, cross correlation and auto-regressive modelling. Four emotions, namely 'exciting', 'happy', 'sad' and 'hate', were identified by a multi class SVM. Tested in a non-real time mode, this algorithm was applied in a dataset containing brain signals of 32 subjects, stimulated by music videos. Multiple combinations of three, six and seven electrodes were evaluated. The highest accuracy obtained was 94.097%, with seven electrodes located at P7, P3, PZ, PO3, O1, CP2 and C4. Using six electrodes on FP1, AF3, F7, P7, P3 and PZ, an accuracy of 69.767% was achieved. Dividing this combination into halves, FP1, AF3 and F7 and P7, P3 and PZ, the accuracies reached 28.447% and 78.465%, respectively.

Commercial brain sensing devices are also used. Dhindsa et al. [28] collected brain signals using a *Muse* headband[1], containing four electrodes located at T9, FP1, FP2 and T10, from 40 subjects. The goal was to identify the emotional reaction to videos, from a set of 11 emotions – 'interest', 'amusement', 'happiness', 'sadness', 'fear', 'disgust', 'anger', 'hope', 'relief', 'surprise' and 'sympathy'. Two classifiers were used: SVM and logistic regression (LR). When performing leave-one-subject-out cross-validation, SVM obtained higher accuracies than LR, with 75% mean.

Another commonly used device for EEG signal extraction is *Emotiv Epoc* headset[2], equipped with 14 electrodes (AF3, AF4, F3, F4, F7, F8, FC5, FC6, P7, P8, T7, T8, O1 and O2). Liu et al. [29] carried an experiment with 30 subjects in order to identify movie-induced emotions. The accuracy was evaluated differently. The distinction between positive and negative emotions obtained 86.62% accuracy. The identification of a specific positive emotion, within the set of 'joy', 'amusement' and 'tenderness', reached 86.43%; for negative emotions, 'anger', 'disgust', 'fear' and 'sadness', this value stood at 65.09%. No accuracy data is presented regarding the emotion recognition within the set containing all aforementioned emotions. This device was also used by Anh et al. [30], but only three electrodes, AF3, FC6 and F4, were used for emotion recognition. 100 subjects were stimulated by a photo database and LIBSVM [31] used as classification method. Five emotions were considered: 'happy', 'relaxed', 'neutral', 'sad' and 'angry'. An overall accuracy of 9% was obtained.

Schaaff et al. [32] developed a three-emotion recognition system with a SVM with radial basis function (SVM-RBF) classifier and leave-one-out cross validation. The classes were 'pleasant', 'unpleasant' and 'neutral'. The visualization of images aroused emotions of five subjects, recorded by four electrodes – FP1, FP2, F7 and F8. Focusing on the alpha frequency, the classification accuracy was 48.89%.

Four electrodes were also used by Ali et. al. [33] to detect one of the four emotions of the 2D emotion model – FP1, FP2, F3 and F4 –, because of "pre-frontal cortex in emotion regulation and conscious experience". Feature extraction involved wavelet energy, modified energy, wavelet entropy and statistical moments. Three classification methods were evaluated: quadratic discriminant analysis (QDA), k-nearest neighbours (KNN) and SVM-RBF. Using EEG signals from DEAP, a preprocessed dataset concerning 32 subjects [34], these experiments obtained 60.78%, 75.53% and 83.87% accuracy,

[1]Muse website: `https://www.choosemuse.com/`; accessed on 19th March 2020.
[2]Emotiv website: `https://www.emotiv.com/`; accessed on 19th March 2020.

respectively.

Wei et al. [35] developed their own wearable emotion detection headband containing four electrodes. The proposed algorithm combined features such as power spectral density, signal power and common spatial pattern. The classification using linear discriminant analysis (LDA) aimed the distinction of positive and negative emotions. They also tested two electrode locations: FP1, FP2, T3 and T4; A1, FP2, F7 and F8. Subject-independent accuracies of 64.73% and 66.74% were respectively achieved. The experiment was undertaken by 12 subjects stimulated by picture visualization.

Chatchinarat et al. [36] studied the effects of both number and location of electrodes and frequency bands for emotion classification, using DEAP dataset and the SVM classifier. This review addresses three out of ten tested combinations of electrodes: FP1 and FP2; T3 and T4; O1 and O2. When using five bands as feature per electrode channel, the classification of four emotions following the 2D emotion model reached accuracies of 30.73%, 30.73% and 28.65%, respectively. The authors also computed the classification accuracy using only the frequency bands alpha and beta as features, comparing it to the five-band case. These results were presented considering two-emotion classification, where arousal and valence axis are separated. Considering only the arousal, the accuracy using two bands was 57.81% for all three electrode combinations; using five bands the accuracy was respectively 57.29%, 56.77% and 55.21%. Equivalently, considering only the valence axis, the accuracy using two bands reached 53.65%, 57.29% and 47.40%, respectively; using five bands the accuracy achieved 47.40%, 56.25% and 56.25%. This demonstrates that using two bands is almost similar to using five bands.

A similar study was conducted by Li et al. [37], where the classification of valence and arousal dimensions was tested varying the amount of EEG channels. The preprocessed signals of the DEAP dataset were decomposed by wavelet transform, to obtain entropy and energy as EEG features of a KNN classifier. In the experiment, it was confirmed that classification accuracy increases with the number of channels. Using 32 channels, the valence and arousal classifications were 95.7% accurate. Using 10 channels, the accuracies dropped to 89.54% and 89.81%, respectively. The article mentions a MATLAB program, however no implementation is provided.

The KNN and SVM classifiers were compared in terms of accuracy of assessing the levels of valence and arousal by Mohammadi et al. [38]. Features were obtained by discrete wavelet transform (DWT) to decompose the EEG signal into the main frequency bands. The input data corresponded to DEAP dataset, however a subset of 10 EEG channels was considered. The results showed that KNN outperformed SVM, and the maximum classification accuracies obtained by KNN for valence and arousal were, respectively, 86.75% and 84.05%. Moreover, this work studied the effects of the temporal window and the combinations of EEG electrodes on classification accuracy, besides the classifier. Wider temporal windows led to more accurate results. The electrode pair FP1-FP2 was regarded as the most reliable for identifying emotions.

Hatamikia et al. [39] analysed the performance of different feature selection methods which were tested using three emotion classifiers – LDA, QDA and KNN. The preprocessed version of the DEAP dataset was consumed. The target application was to distinguish three intensity levels (classes) of valence and arousal. Using each feature selection method, the KNN classifier obtained the highest

classification accuracy results. The best accuracy results regarding the valence class were 51.20% for LDA, 57.42% for QDA and 61.10% for KNN. The best results for the arousal class were, respectively, 52.36%, 57.18% and 65.16%. The KNN obtained the highest accuracy.

Table 2.1 summarizes the characteristics of the aforementioned EEG researches. It includes the classification method, the number of the electrodes, the rounded values for the classification accuracies followed by the classification scope. The source code and the dataset used are also referred, if reported by the authors. The researches did not provide complete open-source implementations of the emotion detection systems, although some refer the tools used during the development. For instance, [27], [28] and [32] referred a MATLAB implementation, without providing its source code publicly. [33] developed an accurate and portable emotion classification system, but no implementation references were provided. [35] pointed a public available tool, which is supported in C language, however no software specification of the algorithm could be extracted.

Most of the available open-source repositories implemented an emotion detector using high-level languages, such as Python and MATLAB. This kind of languages is relevant to bioinformatics, being practical to translate the developed algorithms into preliminary software instructions. Such implementations are not convertible into an embedded system design, since lower level languages are required. Therefore, it is necessary to select the best classifier that may be specified in a low-level language, namely C. According to [12], the SVM and KNN classifiers are the most used in emotion recognition applications. Their classification accuracy results varied among the reviewed researches listed on Table 2.1. Both classifiers are suitable candidates to integrate the biometric system. Analysing the working principle of these methods, and privileging the feasibility of mapping a classifier into a low-level language implementation, the KNN will be considered in the final solution, to distinguish emotions of Russell's 2D model. In fact, at the time this work was carried out, no software implementations of the SVM using high-level languages were publicly available. On the other hand, KNN implementations were found, facilitating the process of translating its algorithm into low-level instructions.

2.2 Photoplethysmography

Photoplethysmography (PPG) is an optical technique that detects blood volume changes in a microvascular tissue [43]. PPG uses a light source for emitting light to the tissue and a photodetector for measuring the consequent received light, by transmission or reflection as seen in Figure 2.4, from which the blood volume variation is estimated. The principle of PPG is as follows. During the cardiac cycle, arteries suffer blood volume reduction when transiting from the systolic phase to the diastolic phase [44]. The PPG sensor detects this change optically and its photodetector converts the received light energy into an electrical current [43]. A waveform can be acquired and some physiological parameters extracted; for instance, the variability of the time between heartbeats [45].

Typically, red LEDs are used for light emission, since red and near infrared light pass more easily through water, the main constituent of tissues [43]. Green LEDs are also commonly chosen in several experiments [46]. Some studies suggested that, comparing to red light, green is more suitable for pulse

Table 2.1: Comparison of existing EEG research works.

Research	Classifier	Electrodes	Accuracy	Classes	Source code	Dataset
[26]	SVM	24	82.4%	2D model	✓ BSVM [40]	own
[27]	SVM	3 (FP1 AF3 F7) 3 (P7 P3 PZ) 6 7	28.4% 78.5% 69.8% 94.1%	2D model	✗ MATLAB	DEAP
[28]	SVM	4 (T9 FP1 FP2 T10)	75.0%	11 emotions	✗ MATLAB ✓ FBAR [41]	own
[29]	SVM	14	86.6% [b]	1D	✗ EEGLAB [42] ✓ LIBSVM	own
[30]	SVM	3 (AF3 FC6 F4)	9.0%	2D model + 'N'[a]	✓ LIBSVM	own
[32]	SVM	4 (FP1 FP2 F7 F8)	48.9% [b]	1D + 'N'[a]	✗ MATLAB ✓ LIBSVM	own
[33]	QDA KNN SVM	4 (FP1 FP2 F3 F4)	60.8% 75.5% 83.9%	2D model	✗	DEAP
[35]	LDA	4 (A1 FP2 F7 F8)	66.7% [b]	1D	✗ OpenViBE	own
[37]	KNN	32	95.7% [b] [c]	2D model	✗ MATLAB	DEAP
[38]	KNN	10	86.7% [b] 84.0% [c]	2D model	✗	DEAP
[39]	LDA QDA KNN	32	51.2% [b], 52.4% [c] 57.4% [b], 57.2% [c] 61.1% [b], 65.2% [c]	2D model	✗	DEAP
[36]	SVM	2 (FP1 FP2)	30.7%	2D model	✓ LIBSVM	DEAP

[a] 'N' stands for 'neutral' emotion.
[b] Classification accuracy referred to class 'valence'.
[c] Classification accuracy referred to class 'arousal'.

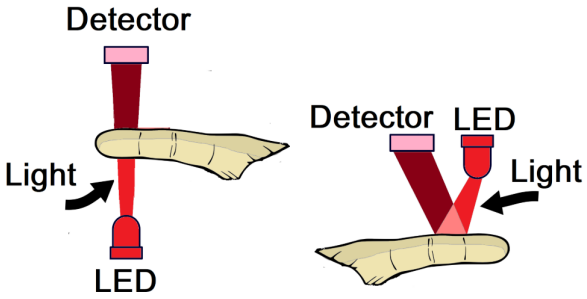


Figure 2.4: Operation of PPG finger sensors by transmission (left) and reflection (right) [45].

rate monitoring [47] and under motion artifact conditions [48].

A PPG signal comprises two components: a pulsatile (AC), given by cardiac variations in blood volume caused by heartbearts, and a superimposed (DC), variable with some anatomic factors, such as respiration, thermoregulation, vasomotor and sympathetic nervous system activities [46] [43]. These components can be seen in Figure 2.5, where a typical PPG waveform is represented.

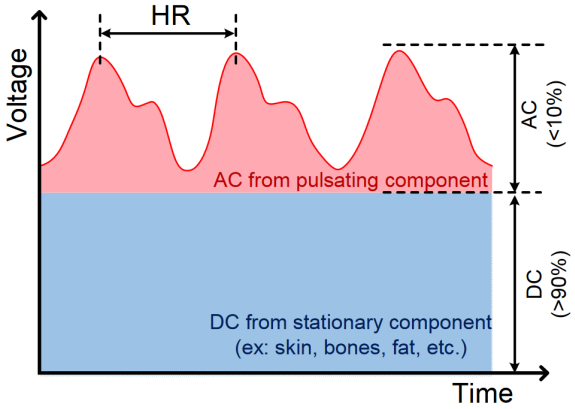


Figure 2.5: Typical PPG waveform [49].

2.2.1 Applications and characteristics

The monitoring and analysis of PPG signal unveils a wide set of clinical applications, namely measurement of heart rate, blood pressure, respiratory rate, blood oxygen saturation and several vascular assessments. It can also be incorporated into detection tools of some cardiovascular diseases, such as vascular ageing [43].

An alternative method is electrocardiogram (ECG), a representation of the electrical activity of the heart, measured by multiple leads attached on body surfaces [50]. Several practical works in the literature that include PPG technologies use ECG for reference values. PPG does not have a complex hardware implementation nor the need of a reference signal, unlike ECG [45]. PPG is also regarded as a non-invasive and low-cost method [43]. Therefore, it is a portable, ready-to-use and convenient device from the user point of view. PPG sensors can be placed on different anatomical positions – forehead, earlobe, torso, wrist, fingertip and ankle –, but PPG signal has higher quality at earlobes or fingertips [46].

PPG also faces some disadvantages, such as sensibility to subject motion, probe-tissue movement and ambient light interference [43]. Also, PPG recordings vary with subject anatomy and the magnitude is controlled by the pressure applied by the wearable device on the skin [46].

2.2.2 Related work

There are several techniques of processing PPG signal that vary with the different kinds of applications. For this application, it is intended to use an algorithm with an implementation associated and preferably supported by both source code and dataset for test and validation purposes. This review addresses research works and publicly available repositories.

Research works

A simple and low complexity algorithm for detecting pulse peak has been presented by Jang et al. [51]. The method used a slope sum function (SSF) with an adaptive thresholding scheme, but also cascaded recursive digital filters, for the initial noise removal. Testing was performed using the health improvement and management system (HIMS) database, containing pulse peaks manually annotated. Two rules were applied for true detection: the index-based and the interval-based. The first one declares 'true' if "the difference between manual and algorithm generated annotations is less than three samples (approximately 5 ms) intervals". The latter considers a similar variance regarding the QRS interval. These evaluation metrics obtained positive predictive values (PPV) of 60.57% and 80.29%, respectively. The authors have concluded that the proposed algorithm was suitable for real-time applications, such as pulse rate monitoring, identification of other characteristic points of the PPG signal referred to the pulse peak, but also investigation of both pulse transit time and pulse rate variability. However, no source code on the developed algorithm is provided.

Zong et al. [52] also developed an algorithm using SSF and adaptive thresholding, for detecting arterial blood pressure (ABP) pulses. The MIT-BIH Polysomnographic Database[3] – containing 368,364 beats annotated in ECG signals – was used for evaluating the accuracy, obtaining 99.31% of correlation. A second evaluation approach compared the ABP pulse detections to a newly created reference database comprising 39,848 ABP annotations, where the accuracy stood at 96.41%. The source code in C language is freely available on PhysioNet [53] website, included in PhysioToolkit.

A heart rate detection system using an Arduino board was developed by Das et al. [54]. The simple algorithm based on short term autocorrelation technique over the time shifted PPG signal, acquired from ten healthy volunteers in real-time. The distance between peaks originated by the autocorrelation function is used to compute heart rate. The system validation consisted on the comparison to BIOPAC MP150 system, while testing was performed using PhysioNet's MIMIC database [55]. Results have shown no difference between the HR measurements given by both systems, whether the signal presented clean or noisy waveforms. Despite the algorithm's simplicity and implementation in Arduino, no code was referred.

[3] PhysioNet website: `https://www.physionet.org/content/slpdb/1.0.0/`; accessed on 17th April 2020.

Similarly, another system for heart rate analysis containing an Arduino was conceived by van Gent et al. [56]. The authors developed HeartPy, a Python toolkit, but also its implementation in embedded C, in order to be deployed in Arduino. The algorithm comprised three steps: preprocessing involved peak enhancement, FIR filtering, and outlier detection; peak detection consisted of the adaptive peak detection threshold method; error detection is corrected by thresholding the sequence of RR-intervals, that is, elapsed time between peaks. This package was validated on a PPG dataset collected in previous work – Physionet's BIDMC PPG and Respiration Dataset [57]. The heart rate estimation presented a root mean squared error (RMSE) of 4.18 bpm. Also, comparing to two "popular available open source algorithms", Pan-Tompkins QRS and HRVAS ECGViewer, the developed toolkit obtained the lowest error rates. However, these algorithms were designed and intended to be computationally efficient on ECG data, rather than PPG. GitHub repositories are available online[4] [5].

More complex algorithms have also been investigated. In order to estimate HR during physical activities, Temko [58] studied the WFPV algorithm. The method comprised "a Wiener filter to attenuate the motion artifacts, a phase vocoder to refine the HR estimate and user-adaptive post-processing to track the subject physiology". The public database from IEEE SP Cup 2015 [59] allowed to test the model, containing data of three activity types with different intensities. Performance results showed an average absolute error (avAE) of 1.97 bpm. Also, regarding the lightest activity and comparing its avAE results to other real-time algorithms – TROIKA [59], JOSS [60], SpaMa [61], EEMD [62], IMAT [63] and MC-SMD [64] –, tge WFPV algorithm outperformed those alternatives, excepting SpaMa. When considering all activity types, WFPV obtained the lowest error. The author shared a GitHub repository containing the MATLAB implementation[6].

Sharma [65] proposed the variational mode decomposition (VMD), a new technique for heart rate estimation from PPG. The concept consists of decomposing the PPG signal into a number of modes, or sub-signals, of different center frequency, energy, and bandwidth. The ones where the heart rate information influences more dominantly are then selected by Fast Fourier Transform (FFT) and processed using principal component analysis (PCA), and the heart rate is extracted applying short-time Fourier transform (STFT). The algorithm is validated over three databases: Capnobase, MIMIC, and University of Queens Vital Sign (UQVS). The root mean square error is assessed, obtaining 0.23 bpm, 0.41 bpm and 1.1 bpm, respectively. Also, this approach outperformed existing algorithms such as PSD [66], EMD [67] and EEMD-PCA [68]. There are no references to any implementation source codes.

Besides heart rate detection, respiratory rate (RR) estimation methods were investigated by Charlton et al. [69]. Generically, the first step of a RR algorithm consisted of extracting respiratory signal(s) using a feature-based technique. Three features were measured: baseline wander (BW), amplitude modulation (AM) and frequency modulation (FM). Then, RR is estimated whether by computing power spectral density of the signals using Fourier Analysis or detecting individual breathing cycles using "count-orig" methodology [70]. A final RR estimate can be computed as the average of the three aforementioned

[4]Python Heart Rate Analysis Toolkit repository: `https://www.github.com/paulvangentcom/heartrate_analysis_python`; accessed on 18th April 2020.

[5]Arduino Heart Rate Analysis Toolkit repository: `https://www.github.com/paulvangentcom/heartrate_analysis_Arduino`; accessed on 18th April 2020.

[6]PPG repository: `https://www.github.com/andtem2000/PPG`; accessed on 18th April 2020.

features if all are within 4 bpm of each other. Performance of multiple methodologies was evaluated using PhysioNet's MIMIC II database (Version 3) [71]. The results showed that, applying the first estimation approach, using individually BW, AM and FM respiratory signals, the mean absolute error (MAE) stood at 8.18 bpm, 11.14 bpm and 12.11 bpm, respectively. However, considering the second estimation approach, the MAE values dropped to 4.28 bpm, 5.58 bpm and 7.95 bpm, respectively. Additionally, applying quality assessment and fusion step over the three respiratory signals, MAE reached 10.52 bpm and 3.36 bpm, when estimating by Fourier analysis and breath detection, respectively. A GitHub repository[7] containing the developed MATLAB code was provided.

Kong et al. [72] developed an heart rate tracking algorithm, TVSMART, standing for "time-varying spectral motion artifact removal technique". Besides handling PPG data acquired from wristbands or forehead devices, the method also uses accelerometer signals and comprises: a preprocessing phase using normalization and bandpass filter; time-frequency spectrum (TFS) estimation and motion artifact removal using variable-frequency complex demodulation (VFCDM); cubic spline regression for HR estimation. The algorithm was further compared to WFPV [58], because of, according to the authors, its public availability and better performance compared to most of the published algorithms. Two databases were used for validation: one from IEEE SP Cup [59] and one own dataset, Chon Lab. For slow walking data, and considering both databases, WFPV obtained an average absolute error (avAE) 4.46 bpm, while TVSMART obtained 3.53 bpm. Regarding only the IEEE SP Cup dataset, avAE values stood at 4.00 bpm and 3.68 bpm, respectively. There are no evidences on how the authors implemented in practice this algorithm.

Table 2.2 summarizes the main characteristics of the reviewed articles: the application scope, main results and supporting source code and datasets. It can be seen that every reviewed work referred a public available dataset. The most accurate results were registered by [54], obtaining a 100% correlation with a complex monitoring system. However, only [56] provided a C code for heart rate measurement, being a suitable option to the implementation.

Publicly available repositories

Open-source implementations have also been investigated. In the next paragraphs, GitHub repositories associated to the "PPG" keyword, and filtered by C/C++ language, are discussed. Several sensors are used, although few are provided a proper documentation.

ProtoCentral[8] supplies an oxygen saturation and heart rate monitoring system[9], implemented in a custom device containing the Texas Instruments's AFE4490 integrated circuit[10] (IC). The supporting documentation addresses the Arduino connections, rather than the PPG extraction and processing algorithms. A similar IC, the AFE4404[11], has been integrated with a STM32L443xx[12] microcontroller. The

[7] Critical Data Book repository: `https://www.github.com/MIT-LCP/critical-data-book`; accessed on 18th April 2020.
[8] ProtoCentral homepage: `https://www.protocentral.com/`; accessed on 30th April 2020.
[9] AFE4490 repository: `https://www.github.com/Protocentral/AFE4490_Oximeter`; accessed on 30th April 2020.
[10] AFE4490 datasheet: `https://www.ti.com/lit/ds/symlink/afe4490.pdf`; accessed on 30th April 2020.
[11] AFE4404 datasheet: `https://www.ti.com/lit/ds/symlink/afe4404.pdf`; accessed on 30th April 2020.
[12] STM32L443xx datasheet: `https://www.st.com/resource/en/datasheet/stm32l443vc.pdf`; accessed on 30th April 2020.

Table 2.2: Overview of reviewed PPG researches.

Research	Purpose	Technique	Results	Source code	Dataset
[51]	Pulse peak	SSF	PPV = 80.29%	✗	HIMS
[52]	ABP pulses	SSF	99.31% ECG correlation	✓ C	MIT-BIH
[54]	HR	Autocorrelation	100% BIOPAC correlation	✗ C/C++	MIMIC
[56]	HR	Adaptive peak detection	RMSE = 4.18 bpm	✓ Python ✓ C	BIDMC PPG
[58]	HR	WFPV	avAE = 1.97 bpm	✓ MATLAB	IEEE SP Cup 2015
[65]	HR	VMD	RMSE = 0.23 bpm RMSE = 0.41 bpm RMSE = 1.1 bpm	✗	Capnobase MIMIC UQVS
[69]	RR	Fourier analysis ”count-orig”	MAE = 10.52 bpm MAE = 3.36 bpm	✓ MATLAB	MIMIC-II
[72]	HR	VFCDM	avAE = 3.68 bpm	✗	IEEE SP Cup 2015

repository[13] is not paired to any documentation.

The intended sensor to be used in the implementation is the Maxim Integrated's MAX3010x, a pulse oximeter and heart-rate sensor integrated circuit series. The pulse oximeter allows the measurement of oxygen saturation in the blood. Regarded as portable, stable and reliable [73] [74], multiple documented implementations were found.

A MAX30100[14] version driver[15], for Arduino deployment, is well documented by a tutorial available online[16]. The algorithm consists of a DC signal removal filter to only keep the AC component, a mean median filter to improve peak detection and a Butterworth filter to remove the higher level harmonies. Heart rate is then determined by the delay between two beats. The author performed a single measurement test and compared it with the value obtained by a blood pressure measuring device. The error obtained was 0.81 bpm. This driver has been implemented by two further open-source projects.

The first one[17] cites the previous tutorial, despite its implementation being targeted to the STM32F4 board[18]. It monitors both oxygen saturation and heart rate in real-time. Secondly, a wearable health monitor[19] has been developed including several sensors, namely the MAX30100 in order to perform similar measurements. This module has been programmed by the aforementioned driver. The documentation consists of a paper report.

An Arduino project featuring identical monitoring, using the MAX30102[20] module, is also available online[21], along with a step-by-step guide[22]. The algorithm's source code is also extensively commented.

The previous sensor is also supported by two relevant drivers for Arduino with similar purposes. One[23] offers documentation on hardware and troubleshooting, but also commented examples. The latter[24], dedicated to MAXREFDES117#[25], a board embedding the sensor, has been developed by the same manufacturer. Support documentation regarding hardware is available on board's website.

SparkFun has also developed an Arduino library[26] for the MAX3010x sensor family, including example sketches. Besides monitoring presence sensing, temperature and oxygen levels, heart rate measurement is implemented by Penpheral Beat Amplitude (PBA) algorithm. The developed code is widely commented.

