Identification of cellular handsets through radio frequency signature extraction on an FPGA platform

JP Hattingh

20568401

Dissertation submitted in partial fulfilment of the requirements for the degree *Magister* in Electrical and Electronic Engineering at the Potchefstroom Campus of the North-West University

Supervisor: Prof JEW Holm

May 2015

Abstract

Identification of cellular handsets through radio frequency signature extraction on an FPGA platform

by

J.P. Hattingh

Promoter: Prof. J.E.W. Holm

University: North West University

Faculty: Faculty of Engineering

Department: School for Electric, Electronic and Computer Engineering

Degree: Master of Engineering (Electronic Engineering)

Specific emitter identification refers to the process of performing identification of radio frequency transmitters by exploiting unique variations in emitted signals, caused by hardware variations. In previous research, specific emitter identification was successfully performed on GSM handsets. However, no research has been done on the implementation of specific emitter identification of GSM handsets on an FPGA platform. This study focuses on feature extraction and identification algorithms, as well as the implementation of the identification algorithm on an FPGA.

During this study, phase modulation error was used, as previous research indicated that phase modulation error is an effective feature set for identification purposes.

As the implementation of a classification algorithm on an FPGA was required, a trade-off between complexity and feasibility needed to be made during the selection process. The artificial neural network was selected as the optimal classifier for implementation on an FPGA. The algorithm was first implemented in software and used as the basis for the design on an FPGA. A piece-wise linear approximation of a sigmoid function was used to approximate the activation function, where a look-up table was used to store the parameters.

The off-line training of the artificial neural network was performed in software using the back-propagation gradient descent algorithm.

Good results for the identification of GSM handsets on an FPGA were obtained, with a true acceptance ratio of 97.0 %. This result is similar to the performance obtained in previous research performed in software. In this study, it was found that specific emitter identification of GSM handsets can be performed on an FPGA. Real-world

applications for this technology include general cellular handset identification and access control.

Keywords - Specific emitter identification, handset identification, GSM identification, FPGA.

Opsomming

Identifikasie van sellulêre toestelle deur radiofrekwensie handtekeningontrekking op 'n FPGA platform

deur

J.P. Hattingh

Promotor: Prof. J.E.W. Holm

Universiteit: Noord-Wes Universiteit

Fakulteit: Fakulteit vir Ingenieurswese

Department: Skool vir Elektriese, Elektroniese en Rekenaar-ingenieurswese

Graad: Magister in Ingenieurswese (Elektroniese Ingenieurswese)

Spesifieke uitsender identifikasie verwys na die proses om radiofrekwensie versenders te identifiseer deur unieke variasie in die transmissie (wat veroorsaak word deur hardeware variasies) te gebruik. Daar is in vorige navorsing bewys dat spesifieke uitsender identifikasie op GSM toestelle suksesvol uitgevoer kan word. Geen navorsing oor die implementering daarvan op 'n FPGA is gedoen nie. Die fokus van hierdie studie is geplaas op kenmerkontrekking- en identifikasie-algoritmes, asook die implementering van 'n identifikasie-algoritme op 'n FPGA.

Die fase-modulasiefoute op die transmisie is gebruik as kenmerkstel, waar 'n kunsmatige neurale netwerk gebruik is as identifisering-algoritme.

Die implementering van 'n identifikasie-algoritme op 'n FPGA was nodig, en daarom moes die kompleksiteit en lewensvatbaarheid van die algoritmes opgeweeg word. 'n Kunsmatige neurale netwerk is gekies as die optimale algoritme om geïmplemeteer te word op 'n FPGA. Eerstens is die algoritme geïmplementeer op 'n PC, waarna dit gebruik is as basis vir die implemetering daarvan op 'n FPGA. 'n Stuksgewyse lineêre benaderde funksie is geskep vir die implementering van die aktiveringsfunksie, waar die parameters gestoor is in 'n opsoektabel.

Die "back-propagation gradient descent" algoritme is geïmplementeer in sagteware om die aflyn-afrigting van die neurale netwerk uit te voer.

Die resultate van die identifikasie van GSM toestelle op 'n FPGA was goed, met 'n ware aanvaringsverhouding van 97.0 %. Die resultate is soortgelyk aan die resultate van vorige navorsing wat in sagteware uitgevoer is. Daar is bevestig dat spesifieke uitsender identifikasie van GSM toestelle wel geïmplementeer kan word

op ’n FPGA. Regte-wêreldtoepassings vir hierdie tegnologie sluit algemene sellulêre toestel-identifikasie en toegangbeheer in.

Sleutelterme - Spesifieke uitsender identifikasie, handstel identifikasie, GSM identifikasie, FPGA.

Acknowledgements

Soli Deo gloria

I would like to acknowledge the following organizations and people for their ongoing support:

Council of Science and Industrial Research (CSIR): For the continued financial support, as well as the study time made available to me throughout the years.

Prof. J.E.W. Holm: Thank you for your continued guidance and support all the way through to the completion of this thesis.

Dresden University of Technology, especially J. Hasse, T. Gloe and M. Beck: For your willingness to make your hard work available for use in this study.

My parents, Danél and Anke: Thank you for your encouragement, motivation and love. I would not have been able to complete my studies without your continued support.

Tinus Willemse and Barbara Bradley: Thank you for taking the time to proofread this thesis.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

ASIC	Application-specific Integrated Circuit
BSS	Base Station Subsystem
CLB	Configurable Logic Block
DA	Distributed Arithmetic
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
EV	Error Vector
EVM	Error Vector Magnitude
FFT	Fast Fourier Transform
FPGA	Field-programmable Gate Array
GMSK	Guassian Filtered Minimum Shift Keying
GSM	Global System for Mobile Communications
HDL	Hardware Description Language
ILA	Integrated Logic Analyser
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
ITD	Intrinsic Time-scale Decomposition
JTAG	Joint Test Action Group
k-NN	k-Nearest Neighbours
LNS	Logarithmic Number System

LUT	Look-up Table
ME	Magnitude Error
MS	Mobile Station
MSC	Mobile Service Switching Centre
MSK	Minimum Shift Keying
PC	Personal Computer
PCB	Printed Circuit Board
PE	Phase Error
PWL	Piece-wise Linear
QDR	Quad Data Rate
RF	Radio Frequency
SEI	Specific Emitter Identification
SIM	Subscriber Identity Module
SNN	Spiked Neural Network
SRAM	Static Random-access Memory
STFT	Short-time Fourier Transform
SVM	Support Vector Machines
TAR	True Acceptance Ratio
TDMA	Time Division Multiple Access
TFR	Time-frequency Representation
UDP	User Datagram Protocol
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VIO	Virtual Input/Output
LSE	Least-squares Error

1. Introduction

In this chapter, identification of the research problem is discussed and a problem statement of the study is presented. An overview of the system under development is provided, followed by the objectives of the research conducted. The research process that was followed is also discussed, as well as the scope of the project and an overview of contributions made through this study. Finally, a summary of the remainder of the document is given.

1.1. Overview

Technology has become an integral part of our lives. The percentage of adults owning a cell phone has increased from 53 % in 2000 to 90 % in 2014 [6], showing a substantial increase in the popularity of cellular phone usage.

Because of an increase in cellular usage over the last decade, an increased possibility exists of cell phone usage during incidences of crime. The identity of a cellular phone can provide law enforcement agencies with vital information. The identity of a cellular phone can also be used for access control if the identification process is reliable.

Historically, the identity of a cell phone has been determined through the use of international mobile station equipment identity (IMEI) or international mobile subscriber identity (IMSI) numbers. IMEI and IMSI provide information on an individual cell phone and subscriber identity module (SIM) card, respectively [7]. However, these methods are not foolproof. As it is easy to change SIM cards in a cellular phone, using IMSI for the identification of cellular phones is unreliable. IMEI contains, among others, a serial number of the device, which is ideal for identification of the device. However, with the correct tools and skills the IMEI can be forged or cloned, preventing any possibility of identification.

Another method of identification is using physical-layer identification, where radio frequency (RF) fingerprinting is performed to determine the identity of each emitter.

A problem was identified during the 1960's where the ability to distinguish unique emitters was required, as well as determine each emitter's identity [8]. A transmitted signal was measured, and any intended or unintended variations were exploited to determine the identity. The identity was, however, not obtained though any information transmitted. The process of performing physical-layer identification of

a transmitter is referred to as specific emitter identification (SEI). SEI refers to the processing of inputs constructed from a cellular device's emissions, to obtain a set of unique characteristics, and exploiting it to determine the identify of the transmitter [3].

In literature SEI has been applied to both military and civil applications [9, 10]. Majority of the military applications for SEI are radar-oriented [11, 12]. A civil application for SEI is access control in wireless networks for improved security [13]. A study was found on access control to cognitive radio networks using physical-layer identification [14]. Interesting applications for this technology include identification of motor vehicles, troubleshooting, fault detection and maintenance [15].

An assumption is made different emitters will have a subtle degree of variation during operation. Even though emitters might transmit exactly the same signal, small variations will be embedded in the emitter waveform, which can be exploited to determine the identity. These embedded variations in emission are due to radio circuitry tolerances [16], manufacturing differences [17], age of the device [8, 18], and differences during non-linear operation [3, 9, 19].

In this study, physical-layer characteristics are used to determine the identity of a global system for mobile communications (GSM) emitter. The objective is to perform identification of specific emitters in a closed set of emitters, with the assumption that each emitter has a unique RF fingerprint within the transmission signal. As physical-layer identification is performed, any changes to the SIM card (IMSI) or modification to the IMEI will not affect classification, and only changes to the physical circuit and components will affect classification.

SEI has previously been performed on various emitters, including GSM handsets, and the results are available in literature. Modulation errors were the most effective feature set for identification of GSM handsets. An accuracy of 96.67 % with a support vector machine (SVM) as the classifier was achieved [1] and up to 100 % with a k-nearest neighbours (k-NN) algorithm [20]. However, no previous literature could be found on the implementation thereof on a field-programmable gate array (FPGA). The challenge was thus posed to implement a GSM SEI on an FPGA platform.

Previous research [1] was used as platform for this study, where a collection of recorded emissions of GSM handsets was obtained. In this study, it is shown that a functionally capable classifier can be implemented on an FPGA for the purpose of identifying unique GSM handsets.

In this thesis the process of evaluating the problem and solutions is discussed. The complete process that was followed to implement a classifier on an FPGA for SEI purposes is discussed.

1.2. Background

SEI can be considered a pattern recognition problem where patterns in emitted signals must be recognised to determine the identity of an emitter. As background to this study, a high-level overview of a generic pattern recognition system is given, which will be used throughout this document.

The concept of classification is simplistic, where a set of inputs is used to extract a set of unique characteristics that can be exploited for classification [3]. The basic operation of the system is illustrated in Figure 1.1.

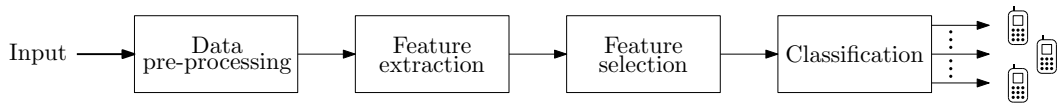


Figure 1.1.: High-level conceptual diagram

As can be seen in Figure 1.1, an input from an emitter is presented to the system resulting in a classification based on processed measurements. The set of unique characteristics (from here onwards referred to as features or a feature set) needs to be obtained from the inputs and this is done through feature extraction. The feature set is then fed through to the classifier for classification. The process of assigning a category or class to the set of inputs received is referred to as classification. In this case each category refers to a specific GSM handset.

The system needs to have prior knowledge to know what inputs will be received, as well as what the proper response to specific inputs should be before any design can be done. The process of obtaining prior knowledge or data is referred to as data collection. During data collection, prior knowledge is collected upon which the design and operation of the system are based. Data used for classification are typically statistical in nature, but can be deterministic as well, although this is not typical. This research focuses on statistical data as the SEI problem dictates.

The objective of classification is to find patterns embedded in data and to assign categories accordingly. Variability in the data is evident in the form of noise or measurement errors, among others. This variability can affect the performance of the classifier adversely. As the classifier should perform classification on underlying patterns, all noisy patterns or inconsistencies that will affect classification should be removed. A great deal of processing can be performed to remove noise or inconsistencies. Kindly refer to subsubsection 2.6.2.3 for more details.

The next step in the design process is to obtain a suitable feature extractor. A feature extractor is used to reduce dimensionality whilst providing unique characteristics that describe each category (or class) effectively for classification purposes. Thus, the objective of the feature extractor is to obtain a set of features, where the variation between feature vectors in the same class is minimised, while the variation between feature vectors of different classes is minimised. This should result

in maximum separability between classes/categories. Feature selection can then be performed to eliminate features that will be less effective.

Based on the feature set, an appropriate classifier is selected. Different classifiers will have different performances depending on the feature set and the effectiveness of a specific classifier when presented with that specific feature set. It is impossible to predict which classifier would be more accurate for a specific feature set, as this is best determined empirically. The features are then used to train the classifier according to the target outputs obtained during the data collection phase. In this research, additionally, hardware implementation was taken into account.

Since data flows through the system sequentially (as illustrated in Figure 1.1), every part of the design process is critical to the eventual classification performance. The eventual results are directly affected by all the functional units of system. As the system consists of sequentially arranged functional units, any unwanted variation will result in a variation throughout the system.

The systematic approach to the design of the system is discussed in chapter 3.

1.3. Research Problem Statement

The need that was identified is for an FPGA-based SEI for GSM handsets. The challenge that had to be addressed was to determine whether a functioning SEI system for unique GSM handsets could be implemented on an FPGA.

The research challenge is to investigate the feasibility of performing GSM SEI on an FPGA platform.

1.3.1. Research Objective

The investigated research objective of this study is as follows:

Show that an FPGA implementation of a functionally capable SEI system for GSM handsets is possible.

Therefore, synthesis and evaluation of a physical layer identification system for GSM handsets must be done on an FPGA platform. The identification system must compare favourably with existing solutions.

1.3.2. Secondary Objectives

To achieve the primary objective, a set of secondary objectives was derived:

- Literature study: Research and investigation of existing feature extraction and classification algorithms;
- Analysis of possible solutions for SEI;
- Determination of the most feasible solutions to SEI for GSM, considering the future implementation on an FPGA;
- Theoretical investigation of SEI systems;
- Design and synthesis of a software-based SEI system;
- Development of operational software for SEI of GSM handsets;
- Simulation of a hardware-based SEI system;
- Design and synthesis of a hardware-based SEI system;
- Assessment and evaluation of the synthesised system.

1.4. Research Methodology

The research methodology that was followed in this research is outlined below:

1. Literature survey:
   a) GSM systems;
   b) FPGA technology;
   c) Pattern recognition systems;
2. Literature study:
   a) Obtain available literature on SEI-related topics;
   b) Study (analyse) existing literature relevant to this research;
   c) Determine possible solutions to this research problem;
   d) Do a feasibility study of SEI, feature extraction, classification, and hardware-related (FPGA) algorithms and techniques for GSM systems;
   e) Do a feasibility study of algorithms with regard to implementation on an FPGA;
   f) Select appropriate features and classifier (specifically applicable to implementation on FPGA);

3. Feature selection and classifier design (depicted in Figure 1.2):
    a) Collect (obtain) data (validated data as obtained from [1]);
    b) Perform feature selection from validated data;
    c) Process obtained data (alignment, normalisation, averaging, frequency correction);
    d) Design and implement of classifier in software (MATLAB), including off-line training;
4. System synthesis:
    a) Implement individual SEI system stages (functional elements) in software to be used as reference against which the FPGA implementation can be verified. That is, the functional elements of the FPGA system are effectively simulated in software, including adjustments that must be made such as function approximations and number representation used on FPGA platforms.
    b) Implement the GSM SEI system in hardware, including a classifier optimised for utilisation on an FPGA platform;
5. Verify all FPGA hardware functional elements to show functional capability of these hardware elements when compared to the software implementation;
6. Validate the complete (integrated) GSM SEI system by first showing functional capability and then comparing performance results from this research with results from a validated reference [1].

The design of a pattern recognition system can be divided into stages. A generic design process of a pattern recognition system is visually illustrated in the block diagram in Figure 1.2.

1.5. Scope of Research

Explicit assumptions were made in this study. This study was based on previously performed research and collected data were obtained from a validated source [1]. This data formed the basis of this research, and since the data were obtained from a reputable source, it was considered valid for this research.

The scope of this study is to make use of the data obtained [1] to design an operational SEI system for GSM handsets in software, and to utilize this data to design a functionally capable SEI system for implementation on an FPGA. As an existing dataset was used for this study, and no real-time requirements existed, an assumption was made that no time limitations on the calculations existed. As this is a feasibility study, the latter assumption does not impact performance in this context.

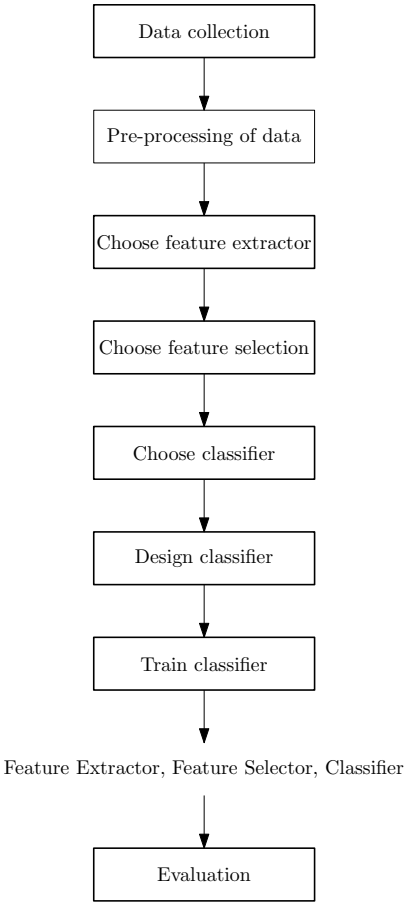


Figure 1.2.: Classification design process (adapted from [2])

Finally, the purpose of this study is not to develop an ANN with regard to classification accuracy, latency or use of resources, but rather to determine whether a classifier can be implemented on an FPGA for SEI purposes.

1.6. Contribution to Research

No solution for SEI for GSM handsets on an FPGA had been found in the literature study. The main contribution of this study is the confirmation that SEI on an FPGA can be performed accurately for GSM handsets. Also, owing to the nature of very high speed integrated circuit hardware description language (VHDL) and FPGA technology, the solution can be used in a scaled-up system with existing functionality.

Together with this, other contributions of this study are as follows:

- Identification of a real-world problem, and using it to obtain a research problem;
- A literature study on the following topics:

  - Feature extraction;
  - Classifiers;
  - Resource reduction.
- A functioning SEI algorithm for GSM handsets in software using an ANN;
- A functioning SEI implementation on an FPGA for identification of emitting GSM handsets on an FPGA;
- Confirmation of another operational method of performing classification for GSM handsets;
- Performance evaluation of software implementation;
- Performance evaluation of hardware implementation;
- Critical analysis of the GSM SEI system based on measured results;
- Documentation.

1.7. Dissertation Outline

The remainder of this report is structured as outlined below.

First, a literature and technology study is presented in chapter 2. In the literature study, feature extraction, classification methods and resource reduction techniques are discussed. An overview of the underlying theory is provided, specifically including data processing, ANN operation and ANN training. An overview on the technology involved in this study is provided, including GSM, FPGAs and the hardware architecture.

In chapter 3, the design and implementation of the complete system are discussed. The software implementation performed in MATLAB is discussed, followed by the VHDL implementation. The debugging of FPGA implementations is critical, and also discussed in this chapter.

The results obtained in this study are presented in chapter 4. A discussion on the composition of the dataset and the effect thereof on the identification results are provided. A comparison between the individual stages of the design is then provided, followed by a discussion. A critical assessment of the results is presented, followed by a comparison of the results obtained to existing research.

In chapter 5 the conclusion to the study is provided, including a brief discussion on the work completed and the results achieved. Future research possibilities are discussed, followed by final concluding remarks.

2. Literature and Technology Study

2.1. Overview

This chapter focuses on feature extraction and classification/identification procedures and algorithms, the efficiency in the application of SEI on GSM handsets and the implementation of an SEI on an FPGA.

One of the major restrictions of implementing algorithms on FPGAs is the number of multipliers required. The implementation of multipliers in combination logic is highly resource-intensive [21, 22]. Multipliers are also resource-intensive on an FPGA, as FPGAs are used to construct a hardware description through the use of combinational logic. Different methods have been presented in literature to reduce the multiplier requirement for implementation on an FPGA as discussed in this chapter.

Following the overview of solutions, in-depth information is provided on the selected feature extraction and classification algorithms.

2.2. Feature Extraction

For SEI, an assumption is made that unique subtle variations embedded in the emitted signal of the unique emitters exist that can be exploited for identification purposes. The role of feature extractors are to extract these unique variations within the signals, for the use of classification.

As GSM handsets are mass-produced, variations on the emissions would probably be minimal. The focus of the feature extraction was placed on subtle variations unintentionally embedded in an emitted signal.

Received waveforms need to undergo processing to obtain features appropriate for identification purposes. The quality of the feature set is extremely important, as the complete pattern recognition system is affected accordingly. Ideally, if it were possible to extract high-quality features, the role of the classifier would become less important. However, in real-world applications, this is rarely the case.

During the design of the system, the choice of feature extraction techniques is based on what unique properties can be extracted from data obtained. Raw data of emitted signals need to be captured and analysed to select the appropriate feature extraction

technique. Analysis is performed on the raw data to establish the differences between emissions of unique emitters. Analyses of different features need to be performed to obtain a set of optimal features for identification. It is impossible to know beforehand what feature extraction algorithms would be optimal for a particular application. The aim of the feature extraction phase is to obtain the best set of features for a specific raw data set and classifier. The best set of features would minimising the distance between feature values of individual emitters, while maximising the distance between samples in feature space for different handsets. The feature extraction method is selected as the method that produces the best set of features. The features are structure in a vector and referred to as a feature vector. The classifier processes this feature vector and a decision is made accordingly [3].

Different pre-processing methods were investigated to obtain the optimal feature set. In Figure 2.1 a taxonomy of pre-processing methods is presented that can be used to prepare data for feature extraction.

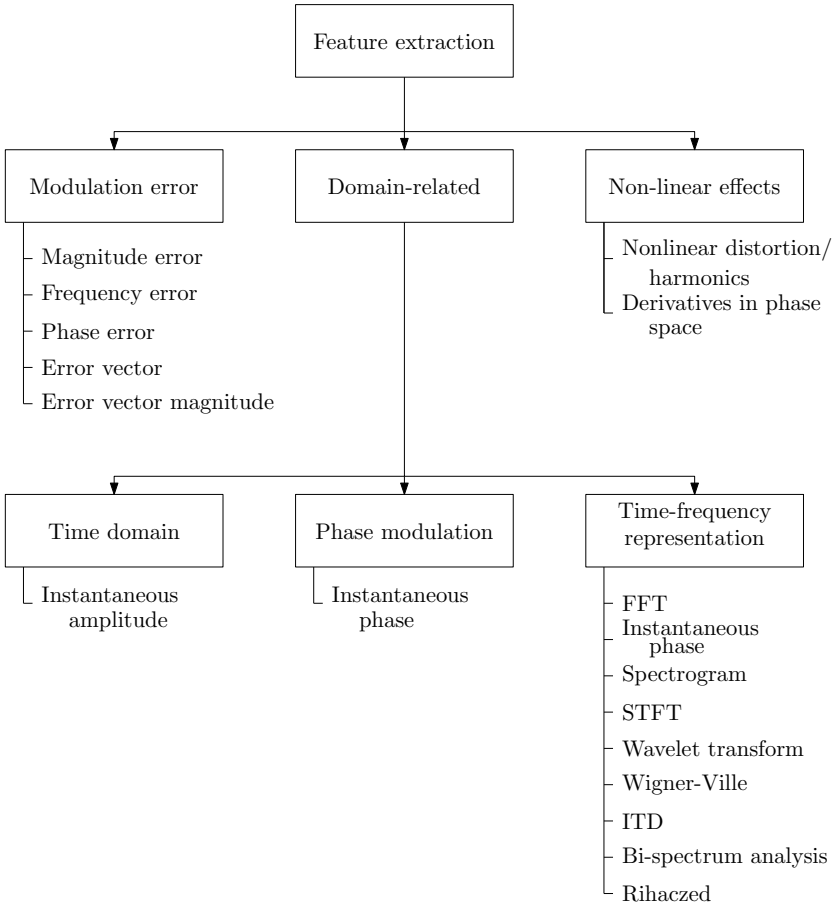


Figure 2.1.: Taxonomy of pre-processing methods as preparation for feature extraction (adapted from [3])

As illustrated in Figure 2.1, the methods that can be used can be divided into three

broad groups, namely domain-related, modulation errors and non-linear effects of the emitter. An overview of the investigated methods is provided below.

2.2.1. Domain-related

In most instances additional processing (termed "pre-processing") is required to aid feature extraction in the reduction of dimensionality of data - this is done to provide a feature set of lower dimensionality. Processing also improves separability of vectors in the feature space by effectively removing information that does not contribute to classification performance. For SEI feature extraction exploits unintentional variations on a waveform [23]. Methods used in different domains have been investigated for feature extraction. Representation using a combination of both time and frequency domains of signals have been widely discussed in literature [14, 15, 24, 25, 26, 27, 28, 29]. The transformed representation of the original signal is then used to prepare for feature extraction, where unique features are more effectively extracted. A discussion on domain transformations is provided below [3].

2.2.1.1. Feature Extraction in Time Domain

In the time domain, one feature extraction method was investigated. This method was found not be sufficiently descriptive, but is included in this thesis for the sake of completeness. This single feature is referred to as the instantaneous amplitude. Instantaneous amplitude is calculated from the in-phase and quadrature components of the signal as follows [3, 30]:

$$a\left(t\right) = \sqrt{i\left(t\right)^2 + q\left(t\right)^2} \tag{2.1}$$

Instantaneous amplitude is also referred to as the power trajectory in some cases. As Gaussian filtered minimum shift keying (GMSK) makes use of a constant envelope, the ideal signal would be a constant value. Any variations from the ideal could be exploited for identification purposes [1, 20].

2.2.1.2. Feature Extraction in Phase Domain

Instantaneous phase of a signal can be used as a pre-processing stage for feature extraction. Instantaneous phase is calculated as follows [3, 20, 30]:

$$\theta\left(t\right) = tan^{-1}\frac{q\left(t\right)}{i\left(t\right)} \tag{2.2}$$

2.2.1.3. Feature Extraction through use of Time-frequency Representations

Because of the expected similarity of waveforms from the GSM emitters, feature extraction from either time or frequency domains independently might not provide suitable features for accurate identification. By using a time-frequency representation (TFR), both time and frequency domains can be considered simultaneously. By using a TFR, a more descriptive representation of a signal can be obtained [10]. This can result in feature extraction of unique variations not present in other representations. A limitation of the standard Fourier transformation is the requirement of the original signal to be periodic. By using a TFR this limitation is overcome [27]. A list of TFRs is discussed below [3].

Spectrogram

The most widely used TFR is the spectrogram. As it is impossible to obtain frequency information with only one time sample, a window is used and the spectrogram is calculated for that window. The spectrogram is then calculated as follows [31]:

$$S_x(t, w) = \left| \int x(\tau)\, h(\tau - t)\, e^{j\omega t} d\tau^2 \right| \tag{2.3}$$

The spectrogram provides a representation of the original signal as a function of time and frequency.

Fast Fourier Transform

The Fourier transform was developed to transform a signal from time to frequency domain, obtaining an alternative representation of the same data. However, the Fourier transform is only valid for continuous waveforms. A similar algorithm was developed for a discrete signal, referred to as the discrete Fourier transform (DFT) to obtain the frequency representation of the discrete or sampled waveform. However, the DFT is considered computationally expensive. Thus, the development of a more efficient method was performed to obtain the frequency representation of a discrete signal, referred to as the fast Fourier transform (FFT). By using the FFT, a very similar result to a DFT is obtained much faster. Thus, the FFT can be used to approximate the transformation of a discrete-time waveform in the frequency domain [32].

