1-1-2010

Software-hardware analysis of signal feature classification algorithms

Hamidreza Asefi-Ghamari
Ryerson University

SOFTWARE-HARDWARE ANALYSIS OF SIGNAL FEATURE CLASSIFICATION ALGORITHMS

by

Hamidreza Asefi-Ghamari, B.Sc, M.Sc.,
Ferdowsi University of Mashhad, Mashhad, Iran, 1995,
Azad University of Tehran, Tehran, Iran, 2001,

A thesis
presented to Ryerson University
in partial fulfillment of the
requirement for the degree of
Master of Applied Science
in the Program of
Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2010

Author's Declaration

I hereby declare that I am the sole author of this thesis.
I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.
Signature

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.
Signature

Instructions on Borrowers

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

SOFTWARE-HARDWARE ANALYSIS OF SIGNAL FEATURE CLASSIFICATION ALGORITHMS

© Hamidreza Asefi-Ghamari, 2010
Master of Applied Science
in the Program of
Electrical and Computer Engineering,
Ryerson University.

Over the last few decades, signal feature analysis has been significantly used in a wide variety of fields. While several techniques have been proposed in the area of signal feature extraction and classification, all of these techniques are achieved by using modern computers, which rely on softwares, such as MATLAB. However, in real-time applications or portable devices, software implementation is not enough by itself, and a hardware-software co-design or fully hardware implementation needs to be considered.

The selection of the right signal feature analysis tool for an application depends not only on the software performance, but also on the hardware efficiency of a method. However, there is not enough studies in existence to provide comparison of these signal feature extraction methods from the hardware implementation aspect. Therefore, the objective of this thesis is to investigate both the hardware and algorithmic perspectives of three commonly used signal feature extraction techniques: Autoregressive (AR), pole modeling, and Mel-frequency Cepstral coefficients (MFCCs).

To fulfill this objective, first, the hardware analysis of AR, pole modeling, and MFCC feature extraction methods is performed by calculating the computational complexity of the mathematical equations of each method. Second the FPGA area usage of each feature extraction methods is estimated. Third, algorithmic evaluation of these three methods is performed for audio scene analysis.

Once the results are obtained from the above stages, the overall performance of each feature extraction method is compared in terms of both the hardware analysis and algorithmic performances. Finally, based on the performed comparison, pole modeling feature extraction approach is proposed as the suitable method for the audio scene analysis application.

The suitable method (pole modeling feature extraction) + linear discriminant analysis (LDA) classifier are implemented in Altera DE2 Board using Altera Nios II soft-core processor. The audio classification accuracy obtained using this implementation is achieved to be equal to the MATLAB implementation. The classification time for one audio sample is determined to be 0.1s, which is fast enough to be considered as a real-time system for audio scene analysis application.

Acknowledgments

I would like to express my deep gratitude to my supervisors Dr. Sridhar Krishnan and Dr. Andy Ye at Ryerson University for their knowledgeable guidance and constant encouragement and support.

I wish to express my sincere gratitude to Dr. Behnaz Ghoraani whose help and technical discussions have been invaluable to my thesis. I offer my regards and blessings to my colleagues and staff at Signal Analysis Laboratory (SAR) and Ryerson University who supported me in every aspect during the completion of my studies.

Finally, I would like to especially thank my family for their nonstop and warm support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

SIGNAL analysis has been a field of considerable interest and significant growth over the last century. A wide variety of fields, such as, communication, security, biomedicine, biology, physics, finance and geology has benefited from the science of signal analysis. They use signal processing methods and algorithms implemented on computers or in electric hardware to design algorithms, develop models, and make informed decisions based on the models. For instance, in the filed of communication, audio signals are important sources of information for understanding the content of multimedia. Therefore, audio analysis techniques have been developed in order to characterize the audio signals for applications, such as, multimedia indexing and retrieval, and auditory scene analysis.

Several signal analysis techniques have been developed to analyze, interpret, manipulate, and process our surrounding signals in an attempt to acquire the useful information toward human's benefit. However, one of the remaining challenges is dealing with the dynamic characteristics of these real world signals. Signals are either stationary or non-stationary. The former is denoted to signals which their statistics (such as mean and variance) are fixed over time and follow a probabilistic distribution. The latter refers signals with variable dynamics in time. A majority of real-world signals generated by nature belong to non-stationary. For example, speech signals, heart signals (ECG: electrocardiogram), and brain signals (EEG: electroencephalogram), are non stationary in nature.

The analysis of real-world signals is challenging as the dynamic nature of the real-world system causes the signal to have stochastical and non stationary behavior. To address this issue, the parametric or non-parametric modeling method has been proposed. This approach, which is termed feature analysis, involves extracting discriminatory features from the signal and feeding them into a classifier. While several techniques have been proposed in the area of signal feature extraction and classification, all of these techniques are achieved by using modern computers, which rely on softwares, such as MATLAB. In recent years, the hardware technology has been significantly developed and gained significant attention from technology leaders. Hardware technologies, such as VLSI (Very Large Scale Integration) technology, have been commonly used in devices, like computers, digital cameras, cell phones, MP3 players, digital TV sets and so forth. Recently, FPGA (Field Programmable Gate Array) is found more appealing due to strong functions of FPGA itself, such as shorter time to market, ability to reprogram, and lower non-recurring engineering costs.

The selection of the right signal feature analysis tool for an application depends on both the software performance and the hardware efficiency of a method. For example, the software implementation evaluates the overall performance of each signal processing method for the application in hand, and the hardware implementation investigates that the implementation of which method is more suitable. While the literature contains a significant amount of software-based analysis and comparison as related to different signal feature extraction methods, there is not enough studies in existence that provide comparison of these methods from the hardware implementation aspect. To address this shortcoming, there is a need to investigate signal analysis in both the hardware and the software domain. The objective of this thesis is to provide such an understanding by investigating both the hardware and software analysis of three commonly used signal feature extraction techniques.

1.2 Signal Feature Analysis State-of-The-Art

Feature analysis aims to classify a given data based on groups of measurements in the data. Depending on the application, these measurements or observations are collected based on either a priori knowledge or a set of statistical information extracted from the data. The block diagram in

Fig. 1.1 shows the four stages exist in a feature analysis system. The first block includes a sensor that gathers the observations to be classified or described. Sensors measure physical quantities and convert them into signals which can be recorded for further analysis. Some examples of sensors include: thermometer for temperature, and microphone for audio and speech. The second block is signal preprocessing. The preprocessing stage may contain of one or two signal processing stages that provide an optimum representation of the signal. This stage could include a segmentation stage which divides the signal into shorter durations which can be considered stationary. The third block is feature extraction that maps the signal into some points in an appropriate multi-dimensional space (ie. feature space). The final block is a classifier that does the actual task of classifying the signals relying on the extracted features.

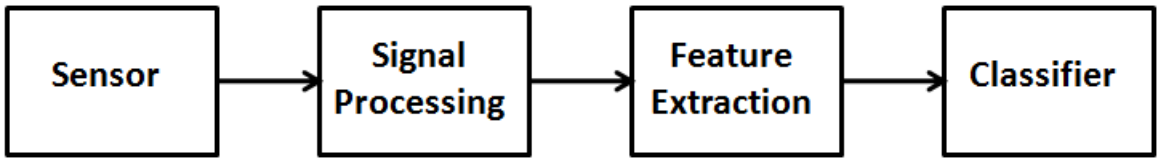

Figure 1.1: General schematic of a signal feature analysis system

1.2.1 Feature Extraction

Feature extraction involves simplifying the amount of resources required to accurately describe a large set of data. When performing analysis of complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy. As mentioned, features play a very important role in any pattern recognition system. If the extracted features are so well defined, even simple classification methods will be good enough to accurately and efficiently classify the data. Therefore, developing more powerful features and understanding the feature space should be a vital consideration in designing automatic decision making algorithms. Several parametric and non-parametric features have been proposed in the literature as explained below.

Parametric Features: Parametric features are obtained based on parametric modeling of random signals with the assumption that the signals are stationary and can be presented in terms of the linear combination of several past values of model output plus the linear combination of present and past values of model input [2]. Many researches have demonstrated that parametric modeling is a useful method when dealing with random time series [3, 4, 5] and segmentation is an efficient approach to deal with non stationary signals [2, 6, 7]. Examples of parametric representations include: reflection coefficients, linear prediction coefficient (LPC), line spectral frequencies (LSFs), autoregressive (AR) modeling, and dominant pole modeling. Among parametric features, autoregressive (AR) modeling and dominant pole modeling are of interests and investigated for use in signal analysis.

It has been demonstrated that in many cases, AR spectrum provides a better resolution than traditional Fourier spectrum [2], which can make the signal analysis easier. To obtain the AR spectrum, one has to obtain the AR coefficients of the signal first [8]. Moreover, AR coefficients can be easily used in pattern classification [9, 10] and data compression application [11]. Pole modeling obtained from AR model of signals have given promising results in classifying the phonocardiogram [12], electrocardiogram [13], electrocorticogram [14], and vibroarthrogram signals [3].

Non-parametric Features: Non-parametric features are derived based on the characteristics of the signals without any assumption about the signal model [15]. Features such as signal energy, pitch, zero crossing rate [16, 17] and Entropy modulation [18] have been used for audio classification. Other non-parametric features include 4 Hz modulation energy, percentage of low-energy frames, spectral roll off point, spectral centroid, mean frequency, cepstral coefficients [19, 20], high and low frequency slopes [21], and spectrum flux (SF) [19]. Mel-frequency Cepstral coefficients (MFCCs) [22] are well-known non-parametric features used for the purpose of modeling the human auditory perception system in the area of audio and speech processing [23, 24, 25, 26].

Fig. 1.2 shows the focus of the present thesis. This thesis is based on three well-known parametric and non-parametric feature extraction methods: AR and pole modeling, and MFCC. A comprehensive explanation of these methods are explained in Chapter 2.

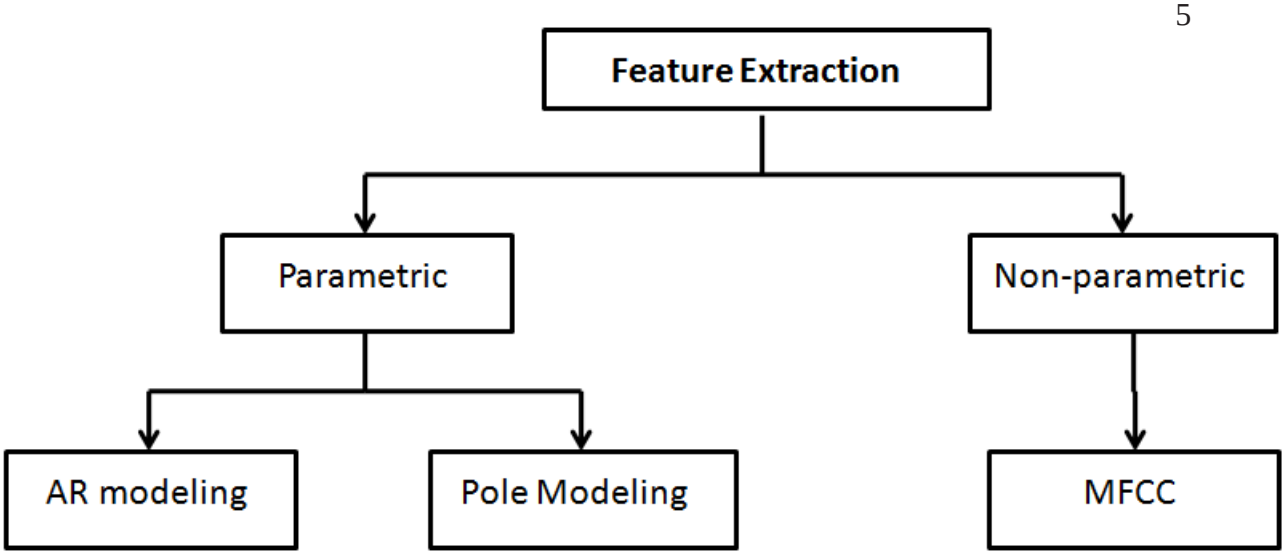

Figure 1.2: Feature extraction methods in this thesis

1.2.2 Classifier

Classification refers to a prediction rule that assigns the signals into different classes. Various classifiers have been utilized in the literature. Audio content analysis at Microsoft research commonly uses Gaussian mixture models (GMM) [27], k-nearest neighborhood (K-NN) [28] and support vector machine (SVM) [29] for audio classification. Other popular classifiers for audio classification include linear discriminant analysis (LDA) [30], hiddenMarkov models (HMM) [31] and artificial neural networks (ANN) [32]. There are some works that focus attention on developing new classifiers, or comparing existing classifiers for audio classification applications. For instance, in [33], Buchler et. al. compare simple classifiers (e.g., rule-based and minimum-distance classifiers) with complex approaches (e.g., Bayes classifier, neural network and hidden Markov model).

While these studies are beneficial, the aim of the present study focuses on investigating the right feature extraction method in hardware device. Therefore, in this thesis, we avoid complex classifiers and apply LDA as a simple linear classifier to evaluate the feature extraction methods. This classifier is further explained in Chapter 2.

1.3 Hardware/Software Implementation State-of-The-Art

Implementation of feature analysis algorithms have been generally performed in computers using commonly used software programs such as MATLAB [34], and Mathematica [35], C programming, and so forth. However, software implementation is not enough by itself in case of many real-world applications as follows: (i) real-time applications and (ii) portable devices. Real-time performance is desirable in many applications, such as audio scene analysis in hearing aids or target tracking in computer vision. However, with the high computational complexity of developed signal analysis algorithms, it is difficult to achieve the real-time goal with software-only implementation. For example, performing the discrete Fourier transform (DFT) of a signal with N samples takes O(N^2) arithmetical operations, which means that the complexity order increases with order 2 of the signal length. Hardware-software co-design or fully hardware implementation can be considered as a solution to this demand in real-time applications.

Electronic portable devices are emerging in the market with the focus on high performance signal analysis in order to improve the quality of life. For example, a portable device that can detect certain medical conditions (blood pressure, breath alcohol level, and so on) from a users touch [36]. Many such capabilities could be integrated into a portable wireless device that also contains the users medical history. It may even be possible to detect certain contextual information, such as the users level of anxiety, based on keystroke patterns. After analyzing data input, the device could transmit an alert message to a healthcare provider, the nearest hospital, or an emergency system if appropriate. Another example is in hearing aids [32]. People with hearing disability depend on assistive devices such as hearing aids to listen to the sounds around them. It is very important for these assistive devices to determine the environment using the auditory clues in order to build better instruments with automatic switching features [37]. This would improve the quality of life of people with disability. Some practical situations would be in adaptively changing the noise reduction strategy depending on the noise environments, alerting the hearing-impaired listener when a fast approaching automobile is detected, and audio source localization for navigation.

An essential part in portable devices is the use of hardware architectures to implement the signal processing algorithms developed and evaluated in a programming software. Since 1970s, VLSI

technology has significantly been used in electronic devices. These devices rely on VLSL chips, including both ASIC (Application-Specific Integrated Circuit) and FPGA (Field Programmable Gate Array). In most recent years, the FPGA technology has been significantly developed and gained more and more preference due to its advantages of low design effort, shorter time to market, ability to reprogram and lower non-recurring engineering costs. Many applications have been achieved by using FPGA techniques in various areas, i.e. digital signal processing, aerospace, medical imaging, computer vision, speech recognition, ASIC prototyping, bioinformatics.

There are some recent publications dealing with FPGA implementations of calculating MFCC and Linear-scale Filterbank Cepstral Coefficients (LFCC) for a real-time feature extraction solution. In [38, 39, 40] the implementation of a feature extraction system based on a dedicated hardware which consists of several stages designed to calculate the feature vectors, based on MFCC and LFCC. There are few papers which discuss the implementation of AR modeling. One implementation can be found in [41] which implements the Burg algorithm onto the AMD29500 microprogrammable byte slice DSP and NECμPD77230 single-chip DSP. The AMD DSP system can have a sixteenth-order modeling rate at 17kHz while the NEC DSP system can have a sixteen-order model at 8kHz. The other hardware implementation of AR modeling is the work in [42] which the authors implement AR model of order 3 based on the Burg-lattice algorithm using fixed point arithmetic. They take advantage of Xilinx System generator to implement the algorithm in Xilinx Virtex II Pro device. In this thesis, an FPGA implementation of a pole modeling method feature analysis approach is presented using the Altera NiosII soft-core processor.

1.4 Research Objective

The objective of this thesis is to investigate both the hardware and software perspectives of three commonly used signal feature extraction techniques: Autoregressive (AR), pole modeling, and Mel-frequency Cepstral coefficients (MFCCs). The present thesis presents a comparison of both the hardware and software analysis of AR, pole modeling and MFCC signal feature extraction techniques. These contributions are explained as follows:

- Hardware analysis of AR modeling, pole modeling, and MFCC. The computational com-

plexity of these three techniques is analyzed in details using the mathematical equations of each method. Based on this analysis, an area comparison of these feature extraction methods in FPGA is provided.

- Implementation and evaluation of AR and pole modeling-, and MFCC-based features for audio scene feature analysis. An algorithmic performance comparison of these three well-known feature extraction techniques is provided using MATLAB programing.

- Performance comparison of AR, pole, and MFCC feature extraction methods. This comparison is performed based on both hardware and algorithmic performance obtained from two previous contributions.

- Implementation of audio signal feature analysis based on AR-pole modeling feature extraction and LDA classifier. Using NiosII soft-core processor, a complete pole modeling feature extraction and classification is implemented in Altera DE2 Board.

1.5 Thesis Organization

Fig. 1.3 displays the organization and contribution of this thesis. The contributions of the present thesis are highlighted. This thesis consists of six Chapters as follows: Chapter 1 introduced the significance of signal feature analysis and the challenges in real-world signal analysis applications. A comprehensive signal analysis is explained. This chapter also described the importance of investigating both the hardware and software analysis of the existing feature analysis methods. It also reviewed some of the commonly employed feature analysis tools as related to non stationary and complex signals.

Chapter 2 covers the detailed analytical procedures of three well-known feature extraction methods (AR, Pole, and MFCC), and a simple and commonly used feature classifier (LDA classifier).

Chapter 3 computes the complexity of AR modeling, pole modeling, and MFCC feature extraction methods for hardware implementation purposes. The computational complexity of these three techniques is analyzed in details using the mathematical equations of each method. The pole

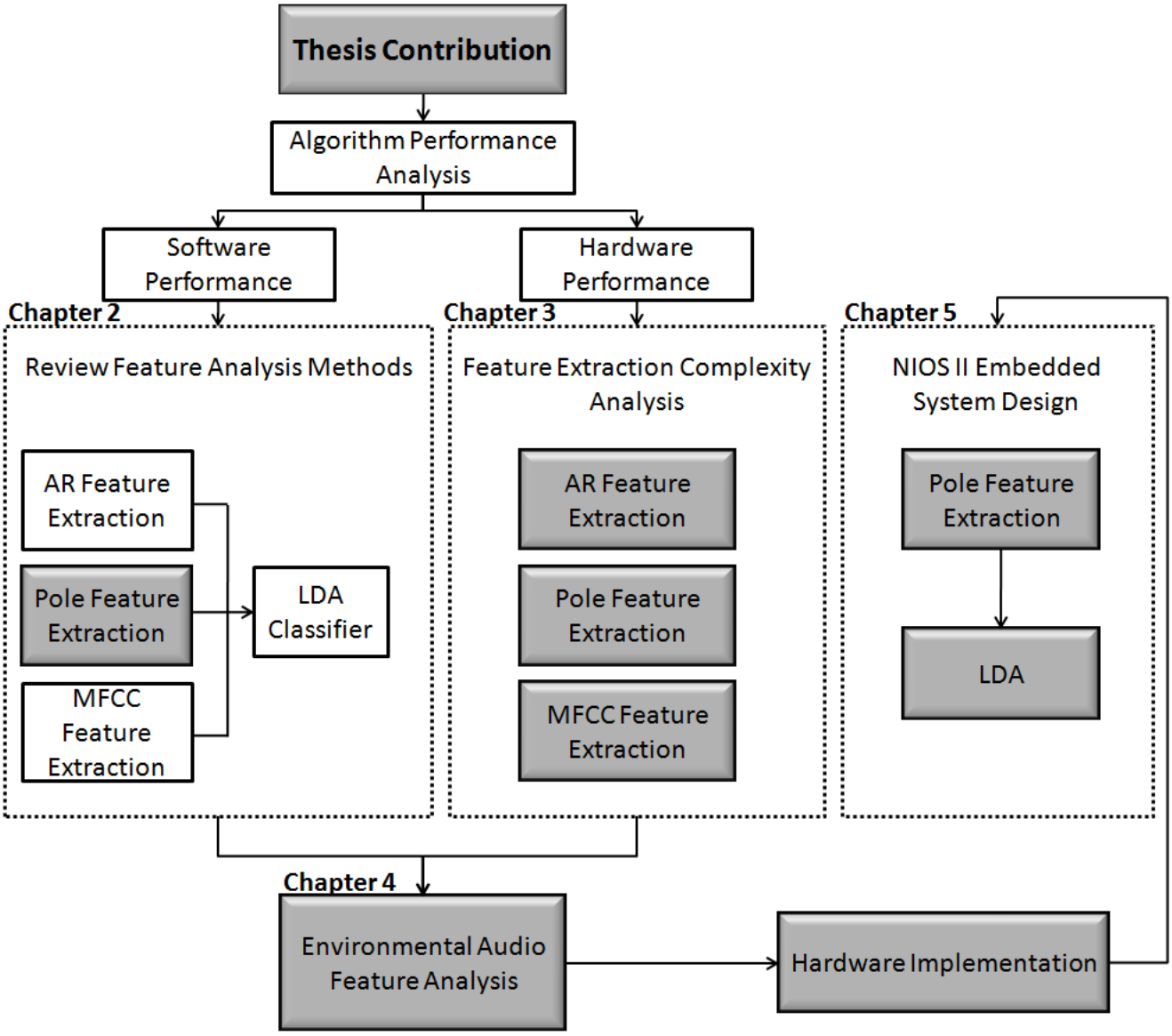

Figure 1.3: Thesis Organization.

modeling computational complexity is the first known investigation for hardware implementation analysis.

Chapter 4 evaluates and compares the algorithmic performance and hardware analysis results of AR, Pole, and MFCC feature analyses for environmental audio scene analysis application. This comparison is the first known work presented in the literature.

Chapter 5 explains the implementation of the pole modeling feature extraction + LDA classifier using ALTERA DE2 development board and Niose II embedded system design. This contribution is the first known work performed in the literature.

Chapter 6 concludes the thesis and presents discussion for future work of this research.

Chapter 2

Feature Analysis Algorithms

2.1 Introduction

Previous Chapter studied current feature analysis methods developed to classify environmental audio signals, and selected three well-known feature extraction methods (ie. AR, pole, and MFCC) and a commonly used feature classifier (ie. LDA). In the present Chapter, a detailed explanation of these methods are described. Fig. 2.1 displays the organization of this Chapter. AR modeling and MFCC features have been previously proposed for audio signal classification; however, as highlighted in the diagram of Fig. 2.1, the pole features are used in audio signal classification for the first time. The classifier used in this thesis is LDA. This classifier is used in software evaluation in MATLAB programming and FPGA implementation using Nios II soft-core processor.

