# Front-end of Wake-Up-Word Speech Recognition System Design on FPGA

**by**

**Mohamed Muftah Eljhani**

**Master of Science**
**in Computer Information Systems**
**Florida Institute of Technology**
**July 2014**

**Master of Science**
**in Computer Engineering**
**Beijing University of Aeronautics & Astronautics**
**December 2004**

**A dissertation submitted to the College of Engineering at**
**Florida Institute of Technology**
**in partial fulfillment of the requirements**
**for the degree of**

**Doctor of Philosophy**
**In**
**Computer Engineering**

**Melbourne, Florida**
**December, 2014**

We the undersigned committee
hereby approve the attached dissertation as fulfilling in part the
requirements for the degree of
Doctor of Philosophy of Computer Engineering

Front-end of Wake-Up-Word Speech Recognition System Design on FPGA

by

Mohamed Muftah Eljhani

| | |
|---|---|
| Veton Z. Këpuska, Ph.D.<br>Associate Professor<br>Electrical and Computer Engineering<br>Committee Chair | Samuel P. Kozaitis, Ph.D.<br>Professor and Department Head<br>Electrical and Computer Engineering |
| Ivica Kostanic, Ph.D.<br>Associate Professor<br>Electrical and Computer Engineering | Marius C. Silaghi, Ph.D.<br>Associate Professor<br>Computer Science |

## Abstract

**Title**

Front-end of Wake-Up-Word Speech Recognition System Design on FPGA

**Author**

Mohamed Muftah Eljhani

**Major Advisor**

Veton Z. Këpuska, Ph.D.

A typical speech recognition system is push button operated (Push-to-talk), which requires hand movement and hence mixed multi-modal interface. However, for disabled patients and those who use hands-busy applications (e.g., where the user has objects to manipulate or device to control while asking for assistance from another device) movement may be restricted or impossible. The only alternative is to use Speech Only Interface. The method that is being proposed is called Wake-Up-Word Speech Recognition (WUW-SR). A WUW-SR system would allow the user to operate (activate) many systems (Cell phone, Computer, Elevator, etc.) with speech commands instead of hand movements.

This work defines a new front-end paradigm of the Wake-Up-Word Speech Recognition on Field Programmable gate Arrays (FPGA). The-State-Of-The-Art Front-end of WUW-SR system is based on three different

subsystems that produce three sets of features: (1) Mel-frequency Cepstral Coefficients (MFCC), (2) Linear Predictive Coding Coefficients (LPC), and (3) Enhanced Mel-frequency Cepstral Coefficients (ENH_MFCC). These extracted features are then compressed and transmitted to the server via a dedicated channel, where subsequently they are decoded. These features are decoded with corresponding Hidden Markov Models (HMMs) in the back-end stage of the WUW-SR.

In the WUW-SR system, the front-end processor is located at the terminal (e.g. Mobile phone) which is typically connected over a data network to remote back-end recognition (e.g., server). WUW's front-end can be added to any hand-held electronic device compatible with WUW-SR and command (activate) it by using our voice only (no push to talk as is presently done).

WUW's front-end is designed, and implemented in Altera DSP development kit with Cyclone III FPGA as a portable system acting as a processor that is capable of computing three different sets of features at a much faster rate than software. It is cost effective, consumes very little power, and it is not limited by having to operate on a general-purpose computer so it can be used on any portable device.

# Contents

## Keywords

Automatic Speech Recognition

Digital Signal Processing (DSP)

Enhanced Mel-frequency Cepstral Coefficients (ENH-MFCC)

Feature Extraction

Field-Programmable Gate Arrays (FPGA)

Front-end

Hardware Description Language (HDL)

Linear Predictive Coding (LPC)

Mel-frequency Cepstral Coefficients (MFCC)

Speech Signal Processing (SSP)

Voice Activity Detector (VAD)

Wake-Up-Word Speech Recognition (WUW-SR)

# Figures

## Acronyms & Abbreviations

| | |
|---|---|
| **ADC** | Analog-To-Digital Converter |
| **ALTFP** | Altera Floating-Point |
| **ARM** | Advanced RISC Machine |
| **ASIC** | Application-Specific Integrated Circuit |
| **ASR** | Automatic Speech Recognition |
| **AVR** | Advanced Virtual RISC |
| **BSD** | Berkeley Software Distribution |
| **CA** | Correct Acceptance |
| **CLB** | Configurable Logic Blocks |
| **CODEC** | Encoder-Decoder |
| **CPLD** | Complex Programmable Logic Device |
| **CPU** | Central Processing Unit |
| **CR** | Correct Rejection |
| **DAC** | Digital to Analog Converter |
| **DC** | Direct Current |
| **DCT** | Discrete Cosine Transform |
| **DDR SDRAM** | Double Data Rate SDRAM |
| **DEL** | Deletion |
| **DFT** | Discrete Fourier Transform |

| | |
|---|---|
| **DIF** | Decimation-Infrequency |
| **DPRAM** | Dual Port Random Access Memory |
| **DSP** | Digital Signal Processing |
| **DTW** | Dynamic Time Warping |
| **DUT** | Device Under Test |
| **DV** | Data Valid |
| **EDA** | Electronic Design Automation |
| **EEPROM** | Electrically Erasable Programmable ROM |
| **ENH-MFCC** | Enhanced-Mel-Frequency Cepstral Coefficients |
| **FFT** | Fast Fourier Transform |
| **FIFO** | First-In, First-Out |
| **FIR** | Finite Impulse Response |
| **FP** | Floating-Point |
| **FPGA** | Field-Programmable Gate Array |
| **GPL** | General Public License |
| **GUI** | Graphic User Interface |
| **HDL** | Hardware Description Language |
| **HMM** | Hidden Markov Model |
| **HPF** | High Pass Filter |
| **HSMC** | High-Speed Mezzanine Card |

| | |
|---|---|
| **I²C** | Inter-Integrated Circuit |
| **IEEE** | Institute of Electrical And Electronics Engineers |
| **INS** | Insertion |
| **INV** | In Vocabulary |
| **I/O** | Input/output |
| **IP** | Intellectual Property |
| **JTAG** | Joint Test Action Group |
| **KB** | Kilo Byte |
| **LCD** | Liquid Crystal Display |
| **LE** | Logic Element |
| **LP** | Low Power |
| **LPC** | Linear Predictive Coding |
| **LPCC** | Linear Predictive Coding Coefficients |
| **LPF** | Low Pass Filter |
| **LS** | Lowest Power with Security |
| **LSB** | Least Significant Bit |
| **LUT** | Look-Up Table |
| **MFCC** | Mel-Frequency Cepstral Coefficients |
| **MHz** | Mega Hertz |
| **MSB** | Most Significant Bit |

| | |
|---|---|
| **OOV** | Out Of Vocabulary |
| **PC** | Personal Computer |
| **PCB** | Printed Circuit Board |
| **PLD** | Programmable Logic Device |
| **PLP** | Perceptual Linear Predictive Coefficients |
| **RAM** | Random Access Memory |
| **RISC** | Reduced Instruction Set Computer |
| **ROM** | Read Only Memory |
| **RTL** | Register Transfer Level |
| **RTR** | Run-Time Reconfiguration |
| **SDRAM** | Synchronous Dynamic RAM |
| **SNR** | Signal-To-Noise Ratio |
| **SOC** | System-On a Chip |
| **SOPC** | System-On a Programmable Chip |
| **SPI** | Serial Peripheral Interface |
| **SR** | Speech Recognition |
| **SRAM** | Static RAM |
| **SUB** | Substitution |
| **SVM** | Support Vector Machines |
| **TB** | Test-Bench |

| | |
|---|---|
| **TSMC** | Taiwan Semiconductor Manufacturing Company |
| **VAD** | Voice Activity Detector |
| **VERILOG HDL** | Verifying Logic Hardware Description Language |
| **VHDL** | Very High Speed Integrated Circuit Hardware Description Language |
| **VLSI** | Very Large Scale Integrated Circuit |
| **WER** | Word Error Rate |
| **WUW** | Wake-Up-Word |

## Acknowledgments

I would like to express my gratitude toward Dr. Veton Z. Këpuska as my Faculty Advisor for pushing this research work forward, always being available to help and support.

I would like to thank Dr. Samuel P. Kozaitis as the head of Electrical and Computer Engineering Department, one of my Ph.D. Committee for his valuable support and providing us with the Altera Software Licenses and FPGA Development Kits.

Special thanks to Brian H. Hight for leading us his knowledge and experience on FPGA design.

Finally, we thank Altera for providing the tools for the project, including the software, documentation about this contest, and a platform to showcase our designs.

# Dedication

**For my Wife**
**A**

**&**

**For my Kids**
**A, M, W, Y, and Y**

# 1 Introduction

"Operator (WUW for Elevator Simulator). Take me to the last floor. " Operator responds "Taking you to the last floor." The ideas of being able to talk to a machine and have it understand you have been a reoccurring theme in science fiction for decades. While we are not yet at the stage where electronic machines can comprehend our every word and act on it, these machines are becoming ever more complex and ubiquitous.

WUW is a new area in speech recognition. The WUW recognizer is a highly efficient and accurate recognizer specializing in the detection of a single word or phrase when spoken in the context of requesting attention, while rejecting all other words, phrases, sounds, noises and other acoustic events with virtually 100% accuracy.

Continuous speech recognition has been acknowledged as one of the most challenging problems today. There are many issues that contribute to the difficulty of automatically recognizing human speech such as corruption of noise, variability of the speaker and speaking mode, change of environment conditions, inaccuracy of model assumption, complexity of language, etc. In addition, as a statistical model based system, a speech recognizer demands sufficient, well transcribed speech data for the model

training in order to achieve satisfactory performance. It is often intractable to fulfill this requirement since the time and expense spent on speech data collection and transcription are not always affordable for many real word applications.

For the past several decades, designers have processed speech for a wide variety of applications ranging from mobile communications to automatic reading machines. Speech has not been used much in the field of electronics and computers due to the complexity and verity of speech signals. However with modern processes and methods we can process speech signals in Field-Programmable Gate Array (FPGA) chips.

While others concentrate on developing the algorithms and models, there still remains the question of how to implement them on programmable chip. Several speech recognition software packages already exist that can run on a PC, including the Wake-Up-Word Speech Recognition System; however, they are limited by having to operate on a general-purpose processor. In the end, to achieve the maximum processing power, application-specific hardware is the answer.

A great deal of work has been conducted in this dissertation to address this problem by designing an efficient hardware front-end of State-Of-The-Art WUW-SR with an FPGA using an DSP Altera-based system,

acting as a processor that is responsible for extracting three types of features from the input audio signal. These features are Mel-frequency Cepstral Coefficients (MFCC), Linear Predictive Coding Coefficients (LPC), Enhanced Mel-frequency Cepstral Coefficients (ENH-MFCC).

## 1.1 Aims and Objects

One of the goals of speech recognition is to allow natural communication between humans and computers. A major obstacle to this is the fact that most systems today still rely to some extent on non-speech input. For example, some systems use a "push to talk" model, meaning that speech recognition is only activated when the user pushes a button. Systems that are "continuously listening" usually suffer from poor accuracy, especially if they are speaker independent.

The WUW Speech Recognizer is a complex system comprising of three major parts: the front-end, Voice Activity Decoder (VAD), and back-end, which have been implemented entirely in C++ and are capable of running live and performing recognitions in real time (V. Z. Këpuska and T. B. Klein) [1]. The aim of this research is to design and implement front-end with built-in VAD of WUW Speech Recognizer in hardware, which is responsible for generating three sets of features from the input audio signals:

- ***MFCC [Mel-frequency Cepstral Coefficients]***
- ***LPC [Linear Predictive Coding Coefficients]***
- ***ENH-MFCC [Enhanced Mel-frequency Cepstral Coefficients]***

These features need to be decoded with corresponding HMMs in the back-end stage of the WUW Speech Recognizer.

We aim to produce an efficient hardware front-end system with an FPGA portable system acting as a processor that is capable of computing three different sets of features at a much faster rate than software. It is cost effective, consumes very little power, and it is not limited by having to operate on a general-purpose processor so it can be used on any portable device. Our state-of-the-art front-end is different from other front-end designs. It has the capability of computing and producing three different sets of features simultaneously.

## 1.2 Automatic Speech Recognition

Automatic speech recognition (ASR) is the computer's ability to convert a speech audio signal into its textual transcription (Xuedong Huang, Alex Acero, Hsiao-Wuen Hon) [2]. Some motivations for building ASR systems are presented in order of difficulty to improve human-computer interaction through spoken language interfaces, to solve difficult problems

such as speech to speech translation, and to build intelligent systems that can process spoken language as proficiently as humans, (Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, Victor Zue (Eds.)) [3].

Speech as a computer interface has numerous benefits over traditional interfaces such as a Graphic User Interface (GUI) with mouse and keyboard. Speech is natural for humans, requires no special training, improves multitasking by leaving the hands and eyes free, and is often faster and more efficient to transmit than using conventional input methods.

Though many tasks are solved with visual, pointing interfaces and/or keyboards, speech has the potential to be a better interface for a number of tasks where full natural language communication is useful, (Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, Victor Zue (Eds.)) [3] and the recognition performance of the speech recognition system is sufficient to perform the tasks accurately (V. Këpuska) [4] , (T. Klein) [5], in circumstances where the user wants to multitask while asking for assistance from the computer may include hands-busy and eyes-busy applications where the user has objects to manipulate or equipment and/or devices to control.

## 1.3 Wake-Up-Word Speech Recognition

Novel speech recognition technology named Wake-Up-Word Speech Recognition (WUW-SR) (V. Këpuska) [6], (V. Këpuska) [7] bridges the gap between natural language and other voice recognition tasks (V. Këpuska, T. Klein) [8]. WUW-SR is a highly efficient and accurate recognizer specializing in the detection of a single word or phrase when spoken in the alerting or WUW context (V. Këpuska, D.S. Carstens, R. Wallace) [9] of requesting attention, while rejecting all other words, phrases, sounds, noises and other acoustic events with virtually 100% accuracy including the same word or phrase uttered in non-alerting (referential) context.

The WUW speech recognition task is similar to keyword spotting. However, WUW-SR is different in one important aspect: to the ability identify the specific word or phrase used in alerting context (and not other contexts; e.g. referential). Specifically, the sentence, ``Computer, begin PowerPoint presentation'' exemplifies the use of the word ``computer'' in alerting context. On the other hand, in the sentence, "My computer has dual Intel 64-bit processors, each with quad cores'' the word 'computer' is used in a referential (non-alerting) context. Traditional keyword spotters will not be able to discriminate between the two cases. The discrimination will be only possible by deploying higher level natural language processing subsystem in

order to discriminate between the two. However, for applications, deploying such solutions is practically impossible. It is very difficult to determine in real-time if the user is speaking to the computer or about the computer. Traditional approaches to keyword spotting are usually based on large vocabulary word recognizers or phone recognizers (J.R. Rohlicek, W. Russell, S. Roukos, H. Gish) [10] , or whole-word recognizers that either use HMMs or word templates, (C. Myers, L. Rabiner, A. Rosenberg) [11]. Word recognizers require tens of hours of word-level transcriptions as well as a pronunciation dictionary. Phone recognizers require phone marked transcriptions and whole-word recognizers require word markings for each of the keywords, (A. Garcia, H. Gish) [12].

The word-based keyword recognizers are not able to find words that are out-of-vocabulary (OOV). To solve the problem of OOV keywords, spotting approaches based on phone-level recognition have been applied to supplement or replace systems based upon large vocabulary recognition ,(D.A. James, S.J. Young) [13]. In contrast the (V. Z. Këpuska and T. B. Klein) [1] approach to WUW-SR is independent to what is the basic unit of recognition (word, phone or any other sub-word unit). Furthermore, it is capable to discriminate whether or not the user is speaking to the recognizer without deploying language modeling and natural language understanding

methods. This feature makes WUW-SR task distinct from keyword spotting.

### 1.3.1 Wake-Up-Word Applications

The applications of WUW-SR are numerous, and the following discussion mentions some of the possibilities.

1. Automobiles - Current voice controlled navigation systems and entertainment systems make use of the "push to talk" paradigm. WUW would enable completely hands free communication.
2. Conference calls - In the business world it is very important to be able to retrieve information on the fly and dynamically manage the participants in a telephone conference call. Currently, adding or dropping users from the conference call is not a trivial undertaking and requires knowledge of the particular telephone system. With WUW, it is be possible to achieve such a dynamic control by simply invoking the system via the WUW and issuing the command thereafter. For example, "operator... <beep> connect John. <percolating sound>."
3. Smart room - With current wireless communications technology and microphone arrays it is possible to interface a computer to all appliances and electronic devices. Adding a speech interface to the

system would allow one to control the electronic devices in a room via voice commands. However, adding a speech interface is currently not practical because state of the art recognizers are insufficiently accurate and robust in the "continuously listening" mode. However, WUW could be used as an interface to the command & control speech recognizer and make this a practical solution.

4. People with disabilities - From the onset, speech recognition technology have been viewed as indispensible to improve the lives of people with disabilities. WUW will further improve the technologies that these people depend on and improve the quality of their lives.
5. Military - Through a personal communication it was conveyed that a military personnel was endangered due to overly restrictive usage requirements of automatic translation system that required its user to push the button when speaking to it. In order to do that, this military officer was required to drop his weapon to free his hands in order to use the device. At that moment he became vulnerable and was attacked.
6. Airlines - Airplane pilots currently spend at least 30 minutes to program their flight plan through a tedious manual data entry procedure. Clearly speech recognition and more so WUW technology

are ideal solutions that would improve productivity of the pilot.

7. Other Uses - Other uses of WUW include the many situations where it is impractical to provide manual input to computer system. However, current speech recognizers are not accurate and robust enough to accommodate the requirements of the problem.

### 1.3.2 Wake-Up-Word Definition

As explained in (V. Z. Këpuska and T. B. Klein) [1], WUW technology solves three major problem areas:

1. ***Detecting WUW Context:*** The Wake-Up-Word is proposed as a means to grab the computer's attention with extremely high accuracy. Unlike keyword spotting, the recognizer should not trigger on every instance of the word, but rather only in certain contexts. For example, if the Wake-Up-Word is "computer," the WUW should not trigger if spoken in a sentence such as "I am now going to use my computer."
2. ***Identifying WUW:*** The Voice Activity Detector (VAD) is responsible for finding utterances spoken in the correct context and segmenting them from the rest of the audio stream. After the VAD makes a decision, the next task of the system is to identify whether or not the segmented utterance is a WUW.

3. ***Correct Rejection of Non-WUW***: The final and critical task of a WUW system is correctly rejecting Out-Of-Vocabulary (OOV) speech. The WUW-SR system should have a correct rejection rate of virtually 100% while maintaining high correct recognition rates of over 99%.

### 1.3.3 WUW-SR Different from other SR Systems

Wake-Up-Word is often mistakenly compared to other speech recognition tasks such as keyword spotting or command and control; but WUW-SR is different from the previously mentioned tasks in several significant ways. The most important characteristic of a WUW-SR system is that it should guarantee virtually 100% correct rejection of non-WUW and same words uttered in non-alerting context while maintaining correct acceptance rate over 99%. This requirement sets WUW-SR apart from other speech recognition systems because no existing system can guarantee 100% reliability by any measure without significantly lowering its correct recognition rate. It is this guarantee that allows WUW-SR to be used in novel applications that previously have not been possible. Secondly, a WUW-SR system should be context dependent; that is, it should detect only words uttered in alerting context. Unlike keyword spotting, which tries to find a specific keyword in any context, the WUW should only be recognized when

spoken in the specific context of grabbing the attention of the computer in real time. Thus, WUW recognition can be viewed as a refined, albeit significantly more difficult, keyword-spotting task. Finally, WUW-SR should maintain high recognition rates in speaker-independent or speaker-dependent mode and in various acoustic environments (V. Z. Këpuska and T. B. Klein) [1].

### 1.3.4 Significant of WUW-SR

The reliability of WUW-SR opens up the world of speech recognition to applications that were previously impossible. Today's speech-enabled human-machine interfaces are still regarded with skepticism and people are hesitant to entrust any significant or accuracy-critical tasks to a speech recognizer.

Despite the fact that Speech Recognition is becoming almost ubiquitous in the modern world, widely deployed in mobile phones, automobiles, desktop, laptop, and palm computers, many handheld devices, telephone systems, etc., the majority of the public pays little attention to speech recognition. Moreover, the majority of speech recognizers use the push to talk paradigm rather than continuously listening, simply because they are not robust enough against false-positives. One can imagine that the

driver of a speech-enabled automobile would be quite unhappy if his or her headlights suddenly turned off because the continuously listening speech recognizer misunderstood a phrase in the conversation between driver and passenger.

The accuracy of speech recognizers is often measured by Word Error Rate (WER), which uses three measures, (Xuedong Huang, Alex Acero, Hsiao-Wuen Hon) [2]:

- Insertion (INS) _ an extra word was inserted in the recognized sentence.
- Deletion (DEL) _ a correct word was omitted in the recognized sentence.
- Substitution (SUB) _ an incorrect word was substituted for a correct word.

  WER is defined as:

$$WER = \frac{INS + DEL + SUB}{TRUE\ NUM.\ OF\ WORDS\ IN\ SENTENCE} 100\%.$$

Substitution errors or equivalently correct recognition in WUW context represent the accuracy of the recognition, while insertion + deletion errors are caused by other factors; typically erroneous segmentation.

However, WER as an accuracy measurement is of limited usefulness to a continuous listening command and control system. To understand why,

consider a struggling movie director who cannot afford a camera crew and decides to install a robotic camera system instead.

The system is controlled by a speech recognizer that understands four commands: “lights”, “camera”, “action”, and “cut”. The programmers of the speech recognizer claim that it has 99% accuracy, and the director is eager to try it out. When the director sets up the scene and utters “lights”, “camera”, “action”, the recognizer makes no mistake, and the robotic lights and cameras and spring into action. However, as the actors perform the dialogue in their scene, the computer misrecognizes “cut” and stops the film, ruining the scene and costing everyone their time and money. The actors could only speak 100 words before the recognizer, which truly had 99% accuracy, triggered a false acceptance.

This anecdote illustrates two important ideas. First, judging the accuracy of a continuously listening system requires using a measure of “rejection”. That is, the ability of the system to correctly reject out-of-vocabulary utterances. The WER formula incorrectly assumes that all speech utterances are targeted at the recognizer and that all speech arrives in simple, consecutive sentences. Consequently, performance of WUW-SR is measured in terms of correct acceptances (CA) and correct rejections (CR). Because it is difficult to quantify and compare the number of CRs, rejection

performance is also given in terms of “false acceptances per hour”.

The second note of interest is that 99% rejection accuracy is actually a very poor performance level for a continuously listening command and control system. In fact, the 99% accuracy claim is a misleading figure. While 99% acceptance accuracy is impressive in terms of recognition performance, 99% rejection implies one false acceptance per 100 words of speech. It is not uncommon for humans to speak hundreds of words per minute, and such a system would trigger multiple false acceptances per minute. That is the reason why today's speech recognizers: a) primarily use a “push to talk” design and b) are limited to performing simple convenience functions and are never relied on for critical tasks. On the other hand, experiments have shown the WUW-SR implementation presented in Këpuska & Klein, 2009 work reach up to 99.99% rejection accuracy. That translates to one false acceptance every 2.2 h (V. Z. Këpuska and T. B. Klein) [1].

### 1.3.5 Wake-Up-Word-SR Implementation

The concepts of WUW-SR have been most expanded in A novel Wake-Up-word speech recognition system, Wake-Up-Word recognition task, technology and evaluation (V. Z. Këpuska and T. B. Klein) [1]. Currently, the system is implemented in C++ as well as JAVA. The WUW Speech Recognizer

consists of three stages. The first is front-end (audio signal preprocessing) stage, described with VAD in more detail below as it is our goal in this dissertation. It takes a speech waveform as its input, and extracts from it three different sets of features: MFCC, LPC, and ENH-MFCC. Features represent the information required to perform recognition in the back-end stage S.P. Davis, P. Mermelstein) [14], (John Makhoul) [15].

The second stage is Voice Activity Detector (VAD), which is performed using a set of statistical models called Hidden Markov Models (HMM). VAD segments the signal into speech and non-speech regions.

The third stage is back-end. The back-end is responsible for classifying observation sequences as in-vocabulary (INV), (i.e. the sequence of frames hypothesized as a segment is a Wake-Up-Word), or out of vocabulary (OOV), (i.e. the sequence is not a Wake-Up-Word). The WUW-SR system uses a combination of HMM and Support Vector Machines (SVM) for acoustic modeling and, as a result the back-end, consists of an HMM recognizer and a SVM classifier. Prior to recognition, HMM and SVM models must be created and trained for the word or phrase which is to be the wake-up-word. When the VAD state changes from VAD_OFF to VAD_ON, the HMM recognizer resets and prepares for a new observation sequence. As long as the VAD state remains VAD_ON, feature vectors are continuously

passed to the HMM recognizer, where they are scored using the novel triple-scoring method. If using multiple feature streams, recognition is performed for each stream in parallel. When VAD state changes from VAD_ON to VAD_OFF, multiple scores are obtained from the HMM recognizer and are sent to the SVM classifier. SVM produces a classification score which is compared against a threshold to make the final classification decision of INV or OOV.

#### *1.3.5.1 Front End Signal Processing*

Front-end signal processing plays a crucial role in the realization of speech recognition systems. This is the result of the fact that better signal feature extraction leads to better recognition performance. The goal of front-end signal processing is to extract relevant feature parameters of speech, which are more suitable for the purpose of speech recognition than the input speech waveform itself in terms of information rate and redundancy reduction. Therefore, intensive efforts have been carried out to achieve a high performance front-end. WUW-SR system makes use of the modulation applied by the vocal tract (throat, tongue, teeth, lips and nasal cavity); the excitation produced by the larynx is not used, even though humans infer much information from it. (Note that in a number of Far-Eastern languages,

the inflection of a syllable can profoundly affect its meaning, requiring this information to be retained, (Lee, L.S., Tseng, C.Y., Lin, Y.H., Lee, Y., Tu, S.L., Gu, H.Y., Liu, F.H., Chang, C.H., Hsieh, S.H., Chen, C.H. & Huang, K.R)) [16].

## 1.4 Field-Programmable Gate Arrays (FPGAs)

The field-programmable gate arrays (FPGA) is a semiconductor device that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware function, an FPGA allows you to program product features and functions, adapt to new standards and reconfigure hardware for specific applications even after the product has been installed in the field—hence the name "field-programmable". You can use an FPGA to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but no ability to update the functionality after shipping offers advantages for many applications.

Unlike previous generation FPGAs using I/Os with programmable logic and interconnects, today's FPGAs consist of various mixes of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Specifically, an FPGA contains programmable logic components called logic elements (LEs) and a hierarchy of reconfigurable interconnects that allow the LEs to be physically connected. You can

configure LEs to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip flops or more complete blocks of memory.

As FPGAs continue to evolve, the devices have become more integrated. Hard intellectual property (IP) blocks built into the FPGA fabric provide rich functions while lowering power and cost and freeing up logic resources for product differentiation. Newer FPGA families are being developed with hard embedded processors, transforming the devices into systems on a chip (SoC), (Altera, Inc) [17].

A field-programmable gate array (FPGA) is a form of programmable logic device (PLD).It typically consists of a rectangular array of configurable logic blocks (CLBs). Each CLB can contain assorted logic resources such as look-up tables (LUTs) capable of implementing any desired Boolean function and dedicated arithmetic logic such as carry chain logic, registers, latches, shift registers, distributed memories, and so on.

The resources within a CLB can be configured as required. Similarly, the data lines that link the resources within the CLB can be configured in order to connect them together in particular ways. The CLB array itself is immersed in a web of configurable routing, allowing the CLBs to be

connected in myriad ways.

There is currently a trend towards combining fixed-function logic with reconfigurable logic, producing a so-called “system on a chip” (SoC). This started with the inclusion of blocks of dedicated RAM — themselves configurable with regard to the widths of their address and data buses — and now includes dedicated multipliers, DSP blocks, and processor cores, (Melnikoff, S.J) [18].

The field-programmable part of an FPGA comes from the fact that FPGAs can be programmed and reprogrammed in situ, without having to be removed from their target PCB and placed in a chip programmer every time a new design needs to loaded, as is the case with some other types of Programmable Logic Device (PLD). Most FPGAs are now SRAM based, and so require a separate ROM to store their configuration data, as they are unable to retain this data themselves when switched off. With so many resources at the designer’s disposal, an FPGA provides a very powerful platform for hardware development. Its flexibility allows for all manner of complex designs; its numerous resources allow for a great deal of parallelism, if the application allows it, and its ability to be reprogrammed without limitation makes it an invaluable tool for hardware development. This is not, however, the only thing that FPGAs are good for. Making ASICs is a very

expensive process, and as feature size shrinks, the cost of producing the die is increasing.