The FFT is usually regarded as a TFR algorithm. Since the time signal varies, an FFT is required at incremental time intervals.

Instantaneous Frequency

Instantaneous frequency is defined as the change of phase over time. The instantaneous frequency is mathematically calculated as follows [20, 30]:

$$f(t) = \frac{1}{2\pi}\frac{\triangle\theta}{\triangle t} \tag{2.4}$$

This can be calculated repeatedly at different time intervals to obtain a TFR.

Short-time Fourier Transform

The original Fourier transform is used to transform a signal to the frequency domain. However, the Fourier transform only applies to periodic signals. Through the use of a window the calculation is performed for individual parts of the signal; a local spectral representation at different instances is obtained. This process is defined as the short-time Fourier transform (STFT) [15, 16, 25]. The Fourier transform is applied using a sliding window along a time-domain signal. The sliding window is translated over time to obtain a frequency representation at different sections of the waveform. The STFT of the original waveform is obtained this way [3]. The STFT is calculated as follows [33]:

$$S_t(\omega) = \frac{1}{\sqrt{2\pi}}\int e^{-jwt'} s(t')\, h(t'-t)\, dt' \tag{2.5}$$

where $h(t)$ is the window of the STFT algorithm.

By using the STFT a similar result to the Fourier transform is obtained, while only the samples inside the window being included in the calculation.

Wavelet Transform

The wavelet transform is used to transform a waveform into the wavelet domain. Through the wavelet transform a signal is represented by using a set of basis functions or wavelets. Adjustable basis functions can be used and changed as required [24]. Adjustable basis functions can, however, only be used in cases where appropriate basis functions are known a priori [27]. Wavelet features have been used to obtain effective features for SEI purposes [3, 34]. The continuous wavelet transform is calculated as shown below [35]:

$$W_\psi(s,\tau) = \int_{-\infty}^{\infty} f(x)\, \psi_{s,\tau}(x)\, dx \tag{2.6}$$

where

$f(x)$ is the input signal,

$W_\psi(s,\tau)$ is the wavelet representation of the original signal, and

$\psi_{d,\tau}(x)$ is the wavelet or basis function.

Parameters s and τ affect the shape of the wavelets used to perform the transformation.

Similarities between the STFT and the wavelet transform exists, as sections of the signal are isolated on which the transformation is performed. The isolation is performed through the use of a window, which is translated across a transient signal. The difference between STFT and the wavelet transform is that STFT is used to decompose the signal into different frequencies, while the wavelet transform decomposes a signal into parameters that relate to the correlation of the time signal with basis functions [3].

Wigner-Ville Distribution

An energy-related method to obtain a TFR is the Wigner-Ville distribution by calculating an instantaneous energy density spectrum. An instantaneous energy density spectrum is obtained from both time and frequency domains [28]. The estimate for the instantaneous frequency and is calculated as follows [3, 31, 36]:

$$w(t,\omega) = \frac{1}{2\pi}\int s^*\left(t-\frac{\tau}{2}\right)e^{-j\tau\omega}s\left(t+\frac{\tau}{2}\right)d\tau \tag{2.7}$$

where

$s^*(t)$ is the complex conjugate of $s(t)$.

Intrinsic Time-scale Decomposition

Intrinsic time-scale decomposition (ITD) is a method used to create a decomposed version of the original signal [37]. This is performed by using signals called a base-line signal and a rotation signal. A signal is divided in to higher and lower frequency sections iteratively. The lower frequency section is continuously decomposed to obtain TFR of the original signal. It has been found in literature that ITD can be used to obtain better features than the wavelet transformation and the Wigner-Ville distribution [3, 34].

Rihaczek Distribution

The Rihaczek time-frequency distribution is a TFR that can be calculated as follows [33]:

$$R(t,f) = x(t)\,X(f)\,e^{-j2\pi ft} \tag{2.8}$$

where $x(t)$ and $X(f)$ are the time and frequency representations, respectively.

As can be seen, the Rihaczek distribution is a function of both time and frequency domains. By defining the time instance, a frequency representation at that time instance is obtained. The same principle can be applied to determine a time representation by specifying the frequency [3].

Variations of the Rihaczek time-frequency distribution exist and are discussed in [33].

Bi-spectral Analysis

Through bi-spectral analysis, unintended phase modulation can be examined for features for SEI purposes. Bi-spectral analysis is used to extract phase noise caused by oscillators [29]. Phase noise are always embedded in an emission [38]. The influence of noise is usually less on the phase of a signal compared to the amplitude [38], which implies that a more robust method for feature extraction is provided when using phase. Problems caused by phase noise being present in a signal are distortion, an offset in frequency (frequency is defined as the change of phase over time) and gradually shift in time and frequency domains. These distortions on the time and frequency representations of a signal might result in reduced efficiency of features. However, by exploiting the phase noise for feature extraction, the effect of deterioration of the time and frequency representations can be eliminated [3, 29].

An indication of the phase noise can be obtained by using bi-spectrum analysis [29]. It has been shown in literature that emitters can effectively be identified by performing bi-spectral analysis as pre-processing before extracting features [29].

2.2.2. Non-linear Effects of Emitters

As mentioned previously, one of the feature extraction methods considered was the use of non-linear effects being transmitted instead of focusing on the waveform alone [39]. By using non-linear effects for feature extraction, the effects of the non-linear operation of the transmitter are considered while the actual waveform used to transmit information is discarded. By using the phase domain of the signal, the non-linear effect of the classifier can be obtained.

When an emitter operates in its non-linear region, non-linear distortion called harmonics are created. It was shown that these harmonics can successfully be used as features for identification of emitters. If an emitter transmits at maximum power, saturation of the emitter occurs, resulting in non-linear effects (harmonics) being transmitted [9]. Harmonics can be represented mathematically using a complex power series expansion model as follows [3, 9, 17]:

$$G\left(z\left(t\right)\right) \approx \sum_{i=1}^{\frac{M-1}{2}} a_{2i-1} z\left(t\right)^{2i-1} \tag{2.9}$$

where

a_n is the coefficient of each harmonic, and

$z\left(t\right)^n$ is the amplitude of the n-th harmonic signal.

In literature the feature set consisted of the coefficients of the harmonics, on which the identification was performed. Variations in amplitude of the power of emitters resulted in a translation in feature space. This effect was used to successfully perform identification on signals where a inconsistent amplitude was used [9, 19]. A method was proposed [17] to reduce the effect of varying amplitude, where the distortions are normalised to the amplitude of the signal.

2.2.3. Modulation Errors

An interesting feature set can be created by using modulation errors on the signal [1, 20]. Modulation errors can, of course, only be a solution in the case where a signal is modulated. The modulation error is established by demodulating a signal received, and re-modulating it to obtain an ideal version of the signal. Modulation errors are then obtained by comparing the original received signal with the ideal signal. The modulation error metrics used in existing research are magnitude error (ME), phase error (PE), error vector (EV), error vector magnitude (EVM) and frequency error. An in-depth discussion on modulation errors is found in subsection 2.6.2.

2.2.4. Existing Solutions for Cellular Phone Specific Emitter Identification

The methods that were used in GSM SEI include instantaneous amplitude (or power trajectory) and modulations errors [1, 20]. The most effective features according to the previous research were found to be modulation errors, especially PE.

2.2.5. Conclusion on Feature Extraction

Feature extraction methods were discussed in section 2.2. The list of candidate methods is not exhaustive, but provides an overview of existing methods that could have been used with varying success. The different techniques can be divided into three categories, i.e. domain-related, modulation-related and non-linear effects, and possibly combinations of methods.

It is difficult to determine a priori what feature extraction method will be more effective compared to other methods. The best method is determined through an empirical study. It has been shown by [1] that by using modulation errors an effective SEI classifier for GSM handsets is available.

From literature, it is clear that more than one feature extraction method and feature set may be used, of which the modulation error method was found to be effective and verified and validated for use in this research.

A factor that needs to be taken into account is dimensionality of the feature space. As the "curse of dimensionality" suggests, a large feature set needs to be avoided, as it affects the classification performance negatively, and if possible, the size of the feature set should be limited.

2.3. Classification and Identification

Classification is the process of making a decision based on prior knowledge or training. In the case of SEI for GSM handsets, the decision to associate a representative feature vector with a particular handset is referred to as identification [3].

The criterion for a classifier to be effective is to maximise the classification performance while maximising performance on samples not yet seen. By achieving this, the classification error is minimised, while the generalisation of the classifier is maximised. These two criteria are contradictory and a trade-off between the two criteria needs to be done to achieve optimal performance. By maximising classification performance the classification error should be minimised. A classifier cannot determine beforehand what samples will be presented as inputs. Thus the requirement exist that the classifier should be able to perform on samples not yet seen. The ability to perform classification on not yet seen samples is referred to as generalisation [3].

Classifiers can be divided into two broad categories, namely parametric and non-parametric approaches [2]. As can be seen from Figure 2.2, maximum likelihood estimation and Bayesian estimation are considered parametric classification methods [2]. The challenge associated with parametric classification is obtaining class conditional probabilities. Also, for implementation of a parametric classifier on an FPGA, a set of different functions needs to be stored. Assumptions must be made regarding the statistical distribution of features, and classification is performed according to assumed distributions [40]. Because of the complexity of the patterns

that must be classified during SEI, most publications refer to using non-parametric classifiers [3].

Identification is similar to classification, but adds information relating to the identity of a specific input vector, as opposed to simply associating a class with an input vector. In the case of this study, the identification of a GSM handset implies the ability to distinguish between unique individual handsets, regardless of the manufacturer, make and model. Classification consists of associating the phone with a specific make and model.

An investigation into different identification methods was performed. As the identifier had to be implemented on an FPGA, the characteristics and restrictions of the FPGA platform were taken into consideration. Specific classification methods are computationally complex and may not be appropriate for FPGA implementation. Therefore, the feasibility of implementation of the algorithms on an FPGA was also considered. A significant amount of research is available on classifiers, as well as research on the implementation of the classifiers on an FPGA. Research [40] was also found that discusses and compares the implementation of k-NN and kernel-based SVM methods for GSM SEI purposes, but not the implementation in VHDL or on an FPGA.

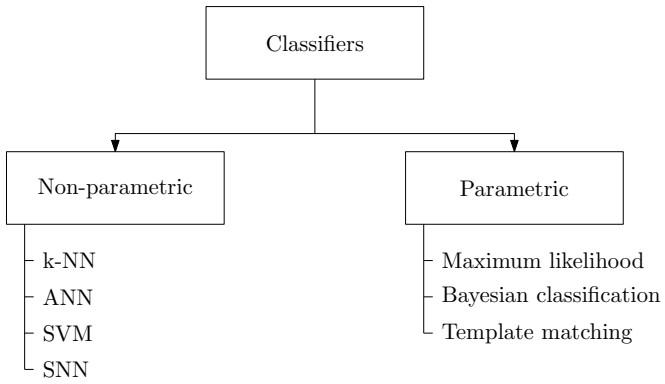


Figure 2.2.: Taxonomy of classification methods (adapted from [3])

A taxonomy, derived from the literature study, for classification and identification of SEI is shown in Figure 2.2 and is discussed below [3].

2.3.1. K-Nearest Neighbour

A popular classification method is the k-NN method. A k-NN classifier make use of the distance between the inputs samples and other samples in the training set to determine the associated class. The input is then classified according to the most frequently represented class in the neighbourhood. The k-NN method has been applied to SEI problems with satisfying results [3, 10, 20].

In [40] the k-NN algorithm resulted in the best performance for GSM classification compared to other existing methods. However, the classification was performed between different manufacturers and models of cellular handsets, and performance may decrease if the overlap of features increase when classification between handsets of the same make and model is required. Research on this topic [20] illustrated that k-NN can be used to perform very accurate identification for unique GSM handsets.

Since a metric is required to calculate the distance between the feature set received and the training features, the k-NN method is computationally complex and resource-intensive [41]. To reduce the computational complexity, a simplified distance metric may be used [42]. However, when a simplified distance metric is used, the performance of the classifier might decrease, as the distance metric might be less representative.

Another disadvantage of the use of k-NN on a hardware platform is that a portion of or the complete dataset needs to be stored in memory [42]. Depending on the number of features and examples that will be used, the k-NN might be a memory-intensive method as all samples need to be stored in memory [40].

With regard to the implementation of the k-NN method on the FPGA platform, most of research investigated 1-NN because of the simplicity of implementation and logic requirements [41, 42, 43]. Because it is highly likely for features to overlap, making use of the k-NN classification becomes less attractive.

2.3.2. Artificial Neural Network

A widely used pattern recognition algorithm is ANN, which was designed to mimic the human brain [44]. An input is fed into a structure of interconnected neurons and is propagated through the network of weights and neurons to calculate a result. An advantage of ANNs is the non-linear characteristics of the classifier (in the case of multi-layer networks), which in general results in an accurate classifier in complex cases [3].

The traditional ANN is considered as the second generation of ANNs [45]. Whereas the first generation ANN was entirely digital (binary), the second generation made use of a continuous activation function that enabled the ANN to perform classification for analogue problems. The activation function can be either linear or non-linear. A non-linear activation function increases the ability to perform complex classification. However, both first and second generation ANNs made use of static features in a feed-forward configuration without taking temporal information into account [3, 45].

ANNs have two steps of operation, namely learning and recalling [2]. The process of learning entails the systematic adjustment of the weight to minimise the classification error [46]. The recalling step is the process of using the weights within the network of neurons to perform classification through propagation of inputs [3].

A standard neuron consists of a set of inputs, weights (synapses), an activation function, bias and output. The output of a neuron is mathematically represented as follows [47]:

$$y_i = f\left(\sum \omega_{ij} x_i + b_j\right) \tag{2.10}$$

where

x_i is the the i-th index of the input vector x,

y_i is the i-th index of the output vector,

ω_{ij} is the corresponding weight value,

b_j is the bias value, and

$f(\cdot)$ is the activation function.

The activation function $f(\cdot)$ gives the relationship between the input and output, while also restricting the output values to predefined ranges, i.e. $[0, 1]$ or $[-1, 0]$. The output of the activation function can be calculated by two widely used methods, i.e. a computational method or a look-up table (LUT) approach [3, 48, 49].

The use of a LUT results in a good trade-off between resource requirements, speed and accuracy when considering FPGA implementation. While the accuracy might be slightly decreased, depending on the resolution of the LUT [50], the resource requirements will be reduced. Another option for an activation function is the use of a piece-wise linear (PWL) approximation [51]. The use of linear as well as non-linear activation functions has been discussed in literature [48, 49]. When an LUT is used to satisfy high-precision requirements, the LUT may become too large and may be not feasible for implementation [52]. Research has previously been performed on the effect of making use of LUT PWL activation functions, and the error obtained by performing the approximation [53, 54, 55]. A variety of estimation errors was found, ranging from 0.008 to 0.04, depending on the number of segments used. By increasing the number of segments, the error decreases, as a more accurate estimation of the activation function is obtained.

An ANN is effective non-linear classifier by virtue of its ability to construct complex decision boundaries in a feature space. ANNs can detect underlying patterns in data that other classification methods might not be able to detect [10]. Because of the complex feature space usually encountered during an SEI problem, ANNs are regularly used in literature [15, 16, 18, 56, 57].

As FPGAs are highly parallel and excellent for modular systems, an ANN is a good solution for this application although other restrictions and limitations do exist [58, 59]. A significant restriction is the amount of combinational logic required to perform a multiplication operation [60, 61]. Because an ANN is constructed by using of a set of neurons, careful consideration needs to be taken during the design and implementation of the individual neuron [48].

An in-depth discussion on ANNs is presented in subsection 2.6.3.

2.3.3. Spiking Neural Networks

When biological neural network systems are considered, information is not transmitted through static analogue values, but rather encoded through the use of temporal information (or temporal coding) [62].

The third generation of ANN takes a step closer to the biological neural network where spatial-temporal information is used [45]. The structure of the spiked neural network (SNN), also referred to as pulsed neural network, is similar to the traditional ANN. The SNN also consists of levels of parallel neurons connected with synapses, but makes use of spikes instead of static analogue values to transmit information [61]. The characteristics of the signals are neglected and only the timing characteristics of the spikes are considered [45].

A model called the hybrid Hopfield neural network is the most accurate model based on the biological neural network, also incorporating temporal information [45, 58]. Because the complexity of the hybrid Hopfield neural network, it is not feasible to implement on an FPGA.

Two primary neuron method approaches have been used for SNN, i.e. spiking response and integrate-and-fire methods [45, 60, 63, 64]. Both methods incorporate the same principle, referred to as the threshold-fire model. The threshold-fire model refers to a model where a level of potential is accumulated, and when a specific threshold is reached, a spike is transmitted at the output (also called action potential). The integrate-and-fire method is the most widely used spiking neuron method [45].

With the integrate-and-fire method, each neuron has a membrane potential that indicates the state of the neuron and gives an indication of the number of spikes received from the incoming synapses. With each incoming spike the membrane potential is increased until the threshold value is crossed, which results in a spike being emitted [45]. After a spike has been emitted, the neuron enters a refractory state for a predetermined amount of time. While in the refractory state, the incoming spikes will have no effect on the spike or membrane potential until the resting potential is reached [45, 58]. Once the membrane potential reaches the resting potential, the operation of the neuron returns to the initial state.

Both the spiking response and integrate-and-fire models are considered computationally expensive [59, 65]. However, a simplified and hardware-friendly integrate-and-fire method has also been developed that reduced slice and multiplication requirements [59, 61].

According to [59] no effective training method for SNN has yet been developed. The most investigated learning method to date is the spike timing dependent plasticity algorithm [63, 59, 61], also referred to as Hebbian learning. Hebbian learning is an unsupervised learning method [45] that uses time differences between spikes and adjusts the weight values accordingly. The complexity of implementing an SNN,

together with the lack of a supervised training method, indicate that a classic non-linear ANN is more desirable for implementation in a case where a workable solution must be implemented on an FPGA. Therefore, this research shall make use of ANNs as opposed to SNNs.

2.3.4. Support Vector Machine

An SVM is a classification method that uses only a limited number of samples in the training set. The samples closest to the separation between two classes are called support vectors [2], and a hyperplane is placed optimally to separate the support vectors.

SVMs make use of a mapping function to increase dimensionality. An assumption is made that by increasing the dimensionality of the feature space, as simpler classifier can be used.

Because the mapping function increases the dimensionality of the problem, a higher resource requirement is created. To prevent an increase of resources, a kernel function is used to prevent the increase of dimensionality while still providing information that would have been provided through a higher dimensionality [66, 67, 68]. A kernel function makes use of dot products between different features to create new features that increase accurate classification [3, 68].

The most widely used kernel functions in literature are linear [69], degree-d polynomials [69, 70], Gaussian, radial basis functions [66, 69], sigmoid functions [69] and other variations [3, 68, 71]. All of these methods require multipliers and a reduction in multipliers will be required. General multiplication reduction techniques have been used to decrease the resource requirements of an SVM implementation on an FPGA, such as the use of logarithmic number system (LNS) and multiplication through shift-and-add operations. More information on multiplier reduction methods can be found in section 2.5.

Inherently, the SVM approach is applicable to two class problems, but can be extended to multi-class problems. Multiple solutions to the multi-class problems exist. This can be overcome by one-against-all, one-against-one, or through the use of directed acyclic graphs [72, 73]. When implementations of the different multi-class approaches on an FPGA were compared, the one-against-one method had the highest classification performance but required most binary classifiers. The one-against-all method had a slightly lower performance but required fewer binary classifiers, while the directed acyclic graphs method was a trade-off between the other methods [3, 72].

An ensemble approach was proposed by [74, 75], with improved results compared to only one SVM being used. Unfortunately, the implementation of an ensemble of classifiers is not feasible on an FPGA because of high resource requirements as multiple classifiers would need to be implemented [3].

It has been shown in the literature that the SVM algorithm outperforms the traditional ANN with regard to classification performance in the field of SEI [40] and otherwise [76]. Since a number of classifiers would be required for the case of multi-class classification, the implementation would require a considerable amount of logic. Despite the performance of SVMs for GSM SEI [1], resource requirements limit its applicability to FPGA implementation.

2.3.5. Existing Solutions for Cellular Phone Specific Emitter Identification

Previous research has shown that identification of GSM handsets is possible. The two classifiers that have been used in available research were an SVM [1] and k-NN [20] (through the use of Euclidean distance). The classification accuracy for the existing classification techniques was 96.67 % for the SVM [1] and up to a 100 % for a k-NN [20]. To our knowledge, no application has been adapted for the implementation of identification of GSM on an FPGA.

2.3.6. Conclusion on Classifiers

A comparison between solutions for the classifier of a GSM SEI system was presented in this section. The trade-offs between different classification algorithms were classification accuracy, complexity of the algorithm and feasibility of implementation of the algorithm on an FPGA platform.

The performance of different classification methods is very difficult to determine up front. According to the "no-free-lunch theorem", it is impossible to determine beforehand which classifier would outperform another [77]. Selection of the most effective classification method for an SEI is not sensible only from literature as the data used by authors differs from the data in this research to a large extent. Thus, classification performance can be determined through an empirical study.

The selection of an appropriate classifier was based on the feasibility of implementation on an FPGA, with more simplistic methods being favoured in this research. Complexity of the algorithm was used as a criterion as an increase in complexity translates to an increase in resource requirement on the FPGA.

It is reasonable to expect that differences between GSM handset features will be small in feature space (due to manufacturing, design, and other quality factors of handset technology that limit differences between emitted signals from similar handsets). Therefore, a classification method is required that supports complex classification boundaries. For this reason, a non-linear method is required in this research.

The different classifiers discussed in section 2.3 were divided into two categories, i.e. non-parametric classifiers (k-NN, ANN, SVM, SNN) and parametric classifiers (maximum likelihood, Bayesian classification, template matching).

In this research, non-parametric classifiers were considered in favour of parametric methods that require relatively complex computations and data manipulation (such as the implementation of distribution functions). Non-parametric methods are generally repetitive in nature, although some of these methods require a significant amount of data manipulation. It was this imperative to find a non-parametric method that is repetitive in nature, simplistic, with limited data manipulation.

The three classifier solutions that were considered are k-NN, ANN and SVM algorithms.

First, the k-NN in general performs well on a wide range of problems. It has been shown that the k-NN can be used for classification of GSM handsets [20]. However, it is not a feasible solution as a large amount of memory would be required, since each example in the training set has to be stored.

Similar to the k-NN, the SVM is a good method for GSM handset SEI, as shown by [1]. The SVM performs a transformation into a different feature space to increase the probability of linear separability. As the SVM is also a non-linear classification, it was an attractive option. Also, the SVM algorithm achieves good generalisation on small datasets as only a limited number of support vectors are selected [78]. However, implementation of the transformations of feature space on an FPGA will result in a complex system with high resource requirements, as transformations have to be implemented for multi-class problems.

An ANN can theoretically perform classification of GSM handsets. The advantage of the architecture of an ANN is that it is highly parallel and repetitive in nature, which results in reuse of hardware resources. In an ANN, sigmoidal transfer functions (operators) and vector multiplications are repeatedly used. This characteristic can be exploited to reduce the resource requirements, without sacrificing classification performance. An advantage of ANN over the SVM with regard to FPGA implementation is the use of only a single function approximation, where different functions need to be implemented for the SVM transformation. However, ANNs tend to over-fit on training sets, resulting in a reduced generalisation [78]. Because of the small dataset obtained, this could result in a reduction in generalisation and classification performance. This issue will be addressed by using adequate training methods.

2.4. Preferred Solution

In conclusion, the chosen solutions were as follows:

- Feature extraction: The use of modulation errors as features;
- Classification: ANN.

The preferred solutions are discussed in more detail in the remainder of this chapter.

2.5. Multiplication Reduction and Avoidance

The implementation of the multiplication operation on an FPGA remains a major problem. Attention should be paid to limiting the use of multipliers as far as feasible as multiplication operators are very area-intensive when implemented in combinational logic [21, 61]. Throughout the literature, extensive research has been done either to reduce the number of multipliers required or even to avoid them entirely, as discussed below.

2.5.1. Parallel/Serial Implementation

Parallel implementation of a system has the advantage of increasing execution speed. It may not be possible to implement an entire GSM SEI system due to limited resources on an FPGA. To fit the GSM SEI on an FPGA, it may be possible to use parallel sections of the SEI process sequentially [48]. The use of sequential building blocks is especially helpful in instances where the same operation is repeatedly performed. The disadvantage of this approach is reduction in speed.

This method can be performed at different levels of the implementation, depending on the algorithm used. In the case of neural networks, the different levels may be multiplier, neuron and layer levels. This approach has been widely used in ANN implementations owing to modular characteristics, but the principle can be used for any algorithm.

2.5.2. Multiplexing Components

A neural network consists of a fixed structure of neurons. The input is propagated through the complete network to the output, where the network consists of a number of layers of neurons in parallel, with the sequential layers connected in series. With this configuration, later layers cannot perform calculations until calculations of the previous layer have been completed. It may be possible to design a single layer performing parallel calculations and sequentially implementing the same layer for

all other layers. Thus, only the resources of one layer are used [21, 49]. However, pipelining is reduced as each layer is considered independently. Different levels of multiplexing can be used according to the requirements. As the ANN is extremely modular, where the same calculation is repeated multiple times, multiplexing of different components is possible. The options of multiplexing within an ANN are multiplexing of layers, neurons and multipliers or any combination of these [21].

2.5.3. Multiplication through Shift-and-add Operations

An ideal approach to the problem would be to avoid the multiplication operation completely. A method to avoid multiplication is to use a shift-and-add multiplication method. When one of the values being multiplied is a power of two or a sum of a power of two, a shift register can be used for multiplication. Through binary shift-and-add operations, addition can be used to add values to obtain the result of the initial multiplication. The calculation of a multiplication can thus be performed through shifting, as multiplication by an integer power of two is equivalent to shifting by that integer power. Through careful selection of the values being multiplied, any multiplication can be performed without the need of a multiplier [48, 71, 79, 80].

2.5.4. Logarithmic Number System

A characteristic of the LNS is the ability to replace the multiplication and division operators with addition and subtraction respectively within the logarithmic domain. As addition and subtraction operators are much less expensive with regard to hardware area and logic, the use of the LNS was investigated in literature to reduce the required number of multipliers.

The advantage of the use of LNS is the trade-off between simplicity of implementation, speed, cost and power. The LNS is considered a good trade-off between fixed-point and floating-point arithmetic [81]. The required logic of the transformation to the logarithmic domain can be considered irrelevant with regard to the amount of logic saved through the avoidance of multiplication [69].

In literature it has been shown that LNS has been used to reduce multiplication during kernel calculations for SVMs [69, 82].

It has been shown in literature that implementation of calculations on an FPGA board using LNS has led to significant improvements with regard to speed and slice requirements, compared to the same implementation using floating-point numbers [52].

2.5.5. Distributed Arithmetic

Another method of performing multiplication reduction during the calculation of the sum of products is referred to as distributed arithmetic (DA). DA expands and rearranges the sum of products in such a manner that a section of the calculation, which contains the majority of multiplication operations, has a limited number of values. These values can then be stored within LUTs, and through the use of these LUTs the multiplication operation can be avoided [83]. However, an LUT can only store a limited number of values. LUTs are dependent on the type of function to be approximated and in complex cases a number of LUTs may be required. The feasibility of this approach is dependent on the number of required LUTs.

2.5.6. Reconfiguration

A situation may arise where there is limited processing power or available logic on the hardware platform, but it may not be feasible to use multiple FPGAs in the system design. A solution to this problem is referred to as reconfiguration, which is the process of using the same section of logic for different applications [84, 85, 86]. Reconfiguration can be divided into two basic categories, namely partial and full reconfiguration. Partial reconfiguration refers to the reconfiguration of specific sections on the FPGA while other sections of the logic are kept intact. Because sections of the logic are kept unchanged, it is possible that the system can operate continuously during the reconfiguration process. The reconfigurable sections can then be replaced by other sections that have different characteristics and operations. In Figure 2.3 a visual representation of partial reconfiguration is shown, where reconfigurable regions can be changed, while other regions are kept constant.