A wide range of C libraries dedicated to MAX3010x sensor is available. This grants multiple solutions

[13] AFE4404 repository: `https://www.github.com/opetany93/PPG-Dev2`; accessed on 30th April 2020.
[14] MAX30100 datasheet: `https://www.datasheets.maximintegrated.com/en/ds/MAX30100.pdf`; accessed on 7th May 2020.
[15] MAX30100 repository: `https://www.github.com/xcoder123/MAX30100`; accessed on 2nd May 2020.
[16] *Implementing pulse oximeter using MAX30100*: `https://www.morf.lv/implementing-pulse-oximeter-using-max30100`; accessed on 7th May 2020.
[17] Pulse-Oximeter-with-MAX3010X repository: `https://www.github.com/GCY/Pulse-Oximeter-with-MAX3010X`; accessed on 2nd May 2020.
[18] STM32F4 series webpage: `https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html`; accessed on 7th May 2020.
[19] MAX30100 repository (2): `https://www.github.com/Joe-Story/GM2-Wearable-Healthcare`; accessed on 2nd May 2020.
[20] MAX30102 datasheet: `https://www.datasheets.maximintegrated.com/en/ds/MAX30102.pdf`; accessed on 2nd May 2020.
[21] MAX30102 repository: `https://www.github.com/aromring/MAX30102_by_RF`; accessed on 2nd May 2020.
[22] *Pulse Oximeter With Much Improved Precision*: `https://www.instructables.com/id/Pulse-Oximeter-With-Much-Improved-Precision/`; accessed on 2nd May 2020.
[23] MAX30102 repository (2): `https://www.github.com/catnull/Max30102Driver-For-Arduino`; accessed on 12th May 2020.
[24] MAXREFDES117# repository: `https://www.github.com/MaximIntegratedRefDesTeam/RD117_ARDUINO`; accessed on 2nd May 2020.
[25] MAXREFDES117# website: `https://www.maximintegrated.com/en/design/reference-design-center/system-board/6300.html/`; accessed on 12th May 2020.
[26] MAX3010x repository: `https://www.github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library`; accessed on 2nd May 2020.

to program the FPGA and to be run by accelerators. However, the reviewed repositories did not evaluate exhaustively the measurement accuracy achieved by the implemented algorithm. The repository referred in footnote 21 provides a clearer support documentation. The applied algorithm is consistent with the one presented in [54], not supported by an open-source implementation, as both base on autocorrelation peak detection. Therefore, this implementation is the most appropriate to be executed.

2.3 FPGA-based medical devices

Considering the targeted platform of this work, the following paragraphs review existing medical prototypes also based on FPGA logic.

Joaquinito [75] developed a wireless biosignal measurement system, using a SoC FPGA, for monitoring heart rate from ECG signal analysis, and temperature in real-time. The system included a Bluetooth Low Energy (BLE) device to transfer processed data with low-power consumption and low data rate connection, paired to a smartphone. The approach consisted of connecting both analog and digital sensors to the SoC, accelerating the processing of ECG signals recorded by three electrodes and sending short but relevant data via BLE to the user interface (the smartphone). ECG processing based on Pan–Tompkins algorithm, where the preprocessing phase composed by amplification and filtering shares some similarities with EEG. Using this method, ECG signal peaks were detected and a hardware module performed further operations to obtain heart rate. The testing results showed 100% sensitivity – that is, the system detected all ECG signal peaks – and an average of 99.7% positive predictability – meaning that 0.3% of the predictions were incorrectly classified as ECG peaks, corresponding to noise signals. The methodology of using custom hardware and accelerating signal processing improved the system performance. The goals of this work are very similar to the proposed ones in this report. Both systems aim to monitor vital signs in real-time and to process them using SoC FPGA, differing in the biometric techniques.

Knežević et al. [76] presented a methodology to detect ECG peaks and the points of maximum slope of PPG. The delay between them is also computed, and is known as pulse transit time (PPT). The system was demonstrated in a FPGA. Both signals passed through a DWT module. Then, resulting ECG signal was processed by modulus maxima, and PPG signal by derivative filtering. The hardware design comprised an Analog-Digital Converter (ADC), two processing cores for ECG and PPG, a PPT calculation circuit followed by an Universal Asynchronous Receiver/Transmitter (UART) controller. All digital modules were specified in VHSIC Hardware Description Language (VHDL) code. The processing cores were tested using two public databases, containing recorded pulses. ECG peaks were detected with an average accuracy of 97.5%. Regarding the PPG, the detection of points of maximum slope had an average accuracy of 97.1%. The contribution of this work is the development of dedicated hardware to process biometric signals. This was achieved by two parallel processing cores with reconfigurable logic. There were no mentions of software modules, so this work is a reference to bio-signal processing by custom logic accelerators.

FPGA-based works aiming at emotion identification from EEG signals are emerging in the literature.

Fang et al. [77] implemented a Convolutional Neural Network (CNN) in a Virtex-7 FPGA for emotion detection from EEG signals from 6 channels. The classifier was integrated in a complete system containing an acquisition headset and a MATLAB program for feature extraction. Two experiments were conducted, one in real-time and a second one offline using the DEAP dataset. During the real-time experiment, the system took 450 ms to detect an emotion, from the acquisition node. The offline processing of DEAP dataset resulted in a valence-arousal classification accuracy of 76.67%.

The system proposed in [77] contributes with a complete execution of the classification process. However, the system is oriented to operate in a laboratory environment, rather than targeting a wearable device for daily use. Actually, this is a gap in the literature of emotion recognition, and represents an opportunity to develop a novel FPGA-based system with this scope.

Chapter 3

Proposed biometric system

This chapter provides a complete description of the algorithms for processing PPG and EEG data, to be implemented on the proposed system. The analysis includes a profiling of those algorithms, to be used when dimensioning the final system. The selection of reference datasets containing samples of bio-signals is also addressed. Sections 3.1 and 3.2 introduce, respectively, the heart rate calculator and the emotion classifier. A high-level architecture of the proposed system is presented in Section 3.3.

3.1 Heart rate calculator using PPG

The sub-system for computing instant heart rates using the PPG technique is proposed in this section. It is intended to acquire the raw PPG signals and to process them in the node. The selection of the algorithm for handling PPG signals is described in Section 3.1.1. The working principle of the selected algorithm to calculate heart rate is detailed in Section 3.1.2, step by step. Section 3.1.3 analyses the effects of sampling frequency and sampling time on the computational cost of the algorithm.

3.1.1 Selection of the algorithm and reference datasets

The main purpose of the algorithm for processing PPG data is to compute instant heart rate values in real-time. A common approach is peak detection, from which the signal periodicity, and thus the pulse, can be retrieved. From literature analysis of Table 2.2 in Section 2.2.2 was selected the proposal by [56]. This proposal provided an open-source software implementation. From the analysis of the publicly available repositories, the footnote 21 was pointed as a good candidate to be implemented in this system. In fact, this repository provided a C implementation of the algorithm, a file containing samples of raw PPG signals and a complete supporting documentation. The proposal [56], by the time of analysis, required code modifications to return instant heart rate as desired, although being written in C language. The footnote 21 alternative turned to be a more practical implementation to be divided into blocks and adapted into a hybrid HW/SW design. Moreover, the code was exhaustively commented. Thus, this ready-to-use solution was the most preferable. Next section describes the operating principle of the selected algorithm.

The provision of datasets was an exclusion factor on algorithm selection. In fact, few solutions provided data to test their implementations. The repository of Footnote 21 included a single file containing 100 samples of red and infra-red channels data, sampled at 25 Hz. Given this scenario, it is reasonable to select external datasets to validate the final design solution of heart rate module. The search for data obtained by `MAX30100` sensor did not return meaningful results. Then, some PPG datasets were found, regardless the sensor used. However, none of them consisted of raw data, but preprocessed. For instance, the 2015 IEEE SP Cup dataset provided recorded PPG data of eight subjects performing different activities, such as walking and running. This database can be used to validate the design of the core responsible for handling PPG signals, described in Chapter 4. Due to the lack of raw PPG data, custom signals were recorded using a `MAX30100` sensor connected to an Arduino. Additional datasets were synthesized from the real ones applying multiple transformations on a specific segment, such as multiplication by a scalar and upsampling or downsampling. Analysing signals with different amplitudes and frequencies is necessary to define the variable's size in software and the signal's wordlength in hardware.

3.1.2 Algorithm description

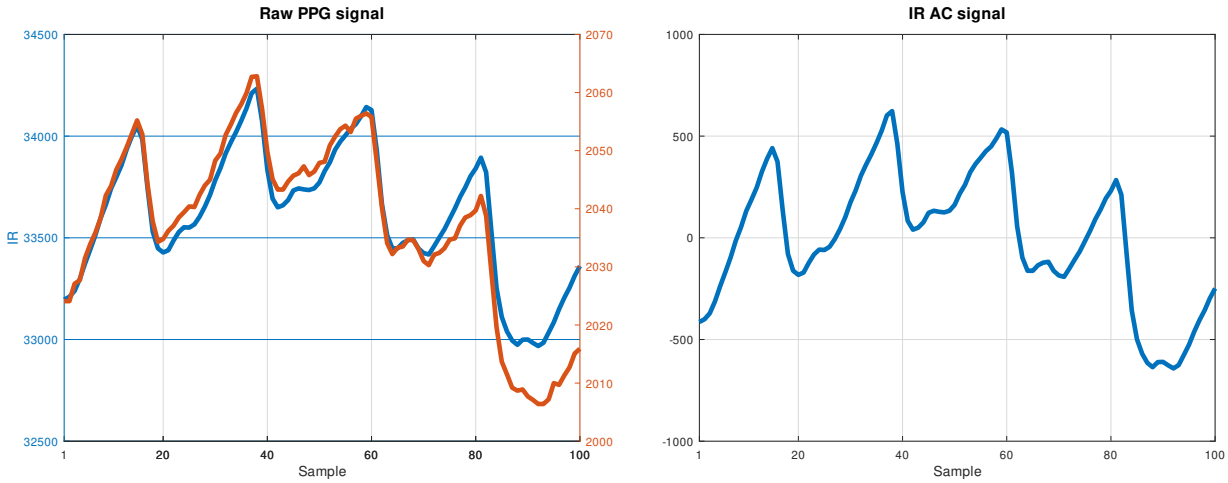
The heart rate calculator algorithm operates over two channels of PPG signal, the red (RED) and the infra-red (IR), probed by distinct LEDs. The algorithm comprises two main stages: preprocessing and periodicity search. The computational operations included in the first one are the following:

1. DC mean calculation: a loop over a buffer containing N signal samples computes the sum of their values, and then the average by dividing the accumulated sum by N;
2. DC mean subtraction: the computed average is subtracted from each channel sample, by an iterative loop;
3. linear regression calculation: a dot product between the sample set and corresponding shifted sample indexes is computed, then divided by a constant;
4. linear regression subtraction: the computed value is multiplied by each shifted sample indexes and subtracted from each channel sample;
5. mean square calculation: the sum square of all sample values is calculated and divided by N;
6. Pearson correlation calculation: a dot product between both channels' samples is determined and then divided by N.

Once these tasks are concluded, the periodicity search over the preprocessed signal begins. This stage is mainly performed by a function called autocorrelation.

For a clearer explanation and better understanding, the steps involved in the heart rate calculation algorithm are next detailed and supported by an example. The example is a 100-sample buffer, provided by the author of the implementation, that corresponds to a good quality signal. Two-channel data, RED and IR, was sampled with 25 Hz using `MAX30102` sensor, and is illustrated in Figure 3.1a. Next, the processing tasks that modify the raw input data are explained, focusing on the IR channel, from which the heart rate is computed.

The first step is to remove the DC component from the signal, keeping only the AC, because this method concentrates on the signal fluctuations, rather than its amplitude. As shown in Section 2.2, heart rate can be identified analysing AC component. The result depicts on Figure 3.1b.



(a) Raw PPG signal.

(b) AC component.

Figure 3.1: Step 1 – AC component of the PPG signal.

The second step consists of subtracting the first-order component, that is, the linear trend of the signal. The value of the slope β is obtained using Equation 3.1,

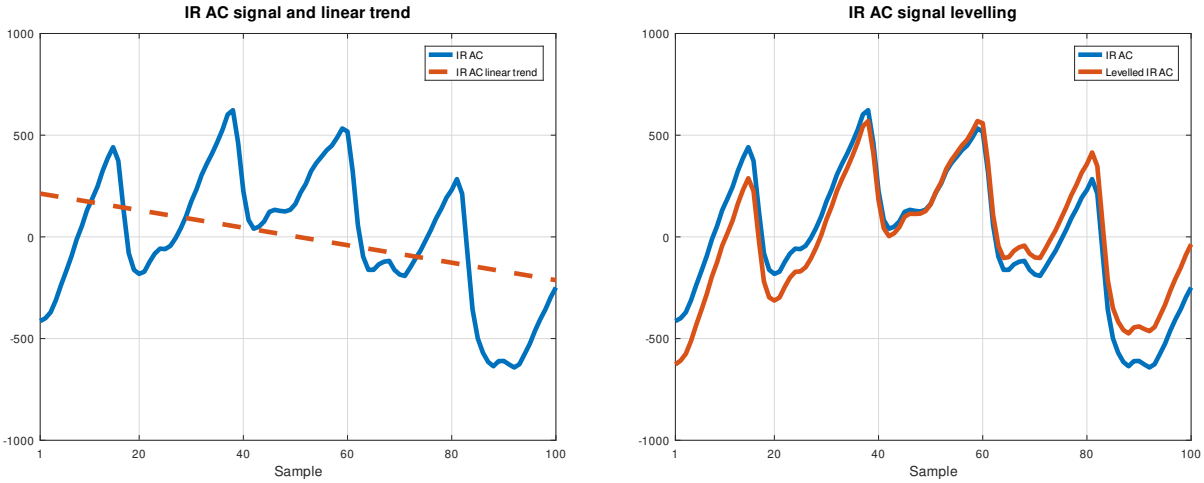
$$\beta = \frac{\sum_{t=-t_{mean}}^{t_{mean}} t \cdot y_t}{\sum_{t=-t_{mean}}^{t_{mean}} t^2} \tag{3.1}$$

where t is the sample index and y_t the sample value. It is worth to notice that indexes start at $t = -t_{mean}$, rather than at origin $t = 0$. This left shift by t_{mean}, defined by Equation 3.2, allows to center and equidistantly display all N samples, so that, in this example, their indexes range $\{-49.5, -48.5, ..., 49.5\}$, instead of $\{0, 1, ..., 99\}$.

$$t_{mean} = \frac{N-1}{2} \tag{3.2}$$

Figure 3.2a clarifies this centring procedure, plotting the linear trend with the computed slope of the sample set, over the AC signal. Figure 3.2b includes the levelled signal – after removing first-order component – and the original one for comparison. This levelling process is useful in further steps to detect peaks correctly.

At this stage, the preprocessing tasks over the collected bio-signals are completed. Then, a first quality metric is evaluated, known as the Pearson correlation. This coefficient denotes the linear association between two variables – in this case, RED and IR channels. Graphically, measures the feasibility of drawing a line to best fit both data. Values range $[-1, 1]$, where -1 and 1 mean, respectively, the strongest negative and positive associations, that is, a perfect linear fit with negative and positive slopes. The absence of linear correlation corresponds to a value of 0. A correlation besides these key values means a linear association that does not fit all data. In short, the closer is the absolute value of Pearson correlation to 1, the more linear is the association between two variables. The Pearson correlation



(a) Linear trend of PPG signal.

(b) PPG signal levelling.

Figure 3.2: Step 2 – PPG linear trend removal.

coefficient r is calculated using Equation 3.3,

$$r = \frac{\sum_{n=1}^{N} (x_n - \bar{x})(y_n - \bar{y})}{\sqrt{\sum_{n=1}^{N} (x_n - \bar{x})^2 \sum_{n=1}^{N} (y_n - \bar{y})^2}} \tag{3.3}$$

where:

- N is the number of samples;
- x_n denotes a preprocessed IR sample;
- $\bar{x}$ is the mean value of preprocessed IR samples, which is 0, because of DC removal;
- y_n denotes a preprocessed RED sample;
- $\bar{y}$ is the mean value of preprocessed RED samples, which is also 0.

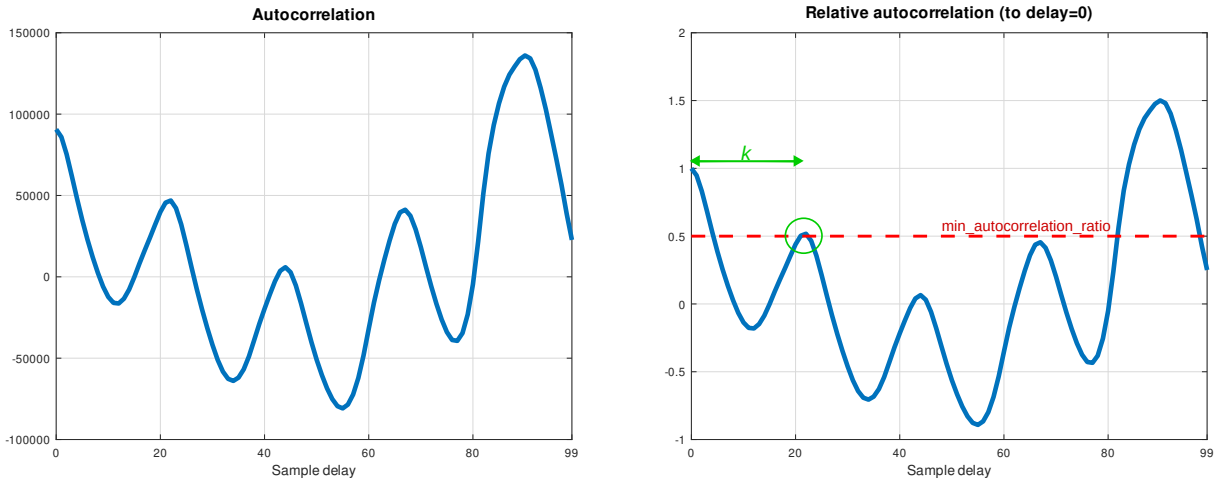
According to the implementation, a good quality signal must have a Pearson correlation equal or greater than 0.8. Otherwise, the sample set is discarded and a new collection is recorded.

From this stage, the algorithm initiates an iterative process of finding the signal periodicity, via peak detection. In this step underlies the concept of autocorrelation, a function that allows to identify patterns in a signal. More specifically, it consists of the correlation – or similarity – between a signal and its delayed copy. As such, taking into account that PPG is a periodic signal, this property is advantageous to determine heart rate, specially in noisy environments, like probing data using bio-sensors. Mathematically, the autocorrelation R at a given delay m is the sum of the products between each sample and its delayed one, over all N samples of set X, shown in Equation 3.4.

$$R(m) = \sum_{n=1}^{N} X(n)X(n+m) \tag{3.4}$$

Figure 3.3a shows the result of computing the values of autocorrelation for all possible sample delays, from 0 to $N-1$, where $N = 100$ in the example of Figure 3.1a. Then, all values of this graphic are normalized to the autocorrelation at delay $N = 0$, as depicted in Figure 3.3b and referred as relative

autocorrelation. Analysing the curve, one can identify local maximums alternating with local minimums,



(a) Autocorrelation values.

(b) Relative autocorrelation values.

Figure 3.3: Step 3 – Autocorrelation of PPG signal for different delays.

which correspond to constructive and destructive interferences, respectively. In other words, shifting the signal gradually to the right, autocorrelation decreases to a local minimum, because the signal and its shifted copy are in phase opposition. As the copy moves to the right, autocorrelation increases again because of signal periodicity, until both waveforms are again in phase. The shift k corresponding to the index of the closest local maximum matches the number of samples containing a complete heart beat. This peak, marked in Figure 3.3b by green, is sufficient to determine PPG signal periodicity. Therefore, pulse period T_{HR} is calculated multiplying the number of samples k by the time gap between two samples, that is, sample period T_s. This way, heart rate is the inverse of pulse period, represented in Equation 3.5,

$$\mathrm{HR}_{\mathrm{bps}} = \frac{1}{T_{\mathrm{HR}}} = \frac{1}{k \times T_s} = \frac{1}{k \times \frac{1}{f_s}} = \frac{f_s}{k} \tag{3.5}$$

where f_s denotes the sampling rate, inverse of T_s. This result corresponds to beats per second (bps), so beats per minute (bpm) are given by Equation 3.6.

$$\mathrm{HR}_{\mathrm{bpm}} = \frac{f_s \times 60}{k} \tag{3.6}$$

Finding the index k that maximizes the first peak of autocorrelation function assumes premises described next. First, the definition of minimum and maximum valid heart rates to mark out a finite range of acceptable pulse values. This aspect prevents the algorithm from returning unreliable ones. Furthermore, given that the highest heart rate corresponds to the lowest period, the index of autocorrelation peak is equal or greater than the lowest period. In other words, the local maximum – and thus the signal periodicity – may be searched starting from the sample index that matches the lowest period possible assumed. For this reason, in the beginning of a real-time execution of the algorithm, the periodicity search of the first sample set is initialized following this criteria. Subsequently, the algorithm is adaptive to the last computed heart rate, where search is focused on finding the shifted autocorrelation peak in the

neighbourhood (adaptation phase). Thus, the initialization occurs once during a continuous execution.

Besides Pearson correlation, there is a second quality metric known as minimum autocorrelation ratio. This metric validates the signal buffer sampled by the PPG sensor. Once the autocorrelation peak is determined, the algorithm evaluates whether its value is equal or greater than 50% of the reference (delay 0). Otherwise, that buffer is considered as being too much corrupted with noise and thus discarded. This criteria prevents the detection of false peaks caused by noise. The minimum threshold is marked by red in the example of Figure 3.3b.

3.1.3 Software application profiling

As mentioned in the previous section, the PPG algorithm features two main tasks – preprocessing and periodicity search. The computational operations regarding preprocessing phase are applied to both IR and RED channels. Every loop operates over both channels concurrently.

The periodicity search is an iterative process where computational cost is uncertain. Depending on the signal characteristics, the autocorrelation function is called a different number of times. The signal variation and the fulfilment of quality criteria influence as well. For instance, according to the algorithm, this function is more often called during the processing of lower pulse rate signals. When periodicity is initialized, this happens because of the starting point of the search, which matches the distance equivalent to the lowest valid period. Apart from this stage, the starting point retrieves the index of previous iteration's peak, acting as an adaptive algorithm. Assuming a sliding-window processing system, it is expected that the frequency of PPG signal does not vary much after each buffer analysis. Thus, the previous peak is used as reference to the next iteration.

However, autocorrelation function calls are countable during initialization of periodic search. This number depends on two factors: range of accepted values for heart rate and sampling frequency. Taking into account that heart rate is computed from a sample index (Equation 3.6), which is a natural number, the results are discrete. Therefore, the sampling rate determines the resolution of the final result. On the other hand, the heart rate values range influences the set of samples from which pulse can be calculated. For this reason, the maximum number of autocorrelation function calls varies with this range. Trivially, the wider the variety, the more often may occur the function calls. From heart rate definition (Equation 3.6), the number of indexes contained inside the interval of acceptable pulse values, and thus the maximum number of autocorrelation calls, is given by Equation 3.8:

$$n = \operatorname{int}\left[\frac{f_s \times 60}{\mathrm{hr}_{\min}}\right] - \operatorname{int}\left[\frac{f_s \times 60}{\mathrm{hr}_{\max}}\right] + 1 \tag{3.7}$$

Considering heart rates ranging from $\mathrm{hr}_{\min} = 40$ to $\mathrm{hr}_{\max} = 180$ bpm, n varies with sampling rate f_s according to Table 3.1. There, $i_{\mathrm{hr}_{\min}}$ and $i_{\mathrm{hr}_{\max}}$ denote the index corresponding to $\mathrm{hr}_{\min}$ and $\mathrm{hr}_{\max}$, respectively, such that:

$$n = i_{\mathrm{hr}_{\min}} - i_{\mathrm{hr}_{\max}} + 1 \tag{3.8}$$

The worst case scenario regarding the periodicity search is the computation of all n values. This

Table 3.1: Variation of number of indexes, n, with sampling frequency, f_s, assuming heart rates ranging from $\mathrm{hr}_{\mathrm{min}} = 40$ to $\mathrm{hr}_{\mathrm{max}} = 180$ bpm.

f_s (Hz)	25	50	100	200	500	1000
$i_{\mathrm{hr}_{\mathrm{min}}}$	37	75	150	300	750	1500
$i_{\mathrm{hr}_{\mathrm{max}}}$	8	16	33	66	166	333
n	30	60	118	235	585	1168

means that the search assumes, at first, an autocorrelation peak corresponding to the highest pulse rate, $i_{\mathrm{hr}_{\mathrm{max}}}$. Then, given that the peak is in fact located at $i_{\mathrm{hr}_{\mathrm{max}}}$, the algorithm calculates autocorrelation n times, according to Equation 3.8. This worst case may occur during the initialization, if the user presents a high heart rate and the algorithm assumes the lowest rate. Also, it may occur, with less probability, between the processing of two distinct buffers during the adaptation phase.

In this implementation, 141 different integer heart rate values are assumed, ranging from 40 and 180 bpm. Observing Table 3.1, it is interesting to notice that, at least, sampling frequencies up to 100 Hz do not cover all possible values of the considered range, because, in those cases, the number of indexes is less than the number of distinct heart rates ($n < 141$). However, this does not imply that remaining frequencies necessarily cover all 141 possibilities. Figure 3.4 indicates which pulse rates are actually missed when sampling with different rates. It shows that lower frequencies miss most heart rate values.