The economics are such that a manufacturer needs to expect to ship a very large number of chips before producing an ASIC becomes cost effective which currently of the order of hundreds of thousands for the smaller feature sizes, and continuing to rise, (Makimoto, T ) [19], (Makimoto, T ) [20]. For smaller quantities, an FPGA or other PLD is cheaper. Additionally, an FPGA's in-system programmability can be put to other uses. Unlike an ASIC, the FPGA's design can be updated after the PCB has been made and populated and after the product has been deployed; akin to software patches being downloaded after a product has been shipped.

Taking this a stage further, one chip can be supplied with a library of designs, enabling it to perform different functions depending on the situation. For example, an FPGA could be used as part of a communications subsystem, with different configurations for different protocols, allowing hardware acceleration for all of them with just one chip.

Some FPGAs allow parts of the device to be reconfigured, while leaving the rest of the chip untouched. The suggestion has therefore been made for run-time reconfiguration (RTR) (e.g., (Laufer, R., Taylor, R.R. & Schmit, H., "PCI-PipeRench and the SWORDAPI) [21], (Sezer, S., Heron, J.,

Woods, R., Turner, R. & Marshall, A)) [22], where some or all of the chip is reprogrammed at run time, provides more processing power than might otherwise be available.

Unfortunately, RTR has not been as successful as hoped for a number of reasons. Firstly, the reconfiguration times for FPGAs, particularly the larger ones, is along the order of milliseconds, which is a lifetime for devices that can operate at hundreds of megahertz. To illustrate this, (James-Roxby and Blodget) [23] use RTR to update the contents of LUTs configured as ROMs, and compares this with the alternative of configuring them as RAMs instead. The authors report that while the RAM-based design has a slower clock-speed and uses more resources, the LUTs can be updated much faster, by a factor of over 100.

Secondly, for partial reconfiguration, reprogramming one chunk of an FPGA affects the routing in neighboring areas, and there is currently no obvious solution as to how to deal with that. The problem can be sidestepped by limiting the reconfiguration to replacing the contents of LUTs or RAM, or by constraining the placement of logic resources so that no routing crosses areas that will be reconfigured.

Thirdly, any complex chip relies heavily on the software that supports it and current tools have limited support for RTR-based designs. FPGA

design software continues to improve, but still requires a lot of skill of the designer. indeed, the question of whether adapting software languages in order to make it easier for software engineers to produce FPGA designs ("C-to-gates") is an ongoing debate.

Additionally, a commercial slant is mentioned in, (IEE, "FPGAs not ready to go embedded," IEE Review, Institution of Electrical Engineers, April 2003) [24]: "There's no market for reconfigurability [right now]. There is a degree of reconfigurability in cellular systems, as in for changing the protocols as you move between countries, but that is a specialist area and is done by software. The case of design reconfigurability in hardware is yet to be proved, as software is a pretty good way of achieving reconfigurability."

At present, FPGAs' power-hungry nature makes them unsuitable for mobile devices. However, once that changes, their versatility and ability to be repeatedly updated—even if only once in a while—could see them become much more widespread than they are now, (Melnikoff, S.J) [18].

## 1.5 Motivation

Some motivations for building Automatic Speech Recognition (ASR) systems, presented in order of difficulty, are to improve human-computer interaction through spoken language interfaces, to solve difficult problems

such as speech-to-speech translation, and to build intelligent systems that can process spoken language as proficiently as humans, (Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, Victor Zue (Eds.)) [3]. Speech as a computer interface has numerous benefits over traditional interfaces using mouse and keyboard: speech is natural for humans and requires no special training, improves multitasking by leaving the hands and eyes free, and is often faster and more efficient to transmit than the information provided than using conventional input methods. Human-machine interaction is likely to take place in natural language in future embedded systems and mobile devices. Speech enabled car navigation; natural language e-learning applications and home automation are among those applications. This inherits all the embedded systems design constraints to the speech recognition domain, like limited hardware, memory, power consumption and cost, which creates the need to re-architecture the already existing speech recognition systems (V. Z. Këpuska and T. B. Klein) [1]. The-State-of-the-art WUW-SR is heading towards embedded systems and hand-held devices. WUW-SR front-end system architecture emerged to address these kinds of applications. The existing implementation of this system is presented in software fashion, with little consideration to the end product platform in which the system will be

deployed. In this dissertation, a hardware implementation of the front-end of WUW-SR is specified and presented in FPGA platform prototype, with consideration of migration to structure ASIC in case of mass production.

Finally: Why use an FPGA? It was originally suggested that this project make use of an FPGA. While there are much excitement (in academic circles, at least) that the FPGA's unique ability to be reconfigured on the fly could be put to great use, the challenge of doing so and the unlimited support of the tools, combined with the ever-increasing quantity of resources available on the device, has seen the idea pushed forward, the FPGA's great value has been shown in its use as a prototyping platform, either as a stepping-stone on the path to an ASIC, or as an end in itself, where an ASIC is either undesirable or uneconomical.

To conclude, even though processor power is always increasing, ASICs and programmable logic devices are subject to the same improvements in technology. Therefore, whatever we can improve using software; we should also be able to improve by using hardware.

# 2 Speech Recognition System on Programmable Chip

It is only in the last few years that desktop PCs have been powerful enough to allow large-vocabulary continuous speech recognition to be performed in real time in software.

At present, for best results, systems still rely on being trained to recognize one speaker, with minimal background noise. Even then, steps have to be taken in order to reduce the computational complexity so that real time recognition is feasible. Before this was possible, or when it was necessary to try out more complex algorithms, only hardware had the computational resources to achieve this.

Initially, hardware implementations tended to be based on parallel arrays of one kind or another, often using custom chips. As the technology has improved, the focus has shifted towards serial implementations, once again making use of custom chips, microcontrollers or DSPs, since the appearance of the FPGA has been used as an experimental platform (Melnikoff, S.J) [18].

One of the three principal stages of the speech recognition process, it is the decoding part that takes center stage in hardware implementations. Pre-processing tends to be performed in software, or left to a DSP (G´omez-

Cipriano, J.L., Pizzatto Nunes, R., Bampi, S. & Barone, D (2001)) [25] use an FPGA for feature extraction.

As a matter of fact, the increased processing power now offered by processors and ASICs — not to mention the lower cost — has led to a shift towards such devices.

(Shozakai (1999)) [26] uses an ASIC containing a DSP core for feature extraction and Gaussian computations, and a RISC microprocessor core for the Viterbi decoding. Tied mixture Gaussian mixtures are used, with 54 Japanese monophone HMMs, (Nakamura, K., Zhu, Q., Maruoka, S., Horiyama, T., Kimura, S. & Watanabe, K) (2001)) [27] describe an embedded system incorporating an ASIC which also performs observation feature extraction and Viterbi decoding. Discrete HMMs of 5 states each, representing 64 monophones, are used. An FPGA is used for training. The authors report that the hardware, running at 17 MHz, can perform recognition in real time. They add that if their ASIC were operating at the same speed as the processor used for testing equivalent software (Pentium III 750), the ASIC would be 5.3 times faster.

In contrast to this, , (Shi, Y.Y., Liu, J. & Liu, R.S) (2001) [28] employs an ASIC containing an 8051 core for almost all the processing, including feature extracting, with only the minimum of support logic (mainly for analogue-

digital conversion). The rationale of not using a DSP core is that they are expensive in comparison—but the trade-off is the reduced processing power available. The system performs both training and recognition. The authors state that the chip is capable of accuracy above 90% for a constrained vocabulary.

What is clear from these implementations is that, although a system-on-chip design can recognize speech, current designs only have enough processing power to cope with small vocabularies and the simpler types of models.

## 2.1 Speech Recognition Software vs. Hardware Design

Providing a compromise between the processing power of hardware and the flexibility of software, the emergence of FPGAs in the 1990s provided a new platform for the development of speech recognition systems.

(Schmit & Thomas (1995)) [29] present an early FPGA implementation of an HMM based application, on a Xilinx 4000-series device. In this case, they use Viterbi decoding to correct errors made by a person typing, resulting in a system 25 times faster than equivalent software.

(Vargas, F.L., Fagundes, R.D.R. & Junior, D.B (2001)) [30] uses two Altera FPGAs to implement a simple isolated word recognition system. The

model uses up to 10 words, with 6 states per discrete HMM. They take advantage of parallelism within the Viterbi algorithm to achieve a speedup over software in the order of 500 times, with accuracies for this task approaching 100%.

A novel implementation is demonstrated by (Jou, J.M., Shiau, Y.H. & Huang, C.J. (2001)) [31] who proposed an "efficient VLSI architecture," prototyped on an FPGA. It takes advantage of the left-right nature of HMM state machines used in speech recognition by merging every four columns of the Viterbi trellis into one. The authors state that this approach saves on time and resources. While this could be useful for faster-than-real-time transcription, there is likely to be little gain when processing real-time speech, as the system would have to wait for the same amount of time between new observations whether it was processing one or four at a time.

(Stogiannos, P., Dollas, A. & Digalakis, V. (2000)) [32], based on (Stogiannos, P (1999)) [33]. They use discrete-mixture HMMs, in which the elements of the observation vector are quantized in advance, allowing the probability associated with each element to be looked up in an off-chip codebook, rather than calculated. These values (in the log domain) are then summed, converted to the linear domain using another look-up, and further summation takes place (as for Gaussian mixtures). The conversion back to

the log domain and the Viterbi decoding are performed in software. This approach uses a lot of external RAM: 64 Mb of SDRAM for the codebook values, and 512 Kb of SRAM for the domain conversion (organized as four 128 Kb LUT look-up tables).

In contrast, all but one of the designs (described later) use continuous probability distributions, and computes the mixture components on the FPGA. Use of an alternative algorithm removes the need for a domain conversion for the mixture component summation, greatly reducing the large storage and bandwidth requirements inherent in a RAM-based implementation. In addition, the Viterbi decoding is performed in hardware.

In all cases, the designs take advantage of more recent devices which are faster and have more resources available. The system is designed for an Altera FLEX 10KE running at 66MHz.

As now, one of the key points is the use of FPGAs as a more cost-effective solution for low-volume applications, though at the expense of lower processing speeds as ASICs.

## 2.2 Commercial Speech Recognition Systems

A small number of commercial speech recognition ASICs are exist, such as (Sensory's RSC-300/RSC-364 and RSC4x family) [34], which use a

RISC microprocessor with a neural network; their Voice Direct 364, which is also based on a neural network; and (Philips' SBF1005 HelloIC) [35], which is based on a DSP. All three are designed for applications requiring a small vocabulary (typically 60 words or less), and boast a speaker-independent recognition accuracy of 97% or more. (Further performance comparisons are not possible due to a lack of suitable information).

While recognition chips and intellectual property (IP) cores only handle small vocabularies,; their prevalence in toys, automotive applications and mobile phones suggests that the market for such devices in embedded and mobile systems will continue to increase as ,(Frostad, K.) [36], (Mozer, T) [37] noted.

With regards to FPGAs, there are no cores designed specifically for speech recognition. However, cores do exist for performing Viterbi decoding for signal processing, such as those produced by (TILAB) [38] and (Xilinx) [39]. In addition, some DSPs have dedicated logic for Viterbi decoding, like the( Texas Instruments) [40] TMS320C6416, and the TMS320C54x family.

In both cases, however, these decoders are designed for signal processing applications, which have different requirements from speech recognition, including narrower data widths, different data formats, and fewer states.

## 2.3 Alternative Speech Recognition Methods

The hidden Markov model is by far the dominant underlying algorithm used in speech recognition systems, both commercially and in research. However, there are alternatives that provide a useful comparison.

Dynamic time warping (DTW), (Cox, S.J) [41] predates HMMs, and is in fact a special case of HMMs. It works by comparing two utterances, stretching or compressing one (warping) in order to try and match it to the other. The degree to which the utterance is warped determines a value without transition probabilities, and with observation probabilities replaced by a distance metric (typically Euclidean or "Manhattan"). This value must be minimized in order to find the most likely match.

DTW was superseded by HMMs because the former provides less flexibility, as it cannot be made more robust by training on large amounts of data. Conversely, it has a use where data is limited, as a single utterance can be used as a template in lieu of training data.

Its relative simplicity was of use when implemented by (Shi, Y.Y., Liu, J. & Liu, R.S)(2001) [28], as described above.

Also mentioned earlier were neural networks. Rather than use any particular algorithm, a neural network is trained on a set of template patterns (e.g. a set of words used for command and control application). It is then

sent data extracted from incoming speech, and the data is compared to the templates. The neural network selects the most likely template or number of most likely candidates, with a final one being chosen after further processing.

Neural networks are simple to train, but their pattern-matching abilities are limited. They are suitable for recognizing a small number of isolated words, but they cannot cope with large-vocabulary continuous speech. Their inherent parallelism, however, does make them suitable for implementations in hardware, such as the FPGA version described by (Eldredge, J.G. & Hutchings, B.L., "RRANN (1994)) [42]. A more general approach is presented by (Chen, R. & Jamieson, L.H (1996)) [43].

Finally, a more unusual approach is introduced by (Bohez, E.L.J. & Senevirathne, T.R (2001)) [44].

They use fractals for clustering phonemes, and report that this method is good for endpoint detection and segmentation, but not dealing with whole words. It is suggested that this method on its own is not suitable for recognition, but could be used in conjunction with other techniques.

## 2.4 The-State-Of-The-Art Wake-Up-Word Speech Recognition

Having looked at how parts of the recognition process have been implemented before, it is now time to propose new designs, inspired and based on the theories described above.

Wake-Up-Word speech recognition system is a new paradigm in speech recognition that is not yet widely recognized, WUW-SR is defined as detection of a single word or phrase when spoken in the alerting context of requesting attention, while rejecting all other words, phrases, sounds, noises and other acoustic events and the same word or phrase spoken in non-alerting context with virtually 100% accuracy.

Novel speech recognition technology named Wake-Up-Word (WUW) (V. Këpuska) [6], ( V. Këpuska) [7] bridges the gap between natural-language and other voice recognition tasks (V. Këpuska, T. Klein) [8]. WUW-SR is a highly efficient and accurate recognizer specializing in the detection of a single word or phrase when spoken in the alerting or WUW context, (V. Këpuska, D.S. Carstens, R. Wallace) [9] of requesting attention, while rejecting all other words, phrases, sounds, noises and other acoustic events with virtually 100% accuracy including the same word or phrase uttered in non-alerting (referential) context. The WUW speech recognition task is similar to keyword spotting; however, WUW-SR is different in one important

aspect: to the ability discriminates the specific word or phrase used only in alerting context, not referential (e.g. conversational) context. Traditional keyword spotters will not be able to discriminate between the two cases. The discrimination will be only possible by deploying higher level natural-language processing subsystem in order to discriminate between the two. When deploying such solutions in applications it is nearly impossible to determine, in real–time, if the user is speaking to the computer or about the computer.

WUW speech recognizer is a highly efficient and accurate recognizer, specializing in the detection of a single word or phrase when spoken in the context of requesting attention (alerting), while rejecting the same word or phrase spoken under referential (non-alerting) context. It rejects all other words, phrases, sounds, noises and other acoustic events with virtually 100% accuracy. This high accuracy enables development of speech recognition driven interfaces that utilize dialogs using speech only.

One of the goals of speech recognition is to allow natural communication between humans and computers via speech, (Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, Victor Zue (Eds.)) [3], where natural speech implies similarity to the ways humans interact with each other on a daily basis. A

major obstacle to this is the fact that most systems today still rely to large extent on non-speech input, such as pushing buttons or mouse clicking. However, much like a human assistant, a natural speech interface must be continuously listening and must be robust enough to recover from any communication errors without non-speech intervention.

Speech recognizers deployed in continuously listening mode are constantly monitoring acoustic input and do not necessarily require non-speech activation. This contrasts with the push to talk model, in which speech recognition is only activated when the user pushes a button. Unfortunately, today's continuously listening speech recognizers are not reliable enough due to their insufficient accuracy, especially in the area of correct rejection. For example, such systems often respond erratically, even when no speech is present. They sometimes interpret background noise as speech, and they sometimes incorrectly assume that certain speech is addressed at the speech recognizer when in fact it is targeted elsewhere (context misunderstanding). These problems have traditionally been solved by the push to talk model: requesting the user to push a button immediately before or during talking or similar prompting paradigm.

Wake-Up-Word speech recognizers are often mistakenly compared to other speech recognition tasks such as keyword spotting or command and

control, but WUW speech recognizer is different from the previously mentioned tasks in several significant ways. The most important characteristic of a WUW-SR system is that it should guarantee virtually 100% correct rejection of non-WUW and same-words-uttered in non-alerting context while maintaining correct acceptance rate over 99%. This requirement sets apart WUW-SR from other speech recognition systems because no existing system can guarantee 100% reliability by any measure without significantly lowering correct recognition rate. It is guarantee that allows WUW-SR to be used in novel applications that previously have not been possible. Second, a WUW-SR system should be context dependent; that is, it should detect only words uttered in alerting context. Unlike keyword spotting, which tries to find a specific keyword in any context, the WUW should only be recognized when spoken in the specific context of grabbing the attention of the computer in real time. Thus, WUW speech recognition system can be viewed as a refined keyword spotting task, albeit significantly more sophisticated. Finally, WUW-SR should maintain high recognition rates in speaker-independent or speaker-dependent mode and in various acoustic environments (V. Z. Këpuska and T. B. Klein) [1].

### 2.4.1 Front End of Wake-Up-Word Speech Recognition

The concepts of WUW have been most recently expanded in (V. Z. Këpuska and T. B. Klein) [1].Currently, the system is implemented in C++ as well as JAVA, and provides three major components for achieving the goals of WUW for use in a real-time environment: front-end, VAD, and back-end. The front-end is responsible for extracting three different sets of features from an input audio signal. The Voice Activity Detector (VAD) segments the signal into speech and non-speech regions. Finally, the back-end performs recognition and scoring.

The WUW Speech Recognizer is a very complex digital system. As mentioned previously, the pre-processing can be done using FPGA, dedicated DSPs, or in software. The backtracking process requires large amounts of data storage, and indexing operations, for which software is better suited. It is the recognition part (including Viterbi decoding), and in particular the computation of observation probabilities, that requires a significant number crunching and for which no suitable device currently exists. Therefore, the front-end with built-in VAD has been the subject of our research resulting in the hardware designs presented in this dissertation. In a typical automatic speech recognition system, a signal processing front-end transforms the speech waveform from an input device, such as a microphone, to a

parametric representation.

This parametric representation, often referred to as “features”, is then used to drive the speech recognition decoder process. The generation of MFCC is one of the most widely used algorithms (L. Rabiner and B. H. Juang) [45], (S. Davis and P. Mermelstein) [46], (Ngoc-Vinh Vu, Jim Whittington, Hua Ye, and John C. Devlin) [47] to implement a front-end.

Although there are some attempts to implement MFCC in hardware (J.-C. Wang, J.-F. Wang, and Y.-S. Weng) [48], (W. Han, C.-F. Chan, C.-S. Choy, and K.-P. Pun) [49], most existed work on MFCC tends to focus on the improvement of the recognition performance. Since the core MFCC algorithm requires a substantial amount of calculation, implementing it in low-end hardware and keeping the design low-cost remains a challenge. In (Nedevschi, S., Patra, R., Brewer, E) [50], and (Melnikoff, S., Quigley, S.F., Rusell, M. J) [51], presented only hardware implementations of some specific part of algorithms for speech recognition or speaker identification that allow a significant acceleration of the processing time.

In our research, we designed and implemented the front-end part of the WUW-SR system in FPGA. We produced a new feature extraction system based on the three different features: MFCC, LPCC, and Enhanced MFCC. The proposed solution is optimized for modest resource usage, which makes

it suitable for a low-cost VLSI or FPGA device. This design not only has a relatively low resource usage, but also maintains a reasonably high level of performance.

# 3 Wake-Up-Word-SR System Architecture

The WUW Speech Recognizer is a complex system comprising of three major parts: Front- end, VAD, and Back-end. The Front-end is responsible for extracting three different sets of features from an input audio signal. The VAD (Voice Activity Detector) segments the signal into speech and non-speech regions. Finally, the back-end performs recognition and scoring. The diagram below shows the overall procedure.

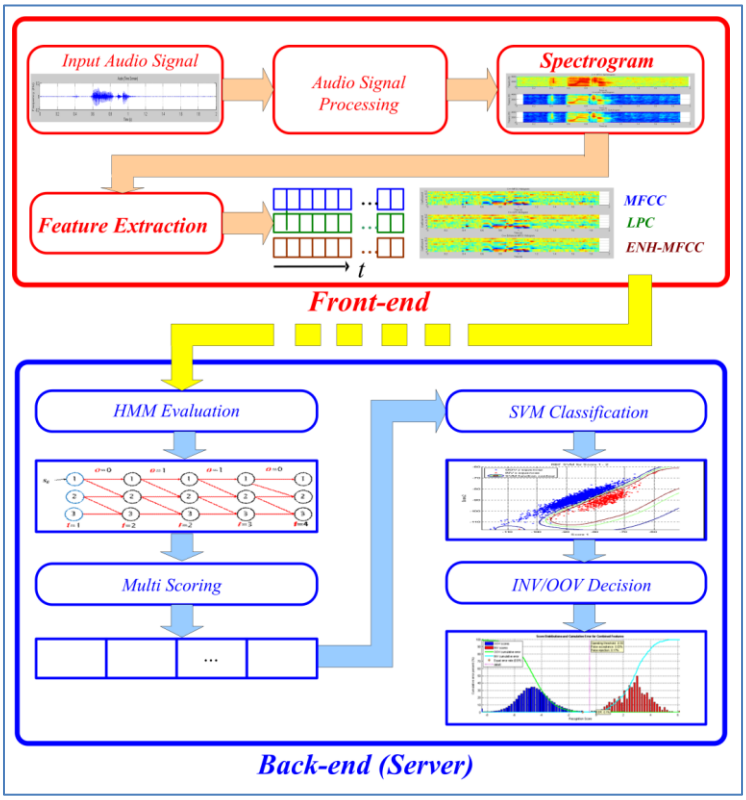


**Figure 3.1 – Wake-Up-Word-Speech Recognition Overall Architecture**

We present an experimental FPGA design and implementation of a novel architecture of a real time feature extraction processor that generates MFCC, LPC, and ENH_MFCC features simultaneously. In the WUW-SR system, the recognizer front-end is located at the terminal which is typically connected over a data network to remote back-end recognition (e.g., server). The three sets of feature extraction of speech (MFCC, LPC, and ENH-MFCC) are performed at the front-end. These extracted features are then compressed and transmitted to the server via a dedicated channel, where subsequently they are decoded.

The front-end system process takes an input pressure waveform (audio signal) and output a sequence of characteristic parameters MFCCs, LPCs, and ENH-MFCCs features. Whereas the back-end process, which is the recognition component, takes the characteristic sequence and outputs an index of the recognized command.

The signal processing module accepts raw audio samples and produces spectral representations of short time (t) signals. The feature-extraction module generates features from this spectral representation, which are decoded with the corresponding hidden Markov models (HMMs). The individual feature scores are classified using support vector machines (SVMs). INV, OOV: in-, out-of-vocabulary speech.

## 3.1 Front End of Wake-Up-Word Speech Recognition

As shown in the diagram below the front-end processor is responsible for extracting three different sets of features out of the input signal simultaneously. These sets of features are extracted: Mel-filtered Cepstral Coefficients (MFCC), LPC (Linear Predictive Coding) smoothed MFCCs, and Enhanced MFCCs.

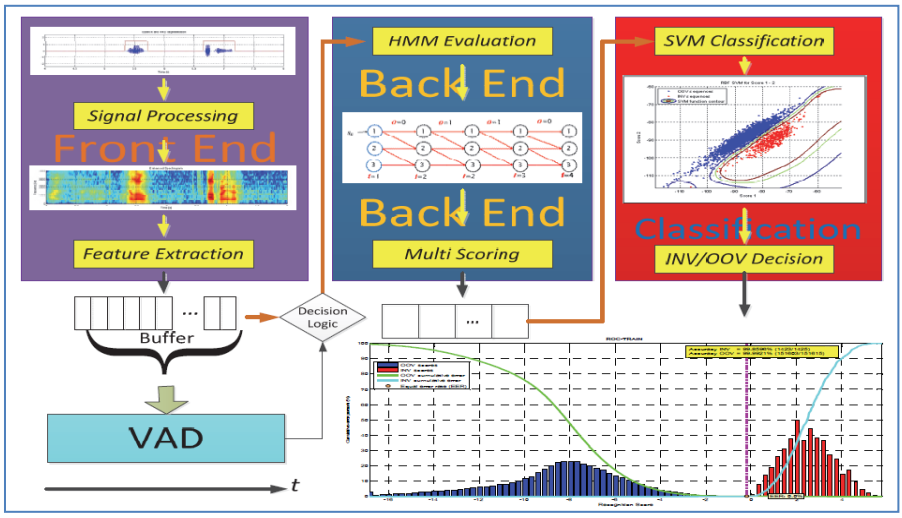


**Figure 3.2 – Front-end, Voice Activity Detector, and Back-end**

The following steps describe the feature extraction process:

1. ***Audio signal is converted from analog to digital***
2. ***DC is filtered out and pre-emphasis filter is applied to signal***
3. ***Audio signal is converted from integer to 32-bit floating point format***

4. ***Signal is windowed using 25ms Hamming window at a rate of 200 windows per second. At a sample rate of 8000 Hz this indicates a window size of 200 samples, shifted by 40 samples each frame***
5. ***Frame energy is computed and sent to VAD module***
6. ***LPC, FFT, and magnitude squared spectrum are computed***
7. ***Spectrum is sent to VAD module***
8. ***Mel-filtering, discrete cosine transform are computed***
9. ***MFCCs are sent to VAD module***
10. ***Frames are buffered for a delay of 20-30 frames in order for VAD to make a decision***
11. ***VAD decision is used for MFCC enhancement process, ENH-MFCC are computed***

## 3.2 Voice Activity Detector (VAD) of WUW-SR

The Voice Activity Detector is responsible for segmenting the signal into speech and non-speech segments as shown in Figure 3.3. For any given frame, VAD reports one of two possible states: VAD_ON or VAD_OFF. Word recognition in the Back-end stage begins when the VAD enters VAD_ON state, and ends when the VAD switches to VAD_OFF. VAD works in two phases. In the first phase, a classifier decides whether a single input frame is

speech-like or non-speech-like. In the second phase, the number of speech-frames and non-speech-frames over a period of time is analyzed and certain rules are applied to report the final decision of VAD_ON or VAD_OFF.

The following image shows “Onward” waveform as input audio data superimposed with its VAD segmentation, its MFCC spectrogram, LPC spectrogram, and enhanced-MFCC spectrogram.

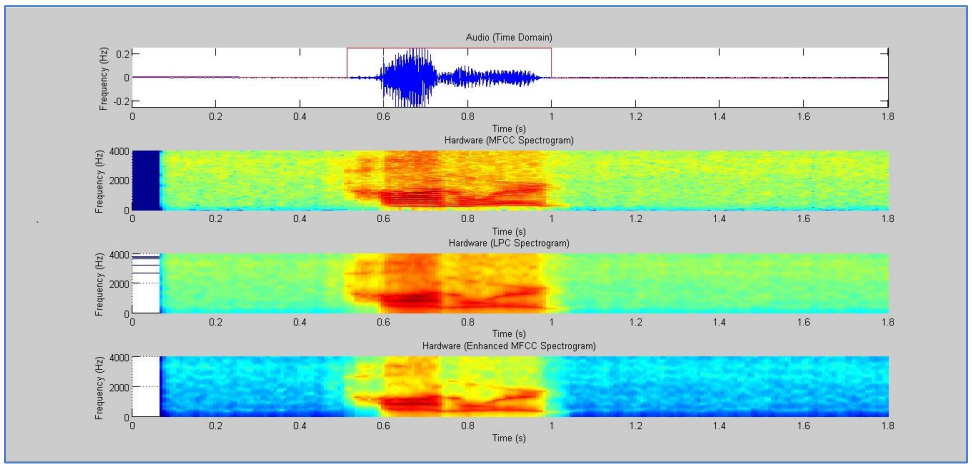


**Figure 3.3 – Speech signal with VAD segmentation, MFCC spectrogram, LPC spectrogram, and ENH-MFCC spectrogram**

### 3.2.1 First VAD Phase - Single Frame Speech/Non-Speech Classification

First, for every input frame VAD decides whether the frame is speech-like or non-speech-like. Several hardware models have been implemented and tested for solving this problem.