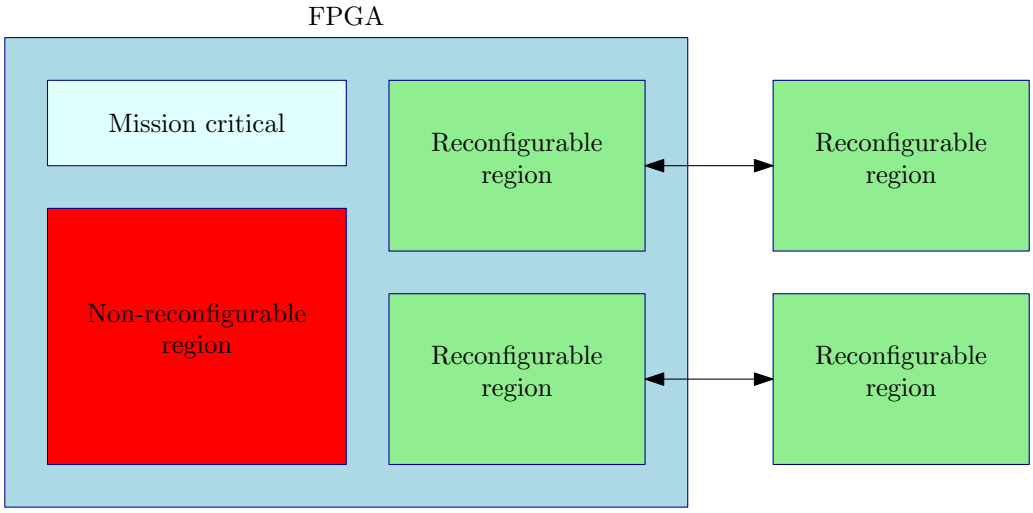


Figure 2.3.: Partial reconfiguration [4]

Full reconfiguration refers to the reconfiguration of the complete FPGA, which requires a complete interruption of the system. Through the use of reconfiguration, the effective logic of an FPGA is increased through hardware re-use, while sacrificing computational time due to the time required for reconfiguration (latency varies between different platforms, methods and technology used for reconfiguration). Another advantage of the use of the reconfigurability is the increase in flexibility, while reducing power requirements [59, 84, 86].

2.6. Theoretical Framework for GSM SEI in this Research

In the literature study above, a broad overview of feature extraction and classification was obtained. It is now necessary to adjust the focus on a theoretical framework specifically applicable to GSM SEI on an FPGA system.

2.6.1. Introduction

First, the data collection, feature extraction and data processing as performed by [1] are discussed (section 2.2), followed by the theory of an ANN and the training of the network (subsection 2.6.3).

2.6.2. Data Collection, Feature Extraction and Data Processing

2.6.2.1. Introduction

A fundamental part of the pattern recognition system design is data collection and feature extraction. The quality and integrity of the database are critical, as the database is used during the design and evaluation of the system. In this section data acquisition and feature extraction on the data are discussed.

2.6.2.2. Overview of Dataset

Prior knowledge is required to prepare the pattern recognition algorithms for their task. In this case, prior knowledge of specific handsets that need to be classified is required.

A collection of emissions was obtained from the Technical University of Dresden [1] and was used in this study. The dataset consisted of a variety of GSM handset transmission data with the intention of performing feature extraction and SEI on the collected data. An extensive set of data and processed information from ten

GSM handsets was obtained from the authors. The data received was used for this research as it had been validated through peer review.

The recording of the signals was performed through the use of a third party receiver (i.e. only a passive receiver, not part of the GSM network). As no transmissions were made, there was no interference on the network, which ensured that the GSM network operated as usual. To prevent interference from unwanted handset signals, both the transmission of the handset and base station were recorded. The recordings were performed through the use of a software defined radio that interpreted both the uplink and downlink of the handset and base station respectively according to the GSM protocol. As the GSM protocol was used, accurate bursts of the corresponding handset were extracted. The receiver was an operational GSM receiver, so received emissions could be verified to confirm the actual emitter from IMEI and IMSI, reducing the probability of errors and increasing the integrity of the data [1].

The process followed by [1] to obtain the variations that were used for feature extraction is illustrated in Figure 2.4, with a description that follows.

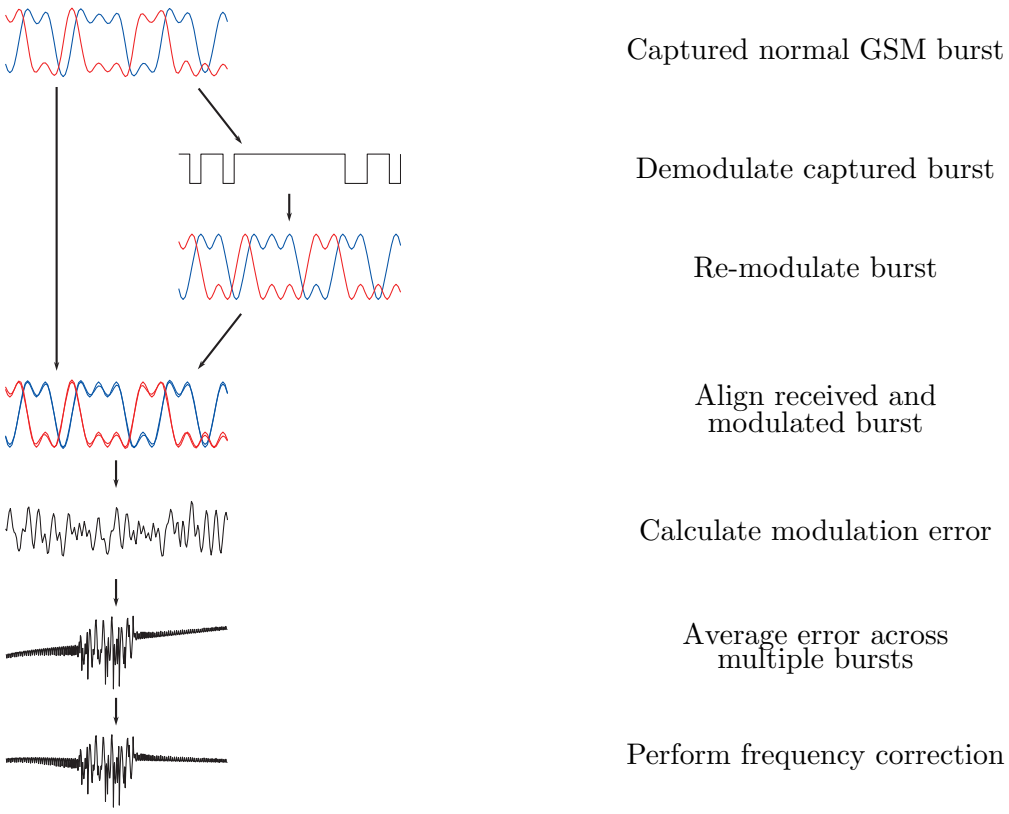


Figure 2.4.: Overview of the feature extraction process [1]

The feature extraction methods considered by [1] were based on the modulation errors, or variations, of emitted signals. The approach followed by the authors is

depicted in Figure 2.4. Firstly, the captured GMSK bursts were demodulated to the digital equivalent. Through the demodulation process the modulation error becomes redundant, owing to the nature of digital communication (noise immunity of digital communication as well as error tolerances of GSM [87, 88]). The demodulated signal was then again modulated to simulate the original signal, where the simulated signal did not contain the modulation errors caused by the hardware. The demodulation and modulation process were completed with a modified OpenBTS system[1]. After alignment and normalisation of the simulated and initially captured signals (subsubsection 2.6.2.3), an error metric (subsubsection 2.6.2.4) was used to obtain a set of modulation errors. The errors were accumulated and the average calculated to obtain an average over multiple bursts (subsubsection 2.6.2.7). As a constant PE for each sample is possible, a constant frequency error would be included in the signal and would also be present in the PE metrics. Since the frequency offset would affect the training and classification performance, it needed to be removed (subsubsection 2.6.2.5) [1].

2.6.2.3. Alignment and Normalisation

An accurate estimation of the error between the simulated and received signal could only be made when corresponding samples in both bursts were compared.

As mentioned in subsection 2.7.3, during operation different types of GSM bursts are used for different reasons. To maintain consistency between different bursts, only normal bursts were used for the creation of the dataset, and therefore also for the remainder of the research.

To compare corresponding samples, the signals need to be aligned with regard to time. Alignment is not only performed between the two individual signals, but also between all bursts to ensure that the errors are aligned over the bursts.

After alignment in time, the signals also need to be aligned according to phase, as well as normalised in magnitude.

The alignment in phase was calculated by the average of the difference between the received and re-modulated signals, and removed accordingly. The normalisation in magnitude was performed by averaging the major peaks within a burst (chosen by using a threshold), and normalising the average to 1 (normalisation of the minimum and maximum peaks were performed to -1 and +1 respectively) [1]. The alignment of the different bursts and error metrics was crucial as discussed in subsubsection 2.6.2.7.

2.6.2.4. Error Metrics

In previous research [1, 20] modulation errors were used as features for the classification process for GSM SEI purposes. The following section discusses the variety of

[1]See http://wush.net/trac/rangepublic/ and http://openbts.org/

different modulation error metrics used.

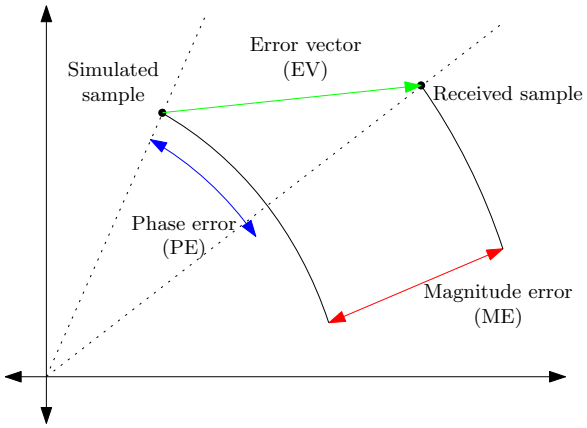


Figure 2.5.: Example of the constellation diagram with received and simulated points, as well as error metrics (adapted from [1]).

Figure 2.5 provides a visual overview of the error metrics, as discussed in the subsections below.

All these error metrics give an indication of the amount of modulation error at each corresponding sample in the burst (i.e. after alignment).

Magnitude Error

After alignment and normalisation of the simulated and received signals, any magnitude variations between the two signals are considered the ME.

Phase Error

The difference between the phase of the simulated signal and the received signal is considered the PE.

Error Vector

As can be seen from Figure 2.5, the EV is a vector in the signal constellation, where the EV is calculated relative to the simulated sample. Thus, a complex value is obtained.

Error Vector Magnitude

The EVM is directly derived from the EV, with the distinction of only using the magnitude of the distance between the two samples, regardless of the angle.

Power Trajectory

Since the GMSK makes use of a constant envelope during transmission, any variations from the expected envelope can be exploited for classification.

2.6.2.5. Frequency Correction

In subsubsection 2.6.2.3, the alignment with regard to phase is discussed. Some phase offset might still remain, that would result in a frequency error. An illustration of the frequency error can be seen in Figure 2.6, where the PEs of two handsets are compared.

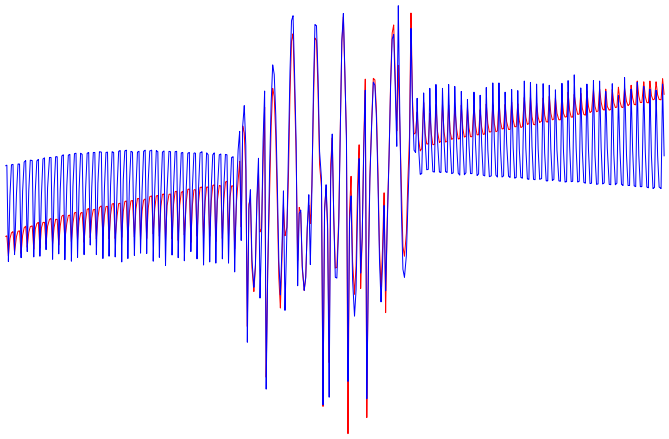
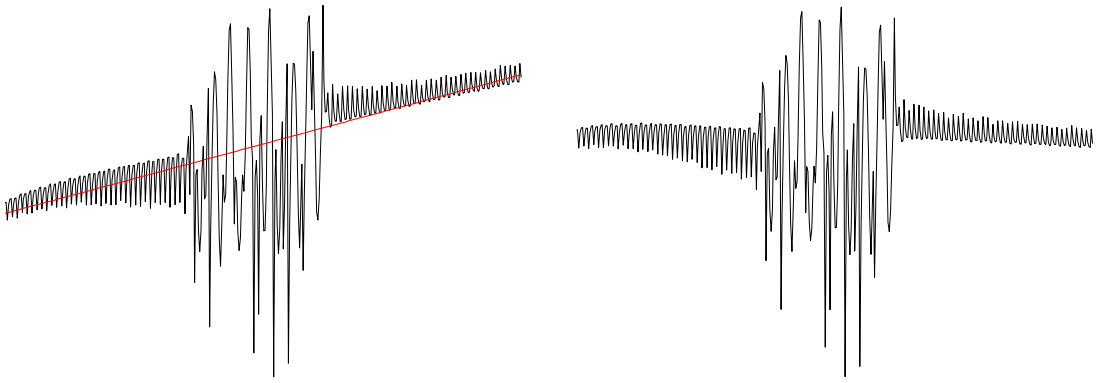


Figure 2.6.: Comparison of constant phase difference between different handsets

As any unwanted component or effects in the signal can affect the classification performance of the classifier, the frequency error needs to be removed. A low-order interpolated function was used to obtain a function that represents the data, to get an approximation of the frequency error (represented by the red line in Figure 2.7a) that was then subtracted. The result is displayed in Figure 2.7b.



(a) Before frequency correction **(b)** After frequency correction

Figure 2.7.: Frequency correction

2.6.2.6. Structure of Dataset

The dataset obtained from [1] consists of recordings of ten unique handsets as listed in Table 2.1.

IMEI	Handset
3522340114588538	Motorola C118 #1
3522340139448767	Motorola C118 #2
3576790048382162	Motorola C118 #3
3583170138857775	Motorola C123 #1
3583170119139375	Motorola C123 #2
3567610006655101	Medion SP 1000
3552150089128130	Motorola C115
3585160261427242	Blackberry 8310
3582560038163709	Motorola C139
3593300285264837	Nokia E51

Table 2.1.: Handsets of the captured data obtained from [1]

The dataset was constructed using R-Project[2], where each *.rdata file was a workspace containing the data of a unique burst. All the bursts found within a folder were captured transmissions of the same handset, while each folder was named according to the handset's IMEI number. Within each workspace, a variable "src" exists. The data are stored in the "src" variable, of which the structure is displayed in Table 2.2.

[2]See http://www.r-project.org/

src	Description
burst_type	Type of burst - 3 represent a normal burst
begin	Expected beginning of burst (sample index)
end	Expected end of burst (sample index)
count	Sample count present in
withsignal	If 1, an ideal simulation is present
fn	GSM's frame number
ts	GSM's time slot
expected_begin	Another expected begining (sample index)
airprobe_corr	Correlation calculated by airprobe
startendphase_len	Length of simulated start and end regions
binary	Demodulated bits
data	Samples - Raw GSM signal
signal	Ideal GMSK simulation of captured bits (binary)
startphase	The start phase of simulation
endphase	The end phase of simulation
cdata	Raw signal as complex numbers (in-phase and quadrature component)
csignal	Simulation of complex numbers (in-phase and quadrature component)
offset	
approved	
feature_cor_raw	Correlation feature vector
feature_timingoffset	Timing offset
feature_constantphaseoffset	Used for signal correction
feature_cor_corrected	Correlation in various regions with corrected signal
features	Feature vectors

Table 2.2.: Structure of the dataset provided by [1]

Within the structure of the dataset, all the extracted features that were obtained were contained in the "features" class. The structure of the "features" class is displayed in Table 2.3

	Features
ME	Magnitude error
PE	Phase error
EVM	Error vector magnitude
EV	Error vector (complex)
si	sample index of feature (index to [data]/[cdata])
ip	
ipc	Count of ip

Table 2.3.: Structure of the feature class within the dataset [1]

2.6.2.7. Averaging of Bursts

As discussed in subsection 2.6.2, the objective of the feature extractor is to obtain a feature set with the least variance between samples within each class while increasing the variance of samples between different classes. It was discovered [1] that individual bursts varied within classes and were not appropriate for features. However, by averaging bursts from the same emitters, consistencies between the samples of the unique emitters were found, while the amount of in-class variance was reduced. It was found that the accumulation and averaging of 30 bursts or more resulted in a reduction in variance between features of the same handset.

The alignment and normalisation of the bursts (and thus also the error metrics of the bursts) with regard to time, phase and amplitude were crucial, as any invariance between error metrics need to be avoided. This alignment and normalisation also resulted in improvement of characteristics present across all bursts, resulting in an improved opportunity to perform classification. As the average between the different error metrics of the bursts was calculated, the corresponding samples needed to be aligned in time and phase and the magnitude normalised, to ensure the average of the bursts was uniformly weighted between the bursts.

For illustration purposes, Figure 2.8 and Figure 2.9 are provided, showing the averaging of two handsets and a closer view of the training sequence, respectively.

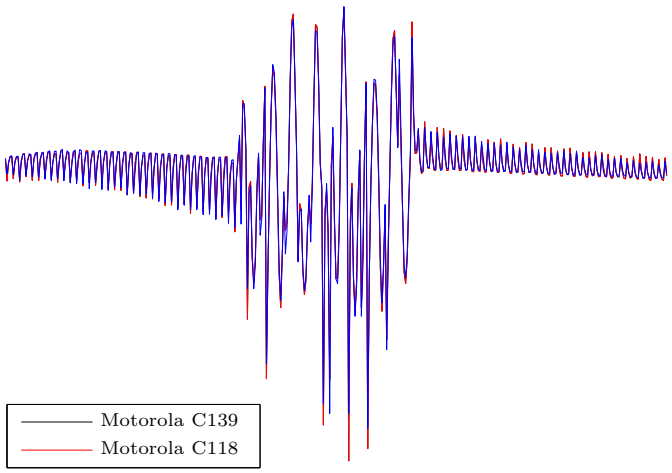


Figure 2.8.: Average of PE for all bursts

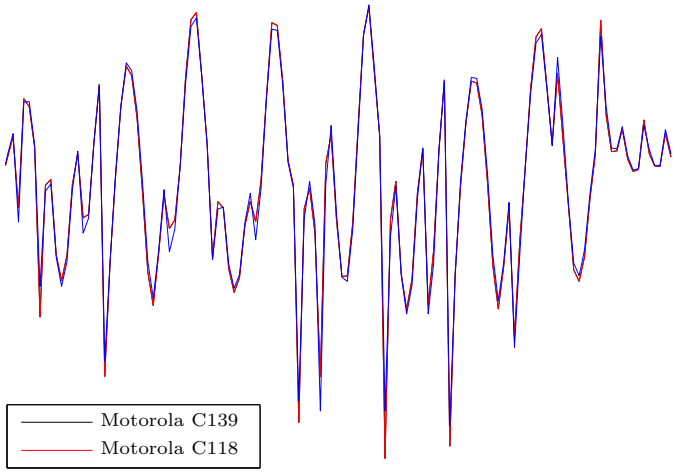


Figure 2.9.: Average of PE for all bursts over training sequence

In Figure 2.10 the average of the PE over the training sequence for three unique Motorola C118 handsets is shown.

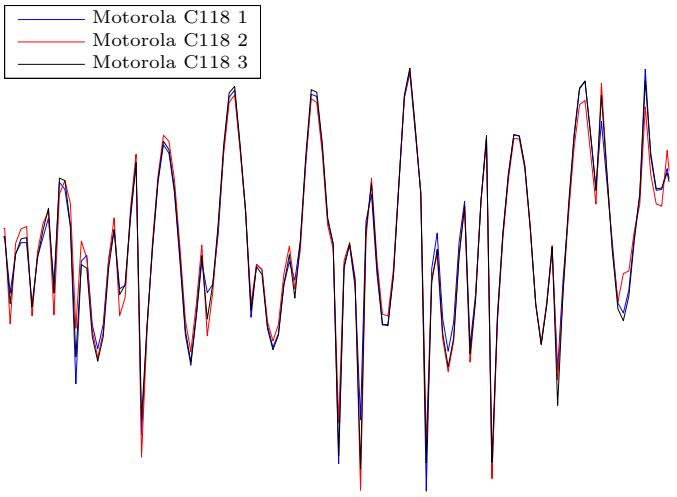


Figure 2.10.: Average of PE for the training sequence of three unique Motorola C118 handsets

2.6.2.8. Division of Dataset

The role of the dataset was twofold. All pattern recognition algorithms need prior knowledge on which a pattern recognition operation is based. In this case, the dataset was used to train the neural network as to how it should perform, and what typical inputs could be expected, together with corresponding outputs. Secondly, it was required to determine the effectiveness of the pattern recognition algorithm when previously unseen patterns were presented. The dataset was divided into two separate datasets, i.e. training- and test set.

To increase generalisation of the classifier, two different sets of data were created for training and evaluation of the classifier respectively. It is not appropriate to use the same data for evaluation of the classifier, since the classifier may become biased/over-fitted to the training data. A classifier is designed to perform classification on unseen data, therefore, the system needs to be tested with previously unseen samples to get an accurate estimation of the classification performance.

2.6.3. Artificial Neural Networks

2.6.3.1. Introduction

Previously, in section 2.3, neural networks were selected for their favourable characteristics for implementation on an FPGA platform. In order to form a deeper understanding of the functioning of a neural network, it is necessary to elaborate on the theory of ANNs. This will underline the decision to use a neural network and provide more information on the implementation of a neural network on an FPGA.

2.6.3.2. Feed Forward (Artificial) Neural Network

ANNs are designed with the intention of loosely mimicking the biological nervous system. As a biological neural network is extremely complex and not yet completely understood, it would be impossible to design a pattern recognition system fully based on it. The ANN has been based on a biological neural network through a set of assumptions that simplify the system and enable the design of a broadly based neural network. The most popular derivative of the neural network is the multilayer perceptron - an overview of its operation is given below. (Please note a bottom-up approach of the layout of the ANN is followed.)

Neurons

Similar to the biological neural network, the ANN consists of a network of neurons (also called perceptrons, nodes or units) that are ordered in a specific structure. The difference in operation between the biological neuron and artificial neuron is in the method of transferring information. The biological neuron's information is transferred through the timing and sequence of pulses through synaptic connections between neurons. To simplify the ANN, the artificial neuron models this behaviour using a magnitude value only. (Please note that from here onwards when a neuron is mentioned it refers to an artificial neuron.)

The neuron can be divided into the following components: inputs, weights, adder (summer), activation function and outputs. An illustration of the structure of a neuron can be seen in Figure 2.11.

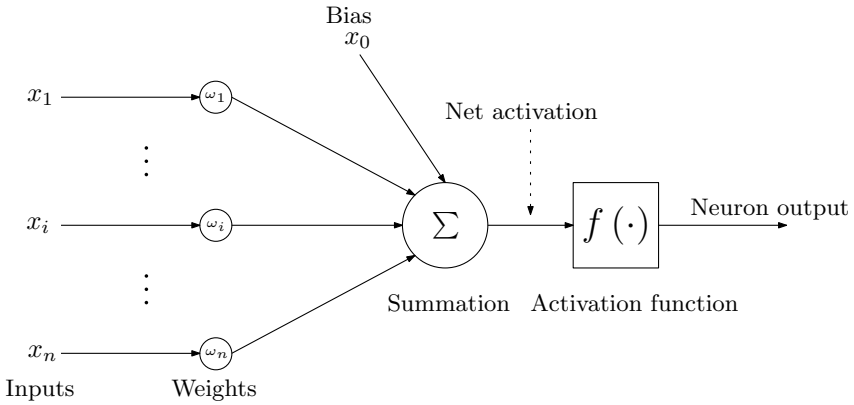


Figure 2.11.: Structure of a standard neuron [2]

As can be seen from Figure 2.11, the standard structure of a neuron consists of multiple inputs (also referred to as the input vector) that are processed, and a single output is provided. Each input also has a corresponding adaptable weight (which is the analogy for synapses in the biological brain), and so the output is a

function of the weighted inputs. The value of weights is used to control the degree of influence each input has on the output. The operation of the complete ANN is dependent on the set of weights of the network, which are calculated during training. The sum of the weighted inputs is calculated and is referred to as the activation net, or simply net. The activation net is mathematically represented as follows [2]:

$$net_j = \sum_i x_i \omega_{ji} \equiv \mathbf{w}_j^t \mathbf{x} \tag{2.11}$$

where

i denotes the index of the input and weight,

j denotes the index of the specific neuron, and

ω_i and x_i denote the weight and input at index i.

The net activation is the input value to the activation function. The activation function introduces non-linearity into the neuron calculation and results in a non-linear relationship between the inputs and the output of the neuron. Different activation functions can be used by different neurons in a network, but in this study the same activation function was used. The use of a single activation function is appropriate in view of the required implementation of the ANN on an FPGA, where the same activation function can be used multiple times, thus increasing the ability to reuse hardware.

A wide variety of activation functions can be used. The most popular activation functions (denoted as $f(\cdot)$) are listed below [2, 89]:

- Threshold (or sign) function:

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{2.12}$$

- Linear function:

$$f(x) = x \tag{2.13}$$

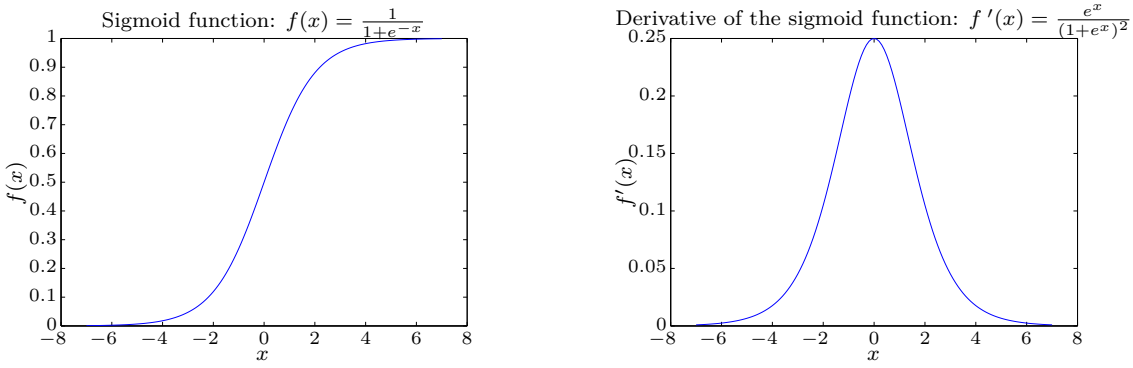
- Sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.14}$$

The sigmoid function was used during this study, so the derivative of the sigmoid function is also required and the mathematical equation is given below.

$$f\,'(x) = \frac{e^x}{(1 + e^x)^2} \tag{2.15}$$

Since the sigmoid function was chosen as activation function, the function as well as its derivative were graphed for illustration purposes in Figure 2.12.



(a) Sigmoid function

(b) Derivative of the sigmoid function

Figure 2.12.: Sigmoid function and the derivative of the sigmoid

The most widely used activation function is the sigmoidal activation function because it models the linear function around zero while modelling the threshold function at higher net values. Another advantage is the ease of calculating the derivative of the function (as shown in 2.12b), which is required during the training process [89] (see subsection 2.6.4). For these reasons, the function was also chosen for implementation in this study.

Layers

The biological neural network has a large number of complexly structured neurons, where a large number of synapses exist to transmit information between neurons. The ANN uses a simplified structure where the network is structured into layers of neurons, to reduce the complexity of the system. These neurons are then connected to create a neural network.