2.2 Autoregressive Modeling

Autoregressive Modeling is one of the commonly used parametric modeling methods in feature extraction algorithms. In parametric modeling the value of the model output is presented by a linear combination of several past values of the model output plus the linear combination of present and past values of model input, this is presented in the following equation:

$$y(n) = -\sum_{k=1}^{m} a_k y(n-k) + G\sum_{l=0}^{Q} b_l x(n-l) \tag{2.1}$$

In the above equation, $b_0 = 1$, $x(n)$ is the model input, $y(n)$ is the model output, and G is

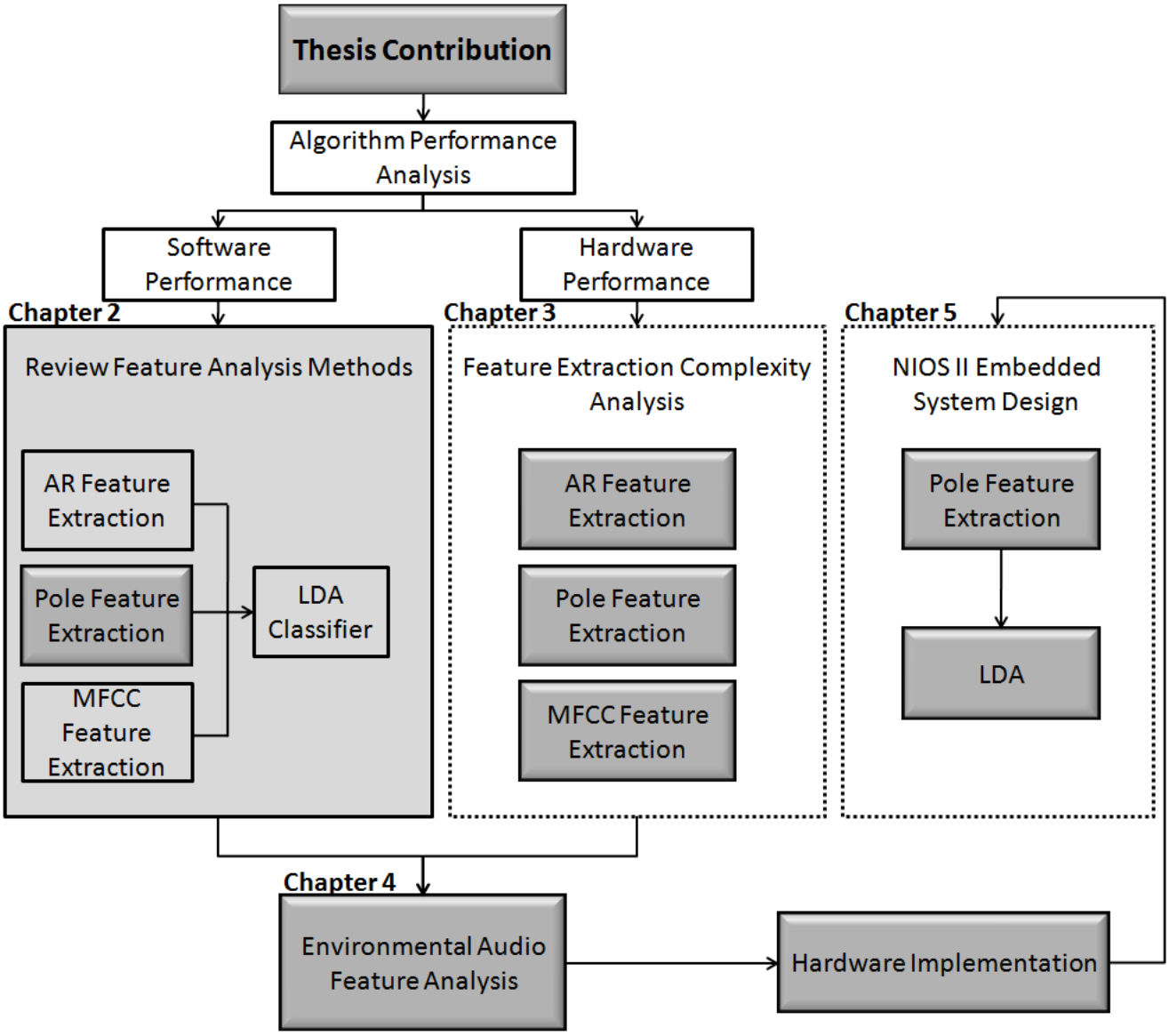

Figure 2.1: Chapter 2 - Feature Analysis Algorithms.

the gain factor. Transfer function for parametric modeling can be easily extracted by applying z-transform to the above equation (2.1):

$$H(z) = \frac{Y(z)}{X(z)} = G\frac{1 + \sum_{l=1}^{Q} b_l z^{-l}}{1 + \sum_{k=1}^{m} a_k z^{-k}} \tag{2.2}$$

Based on the parameters above, three modeling methods can be defined for a signal:

- AR(Autoregressive) modeling corresponds to the situation that b_l in Equation (2.2) is all equal to zero.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 + \sum_{k=1}^{m} a_k z^{-k}} \tag{2.3}$$

- MA(Moving average) modeling corresponds to the situation that a_k is all zero.

$$H(z) = \frac{Y(z)}{X(z)} = G(1 + \sum_{l=1}^{Q} b_l z^{-l}) \tag{2.4}$$

- ARMA(Autoregressive moving-average) modeling corresponds to the situation that a_k and b_l both are not all equal to zero.

$$H(z) = \frac{Y(z)}{X(z)} = G\frac{1 + \sum_{l=1}^{Q} b_l z^{-l}}{1 + \sum_{k=1}^{m} a_k z^{-k}} \tag{2.5}$$

Among these three methods, AR modeling has been most commonly used in dealing with audio signals mainly since audio signals have an underlying autoregressive structure, and can better be represented using this model [43]. For an AR model, the output is modeled as the linear combination of m past values of the model output and the present model input (no past values of the model input are used) as (2.1):

$$y(n) = -\sum_{k=1}^{m} a_k y(n-k) + Gx(n) \tag{2.6}$$

By applying the z-transform to the above equation, the AR transfer function is:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 + \sum_{k=1}^{m} a_k z^{-k}} \tag{2.7}$$

In AR modeling, the purpose is to obtain those AR parameters a_k (also known as AR coefficients). In the majority of real-world applications, e.g. the speech recognition or biomedical signal modeling, the input $x(n)$ is totally unknown. Hence, we are only interested in predicting the output $y(n)$ as the linear combination of previous output samples, which means that $Gx(n)$ has to be removed in Eqn. 2.6:

$$\tilde{y}(n) = -\sum_{k=1}^{m} a_k y(n-k) \tag{2.8}$$

As a result of such an assumption, there will be an error as defined in the following equation:

$$e(n) = y(n) - \tilde{y}(n) = y(n) + \sum_{k=1}^{m} a_k y(n-k) \tag{2.9}$$

From equation in Eqn. (2.9), the general block diagram of an AR model can be shown as in Fig. 2.2. $\tilde{y}(n)$ is the predicted value of the current sample $y(n)$ and the forward prediction error is $e(n)$.

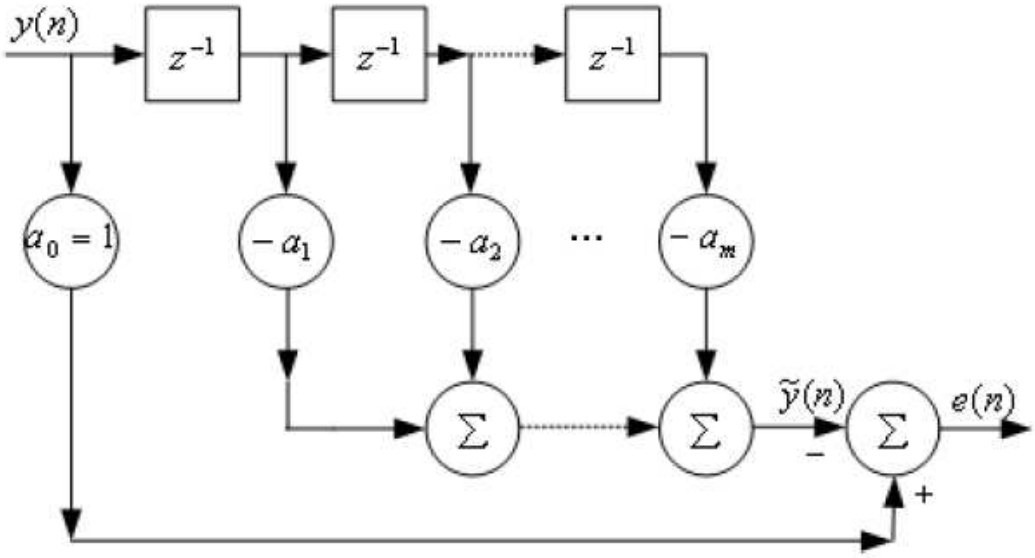

Figure 2.2: Signal-flow diagram of AR model

Computing the AR Coefficients a_k to minimize the prediction error $e(n)$ is the purpose of AR modeling. Several methods have been proposed in the literature to compute the AR model coefficients in such a way that the above prediction error is minimized [44]. Generally, two approaches has been taken in computations of the AR model coefficients: directly or iteratively. However, since

the iterative methods cost more computation to achieve a desired degree of convergence than the direct methods [45], the present thesis focuses on direct approaches.

Burg method is one of the approaches proposed by Burg in 1967 [46] for computation of AR modeling coefficients. Burg algorithm uses the lattice structure for computing forward/backward prediction errors as shown in Fig. 2.3. This method uses a lattice filter and directly estimates re-

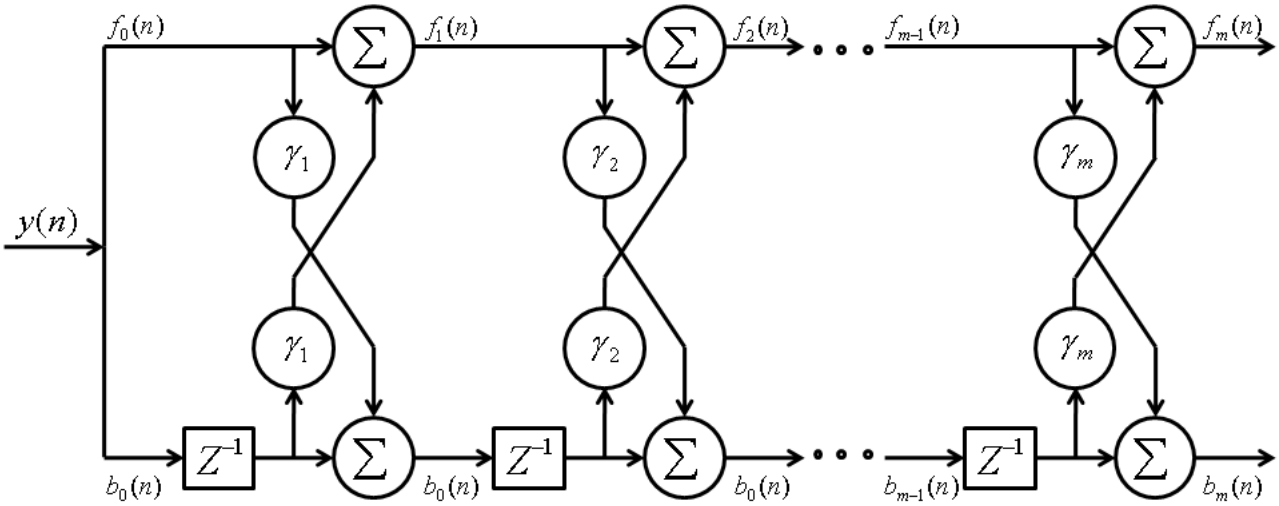

Figure 2.3: Burg-lattice Filter

flection coefficients $\{\gamma_1 \ldots \gamma_m\}$. The key step in the algorithm involves minimizing the sum of the norm of the forward and backward residual vectors, as a function of the reflection coefficient matrices. Since the computed coefficients are the harmonic mean between the forward and backward partial autocorrelation estimates, the Burg procedure is also known as the Harmonic algorithm. This algorithm starts with a first-order model and computes the prediction parameters (reflection coefficients) for successively higher model orders.

The ith reflection coefficient in Fig. 2.3 is a measure of the correlation between $y(n)$ and $y(n-i)$ after the correlation due to the intermediate observations $y(n-1), ..., y(n-i+1)$ has been filtered out. As the recursion constrains the filter poles to fall within the unit circle stability of the filter is guaranteed. The Burg method is particularly useful for estimating coefficients from segments of unequal length. This method is based on Levinsons recursions and estimates the AR filter parameters through the associated reflection coefficients constraining the AR coefficients to

satisfy Levinson equations. As the Burg algorithm uses lattice structure, it inherits the advantages of lattice structure such as stability, modularity, computational simplicity and efficiency. The Burg lattice structure is modular which means by increasing the order of the filter requires adding only one extra module, leaving all other modules and its associated filter parameters the same. Besides these, it is proven to be an efficient linear prediction technique and is probably the most widely known method to estimate AR coefficients [42]. Considering the advantages of Burg-lattice algorithm, in this thesis, Burg method is used for AR parameter modeling.

2.3 Pole Modeling

Fig. 2.4 displays the overall stages in the pole modeling used in this thesis. Pole parameters are calculated from a standard autoregressive model of order m followed by a root finding method to calculate the poles of the AR transfer function. Applying the z-transform to Eqn. 2.6 the AR

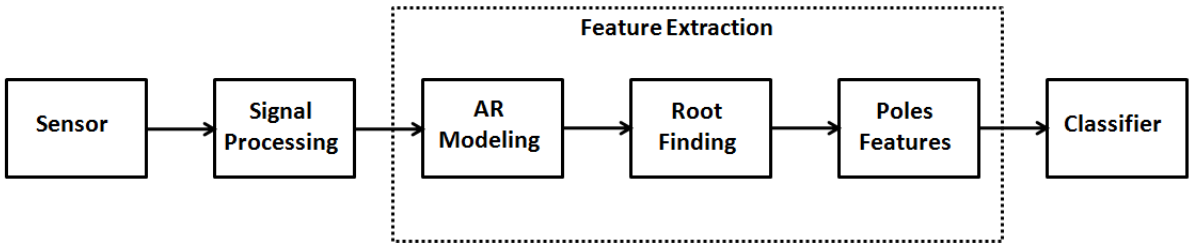

Figure 2.4: Pole Modeling overall diagram.

transfer function can be described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + \sum_{k=1}^{m} a_k z^{-k}} \tag{2.10}$$

Factorizing the denominator polynomials in Eqn. 2.10, the transfer function can be expressed as given below:

$$H(z) = \frac{1}{1 + \prod_{k=1}^{m} (1 - p_k z^{-k})} \tag{2.11}$$

or:

$$H(z) = \frac{1}{(z - p_1)(z - p_2)(z - p_3)...(z - p_m)} \tag{2.12}$$

The parameters p_k, $(k = 1, 2, ..., m)$, are the poles of $H(z)$, the system representation. In order to compute these parameters, a root finding algorithm is applied as follows:

2.3.1 Roots Finding Algorithm

Pole parameters in Eqn. 2.12 can be obtained by finding the roots of the polynomial P in the dominator of Eqn. 2.10 as shown below:

$$P(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + ... + a_m z^{-m} \tag{2.13}$$

Considering $a_m = 1$, Eqn. 2.13 can be re-structured as following monic polynomial without losing generality:

$$P(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + ... + z^m \tag{2.14}$$

The problem of solving the polynomial equation is a well-developed field of mathematics and computer science, and there are several root-finding techniques for solving such a problem. There are two types of roots finding techniques: Analytical and Numerical. Analytical Root-finding Techniques such as quadratic equation for polynomial equation of degree two or analogous formulate exist for polynomials of degree three and four. However, for polynomials of degree five and higher, analytical solutions are not always possible, and only numerical solutions are possible [47]. A numerical method for determining zeros of a polynomial generally is an iterative method to construct one or several sequences of complex numbers supposed to converge to a zero of the polynomial [47]. As one would expect, each algorithm has its advantages and disadvantages and therefore the choice of the 'best' algorithm for a given problem is never easy. Here are some desirable properties that an algorithm may have:

- Converges to a zero of the given polynomial,
- Finds both real and complex roots of a polynomial,
- Satisfies global convergence: Algorithms that do not require a sufficiently close starting value to converge are globally convergent,
- Satisfies unconditional convergence: If an algorithm is convergent (locally or globally) for all polynomials, it is unconditionally convergent,
- Fast speed of Convergence,

Selection of the right technique depends on the nature of a problem. In fact, for some problems, the majority of the techniques may fail to find a solution at all, whereas only one technique can succeed. For other problems, several techniques may, indeed, be able to solve the problem and the numerical analyst may select the one that is more computationally efficient compared to the others [47]. For example, the bisection method is a simple and reliable method for computing roots of a function when they are known to be only real values. Newton's method is locally convergent and will converge to complex zeros only if the initial approximation is complex. However, a good combination of Muller's and Newton's Methods can produce a reliable and fast program. Muller's method computes estimation of roots and these estimated values are used as the initial values for Newton's method. The J-T algorithm is fast and globally converges for any distribution of zeros [48]. Also, few critical decisions have to be made by the program which implements the algorithm. For instance, shifting is incorporated into the algorithm itself in a natural and stable way. Shifting breaks equimodularity and speeds the convergence. Eigenvalues of Companion Matrix is a very accurate method for computing zeros of a polynomial [49]. Table 2.1 summarizes the algorithms used by some of the most popular numerical software programs. In this thesis, Eigenvalues of Companion Matrix algorithm is used, which is commonly used in the literature and also employed in MATLAB software. This algorithm is explained in detail in Chapter 3.

Algorithm	Numerical Software	Command
Modified Laguerre	NAG F77 library	C02AFF
Eigenvalues of companion matrix	MATLAB	roots
Madsen-Reid	HSL	PA16
Jenkins-Traub	IMSL	CPOLY
Jenkins-Traub	Mathematica	NSolve

Table 2.1: Root finding algorithms used by commonly used numerical software.

2.3.2 Pole Modeling Parameters as Features

As shown in Fig. 2.4, the poles of the model could be used as features to construct feature vectors for signal representation and classification [50]. The dimension of the feature vector are the same as the model order. The superior performance of poles in tracking the frequency or spectral behavior of a signal makes them an appropriate choice for parametric representation of signals. The poles should also assist in associating the features with physical characteristics of the signal source. A pole in z-plane can be represented by two characteristics: magnitude and angle. In this thesis two features are extracted from the poles obtained in each signal segment in a way that they best represent the signals' poles in z-plane. These features are appended together to form a combined feature vector. These two features are: the spectral bandwidth and the pole angle in z-plane are as explained below:

Let us consider $p_i = a + jb$ as a complex pole of the system. The spectral bandwidth f_B is measured as the distance $r = \sqrt{a^2 + b^2}$ of a pole from the origin in the complex z-plane [50]:

$$f_B = cos^{-1}\left[\frac{(1 + r^2) - 2\,(1 - r)}{2r}\right] \tag{2.15}$$

The angle of p_i is calculated as follows:

$$\theta = tan^{-1}\left[\frac{b}{a}\right] \tag{2.16}$$

2.4 Mel Frequency Cepstral Coefficients(MFCC)

The original MFCC was introduced by Davis and Mermelstein in 1980 [51]. Mel-scaled Frequency Cepstral Coefficients (MFCC) is a non-parametric method of modeling the human auditory perception system. The term *mel* denotes some kind of measurements of perceived frequency or pitch of a tone. The auditory response of the human ear is non-linear which is mapped by using MFCC. MFCCs are based on the Mel frequency scale which approximates the non-linear way that humans perceive sounds by emphasizing the lower frequencies more than the higher frequencies [51]. The mapping between the real frequency scale (Hz) and the perceived frequency scales (mels) is approximately linear below 1KHz and logarithmic at higher frequency. The formula that models their

relationship is described as[52]:

$$F_{mel} = 2595 \times Log_{10}\left(1 + \frac{F_{HZ}}{700}\right) \tag{2.17}$$

The perceptual masking in MFCC is achieved by using the Mel-filter bank shown in Fig. 2.5.

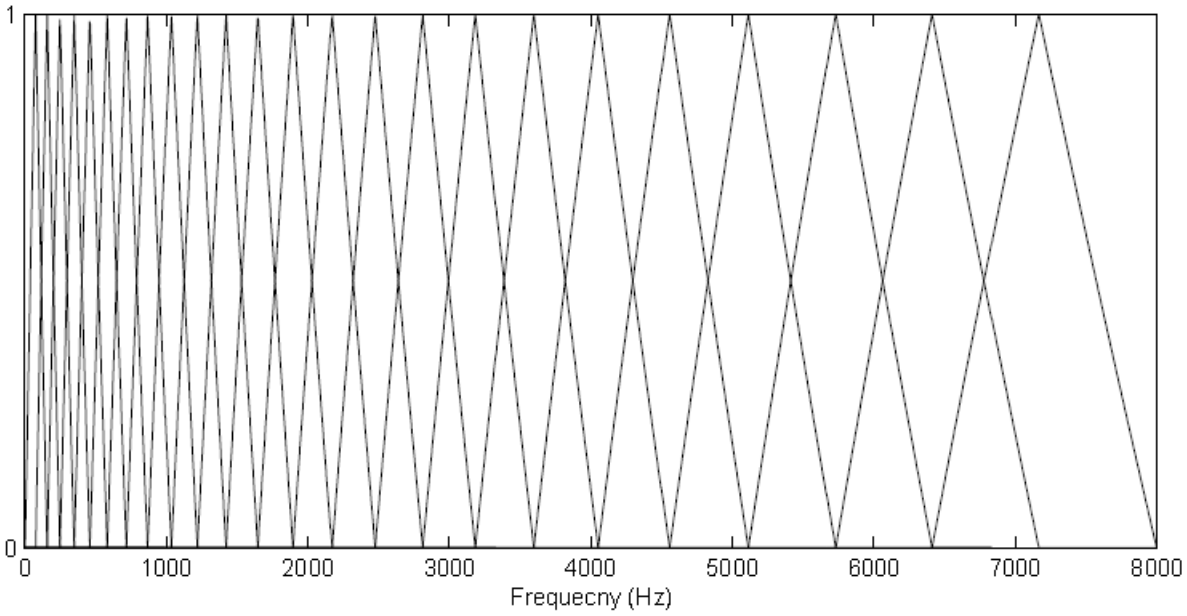

Figure 2.5: Mel scale filter bank

The overall process of the MFCC is shown in Fig. 2.6. First, discrete Fourier transform (DFT) is applied to the speech signal. Next, the output of the DFT is passed through a perceptually spaced bank of twenty equal height triangular filters to obtain the energy of signals. Finally, a set of discrete cosine transform is applied to logarithmically compressed filter-output energies to gain the MFCCs.

2.5 Feature Classifiers

The classifier used in this thesis is LDA. This method is explained as follows.