In the VAD design, the decision was made based on three features: log energy difference (Energy Features), LPC spectral difference (LPC

Spectrogram Features), and MFCC difference (MFCC Features). A threshold was determined empirically for each feature, and the frame was considered speech-like if at least two out of the three features were above the threshold. This was in effect a Decision Tree classifier, and the decision regions consisted of hypercube in the feature space.

In order to improve the VAD classification accuracy, the VAD implementation uses the three features: log energy difference, LPC spectral difference, and MFCC difference; however, classification is performed using a linear Support Vector Machine (SVM). There are several advantages over using this method. First, the classification boundary in the feature space is a hyper plane, which is more robust than the hypercube produced by the decision tree method. Second, the thresholds do not have to be picked manually but can be trained automatically (and optimally) using marked input files. Third, the sensitivity can be adjusted in smooth increments using a single parameter, the SVM decision threshold. Recall that the output of a SVM is a single scalar, $u = w \cdot x - b$. Usually the decision threshold is set at $u = 0$, but it can be adjusted in either direction depending on the requirements. Finally, the linear SVM kernel is extremely efficient, because classification of new data requires just a single dot product computation.

### 3.2.2 Second VAD Phase – Final Decision Logic

In the second phase, the VAD keeps track of the amount of frames marked as speech and non-speech and makes a final decision. There are four parameters:

***MIN_VAD_ON_COUNT, MIN_VAD_OFF_COUNT, LEAD_COUNT, and TRAIL_COUNT.***

The logic requires a number of consecutive frames to be marked as speech in order to set its state to VAD_ON, specified by MIN_VAD_ON_COUNT, and a number of consecutive frames to be marked as non-speech in order to set its state to VAD_OFF, specified by *MIN_VAD_OFF_COUNT*.

Because the classifier can make mistakes at the beginning and the end, the logic also includes a lead-in and a trail-out time. After the minimum number of consecutive speech frames has been observed VAD does not indicate VAD_ON for the first of those frames, but rather several frames earlier, a number specified by LEAD_COUNT. Similarly, when the minimum number of non-speech frames has been observed, VAD waits an additional number of frames before changing to VAD_OFF, specified by TRAIL_COUNT.

## 3.3 Back End of WUW-SR

Front-end of WUW-SR is responsible for generating three sets of features MFCC, LPC, and ENH-MFCC. These features are needed to be decoded with corresponding Hidden Markov Models (HMMs) in the back-end stage of the WUW-SR (e.g., server). The Back-end is responsible for classifying observation sequences as In Vocabulary (INV), i.e. the sequence is a Wake-Up-Word, and Out Of Vocabulary (OOV), i.e. the sequence is not a Wake-Up-Word.

The WUW-SR system uses a combination of Hidden Markov Models and Support Vector Machines for acoustic modeling, and as a result the back-end consists of an HMM recognizer and a SVM classifier. Prior to recognition, HMM and SVM models must be created and trained for the word or phrase which is to be the Wake-Up-Word.

When the VAD state changes from VAD_OFF to VAD_ON, the HMM recognizer resets and prepares for a new observation sequence. As long as the VAD state remains VAD_ON, feature vectors are continuously passed to the HMM recognizer, where they are scored using the triple scoring method. If using multiple feature streams, recognition is performed for each stream in parallel. When VAD state changes from VAD_ON to VAD_OFF, multiple scores are obtained from the HMM recognizer and are sent to the SVM

classifier. SVM produces a classification score which is compared against a threshold to make the final classification decision of INV or OOV.

# 4 Front End System Design

Automatic speech recognition systems are usually implemented on personal computers equipped with high-performance microprocessors. This is because of the computation complexity of applied algorithms, as well as their high confidential levels of security. General purpose processors contain floating-point units able to carry out millions of operations per second at frequencies in the GHz range, which allows for a resolution of the complex algorithms in just a few hundred of milliseconds. However, in the low-cost consumer market, such factors as price, power consumption and size determine the viability of a product.

Since the main drawback of microprocessors based systems are the cost, and the necessary space required to incorporate their external associated peripherals, the use of an FPGA (Field Programmable Gate Arrays) is a better suited way to implement systems that require a high computational capability at an affordable price. Additionally, the FPGA allows dividing and implementing algorithm as parallel parts, which allows running computation at lower operational circuit frequency and requires less power consumption. FPGA circuits can be programmed by the user and adapted to perform the particular task. The term "programming," in case of

FPGA architecture, means changing its internal structure. The programming can also be repeated multiple times. The mechanism that allows for FPGA programming, on the one hand, decreases the operating speed of the FPGA chip comparing to ASIC. On the other hand, it provides the opportunity to tune-up the system to the specific parameters of the implemented algorithm (Staworko, M.; Rawski, M) [52].

## 4.1 Features Extraction

The WUW-SR problem can be roughly divided into two issues: speech analysis (feature extraction) and classification. Feature extraction methods are responsible for reducing the resources required to describe speech samples accurately. In case of speech analysis, various digital signal processing (DSP) algorithms are used to detect desired features of input speech signal. The most common algorithms are LPCC, MFCC, LFCC and others. MFCC is recognized as the best known and most popular. However, the LFCC algorithm is often used in speaker identification applications, since it produces results of comparable quality (D. A. Reynolds) [53], (A.Kaczmarek, M.Staworko) [54], (Charbuillet, C., Gas, B., Chetouani, M., Zarader J.L) [55]. In our work, we aim to design and implement a novel WUW's front-end processor on FPGA to generate three different sets of

features:

- ***Mel-frequency Cepstral Coefficients (MFCC)***
- ***Linear predictive Coding Coefficients (LPC)***
- ***Enhanced Mel-frequency Cepstral Coefficients (ENH-MFCC)***

### 4.1.1 Mel-scale Frequency Cepstral Coefficients (MFCC)

The feature extraction involves identifying the *formants* in the speech, which represent the frequency locations of energy concentrations in the speaker's vocal tract. There are many different approaches used: Mel-scale Frequency Cepstral Coefficients (MFCC), Linear Predictive Coding (LPC), Linear Prediction Cepstral Coefficients (LPCC), Reflection Coefficients (RCs). Among these, MFCC has been found to be more robust in the presence of background noise compared to other algorithms (S. Davis and P. Mermelstein) [56]. Also, it offers the best trade-offs between performance and size (memory) requirements. The primary reason for effectiveness of MFCC is that, it models the non-linear auditory response of the human ear which resolves frequencies on a log scale (H. Combrinck and E. Botha) [57].

### 4.1.2 Autocorrelation Linear Predictive Coding (LPC)

The basic idea of LPC is to approximate the current speech sample as a linear combination of past samples as shown in the following equation:

$$x[n] = \sum_{k=1}^{p} a_k \, x[n-k] + e[n]$$

$x[n-k]$: Previous speech samples

$p$: Order of the model

$a_k$ : Prediction coefficient

$e[n]$: Prediction error

This module gets windowed data from the window module for representing the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model. We use this method to encode good quality speech and provide an estimate of speech parameters. The goal of this method is to calculate prediction coefficients $a_k$ for each frame. The order of LPC, which is the number of coefficients $p$, determines how closely the prediction coefficients can approximate the original spectrum. As the order increases, the accuracy of LPC also increases. This means the distortion will decrease. The main advantage of LPC is usually attributed to the all-pole characteristics of vowel spectra. Also, the ear is also more sensitive to spectral poles than zeros (M. R. Schroeder) [58].

In comparison to non-parametric spectral modeling techniques such as filter banks, LPC is more powerful in compressing the spectral information into few filter coefficients (K. K. Paliwal and W. B. Kleijn) [59].

### 4.1.3 Enhanced Mel-scale Frequency Cepstral Coefficients (ENH-MFCC)

The spectrum enhancement module is designed to generate ENH-MFCC features. We have implemented this module to perform an enhancement algorithm on the LPC spectrum signal. The ENH-MFCC features have a higher dynamic range than regular MFCC features, so these new features will help the back-end in improving the recognition quality and accuracy (V. Z. Këpuska and T. B. Klein) [1].

## 4.2 System Architecture

Front-end part of the novel Wake-Up-Word speech recognition system is designed and implemented on FPGA as efficient implementation of a complete system on a programmable chip (SOPC). Our design will get an impetus with the advent of high-density FPGAs integrated with high-capacity RAMs and the availability of implementation support for soft-core processors, such as the Nios II processor.

FPGAs enable the best of both worlds to be used gainfully for an application: the microcontroller and RISC processor is efficient for performing control and decision-making operations, while the FPGA efficiently performs digital signal processing (DSP) operations and other computation intensive tasks as will be explained later in this dissertation.

We produced efficient hardware front-end system with an FPGA acting as processor that is capable of generating three different sets of features (MFCC, LPCC, and ENH-MFCC) at much faster rate than software. Implementation of systems using an Altera-based system on a programmable chip enables time-critical functions to be implemented in hardware synthesized with Verilog HDL code.

As shown in the diagram below, front-end takes audio signal and processes it as a quantized digitized waveform through a sequence of very complex DSP modules to generate a sequence of 39-dimensions

- ***12-dimensions MFCC plus 1-dimension power***
- ***12-dimensions LPCC plus 1-dimension power***
- ***12-dimensions ENH-MFCC plus 1-dimension power***

  as the base feature for each frame, that can be used in the back-end model, each vector representing the information in a small time window of the signal. This feature is then extended to 39-dimensions for every feature type by augmenting first-order and second-order time derivatives, in order to capture the transition of the spectrum.

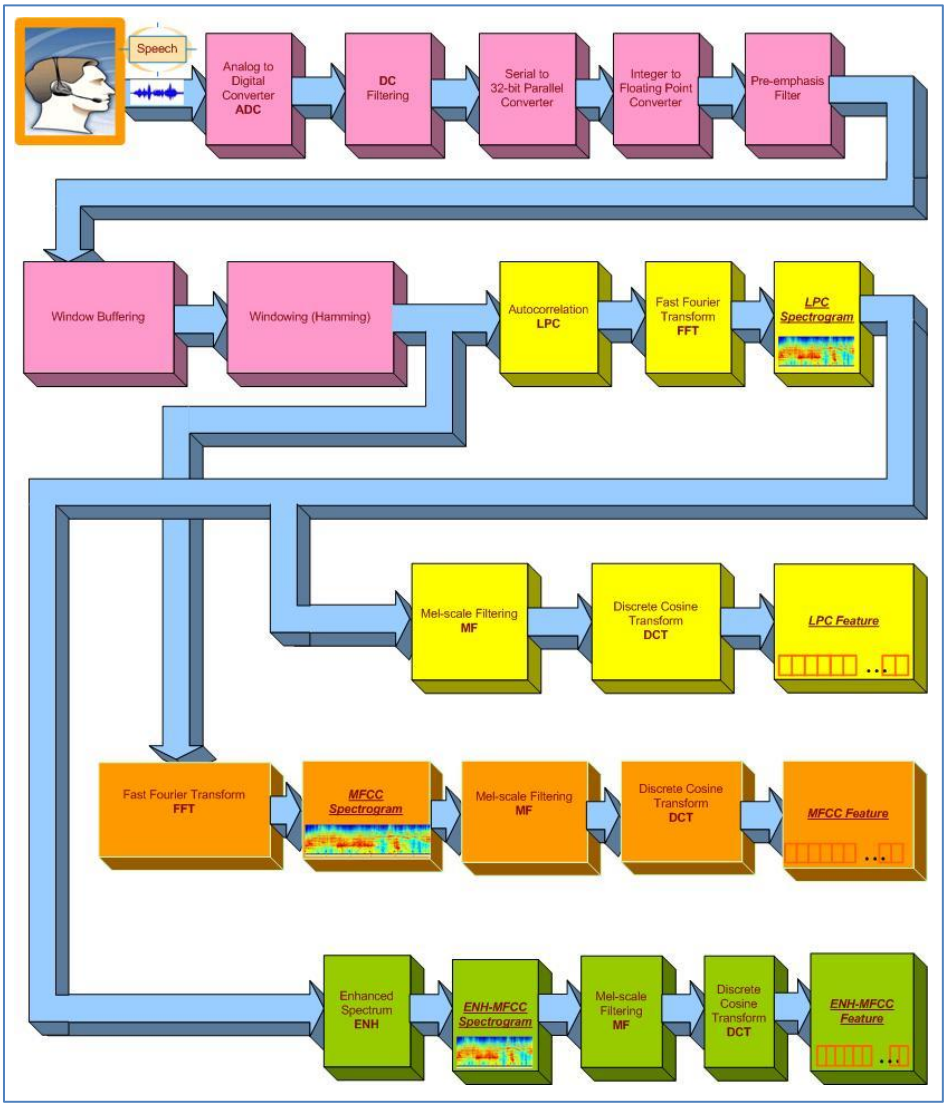


**Figure 4.1 – Front-end of WUW-SR Block Diagram**

## 4.3 Front End with Built-in VAD Architecture

The following diagram illustrates the architecture of the Front-end with Voice Activity Detector. Five blue-colored modules represent the Voice Activity Detector stage. The VAD is responsible for finding utterances

spoken in the correct context and segmenting them from the rest of the audio stream, then the system will identify whether or not the segmented utterance is a WUW. As shown in the diagram below, the design is divided into twenty seven-modules (five-stages).

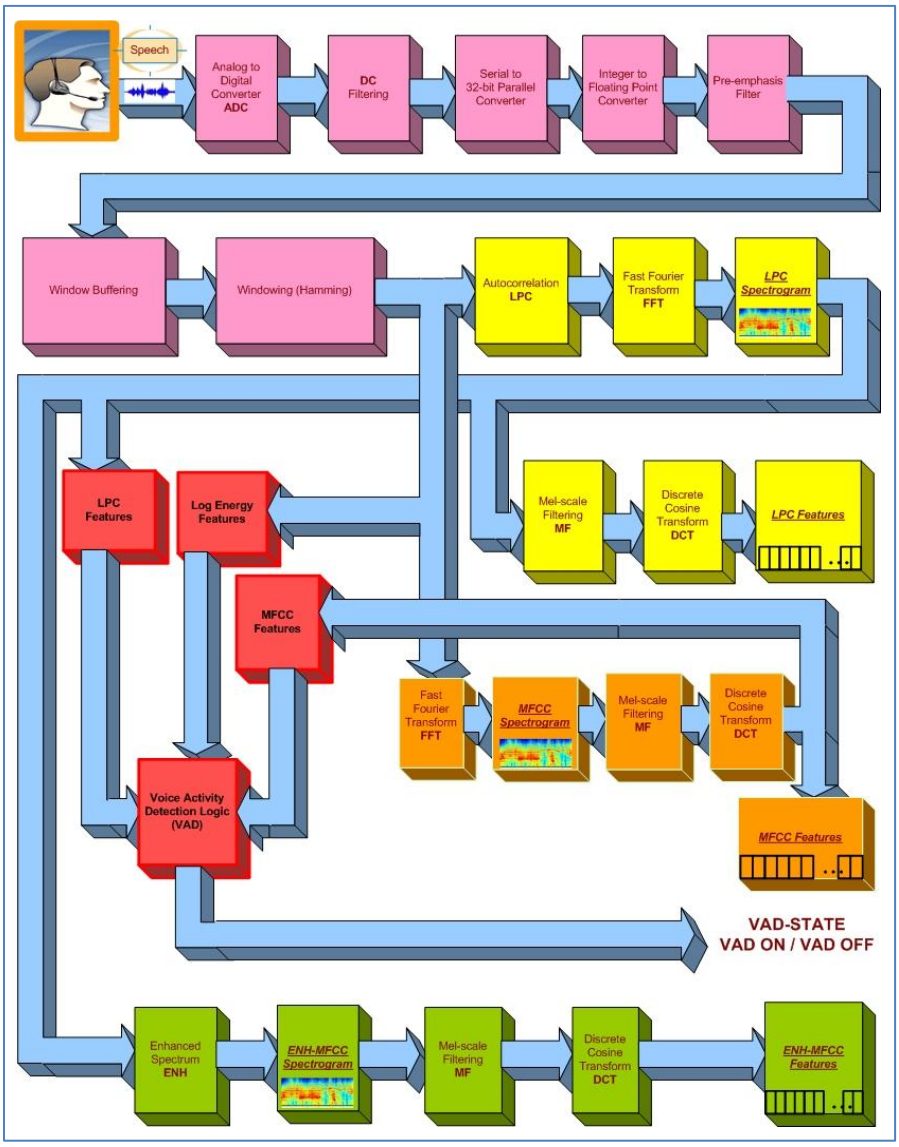


**Figure 4.2 – Front-end of WUW-SR with VAD Block Diagram**

**Pre-Processing stage:**

The first seven yellow-colored modules represent the pre-processing stage and are used as the basic modules to provide windowed speech signal to the other stages.

1. ***Analog to Digital Converter ADC***
2. ***DC Filtering***
3. ***Serial to 32-bit parallel converter***
4. ***Integer to floating-point converter***
5. ***Pre-emphasis filtering***
6. ***Window advance buffering***
7. ***Hamming window***

**Linear Predictive Coding Coefficients stage:**

Five brown-colored modules represent the Linear Predictive Coding Coefficients (LPC) stage and are used to generate 13-Linear Predictive Coding features.

1. ***Autocorrelation Linear Predictive Coding***
2. ***Fast Fourier Transform FFT***
3. ***LPC Spectrogram***
4. ***Mel-scale Filtering***
5. ***Discrete Cosine Transform DCT***

**Mel-Frequency Cepstral Coefficients stage:**

Four pink-colored modules represent the MFCC stage and are used to generate 13 MFCCs features.

1. ***Fast Fourier Transform FFT***
2. ***MFCC Spectrogram***
3. ***Mel-scale Filtering***
4. ***Discrete Cosine Transform DCT***

**Enhanced Mel-Frequency Cepstral Coefficients stage:**

Four green-colored modules represent the ENH-MFCC stage and are used to generate 13 ENH-MFCC features.

1. ***Enhanced Spectrum (ENH)***
2. ***Enhanced MFCC Spectrogram***
3. ***Mel-scale Filtering***
4. ***Discrete Cosine Transform DCT***

**Voice Activity Detector stage:**

Four red-colored modules represent the Voice Activity Detector (VAD) stage. The VAD is responsible for finding utterances spoken in the correct context and segmenting them from the rest of the audio stream, then the system will identify whether or not the segmented utterance is a WUW.

1. ***Spectrogram features***

2. ***Energy features***
3. ***MFCC features***
4. ***Voice activity detection logic VAD***

## 4.4 Design Function Description

The goal of front-end signal processing is to extract relevant feature parameters of speech, which are more suitable for the purpose of speech recognition than the input speech waveform itself in terms of information rate and redundancy reduction. Intensive efforts have been carried out to achieve a high performance front-end. Converting a speech waveform into a form suitable for processing by the decoder requires several stages as follows:

1. ***Filtration:*** The waveform is sent through a low pass filter, typically 4 kHz to 8 kHz. As is evidenced by the bandwidth of the telephone system being around 4 kHz; this is sufficient for comprehension and used a minimum bandwidth required for telephony transmittal.
2. ***Analog-to-Digital Conversion:*** The process of digitizing and quantizing an analog speech waveform begin with this stage. Recall that the first step in processing speech is to convert the analog representations (first air pressure, and then analog electric signals from a microphone), into a digital signal.

3. ***Sampling Rate:*** The resulting waveform is sampled. Sampling rate theory requires a sampling (Nyquist) rate of double the maximum frequency (so 8 to 16 kHz as appropriate). The sampling rate of 8 kHz was used in our front-end. (We used CODEC Chip to perform first, second, and third stages).
4. ***Serial to Parallel Converter:*** This model gets serial digital signal from CODEC and converts it to 32-bit.
5. ***Integer to floating-point converter:*** This module converts 32-bit, signed integer data to single-precision (32-bit) floating-point values. The input data is routed through the int_2_float Megafunction core named ALTFP_CONVERT.
6. ***Pre-emphasis:*** The digitalized speech signal s(n) is put through a low-order LPF to spectrally flatten the signal and to make it less susceptible to finite precision effects later in the signal processing. The filter is represented by:

$$y[n] = x[n] - \alpha x[n-1],$$

***Output = Input – (PRE_EMPH_FACTOR * Previous_input).***

The value of PRE_EMPH_FACTOR (α) where chosen as 0.975.

7. ***Window Buffering:*** A 32-bit, 256 deep dual-port RAM (DPRAM) stores 256 input samples. A state machine handles moving audio data

into the RAM, and pulling data out of the RAM (40 samples) to be multiplied by the Hamming coefficients, which are stored in a ROM memory.

8. ***Windowing:*** The hamming window function smoothes the input audio data with a Hamming curve prior to the FFT function. This stage slices the input signal into discrete time segments. This is done by using window N milliseconds, typically 25 ms wide (200 samples). A Hamming window size of 25 ms which consists of 200 samples at 8 KHz sampling frequency and 5 ms frame shift (40 samples) is picked for our front-end windowing.
9. ***Fast Fourier Transform:*** In order to map the sound data from the time domain to the frequency domain, the Altera IP Megafunction FFT module is used. The module is configured so as to produce a 256-point FFT. This function is capable of taking a streaming data input in natural order, and it can also output the transformed data in natural order, with maximum latency of 256 clock cycles once all the data (256 data samples) has been received.
10. ***Spectrogram:*** This module takes the complex data generated by the FFT and performs the function*:*

$$20 * log10\ (fft\_real2 + fft\_imag2)$$

We designed spectrogram to show how the spectral density of a signal varies with time. We used spectrogram module to identify phonetic sounds. Digitally sampled data, in the time domain, are broken up into chunks, which usually overlap, and Fourier transformed to calculate the magnitude of the frequency spectrum for each chunk. Each chunk then corresponds to a vertical line in the image; a measurement of magnitude versus frequency for a specific moment in time. The spectrums or time plots are then "laid side by side" to form the image surface.

11. ***Mel-scale Filtering:*** While the resulting spectrum of the FFT contains information in each frequency in linear scale, human hearing is less sensitive at frequencies above 1000 Hz. This concept also has a direct effect on performance of ASR systems; therefore, the spectrum is warped using a logarithmic Mel scale. In order to create this effect on the FFT spectrum, a bank of filters is constructed with filters distributed equally below 1000 Hz and spaced logarithmically above 1000 Hz.
12. ***Discrete Cosine Transform:*** DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length,

operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even). A DCT computes a sequence of data points in terms of summation of cosine functions oscillating at various frequencies. The idea of performing DCT on Mel Scale is motivated by extraction of the speech frequency domain characteristics. DCT module reduces the speech signal's redundant information, and reaches the aim of regulating the speech signal into feature coefficients with minimal dimensions.

13. ***Enhanced Spectrum:*** This module gets the spectrogram from LPC and statistical information from the VAD and generates a new enhanced and smoothed Mel-frequency Cepstral Coefficients (ENH-MFCC).
14. ***Feature Representation:*** Speech recognition is a computationally demanding task. Particularly the stage of feature extraction, which is responsible for reducing the resources required to describe speech samples accurately and requires algorithms of large complexity. Normally, speech signal is converted into a parameterized sequence of feature vectors by front-end processing to emphasize the characteristics of spoken words and suppress other irrelevant information. Typical features used in continuous speech recognition

include Linear predictive Coding Coefficients (LPCC), Mel-frequency Cepstral Coefficients (MFCC) and Perceptual Linear Predictive Coefficients (PLP). Some of them are motivated by the nature of human hearing. For the experiments described in this dissertation, the outputs of our front-end are sequences of vectors composed of features:

- ***Mel-frequency Cepstral Coefficients (MFCC)***
- ***Linear Predictive Coding Coefficients (LPC)***
- ***Enhanced Mel-frequency Cepstral Coefficients (ENH-MFCC)***

# 5 Front End Hardware Implementation

## 5.1 System Design Environment

### 5.1.1 Hardware Design Tools

The FPGA development boards used in our design are ALTERA DE2 Development Board with (Cyclone II FPGA) and DSP Development Kit with (Cyclone III FPGA). The earlier designs are implemented on the ALTERA DE2 Development Board and as the design gets larger and more complex we used (DSP Development Kit with Cyclone III FPGA). In both cases the designs are written in Verilog HDL.

The Cyclone® III DSP development board provides a hardware platform for developing and prototyping low-power, high-volume, and feature-rich designs and to demonstrate the Cyclone III device's on-chip memory, embedded multipliers, and the Nios® II embedded processor.

With up to 4 M bits of embedded memory and 288 embedded 18-bit × 18-bit multipliers, the Cyclone III device supplies internal memory while also provides external support for high-speed, low-latency memory access via dual-channel DDR SDRAM and low-power SRAM.

Cyclone III devices are designed to provide low static and dynamic power consumption. Additionally, with the support of the Quartus® II

software's PowerPlay technology, designs are automatically optimized for power consumption. Therefore, the Cyclone III development board provides a power-optimized, integrated solution for memory-intensive, high-volume applications. The Cyclone III Development Kit features the EP3C120F780 device (U20) in a 780-pin BGA package. The board contains two HSMC (High-Speed Mezzanine Cards) interfaces called Port A and Port B. These HSMC interfaces support both single-ended and differential signaling. The HSMC interface also allows for JTAG, SMBus, clock outputs and inputs, as well as power for compatible HSMC cards. The HSMC is an Altera-developed specification, which allows users to expand the functionality of the development board through the addition of daughter cards (HSMC cards), (Altera, Inc) [17].

Cyclone® III device family offers a unique combination of high functionality, low power and low cost. Based on Taiwan Semiconductor Manufacturing Company (TSMC) low-power (LP) process technology, silicon optimizations and software features to minimize power consumption, Cyclone III device family provides the ideal solution for high-volume, low-power, and cost-sensitive applications. To address the unique design needs, Cyclone III device family offers the following two variants:

- ***Cyclone III—lowest power, high functionality with the lowest cost***
- ***Cyclone III LS—lowest power FPGAs with security***

With densities ranging from about 5,000 to 200,000 logic elements (119,088 LEs) and 0.5 Megabits to 8 Mb of memory (3,981,312 Byte) for less than ¼ watt of static power consumption, Cyclone III device family makes it easier for you to meet your power budget. Cyclone III LS devices are the first to implement a suite of security features at the silicon, software, and intellectual property (IP) level on a low-power and high-functionality FPGA platform. This suite of security features protects the IP from tampering, reverse engineering and cloning. In addition, Cyclone III LS devices support design separation which enables you to introduce redundancy in a single chip to reduce size, weight, and power of your application (Altera, Inc) [17].The TLV320AIC23 is a high-performance stereo audio codec with highly integrated analog functionality. The analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) within the TLV320AIC23 use multi bit sigma-delta technology with integrated oversampling digital interpolation filters. Data-transfer word lengths of 16, 20, 24, and 32 bits, with sample rates from 8 kHz to 96 kHz, are supported. The ADC sigma-delta modulator features third-order multibit architecture with up to 90-dBA signal-to-noise ratio (SNR) at audio sampling rates up to 96 kHz, enabling high-fidelity

audio recording in a compact, power-saving design. The DAC sigma-delta modulator features a second-order multibit architecture with up to 100-dBA SNR at audio sampling rates up to 96 kHz, enabling high-quality digital audio-playback capability, while consuming less than 23 mw during playback only (Texas Instruments) [60].

### 5.1.2 Software Design Tools

#### *5.1.2.1 Quartus II 64-bits*

Quartus II is a software tool produced by Altera for analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. The Quartus II software is the leading design software for performance and productivity. It is the only complete design solution for CPLDs, FPGAs, and ASICs in the industry; it is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes an integrated development environment to accelerate system-level design and seamless integration with leading third-party software tools and flows (Altera, Design Software) [61].

#### *5.1.2.2 ModelSim Altera*

Mentor Graphics ModelSim® HDL Simulator is a source-level verification tool, allowing you to verify HDL code line by line. You can perform simulation at all levels: behavioral (pre-synthesis), structural (post-synthesis), and back-annotated, dynamic simulation.

Coupled with the most popular HDL debugging capabilities in the industry, ModelSim is known for delivering high performance, ease of use, and outstanding product support.

Graphical user interface enables you to quickly identify and debug problems, aided by dynamically updated windows. Once a problem is found, you can edit, recompile, and re-simulate without leaving the simulator. ModelSim fully supports current VHDL and Verilog HDL language standards. You can simulate behavioral, RTL, and gate-level code separately or simultaneously. ModelSim supports all Altera FPGA libraries, ensuring accurate timing simulations (Mentor Graphic, ModelSim Software) [62].