Neural networks with feedback also exist, where information is fed back into the system. Because of operation and training complexities, the feed-forward neural network structure is the most popular approach. The feed-forward neural network was also chosen due to re-use of hardware as well as reduced complexity compared to other structures, reducing the complexity of implementation and resource requirement on the FPGA.

Traditionally an ANN has three layers, as follows:

- Input layer;
- Hidden layer;

- Output layer.

The input layer is a layer of neurons directly connected to the inputs of the network, the hidden layer is the layer of neurons connected between the input and output layer, and the outputs of the output layer are also the outputs from the network. More than one hidden layer can be used. In this study a single hidden layer was chosen as it is known that a three-layer network (with one hidden layer) can construct arbitrarily complex decision boundaries (see section 2.6.4.2).

In this study, the input layer is regarded as a layer, and a complete network is regarded as a three-layer neural network. A visual representation of a generic three-layer network is shown in Figure 2.13.

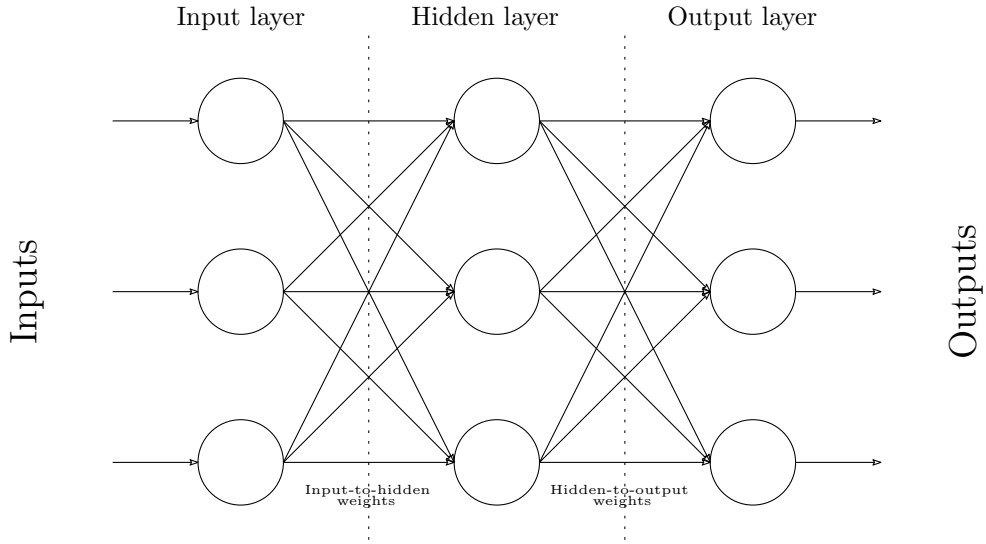


Figure 2.13.: A generic conceptual ANN structure, including the input, hidden and output layers, where each circle represents a neuron in the network

Each layer in the network consists of a set of neurons, where the neurons of the different layers are interconnected (see Figure 2.13). As mentioned earlier, a feed-forward configuration was used, which implies the flow of data only in the direction of the inputs to the outputs. With the feed-forward configuration, only adjacent layers were connected, and no neurons were connected in the same layer. Also, the same activation function for all the neurons in the hidden and output layer was used, while an identity function was used as activation function for the input layer [89]. The inputs were then propagated forward through the network to give an output.

Network Operation

The following section discusses the operation of a generic ANN and the relevant mathematics to calculate the output is shown. Refer to Figure 2.14 for all variables that are used in calculations.

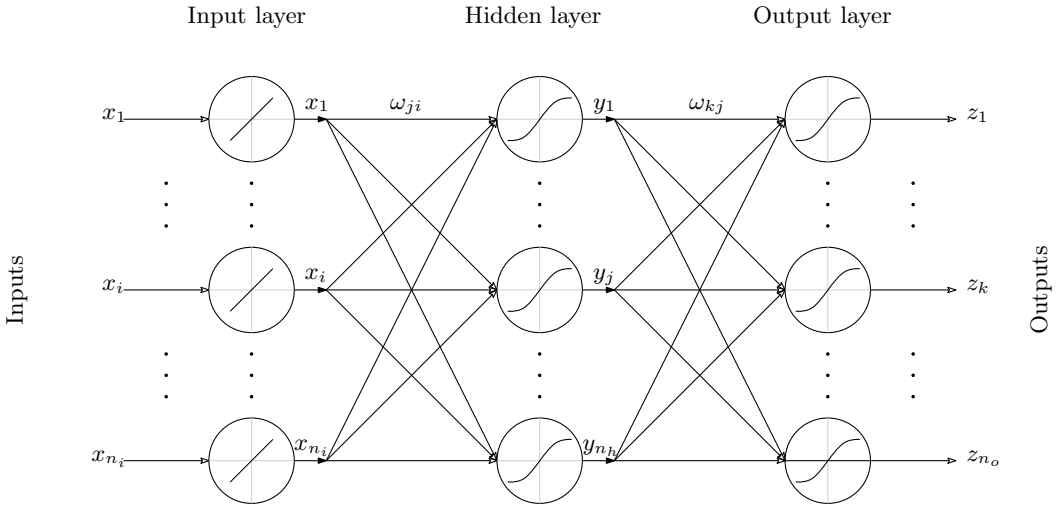


Figure 2.14.: A detailed generic ANN (adapted from [2])

Assume a set of inputs is given, where the inputs are structured into a vector (also referred to as an input vector). As the input layer makes use of the identity function as the activation function, the outputs of the input vector are directly propagated. The input vector $\left(\mathbf{x} = [x_1, x_2, x_3, \cdots, x_{n_i}]^T\right)$ is now also the input vector of the hidden layer. For each neuron in the hidden layer, the net activation can be calculated by using Equation 2.11 [2]:

$$net_j = \sum_{i=0}^{n_i} x_i \omega_{ji} = \sum_{i=1}^{n_i} x_i \omega_{ji} + x_o = \mathbf{w}_{jx}^T \boldsymbol{x} \tag{2.16}$$

where

i and j are the indices of the input x and neuron, respectively,

n_i is the total number of inputs,

ω_{ji} is the weight between input i and neuron j,

x_0 is the bias input,

$\mathbf{w}_{jx}$ is the input-to-hidden weight vector to neuron j, and

$\mathbf{x}$ is the input vector. The net activation can then be used to calculate the output of each hidden neuron, as follows [2]:

$$y_j = f\left(net_j\right) = f\left(\sum_{i=0}^{n_i} x_i \omega_{ji}\right) \tag{2.17}$$

where

y_j is the output of the hidden neuron with index.

The outputs of the hidden neurons were used to construct an output vector, and denoted $\left(\mathbf{y} = [y_1, y_2, y_3, \cdots, y_{n_h}]^T\right)$

As mentioned in section 2.6.3.2, the outputs of the neurons in adjacent layers are used as the inputs to the next layer. Thus, the outputs of the hidden layer $\mathbf{y}$ are the inputs of the output layer. The same approach is then followed to calculate the output of the output neurons, as shown below [2]:

$$net_k = \sum_{j=0}^{n_h} y_j \omega_{kj} = \sum_{j=1}^{n_h} y_j \omega_{kj} + x_o \tag{2.18}$$

where

k and j denote the indices of the output neuron and hidden neuron, respectively,

n_h is the total number of hidden neurons, and

ω_{kj} is the weight between the hidden neuron j and output neuron k.

Since the output of the output layer is the result of the network, the output of each output neuron is considered the discrimination function of the class, and is mathematically represented as follows [2]:

$$g_k(\mathbf{x}) = z_k = f\left(net_k\right) = f\left(\sum_{j=0}^{n_h} y_j \omega_{kj}\right) \tag{2.19}$$

where

$g_k(\mathbf{x})$ is the discrimination function of class k as a function of input $\mathbf{x}$,and

z_k is the output of the output neuron with index k.

And since the results of the output layer are also the output of the network, z_k is also the output for class k.

Combining Equation 2.17 and Equation 2.19, the discrimination function of class k can be mathematically represented as [2]:

$$g_k\left(\mathbf{x}\right) = f\left(\sum_{j=0}^{n_h} f\left(\sum_{i=0}^{n_i} x_i \omega_{ji}\right) \omega_{kj}\right) \tag{2.20}$$

The outputs of the hidden neurons were used to construct an output vector, and denoted $\left(\mathbf{z} = [z_1, z_2, z_3, \cdots, z_{n_o}]^T\right)$, where n_o denotes the number of outputs.

2.6.4. Neural Network Training

2.6.4.1. Introduction

In the following section an overview of the training process of an ANN, as well as the implementation in software, is discussed. All pattern recognition algorithms require previous or statistical data on which the operation is based, and the same applies to ANN. The use of an ANN without training is of no use as the result will be meaningless. The ANN can be trained as to what to expect as inputs during operation, and to classify these inputs according to a previously arranged training set.

The objective of the training of the classifier is to adapt the classifier in such a manner that the classification accuracy of unseen data is increased (i.e. to generalise). In the domain of ANNs, this is achieved through the adjustment of weights between neurons.

In the case of this study, the classifier was trained using the training set, after which the acquired weight vectors were then transferred to the FPGA for classification.

The most frequently used algorithm for training an ANN is the back-propagation gradient descent training algorithm. An overview of the algorithm is given bellow.

2.6.4.2. Back-propagation Training Algorithm

The operation of an ANN is solely based on the values of the weights. The process of changing the weights of a network to increase classification performance is referred to as learning. The back-propagation algorithm is used to adjust the weights to increase the classification performance of the classifier when unseen examples are provided. As the classification performance improves the classification error needs to decrease.

Disadvantages of the back-propagation method are the slow speed and the possibility of finding local minima instead of global minima [90]. This is not a major issue and can be addressed by using a proper training methodology that selects the best network from a set of trained networks.

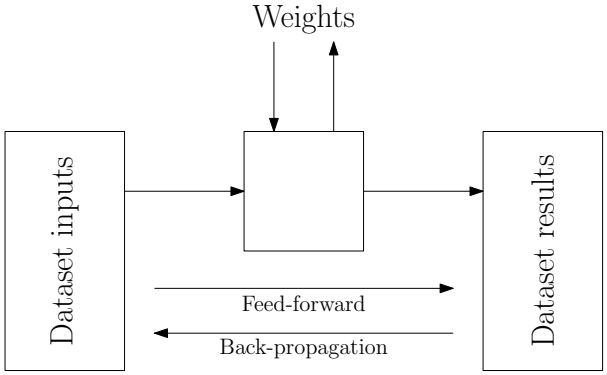


Figure 2.15.: The iterative training process

In Algorithm 2.1 a generic back-propagation algorithm is shown. An overview and more details on the algorithm then follow.

Algorithm 2.1 Generic back-propagation algorithm [2]

Input: Training dataset, with both inputs and corresponding outputs
Output: Adjusted weight matrices
Initialise weight matrices:
$\mathbf{w}_{ji}$ - Input-to-hidden neuron weight matrix
$\mathbf{w}_{kj}$ - Input-to-hidden neuron weight matrix
Initialise learning parameters, stop criterion

1: **repeat**
2: $\mathbf{x}_r \leftarrow$Randomly selected input vector
3: $\mathbf{z}_k :=$ Neural network output as function of $\mathbf{x}_r$
4: $\triangle E \leftarrow$ Error calculated
5:
6: {Update input-to-hidden layer weights}
7:
8: **for** each ω_{ji}: **do**
9: $\omega_{ji} := \omega_{ji} + \triangle\omega_{ji}$
10: **end for**
11:
12: {Update hidden-to-output layer weights}
13:
14: **for** each ω_{kj}: **do**
15: $\omega_{kj} := \omega_{kj} + \triangle\omega_{kj}$
16: **end for**
17: **until** stop criterion reached
18: **return** $\mathbf{w}_{ji}$, $\mathbf{w}_{kj}$

Algorithm 2.1 is discussed in more detail in the following section.

Weight Adjustment

The objective of the training of an ANN is to reduce the classification error. Classification error in this case refers to the incorrect classification of a set of inputs, according to the known output. The classification error is usually calculated by using a test set that contains previously unseen samples to give an indication of the accuracy of the classifier. The classification error is not calculated on the training set since the classifier will be biased as the samples have already been seen. For the training process, an error metric called the training error is used, which is a metric of overall error between the estimated output and the actual required output. This least-squares error (LSE) error can be mathematically represented as follows (adapted from [2]):

$$E(\mathbf{w}) = \frac{1}{2}\|\mathbf{z} - \mathbf{t}\|^2 = \frac{1}{2}\sum_{i=1}^{n_o}(z_k - t_k)^2 \tag{2.21}$$

where

E is the error function,

$\mathbf{z}$ is denoted as the output vector of the network with entries z_k at index k, and

$\mathbf{t}$ is the target or actual output vector with entries t_k at index k.

Finding a minimum of a function with low dimensionality can be a rather simplistic process. However, in the majority of cases of neural network training, the dimensionality is high and locating minima becomes quite complex.

A solution to finding the minima of the LSE is through a technique called gradient descent. Gradient descent takes into account the gradient of the error curve, and change the weights accordingly to decrease the error. The change in weights can be mathematically represented as follows [2]:

$$\triangle\mathbf{w} = -\frac{\eta\partial E(\mathbf{w})}{\partial\mathbf{w}} \tag{2.22}$$

where

$\triangle\mathbf{w}$ is the matrix of changes of all the weights in the network, and

η is the learning rate (discussed in more detail in section 2.6.4.2).

The weight matrices are then updated as follows:

$$\mathbf{w} = \mathbf{w} + \triangle\mathbf{w} \tag{2.23}$$

In Figure 2.16 a detailed diagram is shown and used as the basis for the discussion of the back-propagation training algorithm.

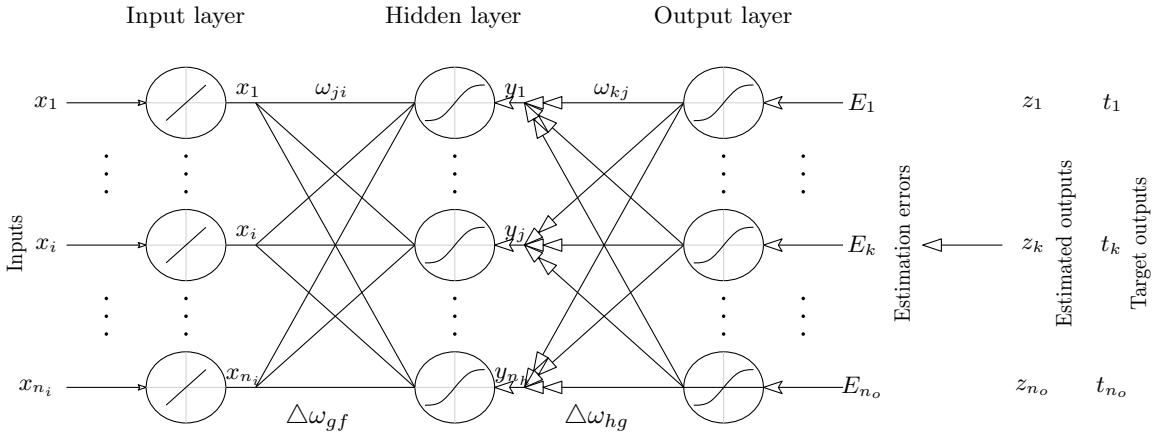


Figure 2.16.: Diagram to explain back-propagation of error to calculate all weights (adapted from [2])

The gradient descent calculation of the error function is discussed according to the location of the weight in the network. The two categories being discussed are hidden-to-output weights and input-to-hidden weights. Note that calculation of the update in weight is used as part of Algorithm 2.1.

Hidden-to-output Weights

First, consider the case of gradient descent with a hidden-to-output weight as parameter. Equation 2.21 and Equation 2.22 can be combined to obtain the following gradient descent equation [2]:

$$\triangle \mathbf{w} = -\eta \frac{\partial}{\partial \mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{n_o} (z_k - t_k)^2 \right] \tag{2.24}$$

It can also be considered on a weight-by-weight basis. Assume ω_{hg} is a specific hidden-to-output weight in the network. The change of ω_{hg} is denoted $\triangle\omega_{hg}$, and is mathematically calculated as follows [2]:

$$\triangle \omega_{hg} = -\eta \frac{\partial}{\partial \omega_{hg}} \left[\frac{1}{2} \sum_{i=1}^{n_o} (z_k - t_k)^2 \right] \tag{2.25}$$

By using Equation 2.19, Equation 2.25 and the chain rule, the change of the weight can be simplified as follows [2]:

$$\triangle\omega_{hg} = -\eta\left[z_g - t_g\right] f'\left(net_g\right) y_h \tag{2.26}$$

where

$f'(\cdot)$ is the derivative of the activation function used.

A differentiable activation function needs to be used as the derivative of the activation function is required for training. For every weight in the hidden-to-output weight matrix, Equation 2.26 is used to calculate the required change of each weight.

Considering a full ANN in forward-propagation operation, only a single neuron's output value will change when a single hidden-to-neuron weight is changed. When a single input-to-hidden weight is changed, the effect will be propagated through the hidden layer and will affect all output neurons. The case of the input-to-hidden weights will now be discussed.

Input-to-hidden Weights

With the input-to-hidden weights, a change in a weight results in a change in the output of a single neuron in the hidden layer, which results in a change in all neurons in the output layer. As the weight is before the hidden layer, the training errors also need to be back-propagated through the hidden layer to calculate the appropriate change for each weight. Even though the input-to-hidden weights are considered, Equation 2.21, Equation 2.22 and Equation 2.24 still apply.

The whole network can be considered on a weight-by-weight basis, assuming ω_{gf} is a specific input-to-hidden weight in the network. The change of ω_{gf} is denoted $\triangle\omega_{gf}$, and is mathematically calculated through the use of the chain rule as follows [2]:

$$\triangle\omega_{gf} = -\eta\frac{\partial}{\partial\omega_{gf}}E(\mathbf{w}) \tag{2.27}$$

$$\triangle\omega_{gf} = -\eta\frac{\partial E(\mathbf{w})}{\partial z_k}\frac{\partial z_k}{\partial\omega_{gf}} = -\eta\sum_{k=1}^{n_o}\left(z_k - t_k\right) \times \frac{\partial z_k}{\partial\omega_{gf}} \tag{2.28}$$

$$\frac{\partial z_k}{\partial\omega_{gf}} = \frac{\partial z_k}{\partial C_k}\frac{\partial C_k}{\partial\omega_{gf}} = f'\left(c_k\right) \times \frac{\partial C_k}{\partial\omega_{gf}} \tag{2.29}$$

$$\frac{\partial C_k}{\partial \omega_{gf}} = \frac{\partial C_k}{\partial y_g} \frac{\partial y_g}{\partial \omega_{gf}} = \omega_{kg} \times \frac{\partial y_g}{\partial \omega_{gf}} \tag{2.30}$$

$$\frac{\partial y_g}{\partial \omega_{gf}} = f'(b_g)\, x_f \tag{2.31}$$

Combining Equation 2.28 with Equation 2.31, the final weight update equation is mathematically represented as follows:

$$\triangle \omega_{gf} = -\eta \sum_{k=1}^{n_o} (z_k - t_k) \times f'(c_k) \times \omega_{kg} \times f'(b_g)\, x_f \tag{2.32}$$

Training Parameters

Weight Initialisation

During training weights are adjusted to reduce the training error. By correct initialisation of the weights before training, the training algorithm can more easily locate the global minima. It is not a clear science to find the best initialisation position, and [2] recommends that the initial weight values should be chosen such that the net activation is in the range $-1 < net_j < 1$, as it covers the complete linear range of the activation function. Thus, the values of ω_{jk} should be chosen such that Equation 2.16 ranges between -1 and 1.

Learning Rate

The learning rate is used in multiple equations in this chapter and is denoted as η. The objective of the training of the ANN is to find a global minimum for the error, which should result in a global maximum for the classification accuracy. To prevent the training from being biased, the change of weights is restricted to find a set of weights that provides a minimum error for all different training samples. Weight adjustment is done by using the learning rate coefficient, which scales the value in such a manner that the changes of the weights are limited. There is no definite value for the learning rate, but a rule of thumb is $\eta = 0.1$ [2]. It was also empirically shown that a learning rate of $\eta = 0.1$ yields the most satisfactory results.

Momentum

Another technique to reduce the possibility of falling into a local minimum during the search for a global minimum is the use of momentum. The principle of momentum is to have a weighted effect of previous weight changes to calculate the weight change at a following iteration. This results in current changes being influenced by previous weight changes. Referring to Equation 2.23, the change of weights with momentum is mathematically represented as follows [2]:

$$\mathbf{w} = \mathbf{w}(n) + (1 - m) \triangle \mathbf{w}(n) + m \triangle \mathbf{w}(n - 1) \tag{2.33}$$

where

n is the iteration index, and

m is the momentum coefficient.

To ensure that momentum is the main source of change in the weights, the momentum coefficient was chosen between 0.5 and 1. The momentum coefficient in this study was chosen at $m = 0.75$.

Number of Hidden Layers

Both feed-forward and back-propagation can be adapted for the use of multiple hidden layers, as the same principles discussed in the chapter can be used to add another layer. Through empirical studies it has been found that in general a single hidden layer is sufficient for most applications [1, 91].

Number of Hidden Neurons

The number of hidden neurons affects the degree of generalisation of the classifier. The ideal number of hidden neurons results in an increase in acceptable performance while maintaining a high level of generalisation. As a rule of thumb, it is recommended that the number of weights in the network be equal to $\frac{n}{10}$, where n is the number of available training samples [2]. However, because of the larger input vector, smaller number of available training samples and complexity of the dataset, it was not an effective rule of thumb. The number of hidden neurons was determined empirically in this study.

Weight Decay

Weight decay is a technique used to reduce the number of weights in a network. In general weight decay can be used in an attempt to improve the accuracy of a

classifier, even though it is not always effective [2]. The most basic concept of weight decay is to reduce the weight value at the end of each training iteration in an attempt to remove weights that do not have an effect on the classification. In this case it was also considered as a method to reduce the number of inputs or hidden neurons, if all weights to or from a neuron are redundantly small. Weight decay can mathematically be represented as follows [2]:

$$\omega_{new} = (1 - d) \times \omega_{old} \tag{2.34}$$

where

ω_{old} and ω_{new} are the value of the weights before and after weight decay, respectively, and

d is the decay coefficient.

Since numerous iterations are performed during training, the value of d is kept small to ensure decay while not affecting the training process.

Weight decay was initially used in this research in an attempt to reduce the number of weights and inputs if the weight values to and from a specific neuron were redundantly small. This was not effective and weight decay had no influence on the performance of the classifier or the number of inputs or hidden neurons, and was eventually removed from the training process.

Under-fitting and Over-fitting of Training

Under-fitting and over-fitting of a classifier are terms used to describe the degree of training performed on the training set as well as the ability to perform classification on a training set. Under-training refers to a classifier where the training process was not completed to the point of maximum performance and generalization, resulting in a classifier that cannot perform classification on the training- or test sets. Over-fitting of a classifier refers to a classifier that was over-trained, resulting in a very good classifier on the training set, but low performance on the test set. This also refers to the generalisation of a classifier. Ideally a middle point between these extremes needs to be found, where the classifier performs optimally on the test set and the classification error is a global minimum.

2.7. Technology Overview

2.7.1. Overview

In this section an overview of the hardware used in this study is provided. The architecture is also briefly discussed, followed by information on the FPGA and memory that were used.

2.7.2. Hardware Architecture

The architecture of the hardware used is briefly discussed in this section. An overview of the hardware architecture used during this study is provided in Figure 2.17.

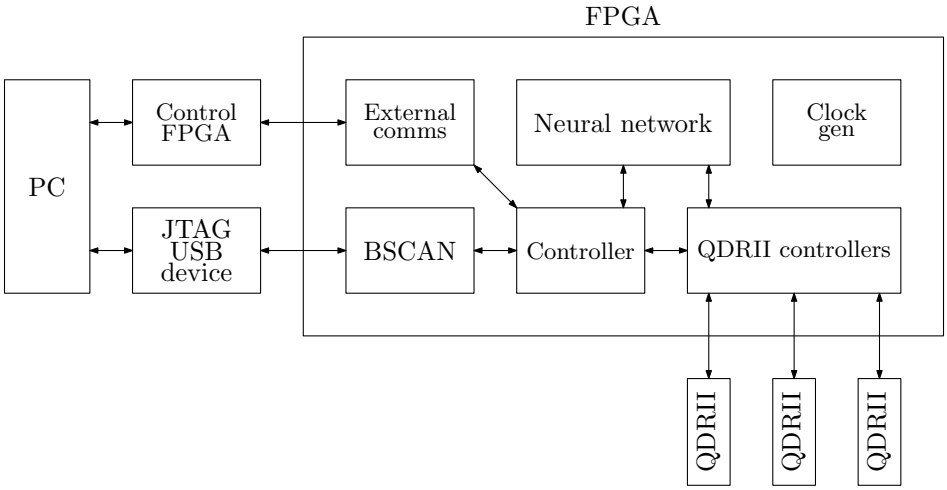


Figure 2.17.: Hardware architecture

The majority of functions were implemented on the FPGA, including the neural network, controllers and communications. Communication between the personal computer (PC) and FPGA was performed through user datagram protocol (UDP), while debugging and programming was performed through a Joint Test Action Group (JTAG) universal serial bus (USB) programmer. The quad data rate (QDR) static random-access memory (SRAM) controller was also used to control write/read functions to/from the QDR memory. The system made use of a 125 Mhz clock throughout, with the exception of the UDP communication protocol, which made use of a 100 Mhz clock.

2.7.2.1. Memory

The QDR memory of the hardware is used for storage of information used by the neural network, i.e. inputs, weights and PWL parameters. The structure of the memory allocation is as illustrated in Figure 2.18.

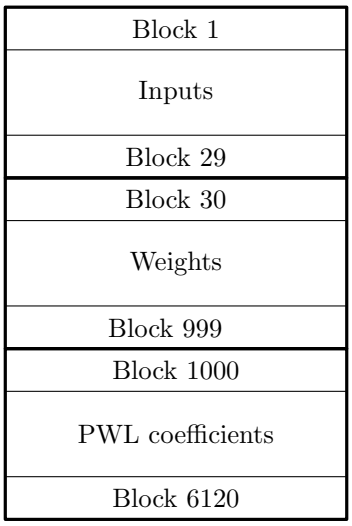


Figure 2.18.: Memory allocations

2.7.2.2. Xilinx Virtex 5 FPGA

FPGAs consist of a set of configurable logic blocks (CLBs), usually structured in an array structure. These CLBs are structure in such a way as to provide a functional circuit as described in the hardware description language (HDL).

The FPGA used in this study is part of the Virtex 5 family FPGAs manufactured by Xilinx. The CLBs of the FPGA used consist of 14 720 Virtex 5 slices, as well as 610 DSP48E slices. DSP48E slices are unique slices contained within the FPGA, of which contains an internal multiplier. In the case of this study, this is a significant advantage, as it was known that a large number of multipliers would be required for the implementation of any of the classification algorithms. By using the DSP48E slices, the implementation of multipliers in combination logic is avoided if the number of required multipliers does not exceed the number of available DSP48E slices.

In this study, the HDL chosen to be used was VHDL.

2.7.2.3. Transfer of Data between FPGA and PC

Communication between a programming PC and FPGA is required in order for the system to function and be evaluated.

The classifier was designed in such a manner as to obtain all information from the memory on the printed circuit board (PCB). The classifier was designed to read the on-board memory, so the actual data needed to be transferred from the PC to the memory on the PCB. The system received from the Council for Scientific and Industrial Research had existing software and hardware to upload data to the network, and was adjusted and configured to fit the application. As the inputs,

weights and LUT values needed to be stored in memory, sections of memory blocks were allocated to each.

2.7.2.4. Advantages and Disadvantages of FPGAs

The advantages and disadvantages of FPGAs are compared with both digital signal processors (DSPs) and application-specific integrated circuit (ASIC) platforms to design a system.

The advantages of the use of FPGAs are as follows:

- Reduced development risk compared to ASIC;
- Reduced development time compared to ASIC;
- Reconfigurability;
- Modularity;
- Parallelism;
- If required, the ability to increase precision;
- Performance of high-speed applications compared to microprocessors.