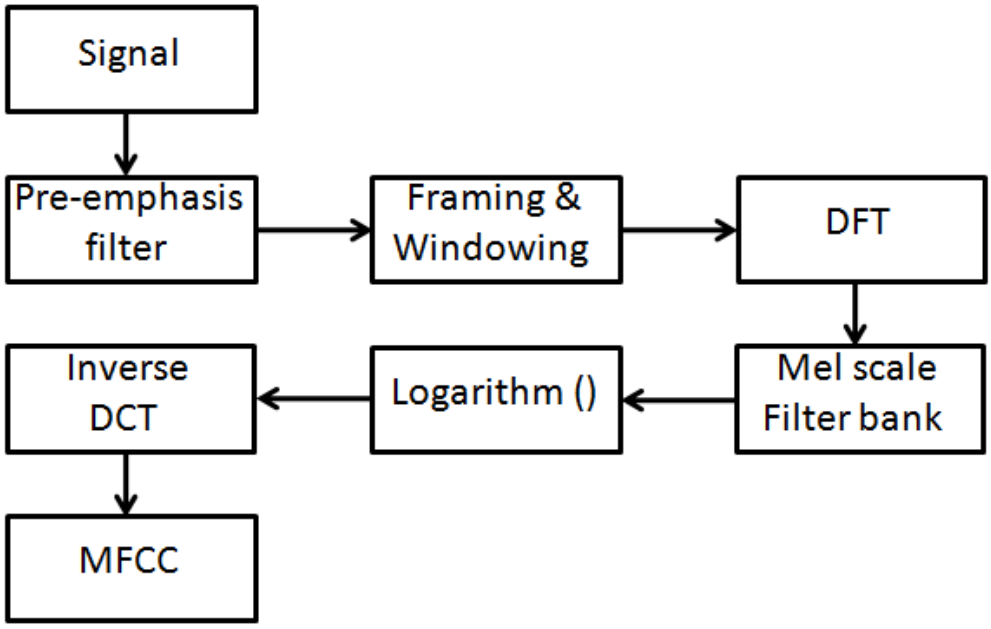

Figure 2.6: MFCC Block diagram

2.5.1 Linear Discriminant analysis (LDA)

Linear Discriminant analysis DA (LDA) (or Fishers linear discriminant) originally developed in 1936 by R.A. Fisher. LDA is a classic method of classification that has been widely used in many signal processing applications. LDA is a simple and efficient discriminant analysis which produces compatible accuracies compared to complex classifier methods. In the discriminant analysis, the feature vector containing the set of the features were transformed into canonical discriminant functions such as:

$$f = a + v_1 b_1 + v_2 b_2 + ... + v_n b_n \tag{2.18}$$

where $\{v_1, v_2, ..., v_n\}$ is the vector containing the set of features, and $\{b_1, b_2, ..., b_n\}$ and a are the classifier coefficients and constant, respectively. Using the discriminant functions values (scores) and the prior probability values of each group, the posterior probabilities of each sample occurring in each of the groups are computed [53]. The sample is then assigned to the group with the highest posterior probability. Fig. 2.7 shows an LDA classifier for two target groups and 2D feature space $\{v_1, v_2\}$. In the scenario shown in Fig. 2.7, the LDA classifier is the line separating the feature samples of Group A and Group B signals. When a new feature sample is going to be classified

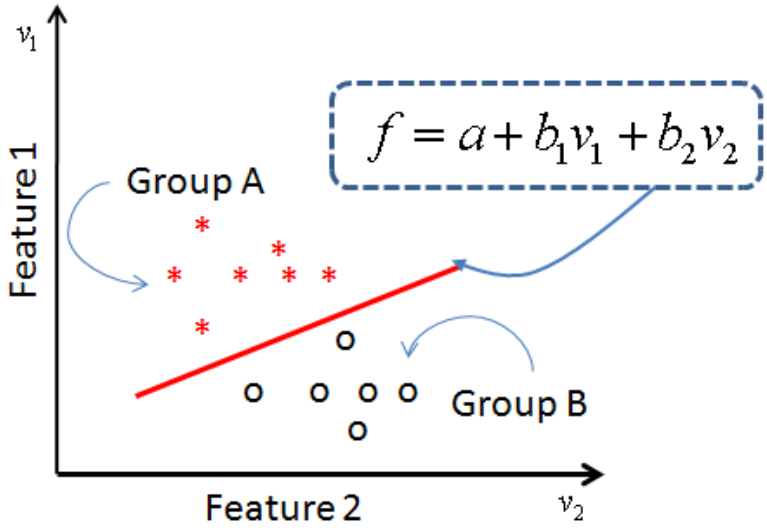

Figure 2.7: LDA shematic.

based on the LDA classifier, the group of the new feature sample will be defined depending on its location in the feature space in respect to the classifier line. If the sample falls above the line, the system will decide that the signal belongs to Group A; otherwise, the signal will be classified as Group B signal.

2.6 Summary

In this Chapter, a detailed explanation of AR, pole modeling and MFCC have been presented. The next Chapter will calculate the computational complexity of each of the three techniques in details using the mathematical equations of each method.

Chapter 3

Computational Requirements Analysis

3.1 Introduction

The algorithmic explanation of three well-known feature extraction method (ie. AR, pole, and MFCC) were provided in Chapter 2. Fig. 3.1 displays the organization of the present Chapter. The aim of this Chapter is to compute the complexity of AR modeling, pole modeling, and MFCC for hardware analysis.

As shown in Fig. 3.2, the computational complexity of each of the three techniques will be analyzed in details using the mathematical equations of each method. For each method, first each algorithm is divided into its consisting mathematical stages. Second, the sequencing graph (SG) of each mathematical stage is plotted. A sequencing graph presents the operations and their partial order. Next, based on the SG of each stage, the number of required mathematical operations is obtained. Finally, the numbers of operations in each stage are added to obtain the total number of operations for each feature extraction algorithm. It should be noted that the pole modeling computational complexity is the first known investigation for hardware implementation analysis.

3.2 AR modeling with Burg algorithm

Fig. 3.3 shows one stage in the lattice structure for computing forward/backward prediction error. For any model order increase, the AR coefficients is computed by simply adding one or more lattice stages without affecting the earlier computations for the lower orders. Fig. 3.4 shows the

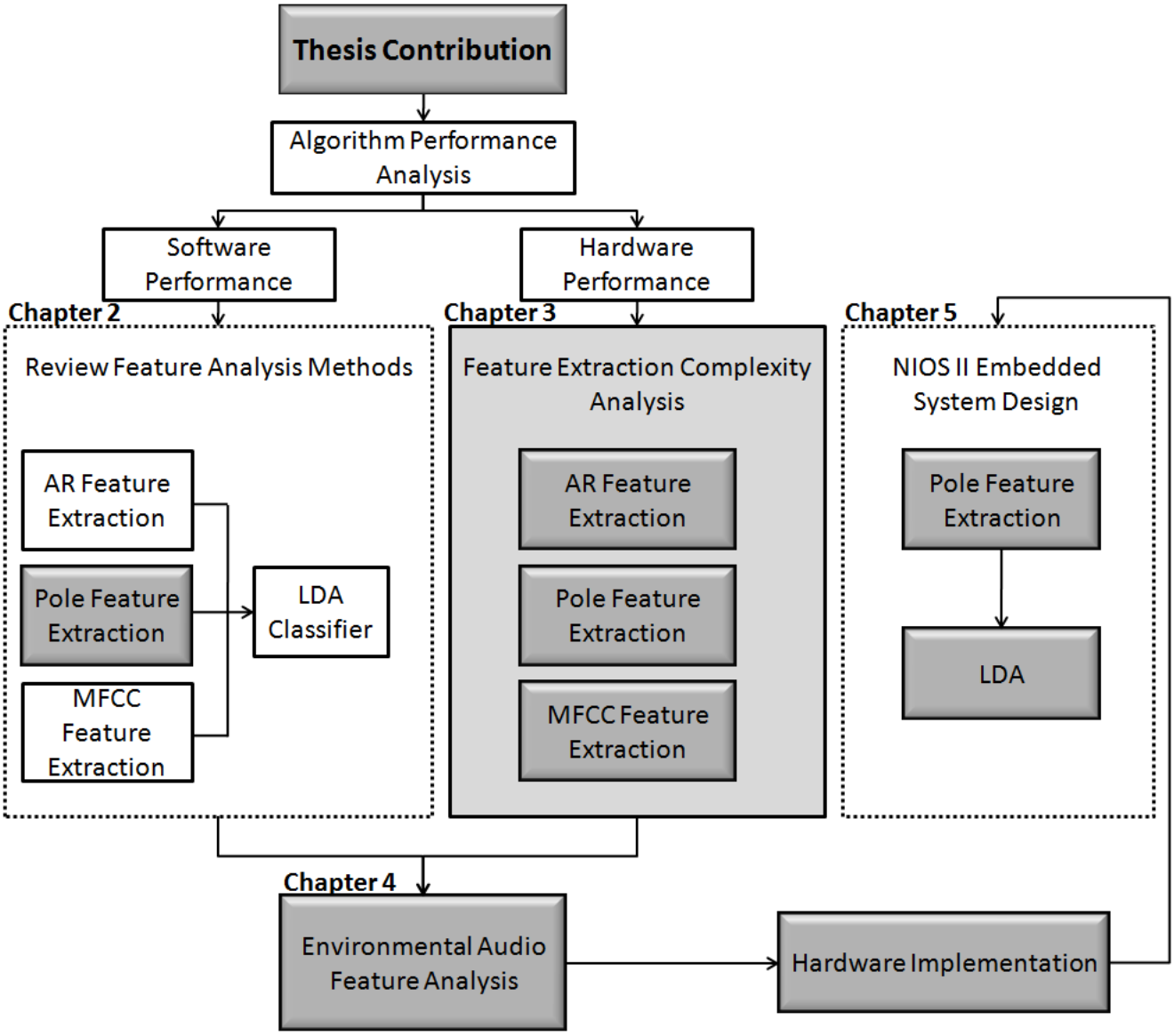

Figure 3.1: Thesis Organization.

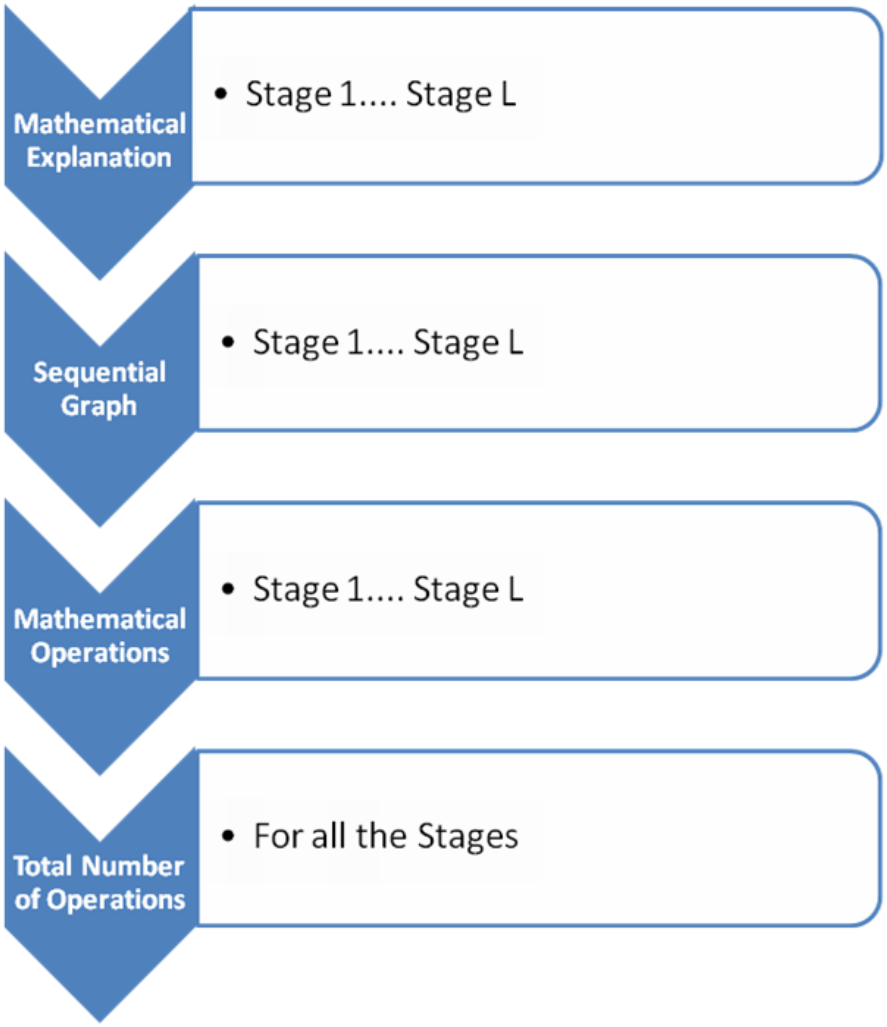

Figure 3.2: Chapter 3 - Computational Complexity Analysis.

flow chart for calculation the AR coefficients. The detailed mathematical analysis used in each step in the flow-chart 3.4 is explained below:

- Initialization of the forward/backward error:

$$f_0(n) = b_0(n) = x(n)$$
$$n = 0, 1, ..., L-1 \tag{3.1}$$
$$a_0 = 1$$

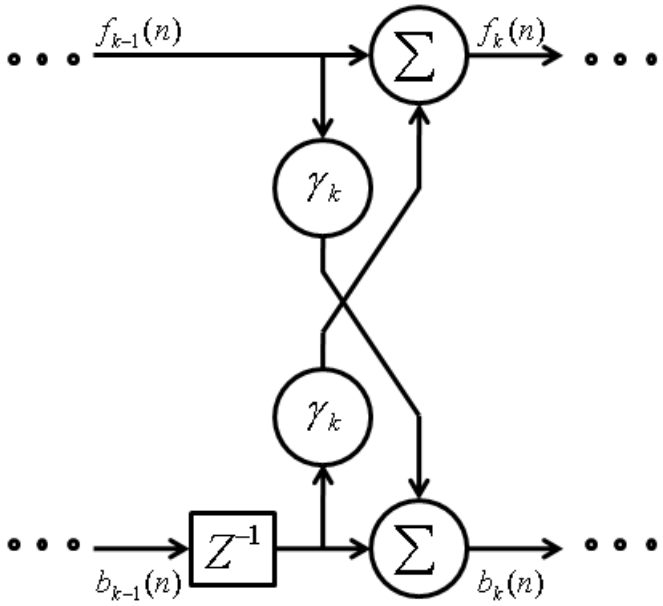

Figure 3.3: The lattice structure for one stage of Burg algorithm.

- Calculation of the reflection coefficient in the Burg algorithm:

$$\gamma_r = 2\frac{\sum_{n=r}^{L-1} f_{r-1}(n)b_{r-1}(n-1)}{\sum_{n=r}^{L-1} f_{r-1}^2(n)+b_{r-1}^2(n-1)} \qquad r = 1, 2, 3, ..., m \tag{3.2}$$

- Calculation of the forward/backward prediction errors:

$$f_r(n) = f_{r-1}(n) - \gamma_r \times b_{r-1}(n-1) \qquad n = r, r+1, ..., L-1 \tag{3.3}$$

$$b_r(n) = b_{r-1}(n-1) - \gamma_r \times f_{r-1}(n) \qquad n = r, r+1, ..., L-1 \tag{3.4}$$

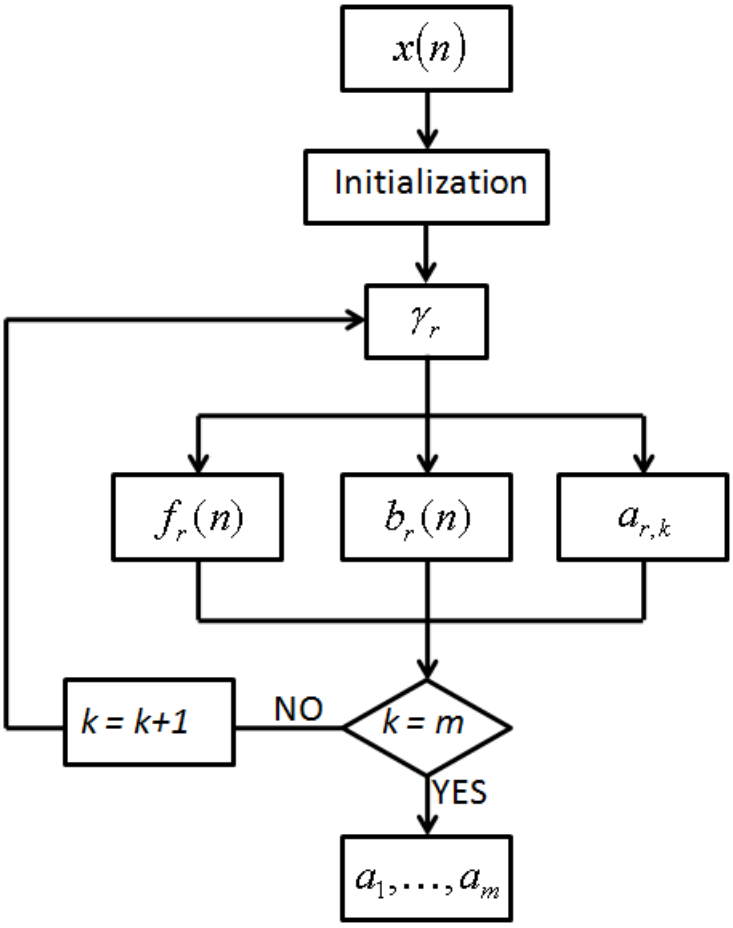

Figure 3.4: Flow chart of calculation the AR coefficients using Burg algorithm.

- Finally, calculation of the AR Coefficients:

$$
\begin{aligned}
a_{r,0} &= 1 \\
a_{r,r} &= -\gamma_r \\
a_{r,k} &= a_{r-1,k} - \gamma_r \times a_{r-1,r-k} \\
r &= 2, 3, ..., m \\
k &= 1, 2, ..., r-1
\end{aligned}
\tag{3.5}
$$

3.2.1 Sequencing Graph and Number of operations

Sequencing graphs is plotted for three computational stage as follows:

Stage 1: Stage one is calculating the reflection coefficient in the Burg algorithm. Fig. 3.5 shows the SG for this step. The reflection coefficient needs to be calculated for each iteration of the

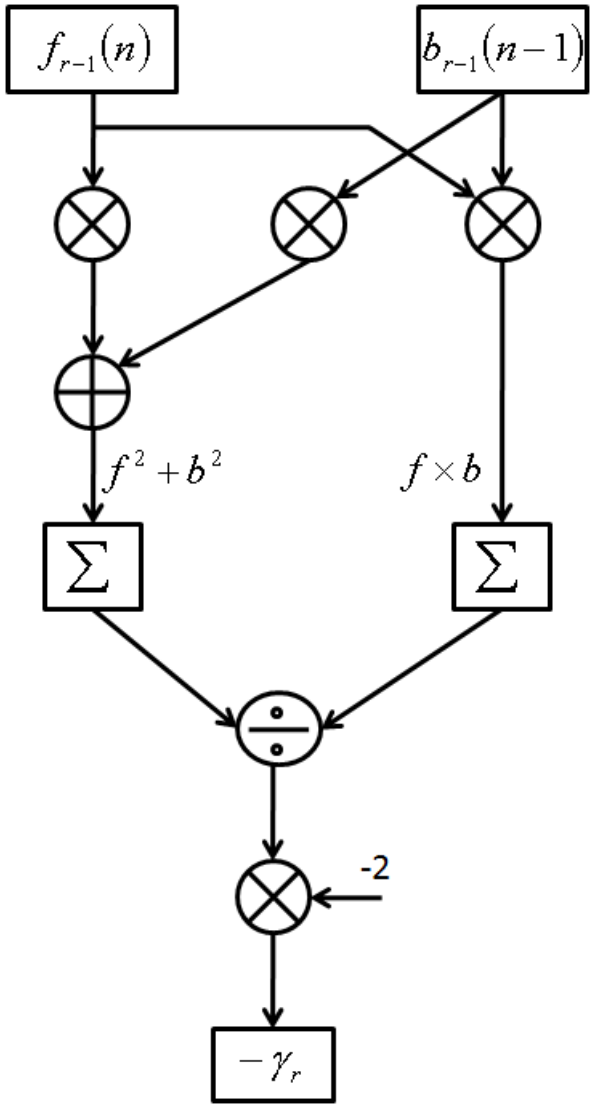

Figure 3.5: SG of the reflection coefficient calculation.

algorithm and each signal segment. This means that for each n in the iteration (r), γ_r is computed from the forward error $f_{r-1}(n)$ and backward error $b_{r-1}(n-1)$. Assuming P is the maximum number of iteration (ie. the AR model order) and N is the maximum number of samples in each segment of the input signal, operations required for this step are calculated using the SG shown in

Fig. 3.5 as follows:

$$\begin{aligned} &Number of multipliers : \sum_{r=1}^{m} [3 \times (L-1-r)+1] \\ &Number of adders : \sum_{r=1}^{m} (L-1-r) \\ &Number of divisions : m \end{aligned} \tag{3.6}$$

Stage 2: Fig. 3.6) shows the SG for calculating forward and backward errors in each iteration of the AR algorithm. The forward error $f_r(n)$ and the backward error $b_r(n)$ are calculated based

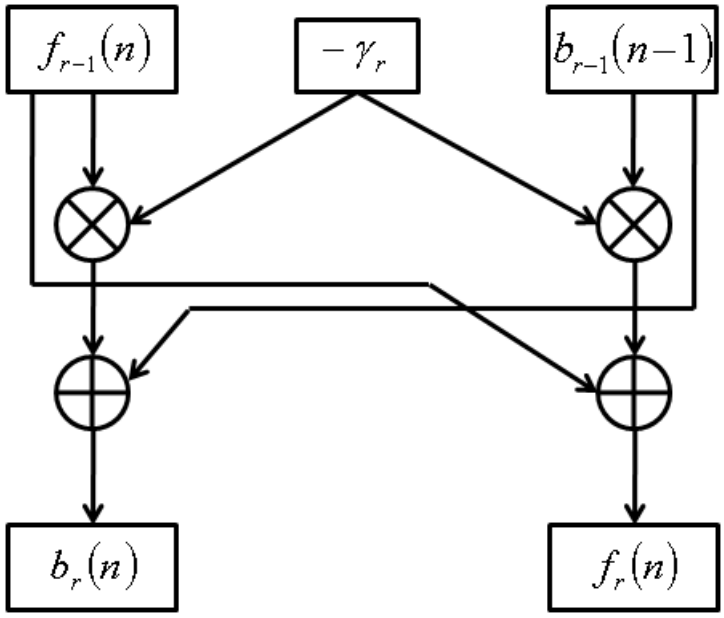

Figure 3.6: SG representing the forward/backward prediction errors calculation.

on γ_r which is calculated in the previous step and the forward error $f_{r-1}(n)$ and backward error $b_{r-1}(n)$ from the previous iteration. Based on sequencing graph shown in Fig. 3.6, the operations required for this step are as follows:

$$\begin{aligned} &Number of multipliers : \sum_{r=1}^{m} [2 \times (L-1-r)] \\ &Number of adders : \sum_{r=1}^{m} [2 \times (L-1-r)] \end{aligned} \tag{3.7}$$

Stage 3: Fig. 3.7 shows the sequencing graph for calculating AR coefficients in each iteration of algorithm. The AR coefficients $a_{r,k}$ are calculated based on γ_r and the AR coefficients obtained in

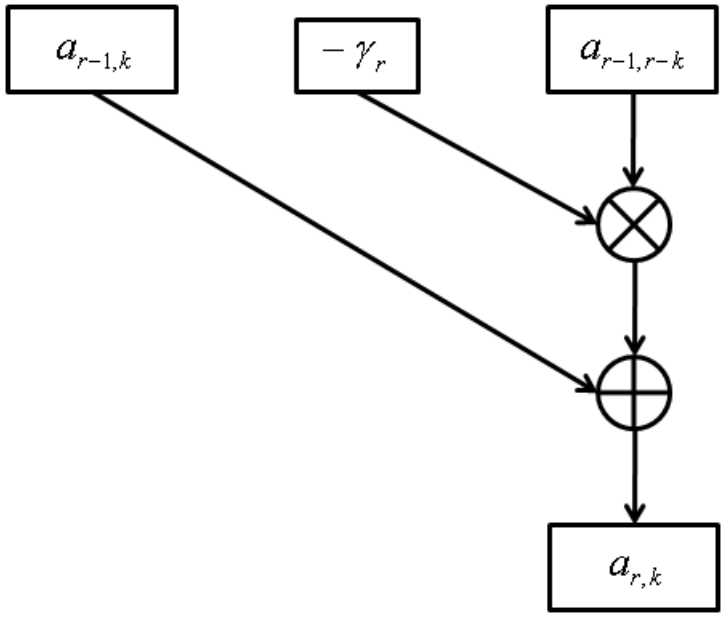

Figure 3.7: SG of the AR coefficients calculation.

the previous iteration. Based on the sequencing graph of Fig. 3.7, operations required for this step are as below:

$$\begin{aligned} &Numberofmultipliers : \textstyle\sum_{r=1}^{m}(r-1) \\ &Numberofadders : \textstyle\sum_{r=1}^{m}(r-1) \end{aligned} \tag{3.8}$$

3.2.2 Total Number of required operations

Total number of mathematical operations required for computing the AR coefficients using Burg algorithm can be extracted based on Eqns. 3.6, 3.7, and 3.8. These operations are listed in Table 3.1.