## 5.2 Front End System Architecture

The Front-end design is broken down into four stages to achieve a high performance design:

- ***Mel-scale Frequency Cepstral Coefficient (MFCC)***

- ***Autocorrelation Linear Predictive Coding (LPC)***
- ***Enhanced Mel-scale Frequency Cepstral Coefficient (ENH-MFCC)***
- ***Voice Activity Detector (VAD)***

Converting a speech waveform into three different sets of spectrograms and features:

- ***Mel-scale Frequency Cepstral Coefficient (MFCC)***
- ***Autocorrelation Linear Predictive Coding (LPC)***
- ***Enhanced Mel-scale Frequency Cepstral Coefficient (ENH-MFCC)***

The following diagram illustrates the architecture of the Front-end and VAD. Blue colored models represent VAD models.

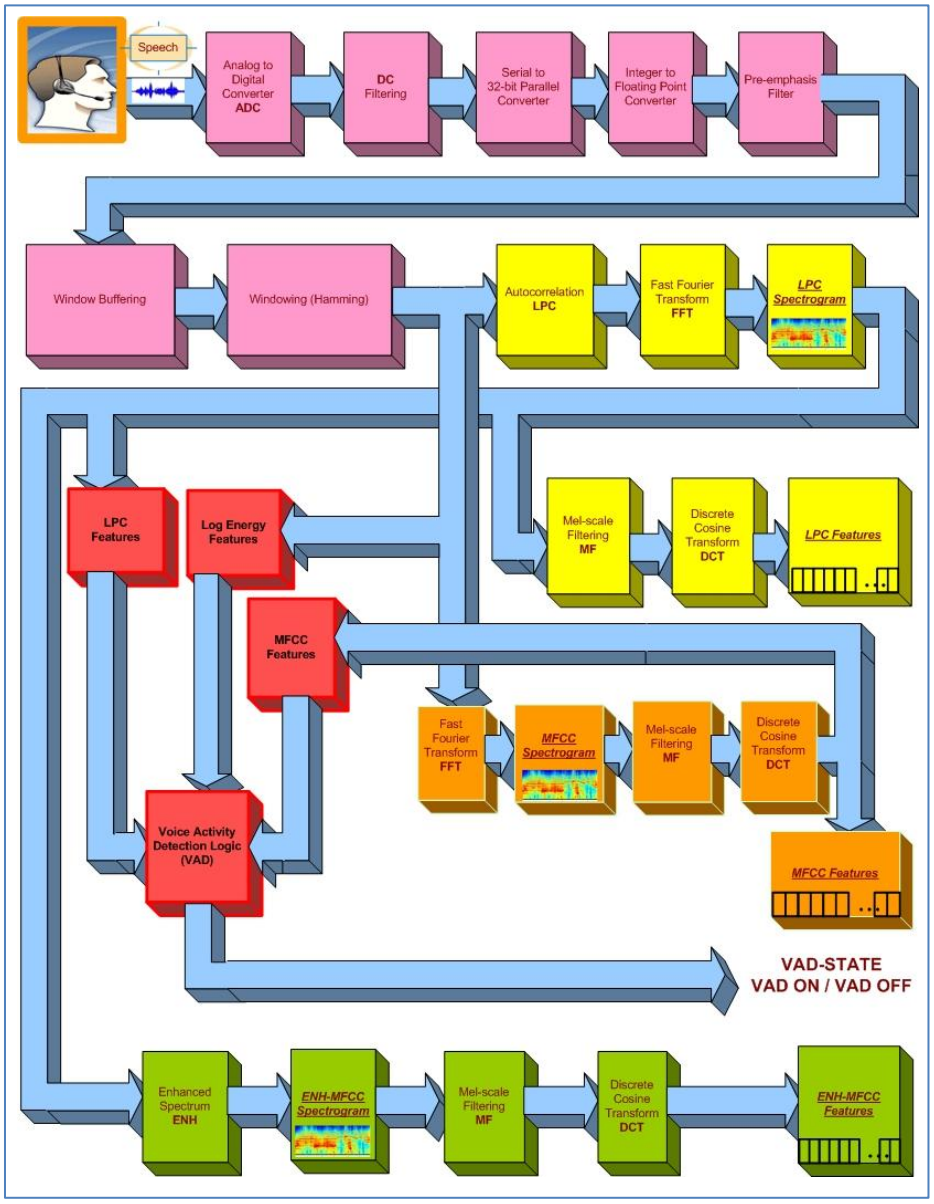


**Figure 5.1 – Front-End of WUW-SR Architecture Block Diagram**

### 5.2.1 MFCC Front End Subsystem Implementation

As shown in the diagram below, we begin with the process of digitizing and quantizing an analog speech waveform. Recall that the first

step in processing speech is to convert the analog representations (first air pressure, and then analog electric signals in a microphone), into a digital signal. This process of analog-to-digital conversion has two steps:

- ***Sampling***
- ***Quantization***

A signal is sampled by measuring its amplitude at a particular time; the sampling rate is the number of samples taken per second. In order to accurately measure a wave, it is necessary to have at least two samples in each cycle: one measuring the positive part of the wave and one measuring the negative part. More than two samples per cycle increases the amplitude accuracy, but less than two samples will cause the frequency of the wave to be completely missed. Thus the maximum frequency wave that can be measured is one whose frequency is half the sample rate (since every cycle needs two samples). This maximum frequency for a given sampling rate is called the Nyquist frequency.

In any signal processing and digital audio, quantization is the process of approximating a continuous range of values (or a very large set of possible discrete values) by a relatively small set of discrete symbols or integer values (http://en.wikipedia.org/wiki/Quantization_(sound_processing) #Audio_ quantization) [63].

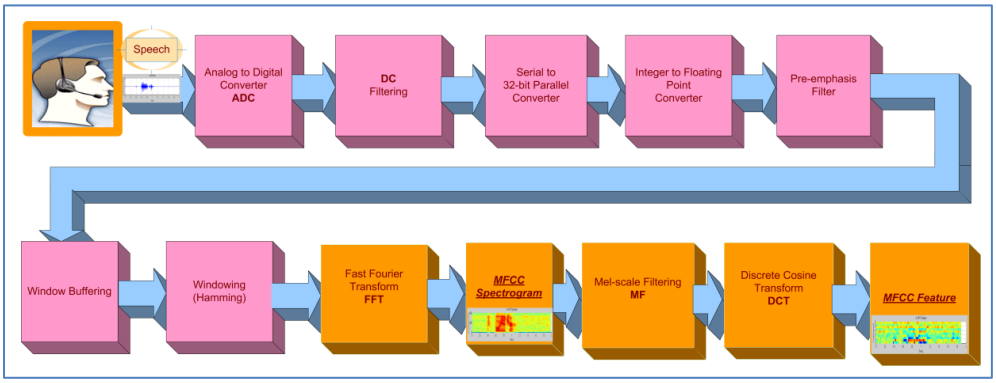


**Figure 5.2 – MFCC Front-end Subsystem**

The first building block includes two modules (ADC module & DC Filtering module). The speech acquisition contains a microphone and a CODEC from which digitized speech data are generated. To perform analog to digital conversion and DC filtering we designed controller module to control the CODEC and acquire the digital data from it, using the specification given by the Philips for I2C protocol & DSP operation mode of CODEC on the Cyclone® III FPGA [EP3C120F780]. A controller was designed using Verilog HDL to perform two operations: I2C protocol operation to drive the Audio CODEC [TLV320AIC23], and sound fetching from Audio CODEC [TLV320AIC23] to FPGA in DSP mode.

As shown in the figure below, the (tlv320_codec_spi_ctrl) module has been created in the design: the I2C bus controller, virtual sound fetcher, and the clock module. The FPGA communicates with the CODEC via the I2C (Inter-Integrated Circuit) protocol using two pins: 'SDIN' (the data line), and

'SCLK' (the bus clock). I2C bus controller modifies internal settings of CODEC, de-mute the microphone input, boost the microphone volume, and change the default sound path (giving the microphone priority over other inputs). After the CODEC digitalizes the input it puts the digital data on a digital audio interface, to fetch the data on SDATA_IN of codec from digital audio interface. DSP operation mode is used in the design. SDATA_IN is the formatted digital audio data stream with left and right channels multiplexed together. LRCOUT (alignment clock) and BCLK (synchronization clock) is used to fetch the data on SDATA_IN. This data can be used for any sound application. The clock module is designed to generate different clock requirements for the controller. CODEC internal registers sittings:

- Generate reset, delay after command.
- ***Sample Rate Control:*** USB Mode, SR = 8 KHz, no clock dividers.
- ***Digital Audio Interface Format:*** Master DSP format, 32-bit word length.
- ***Analog Audio Path Control:*** Side stone disabled, DAC enabled, Bypass disabled, INSEL = line-in, MIC muted, no Mic boost.
- ***Analog Audio Path Control:*** Sidestone disabled, DAC enabled, Bypass disabled, INSEL = MIC, MIC not muted, no Mic boost.

- ***Power Down Control:*** Turn on everything.
- ***Digital Audio Path Control:*** Disable DAC soft mute, enable ADC HPF (DC Filtering).
- ***Digital Interface Activation:*** Activate digital audio interface.
- ***Left Line In Volume Control:*** LRS=1, Mute off, volume = odB.
- ***Right Line In Volume Control:*** LRS=1, Mute off, volume = odB.

#### *5.2.1.1 CODEC Audio Data Interface*

This is the interface to the audio CODEC which receives Analog-to-Digital Converter (ADC) data from the CODEC and sends Digital-to-Analog Converter (DAC) data to the CODEC. The interface operates in DSP mode only and assumes that the CODEC is the transfer master (the CODEC generates the BCLK and L/R clocks.) In DSP mode, the CODEC generates a single clock pulse on the LRCLK_in signal and data is shifted in/out on the subsequent clocks. The CODEC shifts ADC data out on the falling edge of BCLK (CODEC_BCLK_in) as MSB first, so the ADC data (CODEC_SDATA_in) is registered on the rising edge of BCLK in the interface logic. Likewise, the CODEC registers DAC data (CODEC_SDATA_out) on the rising edge of BCLK, so the interface logic shifts DAC data out on the falling edge.

The interface is designed to send/receive 16, 20, 24 or 32-bit wide data to/from the CODEC. The 2-bit input (WORDLENGTH_SEL_in) determines the number of bits to shift per channel: 00 = 16-bit, 01 = 20-bit, 10 = 24-bit, 11 = 32-bit. The output ADC_DATA_out contains the parallel left and right data from the CODEC. Depending on the selected word length the output data is formatted as:

| *Word length* | *Left Channel Data* | *Right Channel Data* |
|---|---|---|
| 16 | ADC_DATA_out[31:16] | ADC_DATA_out[15:0] |
| 20 | ADC_DATA_out[39:20] | ADC_DATA_out[19:0] |
| 24 | ADC_DATA_out[47:24] | ADC_DATA_out[23:0] |
| 32 | ADC_DATA_out[63:32] | ADC_DATA_out[31:0] |

A single BCLK data valid pulse (ADC_DV_out) is generated when all bits are registered. The DAC data interface is designed to read data out of a FIFO, so the signal DAC_FIFO_RDEN_out generates a single BCLK read-enable pulse (if DAC_FIFO_EMPTY_in is not active) at the end of each transfer to retrieve the next DAC words. Since the DAC data is sent MSB first, the input data must be formatted accordingly for the selected word length:

| *Word length* | *Left Channel Data* | *Right Channel Data* |
|---|---|---|
| 16 | DAC_DATA_in[63:48] | DAC_DATA_in[47:32] |
| 20 | DAC_DATA_in[63:44] | DAC_DATA_in[43:24] |
| 24 | DAC_DATA_in[63:40] | DAC_DATA_in[39:16] |
| 32 | DAC_DATA_in[63:32] | DAC_DATA_in[31:0] |

#### *5.2.1.2 Serial-to-Parallel & Integer-to-Floating Point Converter*

We used floating-point converter (ALTFP_CONVERT) Megafunction core to design this module. This operation converts integer bits to the IEEE-754 floating-point representation. Conversions of signed integers to floating-point numbers in single precision are used in this module. All floating-point formats are implemented as shown in the figure below.

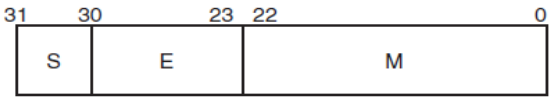


**Figure 5.3 – IEEE-754 Single-precision Floating-point Representation**

- ***S: Represents a sign bit***
- ***E: Represents an exponent field***
- ***M: Represents the mantissa (part of a logarithm or fraction) field***

For a single-precision floating-point number, the most significant bit (MSB) is a sign bit, followed by eight intermediate bits to represent an exponent and 23 least significant bits (LSBs) to represent the mantissa. As a

result, the total width for a single-precision floating-point number is 32 bits. The bias for the representation is 127. The advantage of using floating-point numbers is that they can represent a much larger range of values. In a fixed-point number representation, the radix point is always at the same location. Although the fixed radix point simplifies numeric operations and conserves memory, it limits the magnitude and precision of the number representation. In situations that require a large range of numbers or high resolution, a reloadable radix point is desirable. In the floating-point format, very large or very small numbers can be represented.

#### *5.2.1.3 Pre-emphasis Filter*

Prior to the core feature-extraction component, there is one more stage of pre-processing that is necessary to be carried out. This pre-emphasis phase is basically a high-pass filter that increases the relative energy of the high frequency spectrum. The characteristics of the vocal tract define the properties of speech. Although possessing relevant information, how frequency formants contain high concentrations of energy relative to high frequency formants. As shown in the figure below, the digitized speech signal goes through a pre-emphasis process, which performs spectral flattening with a first-order FIR filter described by:

$$y[n] = x[n] - \alpha x[n-1]$$

$$Output = Input - (PRE\_EMPH\_FACTOR * Previous\ input)$$

Where n is the sample index and $\alpha = 0.975$ the filter coefficient used.

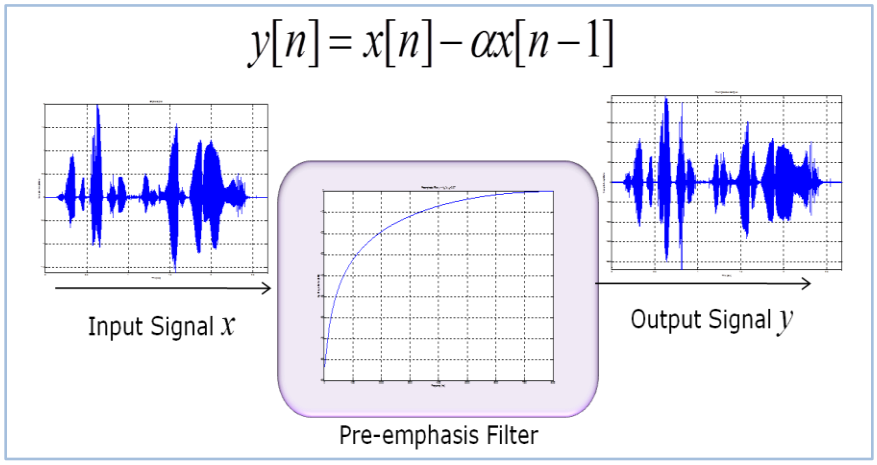


**Figure 5.4 – Pre-emphasis Function**

This module is used to amplify energy in the high-frequencies of the input speech signal. This allows information in these regions to be more recognizable during HMMs training and recognition.

### *5.2.1.4 Hamming Window & Advance Buffering*

For advance buffering we used A 32-bit, 256 deep dual-ports RAM (DPRAM) to store 256 input samples. A state machine handles moving audio data into the RAM, and pulling data out of the RAM (40 samples) to be multiplied by the Hamming coefficients, which are stored in a ROM memory.

The Hamming window function smoothes the input audio data with a Hamming curve prior to the FFT function. This stage slices the input signal into discrete time segments. This is done by using window typically 25 ms wide (200 samples). A Hamming window size of 25 ms which consists of 200 samples at 8 KHz sampling frequency and 5 ms frame shift (40 samples) is picked for our front-end windowing. The diagram and equation below show hamming window function.

$$w(n) = 0.54 - 0.46\cos\left(2\pi\frac{n}{N}\right), \quad 0 \le n \le N$$

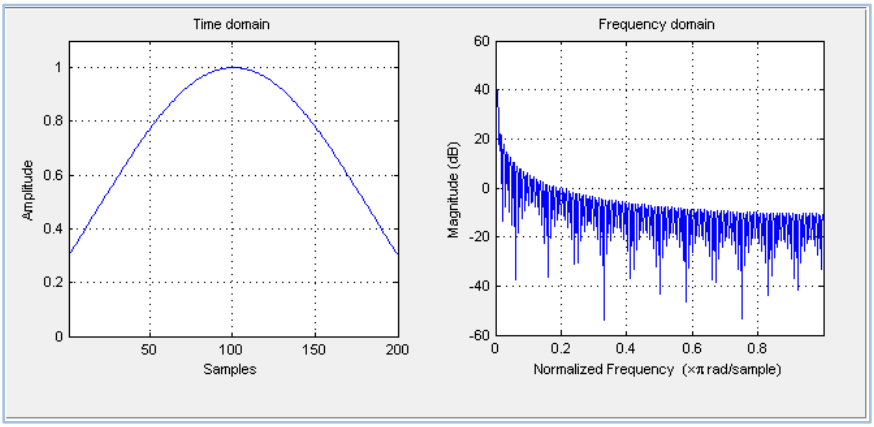


**Figure 5.5 - Hamming Window**

A 32-bit, 256 deep dual-port RAM (DPRAM) stores 200 input samples. A state machine handles moving audio data into the RAM, and pulling data out of the RAM to be multiplied by the Hamming coefficients, which are stored in a ROM memory. The ROM initialization file was created using a

MATLAB script, which uses the MATLAB function "hamming" to generate the 200 data points and convert them to hexadecimal, 32-bit, single precision floating-point values. The script writes the initialization file for the Megafunction IP and also a file used for the test bench.

The state machine first stores 40 incoming audio samples in the RAM and decrementing to a specific location (this mimics a FIFO). Once the 40 samples have been stored, the state machine pulls data out of the RAM (oldest sample first). This data is multiplied by the corresponding Hamming coefficient from the ROM. The result of the multiplication is stored in a FIFO for use by the FFT. The state machine accounts for the latency of the multiplication core. Simultaneously, the data in the RAM is shifted up in the memory by 40 locations (the upper 40 words are discarded), leaving the bottom 40 words open for the next set of audio samples.

To summarize, each FFT calculation is a weighted average of present and past audio samples. The figure below shows speech signal before and after applying hamming window.

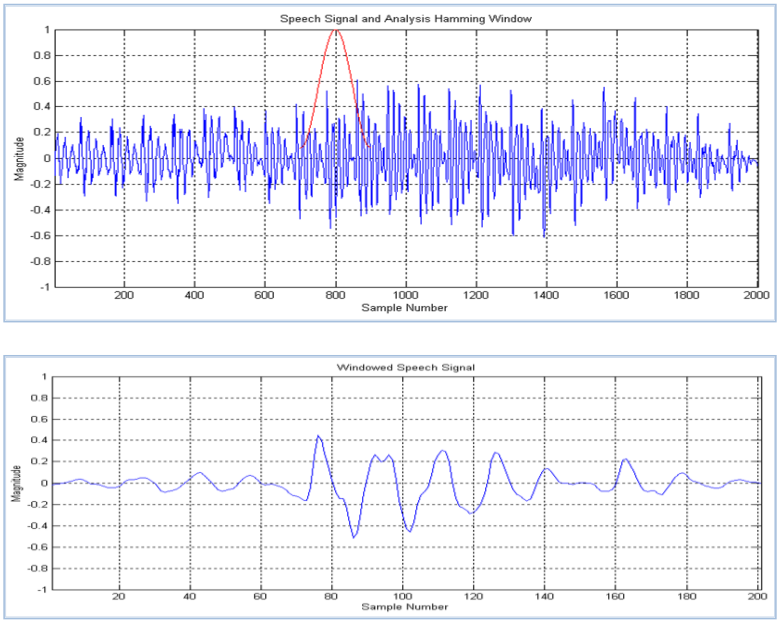


**Figure 5.6 - Speech Signal before and after applying Hamming Window**

### *5.2.1.5 Fast Fourier Transform*

In order to map the sound data from the time domain to the frequency domain, the Altera IP Megafunction FFT module is used. The module is configured so as to produce a 256-point FFT. This function is capable of taking a streaming data input in natural order, and it can also output the transformed data in natural order, with maximum latency of 256 clock cycles once all the data (256 data samples) has been received.

The FFT interface module instantiates a single-precision FFT of length 256. The architecture of the FFT is variable-streaming, natural-order in and out and supports an inverse FFT function as well. A state machine reads data

out of the hamming window FIFO when it is full and streams this data into the FFT. The streaming output of the FFT writes the complex data (real & imaginary) into separate 256 x 32-bit FIFOs for extraction by the spectrogram function.

As shown in the figure below, FFT used to transform the speech signal into frequency domain, where the most important speech/speaker information resides. The windowed time domain samples are converted into frequency domain by discrete fourier transform. The frequency-domain samples generally have complex values. Only the real magnitudes were used in our design.

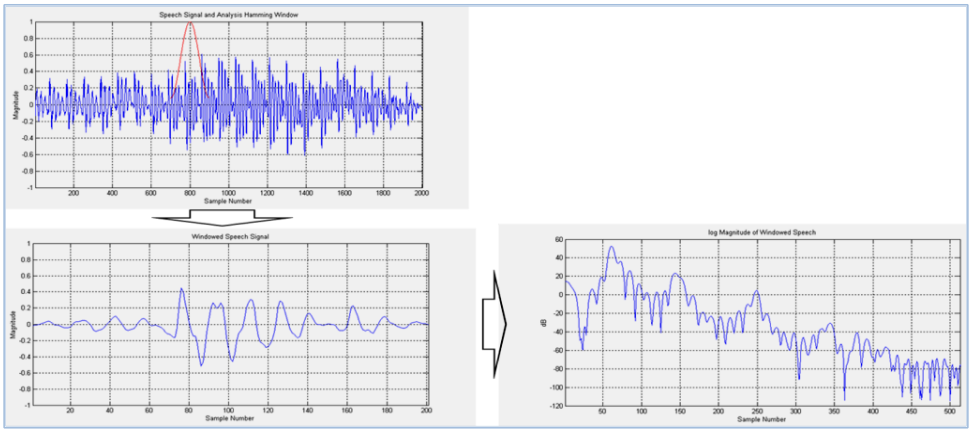


**Figure 5.7 - Windowed Speech to Fourier Transform**

### *5.2.1.6 MFCC Spectrogram*

A spectrogram is a time-varying spectral representation (forming an image) that shows how the spectral density of a signal varies with time. Also

known as spectral waterfalls, sonograms, voiceprints (or voicegrams), and spectrograms are used to identify phonetic sounds, to analyze the cries of animals; they were also used in many other fields including music, sonar/radar, speech processing, seismology, etc. The instrument that generates a spectrogram is called a spectrograph. This module takes the complex data generated by the FFT and performs the function below:

$$20 * \log_{10}(fft\_real^2 + fft\_imag^2)$$

We designed spectrogram to show how the spectral density of a signal varies with time. We used spectrogram module to identify phonetic sounds. Digitally sampled data, in the time domain, are broken up into chunks, which usually overlap, and Fourier transformed to calculate the magnitude of the frequency spectrum for each chunk. Each chunk then corresponds to a vertical line in the image; a measurement of magnitude versus frequency for a specific moment in time. The spectrums or time plots are then "laid side by side" to form the image surface.

The figures below shows waveform (speech signal) for words “Onward” and “Voyager” with 8 KHz sampling rate and its MFCC spectrograms representation generated by the Front End module.

This module reads the real and imaginary FFT data from the FIFOs whenever data is available. The real data is multiplied by itself through a

floating-point multiply Megafunction core and the imaginary data is multiplied by itself through another core. The results of these multiplies are routed to the inputs of the floating-point adder. The result of the adder is then routed to the input of the floating-point log Megafunction. Altera's log Megafunction actually calculates the natural log of the input.

To scale the final result to log10, the output of the log function is multiplied by 20 / ln (10) - this is a constant value of 0x410AF967 (8.685889). A counter is used to compensate for the latency of the Megafunctions (generate the FFT FIFO read-enables and the spectrogram output write-enable).

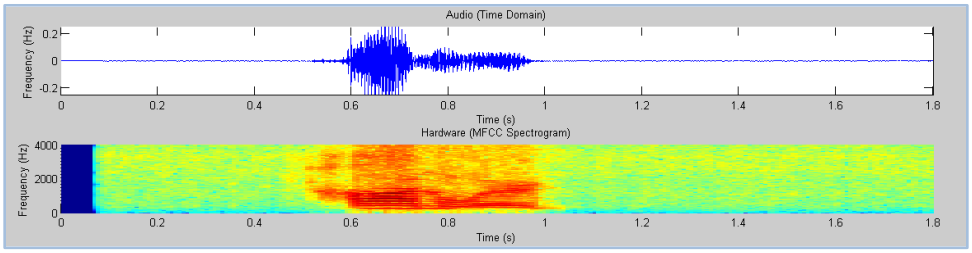


**Figure 5.8 - MFCC Hardware Front-end Spectrogram for the word "Onward"**

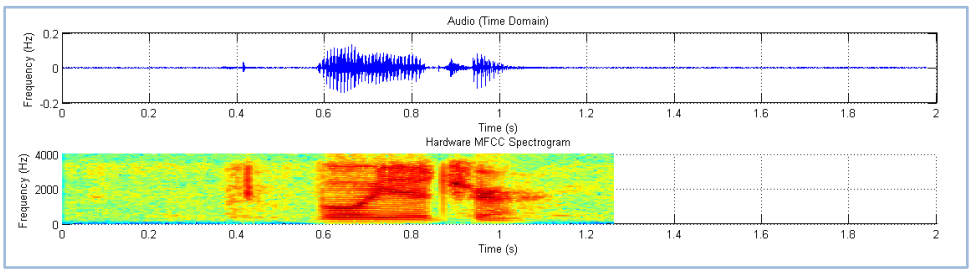


**Figure 5.9 - MFCC Hardware Front-end Spectrogram for the word "Voyager"**

### *5.2.1.7 Mel-scale Filtering*

While the resulting spectrum of the FFT contains information in each frequency in linear scale, human hearing is less sensitive at frequencies above 1000 Hz. This concept also has a direct effect on performance of ASR systems; therefore, the spectrum is warped using a logarithmic Mel-scale as shown in the figure. . In order to create this effect on the FFT spectrum, a bank of filters is constructed with filters distributed equally below 1000 Hz and spaced logarithmically above 1000 Hz. Mel-scale filter was computed using the function below:

$$mel\ (f) = 1127\ ln\ (1+f\ /\ 700)$$

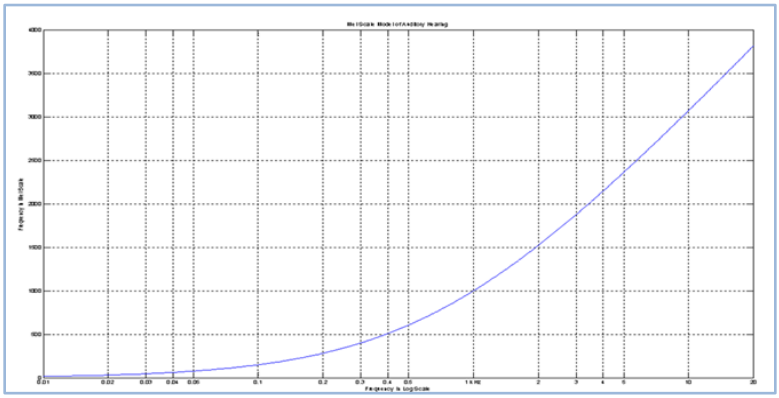

**Figure 5.10 - Mel-scale Function**

The figure below shows Mel-scale filter bank using triangular filters. The output of filtering the FFT signal by each Mel-scale filter is known as the Mel-spectrum.