A list of disadvantages for the use of FPGAs is as follows:

- Complexity of design (timing, parallelism, latencies, etc.);
- Microprocessors are more affordable than FPGAs;
- Less efficient than DSPs on arithmetic calculations.

2.7.3. Global System for Mobile Communications

GSM is globally used as the standard for mobile communication, is a protocol used to perform wireless transmission of voice and data information over a network between different devices. The objective of this study is to determine whether SEI can be performed with GSM handsets on an FPGA. Therefore, basic background information on GSM systems is required.

2.7.3.1. Architecture of Global System for Mobile Communications Networks

To send data from a single device to another, a wide range of subsections are required. These subsections are briefly discussed below [92].

Mobile Station

The mobile station (MS) refers to the the hardware used by the user. Different types of MS exist, such as vehicle-mounted stations, portable stations and handheld stations. In this study the focus was on handheld stations, mostly referred to as a handset. An MS is used to connect a user to the network, usually to send/receive information through the GSM network to/from another MS. The objective of this study is to perform identification of MSs from unique variations in the transmitted RF signal. From the MS, RF signals are emitted to the base station subsystem (BSS) before propagating through the network to the desired destination [92].

Base Station Subsystem

A BSS is a station with RF hardware to provide coverage to a specified geographical area. MS within the coverage area of the BSS can connect and communicate with the BSS. Multiple BSS are located at different geographical locations to cover a wide area [92].

Mobile Service Switching Centre

All transmissions received are sent to the mobile service switching centre (MSC). An MSC is used to control the flow of information to the correct BSS, to be delivered to the correct MS. The MSC is also responsible for the management and optimum operation of the network [92].

2.7.3.2. GSM Radio Interface

Modulation - Gaussian Filtered Minimum Shift Keying

GMSK is used on the GSM 900 band as modulation scheme. Minimum shift keying (MSK) makes use of frequency shift keying to transmit information over a channel. With MSK an optimal bandwidth setup is used to obtain the best trade-off between throughput and amount of ISI. In addition, GMSK makes use of a Gaussian filter to reduce the amount of ISI between different transmission channels [93].

The specifications of GMSK used for GSM are as follows [5]:

Modulation rate = 270,83 kbit/s

Channel spacing = 200 kHz

GSM 900 - 174 RF channels

DCS 1800 - 3374 RF channels

Time Division Multiple Access

GSM makes use of time division multiple access (TDMA), where each TDMA time frame has eight time slots. Each time slot lasts 576.9 μs, while the transmission of information at a modulation rate of 270.833 kbits/s, resulting in 156.25 bits being transmitted within each time slot [5].

A basic visual illustration of the TDMA structure is provided in Figure 2.19.

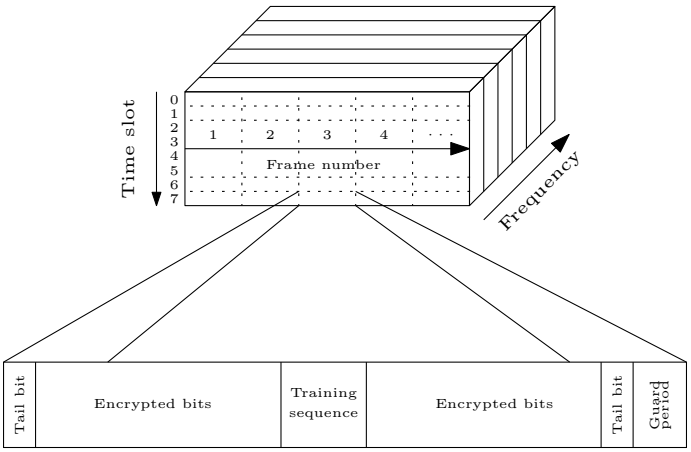


Figure 2.19.: GSM signal structure (Adapted from [5])

Each frequency channel has eight time slots within which a burst is transmitted. Since TDMA is used, eight different transmissions can be performed at the same time while using the same frequency channel. Each TDMA frame lasts for 4.615 ms (576.9 μs $\times$ 8), after which the frame number is increased [5].

2.7.3.3. Burst Structures

The main burst structures used by GSM are as follows [94]:

Normal Burst

A normal burst is used to transmit information over the GSM network. The structure of the normal burst is displayed in Figure 2.19. To reduce unwanted variations between captured bursts, only normal bursts were used during this study, while the other bursts' structures were ignored. The training sequence bits are used as modulation bits and used to control the modulation variables as required [94].

Frequency-correction Burst

Different frequency channels are used in the GSM network. Hence, a method is required to perform the frequency correction of an MS. This is performed through the use of a frequency-correction burst, synchronising the MS to the GSM network [94].

Synchronisation Burst

Synchronisation in time is also required and is performed by using the synchronisation burst. Information such as the TDMA frame number is transmitted within the training sequence to synchronise the handset to the network [94].

Access Burst

Access bursts are used to communicate with an MS where the exact timing of the handset is unknown. This is also useful in cases where the distance between the MS and BSS is 35 km or further [94].

2.7.3.4. Assumptions with regard to GSM

With regard to this study, only normal bursts were considered to reduce any unwanted variations in the system. As discussed in subsubsection 2.6.2.2, a GSM receiver was implemented by [1] to perform the capturing of data. This resulted in the ability to determine the type of burst that was transmitted, as well as the identity of the handset through IMSI or IMEI for verification purposes. A large feature set might have an influence on the efficiency of a classifier owing to the "curse of dimensionality" - in this study only samples in the training sequence where used.

2.8. Conclusion

An overview of previous research concerning SEI was provided in this chapter. A literature study was undertaken on the different stages of the SEI process. Different methods for each stage of the SEI process have been suggested in literature. The different stages of the SEI process that were investigated were feature extraction and classification or identification of the unique emitter. In the literature it can be seen that, with proper feature extraction and classification techniques, effective SEI for GSM handsets can be performed.

The goal of the research is to determine if an SEI system for GSM classification can be implemented on an FPGA. All the different methods and algorithms considering the implementation on FPGA technology have been discussed.

It was found that the largest obstacle for such implementation on FPGA is the amount of logic required to implement multipliers in combinational logic. Methods such as parallel/serial implementations, LNS, shift-and-add operations, DA and reconfiguration of FPGAs were discussed, that can possibly be used to reduce the number of multipliers required.

Techniques for fingerprinting and feature extraction were studied, which include feature extraction in a variety of representations (time, TFR and phase), non-linear effects of emitters as well as modulation errors. Even though different feature extraction methods were investigated, no single method can be universally used for all SEI applications because of the variations in application, type of signal and type of emitter. In each unique case different representations of the signals need to be analysed to find appropriate features. In this study modulation errors, and more particularly PE, was used since it was shown to be the most effective feature for GSM SEI.

Different classification methods have been suggested in literature, most of them suggesting the use of non-parametric classifiers. The classification methods discussed include k-NN, ANN, SVM and SNN.

An ANN was selected as opposed to k-NN, SVM and SNN. And since FPGAs and ANN are both highly parallel, the ANN was found to be the optimal solution. Also, due to the repetitive nature of an ANN, hardware could be re-used. The re-use of resources in hardware was thus supported by the repetitive structure of an ANN.

3. Design and Synthesis of Artificial Neural Networks

3.1. Introduction

In this chapter, material presented in the literature study (chapter 2) is used as the basis for the design of the GSM SEI. In the design phase, objectives set in chapter 1 were kept in mind. As it was required to implement the system in hardware, i.e. an FPGA, design considerations were made while keeping the constraints and limitations of FPGAs in mind. Design and implementation of an ANN in software (section 3.2) as well as in VHDL (section 3.3), are discussed in this chapter. Finally, a conclusion is presented (section 3.5).

3.2. Software Implementation

In this section the software implementation of the GSM SEI system is discussed. Firstly the implementation of an ANN in software is discussed. A discussion on a C++ application used for control and communication follows.

3.2.1. Artificial Neural Network Implementation in Software

A secondary objective of this research is to determine if an ANN can be used for classification of GSM handsets. This objective is verified through implementation and evaluation of the algorithm in software. Properly functioning software implementation of the ANN was also required for offline training of the classifier. The software implementation was used as basis for the design on an FPGA, which decreased the VHDL development time as well as simplifying the debugging process. That is, a PC implementation of the ANN was used to provide a reference against which the FPGA could be verified. This formed an important part of the FPGA design verification. The implementation of the operating software ANN was completed in MATLAB.

The algorithm for the ANN is given in Algorithm 3.1, based on information supplied in subsubsection 2.6.3.2. (see section A.1 for actual MATLAB source code listing.)

Algorithm 3.1 Software implementation of a generic ANN

Input: Input feature set, input-to-hidden and hidden-to-output weights
Output: An array of output, each value corresponding to a class

Initialise: Load input-to-hidden weights $\mathbf{w}_j$ and input vector $\boldsymbol{x}$

Hidden layer calculation:

$\mathbf{x} := [1, \mathbf{x}]$
for $j := 0$ to n_j: **do**
  $net_j := \mathbf{w}_j^t \cdot \boldsymbol{x}$
  $y_j := f(net_j)$
end for

Hidden layer calculation:

Load hidden-to-output weights $\mathbf{w}_k$

$\mathbf{y} := [1, \mathbf{y}]$
for $k := 0$ to n_k: **do**
  $net_k := \mathbf{w}_k^t \cdot \boldsymbol{y}$
  $z_k := f(net_k)$
end for

The following sigmoid activation function was used for both hidden and output neurons:
$f(x) = \frac{1}{1+exp(x)}$

Together with the original software version of the ANN, two additional versions were developed for the purpose of run-time debugging of the VHDL ANN. These two versions incorporated scaling (see subsubsection 3.3.4.1) and approximation of the activation function (see section 3.3.5.2). These software implementations were then used to verify operation of the ANN on the FPGA. The debugging of the system is discussed in section 3.4.

The back-propagation training algorithm, as given in subsection 2.6.4, was also implemented in MATLAB. The resulting weights were used for both software and FPGA implementations of the ANN.

3.2.2. Control and Communications Software

A C++ application was used to control the complete system. The roles of the C++ application include the following:

- Communications between the PC and FPGA;

- Memory tuning;
- Memory access;
- Mode selection;
- Control of the operations on the FPGA;
- Extraction of results from the FPGA;
- Displaying results.

During operation, the hardware (FPGA) and C++ application are initialised. After initialisation, memory tuning and testing are performed to ensure correct operation of memory. Ensuring the correct operation of memory is essential as any error in memory will have a large effect on the final result of the ANN. The C++ software is used to read text files containing input, weights and LUT values. This data are then transmitted through UDP to the FPGA and stored on the QDR memory. Mode selection is used to make adjustments to the mode controller block on the FPGA (see subsubsection 3.3.4.2). When the neural network mode is activated, the propagation of the inputs through the network is initiated. After propagation of the data through the complete network, the output is transmitted back to the C++ software and displayed on the screen.

For evaluation of the operation and to determine the identification performance of the ANN, the test set was consecutively propagated through the ANN. Each output transmitted back to the C++ software was then written to a text file. The text files were then loaded into MATLAB, where performance evaluations and comparison studies between different implementations were performed.

3.3. Very High Speed Integrated Circuits Hardware Description Language Implementation

3.3.1. Introduction

This section focuses on the implementation of the ANN on an FPGA. A similar approach to the software design of the classifier was followed, while the design was adapted for FPGA implementation where required. First an overview is given. The design cycle followed for VHDL designs is then discussed, followed by general VHDL design information. Finally, a detailed description of the design of the ANN for implementation on an FPGA is provided.

3.3.2. Overview

All decisions on the ANN design for the FPGA consisted of a balance between performance in execution speed (latency), performance with regard to accuracy as

well as hardware resource requirements. All design choices in this study were made by considering this balance, to obtain an appropriate classifier. Since no real-time classification was required, the assumption was made that the performance in speed was considered less important, while more consideration was given to accuracy of classification performance and resource requirements during the process of decision-making.

The hardware setup was as explained in subsection 2.7.2.

3.3.3. Design Cycle

The design cycle followed for the design and implementation of the ANN in VHDL was as follows:

1. Create hardware description;
2. Generate in HDL Author;
3. Compile for ModelSim;
4. Functional verification and behavioural simulation in ModelSim;
5. Export to Xilinx ISE;
6. Synthesis, Map, Place-and-route in Xilinx ISE and create programming file;
7. Program FPGA;
8. Initialise system;
9. Functional and timing verification, together with run-time analysis in Chipscope.

A range of software and hardware tools was used during design and implementation of the ANN in VHDL.

Xilinx CoreGen (Core Generator system) is a library of core components that can be generated, assisting in fast and reliable implementation of basic building blocks. In this system CoreGen was used for the implementation of adders and multipliers.

Mentor Graphics HDL Author[1] is a hardware description visualiser, providing a graphical user interface with a visual representation of the VHDL system design, as well as flow of data through a design. HDL Author was used during the design of the system to create the hardware description through modular design, as well as controlling flow of information between the different functional units. With HDL Author, the design can easily be visualised, simplifying the design process. From the design in HDL Author, HDL Author ISE plug-in was used to export the project.

The exported project was then loaded into Xilinx ISE through execution of a script. Xilinx ISE is the vendor software for Xilinx FPGA platforms. Xilinx ISE was used

[1]More at http://www.mentor.com/products/fpga/hdl_design/hdl_author/

for synthesis, mapping, place-and-route and creation of the programming file. Verification of the synthesis and implementation of the design is provided, where timing results, resource usage and allocation are provided.

After verification of the results, Xilinx iMPACT tool together with the JTAG Xilinx programmer are used to transfer the design to the FPGA.

A C++ application is used for management of the system and flow of data between the PC and FPGA, as well as flow of data on the FPGA. Management was performed by software such that data can be provided to the system, and resulting data received. Registers were created in both software and hardware to enable communication and to enable collaboration between software and hardware sub-sections.

For debugging and evaluation purposes, ModelSim and Chipscope were used. ModelSim can be used for behavioural and functional simulation to ensure correct design before the synthesis process. This is beneficial, as synthesis of a design is extremely time-consuming. ModelSim can then be used to verify design changes in simulation. Chipscope Pro is an added accessory to the Xilinx tool flow, providing a runtime analysis feature where specified signals can be monitored within the physical FPGA design. For more information on how runtime analysis and debugging were performed, see section 3.4.

3.3.4. General Design

General design considerations were taken into account during the implementation of the ANN and are discussed below.

3.3.4.1. Scaling of Decimal Values

All calculation, inputs and results are in decimal values, mostly making use of values between -1 and 1. To simplify operations in VHDL, only integer values were used. As mostly decimal values were used, conversion and scaling of the data were performed. Two solutions for this were to use either floating point values or fixed point values. Because of reduced availability of core components in Xilinx Core Generator for floating point values, it was decided to use fixed point values. Fixed point arithmetic is considered similar to scaling of a value and involves multiplication of all values with a value that is kept constant and used as the scaling factor. With this approach, all Xilinx Core Generator components could still be used. However, owing to scaling and rounding, a small rounding error can occur. Scaling of decimal values was performed by using of the following equation:

$$I = d \times \frac{max_I}{max_d} \tag{3.1}$$

where

d is the decimal value being converted,

max_d is the maximum value of d,

I is the resulting integer value with max_I the maximum value of the integer, and

$\frac{max_I}{max_d}$ is the scaling factor.

The traditional binary-to-integer approach was followed to represent integer values in binary, and thus maximum integer values were determined by the number of bits allocated for the specific integer. Thus, Equation 3.1 could also be written as follows:

$$I = \frac{d}{max_d} \times 2^n \tag{3.2}$$

where

n denotes the number of bits to represent the decimal number in integer.

As the number is divided by the maximum decimal value, the complete full-scale (dynamic) range of bits is used.

Scaling was also used with the activation function where an integer value is used as an input to the function, to correspond to the values obtained with the original activation function.

3.3.4.2. Controller

Integration of different functions in the system was required. A controller block was implemented to control functions within the FPGA and ensure correct operation. The controller consists of the following functions:

- Control over the flow of data;
- Mode controller;
- Communications between FPGA and PC;
- Memory controller.

Within the mode controller, registers were defined for mode selection, as well as corresponding strobe signals to indicate when the mode had changed. The mode is selected in software and is transmitted to the memory controller, where the mode register is assigned a corresponding value and initiates the mode. Four basic modes exist within the system, as follows:

- Reset;
- Direct memory access mode (DMA);

- Neural network;
- Debug.

A description of each mode is provided below.

Reset

In reset mode the ANN on the FPGA is reset to its initial state. The system is reset during initialisation to ensure that all values are assigned correctly before any calculations are performed. After completion of the feed-forward propagation of the ANN, the system is also reset to clear all values, before a following iteration can be completed.

Direct Memory Access

DMA provides the ability to access memory directly on the hardware from the software. In DMA mode, data can either be written from a text file to memory or from memory to a text file. In this application, the DMA mode is primarily used to transfer inputs, weights, parameters and outputs between the PC and memory on the hardware. Communication between the PC and hardware is performed through the use of UDP. UDP protocol transmits 32 bits at a time. The memory, however, uses 72-bit words. Therefore, because the difference between the number of bits for UDP communications and the memory words, either padding or splitting and appending was used to ensure transmission of the correct value. To store 32-bit values in the memory, two 32-bit vales are padded to be stored into the memory. Also, when an array of more than 32 bits needs to be transmitted over UDP, the array is divided, sent in two separate packages and joined at the receiving end.

Neural Network

The neural network mode is used to enable the neural network of the FPGA. During neural network operation, the memory controller accesses the necessary information (such as the inputs, weight values and LUT values for the activation function) from memory, which was stored there through DMA. The ANN is then enabled, initiating propagation of the inputs through the network.

Debug

In debug mode all debugging blocks are enabled to provide the ability of validation and debugging of the system.

Memory Controller

In the memory controller, base addresses are defined to specify what sections of memory are allocated to what information, as well as counters to control the reading of information. Allocated registers on the FPGA are then filled with information read from the memory and are provided to the ANN when the neural network mode is enabled.

A discussion on the layout of the hardware of the memory controller follows. A visual representation of the memory and memory controller is provided in Figure 3.1.

First, a multiplexer is used to control the behaviour of the memory according to the selected mode. The behaviour of the memory is controlled by read and write strobes, as well as the address of the read/write command. In DMA mode, the strobes and addresses are controlled through software, whereas in neural network mode the strobes and addresses are adjusted within the hardware, according to what information is required.

The data in the memory is structured into an array of bits, split into three equal sections to be stored into the memory and concatenated during the reading process to obtain the initial data string. As mentioned earlier, the memory makes use of 72-bit words. As UDP makes use of 32 bits, two packages of data (i.e. 64 bits) are stored in memory, while the remainder of the bits are padded with zeros. Thus, a number of 192 bits can be stored in and read from the memory at each address across the three memory modules.

An illustration of the flow of information between the PC, memory and FPGA (ANN) is also provided in Figure 3.2.

In Figure 3.2 an overview of the flow of the inputs, weights, the parameters stored in the LUT and the output between different subsections in the system is visually illustrated.

3.3.4.3. Hardware Resource Reduction

Operations such as multiplication are resource-intensive when implemented in combinational logic. To reduce the required resources within the system, hardware resource requirements reduction techniques were incorporated. These techniques are discussed below.

Multiplexing/Reuse of Hardware

The hardware implementation of the network is similar to the theory explained in subsubsection 2.6.3.2. Different design trade-offs were required to make it suitable for implementation on hardware. During the following discussions of the different components of an ANN, it should be noted that all designs were done as generically

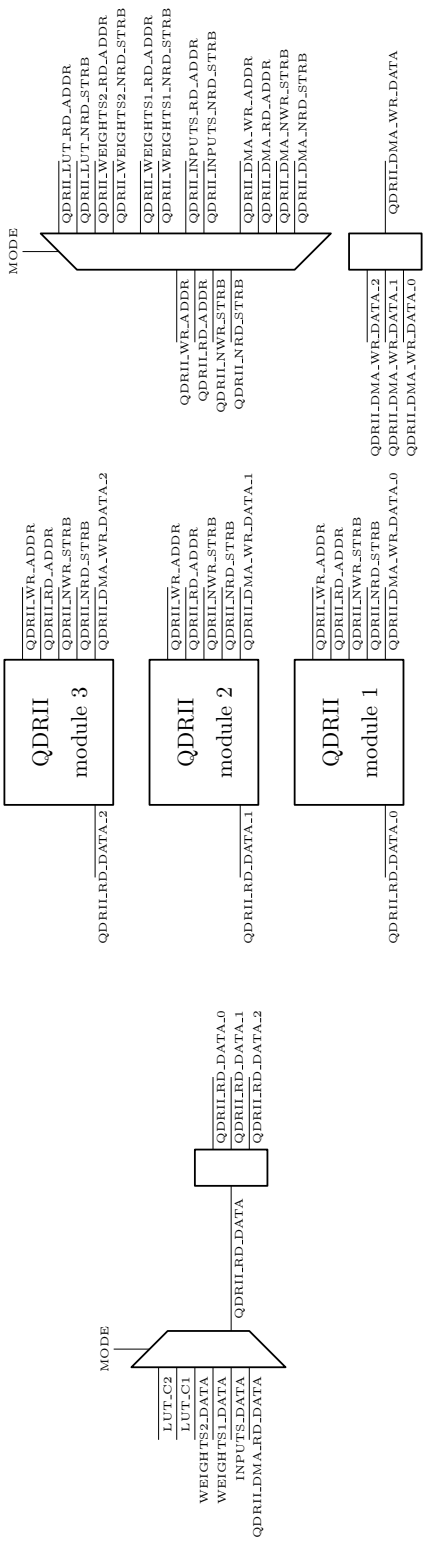


Figure 3.1.: Memory block diagram

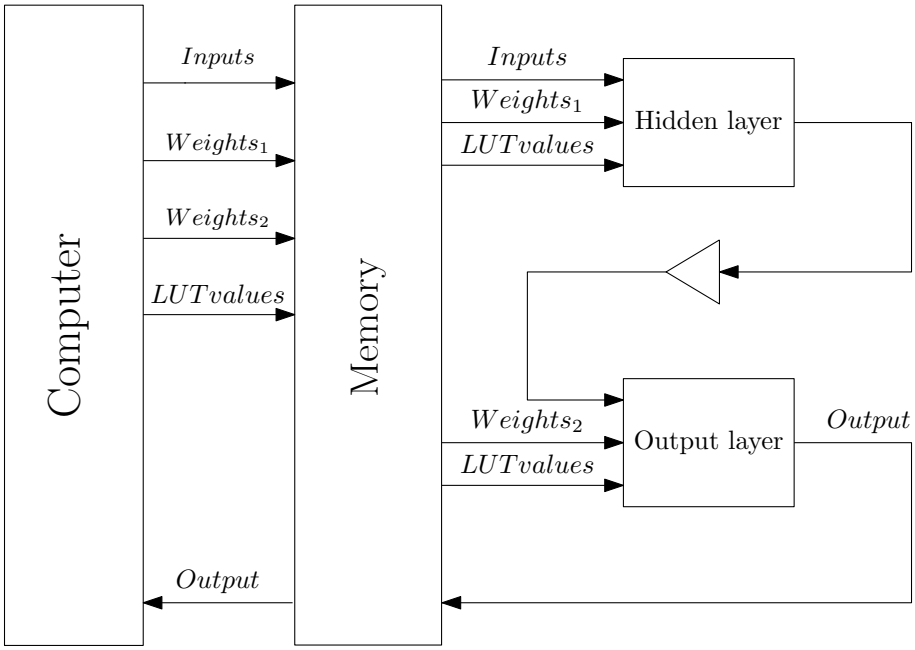


Figure 3.2.: Interfacing between the PC, on-board memory and FPGA

as possible. This was done to increase the re-usability of the different components to reduce area requirements. For this reason, where feasible, exactly the same hardware was used to perform a similar operation. A disadvantage of this approach however was reduced concurrency. Because of the increase of area efficiency through reuse of existing hardware, the design needs to operate more sequentially; parallelism was reduced and execution time was increased.

In the neuron, a single multiplier was used in the multiplication process of the inputs and weights. The inputs and weights were then changed accordingly, to multiply the different corresponding inputs and weights. Through this multiplexing technique only a single multiplier is used in the summing junction of the neuron. Also, only a single neuron was used in each layer, where the inputs and weights were changed according to the position in the network where the calculation was required. This resulted in only the hardware of a single neuron being required within each layer. A detailed discussion on this can be found in subsection 3.3.5.

Multipliers in combinational logic are resource-intensive and excessive use of multipliers could have resulted in excessive logic. For this reason, the use of multipliers was limited. Re-usability could be introduced on different levels of the neural network, as discussed in section 2.5. In this study, the re-usability of different components was implemented on all levels, i.e. multipliers, neurons and layers.

Binary Shift Operations

A technique to reduce the multiplier requirements is making use of bit-wise shift operators. Numbers in a binary representation can be shifted left or right to be

multiplied or divided by the power of two. Thus, by shifting a binary representation of a number by n bits to the left, the value of the representation is multiplied by 2^n. The same approach is followed for division by shifting bits to the right. This applies to truncation of bits of a bit string, where the value is divided by a value equal to the power of two and no division logic required. Through this approach, the use of multipliers was reduced in the design of the system.

Piece-wise Linear Look-up Table Function Approximation

Different design considerations were investigated to reduce hardware resources required for implementation of the activation function.

A PWL approximation through the use of an LUT was recommended by [95] to reduce the resource requirement of the activation function. More information on implementation of the PWL LUT approximation of the activation function is provided in section 3.3.5.2.

All these design approaches that were used to reduce the resource requirements are discussed in greater detail in the remainder of this section.

3.3.4.4. Basic Functionality

Basic functionality was required to create an functionally capable system. This functionality includes counters, buffers and delays.

Counters were mainly incorporated to control read and write addresses, as well as data flow within the FPGA. With multiplexing of data between different building blocks in the design on the FPGA, reliable control was required. Counters were designed through the use of state machines.

Buffers and delays were used for synchronisation and timing purposes. Buffers were mostly used for temporary storage of data, as well as delaying for a single clock cycle. The dynamic delay block can perform delays of multiple clock cycles and can be dynamically adjusted during run-time.

3.3.5. Artificial Neural Network Implementation

In this section implementation of the ANN in VHDL is discussed. The discussion follows a bottom-up approach as this reflects the process that was followed during implementation. Firstly, a single neuron will be discussed (see subsubsection 3.3.5.2), followed by a layer of neurons (see subsubsection 3.3.5.3), and finally the complete network (see subsubsection 3.3.5.4). In addition, a flow chart of the complete integration of the ANN is given in Figure 3.12.

3.3.5.1. ANN Configuration

A process of training and selection was followed to select the neural network with the best performance for GSM SEI. Different networks were trained by systematically increasing the number of hidden neurons while training the ANN on the training set. The test set was used to evaluate performance of the trained neural networks on previously unseen data, and the network with the least hidden neurons and best test set performance (simultaneously) was selected. By following this process, a perceptron neural network with 114 input neurons, 20 hidden neurons, and ten output neurons was obtained. The number of input neurons was determined by the number of elements in the feature vector and the number of output neurons was determined by the number of classes (i.e. the number of unique handsets).

3.3.5.2. Neuron

A similar approach to the theory discussed in section 2.6.3.2 was followed to implement a neuron in hardware, with exceptions to reduce resource requirements. A functional diagram of a neuron is displayed in Figure 3.3 and discussed.