3.3 Poles from AR Modeling using Eigenvalues of Companion Matrix

As discussed in Chapter 2, finding AR-model poles can become a problem of finding roots of the polynomial $P(z)$ as given below:

$$P(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + ... + z^m \tag{3.9}$$

Operation	Total Number
Multipliers	$\sum_{r=1}^{m}(5L-4r-6)$
Adders	$\sum_{r=1}^{m}(3L-2r-4)$
Divisions	m

Table 3.1: Total number of operations required in AR modeling.

Computing roots of a polynomial can be posed as an eigenvalue problem by forming the companion matrix. The eigenvalues of companion matrix method is an accurate method for computing zeros of a polynomial. The companion matrix A associated with this polynomial is defined as follows:

$$A = \begin{pmatrix} -a_{m-1} & . & . & . & -a_2 & -a_1 & -a_0 \\ 1 & 0 & & & & & 0 \\ & 1 & 0 & & & & . \\ & & . & & & & . \\ & & & . & & & . \\ & & & & . & & . \\ & & & & & 1 & 0 \end{pmatrix} \tag{3.10}$$

This matrix has the characteristic polynomial as defined below:

$$P_c(z) = det\,(zI - A) = P(z) \tag{3.11}$$

Finding the zeros of Eqn. 3.9 is equivalent to computing the eigenvalues of matrix A in Eqn. 3.10. The eigenvalues of matrix A can be found using the QR algorithm. Fig. 3.8 is the flow chart of AR model poles calculation using eigenvalues of companion matrix. Fig. 3.8 shows that QR algorithm is deployed to compute the eigenvalues of the companion matrix. The classic QR algorithm performs a QR decomposition or factorization to factorize the matrix A_i as a product of an orthogonal matrix Q_i and an upper triangular matrix R_i. In each iteration of QR algorithm, matrix A_i is factorized to matrix Q_i and R_i using QR factorization. Then matrix A_{i+1} is formed

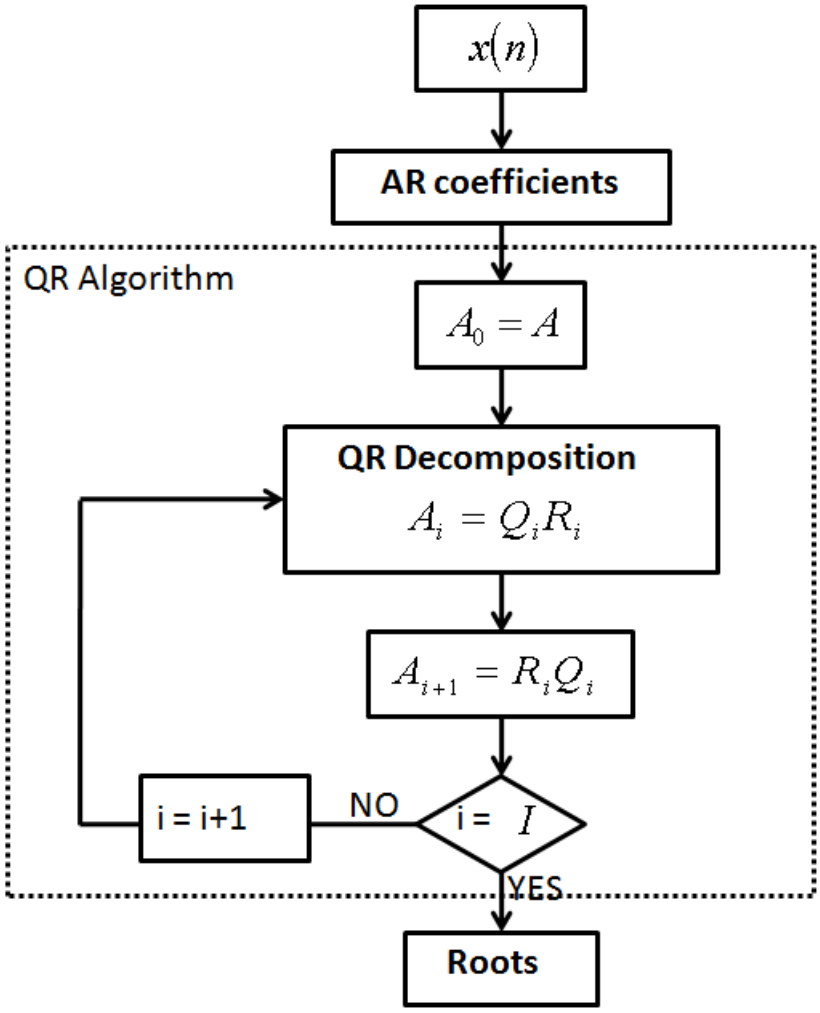

Figure 3.8: AR model poles calculation using eigenvalues of companion matrix.

by multiplying the factors in the following order:

$$A_{i+1} = R_i \times Q_i \tag{3.12}$$

After several iterations, matrix A_i is converted to a triangular matrix, and the diagonal elements of the matrix converge to the matrix eigenvalues.

3.3.1 QR Factorization

There are several algorithms to perform the QR decomposition of a matrix. For example, Cholesky QR, the Gram-Schmidt process, Givens rotations, or Householder reflectors [54]. Most of general-

purpose software for QR uses either Givens rotations or Householder reflectors as two important transformation techniques. One advantage of Givens rotations method over Householder transformations is that they can easily be parallelized, and often for very sparse matrices they have a lower operation count. In this thesis Givens rotations is used for performing the QR decomposition.

Companion matrix A is an upper Hessenberg form and it is already nearly upper-triangular. By applying Givens rotation technique and zeroing out the only one nonzero entry below each diagonal element, matrix A becomes a triangular matrix. The Givens rotation matrices $G_1, G_2, G_3, ..., G_{m-1}$ are constructed so that:

$$G_{m-1} \times G_{m-2}...G_2 \times G_1 \times A = R \tag{3.13}$$

As it is demonstrated below, the orthogonal matrix Q^T is formed from the concatenation of all the Givens matrices :

$$\begin{aligned} Q &= G_1^T \times G_2^T ... \times G_{m-2}^T \times G_{m-1}^T \\ Q^T &= G_{m-1} \times G_{m-2}...G_2 \times G_1 \end{aligned} \tag{3.14}$$

The two Eqns. 3.14 and 3.13 are combined as it is shown below:

$$Q^T A = R \tag{3.15}$$

and then:

$$\begin{aligned} QQ^T A &= QR \\ IA &= QR \\ A &= QR \end{aligned} \tag{3.16}$$

Eqn. 3.16 shows that using Givens rotation method, matrix A can be factorized to Q and R. In factorization the idea is to first find G_1 to zero out the sub-diagonal element below a_{11} which is shown below:

$$A_1 = G_1 A = G_1 \begin{pmatrix} a_{11} & a_{12} & . & . & . & a_{1m} & \\ 1 & 0 & & & & & 0 \\ 0 & 1 & 0 & & & & . \\ . & & . & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ 0 & & & & & 1 & 0 \end{pmatrix} = \begin{pmatrix} \bar{a_{11}} & \bar{a_{12}} & . & . & . & \bar{a_{1m}} & \\ 0 & \bar{a_{22}} & . & . & . & \bar{a_{2m}} & \\ 0 & 1 & 0 & & & & . \\ . & & . & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ 0 & & & & & 1 & 0 \end{pmatrix} \tag{3.17}$$

At the next step G_2 is multiplied by G_1 and the sub-diagonal element in the second column below the $\bar{a_{22}}$ is zeroed :

$$A_2 = G_2 G_1 A = G_2 \begin{pmatrix} \bar{a_{11}} & \bar{a_{12}} & . & . & . & \bar{a_{1m}} \\ 0 & \bar{a_{22}} & . & . & . & \bar{a_{2m}} \\ 0 & 1 & 0 & & & . \\ . & & . & & & . \\ . & & & . & & . \\ . & & & & . & . \\ 0 & & & & 1 & 0 \end{pmatrix} = \begin{pmatrix} \bar{a_{11}} & \bar{a_{12}} & . & . & . & \bar{a_{1m}} \\ 0 & \hat{a_{22}} & . & . & . & \hat{a_{2m}} \\ 0 & 0 & . & . & . & \hat{a_{3m}} \\ . & & . & & & . \\ . & & & . & & . \\ . & & & & . & . \\ 0 & & & & 1 & 0 \end{pmatrix} \tag{3.18}$$

After $m-1$ iterations, matrix A_k is converged to a triangular matrix and the eigenvalues could be found easily from the diagonal elements of the matrix. A Givens rotation matrix to zero out the sub-diagonal element bellow the element $a_{j,j}$ is represented by a matrix of the following form:

$$G(j, j+1, \theta) = \begin{pmatrix} 1 & . & . & . & 0^{1,j} & 0 & . & . & . & 0 \\ . & . & & & . & . & & & & . \\ . & & . & & . & . & & & & . \\ . & & & . & . & . & & & & . \\ 0^{j,1} & . & . & . & c & s & . & . & . & 0 \\ 0 & . & . & . & -s & c & . & . & . & 0 \\ . & & & & . & . & . & & & . \\ . & & & & . & . & & . & & . \\ . & & & & . & . & & & . & . \\ 0 & . & . & . & 0 & 0 & . & . & . & 1 \end{pmatrix} \tag{3.19}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. As the name Givens rotation indicates, $G^T(j, j+1, \theta) \times X$ is a counterclockwise rotation of the vector X in the $(j, j+1)$ plane of θ radians. Givens rotations are clearly orthogonal matrices. Considering $a^{j,j}$ is a diagonal element of matrix A and $b^{j+1,j}$ is a sub-diagonal element of matrix A. Givens rotations technique zeros out the $b^{j+1,j}$ by using this function:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{3.20}$$

where:

$$\begin{aligned} r &= \sqrt{a^2 + b^2} \\ c &= \tfrac{a}{\sqrt{a^2+b^2}} \\ s &= \tfrac{b}{\sqrt{a^2+b^2}} \end{aligned} \tag{3.21}$$

Here the detail algorithm to perform QR factorization is explained. Assuming matrix A is an upper Hessenberg matrix, then matrix A can be factorized as $A = QR$ where Q is orthogonal and R is upper triangular.

For $j = 1 : m - 1$

$$c = \frac{A(j,j)}{\sqrt{A(j,j)^2 + A(j+1,j)^2}}$$
$$s = \frac{A(j+1,j)}{\sqrt{A(j,j)^2 + A(j+1,j)^2}}$$

$$A(j : j+1, j : m) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A(j : j+1, j : m)$$

$$\begin{aligned} G_j(j,j) &= c \\ G_j(j+1,j+1) &= c \\ G_j(j,j+1) &= s \\ G_j(j+1,j) &= -s \end{aligned} \tag{3.22}$$

End

$$R = A$$

$$Q = G_1^T \times G_2^T ... \times G_{m-2}^T \times G_{m-1}^T$$

$Q = G_1^T \times G_2^T ... \times G_{m-2}^T \times G_{m-1}^T$ is a product of Givens rotations $G_j(j, j+1, \theta)$. These algorithm is applied to companion matrix A from Eqn. 3.10 which is an upper Hessenberg matrix. In a companion matrix all the sub-diagonal elements are $A(j+1, j) = 1$, so the algorithm is

simplified as follows:

$$
\begin{aligned}
&\textbf{For } j = 1 : m-1 \\
&c = \frac{A(j,j)}{\sqrt{A(j,j)^2+1}} \\
&s = \frac{1}{\sqrt{A(j,j)^2+1}} \\
&A(j:j+1, j:m) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A(j:j+1, j:m) \\
&G_j(j,j) = c \\
&G_j(j+1,j+1) = c \\
&G_j(j,j+1) = s \\
&G_j(j+1,j) = -s \\
&\textbf{End} \\
&R = A \\
&Q = G_1^T \times G_2^T ... \times G_{m-2}^T \times G_{m-1}^T
\end{aligned}
\tag{3.23}
$$

The QR algorithm is shown Fig. 3.8; the QR decomposition block in the QR algorithm is replaced by the algorithm which is explained above in Eqn. 3.23. Fig. 3.9 shows the block diagram for QR algorithm using Givens rotations technique to performing the QR factorization.

3.3.2 Sequencing Graph and Number of Operations

Fig. 3.9 shows the flow chart of the QR Algorithm which employs the Givens rotation technique to perform QR decomposition. In this section SG for each step of the algorithm is plotted and number of required mathematical operations are calculated.

Stage 1: First step in each iteration of QR algorithm is over writing matrix A_i using the QR factorization as $A_i = Q_i R_i$ where Q_i is orthogonal and R_i is upper triangular. The QR factorization is an iterative algorithm which starts with $j = 1$ and continues till $j = m - 1$. Each iteration of the QR factorization by itself contains the following three steps:

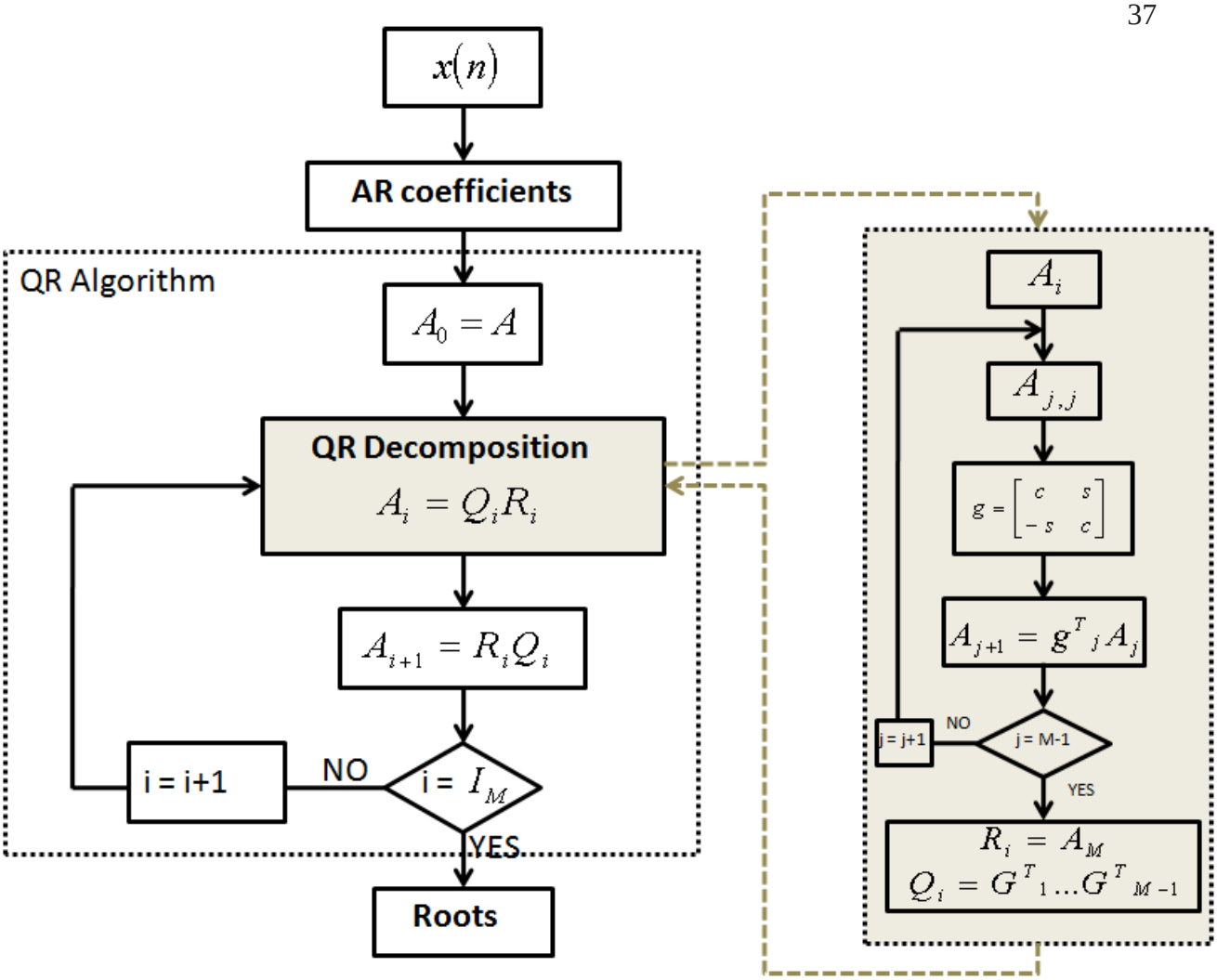

Figure 3.9: QR Algorithm using Givens rotations technique,

QR factorization Step 1: is calculating c_j and s_j. Fig. 3.10 shows the SG for calculating c_j and s_j in each iteration of QR factorization. Based on SG 3.10 number of required mathematical operations for calculate c_j and s_j in each iteration of QR factorization are as follows:

$$\begin{aligned} &Numberofmultipliers : 2 \\ &Numberofadders : 1 \\ &Numberofsquareroots : 1 \\ &Numberofdividers : 1 \end{aligned} \tag{3.24}$$

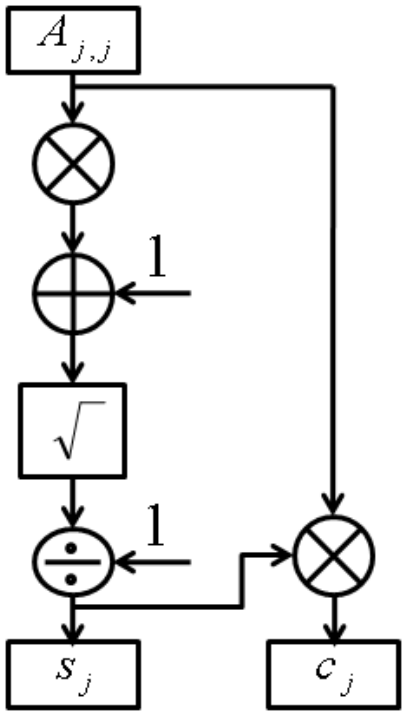

Figure 3.10: Computing c and s

Then total number of operations for calculating c_j and s_j in $m-1$ iterations are:

$$\begin{aligned} &Numberofmultipliers : 2(m-1) \\ &Numberofadders : (m-1) \\ &Numberofsquareroots : (m-1) \\ &Numberofdividers : (m-1) \end{aligned} \tag{3.25}$$

QR factorization Step 2: in QR factorization is calculating new A as follows:

$$\begin{aligned} A(j:j+1, j:m) &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A(j:j+1, j:mM) \\ A(j:j+1, j:m) &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{j,j} & a_{j,j} & . & . & . & a_{j,m} \\ a_{j+1,j} & a_{j+1,j} & . & . & . & a_{j+1,m} \end{bmatrix} \end{aligned} \tag{3.26}$$

Eqn. 3.26 is a matrix multiplication. Matrix g has two rows and two columns. Matrix $A(j : j+1, j : m)$ has two rows and numbers of its column is different in each iteration and equals to $m-j+1$. Number of operation for creating Matrix $A(j : j+1, j : m)$ are:

$$\begin{aligned} Numberofmultipliers &: 4(m-j+1) \\ Numberofadders &: 2(m-j+1) \end{aligned} \tag{3.27}$$

Then total number of operations for creating matrix $A(j : j+1, j : m)$ in $m-1$ iterations are:

$$\begin{aligned} Numberofmultipliers &: 4\sum_{j=1}^{m-1} m-j+1 = 2(m-1)(m+2) \\ Numberofadders &: 2\sum_{j=1}^{m-1} m-j+1 = (m-1)(m+2) \end{aligned} \tag{3.28}$$

QR factorization Step 3: is the last step in QR factorization. In this step product of Givens rotations are calculated $Q = G_1^T G_2^T ... G_{m-1}^T$. This computation is a matrix multiplication. Givens rotation matric $G_j(j, j+1, \theta)$ multiplies by a matrix with m rows and columns as it is presented below:

$$\begin{pmatrix} 1 & . & . & . & 0^{1,j} & 0 & . & . & . & 0 \\ . & . & & & . & . & & & & . \\ . & & . & & . & . & & & & . \\ . & & & . & . & . & & & & . \\ 0^{j,1} & . & . & . & c_j & s_j & . & . & . & 0 \\ 0 & . & . & . & -s_j & c_j & . & . & . & 0 \\ . & & & & . & . & . & & & . \\ . & & & & . & . & & . & & . \\ . & & & & . & . & & & . & . \\ 0 & . & . & . & 0 & 0 & . & . & . & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & . & . & . & a_{1m} \\ a_{21} & a_{22} & . & . & . & a_{2m} \\ a_{31} & a_{32} & . & . & . & a_{3m} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & . & a_{mm} \end{pmatrix} \tag{3.29}$$

Number of operation required to perform this matrix multiplication are:

$$\begin{aligned} Numberofmultipliers &: 4m \\ Numberofadders &: 2m \end{aligned} \tag{3.30}$$

Then number of total required operations for calculating Q in $m-1$ iteration are:

$$\begin{aligned} Numberofmultipliers &: 4m(m-1) \\ Numberofadders &: 2m(m-1) \end{aligned} \tag{3.31}$$

Based on Eqns. 3.25, 3.28 and 3.31 total number of required mathematical operation for performing the QR factorization in one iteration of QR algorithm are as follows:

$$\begin{aligned} &Numberofmultipliers : 2(m-1)(3m+1) \\ &Numberofadders : (m-1)(3m+1) \\ &Numberofsquareroots : (m-1) \\ &Numberofdividers : (m-1) \end{aligned} \tag{3.32}$$

Stage 2: Next step of QR algorithm is calculating the new matrix $A_{i+1} = R_i Q_i$ where Q_i is orthogonal and R_i is upper triangular. Also it is confirmed in [54] that matrix $Q = G_1^T G_2^T ... G_{m-1}^T$ is upper Hessenberg if $G_1, G_2, ..., G_{m-1}$ are the Givens rotations. So calculating A_{i+1} contains multiplication of the matrix R_i which is upper triangular by matrix Q_i which is upper Hessenberg, as it is shown below:

$$\begin{pmatrix} a_{11} & a_{12} & . & . & . & a_{1m} \\ 0 & a_{22} & . & . & . & a_{2m} \\ 0 & 0 & . & . & . & a_{3m} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & . & . & 0 & a_{m,m} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & . & . & . & a_{1m} \\ a_{21} & a_{22} & . & . & . & a_{2m} \\ 0 & a_{32} & . & . & . & a_{3m} \\ . & 0 & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & . & . & a_{m,m-1} & a_{m,m} \end{pmatrix} \tag{3.33}$$

Then number of operations required for creating A_{i+1} are as follows:

$$\begin{aligned} &Numberofmultipliers : \tfrac{m^2(m+3)}{4} \\ &Numberofadders : \tfrac{m(m-1)(m+2)}{4} \end{aligned} \tag{3.34}$$

3.3.3 Total Number of required operations

Total number of required mathematical operations to perform QR algorithm using Givens rotation technique are calculated based on Eqns. 3.32 and 3.34. These operations are listed in Table 3.2.

3.4 Mel Frequency Cepstral Coefficients (MFCC)

The MFCC computation for a given signal consists of the following steps:

Operation	Total Number
Multipliers	$I\left(2(m-1)(3m+1)+\frac{m^2(m+3)}{4}\right)$
Adders	$I\left((m-1)(3m+1)+\frac{m(m-1)(m+2)}{4}\right)$
Divisions	$I(m-1)$
Square Roots	$I(m-1)$

Table 3.2: Total number of operations required in pole modeling.

- Construct the filter bank with M equal height triangular filters $H_i(K)$ based on Mel-scaled frequency from Eqn. 2.17,
- Transform the input signal $X(n)$ from time domain to frequency domain $X(K)$ by applying DFT,
- Find the energy spectrum $|X(K)|$,
- Calculate the energy in each channel$\sum_{K=0}^{L-1} |X(K)| \times H_i(K)$,
- Proceed with logarithm and cosine transforms,

In the next section SG for each stage of the algorithm is plotted and number of required mathematical operations are calculated.