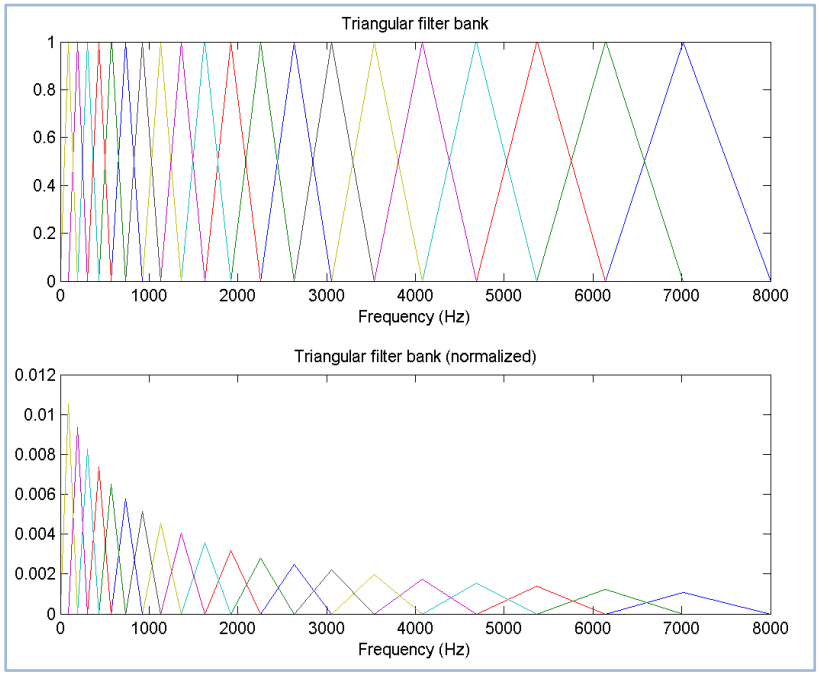


**Figure 5.11 - Mel-scale Bank Filter**

### *5.2.1.8 Discrete Cosine Transform*

DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even).

A DCT computes a sequence of data points in terms of summation of cosine functions oscillating at various frequencies. The idea of performing DCT on Mel-scale is motivated by extraction of the speech frequency domain characteristics. DCT module reduces the speech signal's redundant

information, and reaches the aim of regulating the speech signal into feature coefficients with minimal dimensions.

The DCT of the Mel-scale is computed, resulting in the spectrum. This representation is valuable because it separates characteristics of the source and vocal tract from the speech waveform.

#### *5.2.1.9 MFCC Features*

The stage of feature extraction is responsible for reducing the resources required to describe speech samples accurately and requires algorithms of large complexity. Normally, speech signal is converted into a parameterized sequence of feature vectors by front-end processing to emphasize the characteristics of spoken words and suppress other irrelevant information.The front-end takes audio signal and processes it as a quantized digitized waveform through a sequence of very complex DSP modules to generate a sequence of 39-dimensions

- ***12-dimensions MFCC plus 1-dimension power***
- ***12-dimensions LPC plus 1-dimension power***
- ***12-dimensions ENH-MFCC plus 1-dimension power***

as the base feature for each frame, that can be used in the back-end model, each vector representing the information in a small time window of the

signal. This feature is then extended to 39-dimensions for every feature type by augmenting first-order and second-order time derivatives, in order to capture the transition of the spectrum.

The figures below show waveform (speech signal) for the word "Voyager" with 8 KHz sampling rate and its MFCC features generated by the Front-end module.

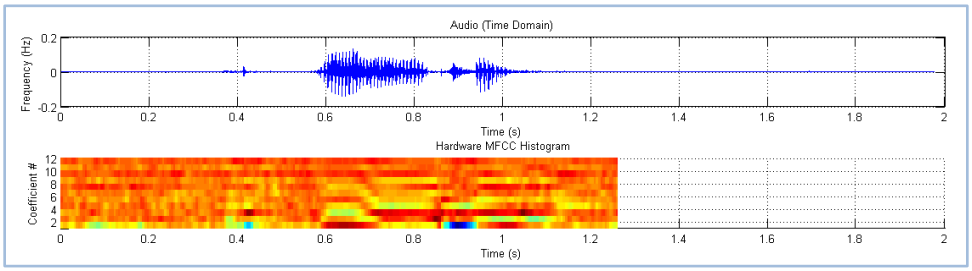


**Figure 5.12 - MFCC Hardware Front-end Features (12-Coefficients)**

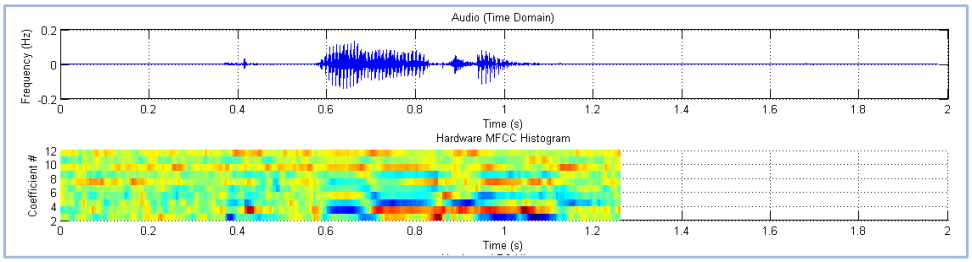


**Figure 5.13 - MFCC Hardware Front-end Features (11-Coefficients)**

### 5.2.2 LPC Front End Subsystem Implementation

As shown in the diagram below, an additional module named Autocorrelation Linear Productive Coding (LPC) used to extract the speech

as LPC features. The basic idea of LPC is to approximate the current speech sample as a linear combination of past samples as shown in the following equation:

$$x[n] = \sum_{k=1}^{p} a_k \, x[n-k] + e[n]$$

| | | |
|---|---|---|
| ***Where*** | ***x[n-k]:*** | ***Previous speech samples*** |
| | ***p:*** | *Order of the model* |
| | ***ak:*** | *Prediction coefficient* |
| | ***e[n]:*** | *Prediction error* |

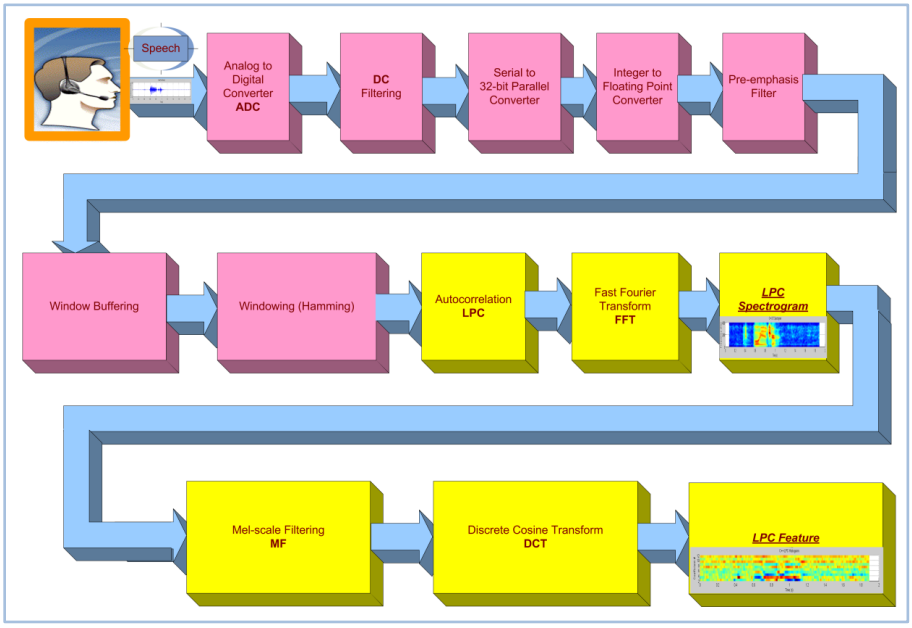


**Figure 5.14 – LPC Front-end Subsystem**

LPC module gets windowed data from the window module for representing the spectral envelope of a digital signal of speech in compressed

form, using the information of a linear predictive model. We use this method to encode good quality speech and provide an estimate of speech parameters.

The goal of this method is to calculate prediction coefficients $a_k$ for each frame. The order of LPC, which is the number of coefficients $p$, determines how closely the prediction coefficients can approximate the original spectrum. As the order increases, the accuracy of LPC also increases. This means the distortion will decrease. The main advantage of LPC is usually attributed to the all-pole characteristics of vowel spectra. Also, the ear is also more sensitive to spectral poles than zeros (M. R. Schroeder) [64]. In comparison to non-parametric spectral modeling techniques such as filter banks, LPC is more powerful in compressing the spectral information into few filter coefficients (K. K. Paliwal and W. B. Kleijn) [65].

#### *5.2.2.1 LPC Spectrogram*

The figures below shows waveform (speech signal) for the words "Onward" and "Voyager" with 8 KHz sampling rate and its LPC spectrogram representation generated by the Front End module.

We used the same MFCC spectrogram function to generate LPC spectrogram:

$$20 * \log_{10}(fft\_real^2 + fft\_imag^2)$$

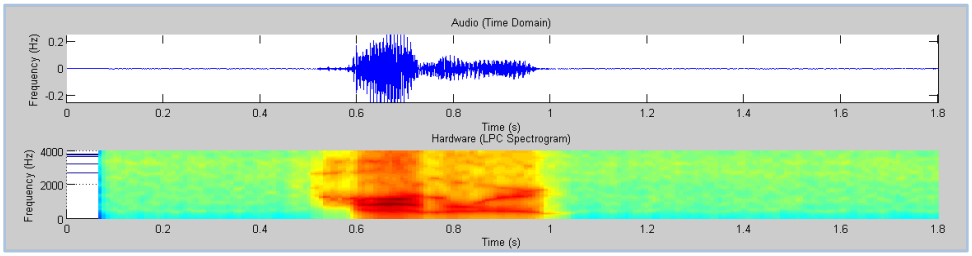


**Figure 5.15 - LPC Hardware Front-end Spectrogram for the word "Onward"**

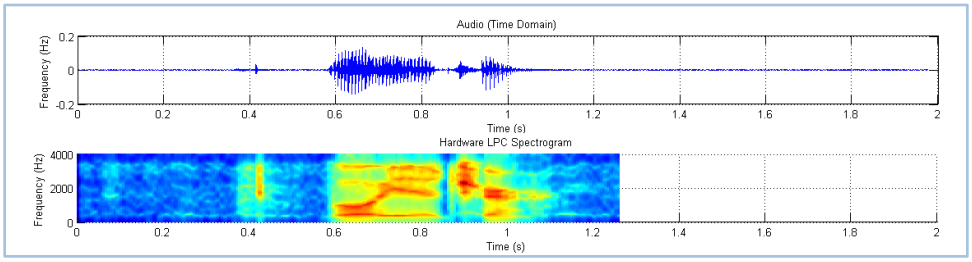


**Figure 5.16 - LPC Hardware Front-end Spectrogram for the word "Voyager"**

This module reads the real and imaginary FFT data from the FIFOs whenever data is available. The real data is multiplied by itself through a floating-point multiply Megafunction core and the imaginary data is multiplied by itself through another core. The results of these multiplies are routed to the inputs of the floating-point adder. The result of the adder is then routed to the input of the floating-point log Megafunction. Altera's log Megafunction actually calculates the natural log of the input.

### *5.2.2.2 LPC Features*

The figures below show waveform (speech signal) with 8 KHz sampling rate and its 12-Coefficients (Figure 5.17) & 11-Coefficients (Figure 5.18) LPC features generated by the Front-end module.

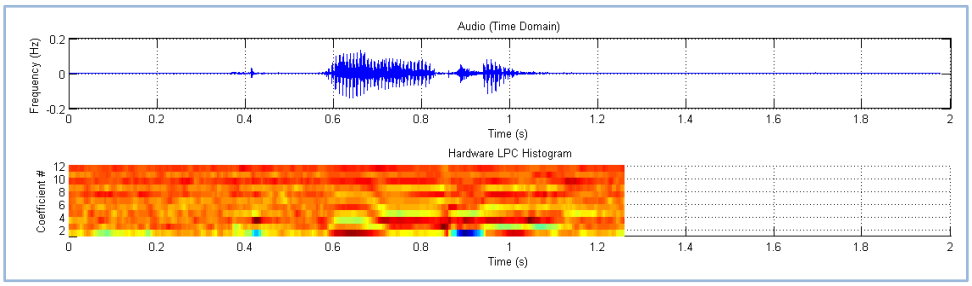


**Figure 5.17 - LPC Hardware Front-end Features (12-Coefficients)**

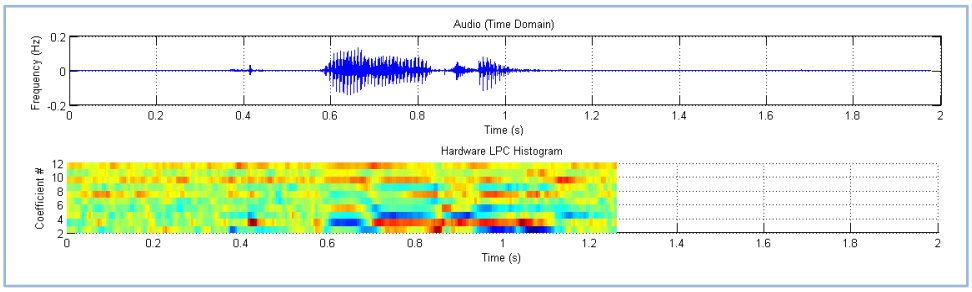


**Figure 5.18 - LPC Hardware Front-end Features (11-Coefficients)**

## 5.2.3 ENH-MFCC Front End Subsystem Implementation

The spectrum enhancement module is used to generate enhanced Mel-scale Frequency Cepstral Coefficients (ENH-MFCC) set of features. We implemented this module as shown in the diagram below to perform the enhancement algorithm on the LPC spectrum signal.

The ENH-MFCC features have a higher dynamic range than regular MFCC features, so these new generated features will help the back-end in improving the recognition quality and accuracy.

The equations below show the enhanced Mel-scale Frequency Cepstral Coefficients (ENH-MFCC) algorithm. The algorithm uses only the single-sided spectrum, so the state machine starts the calculations when 128 data points have been written into the input RAM.

$$xi = ci\ yi + bi \quad \rightarrow \quad e(xi) = e(yi)$$

Where:

***xi = corrupted signal***
***yi = pure signal***
***ci = noise***
***bi = background noise***

***Enhanced Spectrum = Input LPC Spectrum / Global Information (Average) + Local Information (Average) + Estimated Background.***

$$e(i) = G\left(\frac{x_i}{\alpha E_i + \beta N_i + \gamma f_i + F}\right)$$

$$e(i) = G\left(\frac{c_i y_i}{\alpha \sum_{i=0}^{N} c_i y_i + \beta \sum_{\substack{j=-M \\ j \neq i}}^{M} c_{i+j} y_{i+j} + \gamma \overline{c_i}\,\overline{y_i} + F}\right)$$

$x_i = c_i y_i,\ f = \overline{c_i}\,\overline{y_i}\ (estimate\ of\ background)$
where $\alpha, \beta, \gamma, and\ F = constant$

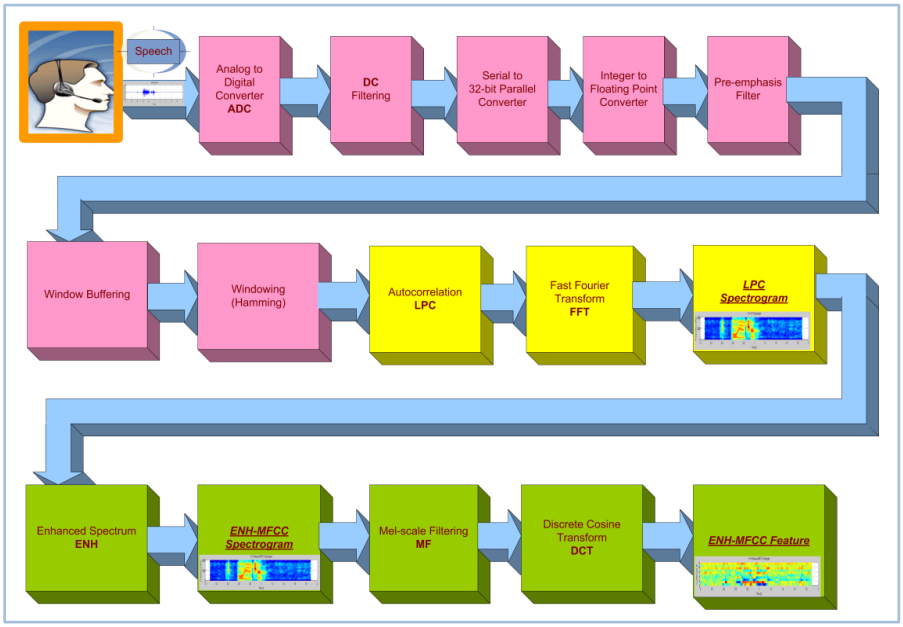


**Figure 5.19 – ENH-MFCC Front-end Subsystem**

### *5.2.3.1 ENH-MFCC Spectrogram*

The figures below show waveforms (speech signals) for the words "Onward" and "Voyager" with 8 KHz sampling rate and its LPC spectrogram representation generated by the Front End module.

We used the same MFCC spectrogram function to generate LPC spectrogram:

$$20 * \log_{10}(fft\_real^2 + fft\_imag^2)$$

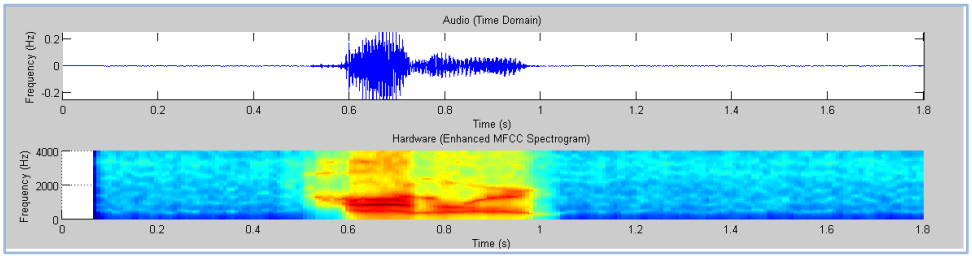


**Figure 5.20 - ENH-MFCC Hardware Front-end Spectrogram for the word "Onward"**

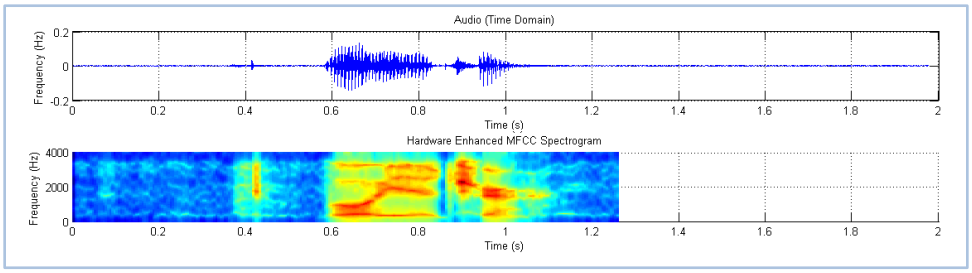


**Figure 5.21 - ENH-MFCC Hardware Front-end Spectrogram for the word "Voyager"**

### *5.2.3.2 ENH-MFCC Features*

The figures below show waveform (speech signal) with 8 KHz sampling rate and its 12-Coefficients (Figure 5.21) & 11-Coefficients (Figure 5.22) LPC features generated by the Front End module.

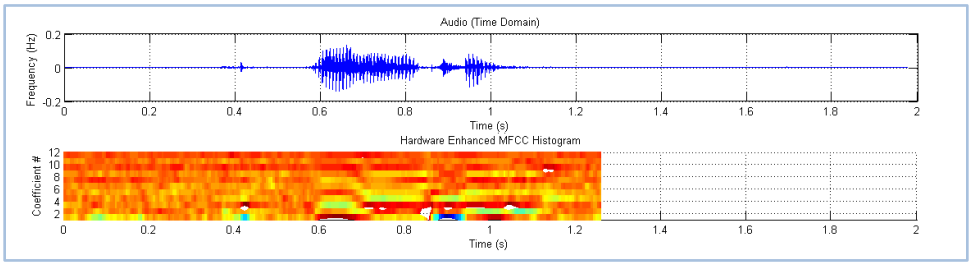


**Figure 5.22 - ENH-MFCC Hardware Front-end Features (12-Coefficients)**

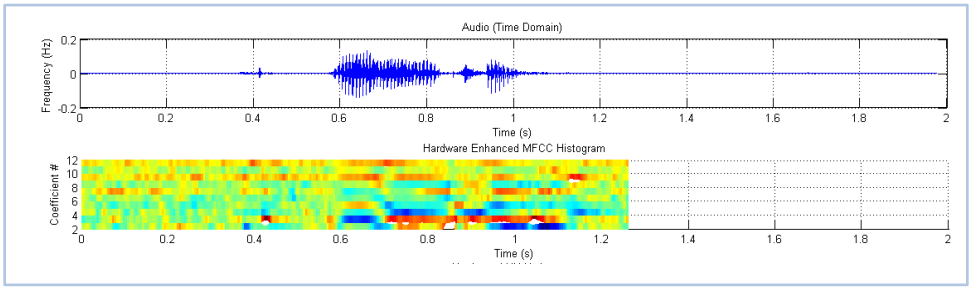


**Figure 5.23 - ENH-MFCC Hardware Front-end Features (11-Coefficients)**

### 5.2.4 Voice Activity Detector Implementation

There are various kinds of voice that may contain actual speech or background noise. In our Front-end of WUW Speech Recognition System it is imperative to monitor and detect any speech signal for efficient usage by the Back-end stage. To achieve speech / non-speech detection, as shown in fig (5.24) we designed and implemented Voice Activity Detector (VAD) models. The VAD continuously monitors system and calculates for every frame if the signal is a speech signal or a background noise. VAD used three inputs (Log Energy, LPC Features, and MFCC Features) to decide whether to turn VAD ON or VAD OFF. Those features are finely tuned to detect any voice in the speech frame. Each feature has an independent detection method. Depending on the calculations done by the features, the overall VAD logic reacts by turning the VAD output on/off (Speech or Noise Background). Each feature calculates the presence of speech by calculating Variance and Mean deviation. Each feature has its own flag to be set. Each

feature is finely tuned to meet certain criteria before its feature flag is set. Once all these flags are true, the overall logic of VAD triggers and the VAD turns on.

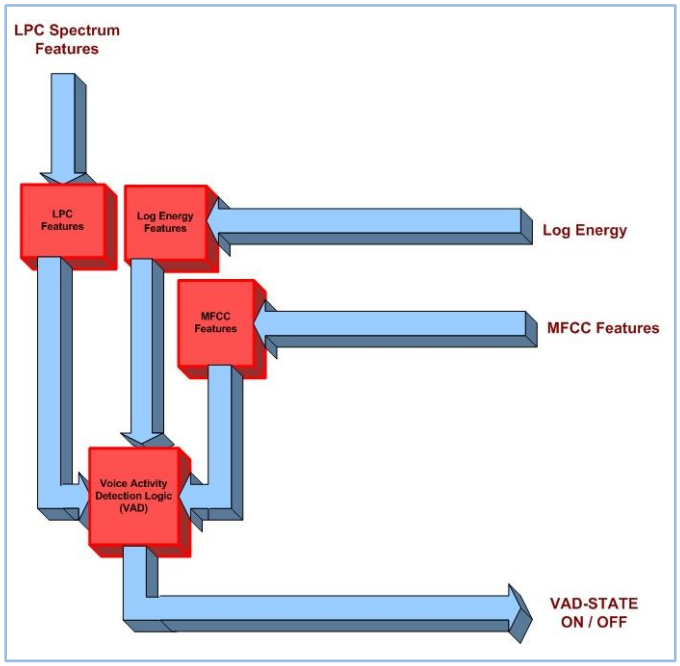


**Figure 5.24 – Voice Activity Detector Modules**

### *5.2.4.1 Log Energy Feature*

The Energy feature uses the change in mean frame energy to detect if there is speech in the frame. If there is a drastic change in frame energy the Energy flag is set. This feature function receives the frame energy parameter (`log2_frame_energy`) from the Hamming Window module. It uses this each new frame energy to calculate the overall frame energy mean and compare it to the present frame energy. Mean frame energy (`mean_log2_fm_en`) is

dependent on a variable called lambda (*lambda_LTE*) which is calculated differently in the first few frames. The mean is ignored during the VAD ON stage and is stored in a different variable (*mean_log2_fm_en_VAD_ON*). If the difference between the mean frame energy and the current frame energy is greater than certain threshold (*SNR_THRESHOLD_VAD*), the Energy flag is set.

#### *5.2.4.2 Mel-frequency Cepstral Coefficient Feature*

The Mel-frequency Cepstral Coefficients are used to calculate current frame feature value based on the values in the MFCC vector. When the mean (*mean_vad_mfcc_feature*) of all the frames measured is over the current frame MFCC feature value (*local_aver_mfcc_fea_val*), the flag is set (VAD_MFCC_State_Flag). The mean is calculated differently depending on the frame counter and the behavior of the system. It dynamically changes to adapt to the system so it can detect the signal in a better way. For example, the mean for the first fifteen frames is equal to the current frame MFCC value or the overall mean up to that point whichever is of higher value. The mean is slowed down if the current frame feature value is too low compared to the mean calculated up to that point. The mean is ignored during the stage where VAD is turned ON and is stored into a different variable (*mean_vad_mfcc_feature_VAD_ON*).

#### *5.2.4.3 Leaner Predictive Coding Feature*

Leaner Predictive Coding Spectrum feature function is different from the aforementioned features mainly because it is uses the variance to detect if a signal is present. The values in the vector are calculated and the variance for each individual frame is stored. When the current variance (`var_spec`) is much larger than the average variance (`aver_var_spec`) of the previous frames, it indicates that there is speech or not in the frame. The average variance is changes based on how much the current value varies. If the current variance value is really low then it indicates that the signal dint change a lot so the overall average doesn't change much either. The variance flag (`VAD_SPEC_State_Flag`) is set when there is a sudden change in variance when compared to previous frames and is turned off if the change is minimal.

# 6 Results and Comparisons

Having Front-end and Voice Activity Detector of WUW designed, implemented, and running on FPGA, it is now necessary to compare and evaluate it with WUW-SR system designed with C++ software and running on personal computer, to prove that both hardware and software WUW-SR systems are working identical.

The experiment results from intermediate stages of the hardware front-end process as well as spectrograms and features are presented, and the results produced by the implementations have been presented.

## 6.1 CODEC Audio Data Interface

*Module: codec_dsp_interface.v*

*Megafunction Cores: None*

This is the interface to the audio CODEC which receives ADC data from the CODEC and sends DAC data to the CODEC. The interface operates in DSP mode only and assumes that the CODEC is the transfer master (the CODEC generates the BCLK and L/R clocks.) In DSP mode, the CODEC generates a single clock pulse on the LRCLK_in signal; data is shifted in/out on the subsequent clocks. The CODEC shifts ADC data out on the falling edge of BCLK (CODEC_BCLK_in) as MSB first, so the ADC data (CODEC_SDATA_in) is registered on the rising edge of BCLK in the interface logic. Likewise, the CODEC registers DAC data (CODEC_SDATA_out) on the rising edge of BCLK, so the interface logic shifts DAC data out on the falling edge.

The interface is designed to send/receive 16, 20, 24 or 32-bit wide data to/from the CODEC. The 2-bit input (WORDLENGTH_SEL_in) determines

the number of bits to shift per channel: 00 = 16-bit, 01 = 20-bit, 10 = 24-bit, 11 = 32-bit. The output ADC_DATA_out contains the parallel left and right data from the CODEC. Depending on the selected wordlength the output data is formatted as:

| *Wordlength* | *Left Channel Data* | *Right Channel Data* |
|---|---|---|
| 16 | ADC_DATA_out[31:16] | ADC_DATA_out[15:0] |
| 20 | ADC_DATA_out[39:20] | ADC_DATA_out[19:0] |
| 24 | ADC_DATA_out[47:24] | ADC_DATA_out[23:0] |
| 32 | ADC_DATA_out[63:31] | ADC_DATA_out[31:0] |

A single BCLK data valid pulse (ADC_DV_out) is generated when all bits are registered. The DAC data interface is designed to read data out of a FIFO, so the signal DAC_FIFO_RDEN_out generates a single BCLK read enable pulse (if DAC_FIFO_EMPTY_in is not active) at the end of each transfer to retrieve the next DAC words. Since the DAC data is sent MSB first, the input data must be formatted accordingly for the selected wordlength.

| *Wordlength* | *Left Channel Data* | *Right Channel Data* |
|---|---|---|
| 16 | DAC_DATA_in[63:48] | DAC_DATA_in[47:32] |
| 20 | DAC_DATA_in[63:44] | DAC_DATA_in[43:24] |
| 24 | DAC_DATA_in[63:40] | DAC_DATA_in[39:16] |

32 DAC_DATA_in[63:32] DAC_DATA_in[31:0]

*Testbench: tb_codec_dsp_interface.v*

The CODEC interface testbench verifies that the module receives ADC data and transmits DAC data correctly. Each wordlength is tested. The testbench reports any errors and issues a test pass/fail message at the end of the test.

