Speech Recognition on DSP: Algorithm Optimization and Performance Analysis

YUAN Meng

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Electronic Engineering

July 2004

Acknowledgements

I would like to thank my supervisor, Prof. P. C. Ching, for his guidance and support throughout my research work. I would also like to express my gratitude to my co-supervisor, Prof. Tan Lee, for providing a lot of useful technical discussions over the course of my research. I would also want to express my sincere appreciation to Prof. Y. T. Chan, Prof. Xianggen Xia, Prof. Frank K. Soong and Mr. H. S. Lam for their helpful suggestions and support.

I would like to remember all the colleagues and friends who shared with me lots of happiness during my study. To name only some of them: Miss Y. C. Chan, Mr. B. R. Chen and his wife Ms. H. Shen, Mr. K. Chen, Miss P. Kam, Miss K. Y. Kwan, Miss S. W. Lee, Miss Y. J. Li, Dr. W. K. Lo, Dr. W. K. Ma, Miss L.Y. Ngan, Mr. S. L. Oey, Ms. Y. Qian, Mr. C. Qin, Miss Y. Y. Tam, Miss S. Y. Tao, Miss C. Yang, Mr. C. H. Yau, Mr. W. Zhang, Mr. W. Zhang (Michael), Mr. N. H. Zheng and Miss Y. Zhu, and Mr. K. O. Luk for his technical support.

I wish to express my deepest gratitude to my parents, my family and Miss S. M. Wang for their disciplines and continuous encouragement.

Abstract of thesis entitled:

Speech Recognition on DSP:

Algorithm Optimization and Performance Analysis

Submitted by **YUAN MENG**

for the degree of **Master of Philosophy**

in **Electronic Engineering**

at **The Chinese University of Hong Kong** in

July 2004.

This thesis describes the exploitation of state-of-the-art automatic speech recognition (ASR) techniques for DSP-based embedded applications. Automatic speech recognition technology has advanced rapidly in the past decades. While many ASR applications employ powerful computers to handle the complex recognition algorithms, there is a clear demand for effective solutions on embedded platforms. Digital Signal Processor (DSP) is one of the most commonly used hardware platforms that provides good development flexibility and requires relatively short application development cycle.

Implementation of speech recognition algorithms generally involves a lot of floating-point arithmetic operations and manipulation of complex functions, which are not desirable for low-cost DSP hardware. Algorithm optimization is therefore necessary to remove such undesirable operations as far as possible. In this thesis, an isolated word recognition system (IWR) and a connected word recognition system (CWR) have been developed. More importantly, we present a thorough understanding and analysis for individual computational steps in the complex ASR algorithms. The most computationally intensive parts are identified and a set of appropriate optimization methods such as numerical quantization, simplification and approximation are applied to facilitate real-time processing on a specific DSP platform. Experimental results show that the optimized recognition algorithms can be run about three times faster than

real time with slight degradation of recognition accuracy. We have also investigated and implemented two practical techniques, namely end-point detection (EPD) and spectral subtraction (SS), which are important for the ASR system to operate in real-world environments. Experimental results show that these techniques can not only satisfy the real-time requirement, but also enhance the recognition accuracy.

With the detailed algorithm analysis and experiments, the relation between computational efficiency and recognition performance has been studied extensively. This research can serve as a useful reference for the engineers with general DSP background to design ASR applications that meet their specific requirements on recognition accuracy and hardware resources.

摘要

本文深入探討了自動語音識別新技術 (ASR) 在基於嵌入式 DSP 系統中的應用。自動語音識別技術在過去的幾十年間得到了快速的發展。儘管大多數 ASR 應用都採用具有強大運算能力及資源的計算機作爲開發平台以便處理複雜的識別算法，但是人們仍舊有將識別算法應用於嵌入式系統的想法和需求。數字信號處理器(DSP)，就是當今最流行的嵌入式平台之一。它以其良好的開發靈活性及相對較短開發周期在嵌入式領域中佔有舉足輕重的地位。

實現語音識別算法通常需要大量的浮點運算，而且要用到很多複雜的函數。這對於低功耗的 DSP 硬件來說是難以實現的。因此，我們需要優化算法來盡量減少不合需要的運算。我們不僅開發了兩個獨立的語音識別系統，孤立詞語音識別系統 (IWR) 和連接詞語音識別系統 (CWR)。更重要的是，我們全面地、徹底地分析了複雜的語音識別算法中各個獨立的計算單元，並且指出了其中計算量最大的單元。對於這些單元，我們採用了不同的優化方法來降低計算量，以實現在特定 DSP 平台上的實時處理。這些方法大致上包括數值量化、簡化及近似。實驗結果表明，通過算法優化，整個處理時間可以被縮減到允許實時處理最長時間的三分之一，而識別率只有微小的下降。我們還研究及實現了兩種實用技術，端點監測(EPD)和譜減法(SS)。這兩種方法對於真實環境中的自動語音識別是十分重要的。實驗結果表明，這兩种技術不僅可以滿足實時處理要求，同時也提高了識別率。

通過詳細的算法分析及完整的實驗，我們清晰地揭示了計算有效性與識別性能之間的內在關係。同時，本研究也可以爲那些嘗試在特定硬件系統上實現自動語音識別應用的工程師提供有效的幫助和參考。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Throughout the human history, speech has been the most dominant and convenient means of communication between people. Today, speech communication is not only for face-to-face interaction, but also between individuals at any moment, anywhere, via a wide variety of modern technological media, such as wired and wireless telephony, voice mail, satellite communications and the Internet. With the rapid development of communication technologies, a promising speech communication technique for human-to-machine interaction has come into being. Automatic speech recognition (ASR) is the core challenge towards the natural human-to-machine communication technology [1].

Automatic speech recognition aims to automatically convert a speech waveform into a sequence of words by machine [2]. Currently, there have been a number of successful commercial ASR products ([3] [4] [5] [6]). However, many problems still exist in real-world ASR applications. The recognition accuracy of a machine is, in most cases, far from that of a human listener, and its performance would degrade dramatically with small modification of speech signals or speaking environment. Due to the large variation of speech signals, speech recognition inevitably requires complex algorithms to represent this variability. As a result more computation and memory capacity are needed. But for real-world applications, such as embedded ASR systems, only limited resources are available. The goal of this research is focused on the exploitation of various optimization methods for state-of-the-art ASR techniques for DSP-based

embedded applications while retaining high recognition accuracy.

1.1 History of ASR development

The earliest attempts to implement automatic speech recognition on machine began in the 1950s. The first significant ASR system was built at Bell Labs in 1952 by Davis, Biddulph and Balashek for isolated digit recognition [7]. In this system, only one single speaker can be recognized based on measuring the spectral resonances during the vowel region of each digit.

In the 1960s and 1970s, a great deal of fundamental ideas in speech recognition emerged and were published. These new techniques include fast Fourier transform (FFT) [8], cepstral analysis [9], and linear predictive coding (LPC), [10] [11] [12] for spectral estimation and feature extraction, as well as dynamic time warping (DTW) [13] and Hidden Markov Models (HMMs) [14] [15] for pattern matching and acoustic modeling.

While isolated word recognition was investigated in the 1970s, the problem of connected word recognition was the focus in the 1980s. In addition, the approach of pattern recognition shifted from template-based to statistical modelling methods. In particular, the HMM approach was researched by Bell Labs [16], CMU [17], IBM [18],etc.

Since 1990s, people have moved their interests to the difficult task of Large Vocabulary Continuous Speech Recognition (LVCSR) and indeed achieved a great progress. Speech recognition has been developed from theoretical methods to practical systems. Meanwhile, many well-known research and commercial institutes have established their recognition systems including ViaVoice system by IBM, HTK system by the University of Cambridge and Whisper system by Microsoft, etc. [2].

1.2 Fundamentals of automatic speech recognition

1.2.1 Classification of ASR systems

A speech recognition system can operate in many different conditions such as speaker dependent/independent, isolated/continuous speech recognition, for small/large vocabulary [19].

Speaker-dependent versus independent system

A speaker-dependent system is a system that recognizes a specific speaker's speech while speaker-independent systems can be used by any unspecified speaker.

Isolated versus continuous speech recognition system

In an isolated word recognition system, each word (note that word may be a simple utterance) is assumed to be surrounded by silence or background noise. This means that both sides of a word must have no speech input, making definite word boundaries easy to construct. This kind of recognition is mainly used in applications where only a specific digit or a word needs to be dictated.

Connected speech (or more correctly 'connected utterances') recognition is similar to isolated word recognition. But it allows several words/digits to be spoken together with minimal pause between them. Longer phrases or utterances are therefore possible to be recognized.