3.4.1 Sequencing Graph and Number of Operations

Stage 1: Stage one in computing MFCC is transforming the input signal from time domain to frequency domain by applying discrete Fourier Transform (DFT). Let $x(0), ..., x(L-1)$ be a vector of input signal samples. Consider L as the frame size or number of samples in each segment. Then

DFT is defined by the formula:

$$X(k) = \sum_{n=0}^{L-1} x(n) e^{-\frac{2i\pi}{L} kn} \qquad k = 0, ..., L-1 \tag{3.35}$$

Consider $W_L = e^{-\frac{2i\pi}{L}}$ is the primitive Lth root of unity, DFT is presented as an L-by-L matrix multiplication as follows:

$$X(k) = \sum_{n=0}^{L-1} x(n) W_L^{kn} \qquad k = 0, ..., L-1 \tag{3.36}$$

$$\begin{pmatrix} X(1) \\ X(2) \\ X(3) \\ . \\ . \\ . \\ X(L-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & . & . & . & 1 \\ 1 & W_L & W_L^2 & . & . & . & W_L^{L-1} \\ 1 & W_L^2 & W_L^4 & . & . & . & W_L^{2(L-1)} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 1 & W_L^{(L-1)} & W_L^{2(L-1)} & . & . & . & W_L^{(L-1)(L-1)} \end{pmatrix} \times \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ . \\ . \\ . \\ x(L-1) \end{pmatrix} \tag{3.37}$$

The direct implementation of the Eqn. 3.37 requires order of L^2 complex multiplications and additions. The direct evaluation of DFT involves the following operations counts:

$$\begin{aligned} &Number of multipliers : L^2 \\ &Number of adders : L(L-1) \end{aligned} \tag{3.38}$$

Stage 2: Second step in MFCC is calculating the energy spectrum as follows:

$$|X(k)| \qquad k = 0, ..., L-1 \tag{3.39}$$

Number of required mathematical operations for calculating energy spectrum is:

$$\begin{aligned} &Number of multipliers : L \\ &Number of square roots : L \end{aligned} \tag{3.40}$$

Stage 3: In the next step, for each channel of Mel scale filter, the energy of signal is calculated. It is assumed that construction of the filter bank with b equal height triangular filters $H_i(K)$ has been done once and it is known during all the iterations. Number of mathematical operations required for creating Mel scale filter bank does not count in this thesis. b is considered as the number of Mel

windows in Mel scale, which usually varies from 20 to 24. H_i is the triangular filter associated with the ith channel in Mel scale:

$$H_i(K) \quad k = 0, ..., L-1 \quad and \quad i = 0, ..., b-1 \tag{3.41}$$

The energy of signal is calculated by the formula which is presented below:

$$S(i) = \sum_{K=0}^{L-1} |X(K)| \times H_i(K) \qquad i = 0, ..., m-1 \tag{3.42}$$

For the convenience of estimation Eqn. (3.42) is expressed in a matrix form:

$$\begin{pmatrix} S(0) \\ S(1) \\ S(2) \\ . \\ . \\ . \\ S(m-1) \end{pmatrix} = \begin{pmatrix} H_0(0) & . & . & . & H_0(L-1) \\ H_1(0) & . & . & . & H_1(L-1) \\ H_2(0) & . & . & . & H_2(L-1) \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ H_{b-1}(0) & . & . & . & H_{b-1}(L-1) \end{pmatrix} \times \begin{pmatrix} |X(0)| \\ |X(1)| \\ |X(2)| \\ . \\ . \\ . \\ |X(L-1)| \end{pmatrix} \tag{3.43}$$

Number of required mathematical operations for this calculation is:

$$\begin{gathered} Number of multipliers : b \times L \\ Number of additions : b \times (L-1) \end{gathered} \tag{3.44}$$

Stage 4: Finally by proceeding with logarithm and cosine transforms, the MFCCs are computed as follow:

$$C(l) = \sum_{i=0}^{b-1} \text{Log}(S(i)) \times \text{Cos}\left[l(i+05)\tfrac{\pi}{b}\right] \qquad l = 0, ..., m-1 \tag{3.45}$$

m is the desired order of MFCC. Again for the convenience of estimation equation (3.45) is expressed in a matrix form as follows:

$$\begin{gathered} F(l,i) = \text{Cos}\left[l(i+05)\tfrac{\pi}{b}\right] \qquad i = 0, ..., b-1 \quad and \quad l = 0, ..., m-1 \\ C(l) = \sum_{i=0}^{b-1} \text{Log}(S(i)) \times F(l,i) \quad i = 0, ..., b-1 \quad and \quad l = 0, ..., m-1 \end{gathered} \tag{3.46}$$

$$\begin{pmatrix} C(1) \\ C(2) \\ C(3) \\ . \\ . \\ . \\ C(m-1) \end{pmatrix} = \begin{pmatrix} F(0,0) & . & . & . & F(0,b-1) \\ F(1,0) & . & . & . & F(1,b-1) \\ F(2,0) & . & . & . & F(2,b-1) \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ F(m-1,0) & . & . & . & F(m-1,b-1) \end{pmatrix} \times \begin{pmatrix} \text{Log}\,(S(0)) \\ \text{Log}\,(S(1)) \\ \text{Log}\,(S(2)) \\ . \\ . \\ . \\ \text{Log}\,(S(b-1)) \end{pmatrix} \tag{3.47}$$

Operation	Total Number
Multipliers	$L^2+L+b\times L+m\times b$
Adders	$L(L-1)+b(L-1)+m(b-1)$
Logarithms	b
Square Roots	L

Table 3.3: Total number of operations required in MFCC.

Number of required mathematical operations for the last stage are:

$$\begin{aligned} &Numberofmultipliers : m \times b \\ &Numberofadditions : m \times (b-1) \\ &Numberoflogarithms : b \end{aligned} \tag{3.48}$$

3.4.2 Total Number of required operations

Total number of mathematical operations required to calculate MFCCs is extracted as below based on Eqns. 3.38, 3.40, 3.44 and 3.48. These operations are listed in Table 3.3.

3.5 Summary

In this Chapter the computational complexity of AR modeling, pole modeling and MFCC are analyzed in detail using the mathematical equations of each method. Each algorithm is divided into its consisting mathematical stages and the number of required mathematical operations is obtained. Next Chapter will estimate the total number of LUT needed for the implementation of AR, pole modeling, and MFCC on the Altera Cyclone FPGA, based on the complexity of each

method calculated in this Chapter. The algorithmic and hardware performances of these three feature analysis methods will be compared in the next Chapter as well.

Chapter 4

Application: Audio Environment Scene Analysis

4.1 Introduction

The algorithmic explanation of three well-known feature extraction methods (ie. AR, pole, and MFCC) were provided in Chapter 2. Chapter 3 computed the complexity of AR modeling, pole modeling, and MFCC for Hardware analysis. As shown in Fig. 4.1, the present Chapter evaluates and compares the algorithmic and hardware performances of AR, Pole, and MFCC feature analyses for environmental audio scene analysis application. This comparison is the first known work presented to the best of the author's knowledge.

This Chapter is continued as follows: first the background and literature review of audio scene analysis is presented. Next, the database used in the present evaluation is explained, followed by the properties of the employed classifier and feature extraction methods. Finally, both algorithmic and hardware performances of these three feature analysis methods are provided and compared. The most efficient feature analysis method is selected for the NIOS II implementation in the next Chapter.

Audio feature extraction and classification are important tools for audio signal analysis in many applications, such as multimedia indexing and retrieval, and auditory scene analysis. However, due to the non-stationarities and discontinuities exist in these signals, their quantification and classification remains a formidable challenge. The general methodology of audio classification involves

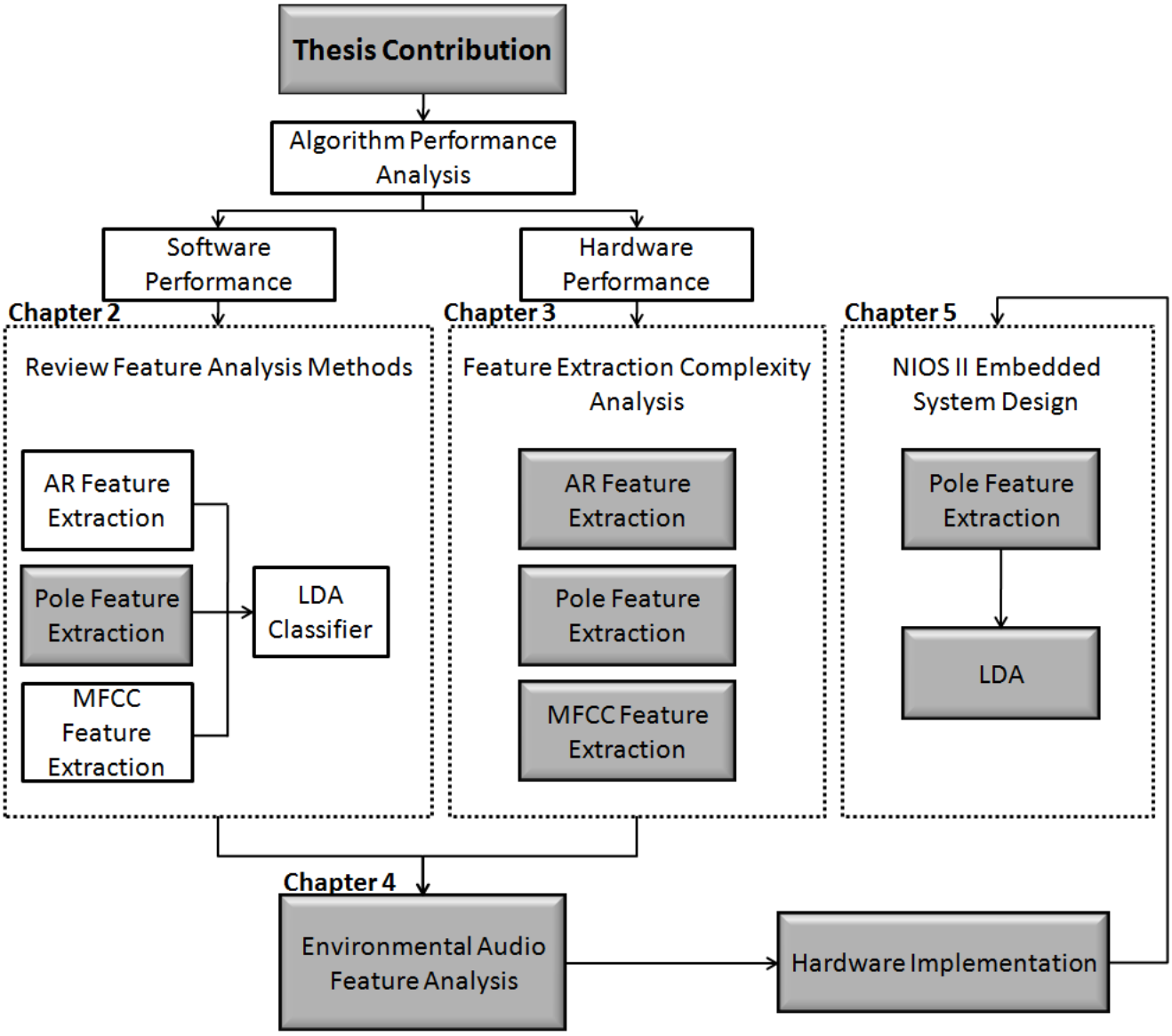

Figure 4.1: Chapter 4 - Environmental Audio Scene Analysis.

extracting discriminatory features from the audio data and feeding them into a pattern classifier. The better and more effective features are extracted from audio signals, the higher performance will be achieved in the audio classification technique. Therefore, the specific aim of this Chapter is to study audio environment classification accuracy vs. hardware cost problem.

4.2 Audio Classification

Audio signals are important sources of information for understanding the content of multimedia. Therefore, developing audio classification techniques that better characterize audio signals plays an essential role in many multimedia applications, such as, multimedia indexing and retrieval, and auditory scene analysis. In multimedia indexing and retrieval, audio classification is used along with other kinds of medium to classify the content of the multimedia. In auditory scene analysis, audio classification is used to distinguish between different environmental sounds. For example, in an efficient hearing aids (HA), in order to improve the quality of the audio for hearing impaired people, before amplifying the audio signals, the HA device uses an audio classifier to distinguish between different environmental sounds, such as speech, music and noise, and then the signals are amplified accordingly [55].

Another application of audio classification is in audio-visual signal processing. One of the examples of audio-visual processing is controlling the observation of a driver which is an application of Multimedia for human's safety purposes. By means of auditory scene detection and the visual data, the watchfulness of the driver can be detected to avoid accidents due to lack of concentration. Audio classification and analysis also help in retrieving the accurate information from the digital media. Development of powerful audio effect processors such as EMU10K1 [56] for the personal computers has made it possible for criminals to create a fake evidence by adding environmental simulation, 3-D positioning, and special effects to audio to create environment adaptations such as train, plane, public place or etc. Determination of the authenticity of the speaker's environment can play a substantial role in investigations of the collected multimedia evidence to prove a crime [57].

4.3 Audio Environment Database

The performance of a classification procedure depends on the design parameters, diversity of the classifiers, as well as the assessment database which are usually different in each analysis method. This fact, together with non-availability of databases or algorithms for other works, makes comparison of methods a difficult task. In literature, although some techniques reported high accuracy rates, they used a few audio groups in the evaluation stage. For example, in [58], the authors use two classes (i.e., speech and music) and achieve 95% accuracy rate, while audio content analysis at Microsoft research [30] uses three audio classes (i.e., speech, music, and environment sound) with 96.5%. Freeman et al [59] use four classes of speech (i.e., babble, traffic noise, typing, and white noise), and achieve 97.9% accuracy rates using artificial neural networks (ANN). The authors in [27] obtain a lower accuracy rate (82.3%) for classification of 14 different environmental scenes (i.e., inside restaurants, playground, street traffic, train passing, inside moving vehicles, inside casinos, street with police car siren, street with ambulance siren, nature-daytime, nature-nighttime, ocean waves, running water, raining, and thundering). Overall, when more diverse signal types are defined in the evaluation stage, the classification accuracy tends to be reduced.

The focus of the present study is to evaluate the performance of audio classification for human and non-human classification. For this purpose, an audio database is used which is developed in the signal analysis research (SAR) group at Ryerson University [32], and contains different environmental audio signals. This database used in this experiment consists of 80 audio signals of 5 s duration each with a sampling rate of 22.05 kHz and a resolution of 16 bits/sample. The arrangement of this database is shown in Fig. 4.2. It is designed to have different signal types including 8 aircraft, 8 helicopters, 8 drums, 8 flutes, 8 pianos, and the speech of 20 males and 20 females. Most of the audio samples were collected from the Internet and suitably processed to have uniform sampling frequency and duration.

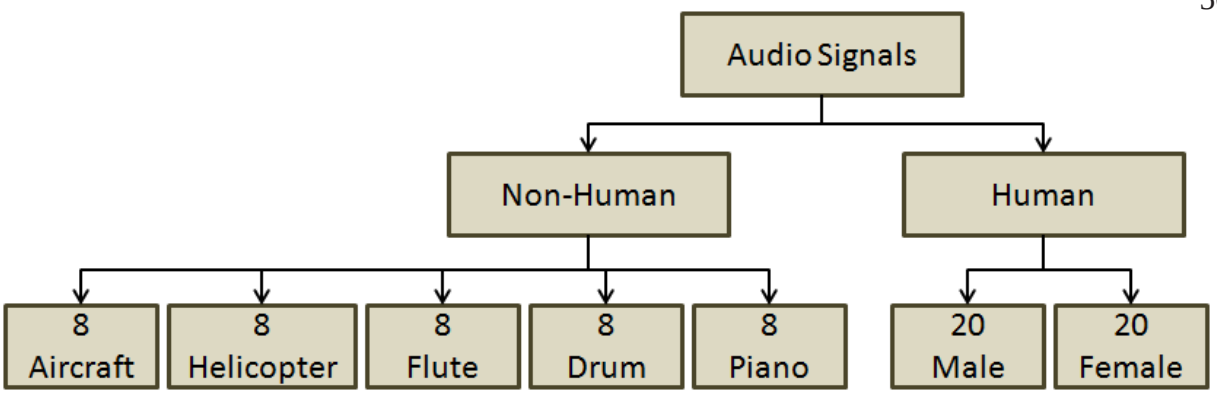

Figure 4.2: Organization of audio database used in this work.

4.4 Audio Classifiers

Various classifiers have been utilized for audio classification. Audio content analysis at Microsoft research commonly uses Gaussian mixture models (GMM) [29], k-nearest neighborhood (K-NN) [30] and support vector machine (SVM) [31] for audio classification. Other popular classifiers for audio classification include linear discriminant analysis (LDA) [32], hidden Markov models (HMM) [33] and artificial neural networks (ANN) [59]. There are some works that focus attention on developing new classifiers, or comparing existing classifiers for audio classification applications. For instance, in [55], Buchler et. al. compare simple classifiers (e.g., rule-based and minimum-distance classifiers) with complex approaches (e.g., Bayes classifier, neural network and hidden Markov model). While these studies are beneficial, the aim of the present study focuses on performance evaluation of three well-known feature extraction methods. Therefore, in this Chapter, we avoid complex classifiers and apply LDA as a simple linear classifier to evaluate the features.

4.5 Audio Features

Over the last few years, several audio feature extraction techniques have been introduced. In general, all the feature extraction methods utilize one of the following three signal representation

domains: temporal domain, spectral or joint time-frequency (TF) distribution. *Temporal domain features*, such as, signal energy, pitch, zero crossing rate [16, 17] and Entropy modulation [18] have been used for audio classification. Examples of *Spectral features* include 4 Hz modulation energy, percentage of low-energy frames, spectral rolloff point, spectral centroid, mean frequency, cepstral coefficients [19, 20], and high and low frequency slopes [21]. Additionally, the variation in the structural behavior of audio signals from one frame to the adjacent frames motivated the development of several features to characterize the existing spectral difference between the neighboring frames. For example, spectrum flux (SF) [19] is defined as the average variation value of spectrum between the two adjacent frames. There have been some attempts to derive joint *TF features* [60, 61, 62, 27, 28] from audio signals. Although, a significant attention is paid to the feature extraction techniques as mentioned above, the hardware implementation of these methods has not been explicitly studied in the literature. In this Chapter, we investigate the hardware and software performances of three well-known features: AR, Pole, and MFCC features. The analytical explanation of these features are explained in Chapters 2 and 3. The following Chapter explains the details of these features as employed in the developed audio classification system.

AR Features: Fig. 4.3 demonstrates the AR feature extraction as used in this analysis. Each audio signal is divided into segments with duration of 25ms. Sine the signal duration is 5s, 200 segments are derived for each signal ($k = 200$ in Fig. 4.3). AR features are extracted from each segment. These AR features are $a_1, \ldots a_m$ in Eqn. 2.7. In this application AR model order of 13 is used which means that 13 AR audio features are extracted ($m = 13$). There are 80 signals in this database ($N = 80$). Therefore, the AR feature vectors are (80×200) vectors with 13 features in each vector.

Pole Features: The Pole feature extraction is shown in Fig. 4.4. Similar to the AR feature extraction, each audio signal is divided into 200 segments with duration of 25ms. Once 13 AR features are extracted from each segment, 13 poles are computed for each segment. Therefore, the polynomial in Eqn. 3.9 is of order 13, and the associated companion matrix in Eqn. 3.10 is a 13×13 matrix. To find the poles (ie. the eigenvalues of the companion matrix), the QR factorization is

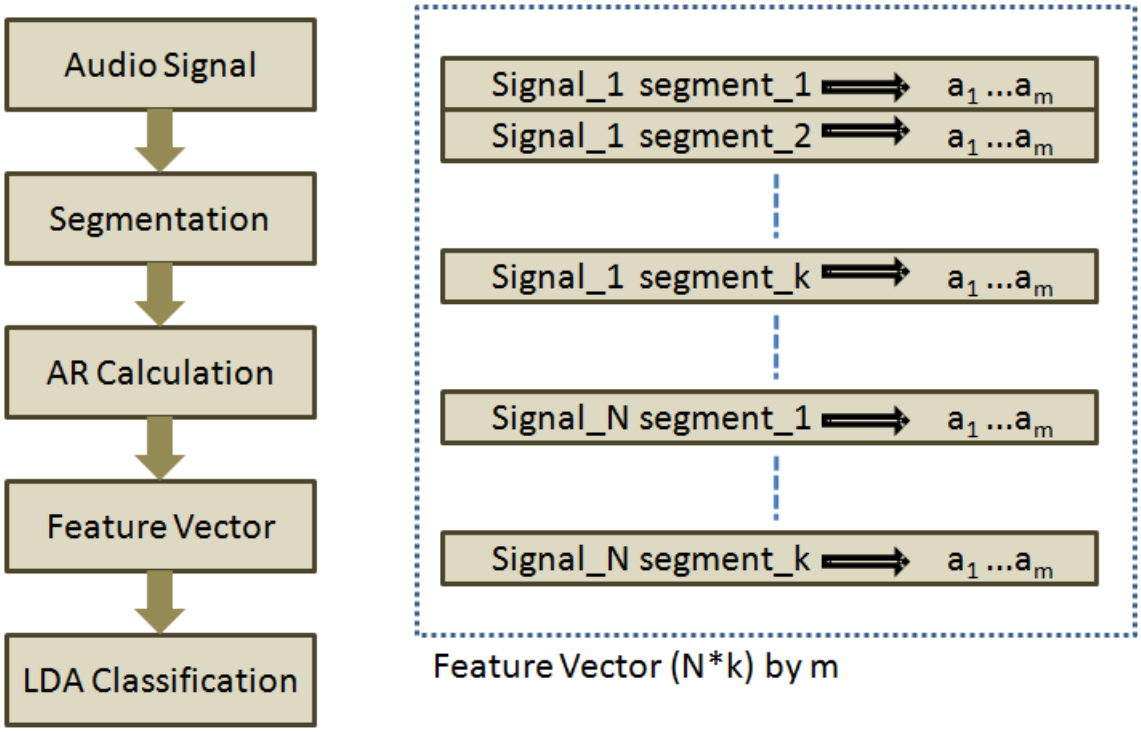

Figure 4.3: General schematic of the AR audio feature extraction.

repeated 13 times. Next, two features are extracted for each pole: the spectral bandwidth and the angle as explained in Eqns. 2.15 and 2.16, respectively. The pole features contain (80×200) vectors with 2×13 features in each vector.

MFCC Features: 13 MFCC features are extracted as explained in Section 2.4. Fig. 4.5 shows that the MFCC features contain (80×200) vectors with 13 features in each vector. In this analysis, 20 mel scale filter bank is used for calculation of the MFCC features.

4.6 Results

As mentioned in Chapter 1, a comprehensive performance evaluation should be based on the accuracy rate of a method as well as the hardware analysis. Therefore, the first part presents the accuracy evaluation of the three audio features as explained in previous sections. The second part compares these methods in terms of their hardware efficiency.