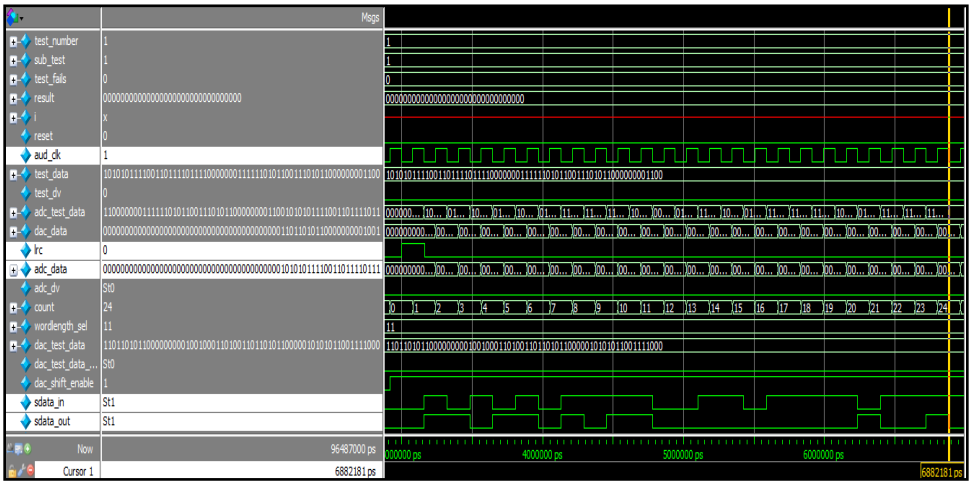


**Figure 6.1 – CODEC DSP Interface Simulation Waveforms**

## 6.2 Integer-to-Floating-Point Function

*Module: int16_to_float32_wrapper.v*

*Megafunction Cores: ALTFP_CONVERT (int_2_float.v)*

This module converts 16-bit, signed integer data to single-precision (32-bit) floating point values. The input data is simply routed through the int_2_float.v Megafunction core. The latency of the core is 6 clocks. The

dv_dly_sr.v module instantiates an 8 clock shift register to delay the AUDIO_DV_in (audio data valid) input to the FP_DV_out (floating-point data valid) signal to allow for the latency of the conversion.

*Testbench: tb_int16_to_float32_wrapper.v*

The integer-to-floating-point testbench writes the following signed, 16-bit integers into the module under test: -32767, -16384, -1, 0, 1 16384, 32768. These values represent the full range of the expected input data. The testbench determines if the correct floating-point values are generated by the module under test. The MATLAB script tb_int2float.m was written to verify the correct hexadecimal, single-precision, floating-point outputs. Any failures are flagged and a pass/fail message is displayed at the end of the test.

*MATLAB script tb_int2float.m:* This M-File displays the integers and the corresponding floating-point values used for the int2float verilog / megafunction test bench. The output of *tb_int2float.m script:*

```
a =
      32767     16384         1         0        -1    -16384    -32768
b =
      46fffe00  46800000  3f800000  00000000  bf800000
      c6800000  c7000000
```

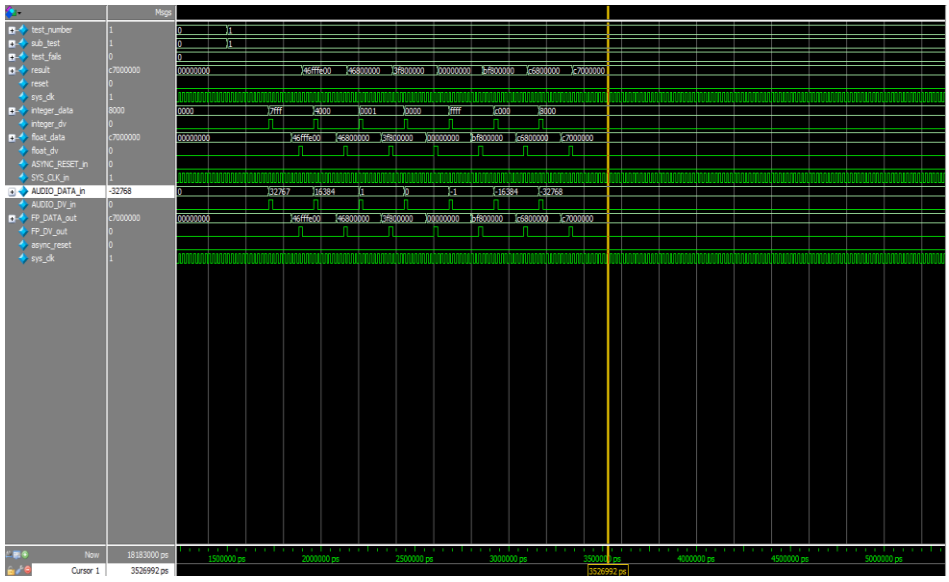


**Figure 6.2 – 16-bit Integer-to-32bit Floating-point Simulation Waveforms**

## 6.3 Pre-Emphasis Function

*Module: pre_emphasis.v*

*Megafunction Cores: ALTFP_MULT (mult_float_32.v)*

*ALTFP_ADD_SUB (add_sub_float_32.v)*

The pre-emphasis function performs the following algorithm:

***output = input - PRE_EMPH_FACTOR * previous_input***

When input data is valid (DV_in), the input data (DATA_in) is registered as the input_sample. At the same time, the previous input_sample is registered as prev_sample. The prev_sample data is multiplied by the PRE_EMPH_FACTOR (0.975, or 0x3F79999A) using the mult_float_32.v

Megafunction. The output of the multiplication is then subtracted from input_sample by the add_sub_float_32.v Megafunction. The latency of the multiplier is 11 clocks and the latency of the subtract is 7 clocks. A 19-clock delay is applied to DV_in to generate DV_out (data valid out) to allow for the latencies of the cores.

*Testbench: tb_pre_emphasis.v*

The pre-emphasis function testbench uses the same test values as the integer-to-float testbench, except in floating point notation. The test values are written in and the output values are compared against the know answers to verify module under test operation. Any failures are flagged and a pass/fail message is displayed at the end of the test.

A MATLAB script (tb_preemphasis.m) was written to generate the expected output values for testbench comparison.

*The output of: tb_preemphasis.m script*

```
result =

 46fffe00   c6732f4e   c679959a   bf79999a   bf800000   c67ffc1a
   c6833333
```

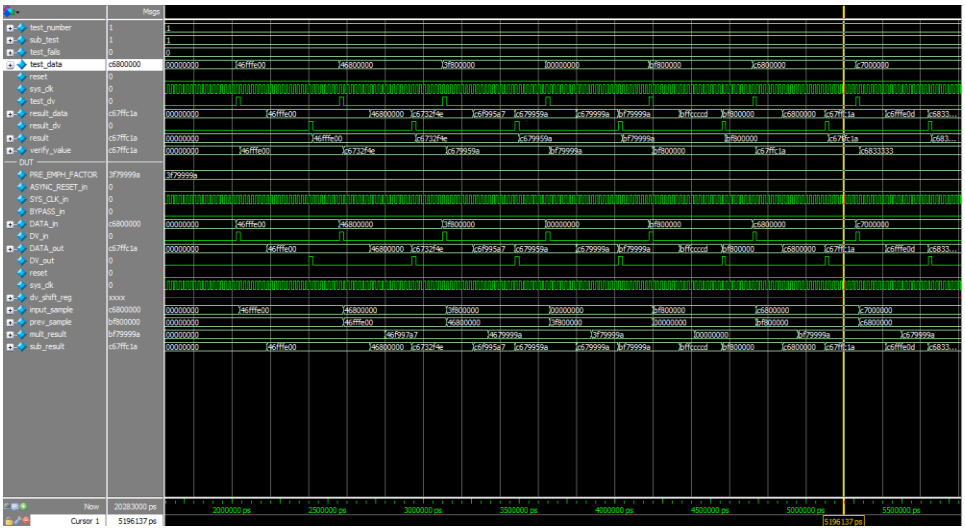


**Figure 6.3 – Pre_emphasis Filter Simulation Waveforms**

## 6.4 Hamming Window Function

*Module: hamming_window.v*

*Megafunction Cores: ALTFP_MULT (mult_float_32.v)*

*ROM: 1-PORT (rom_256x32bits_hamcoef.v)*

*RAM: 2-PORT (dpram_256x32bits.v)*

The Hamming window function smoothes the input audio data with a Hamming curve prior to the FFT function. A 32-bit, 256 deep dual-port RAM (DPRAM) stores 200 input samples. A state machine handles moving audio data into the RAM, and pulling data out of the RAM to be multiplied by the Hamming coefficients, which are stored in a ROM memory. The ROM initialization file was created using a MATLAB script, hamming_coef.m,

which uses the MATLAB function "hamming" to generate the 200 data points and convert them to hexadecimal, 32-bit, single precision floating point values. The script writes the initialization file for the Megafunction IP and also a file used for the testbench.

The state machine first stores 40 incoming audio samples in the RAM, starting at address 0x3F and decrementing to location 0x00 (this mimics a FIFO). Once the 40 samples have been stored, the state machine pulls data out of the RAM starting at location 0xFF (oldest sample first), decrementing to 0x00. This data is multiplied by the corresponding Hamming coefficient from the ROM. The result of the multiplication is stored in a FIFO for use by the FFT. The state machine accounts for the latency of the multiplication core. Simultaneously, the data in the RAM is shifted up in the memory by 40 locations (the upper 40 words are discarded). So the previous contents from 0x00 to 0xBF are moved to 0x40 to 0xFF, leaving the bottom 40 words open for the next set of audio samples.

To summarize, each FFT calculation is a weighted average of present and past audio samples.

*Testbench: tb_hamming_win.v*

The Hamming window testbench writes the value of 1 (0x3F800000) 200 times into the SAMPLE_DATA_in input of the Hamming window

module. This will cause all of the internal multiplications of the Hamming coefficient ROM data to be multiplied by 1. When the input RAM is completely filled, the output FIFO should contain the values of the ROM.

The input data is written in 40 words at a time and then the testbench waits for the FIFO full output to be true. The testbench will then read the FIFO data. The fourth time the FIFO is read, the FIFO output data is compared to the data in the file "tb_ham_coef.txt", which holds the Hamming window coefficients used to initialize the Hamming window lookup table ROM.

The testbench will report that all data has passed, if that is the case. Otherwise, the errors will be reported. The testbench also has a Modelsim wave file, wave_ham_win.do, with test signals. The analog test signal is an analog representation of the FIFO output data. After the fourth read from the FIFO, the analog data should show the Hamming window curve.

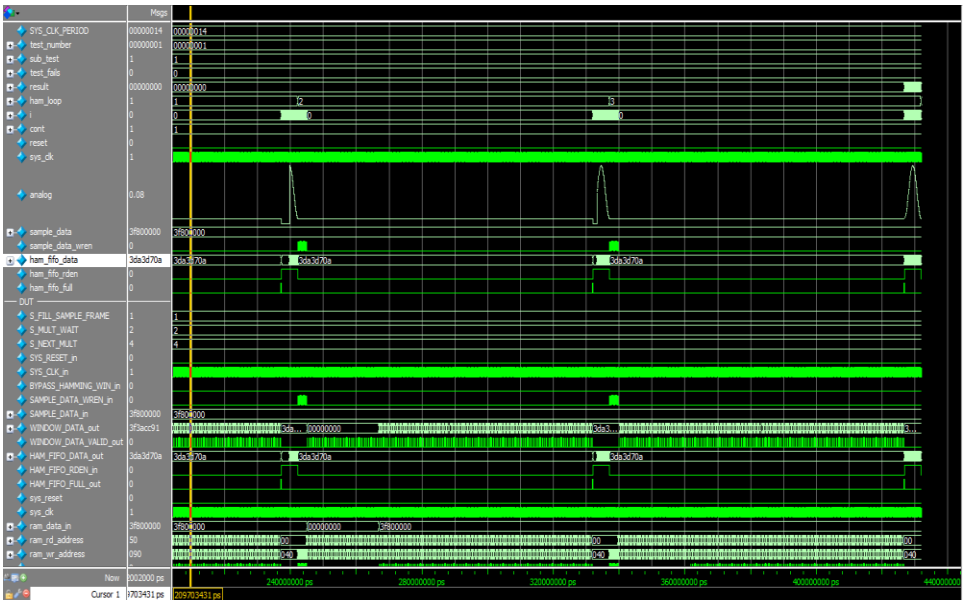

**Figure 6.4 – Hamming window Simulation waveforms**

## 6.5 Fast Fourier Transform (FFT) Function

*Module: fft_interface.v*

*Megafunction Cores: FFT (fft_256.v)*

*FIFO (sync_fifo_256x32bits.v)*

The FFT interface module instantiates a single-precision, FFT of length 256 (actually 2^3 to 2^8). The architecture of the FFT is variable streaming, natural order in and out and supports an inverse FFT function as well. A state machine reads data out of the Hamming window FIFO when it is full and streams this data into the FFT. The streaming output of the FFT writes the complex data (real & imaginary) into separate 256 x 32-bit FIFOs

for extraction by the spectrogram function.

*Testbench: fft_256_tb.v*

The Megafunction FFT wizard automatically generates a testbench for the FFT. This testbench only tests the FFT megafunction, not the FFT interface module. The testbench writes input data for various FFT lengths from the text files "fft_256_real_input.txt" and "fft_256_imag_input.txt" and generates "fft_256_real_output_ver.txt" and "fft_256_imag_output_ver.txt". Two other files are also generated: "fft_256_blksize_report.txt", which contains the FFT data for each test and "fft_256_inverse_report.txt", which contains the inverse (1) or normal (0) selection for each test. The testbench runs four sets of data in this order of lengths: 256, 16, 256, 16. No inverse FFTs are simulated.

The FFT Megafunction wizard also generates a MATLAB simulation for the FFT model (the files are "fft_256_tb.m" and "fft_256_model.m".) This simulation reads the FFT input data text files and the block report and inverse report text files from the Verilog simulation and generates the output "fft_256_real_output_c_model.txt" and "fft_256_imag_output_c_model.txt". The MATLAB script "tb_fft_result_compare.m" compares the outputs of the MATLAB model and the simulation for discrepancies.

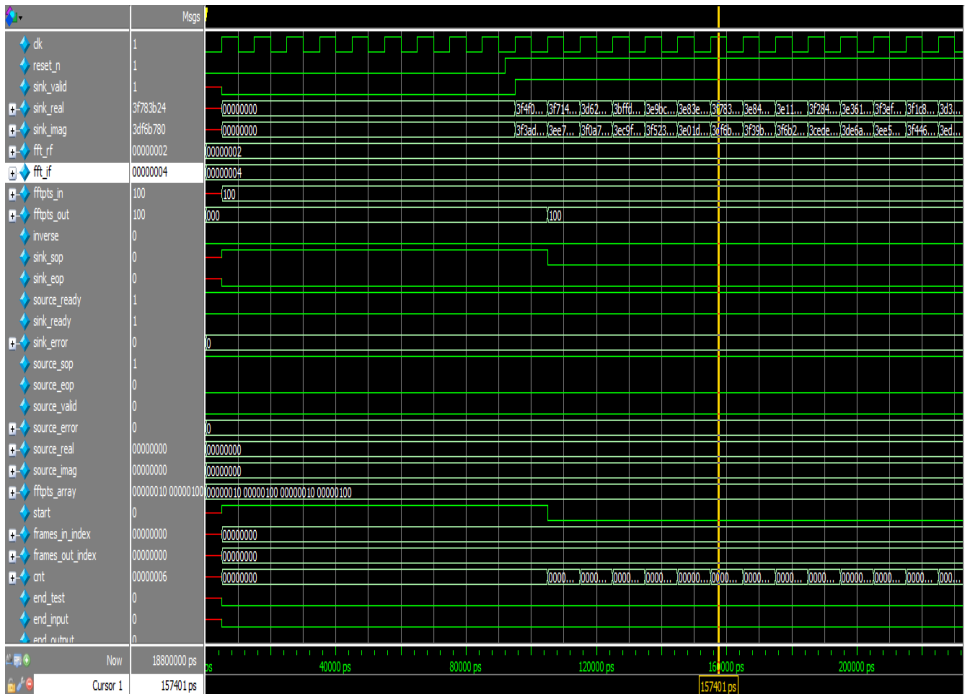


**Figure 6.5 – FFT Simulation Waveforms**

## 6.6 Spectrogram Function

*Module: spectrogram.v*

*Megafunction Cores: ALTFP_MULT (mult_float_32.v) x 3*

*ALTFP_ADD_SUB (add_sub_float_32.v)*

*ALTFP_LOG (log_float_32.v)*

This module takes the complex data generated by the FFT and performs the algorithm:

$$20 * \log_{10}(fft\_real^2 + fft\_imag^2)$$

which is the spectrogram output. The module reads the real and imaginary FFT data from the FIFOs whenever data is available. The real data is multiplied by itself through a floating-point multiply Megafunction core, the imaginary data is multiplied by itself through another. The results of these multiplies are routed to the inputs of the floating point adder. The result of the adder is then routed to the input of the floating-point log Megafunction. Altera's log Megafunction actually calculates the natural log of the input; to scale the final result to log10, the output of the log function is multiplied by 20 / ln(10) - this is a constant value of 0x410AF967 (8.685889).

A counter is used to compensate for the latency of the Megafunctions (generate the FFT FIFO read enables and the spectrogram output write enable.)

*Testbench: tb_spectrogram.v*

The spectrogram testbench reads the files generated by the FFT testbench, "fft_256_real_output_ver.txt" and "fft_256_imag_output_ver.txt" as input to the spectrogram module. The testbench also reads the text file "spec_ver_data.txt", which is created by the MATLAB script “tb_spec_ver_gen.m". The MATLAB script reads the same FFT output text files and calculates the spectrogram value. The verilog testbench compares the DUT output to the data in "spec_ver_data.txt" to verify that the data

matches. Due to rounding differences between MATLAB and the Megafunction cores, there is a slight difference between the values. The testbench reports differences of one or two counts.

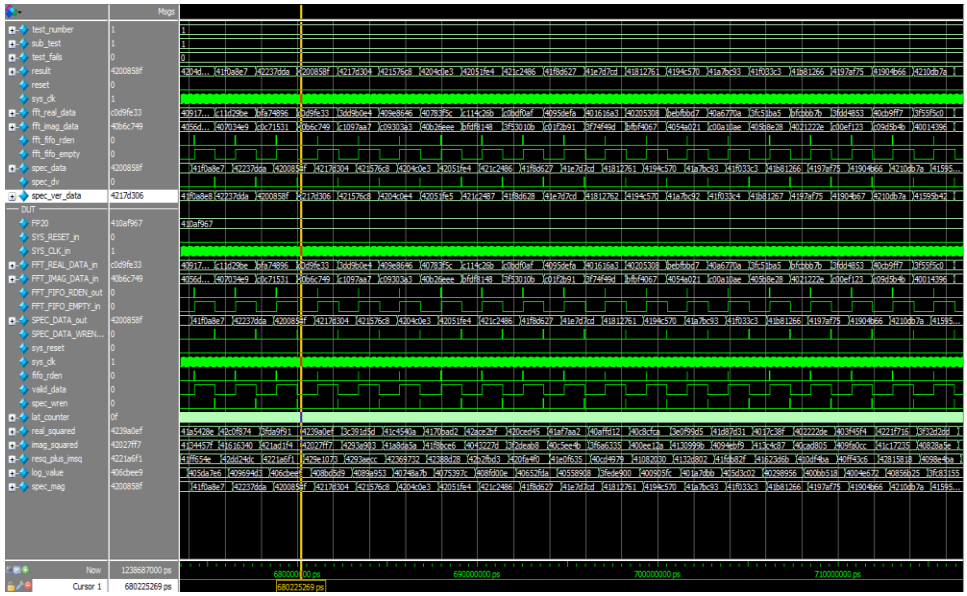

**Figure 6.6 – Spectrogram Simulation Waveforms**

## 6.7 Linear Predictive Coding (LPC) Function

*Module: lpc_module.v*

*Megafunction Cores: ALTFP_MULT (mult_float_32.v) x 3*

*ALTFP_ADD_SUB (add_sub_float_32.v)*

*ALTFP_DIV (div_float_32.v)*

*RAM: 2-PORT (dpram_256x32bits.v)*

*RAM: 2-PORT (dpram_32x32bits.v) (x3)*

*FIFO (sync_fifo_256x32bits.v)*

This module calculates the Linear Predictive Coding coefficients of the post Hamming Window audio data. In the Hamming Window module, when the windowed data is being written into the FFT FIFO, the data is also written out of the Hamming Window module into the LPC module via the HAM_DATA_in and HAM_DV_in inputs. The data is written into a RAM block in the LPC Module.

When 256 data points are written, a state machine in the LPC module begins calculation of the LPC coefficients. The state machine runs the LPC calculations in this order: (1) Calculate Auto-Correlation coefficients, (2) Calculate the LPC Levinson-Durbin coefficients, (3) Store the LPC coefficients in a FIFO to be read by the FFT module. All of the algorithms' calculations share the floating-point Megafunction cores. The state machine sets a calculation select value (calc_sel) which control muxes for the core's inputs for current calculation. A counter (calc_wait) is used by the state machine to wait for the cores to complete their calculations.

*Auto-Correlation Calculation*

Once the input RAM is filled with data from the Hamming window, the state machine will begin executing the auto-correlation coefficient (acc) calculation. The first state of this process initializes the RAM contents from the previous calculations. The S_INIT_RAM state writes zeros into the auto-

correlation, temp and LPC coefficient DPRAMs. When all RAMs have been initialized the algorithm indices are set to zero, as well as the auto-correlation sum (acc_sum) and the auto-correlation function is set for calc_sel. Each auto-correlation coefficient is then calculated as the sum…

```
for j = 0 to 17
    for i = 0 to 255
        acc_sum = acc_sum + (sample_data[i] * sample_data[i+j])
    next i
    acc_ram[j] = acc_sum
next j
```

The states S_AC_READ_OFFSET, S_WAIT_FOR_AC_DATA, S_AC_CALC_WAIT, S_AC_NEXT_WAIT perform and store the auto-correlation coefficients in RAM. When the ACC calculation is complete the state machine will advance to the beginning state for LPC calculation (S_START_LD_ALG).

*LPC Calculation*

The LPC coefficients require several calculations for each coefficient. In state S_START_LD_ALG, the first auto-correlation coefficient is read from the RAM and stored as the first LPC coefficient (lpc_ram[0] = acc_ram[0]) and the first alpha value. The i and j indices are set to 1 and the state

machine advances to S_START_S_CALC. The i index keeps tracks of the LPC coefficient, while the j index is used for the various internal calculations. The LPC algorithm is executed as follows (from MATLAB):

```
for i = 1:17
 % calculate the sum
for j = 1: (i-1)
        s = s + lpc_ram(j) * acc_ram(i-j);
end
% calculate k
 k = -(acc_ram(i) + s) / alpha;
 % calculate lpc_ram[1:(i-1)]
 for j = 1: (i-1)
        temp_ram(j) = lpc_ram(j) + (k * lpc_ram(i-j)));
 end
 for j = 1: (i-1)
        lpc_ram(j) = temp_ram(j);
 end
 % store new value of lpc_ram[i] and calculate new alpha
 lpc_ram(i) = k;
         alpha = alpha * (1-k*k);
```

% next interation

End

States S_START_S_CALC and S_WAIT_FOR_S_CALC compute the s value of the algorithm. States S_START_K_CALC, S_WAIT_FOR_K_CALC_SUM and S_WAIT_FOR_K_CALC compute the k value. States S_START_AJ_CALC, S_AJ_CALC1, S_AJ_CALC2, S_AJ_CALC3, S_WAIT_FOR_AJ_CALC, S_UPDATE_AJ_DLY, S_UPDATE_AJ and S_AI_CALC calculate the temp_ram and lpc_ram coefficients.

When all of the LPC coefficients have been calculated and stored, the state machine jumps to S_LOAD_FFT_FIFO where the FFT_FIFO is loaded with LPC coefficients 1 through 17 and then zero padded for the remaining values.

*Testbench: tb_lpc.v*

The LPC module testbench writes the data from the file "lpc_ham_test_data.txt" into the input of the LPC module (DUT). The testbench displays the final LPC coefficients in transcript window. The MATLAB script "lpc.m" reads the same input file and displays the results. The results between the MATLAB script and the simulation output are compared to verify operation.

## 6.8 Mel-Frequency Cepstral Coefficients (MFCC) Function

*Module: mfcc_module.v*

*Megafunction Cores : ALTFP_MULT (mult_float_32.v) x 3*

*ALTFP_ADD_SUB (add_sub_float_32.v)*

*ALTFP_LOG (log_float_32.v)*

*ALTFP_COMPARE (fp_comp_lt_single.v)*

*ROM: 1-PORT (mel_filter_rom.v)*

*ROM: 1-PORT (rom_512x32bits_dct_matrix.v)*

*RAM: 2-PORT (dpram_32x32bits.v) (x2)*

*RAM: 2-PORT (dpram_256x32bits.v)*

The MFCC HDL module calculates the Mel-Filtered Cepstral Coefficients (MFCC) of the input spectral data (audio, LPC or Enhanced spectrums). The MFCC calculation has two major sections, the Mel-Filter and the Discrete Cosine Transform (DCT).

### 6.8.1 Mel-Scale Filter Function

Input spectrum data is written into a RAM temporarily until all (129) samples are received. When all samples are received, the state machine begins processing the input data through the Mel-Filter. The Mel-Filter consists of a set of triangular coefficients which are stored in mel_filter_rom.

The Mel-Filter consists of 25 bands. The input spectrum is multiplied by the coefficients in these bands and the energy for each band is summed. The resulting sums represent a Mel-Scale optimized spectrum. A graph of the Mel-Filter coefficients is shown below:

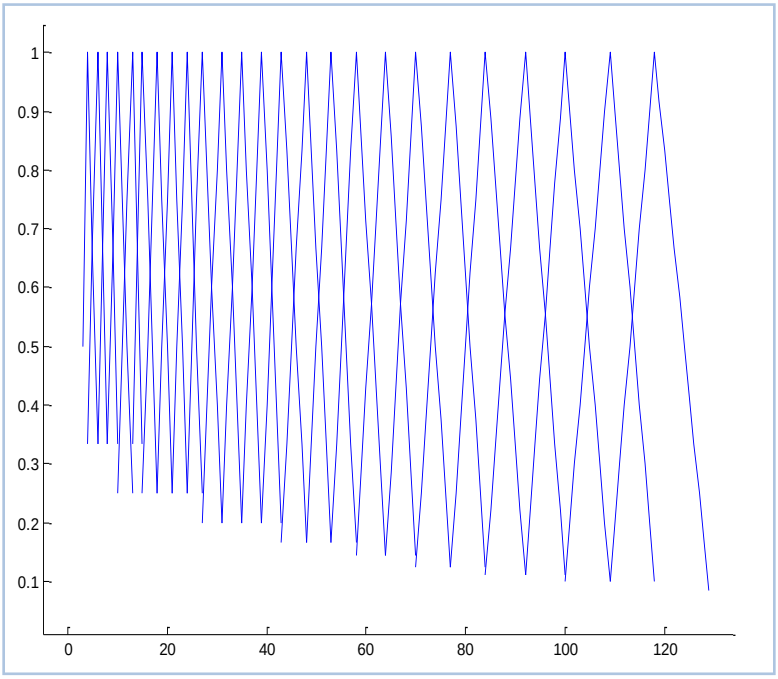


**Figure 6.7 – Mel-scale Filter Output**

The Mel-Filter coefficients are arranged in a ROM (mel_filter_rom.v), with a header for each filter section. The header defines the starting bin (bits 15-0) and length (bits 31-16). The state machine begins by extracting the first 32-bit word from the Mel Filter ROM and stores the starting bin and length. The state machine then reads out the filter coefficients and corresponding spectrum data. The spectrum data is multiplied by the filter coefficient and

the results added to a running sum.