Continuous speech recognition is much more natural and user-friendly. The system is able to recognize a sequence of connected words, which are not separated by pauses, in a sentence. This mode requires much more computation time and memory, and it is more difficult to operate than isolated word recognition mode for the following reasons:

- Speakers' pronunciation is less careful;

- Speaking rate is less constant;
- Word boundaries are not necessarily clear.

Therefore the accuracy of continuous speech recognition is usually low compared with the preceding modes.

Small versus large vocabulary size

It is clear that the smaller the vocabulary size, the higher the recognition accuracy. Usually we classify the difficulty level of implementing a speech recognition system with a score from 1 to 10 according to Table 1.1, where 1 means the simplest system (speaker-dependent, able to recognize isolated words in a small vocabulary (10 words)) and 10 correspond to the most difficult task (speaker-independent continuous speech over a large vocabulary (say, 10,000 words)). State-of-the-art speech recognition systems with acceptable error rates are somewhere between these two extremes [20].

	Isolated Word	Continuous Word
Speaker Dependent	Small Voc 1	Small Voc 5
	Large Voc 4	Large Voc 7
Multi-Speaker	Small Voc 2	Small Voc 6
	Large Voc 4	Large Voc 7
Speaker Independent	Small Voc 3	Small Voc 8
	Large Voc 5	Large Voc 10

Table 1.1: Classification of speech recognition difficulties

1.2.2 Automatic speech recognition process

Fig. 1.1 illustrates the architecture of a typical speech recognition system that employs today's main-stream approach. The acoustic models represent the

acoustic properties, phonetic properties, microphone and environmental variability, as well as gender and dialectal differences among speakers. The language models contain the syntax, semantics, and pragmatics knowledge for the intended recognition task. These models can be dynamically modified according to the characteristics of the previously recognized speech. This is referred to as model adaptation.

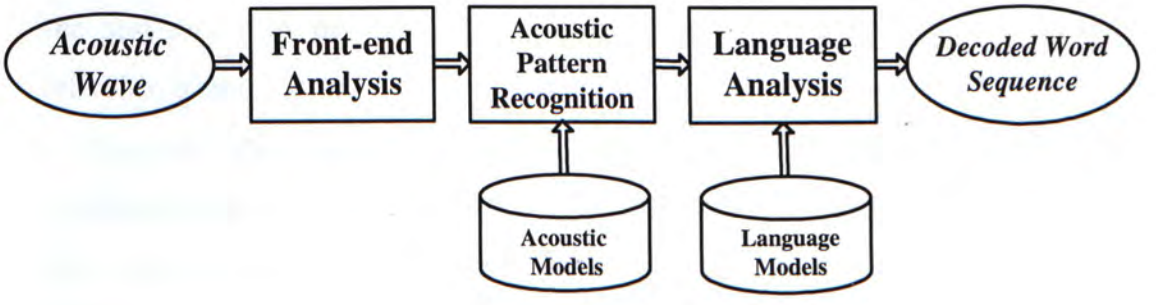


Figure 1.1: General block diagram of an automatic speech recognition system

Front-end analysis

Front-end analysis, also referred to as acoustic analysis or feature extraction, is the first step in an automatic speech recognition system. This process aims to extract acoustic features from the speech waveform. The output of front-end analysis is a compact, efficient set of parameters that represent the acoustic properties observed from input speech signals, for subsequent utilization by acoustic modeling.

There are three major types of front-end processing techniques, namely linear predictive coding (LPC) [21], mel-frequency cepstral coefficients (MFCC) [22], and perceptual linear prediction (PLP) [23], where the latter two are most commonly used in state-of-the-art ASR systems.

Acoustic pattern recognition

Acoustic pattern recognition aims at measuring the similarity between an input speech and a reference pattern or a model (obtained during training) and accordingly determines a reference or a model, which best matches the input speech, as an output. One approach of acoustic pattern matching is called the dynamic time warping (DTW) [24]. DTW is a method which measures the distance between each input frame and each reference frame using the dynamic programming algorithm to find the best warping of the pattern, and determines the best match by minimizing the distance between the input frame and the reference frame.

Another approach is based on statistical models. The most successful one is hidden Markov models (HMMs) [25], which characterize speech signals using a pre-trained "hidden" Markov chain. In the training stage, one or more HMMs corresponding to speech sounds of the same class (e.g., phones, words, phrases) are designed and optimized to represent the statistical features of that class. In the recognition stage, probabilistic measures are taken to calculate how much an unknown input speech matches the given set of HMMs. There are many different kinds of HMM structures, among which the discrete HMM (DHMM), continuous-density HMM (CDHMM) are the most popular and successful. CDHMM will be further discussed in later chapters.

Language analysis

In recent years language analysis is becoming more and more important in speech recognition, especially for large vocabulary continuous speech recognition (LVCSR) tasks. The speech decoding process needs to invoke knowledge of pronunciation, lexicon, syntax, and pragmatics in order to produce a satisfactory text sequence for further interpretation. In particular, the probabilistic N-gram language models, a specific form of probabilistic FSG language models, has been widely used due to its ease of implementation and coherence with the structure of an HMM [26].

1.3 Performance measurements of ASR

The performance of an ASR system mainly consists of three major parts: recognition accuracy, complexity and robustness.

1.3.1 Recognition accuracy

Recognition accuracy is the most important and straightforward measure of speech recognition performance. Empirically, the collected speech data are partitioned into training set and test set. The training set, which usually contains most of the available data, is utilized for parameter estimation of the acoustic models. The remaining data form the test set, which is used to measure the ASR performance over signals unavailable during training. The recognition accuracy of training set and test set is evaluated by the training-set word error rate and test set word error rate, respectively, where the latter is a critical design target for most ASR systems. However, as the recognition accuracy for new data is not available in real implementations, the design objective is usually converted to minimizing both the training-set word error rate and the performance difference between the training set and the test set.

1.3.2 Complexity

Complexity is another issue that needs to be considered in most commercial ASR systems, especially when hardware cost is critical to system success. In general, the complexity of an ASR system refers to its computational complexity and model complexity. Computational complexity concerns the cost of execution time in each module of the system. For most practical implementations where the ASR operation is required to be finished in real time, computational complexity should definitely be well considered. Model complexity is usually measured by the number of distinct model parameters. There is a tradeoff between the model complexity and recognition accuracy. A reduced model complexity can provide the obvious benefits of savings in memory and computation while the recognition accuracy will drop down at the same time.

1.3.3 Robustness

While accuracy is crucial to speech recognition performance, robustness is also of great importance to an ASR system. At present, most ASR systems are trained upon a set of speech samples collected under some intended conditions. They would perform well if the operating conditions match the intended conditions. Unfortunately, this assumption is often violated after system deployment, as variation in operating conditions is virtually inevitable. Important aspects of operating conditions include the level of background noise, channel noise and distortion, speaker difference, speaking style and syntactic deviation, spontaneity of speech, etc. In practice, the deviation of these conditions from those assumed during the design phase may result in substantial degradation in performance. This has been the cause of increasing concerns about ASR robustness in recent years [27], and it has in fact become a critical performance index of almost all speech recognition systems.

1.4 Motivation and goal of this work

After nearly sixty years of research, speech recognition technology has reached a relatively high level. However, most state-of-the-art ASR systems run on desktop with powerful microprocessors, ample memory and an ever-present power supply. In these years, with the rapid evolvement of hardware and software technologies, ASR has become more and more expedient as an alternative human-to-machine interface that is needed for the following application areas:

- Stand-alone consumer devices such as wrist watch, toys and hands-free mobile phone in car where people are unable to use other interfaces or big input platforms like keyboards are not available.

- Single purpose command and control system such as voice dialing for cellular, home, and office phones where multi-function computers (PCs) are redundant.

There is a great request for the implementation of automatic speech recognition for these consumer applications.