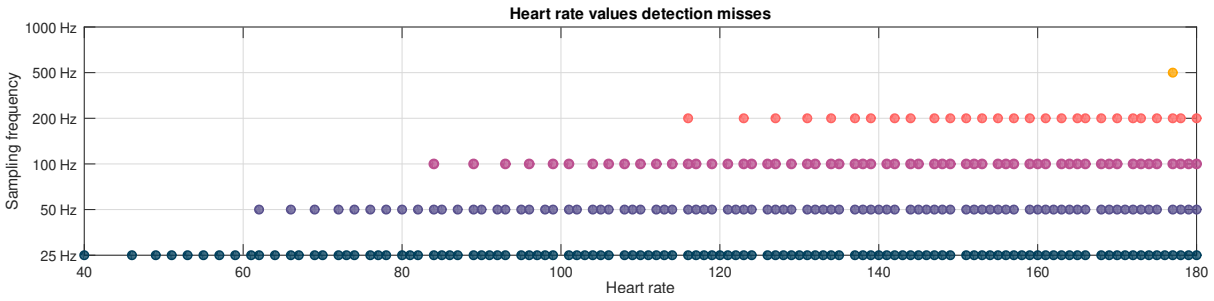


Figure 3.4: Heart rate detection misses varying with sampling frequency.

In other words, using higher frequencies leads to more precise final results. The error of the final result is significantly higher when using low sample rates, as detection misses occur more often. On the other hand, higher frequencies allow to determine the index of autocorrelation peak more precisely. Intuitively and according to Table 3.1, a PPG signal sampled with high frequency rate has more samples which, by turn, are closer to each other. Therefore, n increases, the periodicity search takes longer but the calculation of Equation 3.6 is more accurate. The domain of k is wider in these cases, such that the final results are more diversified. For instance, at $f_s = 500$ Hz, the first and only detection miss occurs near 180 bpm.

Previous paragraphs exposed the impact of sampling rate. Another parameter that should be considered during system parametrization is sampling time. As seen during the algorithm analysis, the first autocorrelation peak is the main concern. Therefore, manipulating buffers holding multiple PPG waves does not improve much performance. Two waveforms could be enough to compute autocorrelation at a delay corresponding to a complete heart beat, and thus determine its periodicity. To assure that at least two pulses are actually encapsulated, three waveforms are considered. As such, depending on the

instant heart frequency, a pre-dimensioned buffer includes an unknown number of pulses. Assuming a minimum acceptable heart rate $\mathrm{hr}_{\mathrm{min}}$, it is possible to estimate how much total sample time is needed to guarantee three beats inside the buffer. Thus, the duration of three complete heart beats $3T_{\mathrm{HR}_{\mathrm{min}}}$ is deducted from Equation 3.9.

$$\mathrm{hr}_{\mathrm{bps}} = \frac{\mathrm{hr}_{\mathrm{min}}}{60} \implies T_{\mathrm{HR}_{\mathrm{min}}} = \frac{1}{\mathrm{hr}_{\mathrm{bps}}} \implies 3T_{\mathrm{HR}_{\mathrm{min}}} = \frac{3 \times 60}{\mathrm{hr}_{\mathrm{min}}} \tag{3.9}$$

Admitting a minimum $\mathrm{hr}_{\mathrm{min}} = 40\ \mathrm{bpm}$, it is required to sample PPG during 4.5 seconds.

3.2 Emotion detector from EEG

The biometric sub-system to detect human emotions is proposed in this section. The goal is to find a classifier that maps EEG signal attributes into emotions. The process of selecting the suitable classifier and reference datasets is described in Section 3.2.1. Due to the scope of this work, the emotion detector targets the classification of EEG attributes that have been obtained after preprocessing raw EEG signals. Such preprocessing tasks include removing artifacts, caused by eye blinking and muscle movement, through blind source separation. This technique separates signal components from a signal mixture. EEG signals are also downsampled, filtered and averaged to a common reference. As a result, these steps reduce the signals to the data containing the essential information and average out the model error associated with EEG channels. Signal attributes, also known as features, are extracted by computing the standard deviation of the five frequency bands of each EEG channel. The standard deviation represents the power spectrum variations of EEG waves over time. Finally, features are normalized to a common scale, since they may assume a wide range of values. The purpose of this procedure is explained next in more detail. The normalized feature set is the input data of the classifier, whose working principle is described in Section 3.2.2. A preliminary software implementation of the selected classifier is presented in Section 3.2.3. Section 3.2.4 addresses the dimensioning of the input data and estimates the computational cost of the emotion detection process.

3.2.1 Selection of the classifier and classification datasets

The EEG algorithm targets the implementation of a classifier to detect human emotions. This algorithm corresponds to the last stage of the cycle of EEG signal processing. This cycle covers complex tasks, such as raw signal acquisition, signal enhancement, feature extraction and classification. The EEG classifier is provided the relevant information for emotion inference. It is desirable to have an open-source software solution describing the targeted goal. Such implementation helps to understand the flow of EEG data and the complexity of the processing tasks. Python and MATLAB applications addressing specific steps of EEG processing algorithms were identified in the literature and public repositories. These included platform-specific signal acquisition drivers – such as commercial headsets –, preprocessing functions and classifiers. Another faced issue was the lack of documentation supporting the

few available open-source repositories. Regarding EEG classification, a KNN classifier[1] was selected, aiming the prediction of emotional state given preprocessed EEG data. The analysis of the classifier's working principle allowed to translate this implementation into a low-level language version, using C. A comprehensive description of its operation is provided in the next section.

The EEG data supported by this classifier comprises up to 32 electrode channels. In fact, the preprocessed version of DEAP dataset [34] is used as data input reference. Commonly cited by EEG-related researches, this set provides both raw and processed signals acquired by 32 electrodes, from 32 subjects watching different music videos. Those signals are appended each subject's ratings on their perceived emotions during the experiments. The ratings include, among others, the two dimensions of Russell's emotion map – valence and arousal –, described in Chapter 2. The number of electrode channels is impractical for a wearable device. However, dimensioning the classifier this way is advantageous to provide a more customizable classifier for further applications.

3.2.2 KNN classifier description

The goal of using a classifier is to predict an emotion (class) given a set of EEG signal attributes (features), taking as a reference already known relations between those attributes and emotions. The predicted class represents the emotion that best fits the EEG features analysed.

K-nearest neighbours (KNN) algorithm is a supervised learning classifier, meaning that a training set containing multiple input-output data observations determines the inference of the output of an unseen input object, the test set. In practice, KNN maps objects into images given a collection of previously memorized training object-image pairs (instances). The principle of KNN is to find the K closest memorized instances to the recently observed set of features. In other words, to find the known instances that are the most similar to the feature set to be classified. Once the most suitable instances are assessed, the emotion classes each instance is associated with are registered. The modal class is declared as the predicted emotion of the queried test set.

The process of measuring the similarity of training and test sets is the distance between their points, considering that feature sets can be viewed as arrays. This KNN version uses the method of Canberra distance, mathematically defined in Equation 3.10 as d_C, where u and v denote two points in n-dimensional space.

$$d_C(u, v) = \sum_{i=1}^{n} \frac{|u_i - v_i|}{|u_i| + |v_i|} \tag{3.10}$$

The input objects of the classifier are EEG features that have been normalized to $[0, 1]$. This way, the distances between test and training instances are not biased by a dominant feature. Normalization methods vary, but a common approach is the rescaling from minimum and maximum values, as stated in Equation 3.11. There, x represents the whole feature set to be normalized; x_{ij} is the j-th element of the i-th array of EEG features; f_{ij} denotes a normalized EEG feature. The equation applies a linear

[1] Emotion Detection from EEG repository: `https://www.github.com/shubhe25p/Emotion-detection-from-EEG`; accessed on 15th January 2021.

transformation to the vector space containing the set of EEG features.

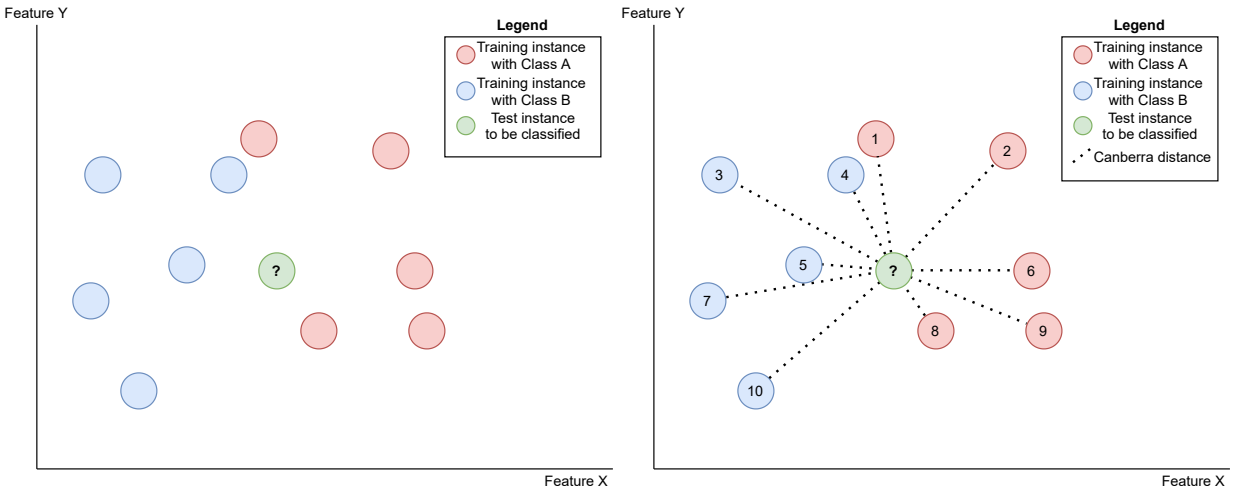
$$f_{ij} = \frac{x_{ij} - \min(x)}{\max(x) - \min(x)} \tag{3.11}$$

In short, to classify an unobserved test instance, the algorithm determines its K most similar instances from the observed training set. This step implies two tasks: the computation of Canberra distance, d_C, between the test and every training instances, and then sorting those distances to obtain the K shortest distances. The K training instances that present more similarity with the test instance correspond to the K shortest Canberra distances. Wider the training set, more Canberra distances are calculated and compared, and thus higher is the computational cost. Once the K shortest Canberra distances are found, the corresponding K training instances are selected to proceed with the algorithm. The next step is to register the emotion classes associated with the selected K instances, finding the most common class. In other words, the K training instances vote for a class. The most voted class – modal class – determines the emotion prediction output.

The calculation of Canberra distances is independent from the images (classes) of the training instances. Distances are obtained from the differences between instances' features. The distances sorting assumes that training instances are equally weighted, meaning that each instance contributes to the voting system with an unitary vote. The modal class is taken as the predicted class because classes are discrete, rather than continuous values. Emotion values take action during the last step of counting the class votes associated with each instance.

Illustrative example

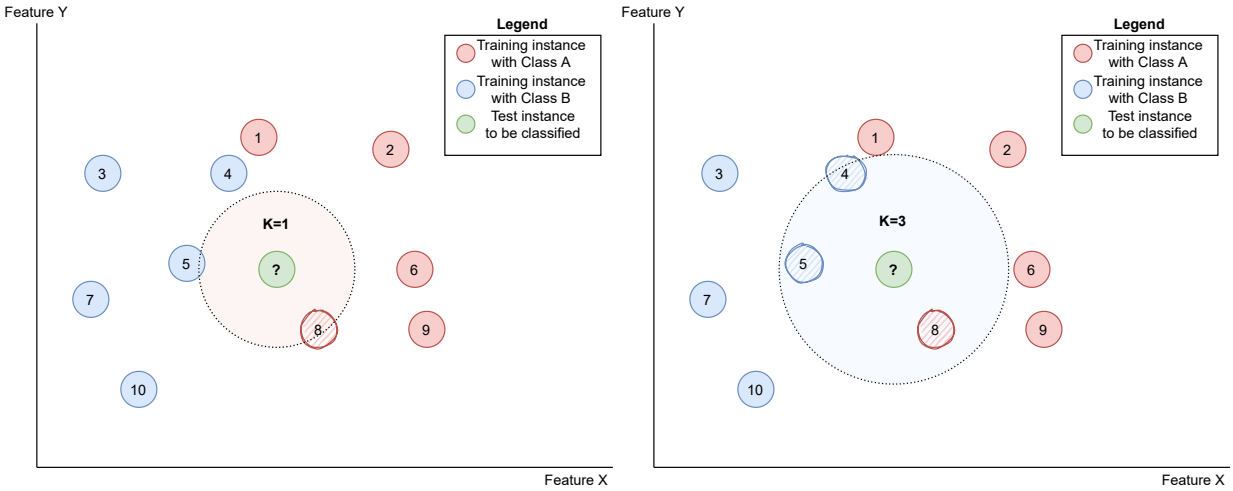
This section provides an example to visualize the process of emotion classification. The KNN must possess a set of instances from which classification decisions are made. This set (training set) is represented in Figure 3.5a. For simplicity, this example only considers two feature dimensions (X and Y) and two classes (A and B). The classifier receives test instances to classify, that is, to assign a class given the information provided by the training set. Figure 3.5b shows an abstract representation of the Canberra distances computed for each training instance, in respect to a received test instance. This figure includes the numbering of training instances, to keep track of their indexes when sorting distances. Once distances are computed, the next step is to consider the decision boundary for classification. The value of K determines the amount of instances participating in the prediction process. Moreover, K defines the number of shortest distances to be retrieved. Figure 3.6a shows a naive example of using a single neighbour for classification ($K = 1$). The closest training instance matches the predicted class (Class A). Figure 3.6b assumes $K = 3$, thus the decision region is wider as three training instances are considered to assess the class. This classification decision can be seen as a voting process of three voters, where Class A receives one vote and Class B receives two. The most voted class is Class B.



(a) Representation of training set and the test instance to classify.

(b) Calculation of Canberra distances.

Figure 3.5: Example of an abstract representation of KNN's training set and test instance. Computation of Canberra distances between training and test instances.



(a) Classification decision assuming $K = 1$.

(b) Classification decision assuming $K = 3$.

Figure 3.6: Classification of the test instance using different values of K. When $K = 1$, the nearest training instance determines the predicted class (Class A). When $K = 3$, the three nearest neighbours vote for their registered class. In this example, Class A has 1 vote and Class B receives 2 votes, so the predicted class is Class B.

3.2.3 Software implementation

The scarce open-source C/C++ solutions describing an EEG classifier led to the need of implementing a custom version, taking as a reference repositories based on programming languages other than C. The chosen implementation uses Python and is available on a public repository. This section explains the main issues regarding the implementation of the KNN classifier.

Meaning of K parameter

To implement the KNN classifier, it is necessary to find the optimal value for its hyperparameter K. This parameter determines the number of training instances that are voting for the final predicted class. Typically, K takes odd values to avoid tied votes. Finding the optimal K implies to balance the effect

of bias and variance of training instances on classification decisions. Bias corresponds to the error associated with the inclusion of multiple training instances to predict a class. Errors due to bias mean that the decision took misleading assumptions from the training set. Variance is the error related to the fluctuations of class values observed at training. Higher values of K lead to a scenario where votes average the class values observed in the training set. The classification decision is thus significantly biased by the K elements, where variance is low due to the high amount of voters. A low K value, with few voters taken into account, leads to a high variance voting, highly sensitive to their class values, with low bias from the training set.

Finding K is an essential task to be considered by the field of Bioinformatics, when developing algorithms for manipulating biological data. The original implementation of the selected KNN classifier considers $K = 3$. However, its supporting documentation does not include the reasons behind such decision. To implement a generic classifier, capable of determining a broad range of closest instances, it was settled that the classifier would be designed with a greater value of K. The reference database (DEAP dataset) was used to identify suitable K values for accurate classification. The dataset composed by 1280 instances was split into two slices: a training set containing 1024 entries (80%) and a test set comprising the remaining 256 instances (20%). Multiple odd values assigned to K were tested and a maximum accuracy of 41% for classifying emotions was obtained using $K = 21$. Therefore, the classifier algorithm is set to assess up to 21 nearest neighbours of any test instance. Moreover, the main goal of tuning the K parameter is to design an architecture prepared to execute the KNN algorithm under a specific range of K values, without neglecting the classification accuracy. This methodology assumes that the effectiveness of the algorithm is previously maximized during algorithm specification.

Implementation in C language

The software description implementing the KNN classifier uses Python and includes the instructions for extracting EEG features from preprocessed signals, besides the KNN classifier. The repository takes as reference input data the preprocessed version of the DEAP database. The classifier was isolated from the complete implementation to be mapped into an equivalent C language version. A software-only version of the KNN classifier was written in C from scratch featuring the tuned value of $K = 21$. During the writing process it was made an effort to define instruction sequences potentially synthesizable into Hardware Description Language (HDL). This caution facilitates the further step of designing dedicated IP cores implementing the targeted algorithm.

3.2.4 Software application profiling

This section examines the algorithm in terms of complexity, performance and data handling, foreseeing the upcoming implementation of an IP core.

The EEG classifier is expecting as input feature sets containing information about power spectrum variations of each brain wave channel, for the major frequency bands. Considering 32 electrode channels and 5 frequency bands, each EEG trial comprises $32 \times 5 = 160$ features. Each feature value ranges

$[0, 1]$. Both test and training instances follow this protocol. Additionally, training trials are appended a value representing the expected emotion class. On the other hand, the emotion class of test trials is exclusively predicted by the classifier. Regarding data volume handled by the classifier, it is assumed a training set similar to the presented by DEAP database. This means that 1024 instances are memorized and an undefined number of test trials can be passed to predict emotions.

The classifier involves three steps – Canberra distances calculation, their corresponding sort and the assessment of the modal class. Next paragraphs discuss their computation complexity.

Evaluating the shortest distances

During the process of predicting a single emotion, the computation of Canberra distances is performed as many times as the training set size. Therefore, 1024 Canberra distances are determined. A single test instance, containing 160 features, is paired to each one of the 1024 training instances, to compute a Canberra distance.

The following step of finding the shortest values from the computed 1024 distances involves the insertion sort algorithm. Such algorithm is advantageous to a solution where elements may be sorted as soon as their values are successfully computed. The principle of insertion sort is to maintain an array of elements sorted after each algorithm iteration. This means that an insertion is preceded by the assessment of the precise position in the array where it will occur. In the best-case scenario, the complexity of the algorithm is $O(n)$, which means it has a linear running time. This case corresponds to the state where the array is initially sorted. During the execution, the algorithm scans the array without performing any insertions. One the other hand, the worst-case scenario corresponds to sorting an array that is initially sorted in the opposite (ascending) order, with a quadratic running time ($O(n^2)$). Each insertion takes as many element comparisons as the amount of sorted elements.

Finally, it should be noted that, during the assessment of the shortest distances, the algorithm keeps track of the arrival order of the computed distances to the sorting stage. This means that each distance is assigned an index, ranging $[0, 1023]$, corresponding to its position in the training set. This procedure is fundamental to translate the nearest neighbours into votes representing emotion classes.

Emotion prediction: the discretization of the emotion classes

The emotion classification problem handles discrete classes, reducing the emotion domain to five emotions. Recalling Russell's cartesian model introduced in the previous chapter, an emotion is a combination of valence and arousal. Following the methodology proposed by the author of the Python KNN classifier, valence and arousal dimensions are labelled according to their intensity level – low, medium and high. These levels are equally spaced. The combination of the different dimension intensities leads to nine potential emotions. Assuming that an emotion containing a dimension with medium intensity is labelled as a neutral emotion, five distinct emotions can be mapped, according to the graphical representation of Figure 3.7.

The mapping proposed in Figure 3.7 is debatable. On one hand, the neutral emotion prevails over the

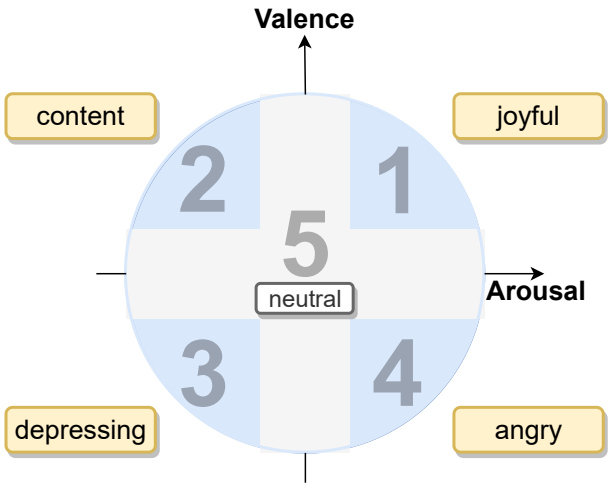


Figure 3.7: Graphical representation of the five-emotion mapping. Blue area represents four different emotion domains. Gray area corresponds to the neutral emotion.

remaining emotions. On the other hand, the map distinguishes few emotions considering the complexity of human behaviour. Nevertheless, the classifier proposes to perform emotion detection within a range of five labels. In the literature there are two main types of EEG systems. One uses typically up to six EEG channels to detect four main emotions, corresponding to the four quadrants of Russell's model. More complete systems use over thirty channels, being able to detect more specific emotions. Detecting emotions with higher discretization level, that is, distinguishing very detailed emotions, decreases the prediction accuracy of the classifier. In the architectural point of view, once the discretization level is set, the definition of valence and arousal boundaries is not critical. One purpose of this work is to implement the EEG classifier in a generic way, such that it can be applied in different biomedical contexts.

Emotion prediction by the KNN classifier consists of a decision based on evidences of the training set. An input test instance is assigned an emotion class after its comparison to the training instances known beforehand. This comparison is materialized by the distances between instance neighbours. Instances are identifiable by their order (index) in the training set. Once the most similar instances are evaluated, an auxiliary array registers their respective indexes. Then, the emotion classes associated to the selected instances are assessed. These classes act as votes for a specific emotion label. The final stage of the algorithm finds the most voted class. The modal class is declared as the predicted emotion of the input test instance.

3.3 High-level HW/SW architecture

The main goal of the thesis is to develop two IP cores dedicated to processing bio-signals. The target platform selected to implement such cores is the Zynq-7010 SoC, a SoC FPGA. This technology combines both software and hardware programmability, provided by a dual-core ARM processor and

FPGA, respectively. The processor is located inside the processing system. The FPGA comprises programmable logic blocks, namely:

- flip-flops (FF), that work as a simple storage unit, and alternate between two stable states;
- block RAMs (BRAM), dual-port random-access memory (RAM) modules that may store large sets of data;
- look-up tables (LUT), which are small RAMs that store the truth table of a logical function;
- digital signal processor (DSP) blocks, that correspond to arithmetic logic units (ALU) containing a chain of three different blocks (adders and multiplier), used to implement arithmetic functions.

These primitives can be configured and combined into more complex circuits, resulting in the IP cores. The HW/SW design offers multiple advantages over microcontroller solutions. For instance, it adds hardware acceleration to process large sets of data and optimizes the performance-per-watt ratio.

The methodology to be followed to design the IP cores starts by running software implementations reviewed and selected in Chapter 2 to obtain reference results for each bio-sensor. More specifically, preprocessed EEG features are classified by KNN, and PPG signals are processed by the peak detection algorithm, as explained in the previous sections of this chapter. Software implementations of the selected algorithms will be tested using signals available at public databases. This procedure allows to draw the software profiling, which includes an analysis of algorithm complexity, sequence of function calls and data structures dimensioning. Then, part of those algorithms is going to be implemented into hardware, according to the previous profiling. The implementation is optimized in terms of latency, throughput and hardware resources. The designed cores will be applied to accelerate the computations of signal processing, taking advantage of SoC FPGA to improve the system performance. The development of IP cores is done using Xilinx's Vitis Unified Software Platform[2], a package that includes three software tools:

- Vivado High-Level Synthesis (HLS)[3], where algorithm functionalities are implemented in high-level descriptions written in C/C++, optimized and interpreted as hardware; the implementation is converted into a Register Transfer Level (RTL) circuit and exported by the HLS tool into an IP core; the RTL instance can be assembled into a design containing multiple blocks; in short, this tool simplifies the process of designing a hardware circuit by providing an abstraction layer to the designer;
- Vivado Design Suite, that allows to integrate and connect multiple IP cores and other functional blocks to the processing system; the complete hardware design can be exported into a Xilinx Support Archive (XSA) file, customized to a specific target platform, carrying the necessary information to create embedded software;
- Vitis IDE, used to develop embedded software targeting the created hardware design, and to deploy such application to a development board.

The high-level architecture of the proposed system is depicted in Figure 3.8. It includes an abstraction of the target platform, Zynq-7010 SoC, from which PS and PL regions are distinguishable. The

[2]Vitis Unified Software Platform webpage: `https://www.xilinx.com/products/design-tools/vitis.html`; accessed on 1st June 2020.

[3]Vivado High-Level Synthesis webpage: `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`; accessed on 1st June 2020.

developed IP cores are incorporated inside PL and connected to the PS through AXI buses. A BLE connection is included, to enable the transmission of processed data to an external device, where real-time measurements may be communicated. A Double Data Rate (DDR) block represents an external memory where auxiliary information may be stored. Once the IP cores are designed, the next step will be to consider the development board and the potential bio-sensors that will collect real data. An EEG sensor collects analog signals, so AXI buses are required to connect the EEG device to the XADC module, the Zynq's ADC. A digital PPG sensor can be directly linked to the PS by Inter-Integrated Circuit (I²C). This protocol uses a bidirectional bus with two signals: Serial Data Line (SDA) and Serial Clock Line (SCL).