When all data is summed for the current filter, the natural logarithm (ln) is calculated for the filter sum. If the ln of the filter sum is less than an energy floor constant, then the ln of the energy floor constant is stored Mel Filter RAM. Otherwise, the ln of the filter sum is stored. This procedure is repeated for each filter band. This is the equivalent algorithm for the Mel Filter calculation:

```
for i = 0 to MEL_FILTER_COUNT - 1
        filter_sum = 0
            for j = starting_bin to starting_bin+length
            filter_sum = filter_sum + filter_coefficient * spectrum_data
next j
            filter_sum_ln = ln(filter_sum)
                if filter_sum <= ENERY_FLOOR then
                        mel_filter_result(i) = ENERGY_FLOOR_LOG
                else
                        mel_filter_result(i) = filter_sum_ln
next i
```

The next step in the MFCC algorithm is to calculate the Mel Filter energy. Each result of the Mel Filter sum calculation is read from the Mel-

Filter RAM and summed. If the resulting Mel-Filter energy is less than or equal to 0, all Mel-Filter Cepstral Coefficients are written as 0. Otherwise the Discrete Cosine Transform of the Mel-Filter sums is calculated.

### 6.8.2 Discrete Cosine Transform (DCT) Function

The DCT is applied to the Mel-Filtered spectrum to compress the spectrum and generate the Mel-Scale Cepstral Coefficients. The DCT algorithm in the MFCC module is implemented with a DCT Matrix coefficient lookup table. The table was specifically generated for a 25-band input spectrum with 13 output coefficients. The algorithm for generating the DCT lookup table is:

```
for i = 1 to NumCepstralCoeff - 1
        for j = 0 to NumChannels - 1
                Mx[(i-1)*NumChannels+j] = cos (pi * i/NumChannels *
(j+0.5))
        next j
next i
```

where: Mx = DCT matrix

NumCepstralCoeff = 13

NumChannels = 25

This results in a table of 300 coefficients. The DCT matrix is stored in the ROM rom_512x32bits_dct_matrix.v.

The MFCC state machine calculates the DCT of the Mel-Filtered spectrum if the Mel-Filter energy result is greater than 0. The state machine implements the DCT using the following algorithm:

```
for i = 0 to NumCepstralCoeff - 2
        DCToutData[i] = 0.0F;
                for j = 0 to NumChannels - 1
                        DCToutData[i] = DCToutData[i] + MelFiltlog[j] *
                        Mx[i*NumChannels+j]
                next j
next i
```

where: Mx = DCT matrix

NumCepstralCoeff = 13

NumChannels = 25

DCToutData = the DCT output vector result

MelFiltlog = the Mel-Filter (ln) table output

The last value of the DCT output vector is set to the Mel-Filter Energy result, calculated prior to the DCT.

```
DCToutData[NumCepstralCoeff-1] = MelFiltEnergy;
```

The DCT results are written into a RAM memory as the Mel-Filtered Cepstral Coefficients. The MFCCs can then be read by subsequent HDL modules.

## 6.9 Enhanced Spectrum Function

*Module: enhanced_spec.v*

*Megafunction Cores: ALTFP_MULT (mult_float_32.v) x 3*

*ALTFP_ADD_SUB (add_sub_float_32.v)*

*ALTFP_DIV (div_float_32.v)*

*RAM: 2-PORT (dpram_256x32bits.v) (x2)*

The Enhanced Spectrum HDL module performs an enhancement algorithm on the LPC Spectrogram output data. When the spectrogram is writing out LPC spectrogram data to the top level, the data is also written into the input of the Enhanced Spectrogram (EnhSpec) module via LPC_SPEC_DATA_in and LPC_SPEC_DV_in. The data is written into a RAM block in the EnhSpec module.

The algorithm uses only the single-sided spectrum, so the state machine starts the calculations when 128 data points have been written into the input RAM. Similar to the LPC Module, all of the algorithms' calculations share the floating-point Megafunction cores. The state machine sets a calculation select value (calc_sel) which control muxes for the core's

inputs for current calculation. A counter (calc_wait) is used by the state machine to wait for the cores to complete their calculations.

*Enhanced Spectrum Algorithm*

Once the input RAM is filled with data from the LPC spectrogram, the state machine will begin the algorithm (from MATLAB):

```
% "Silence Factor" Computation
normz = 0;
for i = 1:SPECL
        normz = normz + temp_spec_vector(i);
end
normz = SF * normz + SIL_EN_FLOOR;
neighborhood_sum = zeros (SPECL,1);
% Computation of initial neighborhood sum for bin i=0
local_sum = temp_spec_vector(1);
for j = 2:HALF_NEIGHB_SIZE+1
        local_sum = local_sum + temp_spec_vector(j);
end
neighborhood_sum(1) = local_sum;
% Computing Neighborhood Sum X[j+i]
for i = 2:SPECL
```

```
        j = i + HALF_NEIGHB_SIZE;
        k = i - HALF_NEIGHB_SIZE-1;
% Handling edge effects
if j >= SPECL
        indx1 = SPECL;
else
        indx1 = j;
end
if (k<1)
        indx2 = 1;
else
        indx2 = k;
end
 % Adding New Element - Dropping Old one from local
local_sum=local_sum+temp_spec_vector(indx1)temp_spec_vector(ind
x2);
% Removing Center Element from local_sum
neighborhood_sum(i) = local_sum - temp_spec_vector(i);
end
 % Computing denominator
```

```
denom = zeros(SPECL,1);
for i=1:SPECL
    denom(i) = normz + NF * neighborhood_sum(i) + BG * background_estm(i);
end
% Scaling the output
out_spec_vector = zeros(SPECL,1);
for i=1:SPECL
    tmp = EG * (temp_spec_vector(i) / denom(i));
    out_spec_vector(i) = tmp * tmp;
end
```

Note: For the initial demonstration, the background noise estimate is considered to be zero, so it is not calculated by the HDL logic at this time. It will be added when the VAD logic is incorporated. The algorithm uses the following constants:

```
LPC_INDEX_MAX          = 8'h7F; // max. spectrum index (128 total)
HALF_NBRHOOD_MAX       = 8'h05;
```

```
SILENCE_FACTOR          = 32'h3C23D70A;  // 1.0e-2
SIL_EN_FLOOR            = 32'h501502F9;  // 1.0e10
NBRHOOD_FACTOR          = 32'h3C23D70A;  // 1.0e-2
ENHANCE_GAIN            = 32'h4CBEBC20;  // 1.0e8
```

States S_CALC_NORMZ_SUM and S_CALC_NORMZ_SF calculate the normz sum.

States S_INIT_LOCSUM0 and S_CALC_LOCSUM0 calculate the initial (0) local sum. States S_CALC_LOCSUM, S_CALC_LOCSUM_WAIT_INDEXH, S_CALC_LOCSUM_WAIT_INDEXL, S_CALC_LOCSUM_STAGE1 and S_CALC_LOCSUM_STAGE2 calculate the local_sum values. State S_CALC_DENOM calculates the denominator. State S_CALC_TEMP calculates the temp values and stores them in the temp RAM.

State S_SCALE_OUTPUT performs the final scaling of the temp data and writes the outputs out of the module via ENH_SPEC_DATA_out and ENH_SPEC_DV_out; State S_BACKFILL writes out 128 zero values to match the length of the normal and LPC spectrograms. *Testbench: tb_enh_spec.v*

The LPC module testbench writes the data from the file "simout_spec_lpc_data_small.txt" into the input of the enhanced spectrogram module (DUT). The testbench writes output data from the DUT into the file "simout_enh_spec_data.txt". The MATLAB script

"tb_enh_spec.m" reads the same input file and the simulation output file and compares and plots the results and any resulting error.

## 6.10 Voice Activity Detector (VAD)

Module Name: vad_module.v

Megafunction Cores : ALTFP_MULT (mult_float_32.v)

ALTFP_ADD_SUB (add_sub_float_32.v) x 2

ALTFP_LOG (log_float_32.v)

ALTFP_DIV (div_float_32.v)

ALTFP_COMPARE (fp_comp_lt_single.v)

ALTFP_CONV (int32_2_float.v)

RAM: 2-PORT (dpram_32x32bits.v)

RAM: 2-PORT (dpram_256x32bits.v)

The VAD module determines if the incoming data is voice content and sets an output flag accordingly. The VAD accepts LPC spectral and MFCC data, along with the frame energy to calculate three metrics which are used to set the VAD state output.

As with the other output modules, a state machine is used to process the incoming data using shared DSP resources. This method cuts down on the amount of DSP cores needed for processing. Two registers: calc_sel and

calc_wait are used, respectively, to select the DSP core calculation inputs and set the number of iterations needed to perform the selected calculation.

A 256 x 32-bit DPRAM is instantiated to store incoming LPC spectrum samples (LPC_SPEC_DATA_in, LPC_SPEC_DV_in) and a 32 x 32-bit DPRAM stores incoming LPC MFCC samples (MFCC_DATA_in, LPC_MFCC_DV_in). The state machine registers FRAME_ENERGY_in when all LPC spectrum samples have been stored. After reset, the state machine waits in state S_WAIT_FOR_LPC_SPEC_DATA for all LPC spectrum samples to be stored; it then registers the input frame energy and begins the log2 (frame_energy) calculation (states S_CALC_LOG2FE_STAGE1 and S_CALC_LOG2FE_STAGE2). The state machine will then wait for all LPC MFCC input samples to be stored (state S_WAIT_FOR_LPC_MFCC_DATA).

When all MFCC input samples have been received, the MFCC feature will be calculated in state S_CALC_MFCC_FEATURE. Next, the signal energy stats are calculated in states S_CALC_SIG_ENERGY_STATS through S_CALC_MEAN_LOG2_FRAME_ENERGY2. Two frame energy stats are calculated for VAD on and off. These values are used later in the algorithm to determine the future VAD state. States S_CALC_VAD_SPEC_STATS through S_CALC_VAD_SPEC_VAR calculate the sum, sum^2, mean and variance of the LPC spectrum for the spectrum stats. States

S_CALC_AVG_SPEC_VAR through S_CALC_AVG_SPEC_VAR2 calculate the LPC spectrum average variance. The spectrum average variance is also calculated for VAD on and off conditions and these values are used to determine the future VAD state. States S_CALC_MEAN_MFCC_STATS through S_CALC_MEAN_MFCC2 calculate the VAD MFCC feature. VAD on and off values for this feature are stored, similar to the frame energy and spectrum features. Each of the VAD features (frame energy, spectrum and MFCC) are now used to make a determination of the VAD state (on or off) based on that single feature. The frame energy VAD state is calculated in states S_SET_VAD_EN_STATE through S_SET_VAD_EN_STATE2, the spectrum VAD state in states S_SET_VAD_SPEC_STATE through S_SET_VAD_SPEC_STATE2 and finally the MFCC VAD state in states S_SET_VAD_MFCC_STATE through S_SET_VAD_MFCC_STATE2.

The final VAD state determination is made by calculating a score based on the weighted sum of the three VAD features. If the score is less than zero, the current frame is considered to contain speech energy and a VAD On Count is incremented. Otherwise a VAD OFF count is incremented. When the VAD ON count surpasses a set threshold, the final VAD state will be set to ON. Likewise, if the VAD OFF count exceeds a set threshold, the

VAD state will be set to OFF. These thresholds build in some hysteresis to prevent false VAD triggering.

States S_VAD_SCORE through S_VAD_SCORE_CALC2 calculate the VAD score. States S_VAD_DECISION_ON and S_VAD_DECISION_OFF set the final VAD state of the current frame and write the VAD state output for use at the backend processor. After final VAD state determination, the state machine will return to the idle state to wait for the next frame's data.

## 6.11 Hardware Front End Output vs. Software Front End Output

Because Wake-Up-Word Speech Recognition is a new concept, it is difficult to compare its performance with existing Speech Recognition Systems. In order to perform a fair analysis we tested the performance of our front-end system by comparing its spectrograms and features i.e. (MFCC, LPC, and ENH-MFCC) with the software (C, C++) WUW's front-end algorithm implementation, and with the MATLAB front-end model which is implemented specially for this reason.

The front-end processor described in this dissertation has been modeled in Verilog HDL and implemented in low cost, high speed, and power efficient (Cyclone III EP3C120F780C7) FPGA on DSP development kit. The development of the front-end was conducted block by block based on

software (C, C++) algorithm implementation and on equivalent floating-point MATLAB implementation. Each block was tested after it was completed to ensure correct operation before the next block was developed. The words “Onward” and “Voyager” with 8KHz sampling rate was chosen as input audio data for testing our Front-end; we tested and compared (MFCC, LPC, and ENH-MFCC) spectrograms and features out of the hardware front-end model with the MATLAB front-end model and the software (C, C++) front-end model as individual models. The results show:

1. For this test we chose the word “Onward” with 8 KHz sampling rate as input audio data for testing our Front-end. As shown in Fig. 6.8, 6.9, and 6.10, the MFCC, LPC, and ENH-MFCC spectrograms generated from MATLAB, Hardware, and Software (C++) are identical.

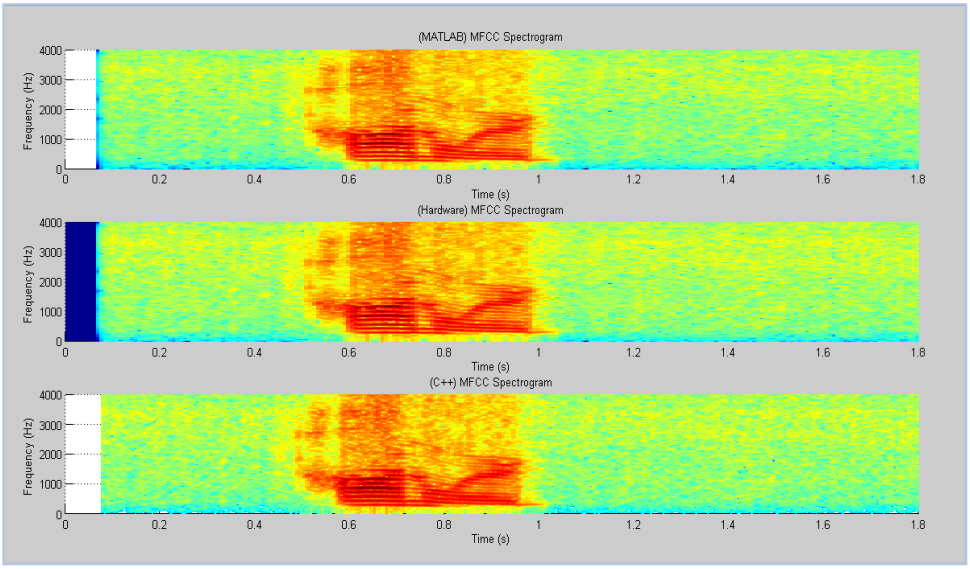

**Figure 6.8 – MATLAB, Hardware, and C++ Front-end MFCC Spectrograms for "Onward" Audio Data**

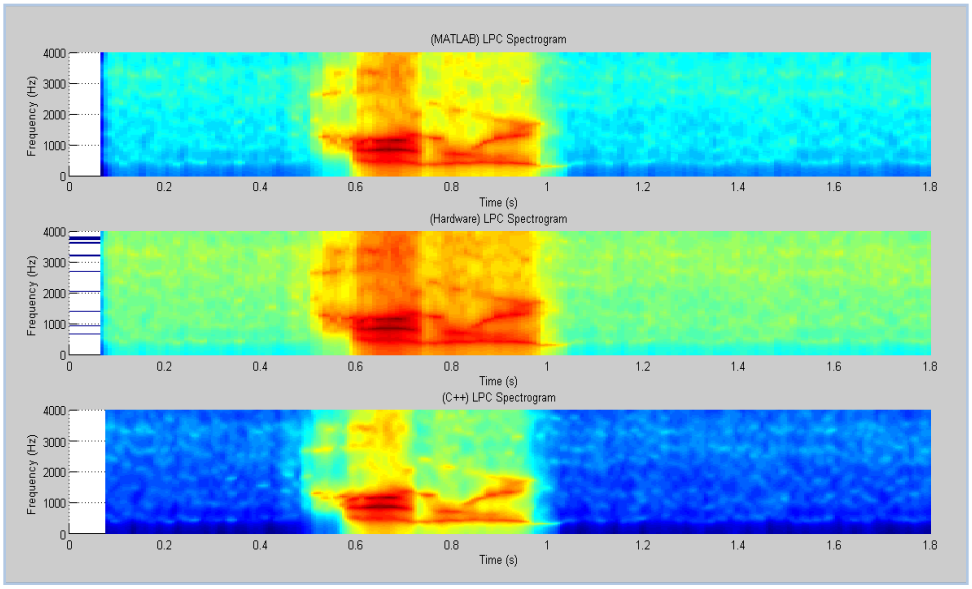


**Figure 6.9 – MATLAB, Hardware, and C++ Front-end LPC Spectrograms for "Onward" Audio Data**

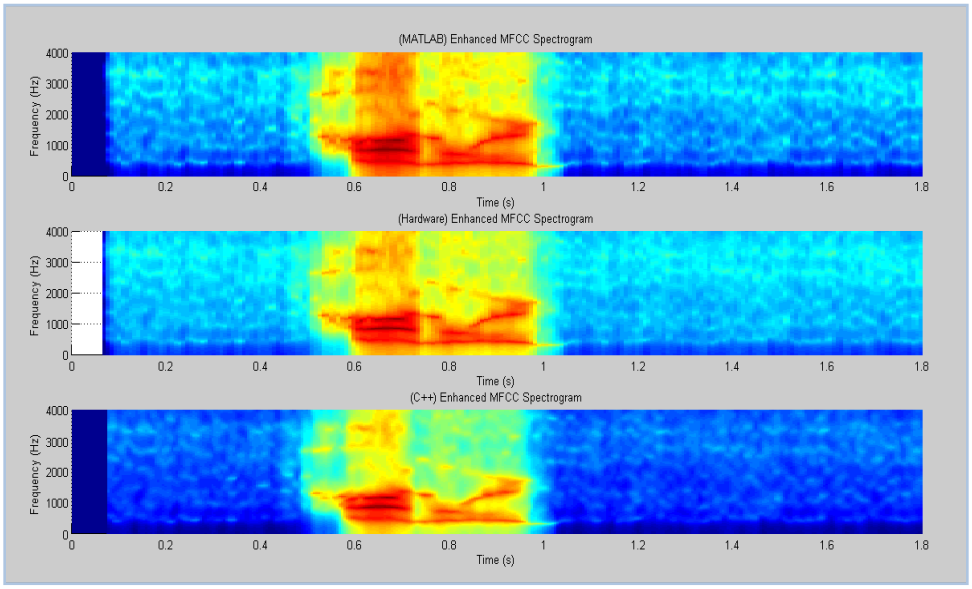


**Figure 6.10 – MATLAB, Hardware, and C++ Front-end Enhanced MFCC Spectrograms for "Onward" Audio Data**

In Fig. 6.11, we generated the "Onward" audio signal with the software (C++) front-end spectrograms that shows are identical with the Hardware front-end spectrograms shown in Fig. 6.12.

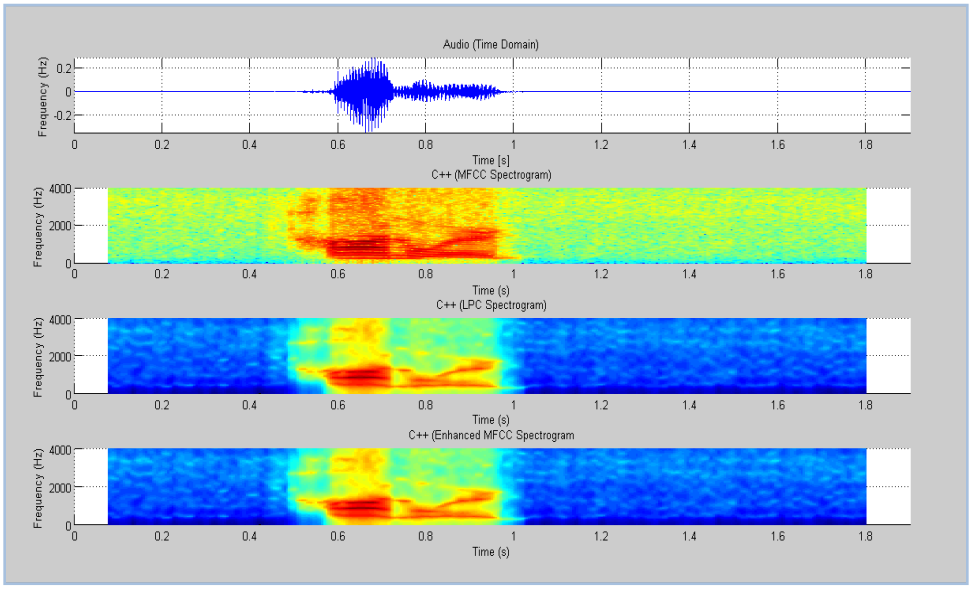


**Figure 6.11 – C++ Front-end MFCC, LPC, and Enhanced MFCC Spectrograms for "Onword" Audio Data**

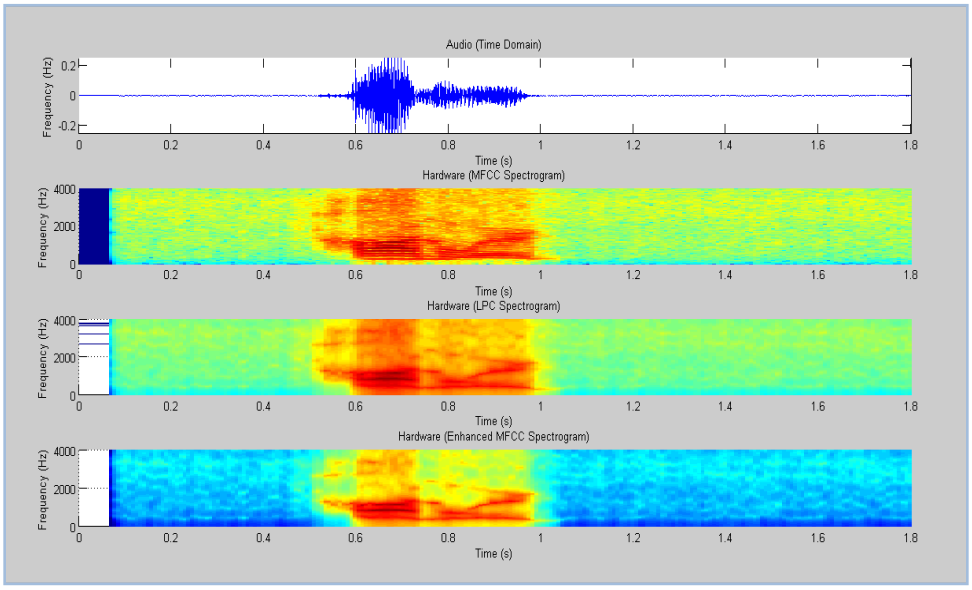


**Figure 6.12 – Hardware Front-end MFCC, LPC, and Enhanced MFCC Spectrograms for "Onword" Audio Data**

2. In the second test we chose the word “Voyager” with 8 KHz sampling rate as input audio data for testing our Front-end. As shown in Fig. 6.13, 6.14 the MFCC, LPC, and ENH-MFCC spectrograms generated from Hardware and Software (C++) are identical.

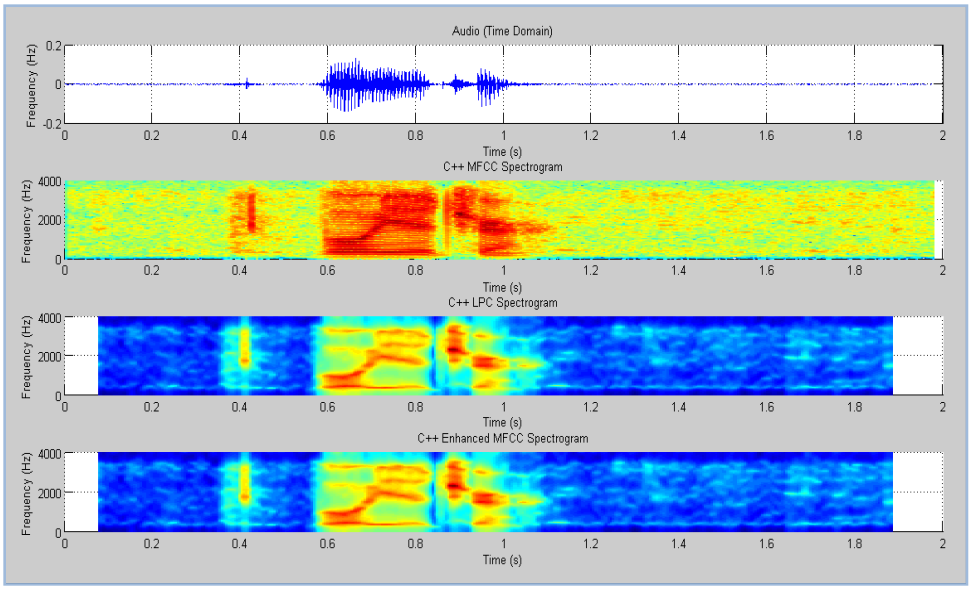


**Figure 6.13 – C++ Front-end MFCC, LPC, and Enhanced MFCC Spectrograms for “Voyager” Audio Data**

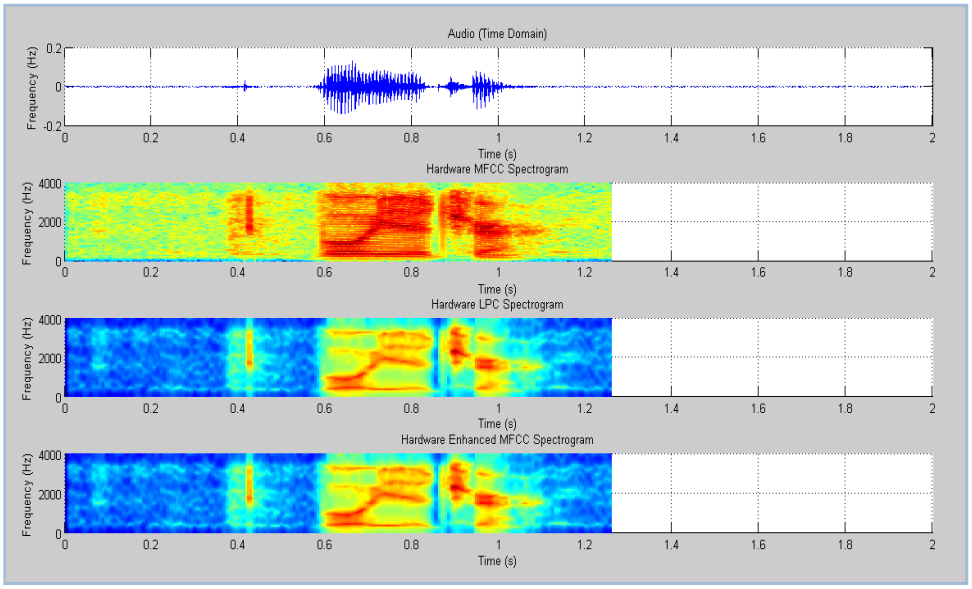


**Figure 6.14 – Hardware Front-end MFCC, LPC, and Enhanced MFCC Spectrograms for "Voyager" Audio Data. (Due to limited amount of hardware resources the part of the data is not show in the resulting spectrograms).**

In Fig. 6.15, 6.16 the MFCC, LPC, and ENH-MFCC 12-features histograms generated from Hardware and Software (C++) are also identical.