As a matter of fact, the development has been impeded by a number of issues: computational costs, robustness in adverse environments, etc. One solution for the first issue is to use embedded systems which aim to provide low-power and high speed characteristics. Among various embedded systems, Digital Signal Processor (DSP) is one of the most popular embedded platforms on which computationally intensive algorithms can be implemented. It has the advantage of providing good flexibility and relatively short application development cycle. Constrained by the current hardware technologies, reliable recognition of large-vocabulary fluent speech is still not state of art, but reliable and highly accurate isolated/connected word recognition (IWR/CWR) with small limited vocabulary (tens of words) systems should be achievable. However, the algorithms of IWR/CWR are developed using many floating point and complex functions such as integer divisions, logarithms, cosines and conditional loops. Low cost embedded systems generally can not fast process these functions because they don't have specific hardware structures to handle them. Optimizations of the recognition algorithms for low-cost hardware are necessary. It is important to implement them with no floating-point and minimal complex functions to reduce computational complexity. In fact, there lies a trade-off that algorithm optimization not only makes the implementation run faster, it degrades the recognition accuracy as well. One goal of this research is to deeply understand the ASR algorithms and investigate the possible algorithmic optimizations to improve the ASR accuracy, while controlling both the model complexity and computational complexity with acceptable performance degradation. In recent years, there are some researches on code optimizations for speech recognition on low cost fixed-point embedded systems (DSP or others) [28] [29] [30] and [31]. They did have outlined some optimization techniques for front-end and/or pattern recognition parts. But unfortunately, they did not give a full presentation of the optimization techniques for the whole recognition system and furthermore there is still no such a framework that generally summarizes the optimizations

and the relationship between the optimizations and the recognition performance. Another goal of this research is intensively setting up a framework which could help the system designers who want to realize real-time ASR on DSP estimate the maximal recognition performance they could achieve within their limited hardware recourses.

In a real-world environment, the background noise and when the speaker speaks and stops are both unknown. We need to explore some practical techniques to make the speech recognition system more adaptive to the environment for practical implementations.

1.5 Thesis outline

The rest of this thesis is as follows:

In Chapter 2, we describe the conventional front-end techniques used in automatic speech recognition in more detail. And pattern recognition technologies will be illustrated in Chapter 3.

Since our work is based on a particular hardware platform which contains a Digital Signal Processor (DSP) chip, we need to know the structure and features of this system. In Chapter 4, we first compare different embedded solutions to show the advantage of choosing DSP as the target platform. And we also give a brief view of the considerations and solutions for ASR implementation on hardware aspect.

Due to the consideration of computational complexity and model complexity, optimization strategies should be taken in ASR algorithms. In Chapter 5, we illustrate many optimizations of ASR algorithms for DSP applications. This is the most innovative and hard work in realization of speech recognition system for DSP applications. Two speech recognition systems are established in Chapter 6. Experiments and the performance analysis will be presented in this chapter.

As we know, in real world, noise-corrupted speech will damage the recognition system seriously. In Chapter 7, we propose a commonly-used simple speech enhancement technique, modified spectral subtraction (SS), to reduce the back-

ground noise influence. Meanwhile, since we don't know exactly the beginning and ending of a speech utterance, we need a method to determine whether an utterance has started or not. In this chapter, the traditional end-point detection technique will be utilized. And experiments will be setup to see how these approaches influence the recognition accuracy.

Chapter 8 provides the conclusion and suggestions for the future work.

Chapter 2

Signal processing techniques for front-end

Fig. 2.1 shows a block diagram of the speech recognition system used in our research. A speech recognition system can be roughly divided into two parts, namely front-end and pattern recognition.

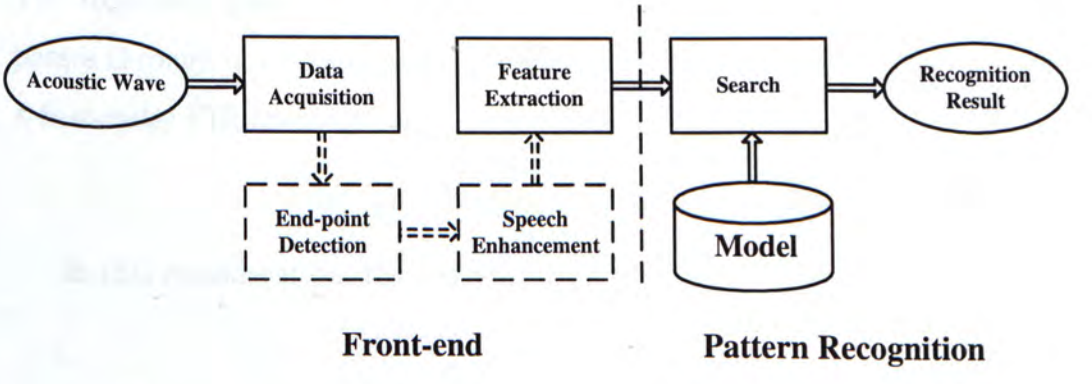


Figure 2.1: Block diagram of ASR system

Typically, the front-end building block includes two modules, namely data acquisition and feature extraction. As an optional choice, the end-point detection and speech enhancement module can be inserted to make the speech signal more adaptive and robust to the noise.

The data acquisition module usually contains a microphone and a codec from which digitized speech data are generated. For DSP applications, speech

data is usually represented as a sequence of 16-bit long signed integers. So this module mainly depends on the hardware system and this will be illustrated in the following chapter. In this chapter, we only focus on the feature extraction module and optional module in the front-end part which contains many signal processing techniques.

2.1 Basic feature extraction principles

Fig. 2.2 is the detailed block diagram of the feature extraction processing. Feature extraction is done on short-time basis. The speech signal is divided into overlapped fixed-length frames. From each frame, a set of frequency-domain or cepstral-domain parameters are derived to form the so-called feature vector. In the following subsections, some basic principles and analysis techniques used in the feature extraction module will be carried out.

2.1.1 Pre-emphasis

The digitized speech signal,$s(n)$, derived from the data acquisition module passes through a pre-emphasis process, which performs spectral flattening with a first-order FIR filter [16]:

$$H(z) = 1 - \alpha \cdot z^{-1}, 0.9 \leq \alpha \leq 1.0 \tag{2.1}$$

In this consideration, the output of the pre-emphasis network,$\tilde{s}(n)$

$$\tilde{s}(n) = s(n) - \alpha \cdot s(n-1) \tag{2.2}$$

where α is the pre-emphasis parameter (a most common value for α is about 0.95). By using this method, the spectrum magnitude of the outgoing pre-emphasized speech will have a 20 dB boost in the upper frequencies and a 32 dB increase at the Nyquist frequency.

2.1.2 Frame blocking and windowing

In this step the pre-emphasized speech signal,$\tilde{s}(n)$, is segmented into frames, which are spaced 20−30 msec apart, with 10−15 msec overlaps for short-time

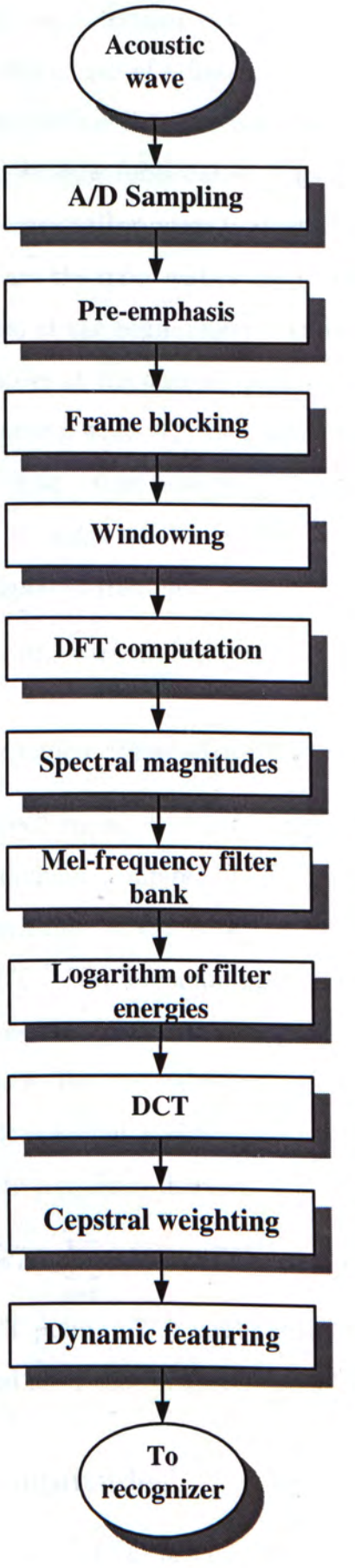


Figure 2.2: Block diagram of MFCC front-end analysis

spectral analysis. Each frame is then multiplied by a fixed length window $[h(n)]_{n=0}^{N_0-1}$, where N_0 is the length of a frame.

Window functions are signals that are concentrated in time, often of limited duration N_0. While window functions such as triangular, Kaiser, Barlett, and prolate spheroidal occasionally appear in digital speech processing systems, Hamming and Hanning are the most widely used to taper the signals to quite small values (nearly zeros) at the beginning and end of each frame for minimizing the signal discontinuities at the edge of each frame [32].