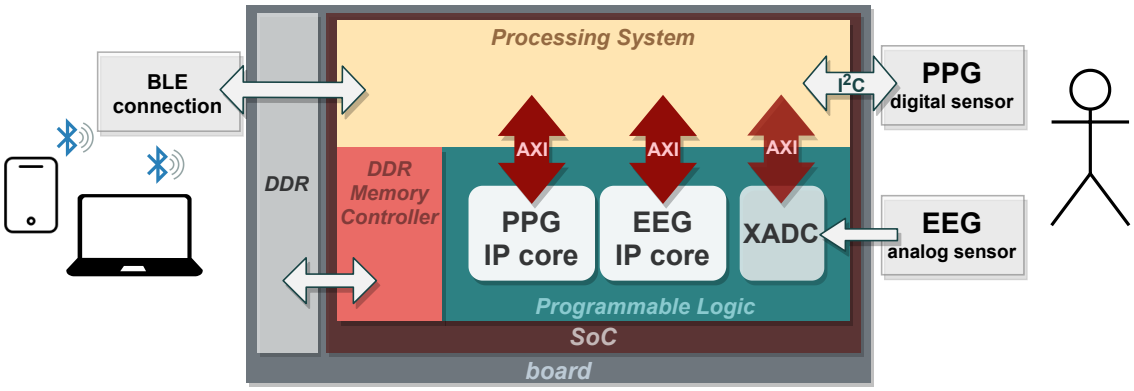


Figure 3.8: Proposed system implementation.

The algorithm analysis described in the previous sections supports the parametrization of the proposed system. Regarding the PPG IP core, the sampling frequency selected is 200 Hz. The algorithm profiling demonstrated that, at this frequency, the system could detect distinct heart rates continuously ranging between 40 and 120 bpm, approximately. Greater pulse rates up to 180 bpm are detectable, without any exception, using higher sampling frequency of 500 Hz. Given that normal heart rates range between 60 and 100 bpm, 200 Hz is sufficient in the context of the application. Section 3.1.3 showed that 4.5 seconds of PPG signal are required to encapsulate at least three complete heart beats inside a buffer. Using the selected sampling frequency, this duration corresponds to acquiring $200 \times 4.5 = 900$ samples. Regarding the EEG core, it is aimed to implement the EEG classifier. An external memory is required to store the training set containing 1024 instances known beforehand. These are compared to unseen EEG features to identify emotions, executing the detection algorithm implemented by the core. The classifier is prepared to receive instances comprising 160 features ranging $[0, 1]$. This amount corresponds to information related to 32 electrodes.

Chapter 4

PPG and EEG IP cores

The proposed system has as foundation two IP cores for processing the signals acquired by bio-sensors in real-time. These cores, denoted by ”PPG IP core” and ”EEG IP core”, take action in the assessment of the user's heart rate and emotional state, respectively. This chapter describes the process behind the development of these cores, using Xilinx's Vitis tools.

4.1 Development process and design techniques

The IP cores were developed using the Vitis tools and the design methodology involved is divided in three steps, associated to a specific tool:

1. selection of functionalities to be incorporated inside the core. The software implementation is adapted into a HW specification, taking into account, for instance, register initialization and management. The specification is written in the C/C++ language. If the behaviour is as expected, the file can be exported as an RTL implementation, to be further incorporated as a Programmable Logic (PL). This step requires Vivado HLS;
2. integration of Processing System (PS) with PL – synthesised IP core – and configuration of its interconnections. Bitstream is generated and included in hardware platform export. This is accomplished using Vivado;
3. development of software projects, integrating the developed HW platform, in C language, using Vitis IDE. A main software application passes data and instructions to the HW, retrieving data from resulting computations. In this step the HW/SW co-project can be tested in a development board.

One of the benefits of this methodology is to take advantage of particular specifications provided by the tools that facilitate the design process. HLS interprets C/C++ programming language to produce hardware, and often the generic syntax is not sufficient to cover all hardware specificities. Therefore, it is necessary to pass extra information to the HLS compiler. This is done via the `#pragma` statements, which will be used in the implementation of the IP cores described in the next sections. For instance, to define how RTL ports are created during synthesis (`#pragma HLS INTERFACE`), and to manage the resources of the designed circuit (`#pragma HLS ALLOCATION`). The IP cores make use of two types of

AXI interfaces to connect to the Zynq's processing system. The AXI4-Lite interface operates in simple memory-mapped communications with low-throughput requirements [78]. The IP cores apply this interface with a configuration (`ap_ctrl_hs`) that provides control signals for triggering the execution of the IP cores and for checking their execution state. The AXI4-Stream interface allows high-speed transmissions of data and provides validation flags to control the data flow [78]. The IP cores using this interface are thus configured to deactivate the control signals (`ap_ctrl_none`). Finally, the implementation of the IP cores considers the application of the pipeline technique (`#pragma HLS PIPELINE`), to execute operations concurrently. This allows to reduce both circuit's critical path and latency, increasing the amount of hardware resources consumed.

4.2 PPG IP core

The PPG IP core aims the execution of the necessary processing tasks over raw PPG signals to obtain instant heart rate values. This section provides a comprehensive description of the process of designing the core dedicated to PPG signals. The decisions made to select the most suitable solution are also described.

4.2.1 Design and optimization

This section describes the first designed solution of PPG IP core, followed by its optimization process. As new versions are presented, more algorithm functionalities are included inside the core. Therefore, the very first version has low complexity, being progressively enhanced. The original implementation of the algorithm selected for computing instant heart rates is written in C language. It follows a conventional software architecture with sequential computations. Designing an IP core to handle the most complex processing tasks requires an insight of the software behaviour, described in Section 3.1.2. The approach to this design problem consists of analysing the algorithm's mechanics and data flows, identifying candidate functionalities to be integrated in a single function (core).

Version 0: software-only

A preliminary software version was tested to be used as reference. It was dimensioned to 100-sample buffers, meeting the specifications of the provided dataset example. The first step of the methodology was skipped, as no hardware was developed. The second step involved the PS only. During the third step, execution time was measured.

Version 1: dot product

Analysing the C implementation, the tasks with most computational cost are the dot products between two arrays. Also, these occur frequently throughout the code. As such, the first version of PPG IP core consists of a single product between two floats. The interfaces declared in this version implementation

are defined in Listing 4.1, showing that function arguments are mapped to AXI-Lite registers. Next IP versions also use this interface type by default.

Listing 4.1: Declaration and interfaces of PPG IP Version 1 core.

```
1 void PPG1 (float *a, float *b, float *c, int16_t *instr) {
2   #pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
3   #pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
4   #pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
5   #pragma HLS INTERFACE s_axilite port=c bundle=BUS_A
6   #pragma HLS INTERFACE s_axilite port=instr bundle=BUS_A
7
8   // program continues...
9 }
```

This first IP core is called at different stages of the algorithm, and these occurrences are discriminated in Table 4.1, applied to the example of Figure 3.1a. There are four distinct functions that make use of the core. Two of them, `linear_regression` and `mean_square`, are executed twice, because each one processes a single channel (RED or IR) at a time. The `Pcorrelation` function computes the correlation between both channels. The `autocorrelation` is generically calculated an undefined number of times, as it depends on periodicity search routine. As explained during the algorithm analysis, the preprocessing stage is applied to both RED and IR channels, regardless the system parameters, unlike periodicity search, that calls `autocorrelation` function until detecting the signal's peak.

Table 4.1: Number of PPG IP Version 1 function calls, applied to the 100-sample example of Figure 3.1a. Accesses are given by the sum of writes and reads.

Function	Writes				Reads	Accesses
	a	b	instr	do_op	c	
1. linear_regression	100	100	100	100	1	401
2. linear_regression	100	100	100	100	1	401
3. mean_square	100	100	100	100	1	401
4. mean_square	100	100	100	100	1	401
5. Pcorrelation	100	100	100	100	1	401
6. autocorrelation	914	914	914	914	11	3667
Total	1414	1414	1414	1414	16	5672

The number of core accesses is the sum of write and read occurrences. "Writes" include:

- `a` and `b`, the floats to be multiplied;
- `instr`, an integer that specifies register initialization or register incrementation;
- `do_op`, a flag that indicates the end of input passing, firing the core computation.

The meaning of the values taken by variables `a` and `b`, in the context of each function, are explicit in Table 4.2. "Reads" come down to `c`, which represents the returned value by the core, in this case the accumulator of products between `a` and `b`. There is also a validation flag, `c_vld`, for announcing that `c` is ready to be read, whose accesses are not counted.

Version 2: array dot product

Version 2 features the dot product between two arrays, instead of two single numbers. This allows to reduce the number of IP accesses, one of the main goals of this optimization process. As seen in

Table 4.2: Data flow of variables passed to the IP. "sample" stands for a raw bio-signal value, while "feature" denotes a preprocessed sample; t_{mean} means a shift value applied to the samples, as explained in Chapter 3.

Function	a	b
1. `linear_regression`	$\{-t_{\text{mean}}, ..., t_{\text{mean}}\}$	IR sample
2. `linear_regression`	$\{-t_{\text{mean}}, ..., t_{\text{mean}}\}$	RED sample
3. `mean_square`	RED feature	RED feature
4. `mean_square`	IR feature	IR feature
5. `Pcorrelation`	IR feature	RED feature
6. `autocorrelation`	IR feature	IR shifted feature

Listing 4.2, variable `instr` is no longer required. In fact, the management of register initialization and incrementation is done inside the core. Also, the data flow of IP input variables is similar to the presented in Table 4.2.

Listing 4.2: Declaration of PPG IP Version 2 core.

```
1 void PPG2 (float a[BUFFER_SIZE], float b[BUFFER_SIZE], float *c);
```

Following the procedure of Version 1, the IP accesses related to Version 2 are registered in Table 4.3. Total number of IP calls has decreased from 5672 to 3232.

Table 4.3: Number of PPG IP Version 2 function calls, applied to the 100-sample example of Figure 3.1a.

Function	Writes			Reads	Accesses
	`a[]`	`b[]`	`do_op`	`c`	
1. `linear_regression`	100	100	1	1	202
2. `linear_regression`	100	100	1	1	202
3. `mean_square`	100	100	1	1	202
4. `mean_square`	100	100	1	1	202
5. `Pcorrelation`	100	100	1	1	202
6. `autocorrelation`	1100	1100	11	11	2222
Total	1600	1600	16	16	**3232**

Version 3: a more efficient usage of PPG IP Version 2 core

The IP calls of the previous version can be optimized, taking into account that duplicated data is naively being passed. Analysing carefully the data flow of Table 4.3, some values do not need to be overwritten: at second call of `linear_regression` function, `Pcorrelation` and `autocorrelation` functions. In these cases, the values passed to `a[]` were already transferred in the preceding operations. As such, a more efficient usage of Version 2 is suggested in Table 4.4.

Version 4: new approach to instruction variable

From analysis of the results from the previous implementation it was found that the number of IP calls can be significantly reduced. A new version recovers the concept of `instr` variable, to distinguish operations to be executed. This way, the complexity of IP specification increases, however its access decreases. Considering the data flow obtained using Version 3, one can notice some room for improvement. In fact,

Table 4.4: Number of PPG IP Version 3 function calls.

Function	Writes			Reads	Accesses
	a[]	b[]	do_op	c	
1. linear_regression	100	100	1	1	202
2. linear_regression	0	100	1	1	102
3. mean_square	100	100	1	1	202
4. mean_square	100	100	1	1	202
5. Pcorrelation	0	100	1	1	102
6. autocorrelation	0	1100	11	11	1122
Total	300	1600	16	16	**1932**

the sequence at which data is written to a[] and b[] compromises the effectiveness of suppressing duplicated data. To overcome this, Version 4 provides a set of dot product combinations, allowing the main application to execute them interchangeably. As such, given two memory data, a[] and b[], it is possible to compute the following products: of an array itself ($a \cdot a$ or $b \cdot b$); both arrays ($a \cdot b$); or an array and its shifted copy ($a \cdot a'$). The product combinations of type $a \cdot a$ (or $b \cdot b$) and $a \cdot b$ allow to manage more efficiently the sequence of data passed to the IP. The product between an array and its shifted copy is useful to calculate autocorrelation. As a consequence of the flexibility of multiplication operands, IP data is managed by the sequence of Table 4.5.

Table 4.5: Data flow of variables passed to the IP Version 4, where a′ denotes the shifted copy of a; tag *hold* means that the content of a variable at a given instruction is not changed, reusing the previous assigned value.

Function	a[]	b[]	instr
1. linear_regression	$\{-t_{\text{mean}}, ..., t_{\text{mean}}\}$	IR samples	$a \cdot b$
2. linear_regression	*hold*	RED samples	$a \cdot b$
3. mean_square	*hold*	RED features	$b \cdot b$
4. mean_square	IR features	*hold*	$a \cdot a$
5. Pcorrelation	*hold*	*hold*	$a \cdot b$
6. autocorrelation	*hold*	*hold*	$a \cdot a'$

Table 4.6 summarizes the result of the optimization. As seen, the improvement is notable, thanks to the management of the data written to the IP's arrays. The autocorrelation function does not require any modification of the values stored in a[] and b[]. Their content is maintained since the second execution of the mean_square function, as stated in Table 4.5. The absence of writes before the execution of autocorrelation, the most called function of the algorithm, reduces the number of IP interactions. In other words, the reuse of data values by a[] and b[] reduced the number of accesses, despite the reintroduction of the instr word.

Versions 5, 6 and 7: integration of preprocessing functions

Analysing closely the sequence of IP calls alternating with software instructions, the possibility of encapsulating functions of Table 4.6 inside the core arises. Ultimately, these may be all executed by a single instruction. This step is divided into three sub-multiple problems, consisting of the cumulative integration of:

Table 4.6: Number of PPG IP Version 4 function calls.

Function	Writes				Reads	Accesses
	a[]	b[]	instr	do_op	c	
1. linear_regression	100	100	1	1	1	203
2. linear_regression	0	100	1	1	1	103
3. mean_square	0	100	1	1	1	103
4. mean_square	100	0	1	1	1	103
5. Pcorrelation	0	0	1	1	1	3
6. autocorrelation	0	0	11	11	11	33
Total	200	300	16	16	16	**548**

1. linear regression, mean square and Pearson correlation functions (Version 5);
2. DC subtraction routine (Version 6);
3. DC mean computation task (Version 7).

These three core versions share a common ground of providing two instructions, so that `instr` has only two meanings: execution of preprocessing tasks (which depend on the version) and autocorrelation calculation. As operations are included inside the preprocessing instruction, constant values are required to perform those calculations. Therefore, additional accesses are carried to pass values as parameters. These versions differ in the meaning and number of auxiliary transferred values. Version 7 achieves the ideal scenario of executing all preprocessing tasks using hardware in a single instruction. Autocorrelation function is executed separately, and repeatedly called by a control logic in software. In short, Version 7 aggregates **241** accesses:

- 200 of which fill buffers `a[]` and `b[]`;
- 3 are dedicated to auxiliary input parameters, for preprocessing;
- 12 specify the instructions (1 for preprocessing, 11 for autocorrelation);
- 12 trigger the operations;
- 14 retrieve output parameters (3 for preprocessing, 11 for autocorrelation).

This way, the process of integrating tasks with high computational cost inside a hardware core is completed.

Wordlength optimization: use of fixed-point representation

Once the functionalities embraced by the core are set, a process of optimizing the wordlength begins. The purpose of this optimization is to define a finite data resolution, such that the resulting error – the difference between exact and optimized values – is acceptable for a given context. Every variable dimension must be specified, as the ultimate goal is to design an optimized hardware solution. Allocating specific wordlengths to variables leads to a discrete range of their assigned values. An advantage of this process is to find the optimal balance between both system precision and required hardware resources. Most variables declared in the software implementation, of type `float`, are now represented by fixed-point. Initially, bit resolution is assigned such that every intermediate calculation matches the exact value. Then, resolution bits are gradually discarded and the resulting precision is evaluated. Attending to this

analysis, an error metric is set and one of the assessed versions is chosen accordingly to parametrize the final PPG module of the system.

Until Version 7, an example buffer containing 100 PPG samples has been used as reference. However, Section 3.3 introduced the high-level architecture of the system, stating that 900 samples were necessary to compute the heart rate of a 4.5-second sampling. The amount of samples can be rounded to the nearest power of 2, so that all system variables are dimensioned more appropriately. Therefore, the PPG core will handle buffers containing 1024 samples each.

The fixed-point notation allows to represent a real number with a specific amount of fractional bits and integer bits. In this notation is implicit a binary point dividing both parts, similar to the decimal point used in decimal numbers. In HLS, a variable is represented by fixed-point notation as `<W,I>`, where `W` identifies the total number of bits and `I` specifies the number of bits of the integer part. The number of fractional bits corresponds to the difference `W-I`. In brief, the methodology consists of designing, at first, the pessimistic version that leads to null wordlength conversion errors. This version is taken as reference from which the number of bits is reduced. This means that every variable is initially assigned a wide number of bits, determined by holding the precision of the arithmetic operations between variables. For instance, a product between two 10-bit variables should be stored as 20-bit. Resuming the notation of IP versions begun in previous sections, this preliminary core using fixed-point is called Version 8. Each IP's internal variable is identified and dimensioned in Table A.1, of Appendix A, and a shortened representation of this table is provided by Table 4.7. The column "V8" corresponds to the pessimistic version, where no fractional bits are discarded, and is followed by further versions presented next in this section.

Regarding the integer part of a variable, it is necessary to determine the minimum and maximum values each variable may carry. As such, datasets were tested and additional data was synthesised to evaluate each variable's range. Synthetic data was obtained modifying a subset of real PPG signal, as described next. Several operations were applied, such as multiplication/division by constants, and duplication/suppression of samples. The purpose is to obtain a wide collection of signals with extreme values of frequency and amplitude. This procedure is useful to assess the range of values the system must support. More concretely, signals with very low and high frequencies, covering pulses between 40 bpm and 180 bpm; signals presenting different AC component variations; signals whose amplitudes range every order of magnitude tolerated by PPG sensor. In this particular case, the sensor provides 16-bit unsigned integer values, so amplitudes range from 0 up to $2^{16} - 1 = 65535$.

After the analysis of minimum and maximum values, a new version V9 was developed. The resolution is reduced, but optimized, where each variable holds at least 12 fractional bits. Then, four additional versions were created – V10, V11, V12 and V13 –, where most variables were provided 8, 4, 2 and 0 fractional bits, respectively. Table 4.7 provides two examples of variables that are assigned a different number of bits at each version. The dimensioning obtained by applying this procedure to each internal variable, when possible and depending on the meaning of the values stored, is listed in Table A.1. The evaluation of the impact of progressively neglecting the arithmetic precision, by reducing the wordlength, is discussed in Section 4.2.2.

Table 4.7: Dimensioning of the PPG IP core internal variables, with different wordlengths. Variables are represented by fixed-point notation, <W,I>. W denotes the wordlength and I the number of integer bits. Six versions are considered, varying in the number of bits dedicated to variables' fractional part. An extended view of this table is provided by Table A.1.

Variable	V8	V9	V10	V11	V12	V13
prod	<69,36>	<38,26>	<34,26>	<30,26>	<28,26>	<26,26>
regC	<79,46>	<42,30>	<38,30>	<34,30>	<32,30>	<30,30>
number of fractional bits	33	12	8	4	2	0

Data stream: use of AXI-Stream protocol

Until this stage, data transfers to the IP have been made element by element, through AXI-Lite memory mapped protocol. A more convenient way of transferring is addressed in this section, by proposing a stream-based IP design. Streaming interface is an unidirectional channel from a master to a slave, so writes and reads are done in separate mediums [78]. The AXI4-Stream protocol is now considered, where data is sent sequentially, referred to the first sample. Stream is operated through a set of a pre-dimensioned data structure, in this case called ap_axis and discriminated in Listing 4.3, which also includes the definition of AXI interface.

Listing 4.3: Declaration and interfaces of PPG IP Version 14 core.

```
1  struct ap_axis {
2    ap_int<64> data;
3    ap_uint<1> last;
4  };
5
6  void PPG14 (hls::stream<ap_axis> &strm_in, hls::stream<ap_axis> &
       strm_out) {
7    #pragma HLS INTERFACE ap_ctrl_none port=return
8    #pragma HLS INTERFACE axis port=strm_in
9    #pragma HLS INTERFACE axis port=strm_out
10
11   // program continues...
12 }
```

According to the specification, a 64-bit channel is considered, which means that each burst of this protocol is interpreted as containing 8 Bytes of data. Variables of Table A.1 marked with bold represent data exchanged through IP core interfaces. Input data is referred to the top two rows, while output data returned by the core corresponds to the bottom bold rows. Taking into account that, in Version 13, most variables are less than 32 bits long, they may be paired and sent in a single 64-bit burst. Particularly, four 16-bit samples of raw input data can be encapsulated into one package.

Finally, the process of core design is concluded. The presented versions are evaluated in the next section, where the reasoning behind the selection of the preferable one is clarified.

4.2.2 Design evaluation

This section summarizes the evolution of IP accesses over the first seven proposed designs. Then, timing results obtained by running these versions in a FPGA are detailed. An error analysis of the fixed-point cores is discussed, from which the most advantageous solution is selected.

Evolution of IP accesses, versions V1-V7

The cumulative incorporation of functionalities inside the core, namely the ones discriminated in Table 4.6, reduces the total accesses to the IP core. The evaluation of IP core interactions is summarized in Table 4.8.

Table 4.8: Summary of the number of PPG IP accesses, from Version 1 (V1) to Version 7 (V7). Accesses are discriminated into preprocessing (functions 1-5 of Table 4.6) and periodicity (function 6) stages. Total accesses comprise total writes and total reads. Improvement ratio is calculated dividing total accesses of a given version by its preceding one.

Version	Total writes		Total reads		Total accesses	Improvement ratio
	Preprocessing	Periodicity	Preprocessing	Periodicity		
V1	2000	3656	5	11	5672	n. a.
V2	1005	2211	5	11	3232	1.75
V3	805	1111	5	11	1932	1.67
V4	510	22	5	11	548	3.53
V5	204	22	3	11	240	2.28
V6	206	22	3	14	245	0.98
V7	205	22	3	11	241	1.02

Analysing the "Improvement ratio" column, the two highest values – 3.53 and 2.28 – can be seen as prominent design enhancements. In fact, the first one marks the simplification of autocorrelation provision. The second one shows the gain of including preprocessing tasks inside the core, reducing drastically IP writes in 56%.

Timing results obtained by IP versions V1-V7

The developed cores are now compared to the software baseline in terms of execution times, running in the FPGA. Figure 4.1 shows the elapsed time of processing a 100-sample buffer, considered in previous sections, distinguishing the stages of preprocessing and periodicity search. The software version is run using only the processing system of the SoC.

The inclusion of functionalities inside the core clearly decreases the execution time. The ratio between Version 7 and SW is 2.34, suggesting a room for further improvement using wider buffers.

Error analysis of IP versions V8-V13

Fixed-point cores vary in the number of bits dedicated to represent the fractional component of its variables. Precision of arithmetic operations differs and thus an error analysis is convenient to evaluate the impact of precision loss throughout the calculation process. The variables declared inside each core implementation, listed in Table A.1, are considered. Also, from the subset of synthetic data produced as described in Section 4.2.1, five sets containing 1024 samples each were selected. This way, these buffers are processed by the fixed-point cores, versions V8-V13, and the values carried by each variable are registered.

At a first glance, the relative errors – given by the difference between observed and expected SW values, divided by the latter – are analysed. Figure 4.2 depicts the maximum relative errors observed

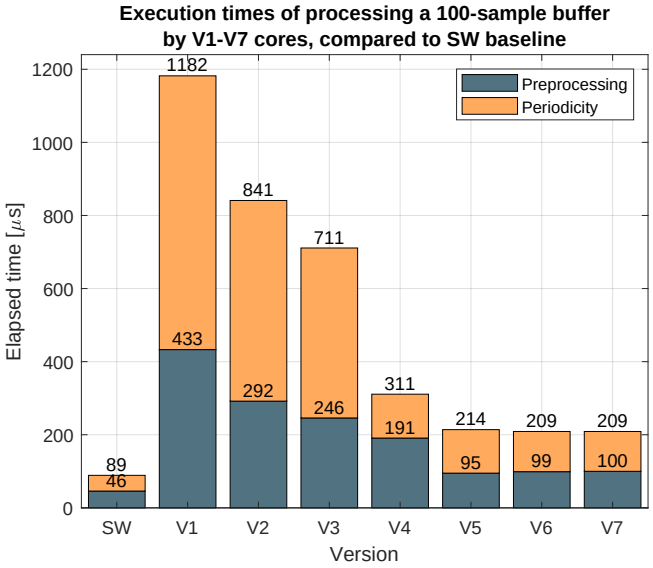


Figure 4.1: Execution times of versions V1-V7, compared to the SW baseline, after processing a 100-sample buffer.

at IP-returned variables, after processing five different sets by each core version, relatively to SW. It also traces the value range that each variable carries, in green lines, useful to interpret the impact of relative errors on the resultant values. This graphic confirms that, removing fractional bits, the relative

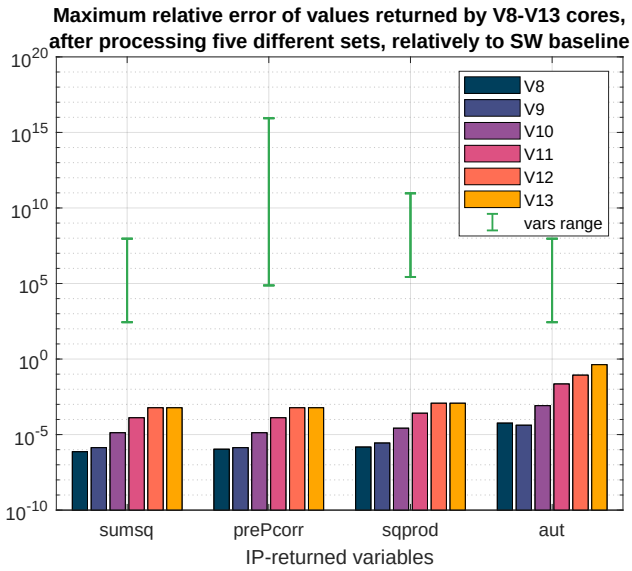


Figure 4.2: Maximum relative error observed after processing five 1024-sample buffers by the fixed-point cores. The sets comprise: S1 (real PPG data, from which the next sets were synthesised; observed HR is 85 bpm); S2 (obtained by dividing each S1's sample by a constant; signal with very low amplitude; HR is 85 bpm); S3 (multiplication of each sample by a constant; signal with very high amplitude; HR is 85 bpm); S4 (sample replication; HR is 40 bpm); S5 (sample removal; HR is 171 bpm).

error increases. The heart rates computed by fixed-point cores match the obtained by SW version, so the errors did not affect. Relative errors are higher at autocorrelation variable, `aut`, followed by `sq_prod`. The latter is an auxiliary variable returned by the core to the software side, by which its square root is calculated. Taking into account that `sq_prod`'s values range over tens of thousands, a loss of fractional precision is acceptable. On the other hand, errors related to `aut` are more significant. As this variable

is requested during periodicity search, errors can be briefly assessed by the sequence of iterations of this process. This means that if the algorithm computes autocorrelation values at the same delays as the SW version, by the exact order, the search routine is done correctly. An error during this process may lead to a period detection from the erroneous delay point. In fact, there is an error margin for each iteration, given by the difference between the autocorrelations of consecutive delays, that is, points of the autocorrelation function. Errors may be discarded if less than that margin, for the purpose of periodicity search. This criteria was validated during the processing tests of the selected sets. Also, a note on the other returned variables. The value of `sumsqIR`, ranging from tens to tens of thousands, is further used to divide values of autocorrelation. Thus, in the context of relative error depicted on the bar graph, fractional bits may be rejected. This is also valid to `prePcorrel`, whose values are used to compute ratios as well.