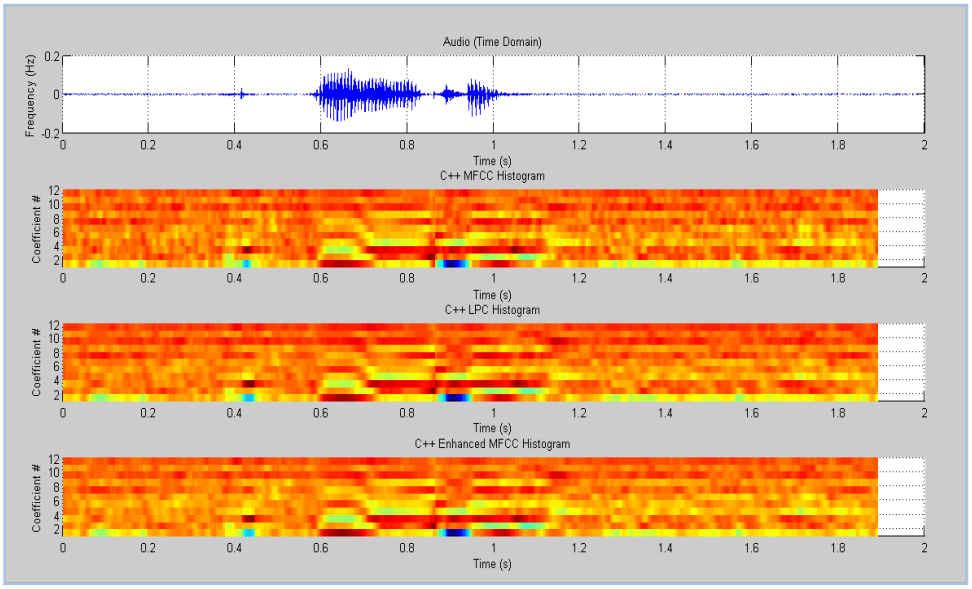


**Figure 6.15 – C++ Front-end MFCC, LPC, and Enhanced MFCC Histograms for "Voyager" Audio Data (12- Coefficients)**

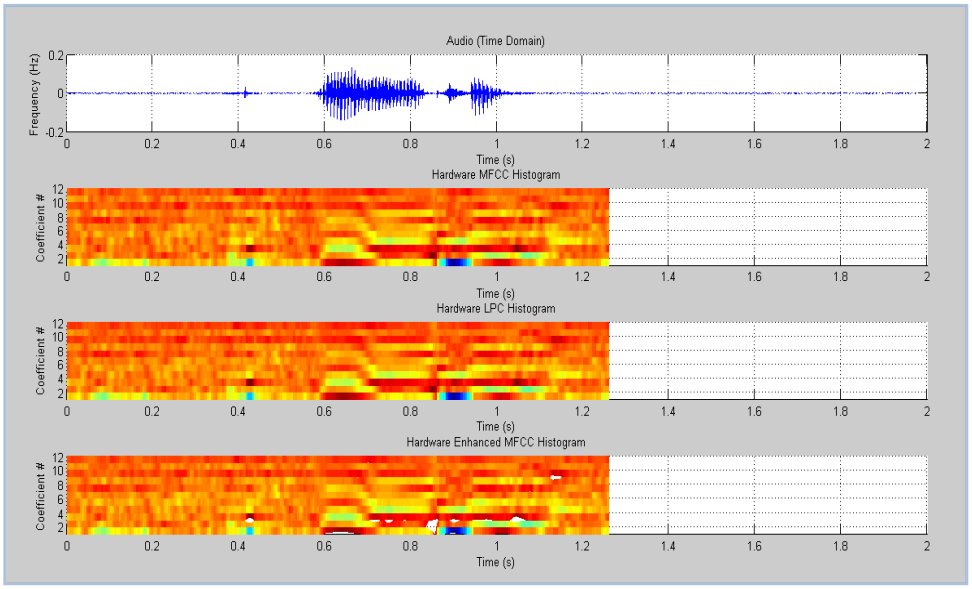


**Figure 6.16 – Hardware Front-end MFCC, LPC, and Enhanced MFCC Histograms for "Voyager" Audio Data (12- Coefficients). (Due to limited amount of hardware resources the part of the data is not show in the resulting Histograms).**

We regenerated new MFCC, LPC, and ENH-MFCC histograms with 11-features by removing the first feature slice because of large dynamic range of the first feature that would make the remaining output features very small.

As shown in Fig. 6.17, 6.18 the MFCC, LPC, and ENH-MFCC 11-features histograms generated from Hardware and Software (C++) are identical.

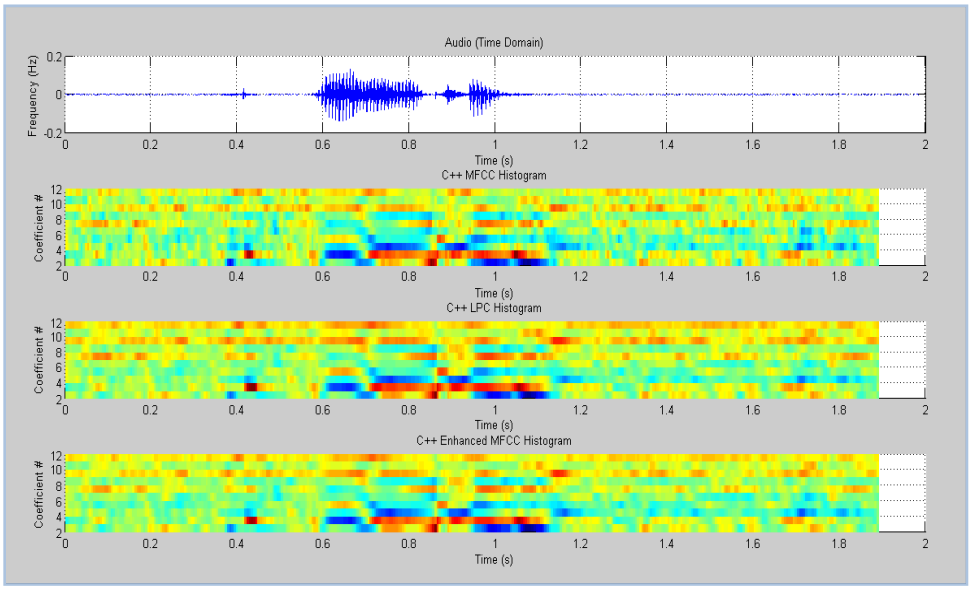


**Figure 6.17 – C++ Front-end MFCC, LPC, and Enhanced MFCC Histograms for "Voyager" Audio Data (11- Coefficients C2-C12)**

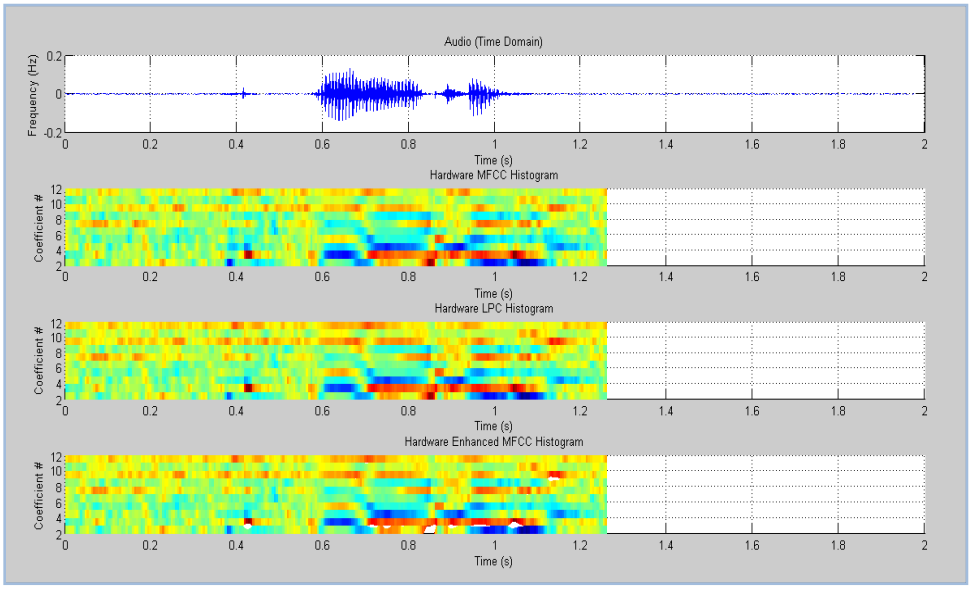


**Figure 6.18 – Hardware Front-end MFCC, LPC, and Enhanced MFCC Histograms for "Voyager" Audio Data (11- Coefficients C2-C12). (Due to limited amount of hardware resources the part of the data is not show in the resulting Histograms)**

3. In the third test we chose the word "Operator" with 8 KHz sampling rate as input audio data for testing our Front-end with VAD built-in. As shown in Fig. 6.19, 6.20 the MFCC, LPC, and ENH-MFCC spectrograms generated from Hardware and Software (C++) are identical.

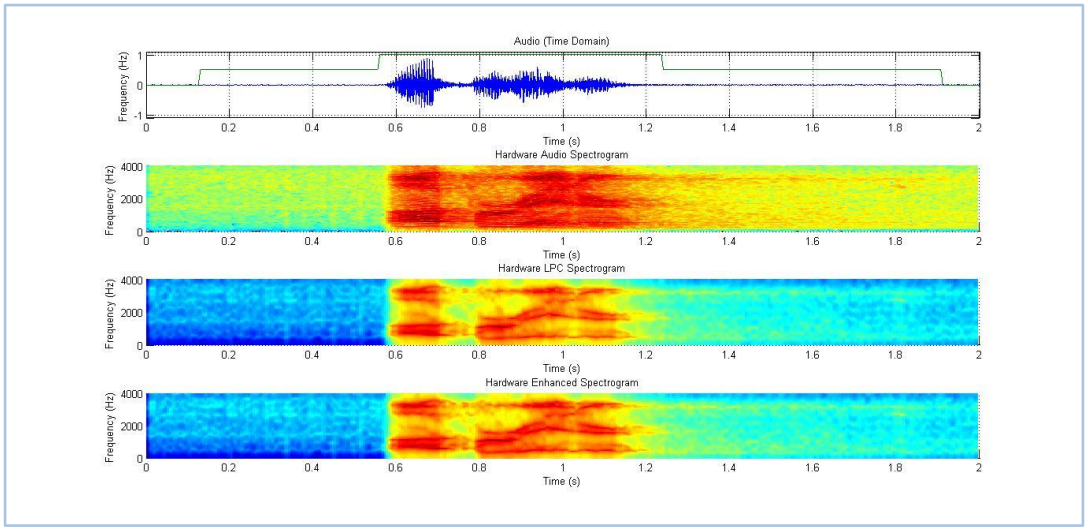


**Figure 6.19 – Hardware Front-end with VAD MFCC, LPC, and Enhanced MFCC Spectrograms for "Operator"**

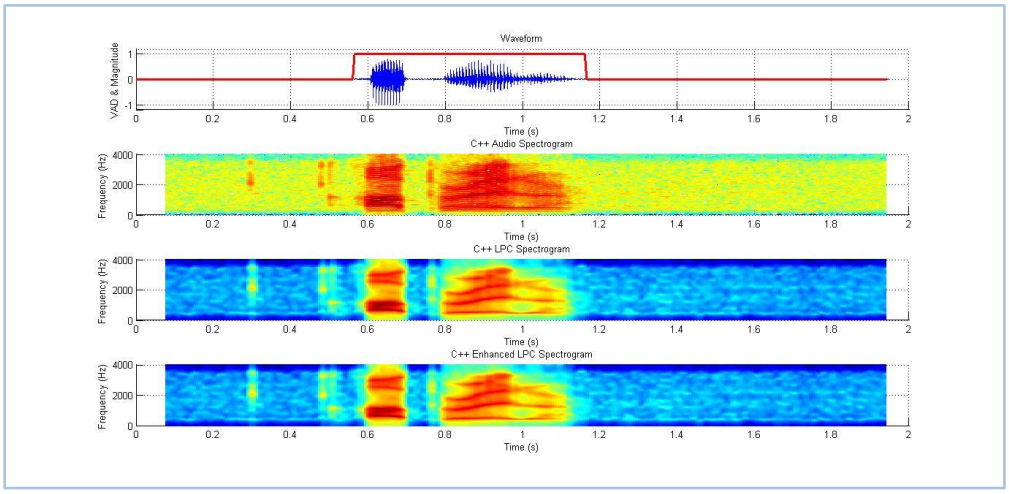


**Figure 6.20 – C++ Front-end with VAD MFCC, LPC, and Enhanced MFCC Spectrograms for "Operator" Audio Data**

In Fig. 6.21, 6.22 the MFCC, LPC, and ENH-MFCC 12-features histograms generated from Hardware and Software (C++) are also identical.

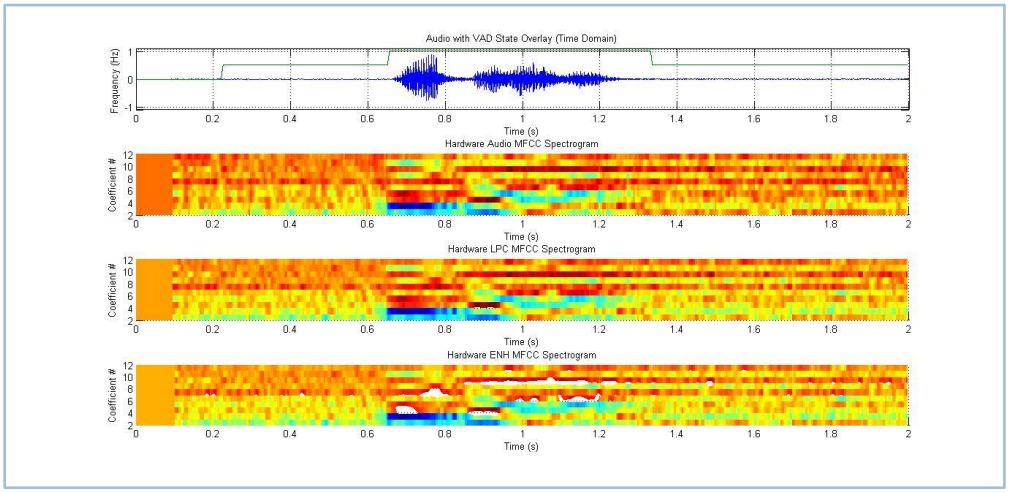


**Figure 6.21 – Hardware Front-end with VAD MFCC, LPC, and Enhanced MFCC Histograms for "Operator" Audio Data**

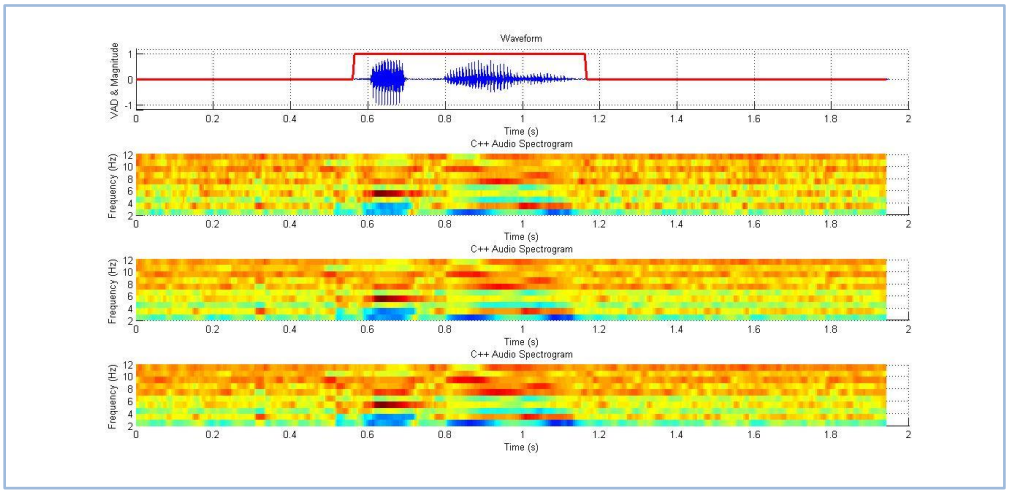


**Figure 6.22 – C++ Front-end with VAD MFCC, LPC, and Enhanced MFCC Histograms for "Operator" Audio Data**

# 7 Conclusions

In this dissertation, the efficient hardware architecture and implementation of front-end of WUW-SR has been presented. We have described relevant parts of front-end from theory to hardware design. Details have been given of how these designs were implemented on FPGA, and the results of these implementations analyzed, which included comparing them with software equivalents. WUW front-end is responsible for generating three sets of features MFCC, LPC, and ENH-MFCC. These features are needed to be decoded with corresponding HMMs in the back-end stage of the WUW Speech Recognizer (e.g., server). The computational complexity and memory requirement of these sets of feature algorithms is analyzed in detail and showed significant identical with software features. The partitioned table look-up method is adopted and modified to be suitable in

our case with very small table memory requirement. The overall performance and area is highly improved. Presented front-end of Wake-Up-Word Speech Recognition is a novel solution which is the first hardware system specifically designed for WUW-SR based on three different sets of features. The most important characteristic of a WUW-SR system is that it should guarantee virtually 100% correct rejection of non-WUW (out of vocabulary words - OOV) while maintaining correct acceptance rate of 99% or higher (in vocabulary words - INV). This requirement sets apart WUW-SR from other speech recognition systems because no existing system can guarantee 100% reliability by any measure. To demonstrate its effectiveness, the presented design has been implemented in cyclone III FPGA hardware. The custom DSP board developed is a power efficient, flexible design and can also be used as a general purpose prototype board. We have presented a detailed breakdown of front-end system into modules and subsystems with initial sketches of block diagrams for these modules and subsystems. We have also implemented a prototype of our design modules algorithm in MATLAB and demonstrated its operation on simple test cases.

Finally, we implemented test bench file for every module to perform simulation, we have identified several points of improvement to our initial naive algorithm, so the prospect of implementing a successful front-end

system in FPGA seems tenable. While software or DSPs have been sufficient to deal with the WUW speech pre-processing, and although software is well-suited to the post-decoding dictionary look-up, future research work may also want to look at the post-processing stage, towards the goal of a complete integrated Wake-Up-Word Speech Recognition System on programmable chip.

## References


[1] V. Z. Këpuska and T. B. Klein, A novel Wake-Up-Word speech recognition system, Wake-Up-Word recognition task, technology and evaluation, Nonlin. Anal.: Theory Meth. Appl. 71, pp. e2772–e2789, 2009. doi:10.1016/j. na. 2009.06.089.

[2] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, Spoken Language Processing: A Guide to Theory, Algorithm and System Development, Prentice Hall PTR, 2001.

[3] Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, Victor Zue (Eds.), Survey of the State of the Art in Human Language Technology, Cambridge University Press and Giardini, 1997.

[4] V. Këpuska, Wake-Up-Word Application for First Responder Communication Enhancement, SPIE, Orlando, 2006.

[5] T. Klein, Triple scoring of hidden markov models in wake-up-word speech recognition, Thesis, Florida Institute of Technology.

[6] V. Këpuska, Dynamic time warping (DTW) using frequency distributed

distance measures, US Patent: 6983246, January 3, 2006.

[7] V. Këpuska, Scoring and rescoring dynamic time warping of speech, US Patent: 7085717, April 1, 2006.

[8] V. Këpuska, T. Klein, On Wake-Up-Word speech recognition task, technology, and evaluation results against HTK and Microsoft SDK 5.1, Invited Paper: World Congress on Nonlinear Analysts, Orlando 2008.

[9] V. Këpuska, D.S. Carstens, R. Wallace, Leading and trailing silence in Wake-Up-Word speech recognition, in: Proceedings of the International Conference: Industry, Engineering & Management Systems 2006, Cocoa Beach, FL., 259_266.

[10] J.R. Rohlicek, W. Russell, S. Roukos, H. Gish, Continuous hidden Markov modeling for speaker-independent word spotting, vol. 1, 23_26 May 1989, pp.
627_630.

[11] C. Myers, L. Rabiner, A. Rosenberg, An investigation of the use of dynamic time warping for word spotting and connected speech recognition, in: ICASSP '80. vol. 5, Apr 1980, pp. 173_177.

[12] A. Garcia, H. Gish, Keyword spotting of arbitrary words using minimal speech resources, in: ICASSP 2006, vol. 1, 14_19 May 2006, pp. I_I.

[13] D.A. James, S.J. Young, A fast lattice-based approach to vocabulary independent wordspotting, in: Proc. ICASSP '94, vol. 1, 1994, pp. 377_380.

[14] S.P. Davis, P. Mermelstein, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences, IEEE

Trans. ASSP 28 (1980) 357_366.

[15] John Makhoul, Linear prediction: A tutorial review, Proc. IEEE. 63 (1975) 4.

[16] Lee, L.S., Tseng, C.Y., Lin, Y.H., Lee, Y., Tu, S.L., Gu, H.Y., Liu, F.H., Chang, C.H., Hsieh, S.H., Chen, C.H. & Huang, K.R., "A fully parallel Mandarin speech recognition system with very large vocabulary and almost unlimited texts," *Proc. IEEE International Symposium on Circuits and Systems*, 1991, pp.578–581.

[17] Altera, Inc. http://www.altera.com/.

[18] Melnikoff, S.J., "Speech recognition in programmable logic," PhD Thesis, University of Birmingham, 2003.

[19] Makimoto, T., "The rising wave of field programmability," *Proc. 10th International Conference on Field Programmable Logic and Applications (FPL 2000)*, *Lecture Notes in Computer Science #1896*, 2000, pp.1–6.

[20] Makimoto, T., "The hot decade of field programmable technologies," *IEEE International Conference on Field Programmable Technology (FPT 2002)*,2002, pp.3-6.

[21] Laufer, R., Taylor, R.R. & Schmit, H., "PCI-PipeRench and the SWORDAPI: a system for stream-based reconfigurable computing," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '99)*, 1999, pp.200–208.

[22] Sezer, S., Heron, J., Woods, R., Turner, R. & Marshall, A., "Fast partial reconfiguration for FCCMs," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, 1998, pp.318–319.

[23] James-Roxby, P. & Blodget, B., "Adapting constant multipliers in a neural network implementation," *Proc. IEEE Symposium on FPGAs for Custom*

*Computing Machines (FCCM 2000)*, 2000, pp.335–336.

[24] IEE, "FPGAs not ready to go embedded," *IEE Review*, Institution of Electrical Engineers, April 2003.

[25] G´omez-Cipriano, J.L., Pizzatto Nunes, R., Bampi, S. & Barone, D. "Design of functional blocks for a speech recognition portable system," *Proc. 14th Symposium on Integrated Circuits and Systems Design (SBCCI '01)*, 2001, pp.20–25.

[26] Shozakai, M., "Speech interface VLSI for car applications", *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '99)*, 1999, pp.141–144.

[27] Nakamura, K., Zhu, Q., Maruoka, S., Horiyama, T., Kimura, S. & Watanabe, K.,"Speech recognition chip for monosyllables," *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC 2001)*, 2001, pp.396–399.

[28] Shi, Y.Y., Liu, J. & Liu, R.S., "Single-chip speech recognition system based on 8051 microcontroller core," *IEEE Transactions on Consumer Electronics*, 4**7**, No.1, 2001, pp.149–153.

[29] Schmit, H. & Thomas, D., "Hidden Markov modeling and fuzzy controllers in FPGAs," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '95)*, 1995, pp.214-221.

[30] Vargas, F.L., Fagundes, R.D.R. & Junior, D.B., "A FPGA-based Viterbi algorithm implementation for speech recognition systems," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*, 2001, pp.1217–1220.

[31] Jou, J.M., Shiau, Y.H. & Huang, C.J., "An efficient VLSI architecture for

HMMbased speech recognition," *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS '01)*, 2001, pp.469–472.

[32] Stogiannos, P., Dollas, A. & Digalakis, V., "A configurable logic based architecture for real-time continuous speech recognition using hidden Markov models," *Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, 24, No.2–3, 2000, pp.223–240.

[33] Stogiannos, P., "FCCM coprocessor for real-time continuous speech recognition," Masters Thesis, Microprocessor and Hardware Laboratory, Department of Electronic and Computer Engineering, Technical University of Crete, 1999.

[34] Sensory, Inc. http://www.sensoryinc.com/

[35] Philips Speech Processing http://www.speech.philips.com/

[36] Frostad, K., "The state of embedded speech," *Speech Technology Magazine*, Mar/Apr 2003 *and* http://www.speechtechmag.com/.

[37] Mozer, T., "The third wave: speech in consumer electronics," *Speech Technology Magazine*, Jul/Aug 2000 *and* http://www.speechtechmag.com/.

[38] Telecom Italia Lab (TILAB): System on Chip. http://www.idosoc.com/

[39] Xilinx, Inc. http://www.xilinx.com/

[40] Texas Instruments DSP Developers' Village

http://dspvillage.ti.com/

[41] Cox, S.J., "Hidden Markov models for automatic speech recognition: theory and application," *British Telecom Technology Journal*, **6**, No.2, 1988, pp.105–115.

[42] Eldredge, J.G. & Hutchings, B.L., "RRANN: a hardware implementation of the back-propagation algorithm using reconfigurable FPGAs," *Proc. IEEE International Conference on Neural Networks / IEEE World Conference on Computational Intelligence*, **4**, 1994, pp.2097–2102.

[43] Chen, R. & Jamieson, L.H., "Experiments on the implementation of recurrent neural networks for speech phone recognition," *Proc. 30th Annual Asilomar Conference on Signals, Systems and Computers*, 1996, pp.779–782.

[44] Bohez, E.L.J. & Senevirathne, T.R., "Speech recognition using fractals," *Pattern Recognition*, **34**, No.11, 2001, pp.2227–2243.

[45] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[46] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 357–366, Aug 1980.

[47] Ngoc-Vinh Vu, Jim Whittington, Hua Ye, and John C. Devlin "Implementation of the MFCC front-end for low-cost speech recognition systems". by: In: ***ISCAS***IEEE (2010) , p. 2334-2337.

[48] J.-C. Wang, J.-F. Wang, and Y.-S. Weng, "Chip design of mfcc extraction for speech recognition," *Integr. VLSI J.*, vol. 32, no. 1-3, pp. 111–131, 2002.

[49] W. Han, C.-F. Chan, C.-S. Choy, and K.-P. Pun, "An efficient mfcc extraction

method in speech recognition," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, 0-0 2006, pp. 4 pp.–.

[50] Nedevschi, S., Patra, R., Brewer, E., "Hardware Speech Recognition for User Interfaces in Low cost, Low Power Devices," 43nd Design Automation Conference, IEEE Press, California, June 2005, pp.684-689.

[51] Melnikoff, S., Quigley, S.F., Rusell, M. J., "Implementing a Simple Continuous Speech Recognition System on an FPGA," Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, California, USA (2002).

[52] Staworko, M.; Rawski, M.; , "FPGA implementation of feature extraction algorithm for speaker verification," *Mixed Design of Integrated Circuits and Systems (MIXDES), 2010 Proceedings of the 17th International Conference* , vol., no., pp.557-561, 24-26 June 2010

[53] D. A. Reynolds. Experimental Evaluation of Features for Robust Speaker Identification. IEEE Trans. Speech and Audio Processing, 2(4):639–643, 1994.

[54] A.Kaczmarek, M.Staworko, Application of Dynamic Timer Warping and Cepstrograms to Text-Dependent Speaker Verification, Signal Processing Algorithms, Architectures, Arrangements and Applications, 24-26 September, Poznan, Poland.

[55] Charbuillet, C., Gas, B., Chetouani, M., Zarader J.L.: Optimizing feature complementarily by evolution strategy: Application to automatic speaker verification. Speech Communication, Vol. 51, No. 9, September, 2009, pp. 724-731.

[56] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” IEEE Trans. Acoust., Speech, Signal Processing, vol. 28, no. 4, pp. 357–366, 1980.

[57] H. Combrinck and E. Botha, “On the mel-scaled cepstrum,” department of Electrical and Electronic Engineering, University of Pretoria.

[58] M. R. Schroeder, “Linear prediction, extremely entropy and prior Information in speech signal analysis and synthesis,” Speech Communication, vol. 1, no. 1, pp. 9–20, May 1982.

[59] K. K. Paliwal and W. B. Kleijn, Speech Synthesis and Coding, chapter Quantization of LPC parameters, pp. 433–466, Elsevier Science Publ., Amsterdam, the Netherlands, 1995.

[60] Texas Instruments
http://www.ti.com/

[61] Altera, Design Software
http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html

[62] Mentor Graphic, ModelSim Software
http://www.mentor.com/products/fv/modelsim/

[63] http://en.wikipedia.org/wiki/Quantization_(sound_processing) #Audio_ quantization.

[64] M. R. Schroeder, “Linear prediction, extremely entropy and prior Information in speech signal analysis and synthesis,” *Speech Communication*, vol. 1, no. 1, pp. 9–20, May 1982.

[65] K. K. Paliwal and W. B. Kleijn, *Speech Synthesis and Coding*, chapter Quantization of LPC parameters, pp. 433–466, Elsevier Science Publ., Amsterdam, the Netherlands, 1995.

## Appendix: Publications

**Journals:**

Këpuska VZ, Eljhani MM, Hight BH (2013) Front-end of Wake-Up-Word Speech Recognition System Design on FPGA. J Telecommun Syst Manage 2:108. doi:10.4172/2167-0919.1000108.

V. Këpuska, M. Eljhani and B. Hight, "Wake-Up-Word Feature Extraction on FPGA," World Journal of Engineering and Technology, Vol. 2 No. 1, 2014, pp. 1-12. doi: 10.4236/wjet.2014.21001.

**Books:**