In this research, Hamming window, which is defined in Eq. 2.3, is applied.

$$h[n] = \begin{cases} 0.54 - 0.46 \cdot \cos(2\pi n/N_0), 0 \leq \mathrm{n} \leq \mathrm{N_0} \\ 0 \quad otherwise \end{cases} \tag{2.3}$$

The output speech signal of Hamming windowing can be described as

$$x(n) = h(n) \cdot \tilde{s}(n), 0 \leq n < N_0 \tag{2.4}$$

2.1.3 Discrete Fourier Transform (DFT) computation

After windowing the speech signal, Discrete Fourier Transform (DFT) is used to transfer these time-domain samples into frequency-domain ones. There is a family of fast algorithms to compute the DFT, which are called Fast Fourier Transforms (FFT). Direct computation of the DFT from Eq. 2.5 requires N^2operations, assuming that the trigonometric functions have been pre-computed. Meanwhile, the FFT algorithm only requires on the order of $N \log_2 N$ operations, so it is widely used for speech processing to transfer speech data from time domain to frequency domain.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, 0 \leq k < N \tag{2.5}$$

If the number of FFT points, N, is larger than the frame size N_0, $N - N_0$ zeros are usually inserted after the N_0 speech samples.

2.1.4 Spectral magnitudes

Generally speaking, the signal $X(k)$ is a complex value containing the real and image parts. But in the speech recognition system which deals with the real

speech signal, the complex value is always ignored by researchers. Therefore, only the magnitude of the complex value $X(k)$ is utilized in this situation. If we assume the real and image parts of $X(k)$ are $\mathrm{Re}(X(k))$ and $\mathrm{Im}(X(k))$, then the spectral magnitude of the speech signal should be

$$|X(k)| = \sqrt{\mathrm{Re}(X(k))^2 + \mathrm{Im}(X(k))^2} \tag{2.6}$$

2.1.5 Mel-frequency filterbank

In order to represent the static acoustic properties, the Mel-Frequency Cepstral Coefficient (MFCC) is used as the acoustic feature in the cepstral domain. This is a fundamental concept which uses a set of non-linear filters to approximate the behavior of the auditory system. It adopts the characteristic of human ears that human is assumed to hear only frequencies lying on the range between 300Hz to 3400Hz. Besides, human's ears are more sensitive and have higher resolution to low frequency compared to high frequency. Therefore, the filterbank should be defined to emphasize the low frequency over the high frequency. Eq. 2.7 is the filterbank with M filters (m=1,2,, M), where filter m is the triangular filter given by:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k-f(m-1)}{(f(m)-f(m-1))} & f(m-1) \le k \le f(m) \\ \frac{f(m+1)-k}{(f(m+1)-f(m))} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.7}$$

which satisfies $\sum_{m=0}^{M-1} H_m(k) = 1$.

The central frequency of each mel-scale filter is uniformly spaced below 1 kHz and it follows a logarithmic scale above 1 kHz as shown in Eq. 2.8 and Fig. 2.3. More filters process the spectrum below 1 kHz since the speech signal contains most of its useful information such as first formant in lower frequencies (Fig. 2.4).

$$Mel(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.8}$$

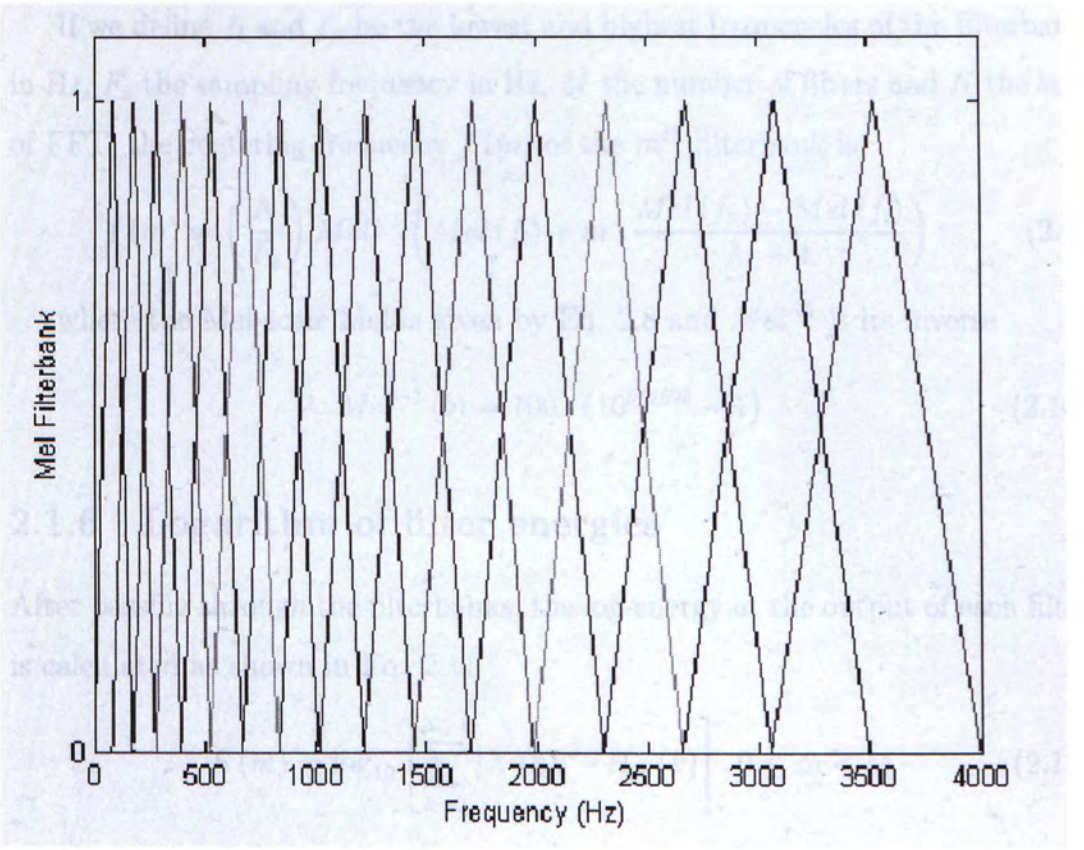


Figure 2.3: Frequency response of mel-filter bank

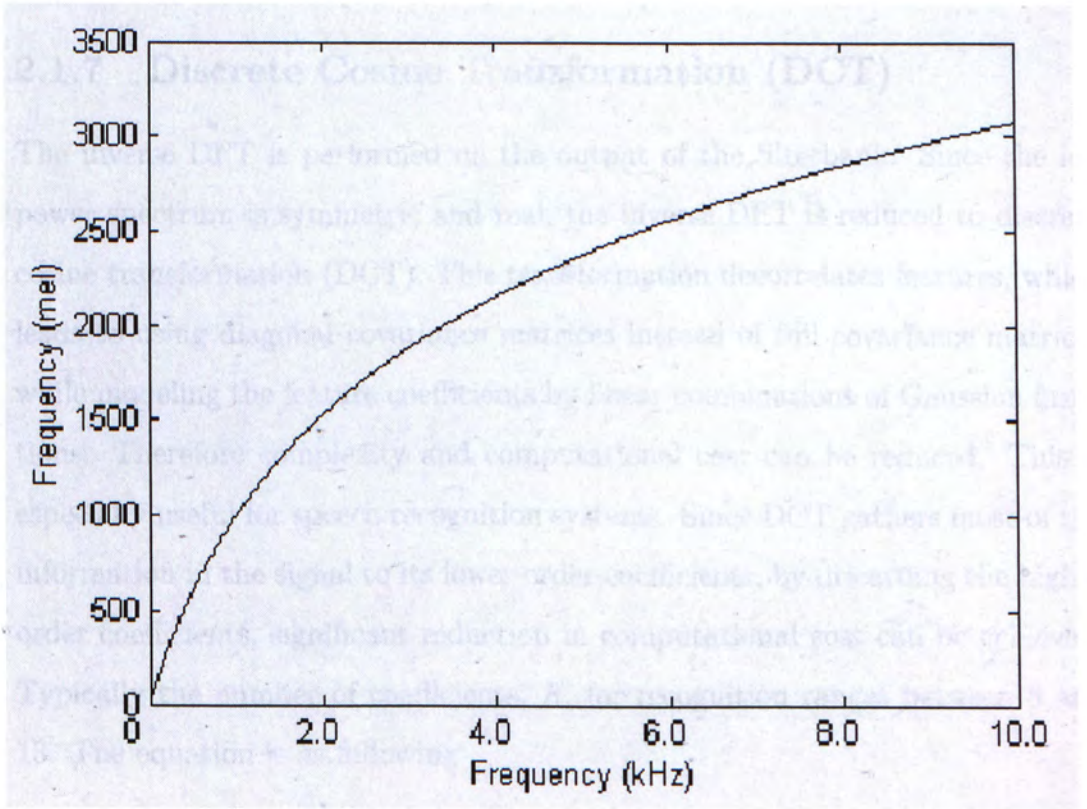


Figure 2.4: Mel scale of central frequency in each filter

If we define f_l and f_h be the lowest and highest frequencies of the filterbank in Hz, F_s the sampling frequency in Hz, M the number of filters and N the size of FFT, the centering frequency $f(m)$ of the m^{th} filterbank is:

$$f(m) = \left(\frac{N}{F_s}\right) Mel^{-1}\left(Mel(f_l) + m \cdot \frac{Mel(f_h) - Mel(f_l)}{M+1}\right) \tag{2.9}$$

where the Mel-scale Mel is given by Eq. 2.8 and Mel^{-1} is its inverse

$$Mel^{-1}(b) = 700 \cdot \left(10^{b/2595} - 1\right) \tag{2.10}$$

2.1.6 Logarithm of filter energies

After passing through the filterbanks, the log-energy at the output of each filter is calculated as shown in Eq. 2.11

$$S(m) = \log_{10}\left[\sum_{k=0}^{N-1} |X(k)|^2 \cdot H_m(k)\right], 0 \leq \mathrm{m} < \mathrm{M} \tag{2.11}$$

Human ears also smooth the spectrum and use logarithmic scale approximately.

2.1.7 Discrete Cosine Transformation (DCT)

The inverse DFT is performed on the output of the filterbank. Since the log power spectrum is symmetric and real, the inverse DFT is reduced to discrete cosine transformation (DCT). This transformation decorrelates features, which leads to using diagonal covariance matrices instead of full covariance matrices while modeling the feature coefficients by linear combinations of Gaussian functions. Therefore complexity and computational cost can be reduced. This is especially useful for speech recognition systems. Since DCT gathers most of the information in the signal to its lower order coefficients, by discarding the higher order coefficients, significant reduction in computational cost can be achieved. Typically the number of coefficients, K, for recognition ranges between 8 and 13. The equation is as following

$$C(k) = \sum_{m=0}^{M-1} S(m) \cos\left(\pi \cdot k\left(m + 1/2\right)/M\right), 0 \leq \mathrm{k} < \mathrm{K} \tag{2.12}$$

2.1.8 Cepstral Weighting

A weighting window (named liftering window) is applied after decorrelating the cepstral coefficients. In this work, the sinusoidal lifter (as shown in Eq. 2.13) is utilized to minimize the sensitivities by lessening the higher and lower cepstral coefficients.

$$\hat{C}(k) = C(k) \cdot \left[1 + \frac{K}{2} \cdot \sin\left(\frac{k\pi}{K}\right)\right], \quad 0 \leq k < K \tag{2.13}$$

2.1.9 Dynamic featuring

In addition to the cepstral coefficients, the time derivative approximations are used as feature vectors to represent the dynamic characteristic of speech signal. To combine the dynamic properties of speech, the first and/or second order differences of these cepstral coefficients may be used which are called the delta and delta-delta coefficients. And these dynamic features have been shown to be beneficial to ASR performance [33]. The first-order delta MFCC may be described as

$$\Delta C^t(k) = \frac{\sum_{l=-P}^{P} l \cdot \hat{C}^{t+l}(k)}{\sum_{l=-P}^{P} l^2} \tag{2.14}$$

where $\hat{C}^t(k)$ denotes the k^{th} cepstral coefficient at frame t after liftering and P is typically set to the value 2 (i.e., five consecutive frames are involved). The second-order delta MFCC is obtained in a straight forward manner [34] [35] [36].

In this work, we use the MFCCs, energy and their first-order delta coefficients to form the feature vectors $O_t, t \in \{1, ..., T\}$. It is worth noting that we do not add the second-order delta coefficients, because they do not show significant improvement in our task and it would definitely increase the computational cost and need more memory to store the model parameters. Also, we apply end-point detection and a speech enhancement technique, namely spectral subtraction to the front-end in an attempt to make the recognition system more robust to noise. These specific issues will be shown latter.

2.2 Practical issues

When people use an automatic speech recognition (ASR) system in real environment, they always hope it can achieve as good recognition performance as human's ears do which can constantly adapt to the environment characteristics such as the speaker, the background noise and the transmission channels. Unfortunately, at present, the capacities of adapting to unknown conditions on machines are greatly poorer than that of ours. In fact, the performance of speech recognition systems trained with clean speech may degrade significantly in the real world because of the mismatch between the training and testing environments. If the recognition accuracy does not degrade very much under mismatch conditions, the system is called *robust*.

2.2.1 Review of practical problems and solutions in ASR applications

Although signal processing technologies show promise in leading to robust systems, the fundamental method for improving robustness is to understand the reason of the mismatch between the training data used in system development and testing data gathered in real environment [16]. Fig. 2.5 illustrates the main causes of acoustic variations resulting from the speech production processes [37]. Generally speaking, there are two main sources of this mismatch:

1. Additive noise, such as fan running, other speaker's speech, car engines, etc.;

2. Channel distortion, such as reverberation, telephone line, microphones, etc.

According to the different sources of the mismatch, robustness techniques may be split into three categories:

1. **Speech enhancement** directly deals with these problems on speech data. It tries to make corrupted speech data much cleaner and matched to the

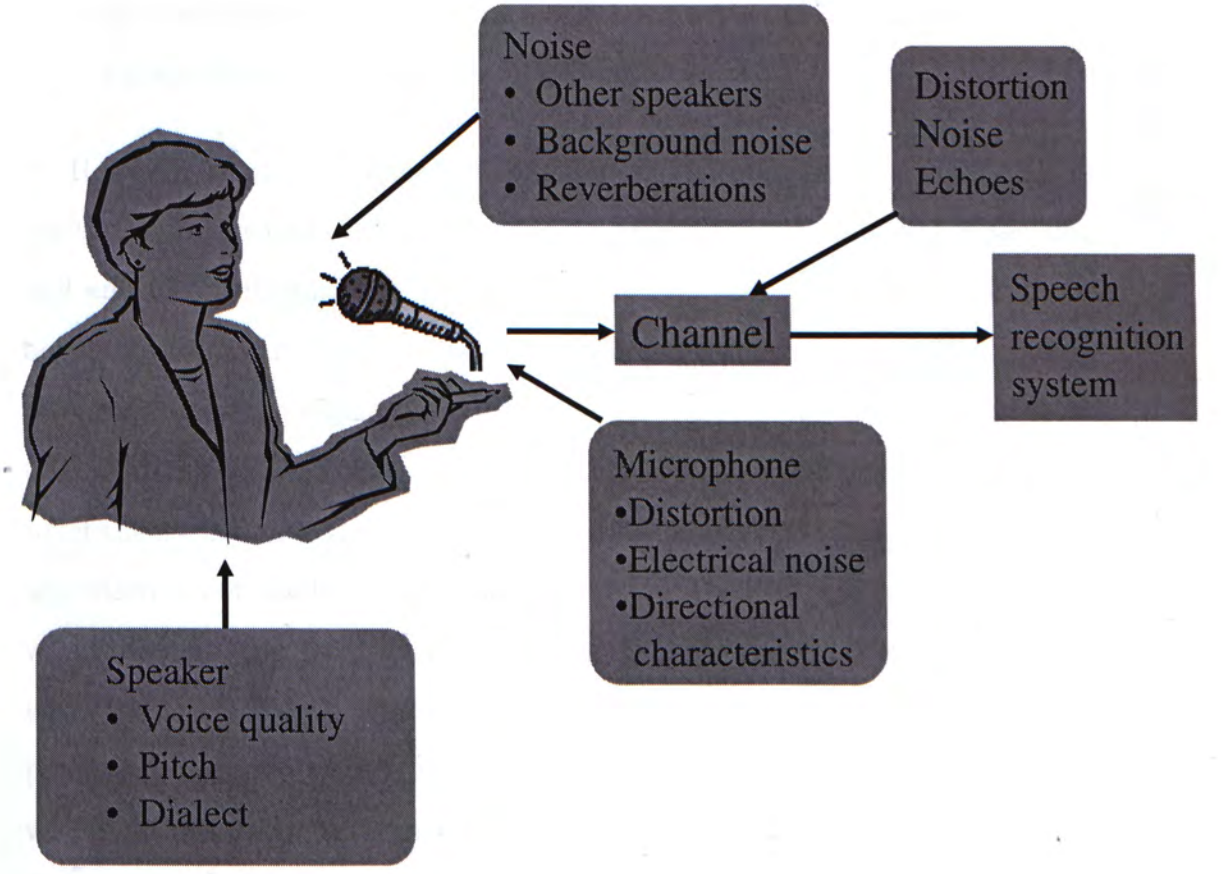


Figure 2.5: Main causes of acoustic variation in speech

acoustic condition of the clean speech data. Spectral subtraction (SS) is a typical example.