Overall, the impact of these errors does not perpetuate throughout the algorithm execution, considering the selected test sets. Heart rate values are correctly computed by the fixed-point cores. The least conservative core, Version 13, is a good candidate to be responsible for PPG signal processing and to integrate the final system. A more concise validation test is though required to confirm its performance and accuracy.

4.2.3 Design validation

To assess the accuracy of heart rate detection by the designed solutions, a large dataset of real PPG data was required. The 2015 IEEE SP Cup competition database [59] was chosen, containing wrist-type signals. This dataset includes records of eight subjects performing physical activities, namely walking and running. This is advantageous to evaluate the algorithm sensitivity to motion artifacts. Data was sampled at a frequency of 125 Hz, using a green LED, and split into 1000-sample sets, corresponding to 8-second frames. The ground truth values for heart rate of these sets have been determined sliding the window by 2 seconds, using ECG. Given that cores have been designed to handle 1024-sample buffers, the dataset was interpreted as slices with this dimension, resulting in 1324 sets. The validation test assumes that heart rates obtained for 1000 samples are identical to the computed from 1024 samples. The dataset was processed by the SW version and the optimized cores. The comparison between SW and core results is summarized in Table 4.9.

One can notice that versions V8-V11 register results similar to the ones obtained by the SW baseline. Version V12 presents a false positive in respect to SW baseline. In version V13, there are only seven contradictions: two false positives and five different values. The first case means that this core outperformed the SW, being able to detect a valid heart rate, rather than the SW version. These events occurred thanks to the rounding of the value of a quality criteria, the autocorrelation ratio, where SW computed 0.49 and the core 0.5. Therefore, this buffer was discarded by the SW version, but validated by the core. The second case, mismatch values of an average 1.6 bpm, relates to precision loss during autocorrelation peak detection.

Next, a comparison to the ground truth values is addressed. The SW version did not compute valid

Table 4.9: Comparison between the results obtained by SW-only and optimized cores. ”Positives” mean that cores’ results comply with SW’s, while ”negatives” mean the opposite; accuracy is the positive rate. ”FP” stands for ”false positive”, corresponding to cases where SW does not return a valid heart rate, but the core does. ”FN”, a ”false negative”, is the vice-versa. ”Diff. val. ” is the short form of ”difference values”; represents the occurrences of when both cores return a valid heart heart differing to one another. In those cases, the average absolute error is appended.

Version	Positives	Negatives	Accuracy	FP	FN	Diff. val.	Avg. abs. error [bpm]
V8	1324	0	1	0	0	0	0
V9	1324	0	1	0	0	0	0
V10	1324	0	1	0	0	0	0
V11	1324	0	1	0	0	0	0
V12	1323	1	0.999	1	0	0	0
V13	1317	7	0.995	2	0	5	1.6

heart rate values corresponding to all 1324 instances, due to algorithm characteristics. In fact, only 35% of them can be compared to the exact values provided by the dataset. The bar graph depicted on Figure 4.3 represents the absolute error of the computed heart rates, referred to the dataset ground truth. Six distinct intervals are defined to distribute the occurrences of different error proportions. This

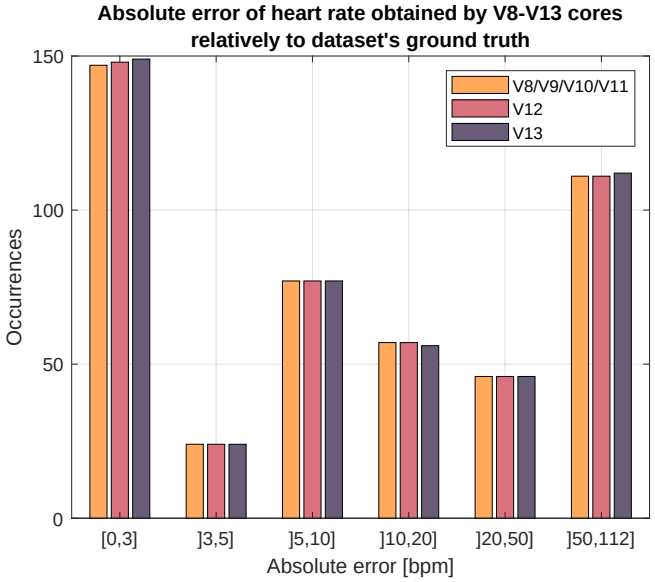


Figure 4.3: Absolute errors obtained by fixed-point cores when processing real PPG database. Versions V8-V11 presented identical results, therefore they appear merged in a single category.

figure shows that the compared versions present similar absolute errors of the computed heart rates. Therefore, precision loss over the fixed-point versions does not interfere much with the final result. More specifically, and according to the information provided by Table 4.9, version V13 obtained seven results differing from the SW baseline, out of 1324 comparisons. This means that the discarding of the fractional bits by version V13 led to an accuracy loss of 0.5%, when compared to the conservative version V8. A simpler core design, rejecting fractional bits, like V13, is seen as the solution that minimizes the hardware resource usage.

Resources utilization

The process of designing the core takes into account the resources available in the targeted platform. If the required resources exceed the maximum available on the device, the designer must perform adjustments to reduce hardware occupation, or select a more resourceful device. The Zynq-7010 SoC resources to be occupied by the PPG IP core are listed in Table 4.10. DSP is the most consumed resource, which is related to the fact that PPG signal preprocessing stage is entirely executed inside the core.

Table 4.10: Estimated resources to be used by the PPG IP core, referred to the Zynq-7010 SoC.

Resources	Available	Utilization
LUT	17600	1943 (11%)
FF	35200	1527 (4%)
BRAM	120	4 (3%)
DSP	80	16 (20%)

4.3 EEG IP core

The objective of creating the EEG IP core is to perform classification of EEG signals in hardware, without intervention from the CPU. The KNN classifier comprises three main tasks. The first one is the Canberra distances computation, the second one is sorting the computed distances and the third one is the translation of the shortest distances into a predicted emotion. Taking into account the profiling analysis of the KNN classifier, the candidate tasks to be integrated into a hardware specification are the calculation of distances between test and training instances and the retrieval of the K shortest values. The assessment of the emotion class does not execute significant processing tasks, and thus it may be assured by software-only instructions. This section addresses the implementation of the module that receives instances of feature sets to output the K nearest ones.

4.3.1 Design concept

Design methodology is similar to the presented in Section 4.2.1 for the PPG IP core. Therefore, the optimization process is explained more briefly, rather than described step by step as previously.

The approach to tackle this problem is to develop two independent IP cores implementing the previously identified tasks. A first one for distances computation and a second one for sorting the distances. Another possible approach would be the development of a single IP core where the distances between points are sorted as they are computed. However, this would create complex nested loops, increasing the latency of the core, and decreasing the throughput. Moreover, it would be impractical to use pipelining as operations have to compute values iteratively, or in other words they can not operate continuously. In the architectural point of view, iterations should be executed in parallel, taking advantage of the hardware resources available, and requiring less clock cycles to completion. By designing a dedicated core for each task assuring that pipeline is feasible, cores accept new data on their inputs per clock cycle.

In other words, it is guaranteed that each core has unitary initiation interval. Thus, once the pipeline is filled, cores' outputs are produced each clock cycle. The process of designing the IP cores lies on the optimization of their throughput. This achievement is crucial given the volume of data each core handles to obtain a classification or numerical result.

Approach to implement the IP cores

A first strategy to implement the distances calculation task is to provide the training set to the core, and memorize it, so that every test instance is then passed and a distance computed and returned, sequentially. However, the storage of the training set is not feasible, due to limitations on the amount of on-chip memory available. The Zynq-7010 SoC contains 120 BRAMs, embedded dual-port memories, holding 18K bits each. This means that a single SoC supports up to around 70K 32-bit values. This amount may not be enough to store a training set inside, since the model must memorize sufficient instances to successfully generalize upon unknown ones. For instance, assuming each instance comprises 160 features implies that the on-chip memory stores less than 440 entries. An alternative approach is to first provide the test set to the IP core, storing it in a BRAM. Then, pass each instance of training set, sequentially, which is previously initialized in an external memory. A distance is thus produced at the rate training instances are transferred. This solution is a more generic way to handle large training sets, since the usage of an external memory provides scalability over on-chip memory. In short, this is the operation of the first EEG core, hereinafter referred to as `EEG_CALCDIST`. Next paragraph addresses the second piece, named `EEG_SORTDIST` core.

Connecting `EEG_CALCDIST`'s output to the input channel of `EEG_SORTDIST` core enables the execution of an insertion sort algorithm as distance values are produced. This algorithm is advantageous to sort an array as its elements are received, despite performing less efficiently when compared to more advanced algorithms, such as quicksort. Informally, insertion sort is equivalent to sorting playing cards in one's hands. The idea is to constantly keep an array in sorted order. Before pushing a new element, the array position (index) where it will be placed is assessed. Then, the already sorted elements at the right of the targeted index are moved, leaving a space for inserting the new element. As such, the sort distances core initializes two local memories: `near`, to store the *K* shortest distances, and `indexes`, to save the order at which those distances arrive at the core. The value of *K* is defined by the analysis of algorithm profiling in Section 3.2.3, where was concluded that $K = 21$ led to more accurate classification results. The number of indexes returned is customizable but limited to *K*, as explained next. After initialization, the core is ready to accept data.

4.3.2 Implementation of distances calculator core

A preliminary version of `EEG_CALCDIST` IP core is dimensioned to handle 64-bit values, meaning that every feature, ranging between 0 and 1, follows the Q1.63 fixed-point format – having 1 integer bit and 63 fractional bits. This dimension was chosen given the fact that it is expected to use a 64-bit width channel for data transfers through AXI Streaming. Then, additional versions supporting input values

with less precision were developed. In summary, four versions were considered, handling:

- 64-bit words, using Q1.63 fixed-point format, where a single feature is sent per transfer;
- 32-bit words, using Q1.31 fixed-point format, where 2 features are wrapped per transfer;
- 16-bit words, using Q1.15, and carrying 4 features per transfer;
- 8-bit words, using Q1.7, and carrying 8 features per transfer;
- 4-bit words, using Q1.3, and carrying 16 features per transfer.

Reducing words' precision allows to transfer more data in a single data transfer, thus decreasing the number of memory accesses and speeding up the execution of the algorithm. However, a less precise system is more vulnerable to computational errors, misleading classification results. Accuracy and performance of the aforementioned versions are summarized and compared in Section 4.3.4. The analysis shows that transferring values more compactly, and thus requiring less clock cycles, pays off precision losses. Hereafter, the second-least conservative version, that uses 8-bit words, is selected to be optimized. The reasons behind this decision are explained in Section 4.3.4.

Interfaces

`EEG_CALCDIST` core is declared as shown in Listing 4.4. Its interfaces comprise an input AXI4-Stream channel of 64 bits (*ap_axis64*) and an output AXI4-Stream channel of 16 bits (*ap_axis*). The input channel passes up to 8 features, with Q1.7 fixed-point format each, per transfer, which makes a total of 64 bits. Given that features vary between 0 and 1, and Canberra distances can be seen as sums of 160 features per instance, those distances are less or equal to 160. As such, 8 bits are needed to represent the integer part of distance values, the output channel. Fractional bits are also dimensioned to 8 bits. Although features contain only 7 fractional bits, 8 are used to round up the data width of output stream to 16 bits. Thus, output channel produces a 16-bit distance per transfer, in Q8.8 fixed-point format.

Listing 4.4: Declaration and interfaces of EEG calculate distances core.

```
1 void EEG_CALCDIST(hls::stream<ap_axis64> &strm_in, hls::stream<
      ap_axis> &strm_out){
2   #pragma HLS INTERFACE ap_ctrl_none port=return
3   #pragma HLS INTERFACE axis port=strm_in
4   #pragma HLS INTERFACE axis port=strm_out
5
6   #pragma HLS ALLOCATION instances=udiv limit=8 operation
7
8   // program continues...
9 }
```

Core operation

The operation of `EEG_CALCDIST` core is depicted in the block diagram of Figure 4.4. This high-level circuit includes the essential elements that compose the datapath, such as memory registers and operators. The module that computes Canberra distances is simplified by a green box named *Canberra*. The diagram allows to visualize the data flowing from incoming stream channel down to the output port. Figure 4.5 shows the expected data to arrive at input stream channel, but also the produced data by the core, over time. First, a test set (orange) is passed, followed by training set (yellow). Distances (green)

are outputted as instances of the training set are received and processed.

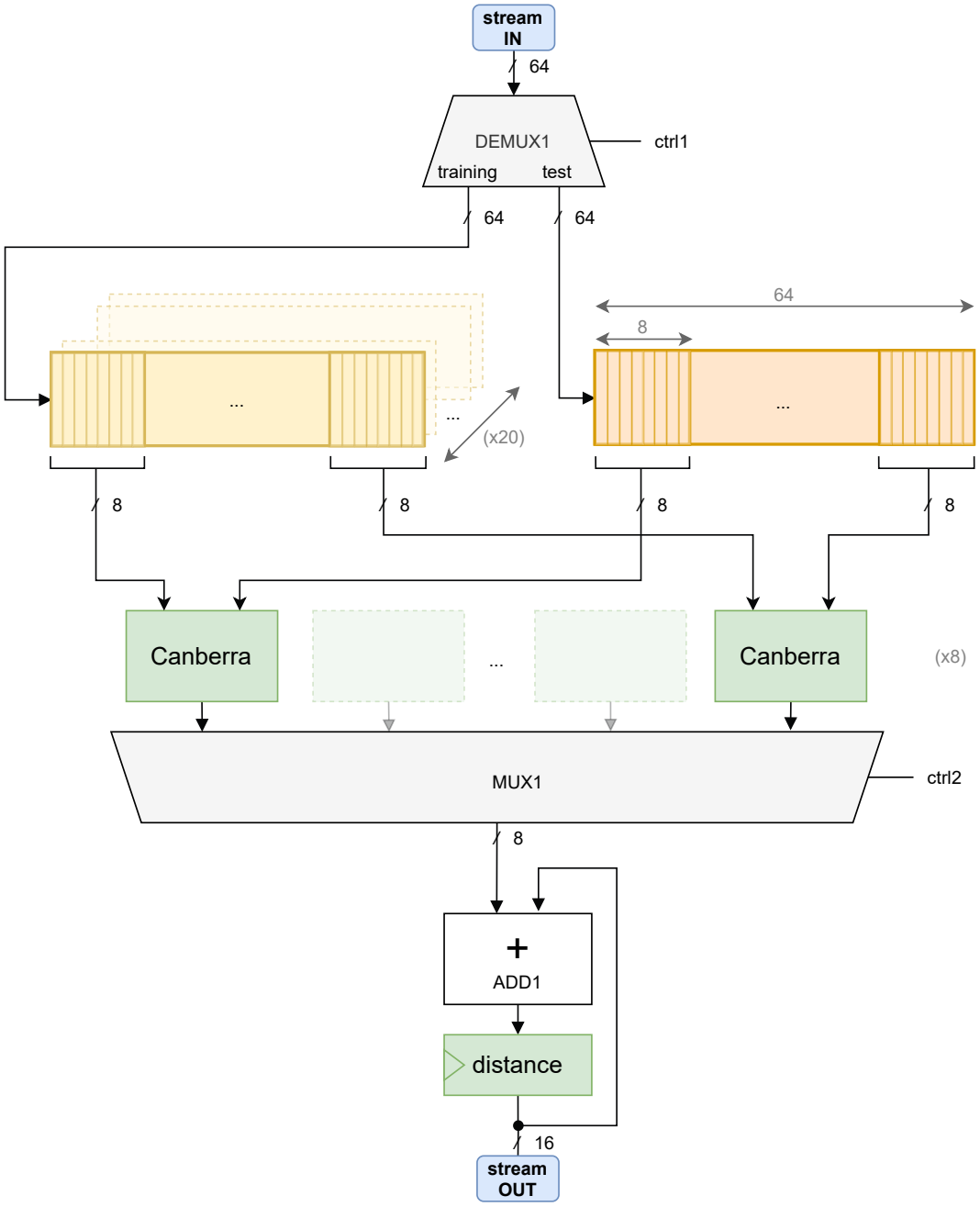


Figure 4.4: Block diagram of calculate distances core.

Canberra boxes implement the logic written in Listing 4.5. It consists of the computation of a partial distance between two features. In other words, given two arrays, x and y, a partial result is the distance between x_i and y_i, regarding a specific arrays' dimension i. To obtain a Canberra distance, this box must iterate over two complete test and training arrays. Then, the final result is the sum of all terms.

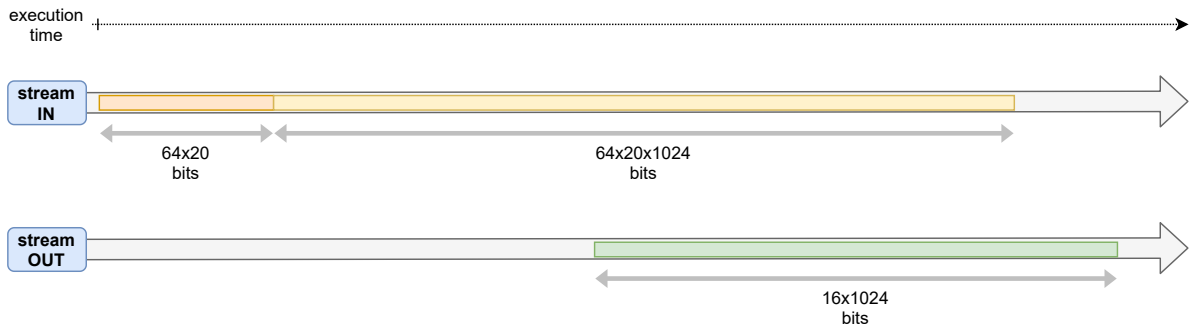


Figure 4.5: Data flow of calculate distances EEG core, through AXI4-Stream ports, over time.

Listing 4.5: Pseudo-code of *Canberra* block.

```
1   input: featureTest, featureTraining
2   output: partialDistance
3   begin
4     num ← | featureTest − featureTraining |
5     den ← featureTest + featureTraining
6     if den = 0
7       partialDistance ← 0
8     else
9       partialDistance ← num / den
10    end
11    return partialDistance
12  end
```

The high-level description of the core of Figure 4.4 has been implemented in C/C++ language, to be synthesized into hardware. It includes the following steps:

1. a loop for reading test set features via an AXI4-Stream channel of 64 bits; loop has $\frac{160}{8} = 20$ iterations, assuming 160 features per instance and 8 features of 8 bits per transfer; data is stored in a 64-bit BRAM;
2. a major loop for reading training set features through the same AXI channel, with $1024 \times 20 = 20480$ iterations, assuming a total of 1024 training instances, read in packages of 8 values each time;
   (a) computation of the remainder of major loop index divided by 20, useful to perform control instructions later on;
   (b) an inner loop that iterates over the received 8 training features to compute the Canberra distance between those training and corresponding 8 test features; this is a partial result that needs to be appended to a register;
   (c) a control instruction that initializes an accumulator register to 0, if the auxiliary remainder matches 0, indicating the consumption of a new training instance;
   (d) an inner loop that iterates 8 times to increment the register with the computed partial distances; after 20 major loop iterations, the register holds a complete Canberra distance;
   (e) a control instruction that assesses the auxiliary remainder; matching 19 implies that previous inner loops completed 20 series, and thus a Canberra distance between test-training instances is produced; if so, the distance value (16 bits) is written to an AXI4-Stream channel, to be later consumed by `EEG_SORTDIST` core; otherwise, the major loop is still computing a

partial Canberra distance.

Notes on the design techniques

The design of the EEG IP core applies particular techniques besides the ones presented in the beginning of the chapter. The declaration in the sixth line of Listing 4.4 (`#pragma HLS ALLOCATION`) sets a limit on resource allocation. This pragma allows to balance the resources and the performance, by restricting the amount of instances of an operator in the RTL. In this case, the limited operator (`udiv`) is used in the division computed inside the *Canberra* block, represented by a green box in Figure 4.4, as seen in line 9 of Listing 4.5. The *Canberra* block is instantiated 8 times, thus 8 `udiv` operators are sufficient.

The instructions that calculate Canberra distances are implemented by a spare high-level function, `Canberra()`, which is called repeatedly, in the software point-of-view. In the hardware perspective, multiple *Canberra* blocks are instantiated, so that distances can be computed in parallel. This way, resource allocation is extended to `Canberra()` thanks to an additional pragma defined inside: `#pragma HLS INLINE`. This tag allows to remove function hierarchy, so that hardware resources are better shared between `Canberra()` and its main function, `EEG_CALCDIST()`. This means that resource allocation specified inside the main function is inherited by its sub-function, in this case `Canberra()`. The allocation of 8 dividers has arisen from the need for limiting this amount to the same number of parallel instances that compute Canberra values, which is 8. By default, HLS tool increases latency if pipeline is applied without any resource allocation, when unrolling internal loops, compromising the task scheduling. Loop unrolling means to optimize the sequence of instructions inside a loop with a predetermined amount of iterations.

Finally, the `#pragma HLS PIPELINE` statement was applied inside the major loop that receives collections of training features. This corresponds to step 2 of the high-level description explained previously. This pragma allows the execution of instructions in pipeline. This way, each time a collection of 8 samples is read, a pre-sum of Canberra distance is computed in parallel, without waiting for reading a complete training example.

Latency estimation

It is crucial to guarantee the pipelined execution of the core's tasks. Before the implementation of the proposed core, it is necessary to define the performance requirements. As so, the estimation of the clock cycles required to compute all output values can be drawn. To determine the latency of `EEG_CALCDIST`, the system producing Canberra distances at full capacity is considered. In other words, a single test instance is paired with a set of 1024 training rows, so 1024 Canberra distances are produced. Assuming that 8 features are transferred per cycle, and having in mind that each instance contains 160 values, it takes:

- $\frac{160}{8} = 20$ cycles to read the test set;
- $1024 \times \frac{160}{8} = 20480$ cycles to read the training set.

Therefore, 20500 cycles are needed to read the incoming data. Supposing that the design allows

pipeline, such that the core accepts new data every clock cycle (initiation interval is unitary), the design latency is expected to be 20500 cycles, plus the cycles to fill the pipeline.

Design optimizations

The core description was obtained after several design iterations. Such iterations included design optimizations to reduce latency and loops' complexity, and they are described in next paragraphs.

Initially, the local memory for storing the features of test set (*featuresTest*) was dimensioned as containing 160 positions of Q1.7 values (8 bits). Taking into account that data is transferred via AXI4-Stream through 64-bit packets, features were extracted by slicing each burst in 8-bit segments, and added to the local memory. The optimized alternative considers a 64-bit memory with 20 values, matching the previous data width. This way, memory filling is done with least clock cycles possible. This is possible due to values being directly stored as 64 bits, avoiding any data decomposition, but later treated as 8-bit.

Listing 4.6: Declaration of local memory for storing test set features.

```
1 #define NFEATURES 160
2
3 static ap_ufixed< 8, 1> featuresTest[NFEATURES];   // first version
4 static ap_ufixed<64, 1> featuresTest[NFEATURES/8]; // optimized
```

Another modification was the rearrangement of data flow inside the main loop, that reads training features. At first, the design implied the extraction of a complete training instance, composed by 160 features, before determining Canberra distance. Instead, the optimized design allows the computation of preliminary results, more specifically regarding a collection of 8 features. This brings the possibility of computing such partial values in a parallel way, rather than sequentially by waiting for a complete training occurrence.