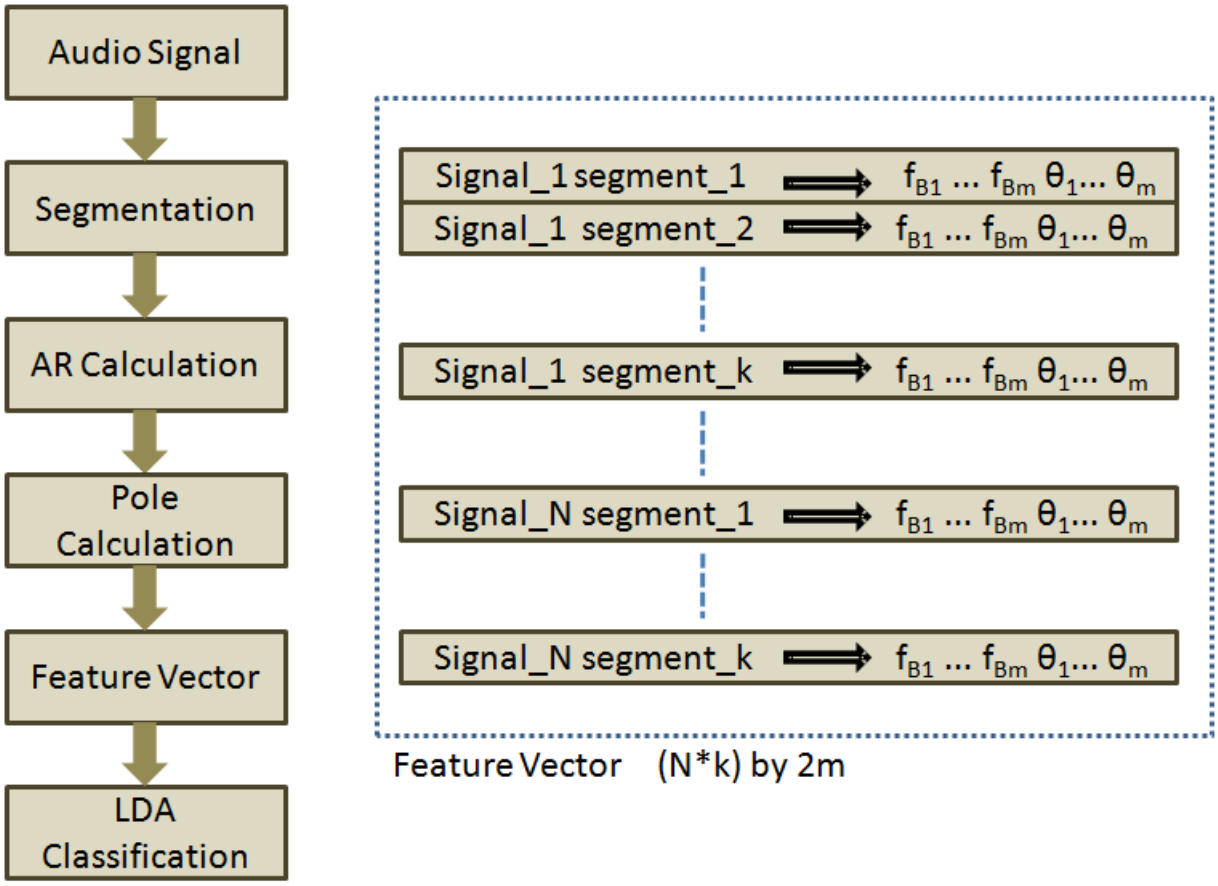

Figure 4.4: General schematic of the pole audio feature extraction.

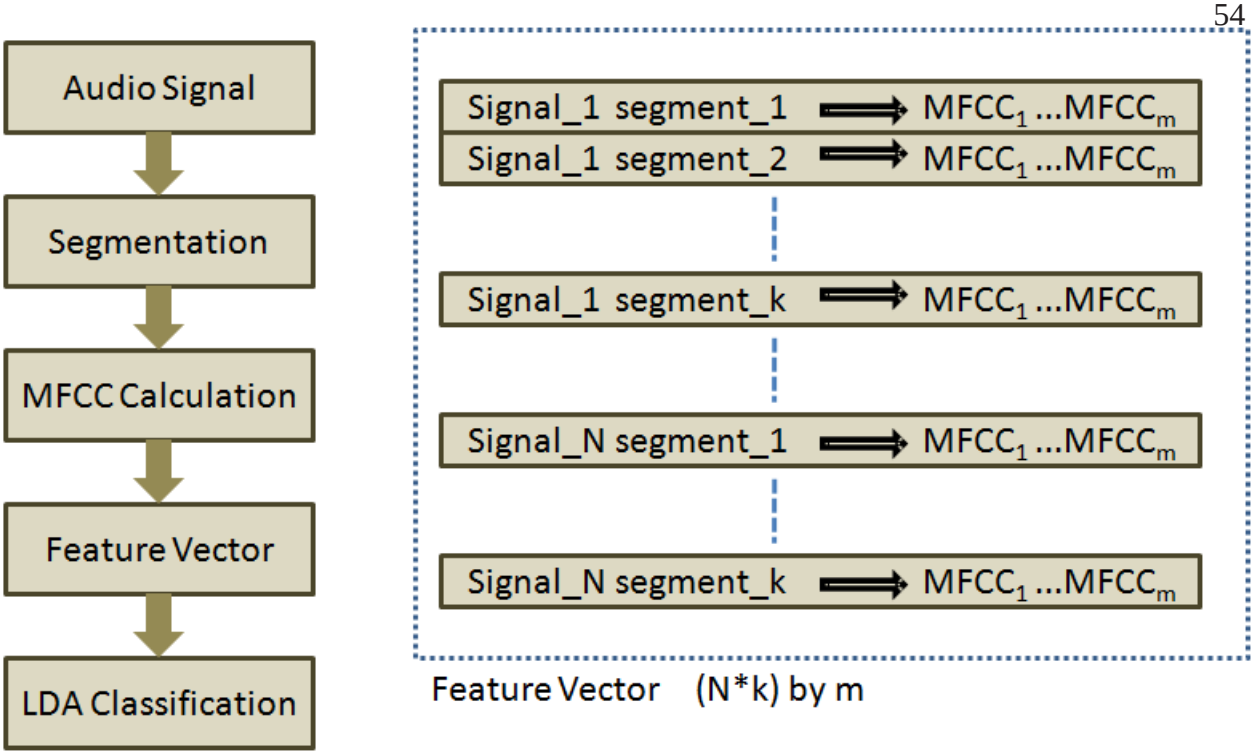

Figure 4.5: General schematic of the MFCC audio feature extraction.

4.6.1 Algorithm Performance

An audio classification is performed in MATLAB as follows: 1) First, all the 80 audio signals are transformed into AR, pole, and MFCC features. Next, the extracted features are fed into the classifier based on LDA. A binary classification is performed (Human and non-human). Table 4.1 displays the accuracy rate of AR, pole, and MFCC features.

As it can be seen in this table, MFCC acquires the highest classification accuracy with 87.5% overall rate, followed by pole and AR modeling features with 85% and 72.5% rates, respectively.

4.6.2 Hardware Efficiency

Implementing an algorithm in hardware is a labor intensive and expensive process [63]. Because of the high development cost, it is very important to have a quick metric to compare the algorithm complexity vs. algorithm accuracy without the need to actually implement the algorithm in hardware. There are several metrics for this purpose, such as instruction count, execution time and

	Human (40)		Non-Human (40)		Overall Accuracy
AR - Human	40	%100	0		%72.5
AR- Non-Human	22		18	%45	
Pole - Human	38	%95	2		%85
Pole – Non - Human	10		30	%75	
MFCC - Human	40	%100	0		%87.5
MFCC–Non-Human	10		30	%75	

Table 4.1: Accuracy evaluation of audio classification methods based on AR, pole, and MFCC.

LUT count. The instruction count, which is the number of instruction executed during each run of the algorithm, is not a good metric since multiply instructions represent a much greater use of hardware resources than addition instructions [64]. Execution time is not a good metric either since the same algorithm might execute differently on different platforms. For example, a processor with multiplication unit and a processor without multiplication unit will have different execution time for the same algorithm. On the other hand, the total number of LUTs needed to implement the algorithm is a suitable metric since it quantifies the maximum amount of FPGA resources that will be needed to implement the algorithm if the algorithm can be fully parallelized in FPGA hardware.

In this thesis, we use the computational complexity to estimate the number of LUTs. First, the number of mathematical operations for an algorithm are calculated. Second, the number of LUTs required for implementing an operation in the Altera Cyclone FPGA using the Altera Qaurtus II is estimated. In the third step, the number of operations required for each algorithm is multiplied by the number of required LUTs for each operation, and the total number of LUTs required for each method is extracted.

The LUT count method used in this thesis is a very fast and low cost method which does not require an actual hardware implementation while represents an absolute mathematical upper bound on the real hardware cost. However there might be many different real hardware costs since there will be many different ways of implementing the same design on an FPGA depending on how someone might schedule the algorithm into hardware.

Operation	AR modeling	Pole Modeling	MFCC
Multipliers	$\sum_{r=1}^{m}(5L-4r-6)$	$I\left(2(m-1)(3m+1)+\frac{m^2(m+3)}{4}\right)$	$L^2+L+b\times L+m\times b$
Adders	$\sum_{r=1}^{m}(3L-2r-4)$	$I\left((m-1)(3m+1)+\frac{m(m-1)(m+2)}{4}\right)$	$L(L-1)+b(L-1)+m(b-1)$
Divisions	m	$I(m-1)$	
Square Roots		$I(m-1)$	L
Logarithms			b

Table 4.2: Total number of operations required in AR and pole modeling, and MFCC.

In Chapter 3, the total numbers of multiplications, additions, and divisions of AR, poles, and MFCC features are computed as shown in Tables 3.1, 3.2, and 3.3, respectively. Table 4.2 lists these calculations for each method. Assume m is the AR model order, which is also the number of poles and the desired order of MFCC. L is the maximum number of samples in each segment of the input signal. b is the number of mel windows in mel scale, which is assumed to be 20 in this application. In pole modeling, the number of iterations before the QR algorithm converges is denoted with I. The value of I depends on the nature of each signal, but the MATLAB analysis showed that $I = 10$ can be selected as the average number of iterations in different segments.

Including the appropriate values of parameters m, L, I, and b as utilized in Table 4.2, the number of the actual operations can be calculated for each method. The total number of operations is shown in Table 4.3.

Table 4.3: The number of operations2

Operations	AR	Poles	MFCC
Multiplications	13858	16360	53280
Additions	8346	10650	52807
Division	13	120	
Square roots		120	220
Logarithms			20

For each operation, the number of area usage is calculated in terms of LUT allocated to perform that operation. According to the Megawizard plug in for Altera Quartus II Web Edition FPGA design software, the number of LUTs allocated for each mathematical operation is shown in Table 4.4. Each operation is considered with Floating point double precision 64bits using Altera Cyclone FPGA.

Table 4.4: Number of LUTs - Altera Cyclone FPGA - Floating point double precision 64bits

Operations	Lut
Multiplications	3399
Additions	812
Division	6842
Square roots	4604
Logarithms	19700

Combining Tables 4.3 and 4.4, the total required LUTs for each audio feature extraction algorithm are calculated and displayed in Table 4.5.

Table 4.5: Number of LUTs for each method

Method	Lut
AR	53969240
Pole	65628960
MFCC	225384884

4.6.3 Accuracy/Hardware Comparison:

Fig. 4.6 shows the classification accuracy vs. hardware area usage for each method. Based on this figure, it can be concluded that pole modeling can be an appropriate tool for audio scene analysis application. Although AR modeling requires a low hardware area usage, it provides a classification performance of much lower than MFCC and pole modeling. However, pole modeling offers a comparable accuracy rate in respect to MFCC (87.5% and 85% respectively) while its hardware area usage is much fewer than the MFCC method. The hardware area perfromance of pole is 88.45% higher than MFCC as calculated by $100\frac{\mathrm{LUT}_{MFCC}-\mathrm{LUT}_{pole}}{\mathrm{LUT}_{pole}}$.

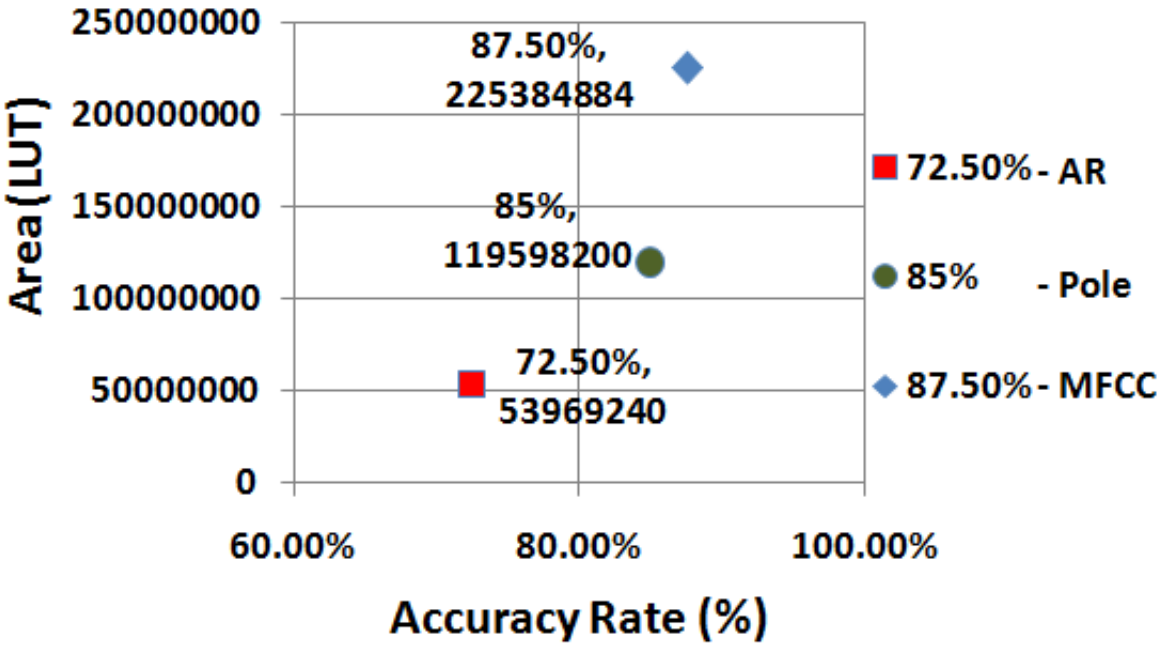

Figure 4.6: Audio environment classification accuracy vs. hardware area usage.

4.7 Summary

In this Chapter, software performance and hardware analysis for AR modeling, Pole modeling, and MFCC were performed and compared for environmental audio scene analysis application. Based on the results obtained, the pole modeling feature analysis method is proposed for NIOS II implementation as will be explained in the next Chapter.

Chapter 5

Pole Modeling FPGA Embedded Implementation

5.1 Introduction

Pole modeling is selected in Chapter 4 as an appropriate feature extraction method for audio scene analysis. As shown in Fig. 5.1, the present Chapter explains the implementation of the pole modeling feature extraction + LDA classifier using ALTERA DE2 development board and Niose II embedded system design. The pole modeling + LDA implementation is the first known work performed to the best of the author's knowledge.

5.2 Pole Modeling Feature Analysis Embedded System Implementation

The main focus of this section is to design the pole modeling feature analysis using pole features and LDA classifier into an embedded system. As it is explained in section 2.2.1, pole modeling includes a roots finding algorithm which calculates the roots of the P polynomial in the dominator of AR model. Roots finding is an iterative and algorithmic intensive method. Because embedded software implementation is more suitable for algorithmic intensive algorithms, in this thesis, pole modeling is implemented in the Altera Nios II processor, which is a softcore processor. The details of the main steps of design with required tool for each step are summarized in Table 5.1, and the system architecture of the design is shown in Fig. 5.2. The first stage is developing the C++

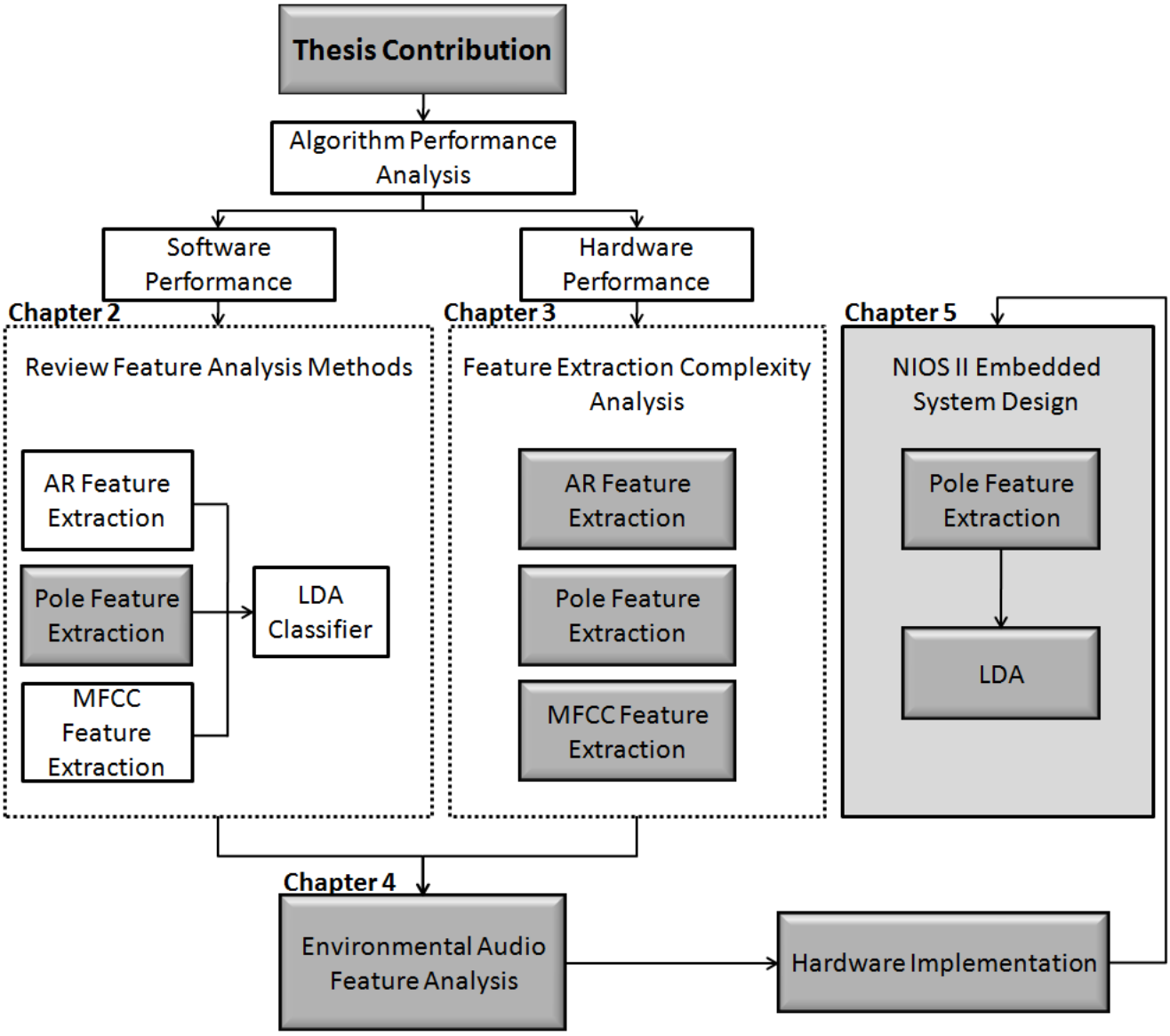

Figure 5.1: Chapter 5 - Hardware Implementation.

Design Steps	Tools
1 - Implement Pole Modeling algorithm in C++ Language	Visual C++ Software
2 - Create the Embedded System into Altera Nios II system programmable-on-chip (SoPC)	Altera Nios II system programmable-on-chip (SoPC) Builder
3 - Upload the system in FPGA (DE2 board)	Quartus II (programmer)
4 - Simulate and upload the Pole modeling codes in FPGA (DE2 board)	Nios II IDE
5 - Create the serial connection between the Hardware and the Software.	Nios II IDE

Table 5.1: Main design steps in the project.

codes for pole modeling and LDA classifier. Stage 2 creates the embedded system into Altera Nios II system using programmable-on-chip (SoPC) builder. Stages 3 to 5 performs the uploading and simulation of the codes in the DE2 development board, and the C++ code is executed on the Nios II platform. The pole modeling SoC hardware is connected to host computer via a JTAG UART serial communication protocol. Finally, the classification accuracy and execution time of pole modeling hardware implementation are compared with the MATLAB implementation.

5.2.1 ALTERA DE2 Development Kit

The target device for hardware implementation on this thesis is a development circuit board available from ALTERA called ALTERA DE2 board. ALTERA'S DE2 Board shown on the Figs. 5.3 and 5.4[1].

The DE2 board features a Cyclone II 2C35 FPGA in a 672-pin package. All important components on the board are connected to pins of this chip, allowing the user to control all aspects of the

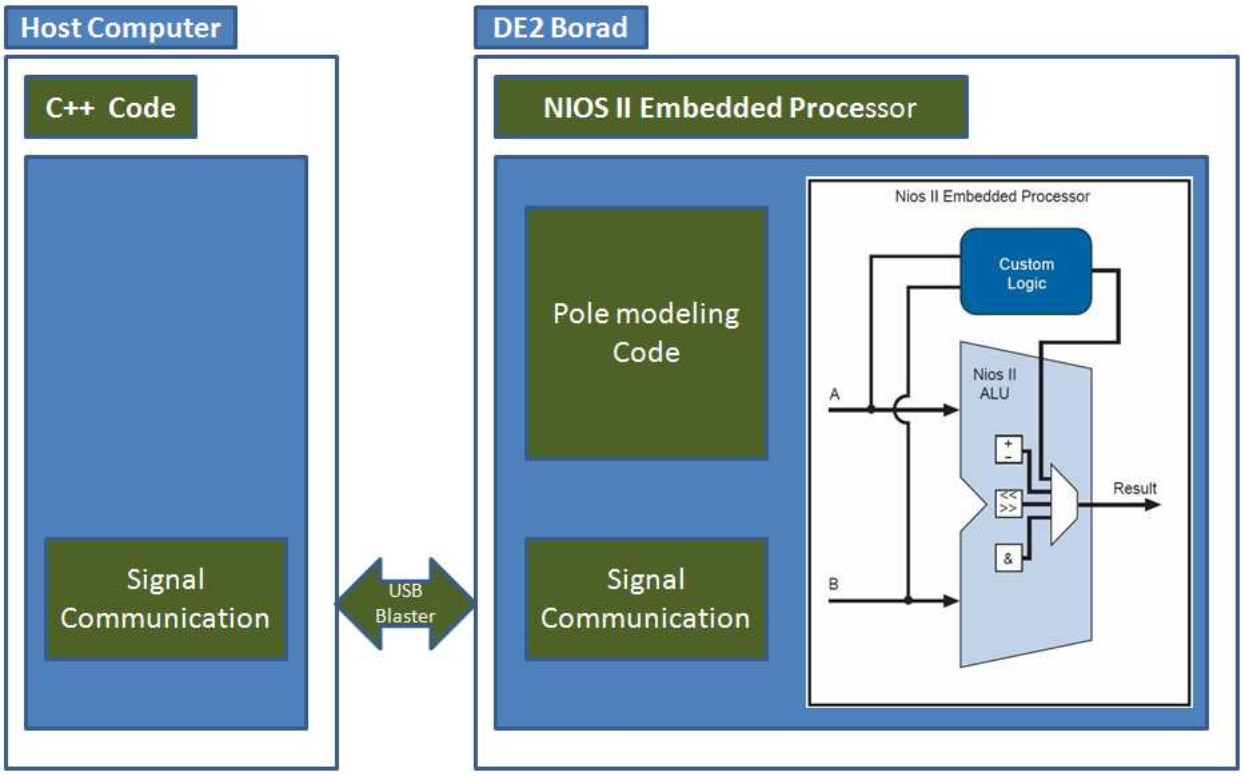

Figure 5.2: System architecture of the design.