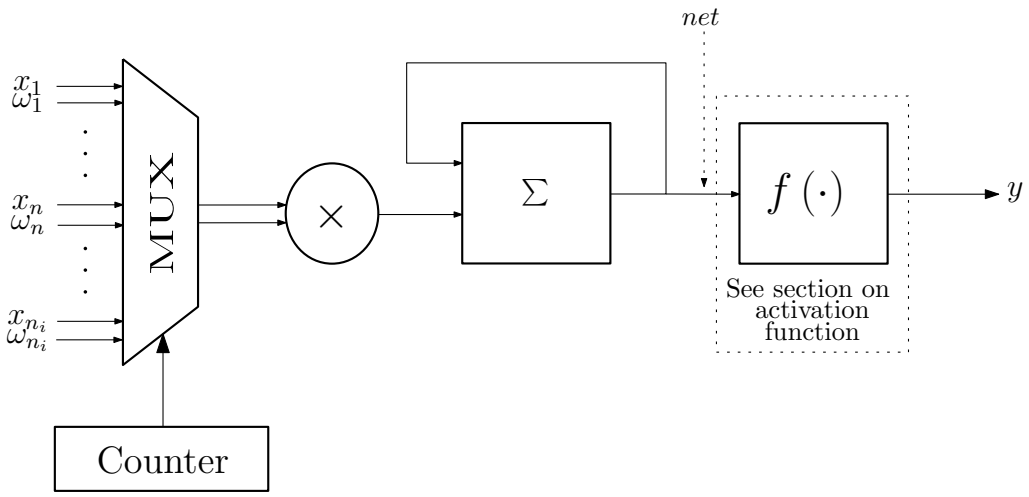


Figure 3.3.: Functional diagram of a neuron

As can be seen in Figure 3.3, the input and weight vectors are inputs to the neuron, where the output of the activation function is the output of the neuron (see Figure 2.11). A single multiplier was used to reduce multiplication operators. As mentioned in subsection 2.7.2 on the hardware architecture, the Xilinx Vertex 5 FPGA used in this study contains 640 DSP48E slices with dedicated 25×18-bit signed multipliers [96]. Since the use of multipliers in logical gates is extremely resource-intensive, it was attempted to limit the number of dedicated multipliers used. The inputs and weights were normalised to 18 bits and 25 bits respectively, which meant a single dedicated multiplier was required for multiplication of the input and weight values.

A synchronous multiplexer was used to transfer a specific input and corresponding weight to the multiplier sequentially. The specific input and weight are determined by the value of the counter, which is incremented until all required weighted inputs have been calculated. A stream of corresponding outputs of the multiplications is then received as output. A cumulative adder was implemented by using an adder, with the output iteratively being fed back to the adder. The addition is continued through the complete stream of values received from the multiplier. This results in only a single multiply and accumulate operator being required. After all the values have been added, the resulting value is used as net activation value for the activation function.

Activation Function

A mathematical function is fairly easily implemented in software, where the input parameter is fed into the function, and the calculated result is obtained at the output. Implementation of a function on an FPGA is not as trivial compared to software implementation. As the actual mathematical implementation of a continuous function, such as a sigmoid, is extremely resource-intensive in hardware, discrete methods were considered in literature. Two extremes to function approximation are::

- LUT;
- High-order polynomial approximation.

With the LUT approach, each input value requires a corresponding LUT value and usually results in an excessively large LUT being required. This results in the LUT being more memory-intensive. The other extreme of using high-order polynomial approximation of the function results in high degrees of computations being required, resulting in increased resource requirements. As a result, a high-order polynomial is computationally expensive. However, by using a hybrid of the two extreme approaches, a more efficient activation function was found, with sufficient trade-off between memory and computational requirements. A smaller LUT in conjunction with lower-order polynomials (typically linear) could be used to obtain a relatively accurate approximation of the activation function.

Piece-wise Linear Look-up Table Function Estimation

A PWL approximation of a function makes use of a sequence of finite linear functions (usually uniformly spaced) to represent each segment of the initial function. As the PWL function is an approximation of the actual function, accuracy is extremely important. One of the determining factors of the accuracy was the number of segments/intervals that were used to approximate the function. Owing to hardware restrictions such as resources and limited slices, it was decided that only a limited number of intervals be used.

Considering a function, assumptions were made to reduce the number of intervals required to approximate the function. As can be seen in Figure 2.12a, ranges exist where values tend to approach constant values (in this case 0 when input was negative, and 1 when input was positive). The number of intervals consequently reduces the resource requirements; all values in these segments could be assumed to be equal to those constant values, assuming that the resulting error was redundant. Two limits to the function are then determined, and only the section between these inputs limits was used for creation of the LUT. A visual representation is given in Figure 3.4.

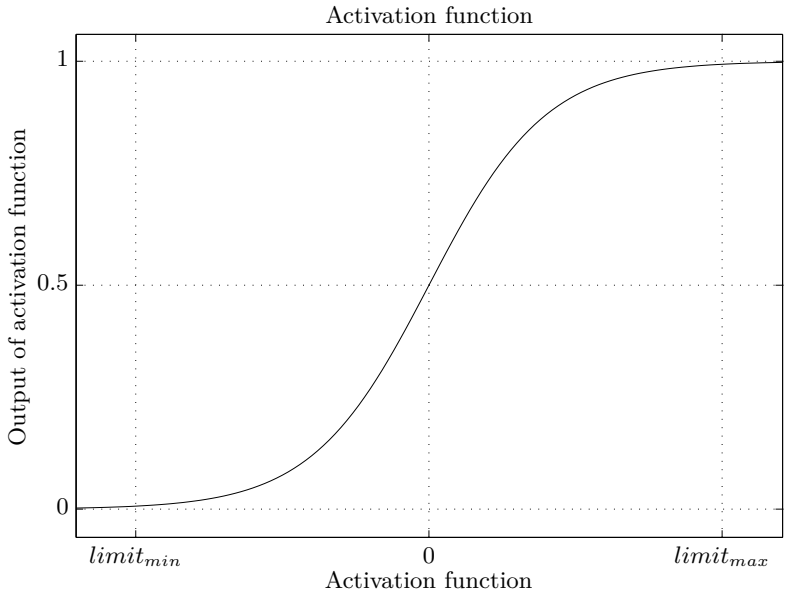


Figure 3.4.: Minimum and maximum limits of activation function

One approach to obtaining these limits between which the function is assumed as non-constant and non-linear is discussed in [97] and was used in this study. The method proposed by [97] was used to calculate the range in which the approximation needs to be made to obtain a good approximation. The limits of the range were calculated as ± 5. The creation of the LUT was based on a combination of the approaches discussed in [95, 97].

The LUT was iteratively created, where each segment between the lower and upper input limits was considered separately. The function between the limits was then sampled to obtain an appropriate number of samples (discussion to follow), and also divided into segments. For each segment, the output of the function for a number of input values was calculated by using an appropriate number of samples per interval. Linear interpolation was used to obtain a linear estimation of the function in a specific segment. The linear function parameters obtained were then stored in the LUT. For clarification, the algorithm used to create the LUT is shown in Algorithm 3.2.

Algorithm 3.2 Algorithm used to construct the LUT, used to approximate a function

Input: High and low limits of the activation function, number of intervals, actual activation function

Output: LUT containing parameters of PWL approximation of activation function

1: Initialisation: Obtain lower (i_{min}) and upper limits (i_{max}) of function that needs to be approximated
2:
3: samples per interval := $\frac{i_{max}-i_{min}}{\text{number of intervals}}$
4:
5: **for** index of segment := i_{min} to i_{max} (increment: samples per interval) **do**
6: Find values within segment
7: **for** index of samples := 1 to n **do**
8: segment samples :=append(segment values at index)
9: **end for**
10: a, b := Linear interpolation of segment samples
11: Append LUT with a, b parameters
12: **end for**

As mentioned above, the chosen number of samples in an interval greatly influences the accuracy of the estimated function (degree of influence is dependent on the complexity or order of the function). A trade-off needed to be made between the accuracy of the LUT and the size of the LUT, as the size of the LUT affected the resource requirements. Another design consideration was that the output of the adder preceding the LUT was used as the address of the LUT. The LUT was divided into segments and was structured so that the adder output could be used as the address of the LUT. The best approach was to truncate some of the least-significant bits of the adder output and use the remaining bits as the address of the LUT. It was required that the number of samples within an interval be chosen as a power of two, to construct an LUT with this type of structure.

$$n_{\text{samples per interval}} = 2^c \tag{3.3}$$

where

c a constant.

The reason for the number of intervals was to ensure that the full range of a specific address for the LUT was contained within the corresponding segment and to ensure that when the address of the LUT was incremented with a bit, the correct segment was used to calculate the output.

The advantage of the use of a PWL LUT approximation is a reduction in the amount of resources required. As the output of the LUT is the linear function parameters, only a single multiplication operation was required to calculate the output of the activation function. If the address falls outside the limits, the output of the activation function is assigned the appropriate constant. The output of the activation

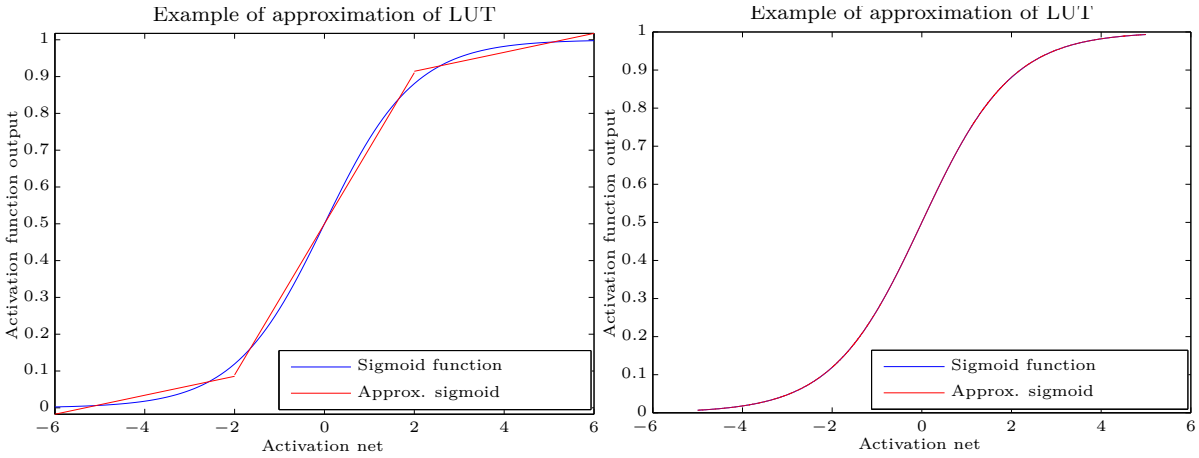
function is described in Algorithm 3.3. (Note that the ranges have been adapted for the ranges that were used during the FPGA design.)

Algorithm 3.3 Algorithm to calculate estimated activation function output

Input: Activation net value
Output: Activation function output

address :=Truncation of adder output (denoted as x)
if $address < i_{min}$ **then**
  $output := 0$
else if $address > i_{max}$ **then**
  $output := 1$
else
  $a, b = LUT(address)$
  $output := a \cdot x + b$
end if

When the address value falls within the limits of the section that needs to be approximated, a combination of the LUT and a linear function is used to calculate the output of the activation function. The operation of the LUT and PWL approximation is visually illustrated in Figure 3.5.



(a) Approximation of sigmoid with three segments **(b)** Approximation of sigmoid with 2560 segments

Figure 3.5.: Visual representation of LUT operation

An illustration of an estimation of the sigmoid function using three sections is given in Figure 3.5a. The use of three segments will not give an accurate estimation of the function. A higher number of segments was used to improve the accuracy of the estimation. Since the number of samples per interval needs to be a power of two, a value of 2^{29} samples per interval was chosen, resulting in a set of 2560 segments being required. The approximation of the sigmoid function using 2560 linear segments is shown in Figure 3.5b.

If the input to the function falls outside the chosen range, a constant value is assigned to the output. A larger range of inputs is provided in Figure 3.6, ranging from ±8.

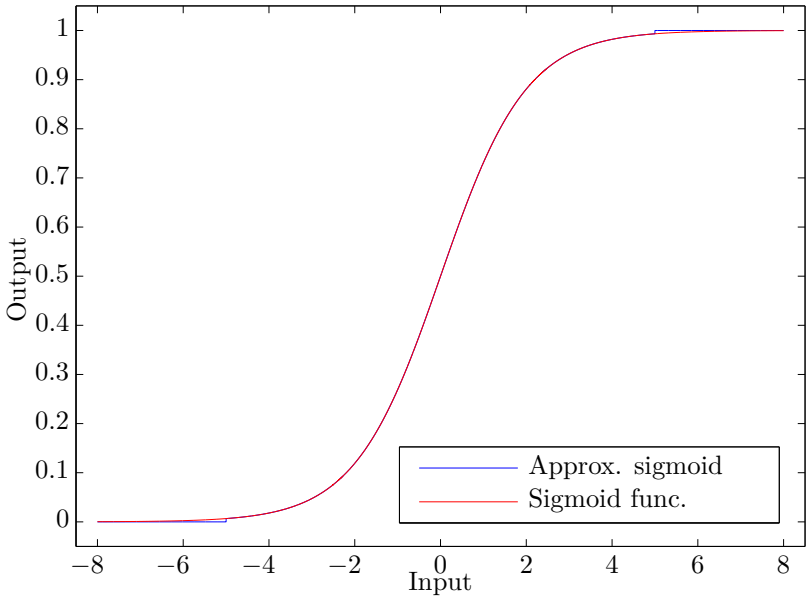


Figure 3.6.: Error of the approximation of the activation function

As can be seen in Figure 3.6, through the use of 2560 segments with PWL approximation, the original sigmoid function could be approximated quite well. As mentioned in section 3.3.5.2, the limits used during the creation of the approximation of the activation function were established as ±5. The average error between the approximated and original function within these limits is 2×10^{-8}, with a maximum error of 1.101×10^{-7}. If one looks closely in the regions of ±5 on the x-axis, a larger error can be observed. The approximation error over the range ±8 was calculated and is displayed in Figure 3.7.

As can be seen in Figure 3.7, within the limits chosen to perform the approximation, the error is relatively small. However, it can be seen that a relatively large error was found outside the chosen limits compared to the errors within the limits. The maximum error within the range ±8 was found to be 6.7×10^{-3}.

The activation function needed to be used in the FPGA design, so a scaled version of the approximated sigmoid function was required and is shown in Figure 3.8.

Since the activation function was approximated by a set of linear segments, where the LUT contained coefficients of the linear segments, the approximated value had to be calculated. The most significant bits of the input value were used as an address to the LUT, where the linear function parameters were then obtained. The equation for a straight line (Equation 3.4) was then used to calculate the approximated output of the activation function:

$$y = a\,x + b \tag{3.4}$$

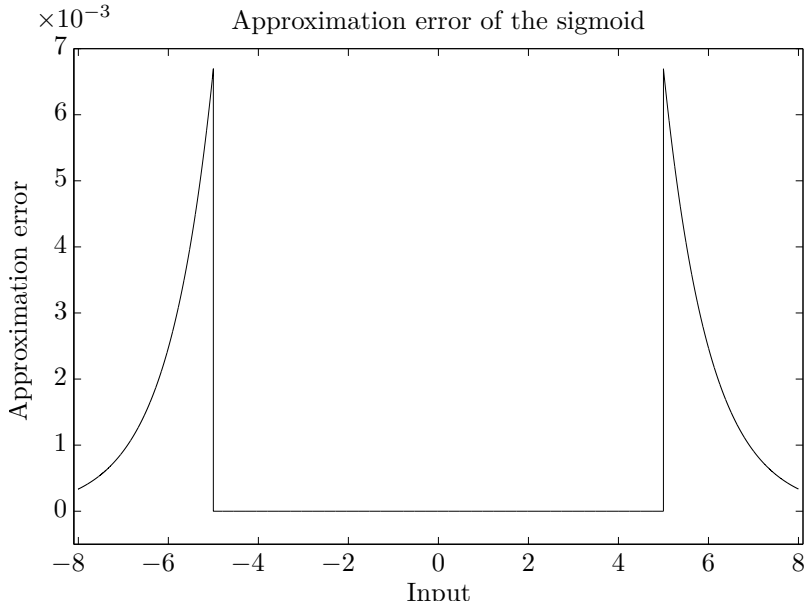


Figure 3.7.: Approximation error between sigmoid and PWL approximation

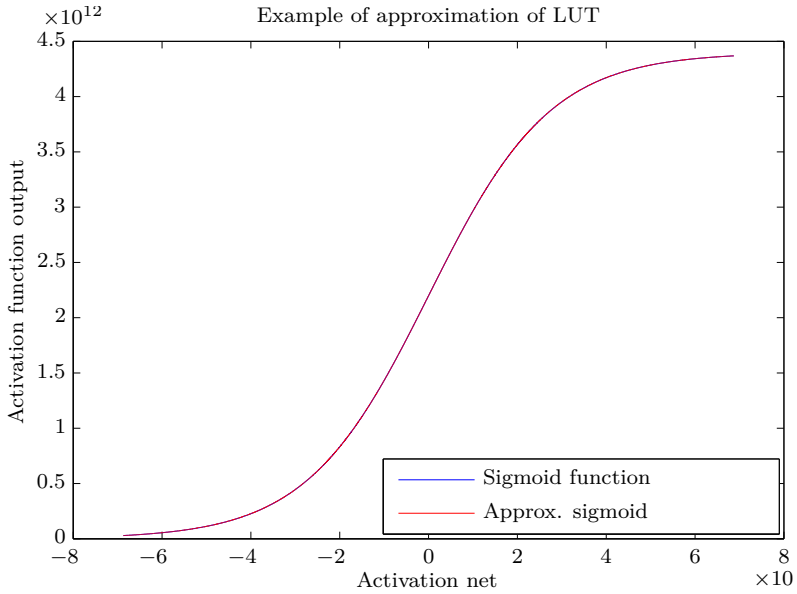


Figure 3.8.: Scaled LUT approximation of the sigmoid function

A visual illustration of the operation of the estimated activation function is shown in Figure 3.9.

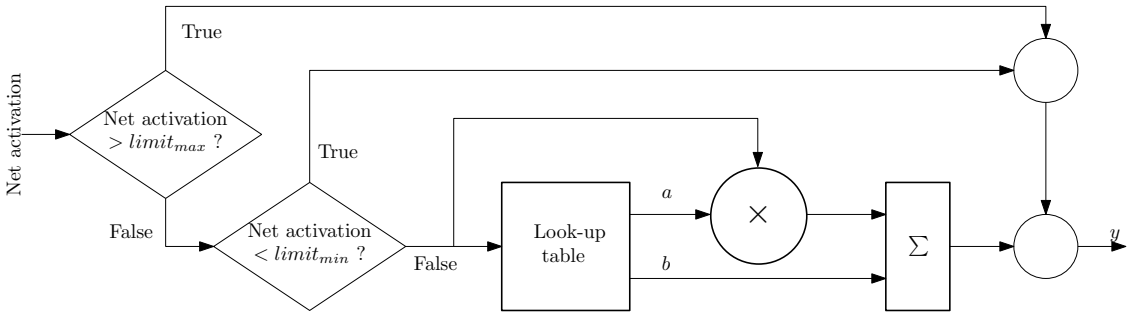


Figure 3.9.: LUT operation

From subsubsection 3.3.5.2, the inputs of the neurons and weights were normalised to 25 bit and 18 bit values, respectively. Owing to the multiplications during the calculation of the output of the neuron, the output is 43 bits long. For the output of the neuron to be propagated through to the next neuron as an input, the value again had to be normalised. The normalisation process was performed through the truncation of the least significant bits, which was in digital terms the same as division by 2^n (or shifting with n bits), where n is the number of truncated bits. The final output is equal to the remainder of the bits after truncation has been completed.

In this case, available QDR memory was incorporated for storage of the LUT values. If no QDR memory were available, a portion of available resources on the FPGA could have been used as storage of the parameters, but a restriction on the number of parameters might be required, depending on the amount of available resources.

3.3.5.3. Layers

From theory (see section 2.6.3.2), an ANN typically consists of three or more layers, each layer consisting of a set of neurons. The assumption was made that every single neuron was identical using the same activation function. Therefore, the hardware described above can also be reused for each neuron within the network. However, for each neuron only the weights are changed between the different neurons within a layer while the inputs remain constant. A multiplexer was used to feed the neuron with the corresponding weight, and a demultiplexer was used to record the correct output for each neuron. A functional diagram is shown in Figure 3.10 to illustrate this.

As can be seen in Figure 3.10, each layer also includes the biasing value that was incorporated in the network.

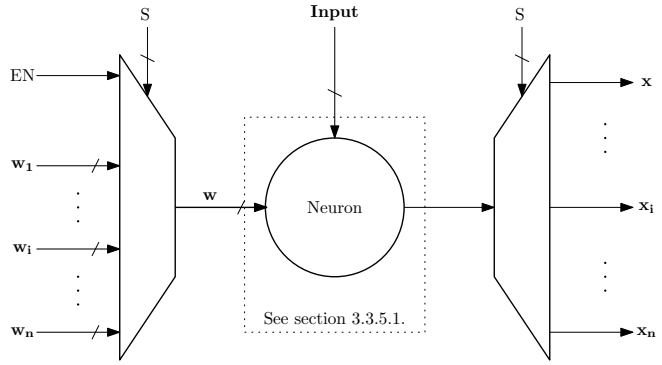


Figure 3.10.: Functional diagram of the operations within a layer in an ANN

3.3.5.4. Neural Network Implementation

Similar to the theory discussed in subsubsection 2.6.3.2, the inputs are propagated through the hidden layer and the output of the hidden layer is then fed into the input of the output layer. The output of the output layer is considered as the discriminator between classes for classification. Below, in Figure 3.11, a functional diagram of the upper-level design of the ANN is shown.

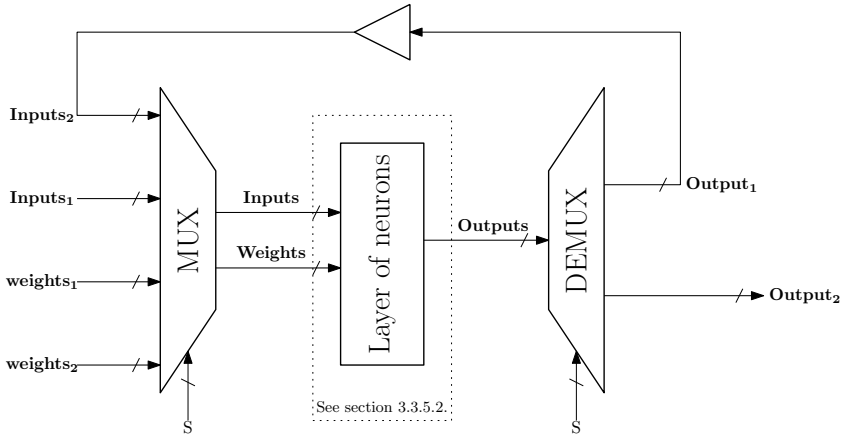


Figure 3.11.: High-level functional diagram of ANN for hardware implementation

Note that $inputs_1$ contains all the input values, where $outputs_1$ is fed back to the multiplexer as the input to the next layer of the network. Matrices $weights_1$ and $weights_2$ contain input-to-hidden and hidden-to-output weights, respectively.

As can be seen in Figure 3.11, the system was developed to increase re-usability, where hardware was reused multiple times in different stages of the network.

A flow chart of the ANN (Figure 3.12) gives an overall flow of operations within the complete ANN. In the inner loop, the calculations of each unique neuron are performed. When the calculations of all neurons in a layer have been completed, the next layer is started. This process is repeated until all calculations for all neurons and layers have been completed.

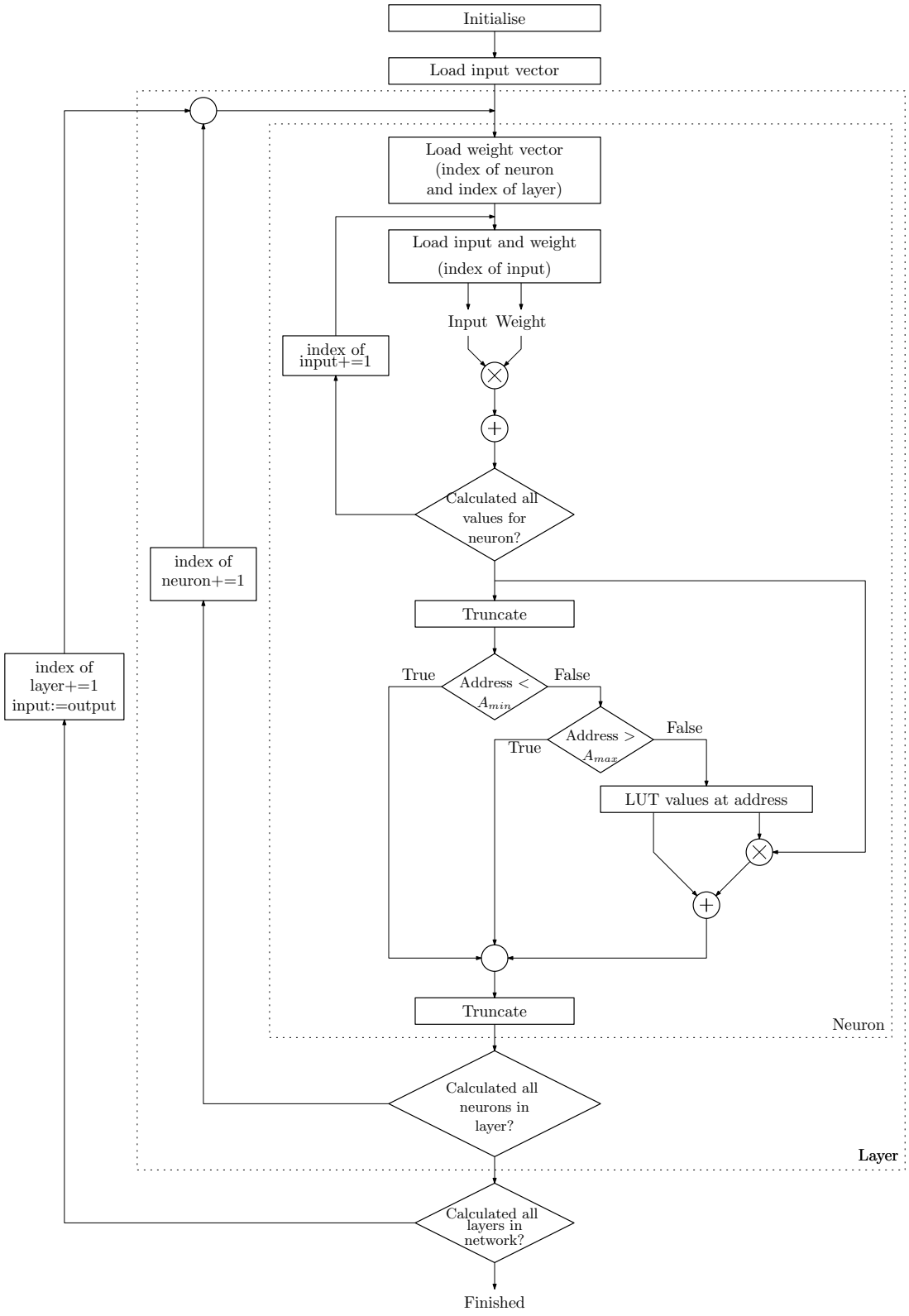


Figure 3.12.: Flow chart of ANN operation

The final ANN building block with inputs and outputs required is shown in Figure 3.13.

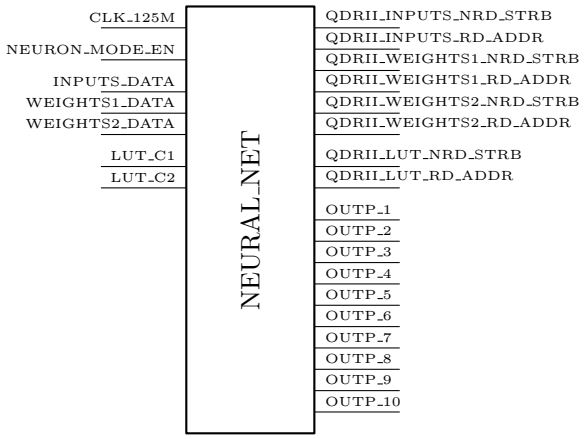


Figure 3.13.: Neural network block diagram

3.4. Debugging

Debugging is an essential part of VHDL design. Because of the concurrent nature of an FPGA, the timing of the system is crucial and can only be managed with run-time debugging.