In summary, `EEG_CALCDIST` is designed to handle instances with a specific dimension; in this case, 160 features per trial, corresponding to 32 electrodes times 5 wave bands. This core loads, counter-intuitively, a single test trial followed by 1024 training instances. This amount was selected due to the publicly available EEG repository on which this work relies, the DEAP dataset. All trials are expected to be scaled between 0 and 1, and interpreted as Q1.7 fixed-point. In a real-time application, test features are provided once EEG signals are processed, while the training set is pre-stored in an external memory. Partial results of Canberra distances are computed as collections of 8 training features arrive at the core. Final Canberra distances are transferred to `EEG_SORTDIST` via AXI4-Stream once a complete training instance is retrieved.

4.3.3 Implementation of sort distances core

`EEG_SORTDIST` is designed to sort the outcome of the previous core, which transfers 1024 Canberra distances using Q8.8 format. It is necessary to keep a record of the order at which those distances were received, such that the shortest ones are identifiable through their indexes, rather than distance values themselves. This procedure is essential to later query the training set to perform an emotion classification successfully. Training instances are uniquely accessed through their memory address.

For this reason, possessing the values of shortest Canberra distances is not useful to access training entries. Indexes act as keys to the memory map.

Interfaces

The interfaces of the sort distances core are represented in Listing 4.7. The purpose of each one is discussed next.

The first argument of `EEG_SORTDIST`, `strm_in`, is an AXI4-Stream channel through which computed Canberra distances are passed to be sorted. Its `ap_axis` structure implies a data width of 16 bits, interpreted as Q8.8 floating point format. It is expected to have the core responsible for computing Canberra distances connected to `strm_in` port. As such, the sorting algorithm is continuously executed as distances are received.

The second argument, `instr`, serves an AXI4-Lite interconnection to customize the amount of distances to be sorted. It is an optional argument and, in its absence, K shortest distances are targeted. In this architecture, $K = 21$.

The last one, `dist_i`, is also an AXI4-Lite interface, representing a memory where the core writes its output. The core returns the indexes – that is, the order at which distances arrive through the streaming interface – of the K shortest distances. The low volume of outputted data requires a simple memory-mapped communication interface, rather than a high-speed one like AXI4-Stream.

Listing 4.7: Declaration and interfaces of EEG sort distances core.

```
1 void EEG_SORTDIST(hls::stream<ap_axis> &strm_in, ap_uint<32> instr,
      ap_uint<32> dist_i[K]){
2   #pragma HLS INTERFACE axis port=strm_in
3   #pragma HLS INTERFACE s_axilite port=return bundle=AXILite
4   #pragma HLS INTERFACE s_axilite port=instr bundle=AXILite
5   #pragma HLS INTERFACE s_axilite port=dist_i bundle=AXILite
6
7   // program continues...
8 }
```

Additionally, `EEG_SORTDIST` contains two internal BRAM memories – `distances`, defined as Q8.8, and `indexes`, defined as 10-bit unsigned integers – both with size $K = 21$. These store, respectively, the shortest Canberra distances and the corresponding order of their arrival at the core. The management of both structures is explained next.

Version 1 approach: sorted array

A possible strategy to implement the previous interface is through the logic presented in Section 4.3.1. First, a memory with $K + 1$ elements is initialized with the maximum value possible, such that every incoming distance is less than the initialization value. Additionally, a memory with $K + 1$ elements is initialized with zeros to store the distance indexes. Then, for each distance received, the memory position where the distance will be stored is assessed. This step compares each memory element to the input value, starting from the first memory element, until a greater value is found. Once this condition is met, the insertion position has been found. Therefore, the elements located in the following positions are moved one position to the right, to leave a space for insertion. Starting from the last memory element

to the insertion address, values are copied to the following address. This procedure is also applied to the memory storing the indexes corresponding to the sorted distances. The input value is then written at the insertion address, completing an insertion cycle. The same occurs at indexes memory, where the index corresponding to the new distance is stored at the equivalent address. The described process is repeated for all input values to be sorted, guaranteeing that `distances` is a sorted array at each iteration. Once insertion sort algorithm is finished, the values stored in indexes memory is outputted to `dist_i`.

The pseudo-code for implementing this solution is displayed in Listing 4.8. It considers two pre-initialized memories as input, `distances` and `indexes`, to be sorted. The interface through which distances are passed is represented by `strm_in`, similarly to the declaration of Listing 4.7. This solution presents a typical software approach of implementing a sorting algorithm. The instruction of line 8 assesses the memory position where the new element will be inserted. When the conditional statement of this instruction is true, the loop declared at line 9 is executed. This routine corresponds to moving memory elements one position to the right, up to a certain `i`-th element of `distances` and `indexes`. After moving the elements and inserting the new distance, a *break* instruction discontinues the loop for assessing an insertion position. This prevents the algorithm from inserting duplicates. Therefore, the number of iterations run by the loop of line 7 is not deterministically countable. It depends on the order at which elements are passed to be sorted.

Listing 4.8: Pseudo-code of insertion sort algorithm, based on a sorted array.

```
1   input:  strm_in
2           distances[K], indexes[K] // previously initialized
3   output: distances[K], indexes[K] // sorted
4   begin
5     index ← 0
6     for each receivedDistance in strm_in
7       for i ← 0 to K
8         if receivedDistance < distances[i]
9           for j ← K to i
10            distances[j] ← distances[j−1]
11            indexes[j] ← indexes[j−1]
12          end for
13          distances[i] ← receivedDistance
14          indexes[i] ← index
15          break // insertion is completed; a new distance can be read
16        end if
17      end for
18      index ← index + 1
19    end for
20  end
```

In the context of the application, one of the goals is to assure that core's throughput is one, that is, a new output is produced at each clock cycle. At least, to keep sorting latency low to minimize the delay between a computed distance and its subsequent memory insertion. It is also important to balance this requirement with the amount of hardware resources used to implement the solution. Next section suggests an alternative way to solve the sorting problem.

Version 2 approach: cell chain

A possible method to sort distances is to pass the input values through all memory elements, comparing the distances to the stored values. The idea is to, at each memory address (cell), update or hold the stored value, depending on its comparison to the received value. If the received distance is less than the distance stored at a given cell, the cell is updated. Before being overwritten, the stored value is passed to the next cell. Otherwise, the stored value is held and the input value is propagated to the next cell, where the logic repeats. This iterative procedure can be seen as a chain, or an array, transferring values between adjacent cells, or elements. This logic guarantees that, for each received distance, a precise number of instructions is executed to complete an iteration of the insertion sort.

The high-level description underlying the hardware implementation of this `EEG_SORTDIST` core version is specified with the following logic flow:

1. loop with K iterations for BRAM memories initialization, to avoid data arrangement conflicts; these memories include two components: distance values (*distances*) and respective indexes representing their order of arrival at the core (*indexes*); distance values are initialized to 255, the maximum integer value supported by Q8.8 format; this procedure allows to perform insertion sort, since the incoming distances are definitely less than 255; in turn, the indexes array is first arbitrarily filled with zeros, as no value comparisons are made;
2. outer loop for reading incoming distances and updating index value; the number of total iterations matches the number of distances, 1024; this is the main loop that encompasses additional steps:
   (a) execution of a function block that:
      i. receives the values of the first elements of *distances* and *indexes*, stored locally, the input distance and its index;
      ii. compares the stored distance to the received one; if the input distance is smaller:
         A. the stored distance and index values are returned;
         B. the stored distance is overwritten by the input distance;
         C. the stored index is overwritten by the respective input index;
      iii. otherwise:
         A. the stored distance and index values remain unchanged;
         B. the input distance and index are returned, to be used in the next iteration;
   (b) an inner loop for propagating the execution of the previous function to the remain elements of *distances* and *indexes* memories; this procedure works as a chain of multiple function instances, passing information from the second element to the last; this loop iterates $K - 1$ times, because the first element is assessed during the insertion of an incoming distance; at each loop execution, a pair (distance, index) is evaluated, taking into consideration the values returned by the previous iteration; similarly, new values are returned to the next cell;
3. once sort is completed, a final loop writes the indexes of the shortest distances through AXI4-Lite.

Block diagram of Version 2

The design diagram of sort distances core is depicted in Figure 4.6. This provides a graphical view of the datapath that implements the insertion sort of *distances* and *indexes*. A stream input containing Canberra distances feeds the orange chain, composed by 21 cells. Inside these cells is represented a local memory address that stores one of the nearest distances, labelled as `distance_i`. The logic of insertion is materialized with two multiplexers (`MUX1`, `MUX2`) and a comparison logic unit (`COMP1`). In short, `COMP1` evaluates if the input stream value is less than `distance_i`, outputting a control signal, `ctrl`. Depending on the outcome of the comparison, the new value stored inside `distance_i` and the value returned by `MUX2` vary. If `ctrl` is *true*, `MUX2` redirects the value stored in `distance_i` to `dist_OUT`, and `MUX1` selects the input distance `dist_IN` to overwrite `distance_i`. Signal `dist_OUT` is passed to the next orange cell, which repeats the described insertion logic.

In parallel, the control logic represented by `ctrl` is also taken to manage `indexes` memory, represented by a purple chain, on the bottom half of Figure 4.6. Whenever a `distance_i` is updated, `index_OUT` carries the value stored in `index_i`, and `index_i` is pushed the value passed by `index_IN`. Otherwise, `index_i` holds the same value and `index_OUT` pushes `index_IN`.

Once the insertion sort algorithm is completed, the values stored inside each `index_i` register are written to `dist_i`, introduced in Listing 4.7. The produced data is transferred via an AXI4-Lite interconnection.

This diagram does not represent the first step related to memory initialization, neither the computation of indexes. Simply put, `index_IN` is 0 when the first Canberra distance is read, being incremented by one each time a new distance is read, up to 1023.

Design selection and optimization

The first implementation of the sort problem followed a software-based approach. It was set that, as distances arrived to the sort function, the algorithm searched for the specific array position where insertion could be done. Then, after insertion, array's elements were shifted one position to the right. The shifting instructions were only executed during an element insertion, and skipped if the element was discarded. Given that the latency was dependent on the order at which elements arrived at the core, it was not possible to estimate a constant latency for the circuit. The execution of the algorithm could take longer at certain iterations. In the best-case scenario, the latency of this implementation was 4096 clock cycles. In the worst-case, latency was 46080 clock cycles.

An alternative solution presents a different way to handle the `distances` memory. From this perspective, the array is effectively sorted by the end of the algorithm, rather than after each element insertion. The core receives distances and propagates them as a data chain, to be picked by logical cells, updating values stored in memory. This approach solves latency issues raised by the first solution. On one hand, latency can be measured, since all memory elements are subject to determined instructions. As seen on the analysis of `EEG_CALCDIST` core, using pipeline, latency is approximately given by the sum of time needed to fill the pipeline and the number of elements to be sorted, 1024. In fact, the core has a latency

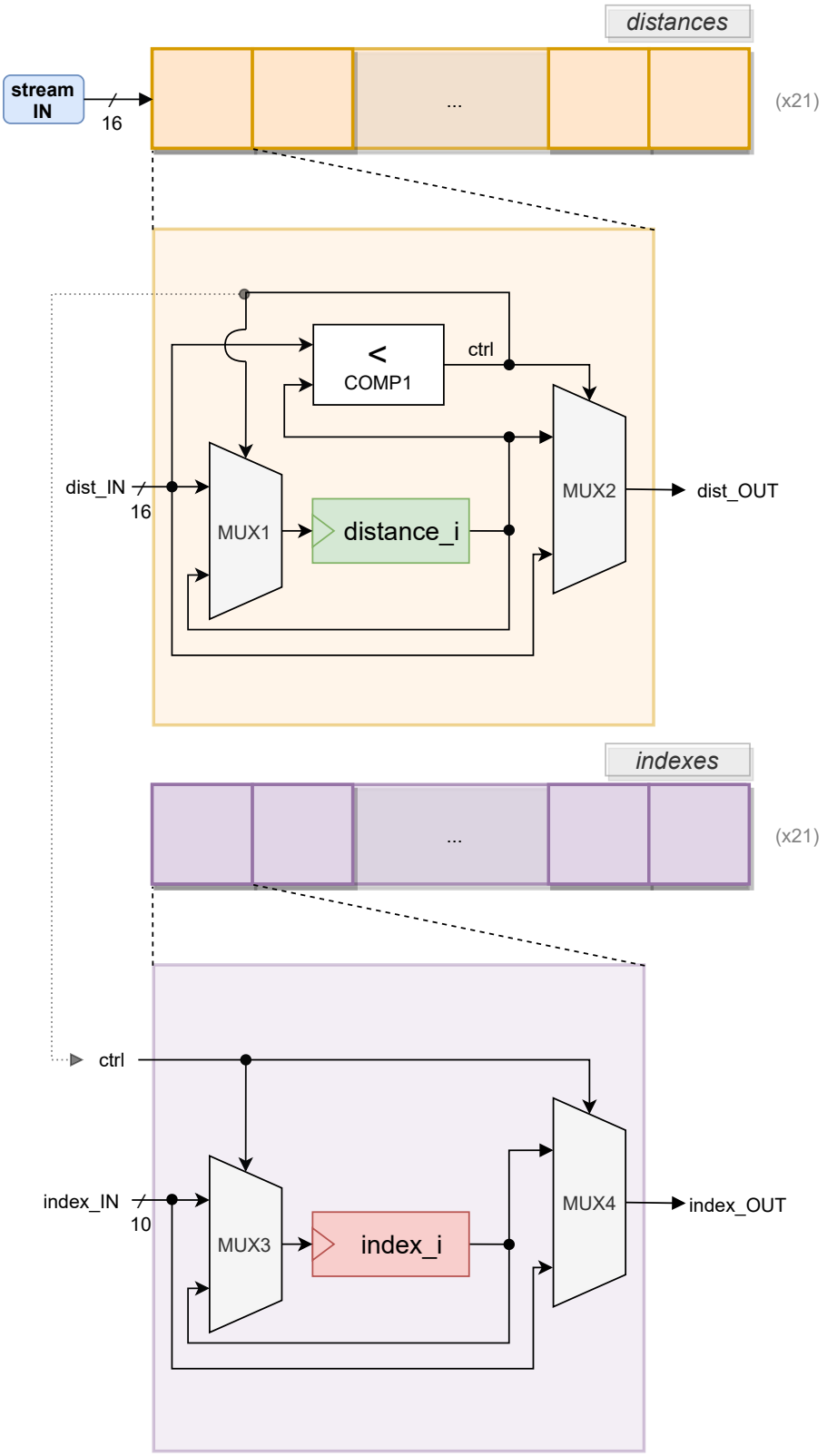


Figure 4.6: Block diagram of sort distances core, inspired from [79].

of 1055 clock cycles, a reduction of the latency of Version 1 in its best-case scenario by a factor of 4. Comparing to the worst-case scenario of Version 1, the reduction factor is 45. Both factors correspond approximately to the iteration latencies of Version 1's best and worst cases, respectively. This occurs

due to the fact that, for each element insertion, a different sequence of instructions is executed. Regarding Version 2, the input data flows from the very first array element to the last one. During a data trip, the value that is being transmitted along the chain may vary, being switched by a greater value if an insertion is performed.

On the other hand, this solution presents a lower execution time due to the feasibility of applying pipeline. As so, the `pragma HLS pipeline` directive is applied inside two loops of the high-level description: at local memory initialization (step 1) and read distances routine (step 2).

An additional pragma is used to reduce latency, known as `pragma HLS array_partition`. This setting converts a memory array into a set of smaller arrays. Partitioning this way increases the amount of read/write ports, improving final design's throughput [80]. Taking into account that latency of Version 1 is significantly greater than the obtained by Version 2, the latter is the selected to perform the distances sorting task.

In summary, `EEG_SORTDIST` is a dedicated core for receiving 1024 Canberra distances, 16-bit values interpreted as Q8.8 in fixed-point. Its main goal is to sort those distances as they arrive at the core, and finally return the indexes corresponding to the shortest ones. Before data retrieval, local BRAMs are initialized, to be filled during insertion sort execution. The amount of produced indexes is customizable by an input argument that should be less or equal to $K = 21$. Indexes are transferred through a AXI4-Lite interconnection. Distances are obtained by using indexes as keys of the table containing training set's entries. The system that makes use of the developed cores may then query the training set to obtain relevant information for classification. This interaction is explained later on Chapter 5. Before that, next section provides a preliminary validation test of the developed designs.

4.3.4 Design validation

The pair of EEG cores is tested with DEAP dataset, to validate the proper operation of the proposed functionality. The dataset is composed by records of 32 individuals, each one stimulated by 40 different trials. Therefore, there are 1280 trials to be split into training and test instances. As such, 80% of these (1024) is taken as the training set, being composed by 32 trials of each subject.

As explained in Section 3.2, the classifier distinguishes five emotions, numbered between 1 and 5. Each discrete label corresponds to a combination of continuous values of valence and arousal. Therefore, it is required to convert the labels of DEAP dataset to fit the discrete scale. To validate the classification results of the designed IP cores, the ground truth is the converted version of the DEAP dataset, where emotions are labelled as $\{1, 2, 3, 4, 5\}$. This procedure grants the generalization of the classifiers, so that they can be used in any system to identify different emotion classes.

Analysis of classification accuracy using features with variable wordlengths

The optimal wordlength to represent an EEG feature is also investigated. The idea is to find the best trade-off between the classification accuracy and the required hardware resources. Using shorter wordlengths allows to transfer more EEG features into the IP core per cycle. The execution of the

tasks for calculating Canberra distances takes less cycles. Thus, reducing the number of bits required to instantiate a single feature speeds up the algorithm execution. On the other hand, the precision loss leads to errors in results, which may cause obtaining the incorrect shortest distances.

The SW-only implementation uses 64-bit features and it is taken as a reference to the different fixed-point versions considered. Section 4.3.2 presented four versions with variable wordlengths: 64, 32, 16, 8 and 4 bits. These values correspond to divisors of 64 bits, the length of the AXI4-Stream channel. EEG features can be encapsulated following different configurations, up to 8 values per transfer. The presented fixed-point versions provide, respectively, 63, 31, 15, 7 and 3 fractional bits to EEG features. Each version is implemented and tested in HLS to evaluate the effects of reducing wordlength on emotion classification. If the resulting errors are acceptable such that the algorithm requires less hardware resources and is executed in shorter periods of time, those implementations are advantageous candidates to be incorporated in the monitoring system.

Table 4.11 shows the accuracy of emotion classification obtained by four different core versions. These cores differ in the wordlength to represent EEG features. The table registers the number of occurrences of differing classification results, when comparing each version to the 64-bit reference.

The least precise version, using 4-bit words, predicts 105 out of 256 (41.0%) instances of the test set correctly. The reference version (64-bit) has an identical classification rate, which leads one to assume that 4-bit version is an appropriate candidate. However, observing the absolute errors of distances, the 4-bit version has an average of 18 units, corresponding to a relative error of 17% of a variable ranging 110 units, approximately. The classification accuracy in respect to the dataset is similar to the obtained by the remaining versions, due to the characteristics of the training set. The majority of the training instances belong to a single class, which biases the classification results. Therefore, the most relevant metric to compare the effects of wordlength variation is the error of Canberra distance values.

The second-least precise version, using 8-bit words, predicts 102 out of 256 (39.8%) instances of the test set correctly. Regarding the values of Canberra distances, this version has a relative error less than 1%, representing an average absolute error of 0.5 units. The 8-bit version compromises approximately 1.2% of the classification accuracy to reduce memory accesses by a factor of 8, when compared to 64-bit implementation. The latency of the IP core is 20500 clock cycles. The low resource utilization is also an advantage provided by this version. Therefore, the 8-bit version is advantageous in terms of both computing time and resources, and it is the selected one to be implemented in the embedded system.

Resources utilization

Table 4.12 summarizes the resources expected to be used by the pair of EEG IP cores, assuming the 8-bit versions selected previously. The utilization percentages are referred to the targeted platform (Zynq-7010 SoC). The `EEG_CALCDIST` core occupies more resources than `EEG_SORTDIST`, and the most consumed primitives are the LUTs.

Table 4.11: Summary of classification accuracy, errors, resource utilization and latency of four wordlength versions of the EEG IP core. The resource utilization is reported to the core that calculates Canberra distances, `EEG_CALCDIST`.

EEG feature wordlength		64-bit	32-bit	16-bit	8-bit	4-bit
Classification mismatches [b]	reference [a]	n. a.	0	0	6	9
	dataset [c]	105	105	105	102	105
Distance average error [d]	absolute	n. a.	0	4.1×10^{-3}	5.9×10^{-1}	1.8×10^{1}
	relative	n. a.	0	3.8×10^{-5}	5.4×10^{-3}	1.7×10^{-1}
Resources utilization [e] (%)	LUT	45	78	44	29	33
	FF	50	61	32	19	22
	BRAM	1	1	1	1	0
	DSP	0	0	0	0	0
Latency cycles ($\times 10^3$)		164	82	41	20.5	10.2

[a] corresponds to the 64-bit version.
[b] in a total of 256 tests.
[c] ground-truth of DEAP, applied to the emotion map of Figure 3.7.
[d] average error relatively to the values computed by 64-bit version.
[e] of a Zynq-7010 device

Table 4.12: Estimated resources to be used by the EEG IP cores, referred to the Zynq-7010 SoC.

Resources	Available	Utilization	
		`EEG_CALCDIST`	`EEG_SORTDIST`
LUT	17600	5196 (29%)	2717 (15%)
FF	35200	7010 (19%)	1411 (4%)
BRAM	120	2 (1%)	2 (1%)
DSP	80	0 (0%)	0 (0%)

Chapter 5

HW/SW implementation

This chapter addresses the integration of the designed IP cores with the embedded software, and the demonstration of the proposed system on the Zynq device. Section 5.1 introduces the selected development board in which the monitoring system is evaluated. The system integration is explained in Section 5.2. The acceleration results are presented in Section 5.3. Section 5.4 suggests a prototype to test the system using suitable bio-sensors.

5.1 Development board

The proposed system is demonstrated using a development board and the custom hardware, which includes designed IP cores. ZYnq BOard (ZYBO), shown in Figure 5.1, is the selected board to test the operation of the proposed system. This low-cost board includes the Zynq-7010 All Programmable SoC, featuring a 650 MHz dual-core ARM Cortex-A9 processor. Inside the SoC device there are custom reconfigurable hardware blocks which are connected via reconfigurable interconnects. The main blocks are: Configurable Logic Blocks (CLB), which contain LUT and FF, Block RAM and DSP blocks [81]. Moreover, the SoC is appropriated to handle multiple signals simultaneously, processing them in real-time. Multiple peripherals are incorporated in the board, such as audio jacks, MicroSD slot, HDMI, VGA and USB ports [2], that can be explored to extend the functionalities of the biometric system.

ZYBO enables the connection of external sensors, either digital or analog. Digital sensors are connected to the processing system (PS). The connection of analog sensors is more complex. ZYBO contains six sets of Peripheral MODule (Pmod) interfaces that allow the attachment of additional components, namely bio-sensors. A specific Pmod, labelled ”JA”, contains four pairs of analog signal inputs, which can be connected to the Zynq’s Xilinx analog-to-digital converter (XADC). The voltage of the input signals must be limited to 1V peak-to-peak. The 12-bit XADC connects to the PS through AXI bus, and supports a maximum sampling rate of 10^6 samples per second [2] [81].

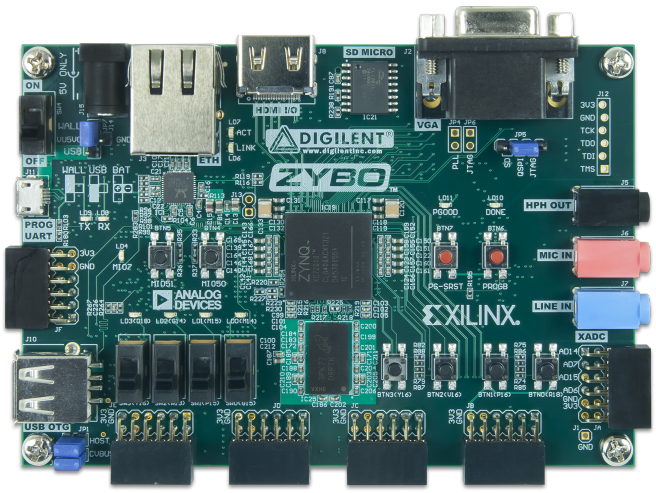


Figure 5.1: ZYBO development board.

5.2 System integration

The biometric system is the integration of the target device (Zynq SoC) with the developed PPG and EEG IP cores. These cores are included in the programmable logic of the SoC. The PPG core receives raw PPG signals and executes the peak detection algorithm to produce heart rate values. The EEG core comprises two distinct modules constituting the foundation of a KNN classifier. The first one (`CALC_DIST`) accepts preprocessed EEG features and computes Canberra distances. The second module (`SORT_DIST`) receives those distances to assess the shortest ones, returning the indexes representing their arrival order. The data flow shared by the IP cores is abstracted in the diagram of Figure 5.2. Heart rate values can be sent to an external device or displayed in a local monitor. Indexes may be loaded into a on-chip-memory, so that the succeeding processing application interprets that information into an emotion classification.