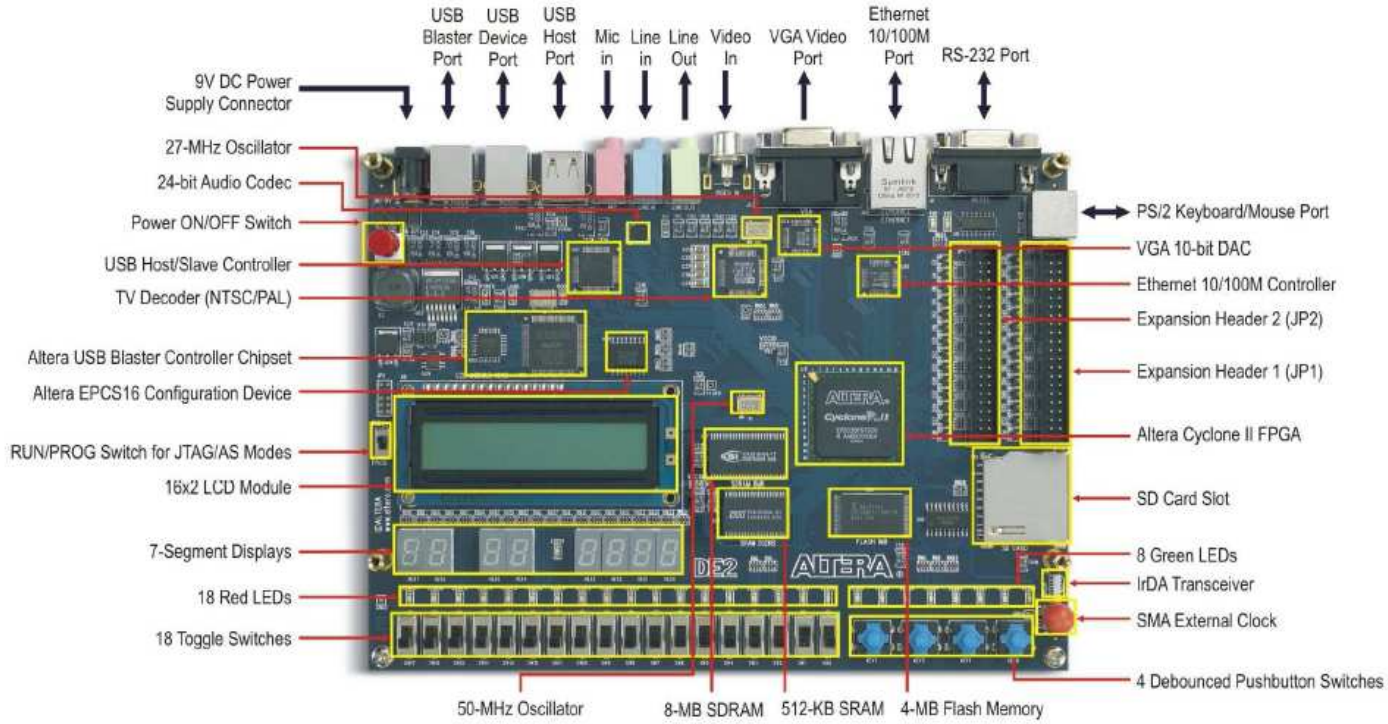

Figure 5.3: ALTERA DE2 board [1].

boards operation. The DE2 board includes a sufficient number of switches, LEDs, and 7-segment displays for simple experiments. Also, there are SRAM, SDRAM, and Flash memory chips, as well as a 16 x 2 character display.

With its advanced Cyclone II FPGA, flexible memory options, and advanced I/O devices, the DE2 board is a very good platform for the implementation and development of digital systems. Furthermore, the DE2 board is a suitable choice for implementing and developing embedded applications such as the ones that feature the Altera Nios II embedded processor.

Altera Nios II is designed as a rapid prototyping resource, and therefore it is used in this thesis. The Nios II processor can be used with a variety of other components to form a complete system. Alteras DE2 Development board contains several components that can be integrated into a Nios II system. An example of such a system is shown in Fig. 5.5 which is used in this work.

The Nios II processor and the interfaces needed to connect to other chips on the DE2 board are implemented in the Cyclone II FPGA chip. These components are interconnected by means of the

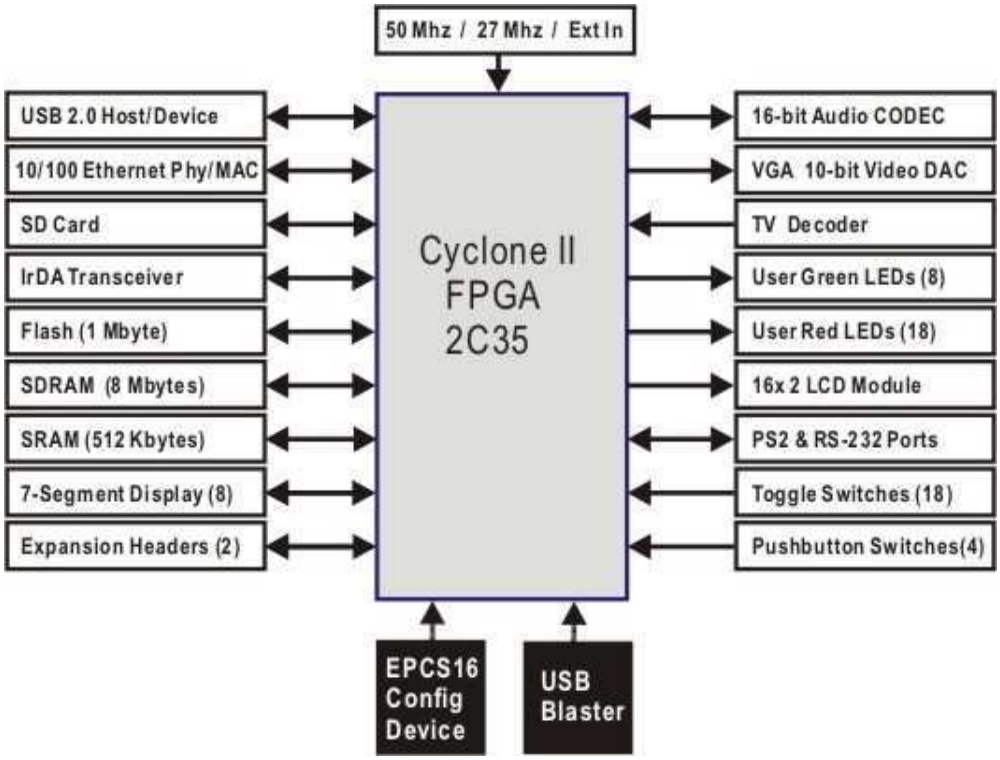

Figure 5.4: Block diagram of the DE2 board [1].

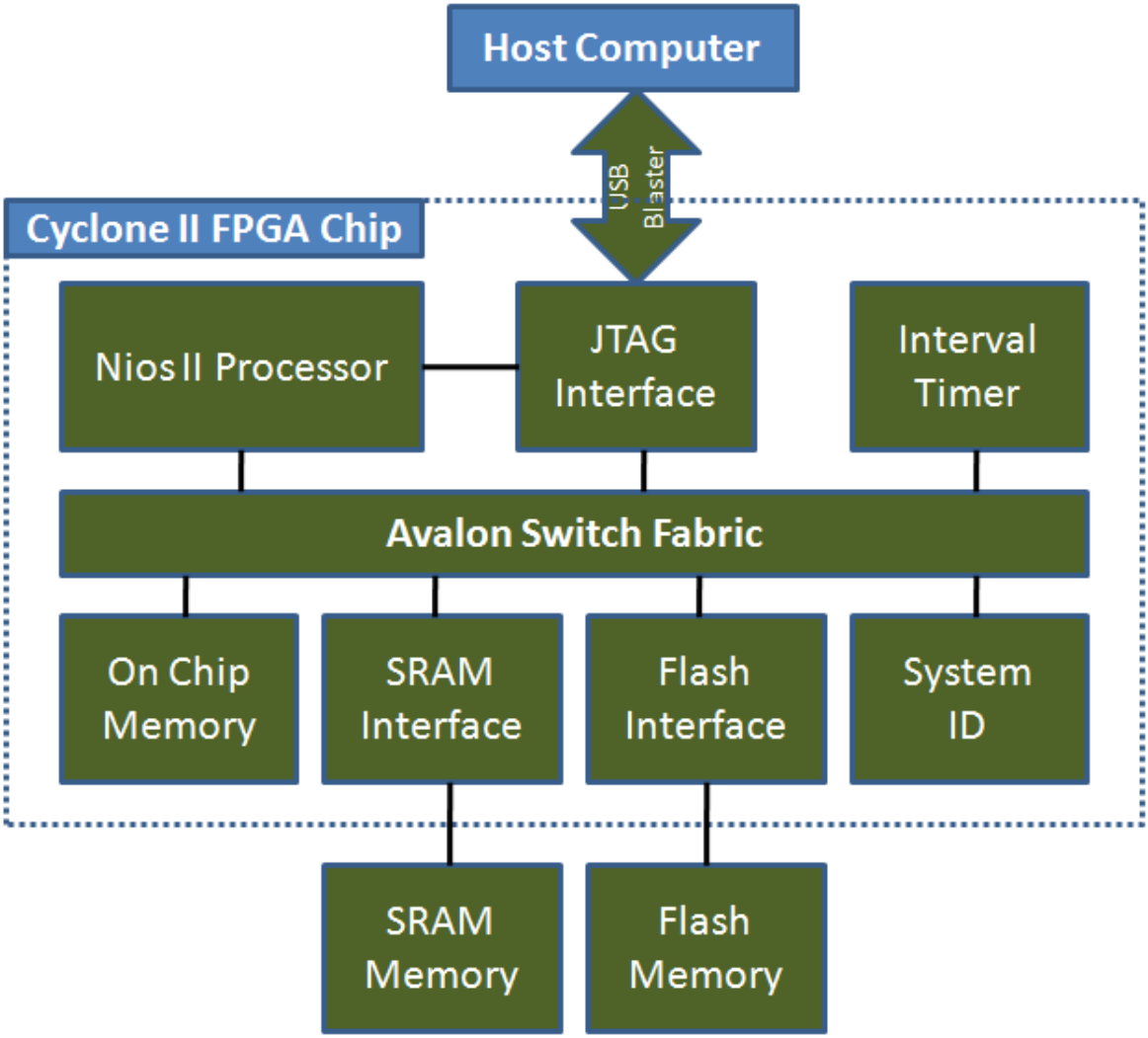

Figure 5.5: Block diagram of Nios II system implemented on the DE2.

interconnection network called the Avalon Switch Fabric. Memory blocks in the Cyclone II device can be used to provide an on-chip memory for the Nios II processor. They can be connected to the processor either directly or through the Avalon network. The SRAM and SDRAM and Flash memory chips on the DE2 board are accessed through the appropriate interfaces. Input/output interfaces are instantiated to provide connection to the I/O devices used in the system.

A special JTAG UART interface is used to connect to the circuitry that provides a Universal Serial Bus (USB) link to the host computer to which the DE2 board is connected. This circuitry and the associated software is called the USB-Blaster. Another module, called the JTAG Debug module, is provided to allow the host computer to control the Nios II processor. It makes it possible to perform operations such as downloading programs into memory, starting and stopping execution, setting program breakpoints, and collecting real-time execution trace data. The SOPC Builder tool in the Quartus II software used to implement a desired system by choosing the required components and specifying the parameters needed to make each component fit the overall requirements of the system. Next subsection is a detail explanation of an NIOS II system which we design using SOPC Builder tool. This embedded system used for implementing the pole modeling feature analysis on Altera DE2 board.

5.2.2 Nios II Embedded System Design

As shown in Table 5.1, once the C code of the pole modeling is developed, the first stage in the hardware design, the second row in the Table, is creating the Nios II Embedded System. Next stage, as shown in the third row of Table 5.1, is to upload the designed Nios II Embedded System program to the FPGA. These stages are explained in this section as follows.

Nios II SoPC development environment, Nios II embedded processor serves as the general-purpose processor, and other peripherals such as UART, various IO controllers, memory, timer and custom instruction are connected to Nios II via Avalon System Bus as shown in Fig. 5.5. For the purpose of this thesis, an Nios II Embedded System is designed using Altera SOPC Builder tool.

As it is shown in Figs. 5.6 and 5.5, the following components are included in the Nios II Embedded System designed in this work:

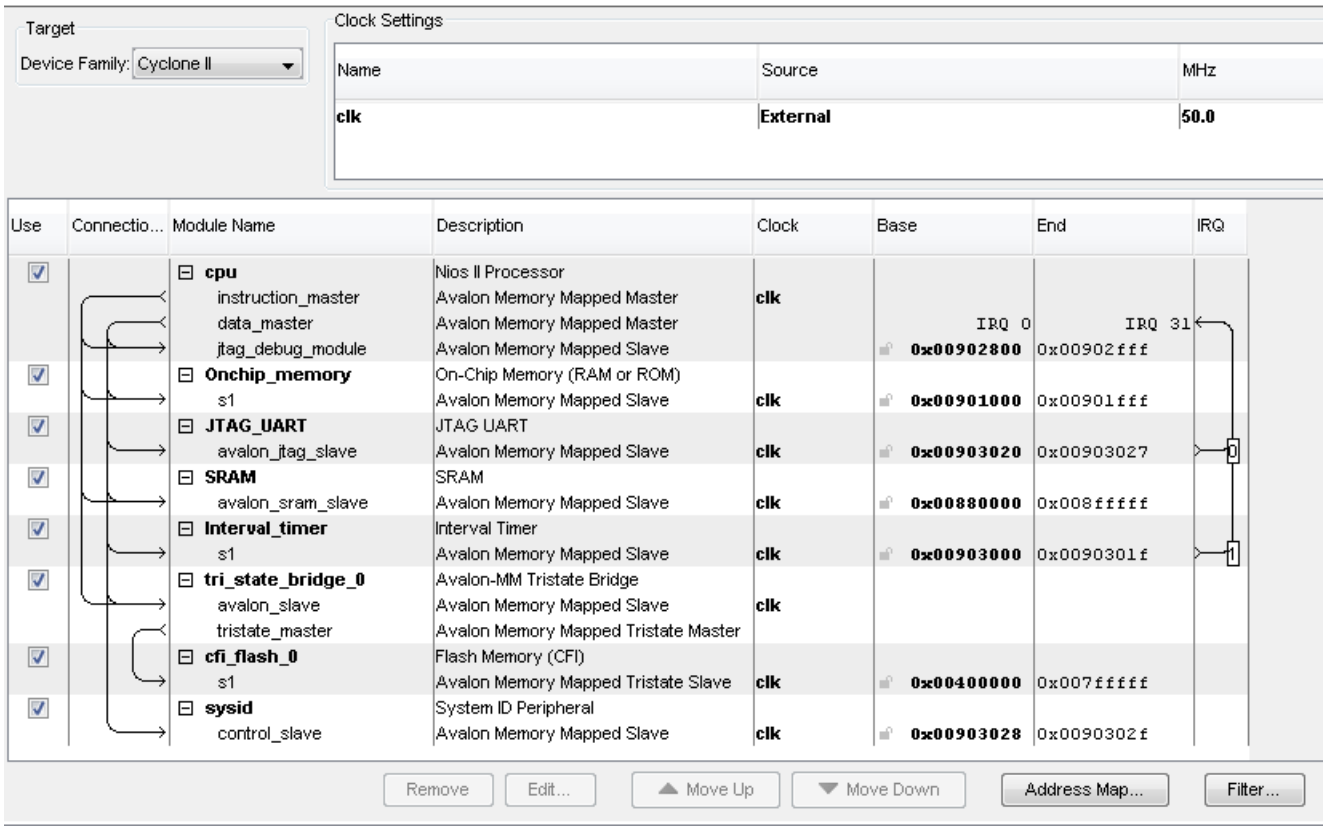

Figure 5.6: Embedded computer system in SOPC builder.

- **Altera Nios II processor (CPU)**: The system includes Altera Nios II/s processor which is a 32-bit CPU that can be instantiated in an Altera FPGA chip.

- **On-chip Memory**: The designed embedded system includes a 32-Kbyte memory that is implemented in the Cyclone II FPGA chip (On-chip Memory). This memory is organized as 8K x 32 bits, and spans addresses in the range 0x00901000 to 0x00901FFF.

- **SRAM**: The SRAM Controller provides a 32-bit interface to the static RAM (SRAM) chip on the DE2 Board. This SRAM chip is organized as 256K x 16 bits, and is mapped to the address space 0x00880000 to 0x0088FFFF.

- **Avalon-MM Tristate Bridge**: The Avalon-MM Tristate Bridge is added , which allows the Nios II processor to interface with the flash memory.

- **Flash Memory Interface (CFI)**: The system includes an interface to the 4MB flash memory on board. To add flash memory chip to the system, first the Avalon-MM Tristate Bridge is added. Nios II Embedded System reads the input signal information from the host computer and stores on flash memory to read during the operation.

- **Interval timer**: The designed Nios II Embedded System contains a timer that can be used to measure various time intervals. The interval timer is loaded with a preset value, and then counts down to zero using the 50-MHz clock signal provided on the DE2 board. The programming interface for the timer includes six 16-bit registers. The 16-bit registers are employed to provides status information about the timer, control the settings such as start or stop, change the period of the timer, capture a snapshot of the counter value, and read to obtain the count value. This timers are used to measure the execution time of the program.

- **System ID Peripheral**: The system ID module provides a unique value that identifies the Nios II Embedded System. The host computer connected to the DE2 board can query the system ID module by performing a read operation through the JTAG port. The host computer can then check the value of the returned identifier to confirm that the DE2 Basic Computer has been properly downloaded into the DE2 board. This process allows debugging tools on

the host computer to verify that the DE2 board contains the required computer system before attempting to execute code that has been compiled for this system.

- **JTAG UART**: The JTAG port implements a communication link between the DE2 board and its host computer. This link is automatically used by the Quartus II software to transfer FPGA programming files into the DE2 board.

After all the required elements are added to the design in the SOPC builder, the base addresses are needed to be set to avoid conflicts between the system components. Final step in Altera SOPC builder tool is generating the system. Now that the SOPC system is built as explained in this section, it is required to be integrated in a block diagram file in the QuartusII project. To complete the hardware design in the QuartusII project, the following stages have to be completed:

- Instantiate the module generated by the SOPC Builder into the QuartusII project: All we need to do is instantiate the Nios II system in our top-level design file, and connect inputs and outputs of the parallel I/O ports, as well as the clock and reset inputs, to the appropriate pins on the CycloneII device. The instantiation of the generated module depends on the design entry method chosen for the overall QuartusII project. We have chosen to use schematic entry, but the approach is similar for both Verilog and VHDL methods.

- Assign the FPGA pins in QuartusII.

- Compile the designed circuit in QuartusII.

- Program and configure the Cyclone II FPGA in the JTAG programming mode on the DE2 board using system programmer tool.

Fig. 5.5 shows the completed Nios II Embedded system symbol in QuartusII. Up to this step a Nios II Embedded System designed and loaded to the FPGA on DE2 board. Next subsection explains the implementation of the software application for the Nios II system using Nios II IDE.

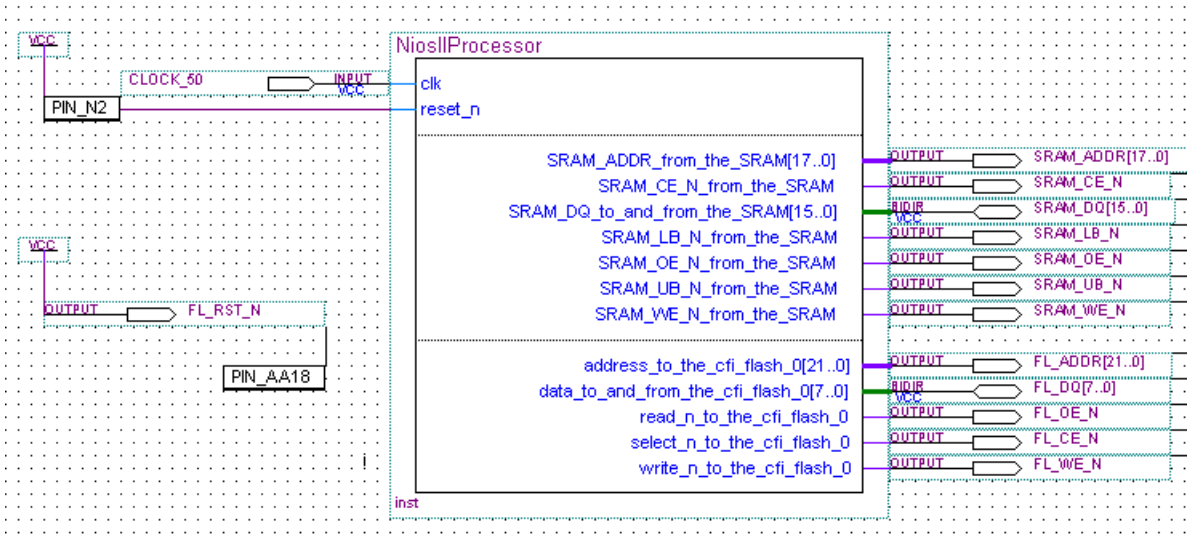

Figure 5.7: Complete system design in Altera Quartus II

5.2.3 Application Simulation

A Nios II Embedded system which is an embedded computer on DE2's FPGA, is designed and implemented as explained in the previous subsection. The final stage in the hardware design is uploading the application C++ codes in the FPGA as shown in Table 5.1. This has been done using Nios II integrated development environment(IDE).

The Nios II integrated development environment (IDE) is the software development graphical user interface (GUI) for the Nios II processor. All software development tasks can be accomplished within the Nios II IDE, including editing, building, and debugging programs. The Nios II IDE is the window through which all other tools can be launched. Information on Nios II IDE can be found on [65]. The Nios II IDE is a thin user interface that manipulates other tools behind the scenes, shields a designer from the details of command line tools, and presents a unified development environment.

The Nios II IDE program compiles C++ language programs and downloads them into the Nios II Embedded system. Based on the industry-standard GNU tool chain, the Nios II IDE provides a graphical user interface (GUI) to the GCC compiler. The Nios II IDE build environment is designed to facilitate software development for Altera's Nios II processors, providing an easy-to-

use push-button flow, while also allowing designers to manipulate advanced build settings. The Nios II IDE build environment automatically produces a makefile based on the user's specific system configuration (the SOPC Builder-generated PTF file). Changes made in the Nios II IDE compiler/linker settings are automatically reflected in this auto-generated makefile. These settings can include options for the generation of memory initialization files (MIF), flash content, simulator initialization files (DAT/HEX), and profile summary files.

The Nios II IDE contains a robust software debugger based on the GNU debugger, GDB. The debugger provides many basic debug features, as well as several advanced debug features not usually available with low-cost processor development kits. Run and debug operations are available by right-clicking the project. The Nios II IDE allows a designer to run or debug the project either on a target board or the Nios II instruction set simulator (ISS).

The Nios II IDE presents a new project wizard used to automate the set-up of the C/C++ application project and system library projects. In addition to the new project wizard, the Nios II IDE provides software code examples (in the form of project templates) to help software engineers bring up working systems as quickly as possible. Each template is a collection of software files and project settings. Designers can add their own source code to the project by placing the code in the project directory or importing the files into the project.

In this thesis, the C++ code of the pole modeling is developed in Microsoft Visual C++. The Nios II IDE tool is next used to build the C++ files and download them into the designed Nios II Embedded system. The properties of the system library have to be changed according to Fig. 5.8.