During the design of the system, a combination of software, behavioural simulation by ModelSim and run-time analysis through Chipscope Pro was used. As it was confirmed that the software classifier operated correctly, the design was based on the software version. For each building block, the VHDL design was first verified through ModelSim to confirm the operation of the design. Only after correct operation of the design has been confirmed in ModelSim were the building blocks transferred to the hardware and tested through the Chipscope. This process was followed to reduce the effects of the hardware on the design, as well as the time-consuming nature of the synthesis process in Xilinx ISE. Another advantage of simulation in ModelSim is the ability to create a stimulus in MATLAB for the system, without any resource limitations. A visual representation of the process is given in Figure 3.14.

The Chipscope Pro setup on the FPGA is shown in Figure 3.15. ChipScope Pro integrated core, Integrated Logic Analyser (ILA) and Virtual Input/Output (VIO) were all created by Xilinx CoreGen. VIO can read/write a 32-bit word continuously, while ILA can monitor up to 256 bits. Different signals would need to be measured at different times without repeating synthesis. The output of the VIO was used as the address of a multiplexer to measure a wide variety of signals without restarting the synthesis process. This setup provides on-chip logic analyser functionality at the clock frequency for measuring a wide range of signals on the FPGA.

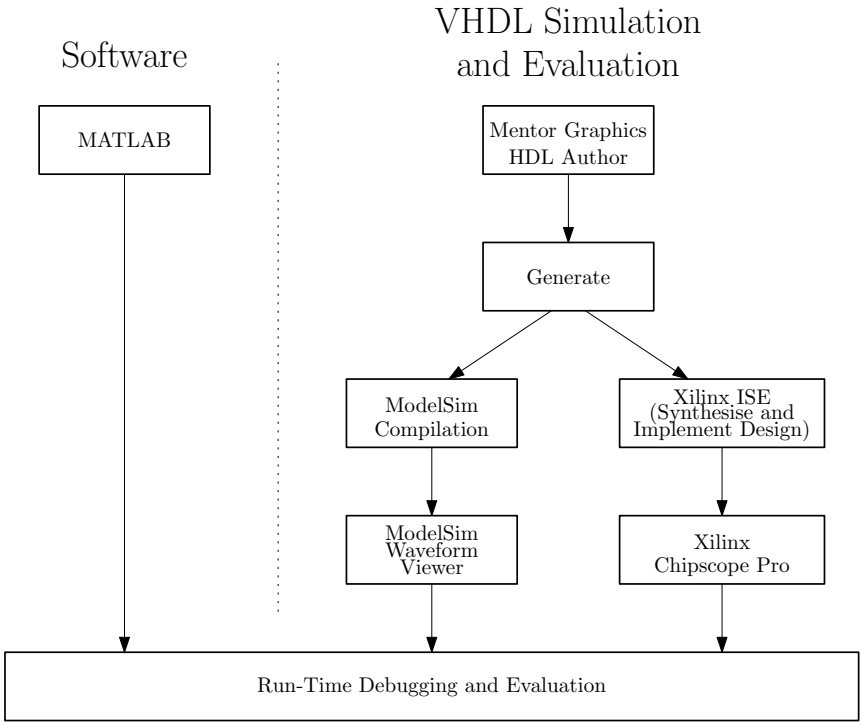


Figure 3.14.: Run-time debugging

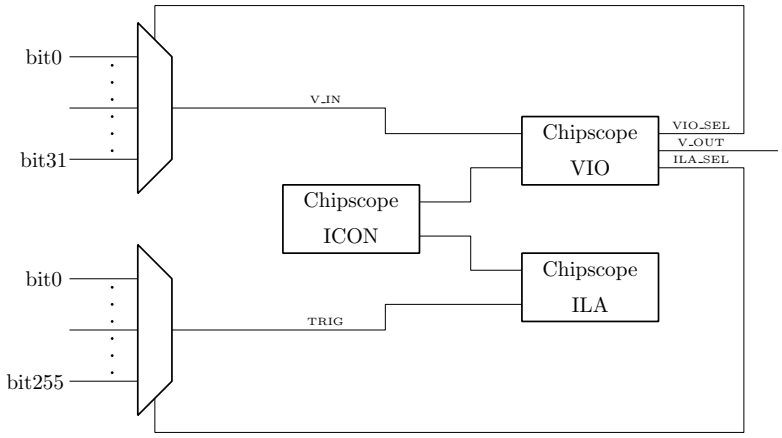


Figure 3.15.: Debugging process

3.5. Conclusion

In this chapter, the knowledge obtained from the literature study (see chapter 2) was incorporated into the design.

First, an ANN was implemented from first principles in MATLAB. Two additional versions of the software were also implemented. The first version made use of scaled values, where the second version used a PWL approximation of the sigmoid function as activation function. This was done to assist in the design and validation of the final FPGA design of the ANN. A back-propagation gradient descent algorithm was implemented in MATLAB for training of the ANN. After training is performed, the weights are transferred to QDR memory for use by the ANN on the FPGA. C++ application was used for control of the functions on the FPGA and the communications between the PC and the FPGA.

The MATLAB implementation was used as basis in the design of a VHDL implementation of an ANN. A bottom-up approach was followed during the design of the ANN. Firstly, a single neuron was completed. A PWL approximated sigmoid was used as activation function, where a LUT was created in on-board memory to store the PWL parameters. Multiplexing was used on different levels of the design to reduce resource requirements. In this study, three levels of multiplexing was used. A single multiplier was used to obtain weighted inputs, where a multiplexer was used to ensure the correct combination of inputs and weights are multiplied. Multiplexing of the inputs, weights and output of the neuron was used to create a complete layer by using only a single neuron's hardware. Similarly multiplexing was used to perform the calculations of the complete ANN. By using different levels of multiplexing, the required resources was reduced.

A combination of MATLAB, ModelSim and ChipScope was used to assist in the debugging of the VHDL implementation.

The software and hardware implementations of the ANN were evaluated, and the results are discussed in the following chapter.

4. Results

4.1. Introduction

In this chapter performance results of the proposed classifier and system are provided and discussed. Results obtained with software and FPGA implementations are also compared and discussed. Following the design cycle of the system, results for different phases of the system are compared. As the software version of the neural network was first implemented as the basis of the design of the VHDL system, results of the software implementation (discussed in section 3.2) are presented as an indication of performance. To assist in the FGPA design of the system, a scaled version of the neural network, as well as a version incorporating the LUT PWL approximation of the activation function, was implemented in software. The results obtained for these software versions are also presented. Results of the scaled and LUT versions of the neural network provide an indication of the influence of the identification performance caused by these adaptations to the system. Thereafter identification accuracy of the FPGA implementation is presented and discussed.

4.2. Dataset

The dataset has an immense impact on the performance and quality of the system. The partial dataset used in this study was obtained from [1]. The composition of the dataset is provided in Table 4.1. Samples from ten unique handsets were provided.

As can be seen, the number of samples per handset varies to a large extent. As discussed in subsubsection 2.6.2.7, the average of multiple bursts over time has proven to be more accurate that individual bursts. According to [1], 30 is an appropriate number of bursts to calculate the average. For the creation of the actual features, the average of 30 bursts was calculated and added to the dataset.

IMEI	Handset	# of Samples	# of Features
3522340114588538	Motorola C118 #1	1808	60
3522340139448767	Motorola C118 #2	15896	529
3576790048382162	Motorola C118 #3	8164	272
3583170138857775	Motorola C123 #1	5066	168
3583170119139375	Motorola C123 #2	2353	78
3567610006655101	Medion SP 1000	1919	63
3552150089128130	Motorola C115	7941	264
3585160261427242	Blackberry 8310	2274	75
3582560038163709	Motorola C139	12083	402
3593300285264837	Nokia E51	3281	109

Table 4.1.: Composition of the dataset received from [1]

Since the number of features varies between different emitters, the dataset is unbalanced. When an unbalanced dataset is used for training a classifier, an a priori probability for each emitter is assumed. This assumption is made as an emitter with more feature vectors would be trained more regularly, compared to other handsets. However, during operation there is no certainty as to the a-priori probability of any emitter transmitting being higher than any other. Thus, the a priori probability between the different handsets was set to be equal (and thus also balanced).

To ensure that training is performed with equal a priori probabilities of handsets, the classifier would need to be trained the same number of times for each handset. This was performed by creating copies of entries in the dataset where a limited number of features were available. By creating duplicates, an equal number of training samples was made available for each handset. This approach has the advantage of creating an equally sized dataset for training and testing, providing a less biased classifier.

The disadvantage of this approach is training and testing of the classifier with duplicate samples. Adding duplicates to the dataset might result in an over-trained classifier for the handsets with a limited number of data, possibly reducing generalisation of the classifier. As duplicates in the testing set were created, results are also affected. For instance, when a sample is classified correctly, multiple positive classifications will exist for the same testing sample. This problem can be avoided by using a larger dataset, but since collection of this data uses specialized equipment and is very costly, the data from [1] had to suffice.

4.3. Overview

This study is based on previous research [1], where SEI in the GSM domain was performed in software. In this study the chosen classifier was an ANN and it was

designed to perform SEI, first in software (for verification purposes) and finally on an FPGA (the primary objective of this study).

The results are divided into four stages of the design of the ANN.

Stage one comprised of the implementation of a GSM SEI in software using an ANN. The results on the first stage is presented in subsubsection 4.4.1.1.

In stage two the GSM SEI was implemented in software by using scaling of all the values in the network (as discussed in subsubsection 3.3.4.1) to show that an FPGA implementation of the SEI will work in the last stage. The results obtained from software implementation of the ANN with scaling of values are presented in subsubsection 4.4.1.2.

The third stage included the use of PWL LUT in software to show that such an approximation will work in the context of this research. The PWL LUT activation function was implemented in software, as discussed in section 3.3.5.2. Results obtained after incorporating the PWL LUT activation function in software are provided in subsubsection 4.4.1.3.

In the last stage, a full FPGA implementation was done of the GSM SEI using an ANN, scaling, and an LUT PWL approximation of the sigmoidal activation function. Results of the FPGA version of the identifier are provided in subsection 4.4.2.

4.4. Identification Accuracy Results

This section gives results of the different design stages and provides verification of the functional capability of each of the functional elements in the system.

Firstly, results of the software implementation of the ANN are provided. The results of the software implementation provide confirmation of functional capability. Operation of a software implementation of an ANN in software performing SEI for GSM handsets was defined as a secondary criterion. The software implementation was used as the basis for the design of the VHDL version of the ANN, hence classification accuracy was also used as performance metric of the FPGA implementation.

Corresponding results for the VHDL version of the ANN are also provided.

For classification accuracy a confusion matrix is provided. Confusion matrices provide an overall view of performance as well as a more in-depth view of individual identification performances. Confusion matrices were iteratively created, where each input is provided to the ANN. The obtained results were then compared to the true result. The confusion matrices provide a comparison of the handset that was emitted and the handset as which it was identified. Throughout this study the primary metric is the true acceptance ratio (TAR), which is the percentage of correct predictions of identity compared to the actual identity. TAR provides the true positive classification rate.

A comparison between the results of [1] and the proposed systems results is provided for validation purposes.

4.4.1. Software Implementation Results

In this section identification results from the software implementation are provided.

Specific stages were designed and implemented during the design process, so the results of each stage are included to illustrate the effects of each stage on classification performance. Firstly, results obtained by a traditional ANN is provided, followed the an ANN including scaling of all values in the system. The results obtained by including an LUT PWL activation function are then provided.

4.4.1.1. Traditional ANN Results

The confusion matrix of the results obtained from the first stage of the software implementation is provided in Table 4.2. It can be seen in Table 4.3 that all handsets except three were perfectly identified. Table 4.2 also reflects that errors that occurred were a mixture of three emitters, that is:

- Motorola C123 #2;
- Medion SP 1000;
- Blackberry 8310.

It can be seen that the Blackberry 8310 achieved the lowest classification of all handsets, with a TAR of 92 %. It can be seen from Table 4.2 that the classifier had minor difficulty in instances distinguishing between the Blackberry 8310 and Medion SP 1000 handsets. Also, two incorrect identifications of one of the Motorola C123 handsets were made as the Blackberry 8310. Overall identification accuracy of handsets for the software implementation was 98.8 %.

		Identified as:									
		Motorola C118 #1	Motorola C118 #2	Motorola C118 #3	Motorola C123 #1	Motorola C123 #2	Medion SP 1000	Motorola C115	Blackberry 8310	Motorola C139	Nokia E51
Actual Phone:	Motorola C118 #1	150	0	0	0	0	0	0	0	0	0
	Motorola C118 #2	0	150	0	0	0	0	0	0	0	0
	Motorola C118 #3	0	0	150	0	0	0	0	0	0	0
	Motorola C123 #1	0	0	0	150	0	0	0	0	0	0
	Motorola C123 #2	0	0	0	0	148	0	0	2	0	0
	Medion SP 1000	0	0	0	0	0	146	0	4	0	0
	Motorola C115	0	0	0	0	0	0	150	0	0	0
	Blackberry 8310	0	0	0	0	0	12	0	138	0	0
	Motorola C139	0	0	0	0	0	0	0	0	150	0
	Nokia E51	0	0	0	0	0	0	0	0	0	150

Table 4.2.: Confusion matrix of the ANN in software

	Accuracy per handset:
Motorola C118 #1	100 %
Motorola C118 #2	100 %
Motorola C118 #3	100 %
Motorola C123 #1	100 %
Motorola C123 #2	98.7 %
Medion SP 1000	97.3 %
Motorola C115	100 %
Blackberry 8310	92 %
Motorola C139	100 %
Nokia E51	100 %
Accuracy	98.8 %

Table 4.3.: Accuracy of ANN in software

4.4.1.2. Software ANN Results with Scaling

The confusion matrix, identification accuracies for each handset and overall identification accuracy for the scaled version of the software implementation are provided in Table 4.4 and Table 4.5. By comparing Table 4.2 and Table 4.4, it can be seen

that the same performance was achieved by the traditional ANN compared to the scaled version.

		Identified as:									
		Motorola C118 #1	Motorola C118 #2	Motorola C118 #3	Motorola C123 #1	Motorola C123 #2	Medion SP 1000	Motorola C115	Blackberry 8310	Motorola C139	Nokia E51
Actual Phone:	Motorola C118 #1	150	0	0	0	0	0	0	0	0	0
	Motorola C118 #2	0	150	0	0	0	0	0	0	0	0
	Motorola C118 #3	0	0	150	0	0	0	0	0	0	0
	Motorola C123 #1	0	0	0	150	0	0	0	0	0	0
	Motorola C123 #2	0	0	0	0	148	0	0	2	0	0
	Medion SP 1000	0	0	0	0	0	146	0	4	0	0
	Motorola C115	0	0	0	0	0	0	150	0	0	0
	Blackberry 8310	0	0	0	0	0	12	0	138	0	0
	Motorola C139	0	0	0	0	0	0	0	0	150	0
	Nokia E51	0	0	0	0	0	0	0	0	0	150

Table 4.4.: Confusion matrix of the ANN in software, with scaled values

	Accuracy per handset:
Motorola C118 #1	100 %
Motorola C118 #2	100 %
Motorola C118 #3	100 %
Motorola C123 #1	100 %
Motorola C123 #2	98.7 %
Medion SP 1000	97.3 %
Motorola C115	100 %
Blackberry 8310	92 %
Motorola C139	100 %
Nokia E51	100 %
Accuracy	98.8 %

Table 4.5.: Accuracy of software ANN, scaled

4.4.1.3. Software ANN Results with LUT PWL Activation Function

The results obtained from the third stage of the software implementation, where a LUT PWL activation function was used, is presented in Table 4.6 and Table 4.7. As

can be seen, overall identification TAR reduced slightly owing to the PWL LUT activation function being used. Handsets that were occasionally identified incorrectly were:

- Motorola C118 #1;
- Motorola C123 #2;
- Medion SP 1000;
- Blackberry 8310.

When referring to Table 4.1, it can be seen from the composition of the dataset that the above-mentioned handsets had a limited number of samples to use for training and testing. This could have played a role in identification performance, as these handsets might have been over-fitted during training. Possibly this could be improved by increasing the size of the dataset and future research can be performed on a larger dataset.

		Identified as:									
		Motorola C118 #1	Motorola C118 #2	Motorola C118 #3	Motorola C123 #1	Motorola C123 #2	Medion SP 1000	Motorola C115	Blackberry 8310	Motorola C139	Nokia E51
Actual Phone:	Motorola C118 #1	140	0	0	0	10	0	0	0	0	0
	Motorola C118 #2	0	150	0	0	0	0	0	0	0	0
	Motorola C118 #3	0	0	150	0	0	0	0	0	0	0
	Motorola C123 #1	0	0	0	150	0	0	0	0	0	0
	Motorola C123 #2	0	0	0	0	148	0	0	2	0	0
	Medion SP 1000	0	0	0	0	0	146	0	4	0	0
	Motorola C115	0	0	0	0	0	0	150	0	0	0
	Blackberry 8310	0	0	0	2	2	12	0	134	0	0
	Motorola C139	0	0	0	0	0	0	0	0	150	0
	Nokia E51	0	0	0	0	0	0	0	0	0	150

Table 4.6.: Confusion matrix of the ANN in software, incorporating the LUT PWL activation function

	Accuracy per handset:
Motorola C118 #1	93.3 %
Motorola C118 #2	100 %
Motorola C118 #3	95.3 %
Motorola C123 #1	100 %
Motorola C123 #2	98.7 %
Medion SP 1000	97.3 %
Motorola C115	100 %
Blackberry 8310	89.3 %
Motorola C139	100 %
Nokia E51	100 %
Accuracy	97.4 %

Table 4.7.: Accuracy of software ANN, with an incorporated LUT PWL activation function

4.4.2. Field-programmable Gate Array Artificial Neural Network Results

The results of the FPGA implementation are provided in Table 4.8 and Table 4.9.

Once again, the Blackberry 8310 was the handset most frequently identified incorrectly, with a TAR of 85.3 %, with a very slight decrease in performance of some of the other handsets, such as the Blackberry 8310, Motorola C139, Motorola C123 #2, Medion SP 1000. A slight increase in identification performance was seen on the Motorola C123 #3. This increase in performance was probably caused by rounding errors, resulting in a correct classification, while it was classified incorrectly after the PWL LUT was incorporated. An overall identification TAR of 97.0 % was achieved with the FPGA implementation.

		Identified as:									
		Motorola C118 #1	Motorola C118 #2	Motorola C118 #3	Motorola C123 #1	Motorola C123 #2	Medion SP 1000	Motorola C115	Blackberry 8310	Motorola C139	Nokia E51
Actual Phone:	Motorola C118 #1	140	0	0	0	10	0	0	0	0	0
	Motorola C118 #2	0	150	0	0	0	0	0	0	0	0
	Motorola C118 #3	4	0	146	0	0	0	0	0	0	0
	Motorola C123 #1	0	0	0	150	0	0	0	0	0	0
	Motorola C123 #2	0	1	0	0	147	0	0	2	0	0
	Medion SP 1000	0	0	0	0	2	144	0	4	0	0
	Motorola C115	0	0	0	0	0	0	150	0	0	0
	Blackberry 8310	0	0	0	4	6	12	0	128	0	0
	Motorola C139	0	0	1	0	0	0	0	0	149	0
	Nokia E51	0	0	0	0	0	0	0	0	0	150

Table 4.8.: Confusion matrix of the ANN on the FPGA

	Accuracy per handset:
Motorola C118 #1	93.3 %
Motorola C118 #2	100 %
Motorola C118 #3	97.3 %
Motorola C123 #1	100 %
Motorola C123 #2	98 %
Medion SP 1000	96 %
Motorola C115	100 %
Blackberry 8310	85.3 %
Motorola C139	99.3 %
Nokia E51	100 %
Accuracy	97.0 %

Table 4.9.: Accuracy of ANN on the FPGA

4.4.3. Comparison of Results at Different Implementation Stages

An comparison of the identification accuracy of each of the different stages of implementation of the system is provided in Table 4.10.

	Accuracy per handset:			
	Software	Software Scaled	Software LUT	FPGA
Motorola C118 #1	100 %	100 %	93.3 %	93.3 %
Motorola C118 #2	100 %	100 %	100 %	100 %
Motorola C118 #3	100 %	100 %	95.3 %	97.3 %
Motorola C123 #1	100 %	100 %	100 %	100 %
Motorola C123 #2	98.7 %	98.7 %	98.7 %	98 %
Medion SP 1000	97.3 %	97.3 %	97.3 %	96 %
Motorola C115	100 %	100 %	100 %	100 %
Blackberry 8310	92 %	92 %	89.3 %	85.3 %
Motorola C139	100 %	100 %	100 %	99.3 %
Nokia E51	100 %	100 %	100 %	100 %
Accuracy	98.8 %	98.8 %	97.4 %	97.0 %

Table 4.10.: Comparison of accuracy of all platforms for each handset

As can be seen in Table 4.10, a slight decrease in performance was obtained by making use of the PWL LUT activation function (TAR decreased by 1.4 %). Moreover, the transition of the implementation to the FPGA resulted in a decrease of the TAR by 0.4 %.

4.4.4. Comparison with Previous Research

The results obtained in this study need to be compared to existing results to determine how well the system performs. As a partial dataset was obtained by [1], a comparison between the results can be made. A comparison of the results is provided in Table 4.11.

	Results by [1]	Software Results	FPGA Results
Accuracy	96.7 %	98.8 %	97.0 %

Table 4.11.: Accuracy of the system compared to previous research

The comparison of the results provided in Table 4.11 compare the best classification result obtained by [1], the TAR of the software and FPGA implementations achieved during this study. It can be seen that an improvement of 2.1 % was obtained in software while a marginal improvement of 0.3 % was seen on the FPGA. This is not a completely accurate comparison, as variations of the dataset used for their study [1] and the dataset used during this study exist, as illustrated in Table 4.12.

Handset	Dataset in [1]	Dataset for this study
Motorola C118	4	3
Motorola C123	1	2
Motorola C140	1	1
Medion SP 1000		1
Motorola C115	1	1
Blackberry 8310		1
Motorola C139	1	1
Sony Ericsson J100i	1	
HTC TyTNII	1	
Palm Pre	1	
Nokia E51	1	1
Nokia 6100	1	
Number of Handsets	13	10

Table 4.12.: Comparison of composition of datasets used in [1] and this study

The differences in composition of the datasets used in this study compared to [1] can be seen in Table 4.12. Firstly, in [1] 13 handsets were used, compared to 10 in this study. As illustrated throughout this chapter, the emitter most frequently identified incorrectly was the Blackberry 8310. The Blackberry 8310 was not included in their study. Also, their classifier's most frequently incorrectly identified handset is a Motorola C123. It is uncertain whether it was one of the Motorola C123 handsets that were included in the partial dataset. Furthermore, identification errors were made on the Nokia E51 and Nokia 6100 in their study. These handsets were not included in the partial dataset, and no accurate direct comparison between the identification processes could be made to determine the superior algorithm.

In summary, it is safe to state that the results obtained in this study compare favourably with results obtained in a different study. The difference between performances is small, with the result that the performance results from this study are considered to have been validated.

4.5. Critical Assessment

Good results were obtained in this study from the software as well as the FPGA implementations of the SEI system for GSM handsets. In this section, a critical assessment of the results is provided.

The incorporation of the PWL approximation of the activation function caused a decrease of 1.4 % in identification accuracy. As mentioned in section 3.3.5.2, the maximum approximation error in the range ± 8 was found to be 6.7×10^{-3}, while the maximum approximation error of 1.101×10^{-7} was found in the range ± 5. With

20 neurons in the hidden layer and ten neurons in the output layer, any errors caused in the hidden layer can be amplified during propagation through the output layer. Because the large amount of memory that was available in this study, the approximation was performed using 2560 segments.

The increase in the approximation error outside the limits of the PWL approximation can be ascribed to a decrease in identification accuracy with the implementation using the PWL LUT activation function. Because of the number of segments used, the chosen range for which the PWL approximation was calculated could have been selected more effectively, which would have resulted in a more accurate approximation of the function. Because of the large number of segments used, a very good approximation of the function was found within the range specified in this work ($-5 \leq net\,activation \leq 5$). The limits could, however, have been increased over a wider range, which would have resulted in a smaller average approximation error over a larger range. A reduced approximation error would have resulted in an increase in identification accuracy, resulting in an accuracy similar to the original software implementation.

Training was performed by using the analogue version of the activation function. However, the actual identification is performed using the PWL approximation of the activation function. An increase in accuracy would have resulted, had the training been performed with the PWL approximation.

The decrease of 0.4 % in accuracy in the transition from the software implementation to the FPGA implementation was caused by the limited number of bits to represent the values in the system. All values are digitally represented by a number of bits. However, as values were represented by a limited number of bits, the precision is reduced and the remaining bits were lost. A large number of operations are performed during a single iteration of the identification process. To perform identification of a specific handset, the ANN is provided with 115 input features. These inputs are propagated to 20 hidden neurons, and then further propagated to ten output neurons. This results in 2500 multiplications of inputs with weights and summation of those values. The multiplication and addition during the activation function operation for each of the 30 neurons in the network are also performed. Rounding and truncation occur at multiple operations within the ANN, and the effect of each is propagated through the complete network. Throughout the complete network, values are represented by 18, 25 and 43 bits on the FPGA. However, MATLAB by default makes use of double-precision floating points, i.e. 64-bit floating points. As a result of the smaller number of bits used on the FPGA to represent values compared to MATLAB, the precision was reduced and had a negative effect on the accuracy of the identification.

Overall, the differences between software and FPGA implementations are ascribed to approximation and representation errors. An approximation of the activation function for practical reasons and the representation of values in the FPGA. These errors were small but can be reduced in future implementations by considering re-

commendations from the discussion above.

4.6. Conclusion

In this chapter results obtained throughout this study are provided and discussed. First, an overview of the dataset was provided, as it had an effect on the final result.

A secondary objective was to obtain another method of performing SEI of GSM handsets. Therefore, an ANN was selected as a different type of classifier. It was shown that good identification results could be obtained using an ANN. In addition, as the FPGA implementation was based on the software implementation of the ANN, different stages in the software implementation were implemented to verify the FPGA implementation in total, showing that each functional block performed its function as intended.

The complete FPGA implementation of the ANN was tested and results were obtained. It was shown that identification of GSM handsets could be performed on an FPGA while obtaining performance results similar to existing research in software. Thus, the favourable performance results of an ANN implemented on an FPGA to perform SEI for GSM handsets validate the results of this study.

Finally, results obtained in this study were compared to the results obtained by [1], who provided the dataset on which this study was based.

The results also compared favourably with the results from [1], which shows that the primary goal of this research had been achieved, namely the successful implementation of a GSM SEI on an FPGA platform.

5. Conclusion

5.1. Overview

Successful GSM SEI systems have been implemented in software as was confirmed by a literature study. In existing research, SEI was shown to be effective for GSM handsets [1, 20] by using k-NN and SVM algorithms. However, no previous literature could be found on the implementation of SEI on GSM handsets on an FPGA. Therefore, the research challenge was to show that a GSM SEI can be implemented on an FPGA platform. In order to address the research problem, it was necessary to conduct a literature study on GSM SEI, pattern recognition systems (including feature extraction and classification), and FPGA-related topics. The final objective was to implement a functionally capable GSM SEI system on an FPGA platform that matches performance of validated systems from literature.

The objectives of this study are shown in Table 5.1. It is also shown whether the objectives were achieved, as well as the performance obtained (where applicable).

Objectives	Achieved?	Performance
Literature study: Feature extraction, classification, multiplier reduction	✓	
Analysis of possible solutions for SEI	✓	
Determine the most feasible solutions to SEI for GSM, considering the future implementation on an FPGA	✓	
Theoretical investigation of SEI system	✓	
Design and synthesis of a software-based SEI system	✓	
Operational software SEI for GSM handsets	✓	98.6 %
Simulation of hardware-based SEI system	✓	
Design and synthesis of a hardware-based SEI system	✓	
Assessment and evaluation of synthesised systems	✓	
Synthesis and evaluation of a physical-layer identification system for GSM handsets on an FPGA.	✓	97.0 %

Table 5.1.: Primary and secondary objectives

5.2. Discussion on Work Completed

An overview on the work completed in this study is provided below.