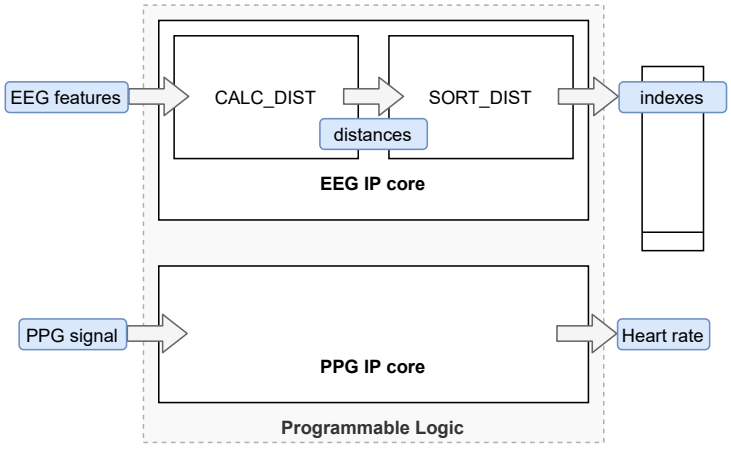


Figure 5.2: Data flow between the designed IP cores.

5.2.1 System description

To allow data transfers between the PS and the IP cores, it is necessary to mediate the connection between these modules through additional components. A block diagram containing the final arrangement of the involved components inside the biometric system is represented in Figure 5.3. This diagram is a simplified version of the Appendix B diagram, reducing the majority of the signals to a generic bus. Both diagrams were generated by Vivado IDE. Next paragraphs describe the purpose of the constituent components.

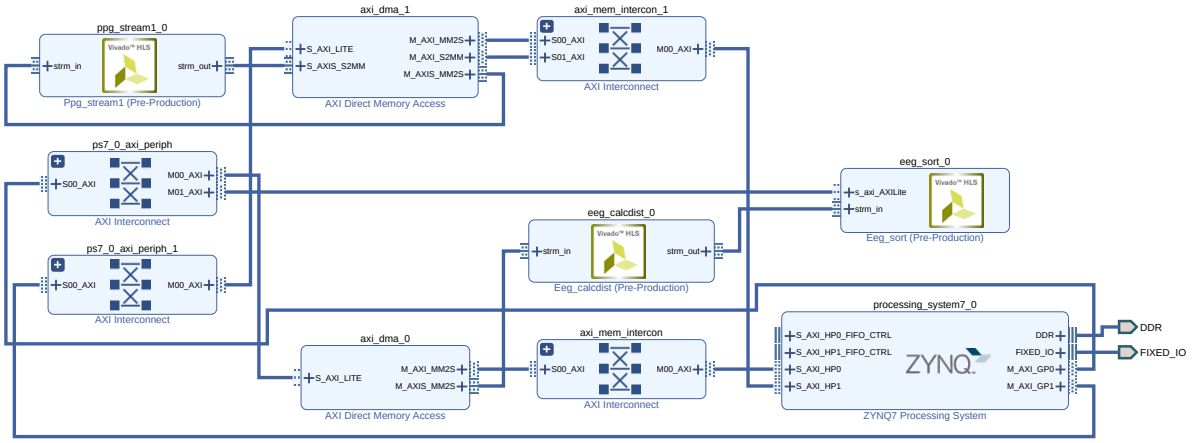


Figure 5.3: Block diagram representing the integration of the biometric system, obtained in Vivado IDE.

The Zynq's PS, located at the bottom right of the diagram, is the diagram's main block. This component is the software interface responsible for managing the data flow depicted in Figure 5.2. The PS contains essential modules and interfaces, such as:

- a dual-core ARM processor to run the embedded software (not represented in the diagram);
- a DDR memory controller to transfer data from external memory;
- two I^2C interfaces to connect peripherals such as bio-sensors (not represented in the diagram);
- four High Performance (HP) AXI slave ports of 32 or 64 bits, to connect to AXI Interconnects with AXI4-Stream transfers;
- two General Purpose (GP) AXI master ports of 32 bits, to connect to AXI Interconnects with AXI4-Lite transfers.

The AXI buses are represented by two AXI Interconnect blocks connected to the HP ports of the PS. These blocks establish a bridge between PS and PL ports. In the diagram of Figure 5.3, AXI Interconnects link PS's HP ports to the AXI4-Stream port of AXI Direct Memory Access (DMA) blocks located in the PL. Also, AXI connects PS's GP ports to AXI DMA's AXI4-Lite ports.

AXI DMA provides a direct high-bandwidth access to the external memory to a AXI4-Stream port. This feature allows to transfer volumes of data without the control of the PS, speeding up data transfers. The block diagram contains two AXI DMA block with different configurations. The bottom one provides a one-way channel to transfer EEG features from the memory to `EEG_CALCDIST` IP core via AXI4-Stream. The top AXI DMA block is a two-way channel, that allows the transfer of PPG samples from the memory to the PPG IP core, but also to return PPG IP core's products to the PS.

The custom IP cores designed in HLS are depicted by three blocks featuring the Vivado HLS logo. These blocks contain their main interfaces represented (AXI4-Lite and AXI4-Stream).

5.2.2 Embedded software

Embedded software targeting the created hardware design is required to coordinate the IP cores with the software instructions and to control specific accesses to the device. The embedded software application is developed using the Vitis IDE tool and run by the processing system. The application coordinates software instructions with IP core calls, being responsible for several tasks, such as:

- specifying the memory addresses and IP core interfaces where data is loaded or retrieved;
- enabling data transfers through DMA;
- triggering the execution of the cores;
- executing software-only instructions, namely determining the PPG's autocorrelation maximum, assigning a class vote to each sorted index and assessing the final results of heart rate and emotion class;
- measuring the execution time of IP cores and pieces of code.

A fundamental issue of the design methodology is the acceleration results obtained by the designed cores, when compared to the software-only implementation. As such, this section provides the elapsed time of running a complete execution of both PPG and EEG algorithms in the ZYBO board. First, the respective SW-only versions are executed by the PS of the Zynq SoC, and the execution times are registered. Then, the projects including the designed IP cores are loaded into the ZYBO board. Each IP core is transferred to the PL of the Zynq, while the embedded software is run by the PS. The execution times related to PPG and EEG algorithms are measured independently. The `XTime` library provides a function for assessing the time from the timer counter registers of ARM, the Zynq's processor [82]. This function, *XTime_GetTime()*, is included in the application to be run by the PS.

Regarding the execution of software-only instructions, it is important to introduce the concept of optimization levels. The applications developed at Vitis IDE are written in C, whose compiler is the GNU Compiler Collections (GCC). Optimization levels define the way the GCC compiler optimizes a source code by, for instance, rearranging the sequence of instructions and attempting to improve the performance [83]. Such optimization consumes memory and compilation time. There are five different levels that offer optimizations with variable intensity. This section addresses two optimization levels: `O0`, denoting the default absence of optimization, and `O3`, which applies a full optimization. An optimization example is the function inlining, meaning that a function call is replaced by its body, rather than being executed by a separate sequence of instructions in memory.

Optimization levels offer the possibility of reducing the execution time of a compiled code. Section 5.3 compares the results obtained by optimized and non-optimized codes implementing the PPG and EEG algorithms.

5.2.3 System performance

The performance of the integrated system described previously is analysed in this section. Namely, the Static Timing Analysis (STA) is performed, to verify that the design of Figure 5.3 meets the timing constraints, before being loaded into the development board. This analysis reports any timing violations observed during signal propagation through design's paths. To understand STA, essential timing concepts are introduced next.

Vivado provides a STA report, where three main sections can be identified: Setup, Width and Pulse Width. Setup refers to the data changes at a destination synchronous element before the clock arrival. The time during which data must be stabilized before a clock edge is called setup time. Hold refers to the data changes at a destination synchronous element after the clock arrival. The time during which data must be stabilized after a clock edge is called hold time. In other words, hold time is the period during which new data must not arrive before the next clock edge. Slack is the difference between the achieved and the projected times for a specific path. A positive slack means that the design is working at the specified clock and there is a margin between both times. A negative slack is a timing violation, meaning that the design does not work correctly at the specified clock.

Inside Setup area, the main requirement to be met is the Worst Negative Slack (WNS). Similarly, Hold's main parameter is the Worst Hold Slack (WHS). The circuit is valid if both WNS and WHS are positive values. Total Negative Slack (TNS) and Total Hold Slack (THS) represent the sum of all WNS and WHS violations, respectively. If their values are zero, the timing constraints are met.

Table 5.1 shows that timing constraints of the complete system featuring PPG and EEG cores are met, using a clock frequency of 100 MHz.

Table 5.1: Summary of timing constraints of the complete monitoring system, reported by Vivado. All timing constraints are met.

Setup	Worst Negative Slack (WNS) Total Negative Slack (TNS)	1.313 ns 0.000 ns
Hold	Worst Hold Slack (WHS) Total Hold Slack (THS)	0.020 ms 0.000 ns
Pulse Width	Worst Pulse Width Slack (WPWS) Total Pulse Width Slack (TPWS)	3.750 ns 0.000 ns

5.2.4 Hardware resources utilization

The hardware resources consumed by the integrated system are listed in Table 5.2. The utilization rates are reported to the available resources of the Zynq-7010's PL. Some observations can be highlighted:

- LUTs are the most used resource, with 51% occupation rate, when compared to FF (30%), DSP (20%) and BRAM (14%);
- `EEG_CALCDIST` IP core takes 32% of the used LUTs and 31% of the used FFs;
- DSPs are only occupied by the PPG IP core;
- the three custom IP cores represent 60% of the consumed LUTs, 53% of the FFs, 47% of the

BRAMs and 100% of the DSPs; this shows that DMA and AXI peripherals demand significant hardware resources;

- overall, the Zynq is not fully occupied, which means that further functionalities may be added to the biometric system.

Table 5.2: Hardware resources used by the complete monitoring system. The block names correspond to the modules of block diagrams in Figures 5.3 and B.1.

Group	Block name	LUT	FF	BRAM	DSP
PPG sub-system	ppg_stream1_0	1319	995	2	16
	ps7_0_axi_periph_1 axi_dma_1 axi_mem_intercon_1	2409	3318	3	0
EEG sub-system	eeg_calc_dist_0	2913	3307	1	0
	eeg_sort_0	1225	1325	1	0
	ps7_0_axi_periph_0 axi_dma_0 axi_mem_intercon_0	1101	1549	1.5	0
Processing system	processing_system7_0	0	0	0	0
	rst_ps7_0_100M	16	33	0	0
	Total used	8983	10527	8.5	16
	Total available	17600	35200	60	80

5.3 Acceleration results

The results obtained by running the embedded solution in the Zynq are described in this section.

5.3.1 PPG sub-system

The processing of raw PPG signals by the PPG sub-system comprises two main stages: preprocessing and periodicity search. The first stage is executed by the designed IP core, present at the PL. The second stage is executed by the PS and recurring calls of the IP core. Table 5.3 shows the total elapsed time of a complete execution of the PPG algorithm, discriminating the split times of both stages. The times are referred to input PPG signals comprising two buffers of 1024 16-bit samples. These buffers are shared with the channels of an optoelectronic sensor that collects PPG data. The values of the table include the application of `O0` and `O3` optimizations. Regarding the non-optimized versions, the embedded system (`HW/SW O0`) outperforms the results of the software-only version (`SW O0`). The overall execution time was reduced by 64%, while the preprocessing and periodicity search stages were respectively reduced by 86% and 58%. These values correspond to a speedup ranging between 2.4 and 7.4. The `O3` optimization applied to the HW/SW design (`HW/SW O3`) increased the overall execution time of the equivalent software-only (`SW O3`) by 58%. This is due to the 90% increase of the execution time of the periodicity search stage. However, the preprocessing stage is outperformed and its execution time reduced by 50% (speedup of 2 times).

Table 5.3: Execution times, in μs, obtained by software-only (SW) and hardware/software (HW/SW) implementations of the PPG sub-system. The execution times correspond to one complete execution of the heart rate calculator. The speedup of the HW/SW implementations are referred to the SW times.

Processing stage	SW		HW/SW (*speedup*)	
	O0	O3	O0	O3
Preprocessing	451	99	61 (7.4)	48 (2.1)
Periodicity search	1709	340	722 (2.4)	645 (0.53)
Total	2160	439	783 (2.8)	693 (0.63)

The proposed way of tackling the preprocessing tasks has improved the performance, reducing the execution times achieved by the software baselines. However, the periodicity search has not been enhanced by the HW/SW implementation, which has room for improvement. The observed underperformance is motivated by the dependence of the periodicity task on control instructions executed between autocorrelation computations. Once an autocorrelation value is calculated, the algorithm assesses the next iteration step by comparing the computed value with the previous one, or a threshold. Moreover, the delay to which the autocorrelation refers has to be updated. The periodicity search does not follow a continuous data flow, as opposed to the preprocessing stage, since the computational tasks are intercalated with control logic.

5.3.2 EEG sub-system

The EEG embedded system is a KNN classifier composed by a pair of IP cores, dedicated to the calculation and sorting of Canberra distances between sets of EEG features. Because of the direct connection between first core's output and second core's input, the PS does not interact with the results obtained by the first core. Therefore, the measurement of the execution time of calculation and sorting stages is done jointly. The PS is responsible for assessing the classification given the results produced by the IP cores pair.

Table 5.4 summarizes the execution times of the processing steps, applied to optimized and non-optimized implementations. The high number of operations to be executed over a memory (training set) containing 1024 sets of 160 EEG features created an opportunity to acceleration via HW. The results show that the HW/SW co-design outperforms the SW-only O0 baseline by 100 times and the O3 version by 40. The problem of calculating distances was approached by launching eight instances of *Canberra* blocks to execute in parallel the correspondent arithmetic instructions. Moreover, the sorting task was unlocked by the concept of a chain of sorting cells through which data (distances) propagated continuously. The developed design is suitable to perform KNN classification exercises handling high volumes of data in real-time.

Table 5.4: Execution times, in μs, obtained by software-only (SW) and hardware/software (HW/SW) implementations of the EEG sub-system. The execution times correspond to one complete execution of the KNN emotion classifier. The speedups of the HW/SW implementations are referred to the SW times.

Stage	SW		HW/SW (*speedup*)	
	O0	O3	O0	O3
Distances calculation	24130	8593	235.4 (100)	217.9 (41)
Distances sort	896.6	309.6		
Classification	1.67	0.51	14.67 (0.11)	4.23 (0.12)
Total	25028	8903.3	250.03 (100)	222.22 (40)

5.4 Prototype concept

The system described in Section 5.2 performs two main tasks: heart rate calculation from PPG and emotion classification from EEG features. Two IP cores were designed to execute these tasks. The PPG IP core implements a full algorithm, processing raw collected signals, to obtain the final result. The EEG IP core is dedicated to the classification step, a specific task of the complex process of emotion recognition.

To build an operational prototype, besides the PPG and EEG IP cores, it is necessary to develop an additional block that processes raw EEG signals and obtains EEG features. This block, called "EEG preprocessing", works as a Digital Signal Processor (DSP) integrated in the PS, for instance. Taking into account that EEG signals are collected by analog sensors, an analog-to-digital converter is also required. Moreover, a connection to the sensors and a connection to a Bluetooth module to support wireless communication must be established. With these components added, the bio-sensors can be attached to the system, constituting the prototype concept of Figure 5.4.

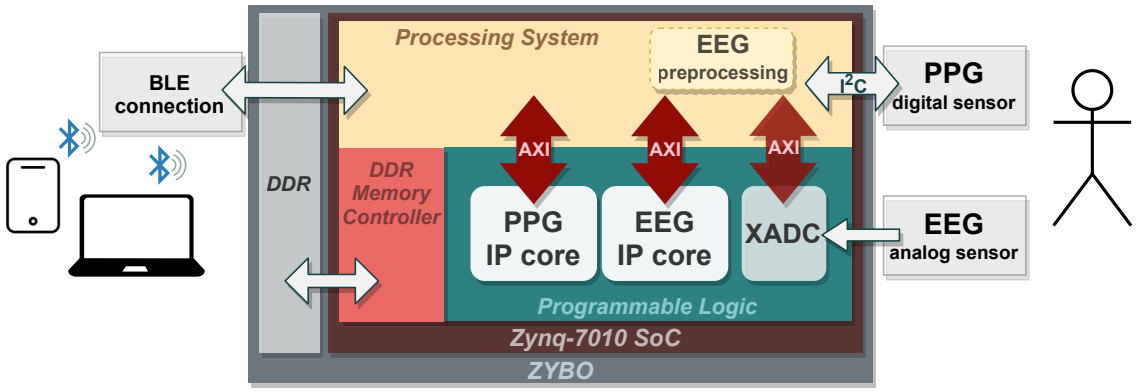


Figure 5.4: Materialization of the implemented system into a prototype.

A block representing a BLE module is suggested, so that the main results produced in the end of each processing cycle are sent to an external device. This technology presents low power consumption, being advantageous to transfer reduced data buffers in proximity of a host computer or mobile phone. Assuming that an user's heart rate is computed each second and their emotional state is assessed every five seconds, it means that, per second, are sent:

- 1 byte representing a 8-bit heart rate value;

- $\frac{3}{5}$ bytes corresponding to emotion classes of 3 bits.

In this example, the prototype throughput is 1.6 bytes per second.

The figure includes a DDR block representing an external memory. It may be used to load the training set of the KNN classifier and to store the indexes produced by the EEG core. A XADC block integrates the PL to enable the analog-to-digital conversion. Thanks to this component, EEG signals can be quantized and handled by the PS, which in turn transfers the digital signal to the EEG IP core. These transfers are mediated by AXI buses.

The biometric sensors recommended to be used are the Maxim Integrated's MAX3010x[1] and Olimex's passive EEG electrodes[2], displayed in Figure 5.5. MAX3010x is a low-cost pulse oximeter operated by light reflection, thus enabling PPG digital signal acquisition. This sensor is reliable for measuring heart rate and oxygen saturation in the blood, as mentioned in Section 2.2.2. The EEG electrodes are inexpensive devices for recording EEG analog signals. Olimex also provides the EEG-SMT kit[3], a preprocessing module supported by open-source software to visualize EEG signals. This kit supports four electrodes and it is useful to understand how EEG waves behave.

(a) MAX30100 sensor

(b) EEG electrode

Figure 5.5: Biometric sensors.

[1] MAX3010x webpage: `https://www.maximintegrated.com/en/design/technical-documents/userguides-and-manuals/6/6409.html`; accessed on 1st June 2020.

[2] EEG-PE webpage: `https://www.olimex.com/Products/EEG/Electrodes/EEG-PE/`; accessed on 1st June 2020.

[3] EEG-SMT webpage: `https://www.olimex.com/Products/EEG/OpenEEG/EEG-SMT/`; accessed on 1st June 2020.

Chapter 6

Conclusions

A biometric system for monitoring vital signals in the context of remote assistance was presented. The proposed system is comprised of two sub-systems designed to compute instant heart rate values and to determine the emotional state of a user. The main goal of the thesis was to study the implementation of the sub-systems on an HW/SW embedded system, targeting a SoC FPGA, to accelerate the execution of the proposed tasks. Such approach is an alternative to both general-purpose unit and microcontroller-based applications. The sub-systems included hardware blocks dedicated to the most critical signal processing routines. Heart rates are estimated processing PPG signals acquired by a digital sensor, whereas emotions are assessed using a KNN classifier that operates over already computed EEG features. Globally, the acceleration objective has been successfully achieved by the EEG sub-system, in contrast to the PPG sub-system.

The system included IP cores dedicated to the execution of each sub-system's tasks, targeting the Xilinx's Zynq-7010 SoC platform. The development of each sub-system's IP cores followed a common process. First, the algorithm for processing PPG signals and the classifier to map EEG features into emotions were selected according to the state-of-the-art procedures and open-source repositories. More particularly, the EEG classifier was mapped into a software specification using a low-level language, taking as a reference a higher level implementation. The algorithms were analysed to understand the data flow and to identify the critical tasks. The selected functions were isolated and converted into custom IP cores. The development of the IP cores involved optimization processes, such as reducing the number of IP accesses, the variables' wordlength and the circuit's latency. Dimensioning the wordlengths involved a trade-off between the computation accuracy, the hardware resources consumed and the time required to execute the processing tasks. The cores, located in the Zynq's programmable logic, were integrated with embedded software containing instructions executed by the processing system.

To evaluate the performance of the proposed system architecture, the execution times obtained by both sub-systems were compared to the software-only benchmarks, and the number of consumed resources was estimated. The results shown that the proposed PPG sub-system executed the preprocessing stage 2 times faster than software-only and performed the periodicity search 2 times longer. The gap observed at the periodicity search is due to the dependency of its execution on control instruc-

tions, rather than heavy computational tasks. The execution of the preprocessing stage of the heart rate calculator takes $48\ \mu s$. The results obtained by the EEG sub-system are more promising. Overall, the classification of a single emotion by the proposed EEG sub-system outperformed the software benchmark by 40 times. The computation and sort of distances benefit from the integration of dedicated hardware blocks in a hybrid HW/SW architecture. The analysed example required $250\ \mu s$ to translate EEG features into an emotion, which confirms the possibility of performing a classification in real-time, at the node.

This work studied the impact of reducing both the wordlength of the input data and the precision of internal IP cores' operators. A reduced wordlength allowed to compress the data, thus reducing the number of memory accesses and execution times. Dimensioning the IP core operators with fewer fractional bits allowed the IP cores to consume less resources. This has resulted in a cost of having increased of the errors of arithmetic calculations, which were tolerated depending on the context of the final result.

Regarding the hardware utilization, the proposed biometric system is feasible to be implemented with the resources available in the targeted platform. The occupation rate of the Zynq-7010's primitive blocks is 36%. There is room for upgrading the developed IP cores and for implementing additional processing modules. The IP cores were designed to be reused in further monitoring systems. The PPG IP core may be integrated in different algorithms besides heart rate calculation. For instance, the specification of the preprocessing task can be exploited in multiple PPG-based applications. Moreover, the EEG IP core is prepared to process data from up to 32 EEG electrodes, supporting the implementation of multi-channel systems in portable devices. This work is a starting point of the development of more complex biometric systems that may offer autonomy, portability and high processing capability to wearable monitoring devices.

6.1 System improvements and future work

The proposed system provides the foundation of a more complex wearable device targeting a medical remote application. Introducing a connection to the bio-sensors (an optoelectronic sensor and EEG electrodes) and the DSP block for EEG signal preprocessing would be sufficient to have a fully operational system. This way, an improvement regarding the EEG sub-system is the development of a processing module of EEG signals. This module would handle the preprocessing stage, which includes noise removal, signal enhancement and decomposing the signal into the major frequency bands to extract the relevant patterns. The preprocessing module returns the EEG features that are loaded into the KNN classifier. The raw EEG signals acquired by the analog electrodes require an analog-to-digital conversion, before being preprocessed. This step is carried out by the Zynq's XADC. Moreover, the future work may include the study of the implementation of a SVM classifier on a similar HW/SW embedded system, due to its broad use in the literature in emotion recognition applications.

The results obtained by the developed PPG IP core suggest a future improvement of the PPG subsystem. The algorithm's routine of detecting the peak of PPG signals alternates between control instruc-

tions and computation of autocorrelation values. This behaviour explains the deceleration observed when comparing the execution times of the software-only benchmark to the obtained by the PPG IP core. An alternate approach would be to start by tackling the computational tasks necessary to obtain autocorrelation values, followed by the execution of the control instructions, such as the comparison between the computed values. This method implies the computation of the autocorrelation values for each possible delay, once the PPG signal is preprocessed. This approach offers the possibility of executing the autocorrelation function concurrently, leaving the peak detection for a later stage, selecting the appropriate autocorrelation values. Moreover, the execution of the autocorrelation function could be improved using the partial results calculated in previous executions. For instance, the computation of the autocorrelation value at a given delay m executes iteration steps common to the ones performed when computing the autocorrelation at the preceding delay $m - 1$. If autocorrelation is calculated at adjacent delays, it is reasonable to see the autocorrelation function as a first-in, first-out method. This approach allows to reuse the partial results of autocorrelation function, when applied to near delays, and thus possibly reducing the execution time.

Bibliography

[1] A. J. Casson. Wearable EEG and beyond. *Biomedical Engineering Letters*, 9(1):53–71, 2019. ISSN 2093985X. doi: 10.1007/s13534-018-00093-6.

[2] Digilent. Zybo Reference Manual - Digilent Reference. URL `https://reference.digilentinc.com/programmable-logic/zybo/reference-manual`.

[3] N. Jatupaiboon, S. Pan-Ngum, and P. Israsena. Real-time EEG-based happiness detection system. *The Scientific World Journal*, 2013, 2013. ISSN 1537744X. doi: 10.1155/2013/618649.

[4] A. Khosla, P. Khandnor, and T. Chand. A comparative analysis of signal processing and classification methods for different applications based on EEG signals. *Biocybernetics and Biomedical Engineering*, 40(2):649 – 690, 2020. ISSN 0208-5216. doi: 10.1016/j.bbe.2020.02.002.

[5] G. H. Klem, H. O. Lüders, H. Jasper, C. Elger, et al. The ten-twenty electrode system of the International Federation. *Electroencephalogr Clin Neurophysiol*, 52(3):3–6, 1999.

[6] V. Jurcak, D. Tsuzuki, and I. Dan. 10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems. *NeuroImage*, 34(4):1600–1611, 2007. ISSN 10538119. doi: 10.1016/j.neuroimage.2006.09.024.

[7] J. A. Russell. A Circumplex Model of Affect. *Journal of personality and social psychology*, 39(6): 1161, 1980.

[8] L. Parisi, S. Francia, S. Olivastri, and M. Tavella. Exploiting Synchronized Lyrics And Vocal Features For Music Emotion Detection. 01 2019.