2. **Robust speech feature extraction techniques** are used to extract feature more immune to additive noise or channel distortion. It includes Cepstral Mean Subtraction (CMS), Relative Spectral (RASTA), etc.

3. **Model-based compensation approaches** try to adapt the pre-trained recognition models such that they are more matched to the real acoustic environment. The famous examples are Vector Taylor Series (VTS), Parallel Model Combination (PMC), etc.

However, there is another problem in practical implementation of ASR algorithms. In theoretical analysis, a speech recognizer is assumed that the start and end of an utterance is known. It starts the recognition search from the beginning frame and ends the search at the end frame. However, in a real application, utterance segmentation is unknown. One should find a way to detect the beginning and end of an utterance. Some systems use a push-to-talk button to let the users decide the utterance segmentation; therefore utterance detection algorithm is not needed. This model sometimes requires you to push and hold while talking. You push to indicate when the speech begins and then release when the end of speech. The disadvantage is the necessity to activate the speech processing manually each time one speaks. To make the system more flexible, we use an automatic utterance detection algorithm, called end-point detection (EPD), to decide when to start recognition, when to stop recognition in real time [38].

In the following subsections, we will first illustrate a typical model of environment which is critical to the recognition system. Then we will move on to the practical issues, end-point detection and spectral subtraction, in more detail.

2.2.2 Model of environment

As mentioned in Subsection2.2.1, the main effects of environment are additive noise and channel distortion. Additive noise, such as door slams, a fan running in the background, or other speaker's speech, is common in our daily life. Channel distortion may be caused by reverberation, the presence of an electrical filter in the A/D circuitry, the response of the local loop of a telephone line, etc. Fig. 2.6 shows a widely used model of the speech signal corrupted by both additive noise and channel distortion [39].

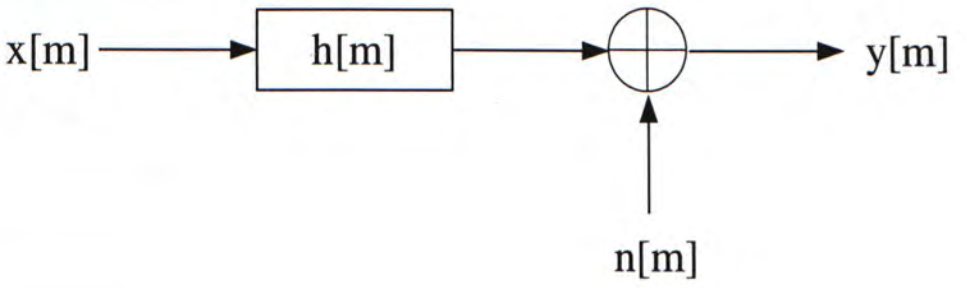


Figure 2.6: A model of the environment

Assume $x\,[m]$ is clean speech signal, $h\,[m]$ is the impulse response of channel distortion, $n\,[m]$ is noise signal, and $y\,[m]$ is corrupted speech signal. In time domain, the relation about these values is:

$$y\,[m] = x\,[m] * h\,[m] + n\,[m] \tag{2.15}$$

Additive noise $n\,[m]$ is defined as white noise. And it is assumed to be stationary and uncorrelated with the clean speech signal $x\,[m]$. In this work, we ignore the effect of channel distortion.

2.2.3 End-point detection (EPD)

When implementing the end-point detection method, it is often assumed that during several frames (10 to 200 milliseconds) at the beginning of the incoming

speech signal, the speaker has not said anything as shown in Fig. 2.7. Thus, within this interval, the statistics of the background silence or noise is measured. The end-point detection is often based on the energy threshold which is a function of time [39]. Actually, several energy-based measurements have been proposed for end-point detection [40] [41] [42] [43]. In this work, we use the mean square energy (MSE) measurement. This is defined as the average of the squared sum of the speech samples in a frame in time domain, which is shown in Eq 2.16.

$$E_s = \frac{1}{N}\sum_{n=1}^{N} s\left(n\right)^2 \tag{2.16}$$

where N is the number of speech samples of one frame.

If we suppose there are M frames of silence or noise at the beginning of the speech utterance, the estimated noise can be achieved by smoothing the MSE derived from these M frames of signals. Depending on the estimated noise, we can set thresholds or noise levels to detect the boundary of speech. If the MSE of a frame is higher than the first predefined threshold, it is possibly believed as the beginning point. Then, if the MSE of a following frame is lower than the second predefined threshold, the end of the speech may be detected.

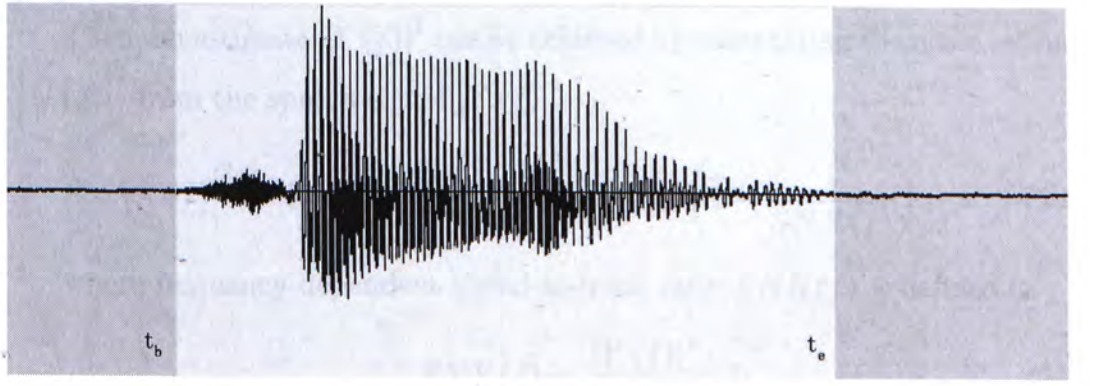


Figure 2.7: End-point detection with boundary t_b and t_e

2.2.4 Spectral subtraction (SS)

In spectral subtraction, we assume that there is no channel distortion but only additive noise in speech data. Then the environment model Eq. 2.15 can be simplified as:

$$y[m] = x[m] + n[m] \tag{2.17}$$

After Fast Fourier Transform, the noisy speech signal is transformed to frequency domain. Basically, most methods of speech enhancement have in common the assumption that the power spectrum of a signal corrupted by uncorrelated noise is equal to the sum of the signal spectrum and the noise spectrum [44].

$$|Y(f)|^2 = |X(f)|^2 + |N(f)|^2 \tag{2.18}$$

However, taking this assumption as a reasonable approximation for short time analysis (20-25msec), it can lead to a simple noise subtraction method. In order to reduce the additive noise, what we need to do is to estimate $|N(f)|^2$ by the received speech data $|Y(f)|^2$. Commonly, a method to achieve this is to use the average $|Y(f)|^2$ over M frames which are known to be just silence/noise (i.e., when no speech is present) as noise estimate. If the noise is stationary, this estimate will be rather accurate.

$$\left|\hat{N}(f)\right|^2 = \frac{1}{M}\sum_{i=0}^{M-1}|Y_i(f)|^2 \tag{2.19}$$

Then, an estimate $|X(f)|^2$ can be achieved by subtracting the noise estimate $\left|\hat{N}(f)\right|^2$ from the speech signal $|Y(f)|^2$:

$$\left|\hat{X}(f)\right|^2 = |Y(f)|^2 - \left|\hat{N}(f)\right|^2 = |Y(f)|^2\left(1 - \frac{1}{SNR(f)}\right) \tag{2.20}$$

where frequency-dependent signal-to-noise ratio $SNR(f)$ is defined as

$$SNR(f) = \frac{|Y(f)|^2}{\left|\hat{N}(f)\right|^2} \tag{2.21}$$

Eq. 2.20 shows the magnitude of the frequency-domain speech signal but not the phase. It is not a problem since the phase information is always ignored in speech recognitions [44].

Chapter 3

HMM-based Acoustic Modeling

In a speech recognition system, the incoming speech features from the front-end part are modelled by hidden Markov models (HMM). Since people traditionally suppose the speech signal to be short-time stationary and the speech features carry the information of the speech acoustic properties, the features should obey some kind of distribution. So HMMs, which have long dominated the world of acoustic modeling, are used to characterize the speech signal as a parametric stochastic process and statistically represent the variation of a speech unit (a word/phoneme/syllable).