5.2.1. Literature Study

A literature study was conducted to find and analyse existing literature on a feature extraction and classification algorithms. The feature extraction methods that were investigated can be divided into three broad categories, namely domain-related (time-domain, phase-domain, TFRs), modulation error (ME, PE, EV, EVM) and non-linear effects of the emitter. Classifiers can be divided into two broad categories, namely parametric and non-parametric methods. Because of the complexity of the patterns that must be classified for SEI problems, most publications refer to using non-parametric classifiers. Non-parametric classifiers that were investigates are k-NNs, ANNs, SVMs and SNNs. As the algorithm had to be implemented on an FPGA, available literature was searched for FPGA implementations of classifiers, feature extractors, and SEI. It was expected that multipliers implemented in combinational logic would be problematic as they are resource-intensive. Several different solutions to FPGA-related implementations were investigated, such as parallel/serial implementations, multiplexing of components, multiplication through shift-and-add operations, LNS, DA and reconfiguration of FPGAs, to reduce resource requirements.

5.2.2. Data Processing

The data used in this study were obtained from the Technical University of Dresden [1], which included data and processed information from ten unique GSM handsets. Previous research [1, 20] shown that the use of phase modulation error as feature set was most effective for identification of GSM handsets. Thus, phase modulation error was also used as a feature in this study.

A dataset consisting of feature vectors had to be created. All PE data were obtained from the validated data received from [1]. The PE bursts needed to be aligned and normalised. Variances of the features were reduced by averaging 30 unique bursts to create a single feature set. This process was repeated to create the complete dataset.

5.2.3. Software Implementation

The design phase consisted firstly of the implementation of an ANN in MATLAB. The ANN was designed from first principles, as the different steps in the design were required during debugging of the hardware implementation.

As an operational ANN is required during the training process, the implementation of the training of the ANN could only be done once the implementation of the ANN in software had been completed. The training algorithm was the back-propagation gradient descent algorithm, designed from first principles. Both the ANN and the training algorithm needed to be operational before training could be performed as the training algorithm made use of the ANN.

The software ANN had to be adapted for use as the basis for the implementation of the ANN on an FPGA. This adaptation was done by incorporating scaling as well a PWL approximation of the activation function, using an LUT in the software implementation. In doing so, implementation on FPGA hardware was in effect simulated to serve as verification when results from the FPGA implementation are evaluated.

A C++ application was used to control the complete system, including initialisation of the system, communications between the PC and FPGA, memory access, memory tuning, mode selection, control of the operation on the FPGA, display and extraction of results from the FPGA.

Because of the nature of the weight initialisation during the training process, the possibility existed that a more accurate set of weights could be obtained for the ANN, which would improve identification performance. The training process was repeated in MATLAB to obtain the best possible weight values for the specific parameters, making use of 20 hidden neurons. The weights obtained through training were written to text files for simple importing to the FPGA and QDR memory storage.

5.2.4. Hardware Implementation

The software implementation of the ANN was used as the basis for the design of the ANN on the FPGA. The ANN was implemented in VHDL. The complete synthesis process was then completed, and the programming file was used to program the FPGA.

Multiplexing was incorporated in the design of the system to reduce the number of resources used on the FPGA. The implementation of the ANN in VHDL started with the implementation of a single neuron, followed by the implementation of a single layer of neurons, and finally the complete ANN. Within the neuron multipliers were also multiplexed in order to reduce multiplier requirements.

The hardware implementation of the activation function was also designed, making use of QDR memory for storage of the parameters. The parameters stored in memory were used for calculating the estimate of the output of the activation function, required within each neuron.

A controller was implemented to ensure correct operation and flow of data between different sub-units. A memory controller to read/write to the QDR memory was also required as memory was used as storage for different type of data required

by the ANN. Data could be loaded to memory by using the C++ application. Communication between the FPGA and PC had to be used, for which UDP was used.

Throughout the design of the ANN function on the FPGA, a combination of ModelSim, Chipscope and MATLAB was used for debugging and verification of the system.

5.3. Results

Satisfactory results were obtained from the VHDL implementation in this study. The final identification performance obtained was a TAR of 98.8 % for the software implementation and 97.0 % for the FPGA implementation. The reduction in performance was ascribed to small errors caused by the PWL LUT activation function, as well as rounding errors and number representation errors due to limitations on the FPGA platform. Performance of [1], from whom the data were obtained, also using PE as features and making use of SVM for identification, achieved a TAR of 96.67 % accuracy. The results obtained in this study are slightly better compared to the results obtained in previous research performed in software [1]. However, because of the variation in the data obtained compared to the data used in their study, the results cannot be compared on a direct basis. Nonetheless, results from this research are comparing favourably with published results.

5.4. Recommended Future Actions and Research

Good results were obtained for the FPGA implementation of a SEI system for GSM handsets. However, future actions and research can be used to make the system more applicable or to optimise the results further. The following opportunities are offered:

- Applying this system in a real-world situation:
  - Include interleaving of signals;
  - Design an RF system for capturing emissions;
  - Implement feature extraction on FPGA;
  - Include on-line training of the classifier for unknown emitters.
- Making use of a larger dataset to avoid over-training of the classifier, and increasing identification performance;
- Adding optimal feature selection, which may increase performance slightly;
- Increasing the range of approximation of the activation function, which can improve identification results.

5.5. Concluding Remarks

In this study it was shown that phase modulation errors can be used as input to an ANN for SEI purposes for the identification of GSM handsets. This was confirmed through the implementation of an ANN in MATLAB. It was also shown that the same system could be deployed on an FPGA. This was done through the use of multiplexing of hardware, scaling of values and the use of a PWL activation function approximation through the use of a LUT. The final conclusion is thus that SEI for GSM can be performed on an FPGA platform by using phase modulation (and its associated errors) as features and an ANN as classifier.

Bibliography

[1] J. Hasse, T. Gloe, and M. Beck, "Forensic Identification of GSM Mobile Phones," in *The 2nd Information Hiding and Multimedia Security Workshop*. Technical University of Dresden, Jun. 2013.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2001.

[3] J. P. Hattingh, "Specific Emitter Identification through Analysis of Radiated Emissions," Tech. Rep., 2011.

[4] K. Parnell and R. Bryner, "Comparing and Contrasting FPGA and Microprocessor System Design and Development," Tech. Rep., 2004.

[5] ETSI, "Digital Cellular Telecommunications System (Phase 2+); Physical Layer on the Radio Path; General Description," Tech. Rep., 1996.

[6] Pew Research Center, "Device Ownership over Time," 2014. [Online]. Available: http://www.pewinternet.org/data-trend/mobile/device-ownership/

[7] ETSI, "Digital Cellular Telecommunications System (Phase 2+); International Mobile Station Equipment Identities (IMEI) (GSM 02.16 version 5.2.0 Release 1996)," Tech. Rep., 1996.

[8] K. Talbot, P. Duley, and M. H. Hyatt, "Specific Emitter Identification and Verification," *Tecnology Review Journal*, vol. 11, no. 1, pp. 113–133, 2003.

[9] Z. Yu, C. Chen, W. Jin, and G. Zhang, "Feature Extraction of Radar Emitter Harmonic Power Contraint Based on Nonlinear Characters of the Amplifier," in *2nd International Congress on Image and Signal Processing*, no. 1. Tianjin: IEEE, Oct. 2009, pp. 1–4.

[10] A. Kawalec and R. Owczarek, "Specific Emitter Identification using Intrapulse Data," in *First European Radar Conference*, no. 2, Amsterdam, 2004, pp. 249–252.

[11] J. Matuszewski, "Specific Emitter Identification," in *International Radar Symposium*, 2008, pp. 1–4.

[12] C. Shieh and C. Lin, "A Vector Neural Network for Emitter Identification," *IEEE Transactions on Antennas and Propagation*, vol. 50, no. 8, pp. 1120–1127, 2002.

[13] K. A. Remley, C. A. Grosvenor, R. T. Johnk, D. R. Novotny, P. D. Hale, M. D. McKinley, A. Karygiannis, and E. Antonakakis, "Electromagnetic Signatures of WLAN Cards and Network Security," in *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology.* Athens: Ieee, 2005, pp. 484–488.

[14] O. R. Afolabi, K. Kim, A. Ahmad, and A. Ahmed, "On Secure Spectrum Sensing in Cognitive Radio Networks Using Emitters Electromagnetic Signature," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks.* San Francisco: Ieee, Aug. 2009, pp. 1–5.

[15] X. Dong, H. Weng, D. G. Beetner, T. H. Hubing, D. C. Wunsch, M. Noll, H. Göksu, B. Moss, and S. Member, "Detection and Identification of Vehicles Based on Their Unintended Electromagnetic Emissions," *IEEE Transactions on Electromagnetic Compatibility*, vol. 48, no. 4, pp. 752–759, 2006.

[16] H. Weng, X. Dong, X. Hu, D. G. Beetner, T. Hubing, and D. Wunsch, "Neural Network Detection and Identification of Electronic Devices Based on their Unintended Emissions," in *International Symposium on Electromagnetic Compatibility*, vol. 1. Chicago: IEEE, 2005, pp. 245–249.

[17] M. W. Liu and J. F. Doherty, "Nonlinear Estimation for Specific Emitter Identification in Multipath Environment," in *IEEE Sarnoff Symposium*, Princeton, 2009, pp. 1–5.

[18] J. A. Anderson, M. T. Gately, P. A. Penz, and D. R. Collins, "Radar Signal Categorization Using a Neural Network," *Preceedings of the IEEE*, vol. 78, no. 10, pp. 1646–1657, 1990.

[19] M. W. Liu and J. F. Doherty, "Specific Emitter Identification using Nonlinear Device Estimation," in *IEEE Sarnoff Symposium*, Princeton, 2008, pp. 1–5.

[20] D. Zanetti and L. Vincent, "Exploring the Physical-layer Identification of GSM Devices," ETH Zürich, Zürich, Tech. Rep., 2012.

[21] R. Lange, "Design of a Generic Neural Network - FPGA-implementation," Tech. Rep., 2005.

[22] H. Zhuang, K. Low, and W. Yau, "A Multiplier-less GA Optimized Pulsed Neural Network for Satellite Image Analysis Using a FPGA," *3rd IEEE Conference on Industrial Electronics and Applications*, pp. 302–307, 2008.

[23] L. E. Langley, "Specific Emitter Identification (SEI) and Classical Parameter Fusion Technology," in *Proceedings of WESCON*, San Francisco, 1993, pp. 377–381.

[24] R. D. Hippenstiel and Y. Payal, "Wavelet Based Transmitter Identification," in *International Symposium on Signal Processing and its Applications*, Gold Coast, 1996, pp. 740–743.

[25] B. W. Gillespie and L. E. Atlas, "Optimization of Time and Frequency Resolution for Radar Transmitter Identification," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Phoenix, 1999, pp. 1341–1344.

[26] J. Hall, M. Barbeau, and E. Kranakis, "Enhancing Intrusion Detection in Wireless Networks using Radio Frequency Fingerprinting (Extended Abstract)," in *Proceedings of the Conference on Communications, Internet, and Information Technology*, St. Thomas, 2004, pp. 201–206.

[27] C. Song, Y. Zhan, and L. Guo, "Specific Emitter Identification Based on Intrinsic Time-Scale Decomposition," in *6th International Conference on Wireless Communications Networking and Mobile Computing*, Chengdu, 2010, pp. 1–4.

[28] J. D. Wu and P. H. Chiang, "Application of Wigner-Ville Distribution and Probability Neural Network for Scooter Engine Fault Diagnosis," *Expert Systems with Applications*, vol. 36, pp. 2187–2199, 2009.

[29] T. W. Chen, W. D. Jin, and J. Li, "Feature Extraction using Surrounding-line Integral Bispectrum for Radar Emitter Signal," in *IEEE International Joint Conference on Neural Networks*, Hong Kong, 2008, pp. 294–298.

[30] M. Conning and F. Potgieter, "Analysis of Measured Radar Data for Specific Emitter Identification," in *IEEE Radar Conference*, Washington, 2010, pp. 35–38.

[31] M. Martina, A. Terreno, F. Vacca, A. Molino, G. Masera, G. D'Angelo, and G. Pasquettaz, "Real-time Implemetation of a Time-frequency Analysis Scheme," in *17th ACM Great Lakes Symposium on VLSI*, Stresa, 2007, pp. 180–183.

[32] J. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, United States of America: McGraw-Hill, 2008.

[33] L. Cohen, "Time-frequency Distributions - A Review," *Proceedings of the IEEE*, vol. 77, pp. 941–981, 1989.

[34] M. G. Frei and I. Osorio, "Intrensic Time-scale Decomposition: Time-frequency-energy Analysis and Real-time Filtering of Non-stationary Signals," *Royal Society A: Mathematics, Physical and Engineering Sciences*, vol. 463, pp. 321–342, 2007.

[35] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Pearson, 2008.

[36] W. Martin, "Time-frequency Analysis of Random Signals," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Paris, 1982, pp. 1325–1328.

[37] A. Hunter, "Feature Selection Using Probabilistic Neural Networks," *Neural Computing and Applications*, vol. 9, pp. 124–132, 2000.

[38] J. Hall, M. Barbeau, and E. Kranakis, "Detection of Transient in Radio Frequency Finterprinting using Signal Phase," in *IASTED International Multi-Conference on Wireless and Optical Communications*, Banff, 2003, pp. 13–18.

[39] T. L. Carroll, "A Nonlinear Dynamic Method for Signal Identification," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 17, pp. 1–7, 2007.

[40] S. Bhatikar, S. Khatri, G. Grudic, N. Jayakumar, T. Strohmann, J. Dobsa, and R. Mahajan, "Classification Techniques for GSM Emitters," Tech. Rep., 2005.

[41] M. A. Tahir and A. Bouridane, "An FPGA-based Coprocessor for Cancer Classification using Nearest Neighbour Classification," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, 2006, pp. III–1012 – III–1015.

[42] M. A. Tahir, "Hardware and Software Solutions for Prostate Cancer Classification using Multispectral Images," Belfast, Tech. Rep., 2006.

[43] T. Schumacher, R. Meiche, P. Kaufmann, E. Lübbers, C. Plessl, and M. Platzner, "A Hardware Accelerator for k-th Nearest Neighbor Thinning," in *Proceedings of ERSA*, Las Vegas, 2008, pp. 245–251.

[44] J. Liu, B. Li, and D. Liang, "Design and Implementation of FPGA-based Modified BKNN Classifier," *International Journal of Computer Science and Network Security*, vol. 7, no. 3, pp. 67–71, 2007.

[45] J. Vreeken, "Spiking Neural Networks, an Introduction," Tech. Rep., 2003.

[46] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial Neural Networks: A Tutorial," *IEEE Computational Science & Engineering*, vol. 29, pp. 31–44, 1996.

[47] R. Rojas, *Neural Networks: A Systematic Introduction*, 1st ed. Springer, 1996.

[48] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural Network Implementation Using FPGA: Issues and Application," *International Journal Of Information Technology*, vol. 4, pp. 86–92, 2008.

[49] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward Neural Network Implementation in FPGA using Layer Multiplexing for Effective Resource Utilization," in *IEEE Transactions on Neural Networks*, 2007, pp. 880–888.

[50] A. Braga, "Performance. Accuracy, Power Consumption and Resource Utilization Analysis for Hardware / Software Realized Artificial Neural Networks," in *IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications*, Liverpool, 2010, pp. 1629–1636.

[51] Y. Chen and W. du Plessis, "Neural Network Implementation on a FPGA," in *IEEE Africon*, George, 2002, pp. 337–342.

[52] H. Fu, O. Mencer, and W. Luk, "FPGA Designs with Optimized Logarigthmic Arithmetic," *IEEE Transactions on Computers*, vol. 59, pp. 1000–1006, 2010.

[53] K. Basterretxea, J. M. Tarela, and I. del Campo, "Approximation of Sigmoid Function and the Derivative for Hardware Implementation of Artificial Neurons," in *IEE Proceedings - Circuits, Devices and Systems*, vol. 151, no. 1, 2004, pp. 18–24.

[54] D. R. Hush and B. Horne, "Efficient Algorithms for Function Approximation with Piecewise Linear Sigmoidal Networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1129–1141, 1998.

[55] B. Zamanlooy and M. Mirhassani, "Efficient VLSI Implementation of Neural Networks with Hyperbolic Tangent Activation Function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 39–48, 2014.

[56] C. S. Shienh and C. T. Lin, "A Vector Neural Network for Emitter Identification," *IEEE Transactions on Antennas and Propagation*, vol. 50, pp. 1120–1127, 2002.

[57] L. Rutkowski, "Adaptive Probabilistic Neural Networks for Pattern Classification in Time-varying Environment," *IEEE Transactions on Neural Networks*, vol. 15, pp. 811–827, 2004.

[58] A. Upegui, C. A. Peña Reyes, and E. Sanchez, "A Functional Spiking Neuron Hardware Oriented Model," in *Computational Methods in Neural Modeling: 7th International Work-conference on Artificial and Natural Neural Networks*, Mahon, 2003, pp. 136–143.

[59] A. Upegui, C. A. Peña Reyes, and E. Sánchez, *A Hardware Implementation of a Network of Functional Spiking Neurons with Hebbian Learning.* Lausanne, Switzerland: Springer Berlin Heidelberg, 2004, pp. 233–243.

[60] L. Bakó and S. Brassai, "Hardware Spiking Neural Networks: Parallel Implementations using FPGAs," in *Proceedings of the 8th WSEAS International Conference on Automatic Control, Modeling & Simulation*, Prague, 2006, pp. 261–266.

[61] L. P. Maguire, T. M. McGunnity, B. Glachin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for Large-scale Implementations of Spiking Neural Networks on FPGAs," *Neurocomputing*, vol. 71, pp. 13–29, 2007.

[62] D. Kunkle and C. Merrigan, "Pulsed Neural Networks and their Application," Rochester Institute of Technology, Rochester, Tech. Rep., 2002. [Online]. Available: http://www.ccs.neu.edu/home/kunkle/papers/techreports/pnn.pdf

[63] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware Spiking Neural Network with Run-time Reconfigurable Connectivity in an Autonomous Robot," in *NASA/DoD Conference on Evolvable Hardware*, Chicago, 2003, pp. 189–198.

[64] B. Schrauwen and M. D'Haene, "Compact Digital Hardware Implementation of Spiking Neural Networks," Tech. Rep., 2005.

[65] A. Ghani, T. M. McGinnity, L. P. Maguire, and J. Harkin, "Area Efficient Architecture for Large Scale Implementation of Biologically Plausible Spiking Neural Networks on Reconfigurable Hardware," in *International Conference on Field Programmable Logic and Applications*, Madrid, 2006, pp. 1–2.

[66] J. Weston, "Support Vector Machine (and Statistical Learning Theory) Tutorial." [Online]. Available: http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf

[67] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The Entire Regularization Path for the Support Vector Machine," *Journal of Machine Learning Research*, vol. 5, pp. 1391–1415, 2004.

[68] A. Ben-Hur and J. Weston, *A User's Guide to Support Vector Machines*. Springer, 2010, pp. 223–240.

[69] F. M. Khan, "Hardware-based Support Vector Machine Classification in Logarithmic Number Systems," in *IEEE International Symposium on Circuits and Systems*, Kobe, 2005, pp. 5154–5157.

[70] S. R. Gunn, "Support Vector Machines for Classification and Regression," University of Southhampton, Southampton, Tech. Rep., 1998.

[71] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-forward Support Vector Machine without Multipliers," *IEEE Transactions on Neural Networks*, vol. 17, pp. 1328–1331, 2006.

[72] C. Hsu, "A Comparison of Methods for Multiclass Support Vector Machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, 2002.

[73] D. Mahmoodi, A. Soleimani, H. Khosravi, and M. Taghizadeh, "FPGA Simulation of Linear and Nonlinear Support Vector Machine," *Journal of Software Engineering and Application*, vol. 4, pp. 320–328, 2011.

[74] L. Li and H. Ji, "Combining Multiple SVM Classifiers for Radar Emitter Recognition," in *Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Tianjin, 2009, pp. 140–144.

[75] G. Zhang, W. Jin, and L. Hu, "Radar Emitter Signal Recognition Based on Support Vector Machines," in *8th Control, Automation, Robotics and Vision Conference*, Kunming, 2004, pp. 826–831.

[76] D. Anguita, A. Boni, and S. Ridella, "A Digital Architecture for Support Vector Machines: Theory, Algorithm, and FPGA Implementation," *IEEE Transactions on Neural Networks*, vol. 14, pp. 993–1009, 2003.

[77] D. Wolpert and W. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[78] M. A. Wiering, M. H. Van Der Ree, A. Nolte, M. J. Embrechts, M. F. Stollenga, A. Meijster, and L. Schomaker, "The Neural Support Vector Machine," *Neural Networks*, vol. 23, no. 5, pp. 607–613, 2010.

[79] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast Neural Networks without Multipliers," *IEEE Transactions on Neural Networks*, vol. 4, pp. 53–62, 1993.

[80] D. Anguita, A. Ghio, S. Pischiutta, and S. Ridella, "A Hardware-friendly Support Vector Machine for Embedded Automotive Applications," in *International Joint Conference on Neural Networks*, Orlando, 2007, pp. 1360–1364.

[81] M. Arnold, "21st Century Slide Rules with Logarithmic Arithmetic: High-speed, Low-cost, Low-power Alternatives to Fixed-point Arithmetic," in *Second Online Symposium for Electronic Engineers*, 2001.

[82] F. M. Khan, M. G. Arnold, and W. M. Pottenger, "Finite Precision Analysis of Support Vector Machine Classification in Logarithmic Number Systems," in *Euromicro Symposium on Digital Systems Design*, Rennes, 2004, pp. 254–261.

[83] Xilinx, "The Role of Distributed Arithmetic in FPGA-based Signal Processing," Tech. Rep. [Online]. Available: http://www.xilinx.com/appnotes/theory1.pdf

[84] J. Becker, M. Hübner, and M. Ullmann, *Run-time FPGA Reconfiguration for Power-/Cost-optimized Real-time Systems.* Darmstadt, Germany: Springer US, 2003, pp. 119–132.

[85] E. J. McDonald, "Runtime FPGA Partial Reconfiguration," in *IEEE Aerospace Conference*, Montana, 2008, pp. 1–7.

[86] S. Donthi and R. L. Haggard, "A Survey of Dynamically Reconfigurable FPGA Devices," Cookeville, 2003, pp. 422–426.

[87] B. P. Kimuli, "Introduction to GSM and GSM Mobile RF," *RF Design*, no. June, pp. 26(6):12–21, 2003.

[88] ETSI, "Digital Cellular Telecommunications System (Phase 2+) - Radio Transmission and Reception - 3GPP TS 05.05 version 8.20.0," Tech. Rep., 1993.

[89] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms.* Wiley-Interscience, 2004.

[90] M. Gori and A. Tesi, "On the Problem of Local Minima in Backpropagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 1, pp. 76–86, 1992.

[91] C. Peterson and T. Rögnvaldsson, "An Introduction to Artificial Neural Networks," in *Proc. 1991 CERN Summer School of Computing, CERN Yellow Report*, 1991, pp. 113–170.

[92] ETSI, "Digital Cellular Telecommunications System (Phase 2+); General Description of a GSM Public Land Mobile Network (PLMN)," Tech. Rep., 1997.

[93] ——, "Digital Cellular Telecommunications System (Phase 2+); Modulation," Tech. Rep., 2001.

[94] ——, "Digital Cellular Telecommunications System; (GSM Radio Access Phase 3); Multiplexing and Multiple Access on the Radio Path," Tech. Rep., 1998.

[95] A. R. Omondi, J. Rajapakse, and M. Bajger, "FPGA Neurocomputers," in *FPGA Implementations of Neural Networks.* Springer, 2006, ch. 1, pp. 1–36.

[96] Xilinx, "Virtex-5 Family Overview," Tech. Rep., 2009.

[97] A. Canas, E. M. Ortigosa, E. Ros, and P. M. Origosa, "FPGA Implementations of Neural Networks," in *FPGA Implementation of a Fully and Paritially Connected MLP.* Springer, 2006, ch. 10, pp. 271–296.

A. Appendix: Software Implementations

A.1. Feed-Forward Neural Network

Algorithm A.1 Feed-Forward Neural Network

```
function [layer1_sums,layer1_output,output_sums,output]=neural_net(

input,weights1,weights2)

%This is an ideal Neural Network, benchmark
layer1_sums=zeros(size(weights1,1),1);
layer1_output=zeros(size(weights1,1),1);
output_sums=zeros(size(weights2,1),1);
output=zeros(size(weights2,1),1);
for index_layer1=1:size(weights1,1)
layer1_sums(index_layer1,1)=sum(weights1(index_layer1,:)*input);
end;
layer1_output=1./(1+exp(-layer1_sums));
layer1_output=[1;layer1_output];
for index_output=1:size(weights2,1)
output_sums(index_output)=sum(weights2(index_output,:)*layer1_output);

end;
output=1./(1+exp(-output_sums));
```

A.2. Feed-Forward Neural Network (Scaling and LUT included)

Algorithm A.2 Feed-Forward Neural Network (including scaling)

```
function [layer1_sums,layer1_lut_output,layer1_output,layer2_sums

,layer2_lut_output,layer2_output]=test_neural_network(inputs,
weights1,weights2,resolution)

%INITIALIZATION
input_an=[inputs];
bits=25;
abs_max=1;
input_dis=round(input_an*(2^bits-2)/(2*abs_max));
weights1_an=weights1;
bits=18;
abs_max= 2^ceil(log2(8.2918));
weights1_dis=round(weights1_an*(2^bits-2)/(2*abs_max));
weights2_an=weights2;
bits=18;
abs_max= 2^ceil(log2(8.2918));
weights2_dis=round(weights2_an*(2^bits-2)/(2*abs_max));
%LAYER1

[layer1_sums,layer1_lut_output,layer1_output]=
neural_network_layer1(input_dis,weights1_dis,resolution);

layer1_lut_output=round([2^42;layer1_lut_output]./(2^18));
layer1_lut_output(layer1_lut_output>2^24)=2^24;
layer1_output=round([2^42;layer1_output]./(2^18));
layer1_output(layer1_output>2^24)=2^24;
%LAYER2

[layer2_sums,layer2_lut_output,q]=neural_network_layer1(
layer1_lut_output,weights2_dis,resolution);

[layer2_sums,q,layer2_output]=neural_network_layer1(
layer1_output,weights2_dis,resolution);
```

Algorithm A.3 Feed-Forward Neural Network ayer alculations

```
function [layer1_sums,layer1_lut_output,layer1_output]=
neural_network_layer1(input,weights,resolution)

layer1_sums=[];
layer1_lut_output=[];
layer1_output=[];
for index_layer1=1:size(weights,1)

[neuron_sum,neuron_output,neuron_lut_out,address]=
neuron(weights(index_layer1,:),input,resolution);

layer1_sums=[layer1_sums;neuron_sum];
layer1_lut_output=[layer1_lut_output;neuron_lut_out];
layer1_output=[layer1_output;neuron_output];
end;
```

Algorithm A.4 LUT PWL calculations

```
function [neuron_sum,neuron_output,neuron_lut_out,address]=
neuron(weights,inputs,resolution)
neuron_sum=sum(inputs.*weights');
address=dec2bin(mod((neuron_sum),2^43),43);
address=address(1:14);
address=[address(1),address];
address=[address(1),address];
address_i=typecast(uint16(bin2dec(address)),'int16');
if address_i<-1280 neuron_lut_out=0; elseif address_i>1279
neuron_lut_out=2^42;
else
[c1,c2]=get_LUT_vals(address(3:16));
neuron_lut_out=round(c1)*neuron_sum+round(c2);
end;
neuron_output=((2^42)./(1+exp(-neuron_sum.*resolution)));
address=address(3:16);
```