[9] R. J. Davidson, P. Ekman, C. D. Saron, J. A. Senulis, and W. V. Friesen. Approach-withdrawal and cerebral asymmetry: emotional expression and brain physiology: I. *Journal of personality and social psychology*, 58(2):330, 1990.

[10] T. J. Eric Kandel, James Schwartz. *Principles of Neural Science*. McGraw-Hill Medical, 4 edition, 2000. ISBN 9780838577011,0838577016.

[11] M. Lakshmi, D. Prasad, and D. Prakash. Survey on EEG signal processing methods. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(1):84–91, 2014.

[12] S. M. Alarcao and M. J. Fonseca. Emotions Recognition Using EEG Signals: A Survey. *IEEE Transactions on Affective Computing*, 10(3):374–393, 2017. ISSN 19493045. doi: 10.1109/TAFFC.2017.2714671.

[13] M. Z. Ilyas, P. Saad, and M. I. Ahmad. A survey of analysis and classification of EEG signals for brain-computer interfaces. *Proceedings - 2015 2nd International Conference on Biomedical*

Engineering, ICoBE 2015, (March):1–6, 2015. doi: 10.1109/ICoBE.2015.7235129.

[14] F. Lotte, L. Bougrain, A. Cichocki, M. Clerc, M. Congedo, A. Rakotomamonjy, and F. Yger. A review of classification algorithms for EEG-based brain-computer interfaces: A 10 year update. *Journal of Neural Engineering*, 15(3), 2018. ISSN 17412552. doi: 10.1088/1741-2552/aab2f2.

[15] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411 – 430, 2000. ISSN 0893-6080. doi: 10.1016/S0893-6080(00)00026-5.

[16] K. A. Ludwig, R. M. Miriani, N. B. Langhals, M. D. Joseph, D. J. Anderson, and D. R. Kipke. Using a Common Average Reference to Improve Cortical Neuron Recordings From Microelectrode Arrays. *Journal of Neurophysiology*, 101(3):1679–1689, 2009. doi: 10.1152/jn.90989.2008. PMID: 19109453.

[17] C. Carvalhaes and J. A. de Barros. The surface Laplacian technique in EEG: Theory and methods. *International Journal of Psychophysiology*, 97(3):174–188, 2015.

[18] X. Yu, P. Chum, and K.-B. Sim. Analysis the effect of PCA for feature reduction in non-stationary EEG based motor imagery of BCI system. *Optik*, 125(3):1498 – 1502, 2014. ISSN 0030-4026. doi: 10.1016/j.ijleo.2013.09.013.

[19] L. F. Velásquez-Martínez, A. M. Álvarez-Meza, and C. G. Castellanos-Domínguez. Motor imagery classification for bci using common spatial patterns and feature relevance analysis. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 365–374. Springer, 2013.

[20] D. Bansal and R. Mahajan. Eeg-based brain-computer interfacing (bci). *EEG-Based Brain-Computer Interfaces: Cognitive Analysis and Control Applications*, page 21, 2019.

[21] D. Cvetkovic, E. D. Übeyli, and I. Cosic. Wavelet transform feature extraction from human PPG, ECG, and EEG signal responses to ELF PEMF exposures: A pilot study. *Digital Signal Processing*, 18(5):861 – 874, 2008. ISSN 1051-2004. doi: 10.1016/j.dsp.2007.05.009.

[22] W. Ting, Y. Guo-zheng, Y. Bang-hua, and S. Hong. EEG feature extraction based on wavelet packet decomposition for brain computer interface. *Measurement*, 41(6):618 – 625, 2008. ISSN 0263-2241. doi: 10.1016/j.measurement.2007.07.007.

[23] V. Lawhern, W. D. Hairston, K. McDowell, M. Westerfield, and K. Robbins. Detection and classification of subject-generated artifacts in EEG signals using autoregressive models. *Journal of Neuroscience Methods*, 208(2):181 – 189, 2012. ISSN 0165-0270. doi: 10.1016/j.jneumeth.2012.05.017.

[24] S. Kanoga and Y. Mitsukura. Review of Artifact Rejection Methods for Electroencephalographic Systems. *Electroencephalography*, page 69, 2017.

[25] A. A. Chi Qin Lai, Haidi Ibrahim, Mohd Zaid Abdullah, Jafri Malin Abdullah, Shahrel Azmin Suandi. Literature survey on applications of electroencephalography (EEG). 020070(September), 2018.

[26] Y. P. Lin, C. H. Wang, T. L. Wu, S. K. Jeng, and J. H. Chen. EEG-based emotion recognition in music listening: A comparison of schemes for multiclass support vector machine. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 489–492, 2009. ISSN 15206149. doi: 10.1109/icassp.2009.4959627.

[27] A. E. Vijayan, D. Sen, and A. P. Sudheer. EEG-Based Emotion Recognition Using Statistical Mea-

sures and Auto-Regressive Modeling. In *2015 IEEE International Conference on Computational Intelligence Communication Technology*, pages 587–591, Feb 2015. doi: 10.1109/CICT.2015.24.

[28] K. Dhindsa and S. Becker. Emotional reaction recognition from EEG. *2017 International Workshop on Pattern Recognition in Neuroimaging, PRNI 2017*, pages 1–4, 2017. doi: 10.1109/PRNI.2017.7981501.

[29] Y. J. Liu, M. Yu, G. Zhao, J. Song, Y. Ge, and Y. Shi. Real-time movie-induced discrete emotion recognition from EEG signals. *IEEE Transactions on Affective Computing*, 9(4):550–562, 2018. ISSN 19493045. doi: 10.1109/TAFFC.2017.2660485.

[30] V. H. Anh, M. N. Van, B. B. Ha, and T. H. Quyet. A real-time model based Support Vector Machine for emotion recognition through EEG. *2012 International Conference on Control, Automation and Information Sciences, ICCAIS 2012*, pages 191–196, 2012. doi: 10.1109/ICCAIS.2012.6466585.

[31] C. C. Chang and C. J. Lin. LIBSVM: A Library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 2011. ISSN 21576904. doi: 10.1145/1961189.1961199.

[32] K. Schaaff and T. Schultz. Towards emotion recognition from electroencephalographic signals. *Proceedings - 2009 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops, ACII 2009*, pages 1–6, 2009. doi: 10.1109/ACII.2009.5349316.

[33] M. Ali, A. H. Mosa, F. Al Machot, and K. Kyamakya. EEG-based emotion recognition approach for e-healthcare applications. *International Conference on Ubiquitous and Future Networks, ICUFN*, 2016-Augus:946–950, 2016. ISSN 21658536. doi: 10.1109/ICUFN.2016.7536936.

[34] S. Koelstra, C. Mühl, M. Soleymani, J. S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras. DEAP: A database for emotion analysis; Using physiological signals. *IEEE Transactions on Affective Computing*, 3(1):18–31, 2012. ISSN 19493045. doi: 10.1109/T-AFFC.2011.15.

[35] Y. Wei, Y. Wu, and J. Tudor. A real-time wearable emotion detection headband based on EEG measurement. *Sensors and Actuators, A: Physical*, 263:614–621, 2017. ISSN 09244247. doi: 10.1016/j.sna.2017.07.012.

[36] A. Chatchinarat, K. W. Wong, and C. C. Fung. A comparison study on the relationship between the selection of EEG electrode channels and frequency bands used in classification for emotion recognition. *Proceedings - International Conference on Machine Learning and Cybernetics*, 1: 251–256, 2016. ISSN 21601348. doi: 10.1109/ICMLC.2016.7860909.

[37] M. Li, H. Xu, X. Liu, and S. Lu. Emotion recognition from multichannel EEG signals using K-nearest neighbor classification. *Technology and health care*, 26(S1):509–519, 2018.

[38] Z. Mohammadi, J. Frounchi, and M. Amiri. Wavelet-based emotion recognition system using EEG signal. *Neural Computing and Applications*, 28(8):1985–1990, 2017.

[39] S. Hatamikia, K. Maghooli, and A. M. Nasrabadi. The emotion recognition system based on autoregressive model and sequential forward feature selection of electroencephalogram signals. *Journal of medical signals and sensors*, 4(3):194, 2014.

[40] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[41] K. Dhindsa. Filter-Bank Artifact Rejection: High performance real-time single-channel artifact de-

tection for EEG. *Biomedical Signal Processing and Control*, 38:224 – 235, 2017. ISSN 1746-8094. doi: 10.1016/j.bspc.2017.06.012.

[42] A. Delorme and S. Makeig. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods*, 134(1):9 – 21, 2004. ISSN 0165-0270. doi: 10.1016/j.jneumeth.2003.10.009.

[43] J. Allen. Photoplethysmography and its application in clinical physiological measurement. *Physiological Measurement*, 28(3), 2007. ISSN 09673334. doi: 10.1088/0967-3334/28/3/R01.

[44] T. Tamura, Y. Maeda, M. Sekine, and M. Yoshida. Wearable photoplethysmographic sensors—past and present. *Electronics*, 3(2):282–302, 2014. ISSN 20799292. doi: 10.3390/electronics3020282.

[45] J. L. Moraes, M. X. Rocha, G. G. Vasconcelos, J. E. Vasconcelos Filho, V. H. C. de Albuquerque, and A. R. Alexandria. Advances in photopletysmography signal analysis for biomedical applications. *Sensors (Switzerland)*, 18(6):1–26, 2018. ISSN 14248220. doi: 10.3390/s18061894.

[46] D. Castaneda, A. Esparza, M. Ghamari, C. Soltanpur, and H. Nazeran. A review on wearable photoplethysmography sensors and their potential future applications in health care. *Physiology & behavior*, 176(12):139–148, 2017. doi: 10.1016/j.physbeh.2017.03.040.

[47] Y. Maeda, M. Sekine, and T. Tamura. The advantages of wearable green reflected photoplethysmography. *Journal of Medical Systems*, 35(5):829–834, 2011. ISSN 01485598. doi: 10.1007/s10916-010-9506-z.

[48] K. Matsumura, P. Rolfe, J. Lee, and T. Yamakoshi. iPhone 4s photoplethysmography: Which light color yields the most accurate heart rate and normalized pulse volume using the iPhysioMeter application in the presence of motion artifact? *PLoS ONE*, 9(3):1–12, 2014. ISSN 19326203. doi: 10.1371/journal.pone.0091205.

[49] P.-Y. Chiang, P. C.-P. Chao, T.-Y. Tu, Y.-H. Kao, C.-Y. Yang, D.-C. Tarng, and C.-L. Wey. Machine Learning Classification for Assessing the Degree of Stenosis and Blood Flow Volume at Arteriovenous Fistulas of Hemodialysis Patients Using a New Photoplethysmography Sensor Device. *Sensors*, 19(15):3422, 2019.

[50] D. Chakraborty and J. B. Jeeva. Detection of Heart Rate Using Reflectance Mode Photoplethysmography. *2019 Innovations in Power and Advanced Computing Technologies, i-PACT 2019*, pages 1–3, 2019. doi: 10.1109/i-PACT44901.2019.8960204.

[51] D.-G. Jang, S. Park, M. Hahn, and S.-H. Park. A Real-Time Pulse Peak Detection Algorithm for the Photoplethysmogram. *International Journal of Electronics and Electrical Engineering*, 2(1):45–49, 2014. ISSN 2301380X. doi: 10.12720/ijeee.2.1.45-49.

[52] W. Zong, T. Heldt, G. B. Moody, and R. G. Mark. An open-source algorithm to detect onset of arterial blood pressure pulses. *Computers in Cardiology*, 30:259–262, 2003. ISSN 02766574. doi: 10.1109/cic.2003.1291140.

[53] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13).

[54] S. Das, S. Pal, and M. Mitra. Real time heart rate detection from PPG signal in noisy environment. *2016 International Conference on Intelligent Control, Power and Instrumentation, ICICPI 2016*, pages 70–73, 2017. doi: 10.1109/ICICPI.2016.7859676.

[55] G. B. Moody and R. G. Mark. A database to support development and evaluation of intelligent intensive care monitoring. In *Computers in Cardiology 1996*, pages 657–660, Sep. 1996. doi: 10.1109/CIC.1996.542622.

[56] P. van Gent, H. Farah, N. van Nes, and B. van Arem. Analysing noisy driver physiology real-time using off-the-shelf sensors: Heart rate analysis software from the taking the fast lane project. *Journal of Open Research Software*, 7(1), 2019. ISSN 20499647. doi: 10.5334/jors.241.

[57] M. A. F. Pimentel, A. E. W. Johnson, P. H. Charlton, D. Birrenkott, P. J. Watkinson, L. Tarassenko, and D. A. Clifton. Toward a Robust Estimation of Respiratory Rate From Pulse Oximeters. *IEEE Transactions on Biomedical Engineering*, 64(8):1914–1923, 2017.

[58] A. Temko. Accurate Heart Rate Monitoring during Physical Exercises Using PPG. *IEEE Transactions on Biomedical Engineering*, 64(9):2016–2024, 2017. ISSN 15582531. doi: 10.1109/TBME.2017.2676243.

[59] Z. Zhang, Z. Pi, and B. Liu. TROIKA: A General Framework for Heart Rate Monitoring Using Wrist-Type Photoplethysmographic Signals During Intensive Physical Exercise. *IEEE Transactions on Biomedical Engineering*, 62(2):522–531, 2015.

[60] Z. Zhang. Photoplethysmography-Based Heart Rate Monitoring in Physical Activities via Joint Sparse Spectrum Reconstruction. *IEEE Transactions on Biomedical Engineering*, 62(8):1902–1910, 2015.

[61] S. Salehizadeh, D. Dao, J. Bolkhovsky, C. Cho, Y. Mendelson, and K. Chon. A Novel Time-Varying Spectral Filtering Algorithm for Reconstruction of Motion Artifact Corrupted Heart Rate Signals During Intense Physical Activities Using a Wearable Photoplethysmogram Sensor. *Sensors*, 16(1):10, dec 2015. ISSN 1424-8220. doi: 10.3390/s16010010.

[62] E. Khan, F. Al Hossain, S. Z. Uddin, S. K. Alam, and M. K. Hasan. A Robust Heart Rate Monitoring Scheme Using Photoplethysmographic Signals Corrupted by Intense Motion Artifacts. *IEEE Transactions on Biomedical Engineering*, 63(3):550–562, 2016.

[63] M. Boloursaz Mashhadi, E. Asadi, M. Eskandari, S. Kiani, and F. Marvasti. Heart Rate Tracking using Wrist-Type Photoplethysmographic (PPG) Signals during Physical Exercise with Simultaneous Accelerometry. *IEEE Signal Processing Letters*, 23(2):227–231, 2016.

[64] J. Xiong, L. Cai, D. Jiang, H. Song, and X. He. Spectral Matrix Decomposition-Based Motion Artifacts Removal in Multi-Channel PPG Sensor Signals. *IEEE Access*, 4:3076–3086, 2016.

[65] H. Sharma. Heart rate extraction from PPG signals using variational mode decomposition. *Biocybernetics and Biomedical Engineering*, 39(1):75–86, 2019. ISSN 02085216. doi: 10.1016/j.bbe.2018.11.001.

[66] A. Garde, W. Karlen, J. M. Ansermino, and G. A. Dumont. Estimating respiratory and heart rates from the correntropy spectral density of the photoplethysmogram. *PloS one*, 9(1):e86427, 2014.

[67] A. Garde, W. Karlen, P. Dehkordi, J. Ansermino, and G. Dumont. Empirical mode decomposition

for respiratory and heart rate estimation from the photoplethysmogram. In *Computing in Cardiology 2013*, pages 799–802, 2013.

[68] M. A. Motin, C. K. Karmakar, and M. Palaniswami. Ensemble Empirical Mode Decomposition With Principal Component Analysis: A Novel Approach for Extracting Respiratory Rate and Heart Rate From Photoplethysmographic Signal. *IEEE Journal of Biomedical and Health Informatics*, 22(3): 766–774, 2018.

[69] P. H. Charlton, M. Villarroel, and F. Salguiero. Waveform analysis to estimate respiratory rate. In *Secondary Analysis of Electronic Health Records*, pages 377–390. Springer International Publishing, jan 2016. ISBN 9783319437422. doi: 10.1007/978-3-319-43742-2_26.

[70] A. Schäfer and K. W. Kratky. Estimation of breathing rate from respiratory sinus arrhythmia: Comparison of various methods. *Annals of Biomedical Engineering*, 36(3):476–485, mar 2008. ISSN 00906964. doi: 10.1007/s10439-007-9428-1.

[71] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L.-W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark. Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): a public-access intensive care unit database. *Critical care medicine*, 39(5):952, 2011.

[72] Y. Kong and K. H. Chon. Heart Rate Tracking Using a Wearable Photoplethysmographic Sensor during Treadmill Exercise, 2019. ISSN 21693536.

[73] A. Onubeze. Developing a Wireless Heart-Rate Monitor with MAX30100 and nRF51822. 2016.

[74] J. Wan, Y. Zou, Y. Li, and J. Wang. Reflective type blood oxygen saturation detection system based on MAX30100. In *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 615–619, 2017.

[75] R. Joaquinito. A Wireless Biosignal Measurement System using the Zynq SoC and Bluetooth Low Energy. Master's thesis, Instituto Superior Técnico, 2016.

[76] S. Knežević, R. Stojanović, B. Ašanin, and D. Karadaglić. A single chip solution for pulse transmit time measurement. *13th IEEE International Conference on BioInformatics and BioEngineering, IEEE BIBE 2013*, pages 13–16, 2013. doi: 10.1109/BIBE.2013.6701625.

[77] W. C. Fang, K. Y. Wang, N. Fahier, Y. L. Ho, and Y. D. Huang. Development and Validation of an EEG-Based Real-Time Emotion Recognition System Using Edge AI Computing Platform With Convolutional Neural Network System-on-Chip Design. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(4):645–657, 2019. ISSN 21563365. doi: 10.1109/JETCAS.2019.2951232.

[78] *High-Level Synthesis. Vivado Design Suite User Guide*. Xilinx, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug902-vivado-high-level-synthesis.pdf`, v2019.1 edition, 2019.

[79] R. Kastner, J. Matai, and S. Neuendorffer. Parallel Programming for FPGAs. *ArXiv e-prints*, May 2018.

[80] Xilinx and Inc. SDx Pragma Reference Guide, 2019. URL `https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/nnz1534452175410.html`.

[81] Xilinx and Inc. Zynq-7000 SoC First Generation Architecture, 2012.

[82] Xilinx and Inc. Xilinx Standalone Library Documentation OS and Libraries Document Collection. Technical report, 2019.

[83] R. M. Stallman. Using the GNU Compiler Collection. Technical report, 2008. URL https://www.gnupress.org.

Appendix A

Dimensioning of PPG IP core internal variables

Table A.1: Extended table containing the dimensioning of the PPG IP core internal variables, with different wordlengths. Variables are represented by fixed-point notation, <W,I>. W denotes the wordlength and I the number of integer bits. Six versions are considered, varying in the number of bits dedicated to variables' fractional part. Bold rows represent input (a[], b[]) and output (prePcorr, sumsq_x, sq_prod) data.

Variable	Meaning	Size					
		V8	V9	V10	V11	V12	V13
a[]	**raw RED**	**u<16,16>**					
b[]	**raw IR**	**u<16,16>**					
red[]	processed RED	<58,26>	<32,18>	<26,18>	<22,18>	<20,18>	<18,18>
ir[]	processed IR	<58,26>	<32,18>	<26,18>	<22,18>	<20,18>	<18,18>
regA	accumulator(a)	u<26,26>					
regB	accumulator(b)	u<26,26>					
inv_bsize	(1/BUFFERSIZE)	u<32,1>			u<11,1>		
red_mean	regA*inv_nsize	u<57,26>	u<32,16>	u<24,16>	u<20,16>	u<18,18>	u<16,16>
ir_mean	regB*inv_nsize	u<57,26>	u<32,16>	u<24,16>	u<20,16>	u<18,18>	u<16,16>
xmean	(-BUFFERSIZE/2+0.5)	u<11,10>					
prod	xmean*red	<69,36>	<38,26>	<34,26>	<30,26>	<28,26>	<26,26>
regC	BUFFERSIZE*prod	<79,46>	<42,30>	<38,30>	<34,30>	<32,30>	<30,30>
regD	BUFFERSIZE*prod	<79,46>	<42,30>	<38,30>	<34,30>	<32,30>	<30,30>
inv_sumx2	(1/SUM_X2)	u<32,1>			u<29,1>		
betaX	regC*inv_sumx2	<79,20>	<32,5>		<24,5>		
betaY	regD*inv_sumx2	<79,20>	<32,5>		<24,5>		
prod2	beta*xmean	<90,30>	<32,13>	<21,13>	<17,13>	<15,13>	<13,13>
prod3	red*red	<116,58>	<44,32>	<40,32>	<36,32>	<34,32>	<32,32>
regAA	BUFFERSIZE*prod3	<126,68>	<52,40>	<48,40>	<44,40>	<42,40>	<40,40>
regAB	BUFFERSIZE*prod3	<126,68>	<52,40>	<48,40>	<44,40>	<42,40>	<40,40>
regBB	BUFFERSIZE*prod3	<126,68>	<52,40>	<48,40>	<44,40>	<42,40>	<40,40>
prePcorr	**regAB*inv_nsize**	**<116,58>**	**<42,30>**	**<38,30>**	**<34,30>**	**<32,30>**	**<30,30>**
prod4	regBB*inv_nsize	<116,58>	<42,30>	<38,30>	<34,30>	<32,30>	<30,30>
sumsq_x	**regAA*inv_nsize**	**<116,58>**	**<42,30>**	**<38,30>**	**<34,30>**	**<32,30>**	**<30,30>**
sq_prod	**prod4*sumsq_x**	**u<232,116>**	**u<68,55>**	**u<63,55>**	**u<59,55>**	**u<57,55>**	**u<55,55>**

Appendix B

Block diagram of the biometric system

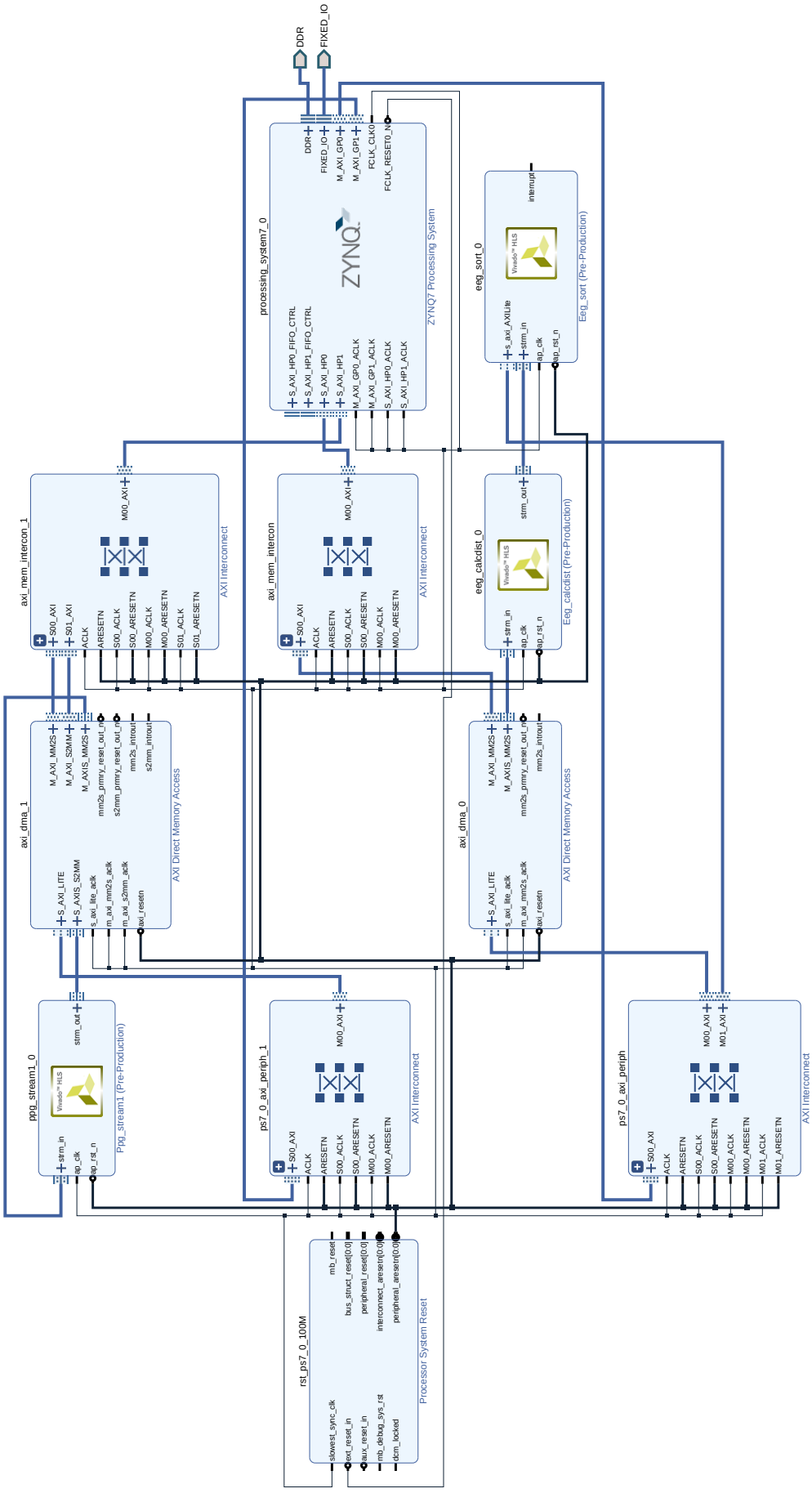


Figure B.1: Block diagram of the complete biometric project.