3.1 HMMs for ASR

Generally speaking, a hidden Markov model in an automatic speech recognition system is defined by the following elements:

- $O = \{o_1, o_2, ..., o_T\}$ - The output observation sequence. The observation symbols correspond to the physical output of the recognition system being modeled;
- $\Omega = \{1, 2, ..., N\}$ - A set of states corresponding to the state space;
- $A = \{a_{ij}\}$ - State-transition probability matrix, where a_{ij} represents a transition from state i to state j;

- $B = \{b_j(\vec{o}_t)\}$ - The output probability matrix, where $b_j(\vec{o}_t)$ is the probability of emitting symbol $\vec{o}_t$ when state j is entered;
- $\pi = \{\pi_i\}$ - The initial state distribution where

$$\pi_i = P(s_0 = i) \qquad 1 \leq i \leq N.$$

Since a_{ij} and π_i are all probabilities, they must follow these probabilities:

$$\sum_{j=1}^{N} a_{ij} = 1$$

$$\sum_{i=1}^{N} \pi_i = 1$$

On all accounts, we can use the following notation to represent the whole parameter set of an HMM for convenience.

$$\Phi = (A, B, \pi)$$

In this work, we use the widely applied model (as shown in Fig. 3.1) with the following properties:

- Continuous density hidden Markov model (CDHMM);
- States with Gaussian mixtures;
- Left-to-right word model, no state skip.

3.2 Output probabilities

The output probabilities can be discrete or continuous. In order to achieve a higher resolution of the output result, we choose the continuous density hidden Markov model. In this case, the output probability density function (PDF) is the multivariate Gaussian density:

$$b_j(\vec{o}) = G(\vec{o}, \vec{\mu}_j, U_j) = \frac{1}{\sqrt{(2\pi)^D \det(U_j)}} e^{\left(-\frac{1}{2}(\vec{o}-\vec{\mu}_j)^T U_j^{-1}(\vec{o}-\vec{\mu}_j)\right)} \tag{3.1}$$

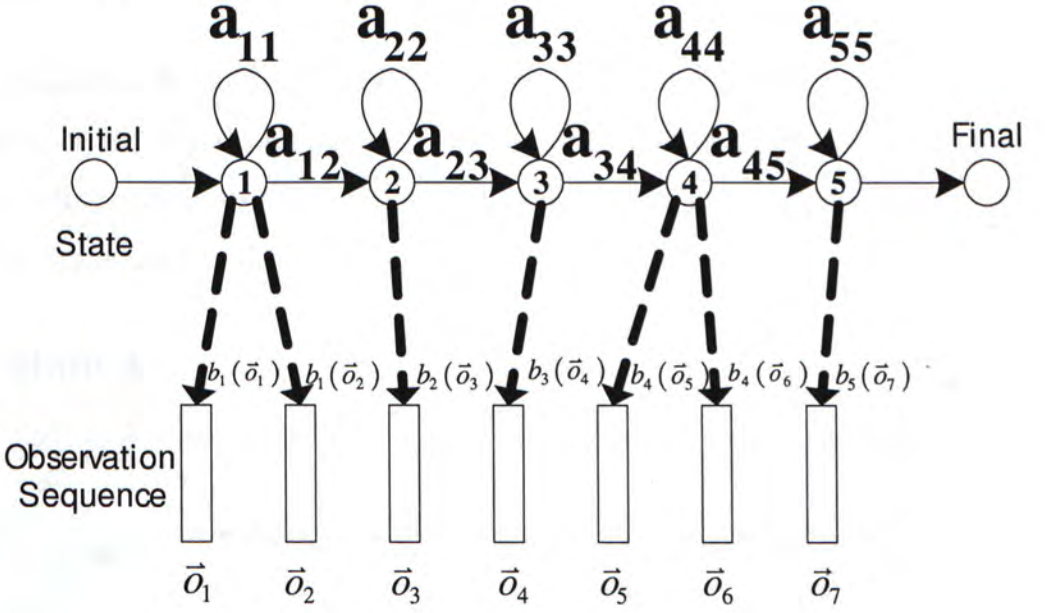


Figure 3.1: A simple graph view of an HMM

where D is the dimension of the vector space. It is the length of the feature vector. The parameters of the Gaussian PDF are the mean vector $\vec{\mu}_j$ and the symmetric covariance matrix U_j. Due to the decorrelated characteristic derived from the feature extraction part, the covariance matrix is diagonal. The determinant of the covariance matrix U_j can be simplified to the multiplication of all the diagonal elements, $\prod_{d=1}^{D} \sigma_{jd}^2$.

Practically, only one Gaussian distribution can not appropriately estimate the speech parameters. In this consideration, Gaussian mixtures are often used as following:

$$\begin{aligned} b_j\left(\vec{o}\right) &= \sum_{m=1}^{M} c_{j,m} G\left(\vec{o}, \vec{\mu}_{j,m}, U_{j,m}\right) \\ &= \sum_{m=1}^{M} c_{j,m} \frac{1}{\sqrt{(2\pi)^D \det(U_{j,m})}} e^{\left(-\frac{1}{2}\left(\vec{o}-\vec{\mu}_{j,m}\right)^T U_{j,m}^{-1}\left(\vec{o}-\vec{\mu}_{j,m}\right)\right)}, \\ \sum_{m=1}^{M} c_{j,m} &= 1 \end{aligned} \tag{3.2}$$

Mixture densities are in principle flexible enough to sufficiently approximate "arbitrary" densities when an appropriate number of components is used.

3.3 Viterbi search engine

The algorithm for finding the best state sequence (Viterbi, 1967; Forney, 1973) is known as the Viterbi algorithm. It is an application of dynamic programming for finding a best scoring path in a directed graph with weighted arcs. This trellis framework is shown in Fig. 3.2.

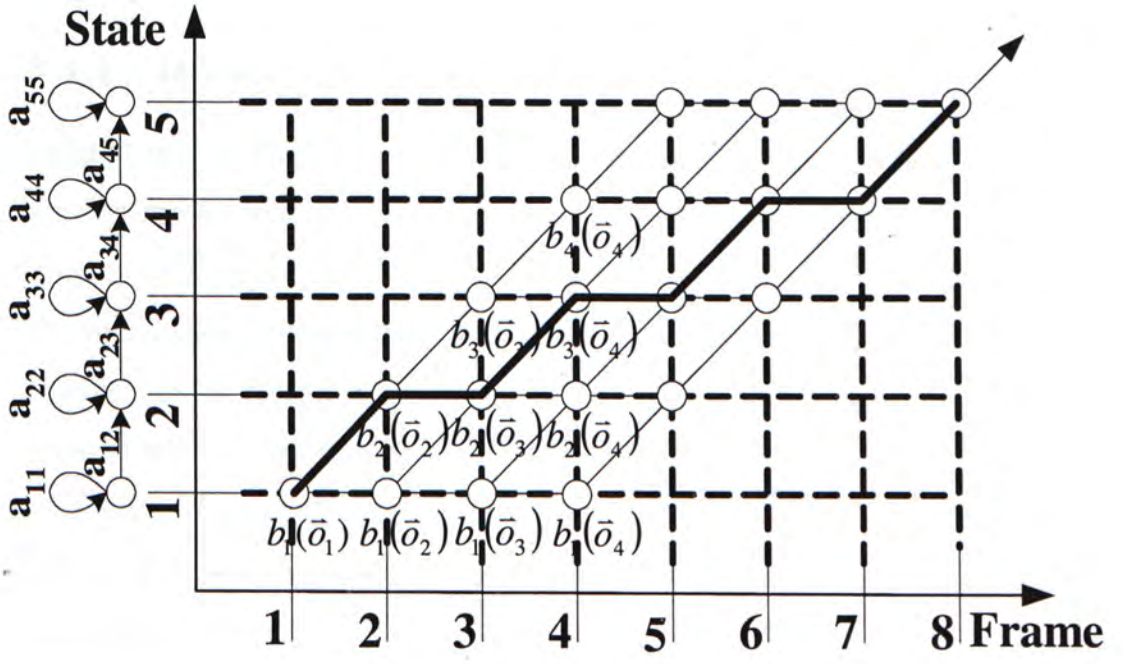


Figure 3.2: The Viterbi trellis computation for the HMM

Generally speaking, the Viterbi algorithm for isolated word recognition can be separated into three major parts

- Initialization

$$\delta_1(i) = \pi_i b_i(\vec{o}_1), \qquad 1 \le i \le N$$

- Recursion

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij}\right] \cdot b_j(\vec{o}_t), 1 \le j \le N, 2 \le t \le T$$

- Termination

$$P = \delta_T(N)$$

P is the score from one model. The best matched model is achieved by comparing each P of all the models.