Nios II Flash Programing In the standard C++ programming, a text file (say, ”arcof.txt”) can be uploaded as system's input using the following command:

pFile = fopen (”arcof.txt” , ”r”);

However, in a hardware design, the input programming of the FPGA cannot be performed using the above command. It is required to upload the text file into the flash memory embedded in the DE2 Board, and then the data can be read by the FPGA from the flash memory. The Nios II IDE provides the Read-Only Zip File System software component, which is easy-to-use tool for storing

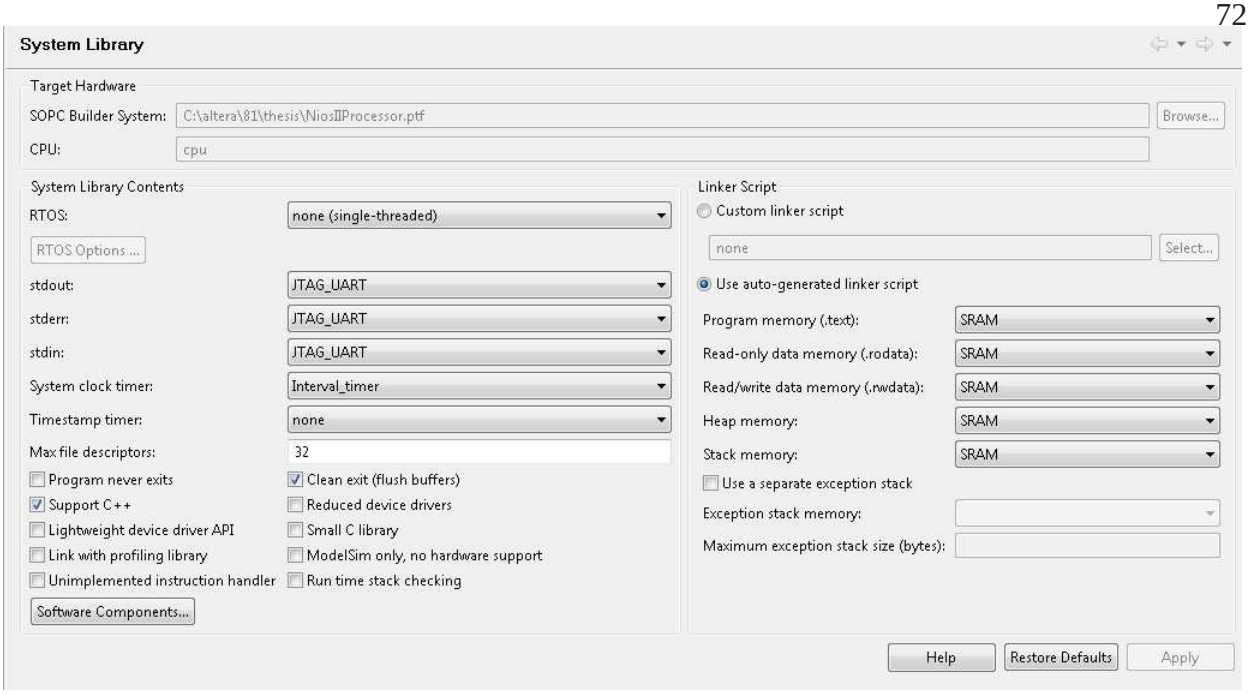

Figure 5.8: Setting of the System Library.

data to flash memory. As shown in Fig. 5.9, the following four parameters have to be specified to configure the file system:

1. The name of the flash device where it is needed to program the file system. In the present design, this name is cfi-flash-0.

2. The offset in the address space of this flash device. It is $0X000000$ in the present work.

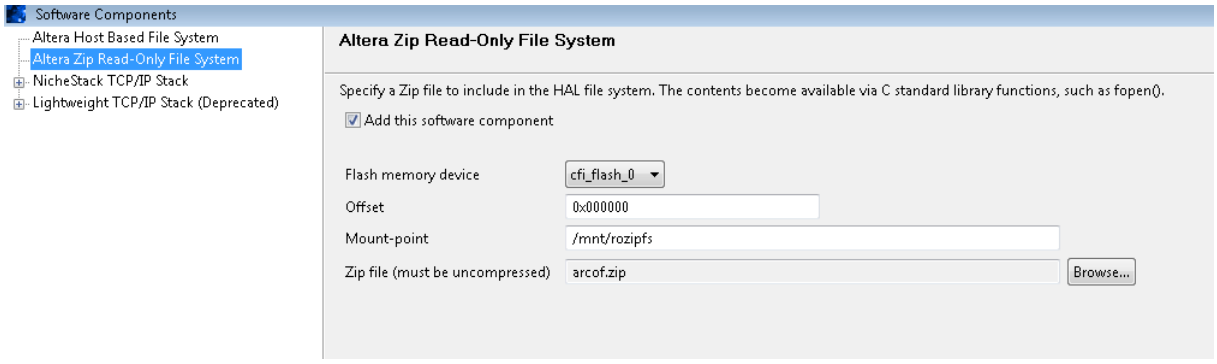

Figure 5.9: Setting of the ALTERA zip read-only file system.

3. The name of the mount point for this file subsystem in the HAL file system. For example, if the mount point is named /mnt/zipfs, the following code opens the text file (”arcof.txt”) in the zip file:

   pFile = fopen (”/mnt/rozipfs/arcof.txt” , ”r”);

   The above code opens the zip file ”arcof.txt” for reading.

4. The name of the zip file to be used is ”arcof.zip”. This file is stored in the system library folder of the C++ project.

5.2.4 Results

The C++ codes for pole modeling and LDA classifier are developed in Nios II IDE, and uploaded and executed in the designed Nios II Embedded System via a JTAG UART serial communication protocol. The feature vetors are listed in a text file and converted to a zip file so that they can be uploaded to the flash memory using Read-Only Zip File System software component in the FPGA. Additionally, the trained LDA classifier parameters are also uploaded in the FPGA device. Once the pole modeling feature analysis program is executed in the Altera DE2 board, the classification results are displayed in the Nios II IDE console. The classification accuracy results obtained using this implementation is equal to the MATLAB implementation shown in Table 4.1.

The execution time of the implementation is recorded using Interval Timer in NIOS II Embedded System. To achieve this period, the difference between the starting value of the Interval Timer and its ending number is measured. Once this difference is divided by the interal clock frequency of the hardware, it represents the execution time. In the present implemenation, the starting value of the timer is 65462 and after the cound down, this value decreases to 41582. Since the interal clock frequency of the hardware is set to be $50MHz$, the execution time is found to be about 0.5 ms. This time represents the average time that each implementation technique spends for classification of one feature vector. The average total execution time for one audio signal is measured to be 0.1s.

As measured above, the total classification time in the developed embedded system is 0.1 s

for one audio sample. Depending on the application on hand, we can decide whether this system can be considered real-time or not. For example, in the audio scene application which is considered in the present thesis, the developed system can be considered for real-time analysis. This can be explained in the following scenario: When a person with hearing disability enters into a new environment, or when the environment condition changes, the adaptive hearing system has to determine whether the audio belongs to human or non-human class so that the hearing aid instrument can accordingly switch its mode in order to provide a better quality hearing for the person. Considering the human perception which requires at least 0.5 s of an audio to understand the content [66], the system with only 0.1 s execution time can be considered a real-time system.

5.3 Summary

The present Chapter explained the NIOS II implementation of the pole modeling feature extraction as an appropriate feature extraction method for audio scene analysis. LDA was used as the classifier in the implementation. The classification accuracy results obtained using this implementation was equal to the results from MATLAB implementation. The classification time for one audio sample is determined to be 0.1s, which is fast enough to be considered as a real-time system for audio scene analysis application.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The objective of this thesis was to investigate and compare both the hardware and software implementation aspects of three commonly used signal feature extraction techniques: AR modeling and pole modeling, and MFCC. Chapter 1 provided the motivation behind the present work as well as the significance of signal feature analysis and the challenges in real-world signal analysis applications. A comprehensive signal analysis is explained. This Chapter also described the importance of investigating both the hardware and software implementation aspects of the existing feature analysis methods. It also reviewed some of the commonly employed feature analysis tools as related to non stationary and complex signals.

Chapter 2 provided the diagram and theroritical background of each method used in this thesis. It explained the detailed analytical procedures of AR, Pole, and MFCC feature extraction methods, as well as LDA classifier. The computational complexity of AR, pole, and MFCC is investigated in Chapter 3. This Chapter used the theoretical explanation in Chapter 2 in order to compute the number of arithmetical operations each method requires through detailed analytical calculations. Chapter 4 employed AR, pole, and MFCC feature analysis methods for audio scene analysis application. Furthermore, this Chapter compared these three methods with respect to two domains: hardware analysis and algorithmic accuracy. The pole modeling feature analysis was selected as the suitable tool among the three approaches and was used in the following Chapter. Chapter 5 proposed the implementations of pole modeling feature extraction + LDA feature classifier using

the Altera NIOS II on DE2 board . The summary of the research outcome and the discussion of the future work are presented in Chapter 6.

6.1.1 Analytical Contributions

- The computational complexity of AR modeling, pole modeling, and MFCC are calculated as shown in Tables 3.1, 3.2, and 3.3. This contribution is essential in computation of the hardware analysis performance. This analysis required us to take apart each algorithmic stage into its arithmetic equations, and calculate the number of operations in each step, ie. number of multipliers, adders, or dividers.

- The pole modeling computational complexity is the first known work performed to the best of author's knowledge. Furthermore, one of the most important stages in the pole modeling method is the root finding algorithm. In this thesis we selected '*eigenvalues of companion matrix and QR algorithm*' as the root finding method which is suitable for the pole calculation of an AR system.

- FPGA area for AR modeling, pole modeling, and MFCC are estimated. Using the Megawizard plugin for Altera Quartus II Web Edition FPGA design software, the number of LUTs allocated for each mathematical operation is extracted. Each operation is considered with Floating point double precision 64bits using Altera Cyclone FPGA. Having the allocated LUT for each operation and the number of operations in each method, the total required LUTs for each audio feature extraction algorithm are calculated as shown in Table 4.5. The number of required LUTs is used as an estimate of the FPGA area usage of each method.

- AR and pole modeling, and MFCC features are implemented and evaluated in MATLAB software for audio scene analysis as displayed in Table 4.1. Our pole modeling feature extraction is the first known work proposed for audio scene analysis.

- The classification accuracy vs. hardware area usage for AR modeling, pole modeling, and MFCC is compared as displayed in Fig. 4.6. Based on this comparison, the pole modeling

feature analysis method is proposed as the optimum solution for hardware implementation of audio scene analysis application.

6.1.2 FPGA Embedded Implementation

- The pole modeling feature extraction + LDA classifier is implemented in Altera DE2 Board using Altera Nios II soft-core processor. The classification accuracy results obtained using this hardware implementation is achieved to be equal to the MATLAB implementation. The classification time for one audio sample is determined to be 0.1s, which is fast enough to be considered as a real-time system for audio scene analysis application.

6.2 Future Work

The future work of the research could be:

- Although pole modeling was proposed as a suitable algorithm for audio scene analysis application, for other applications any of AR, pole modeling and MFCC algorithms may perform better depending on the available hardware resources and real-time requirements. An study performed for each application can indicate which method is more suitable in terms of classification accuracy vs. hardware area usage.

- In this work, the FPGA usage is estimated and quantified based on the total number of LUT needed to implement each algorithm. This calculation is based on computational complexity of each algorithm. A hardware architecture design can be done for each method to estimate the hardware performance in terms of area and execution time.

Bibliography

[1] Altera Corporation, "DE2 development and education board user manual."

[2] R. M. Rangayyan, "Biomedical signal analysis: a case-study approach," *New York, N.Y.: Wiley-Interscience*, 2002.

[3] R. M. Rangayyan, S. Krishnan, G. D. Bell, C. B. Frank, and K. O. Ladly, "Parametric representation and screening of knee joint vibroarthrographic signals," *IEEE Transactions on biomedical engineering*, vol. 4, pp. 1068 – 1074, November 1997.

[4] S. Tavathia, R. Rangayyan, C. Frank, G. Bell, K. Ladly, and Y. Zhang, "Analysis of knee vibration signals using linear prediction," *IEEE Transactions on biomedical engineering*, vol. 39, no. 9, pp. 959 – 970, 1992.

[5] M. Akay, J. L. Semmlow, W. Welkowitz, M. D. Bauer, and J. B. Kostis, "Detection of coronary occlusions using autoregressive modeling of diastolic heart sounds," *IEEE Transactions on biomedical engineering*, vol. 37, pp. 366 – 373, April 1990.

[6] B. Ahmadi, R. Aimrfattahi, E. Negahbani, M. Mansouri, and M. Taheri, "Comparison of adaptive and fixed segmentation in different calculation methods of electroencephalogram time-series entropy of estimating depth of anesthesia," *6th International special topic conference on ITAB*, pp. 265 – 268, 2008.

[7] Z. M. K. Moussavi, R. M. Rangayyan, G. D. Bell, C. B. Frank, K. O. Ladly, and Y.-T. Zhang, "Screening of vibroathrographic signals via adaptive segmentation and linear prediction modeling," *IEEE transactions on biomedical engineering*, vol. 43, pp. 15 – 23, January 1996.

[8] M. Akay, M. Bauer, J. L. Semmlow, W. Welkowitz, and J. Kostis, "Autoregressive modeling of diastolic heart sounds," *IEEE engineering in medicine and biology society 10th annual international conference*, 1988.

[9] S. H. Kim, H. B. Han, K. R. Hong, M. H. Lee, and S. H. Park, "Pattern classification of AR model parameters of EMG signal for the diagnosis of TMJ dysfunction syndrome," *IEEE Proceedings, Computers and Communications Technology Toward 2000*, vol. 43, pp. 1317 – 1321, January 1987.

[10] Z. Luo, F. Wang, and W. Ma, "Pattern classification of surface electromyography based on AR model and high-order neural network," *IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications*, pp. 1 – 6, 2006.

[11] A. Angelidou, M. Strintzis, S. Panas, and G. Anogianakis, "On AR modelling for MEG spectral estimation, data compression and classification," *IEEE Transactions on Computers in Biology and Medicine*, pp. 379 – 87, 1992.

[12] A. Iwata, N. Suzumura, and K. Ikegaya., "Pattern classification of the phono-cardiogram using linear prediction analysis," *Medical and Biological Engineering and Computing*, vol. 15, pp. 407 – 412, July 1977.

[13] I. S. N. Murthy and G. S. S. D. Prasad, "Analysis of ECG from pole-zero models," *IEEE Transactions on Biomedical Engineering*, vol. 39, pp. 741 – 751, July 1992.

[14] M. B. Kuzmicki, S. L. Weinstein, and G. B. Schiff, "Development of an AR-pole statistic for ECoG seizure detection," *In CDROM proceedings, 17th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Montreal, PQ, Canada*, 1995.

[15] S. B. Ghazale and J. Hansen, "A comparative study of traditional and newly proposed features for recognition of speech under stress," *IEEE Transactions on Speech and Audio Processing*, vol. 8, pp. 429 – 442, 2000.

[16] M. Carey, E. Parris, and H. Lloyd-Thomas, “A comparison of features for speech, music discrimination,” *International Conference Acoustics, Speech, and Signal Processing, Phoenix, AZ, USA*, vol. 1, pp. 149 – 152, 1999.

[17] J. Saunders, “Real-time discrimination of broadcast speech/music,” *In Proc. of International Conference Acoustics, Speech, and Signal Processing (ICASSP), Atlanta, GA, USA*, vol. 2, pp. 993 – 996, May 1996.

[18] J. Pinquier, J.-L. Rouas, and R. A. Obrecht, “Robust speech / music classification in audio documents,” *International Conference on Spoken Language Processing, Denver, USA*, vol. 3, pp. 2005 – 2008, September 2002.

[19] E. Scheirer and M. Slaney, “Construction and evaluation of a robust multifeature speech and music discriminator,” *In Proc. of International Conference Acoustics, Speech, and Signal Processing (ICASSP), Munich, Germany*, vol. 2, pp. 1331 – 1334, April 1997.

[20] N. Mesgarani, M. Slaney, and S. Shamma, “Discrimination of speech from nonspeech based on multiscale spectro-temporal modulations,” *International Conference Acoustics, Speech, and Signal Processing, Phoenix, AZ, USA*, vol. 14, pp. 920 – 930, May 2006.

[21] J. M. Kates, “Classification of background noises for hearing-aid applications,” *The Journal of the Acoustical Society of America*, vol. 97, pp. 461 – 470, January 1995.

[22] O. Tuzun, M. Demirekler, and K. Nakiboglu, “Comparison of parametric and non-parametric representations of speech for recognition,” *Electrotechnical Conference, Proceedings., 7th Mediterranean*, vol. 1, pp. 65 – 68, 1994.

[23] D. OShaughnessy, “Speaker recognition,” *IEEE ASSP Magazine*, vol. 3, pp. 4 – 17, October 1986.

[24] D. Ohaughnessy, “Cepstral analysis technique for automatic speaker verification,” *IEEE Transactions on Acoust., Speech, Signal Process.*, vol. ASSP-29, pp. 254 – 272, April 1981.

[25] K. Murty and B. Yegnanarayana, "Combining evidence from residual phase and MFCC features for speaker recognition," *IEEE Signal Processing Letters*, vol. 13, pp. 52 – 55, 2006.

[26] B. Milner and X. Shao, "Prediction of fundamental frequency and voicing from mel-frequency cepstral coefficients for unconstrained speech reconstruction," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, pp. 24 – 33, 2007.

[27] S. Chu, S. Narayanan, and C.-C. J. Kuo, "Environmental sound recognition using MP-based features," *In the Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2008*, pp. 1 – 4, March 31 - April 4 2008.

[28] K. Umapathy, S. Krishnan, and S. Jimaa, "Multigroup classification of audio signals using time-frequency parameters," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 2, pp. 308 – 315, April 2005.

[29] A. Abu-El-Quran, R. Goubran, and A. Chan, "Adaptive feature selection for speech / music classification," *In the Proceedings of the IEEE 8th Workshop on Multimedia Signal Processing*, pp. 212 – 216, October 2006.

[30] L. Lu, H.-J. Zhang, and H. Jiang, "Content analysis for audio classification and segmentation," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 7, pp. 504 – 516, October 2002.

[31] G. Guodong and S. Li, "Content-based audio classification and retrieval by support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, pp. 209 – 215, January 2003.

[32] K. Umapathy, S. Krishnan, and R. Rao, "Audio signal feature extraction and classification using local discriminant bases," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, pp. 1236 – 1246, 2007.

[33] S. Xi, X. Changsheng, and M. Kankanhalli, "Unsupervised classification of music genre

using hidden Markov model," *In the proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, vol. 3, pp. 2023 – 2026, June 2004.

[34] The MathWorks Inc. MathWorks MATLAB - The Language Of Technical Computing. [Online]. Available: http://www.mathworks.com/products/matlab/

[35] Wolfram Research Inc. Wolfram Mathematica: Technical Computing Software. [Online]. Available: http://www.wolfram.com/mathematica/

[36] U. Varshney and R. Vetter, "Emerging mobile and wireless networks," *Communications of the ACM*, vol. 43, pp. 73 – 81, June 2000.

[37] S. Ravindran and D. Anderson, "Audio classification and scene recognition and for hearing aids," *In IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 860 – 863, May 2005.

[38] N. Vu, J. Whittington, H. Ye, and J. Devlin, "Implementation of the MFCC front-end for low-cost speech recognition systems," *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2334 – 2337, June 2010.

[39] R. Ramos-Lara, M. Lopez-Garcia, E. Canto-Navarro, and L. Puente-Rodriguez, "Svm speaker verification system based on a low-cost FPGA," *Field Programmable Logic and Applications*, pp. 582 – 586, September 2009.

[40] M. Staworko and M. Rawski, "Fpga implementation of feature extraction algorithm for speaker verification," *Proceedings of the 17th International Conference, Mixed Design of Integrated Circuits and Systems (MIXDES)*, pp. 557 – 561, June 2010.

[41] M. R. Smith, T. J. Smith, S. W. Nichols, S. T. Nichols, H. Orbay, and K. Campbell, "A hardware implementation of an autoregressive algorithm," *Meas. Sci Technol, IOP Publishing Ltd*, vol. 1, pp. 1000 – 1006, 1990.

[42] B. Jiao, "High level FPGA implementation of adaptive signal segmentation and autoresgressive modeling techniques," *Master of Applied Science thesis, Ryerson University*, 2009.

[43] M. Aboy, O. W. Marques, J. McNames, R. Hornero, T. Trong, and B. Goldstein, "Adaptive modeling and spectral estimation of nonstationary biomedical signals based on kalman filtering," *IEEE Transactions on Biomedical Engineering*, vol. 52, pp. 1485 – 1489, 2005.

[44] J. Pardey, S. Roberts, and L. Tarassenko, "A review of parametric modelling techniques for EEG analysis," *Med. Eng. Phys.*, vol. 18, pp. 2 – 11, 1996.

[45] J. Makhoul, "Linear prediction: a tutorial review," *Proceedings of the IEEE,* vol. 63, pp. 561 – 580, April 1975.

[46] J. P. Burg, "Maximum entropy spectral analysis," *Ph.D. dissertation, Stanford University*, May 1975.

[47] W. Mekwi, "Iterative methods for roots of polynomials," *Thesis of MSc in Mathematical Modelling and Scientific Computing, University of Oxford*, 2001.

[48] M. JENKINS and J. TRAUB, "A three-stage algorithm for real polynomials using quadratic iteration," *SIAM J. Numer. Anal.*, vol. 7, no. 4, 1970.

[49] V. Pan, B. Murphy, R. Rosholt, D. Ivolgin, Y. Tang, X. Yan, and X. Wang, "Root finding with Eigen solving," *SYMBOLIC NUMERIC COMPUTATION, Trends in Mathematics*, pp. 185 – 210, 2007.

[50] S. Krishnan, "Adaptive signal processing techniques for analysis of knee joint vibroarthrographic signals," *Ph.D. dissertation, University of Calgary*, 1999.

[51] S. B. Davis, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, vol. ASSP-28, no. 4, 1980.

[52] J. C. Wang, J. F. Wang, and Y. S. Weng, "Chipdesign of mfcc extraction for speech recognition," *J. VLSI Integration*, vol. 32, pp. 111 – 131, 2002.

[53] "Spss advanced statistics users guide," *SPSS, Inc., Chicago, IL,*, 1990.

[54] G. H. Golub and C. F. V. Loan, "Matrix computations," *The Johns Hopkins University Press*, 1996.

[55] M. Buchler, S. Allegro, S. Launer, and N. Dillier, "Sound classification in hearing aids inspired by auditory scene analysis," *EURASIP Journal on Applied Signal Processing*, no. 18, pp. 2991 – 3002, March 31 - April 4 2005.

[56] T. Savell, "The emu10k1 digital audio processor," *Digital Object Identifier 10.1109 40.755467*, vol. 19, no. 2, pp. 49 – 57, March-April 1999.

[57] A. Oermann, A. Lang, and J. Dittmann, "Verifier-tuple for audio-forensic to determine speaker environment," *ACM, Proceedings of the 7th workshop on Multimedia and security*, pp. 57 – 62, March-April 2005.

[58] C. Panagiotakis and G. Tziritas, "A speech music discriminator based on RMS and zero-crossings," *IEEE Transactions on Multimedia*, vol. 1, no. 7, pp. 155 – 166, Febuary 2005.

[59] G. Freeman, R. Dony, and S. Areibi, "Audio environment classication for hearing aids using artificial neural networks with windowed input," *IEEE Symposium on Computational Intelligence in Image and Signal Processing*, vol. 2846, pp. 183 – 188, April 2007.

[60] H. Deshpande, R. Singh, and U. Nam, "Classification of music signals in the visual domain," *Proc. the COSTG6 Conference on Digital Audio Effects*, 2001.

[61] I. Paraskevas and E. Chilton, "Audio classification using acoustic images for retrieval from multimedia databases," *EURASIP Conference focused on Video/Image Processing and Multimedia Communications*, vol. 1, pp. 187 – 192, July 2003.

[62] S. Esmaili, S. Krishnan, and K. Raahemifar, "Content based audio classification and retrieval using joint time-frequency analysis," *In Proc. of Inter. Conference Acoustics, Speech, and Signal Processing*, vol. 5, pp. 665 – 668, May 2004.

[63] P. Lysaght, "The field programmable logic perspective," *20th International Conference on Field Programmable Logic and Applications*, 2010.

[64] V. Paliouras, J. Vounckx, and D. Verkest, *Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation*. The Springer, 2005, iSBN 0-07-016333-2.

[65] Altera Corporation. Software development tools for the NiosII processor. [Online]. Available: http://www.altera.com/products/ip/processors/nios2/tools/ide/ni2-ide.html

[66] B. Ghoraani, “Time-frequency Feature Analysis,” *Ph.D. thesis, Ryerson University*, 2010.