3.4 Isolated word recognition (IWR) & Connected word recognition (CWR)

In this work, we mainly focus on the isolated word recognition and connected word recognition algorithms which are computationally saving and have been implemented on DSP platforms.

3.4.1 Isolated word recognition

Isolated word recognition, by its name, is a task within which only a single word is recognized. Each word forms a unique model which can be a digit, a letter or a string of syllables. These words or strings are all modelled in the vocabulary. The vocabulary size is always finite, and there are only a few models can be recognized. Due to the simplicity of the searching structure, isolated word recognition has been commercially available [45].

Let $V = \{w_1, w_2, ..., w_v\}$ be a set of v words to be recognized. If A is an input utterance to be recognized and all words are equally probable to be spoken, then the recognized word is

$$\tilde{w} = \arg\max_{w \in V} P(A|w)$$

The process of an isolated word recognition system is shown in Fig. 3.3.

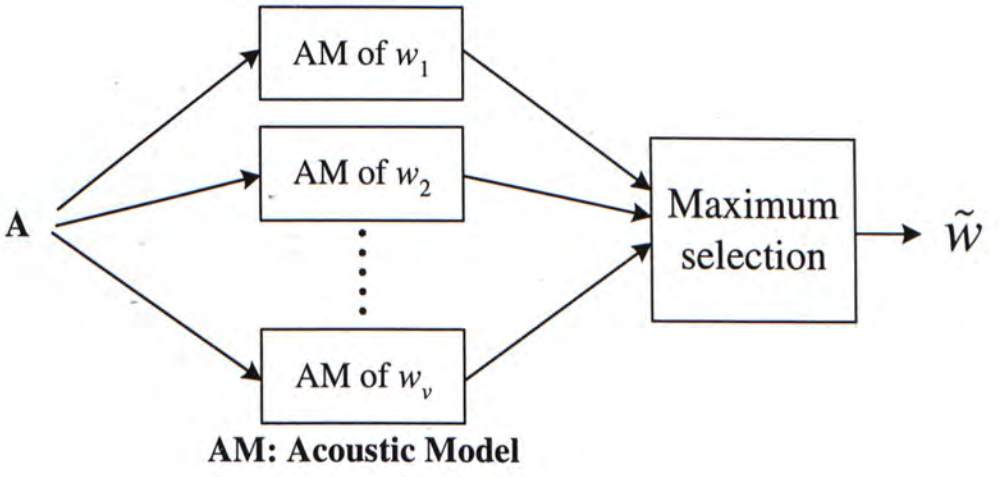


Figure 3.3: Process of isolated word recognition

3.4.2 Connected word recognition (CWR)

Connected word recognition is an extension of isolated word recognition. For connected word recognition, the exit state of one HMM is connected to the entry state of another HMM. In this case, path extension can be within or across models. The recognition result is given by the best model sequence $\tilde{W} = \{\tilde{w}_1, \tilde{w}_2, ..., \tilde{w}_n\}$ that has the highest accumulated probability at the last frame. The search network of connected word recognition is shown in Fig. 3.4.

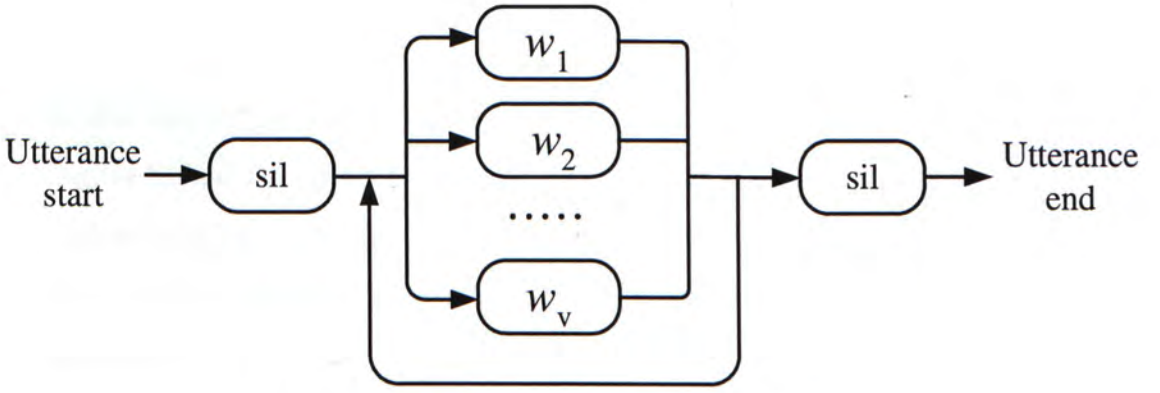


Figure 3.4: The search network of connected word recognition

Chapter 4

DSP for embedded applications

In this chapter, we focus on reviewing the hardware environment: a fixed-point Digital Signal Processor (DSP) platform. First, a brief view of the embedded systems will be given and comparisons will be made to illustrate the advantage of using DSP platform for our work. Second, the key features of the specific platform will be shown to support the reason of choosing this DSP platform for the ASR implementation. At last, we will roughly explain the difference between fixed-point and floating-point algorithms.

4.1 Classification of embedded systems (DSP, ASIC, FPGA, etc.)

For embedded systems, the cost of speech recognition can be negligible comparing to the total system cost. Typically, only 5 to 10% of the cost can be used for additional speech module to the product. With the advances in semiconductor technologies, a number of mainstream Integrated Circuit (IC) products which use digital circuit techniques had emerged. The digital revolution not only increases circuit robustness, cost efficiency, system integration, and flexibility, but also enables a continuous stream of completely new consumer applications, increasingly based on embedded (re)programmable processor cores like Digital Signal Processors (DSPs), and Field-Programmable Gate Arrays (FPGA) chips and un-programmable processors like Application-specific ICs

(ASICs) [46]. There are many principles such as flexibility, scalability, reusability, portability and short development times for selecting the right hardware for the embedded speech recognition engine. The following is the comparison of some alternatives available for digital signal processing with DSP, which is the hardware applied in this research:

- The FPGA alternative

  Field-Programmable Gate Arrays have the capability of being reconfigurable within a system, offering reasonably fast time to market. However, FPGAs are significantly more expensive and typically have much higher power dissipation than DSPs with similar functionality.

- The ASIC alternative

  Application-specific ICs can be used to perform specific functions extremely well, and can be made quite power efficient. However, since ASICs are not field reprogrammable, they can not be changed after development. Consequently, every new version of the product requires a redesign and has to be re-produced. This is a big impediment to rapid time-to-market. On the other hand, programmable DSPs can merely change the software program without changing the silicon, which greatly reduces the development cost and makes aftermarket feature enhancements possible with mere code downloads.

Furthermore, more often than not, in real time signal processing applications, ASICs are typically employed as bus interfaces, glue logic, and functional accelerators for a programmable DSP-based system. According to the above description on various popular embedded systems nowadays, the reason of choosing DSP as the hardware platform for our research is quite obvious:

- Single-cycle multiply-accumulate operation;
- Real-time performance, simulation and emulation;
- Flexibility;

- Reliability;
- Increased system performance;
- Reduced system cost.

Applications of DSP:

- Classic signal processing: digital filtering, adaptive filtering, FFT, spectrum analysis;
- Modern signal processing: AR, ARMA, wavelet analysis;
- Speech processing: speech coding, speech synthesize, speech recognition, speech enhancement, speech mail, speech storage;
- Image processing: image compressing/transferring, image recognition, animation, image enhancement;
- Military electronics;
- Automatic control;
- Consumer electronics.

Although DSP is powerful for real-time processing, especially for speech processing, it is not that easy to implement the functionally complex automatic speech recognition algorithm on it.

In recent years, some new embedded systems have emerged, such as MIPS and Intel StrongArm. These systems are mainly embedded in iPAD and they are all for general purpose use. They generally don't have any special architectures for signal processing like DSP does.

4.2 Description of hardware platform

In our research, the hardware platform we are using is a DSP Starter Kit (DSK) board (Fig. 4.1) with a fixed-point, 160 MHz clock cycle DSP on it, namely TMS320VC5416 DSP. This starter kit is a product of Texas Instrument Inc.

(TI). It is a low-cost platform which enables customers to evaluate and develop applications for the TI C54X DSP family. The primary features of the DSK are:

- 160 MHz TMS320VC5416 DSP;
- PCM3002 Stereo Codec;
- A microphone with amplification;
- On-board Flash and SRAM.

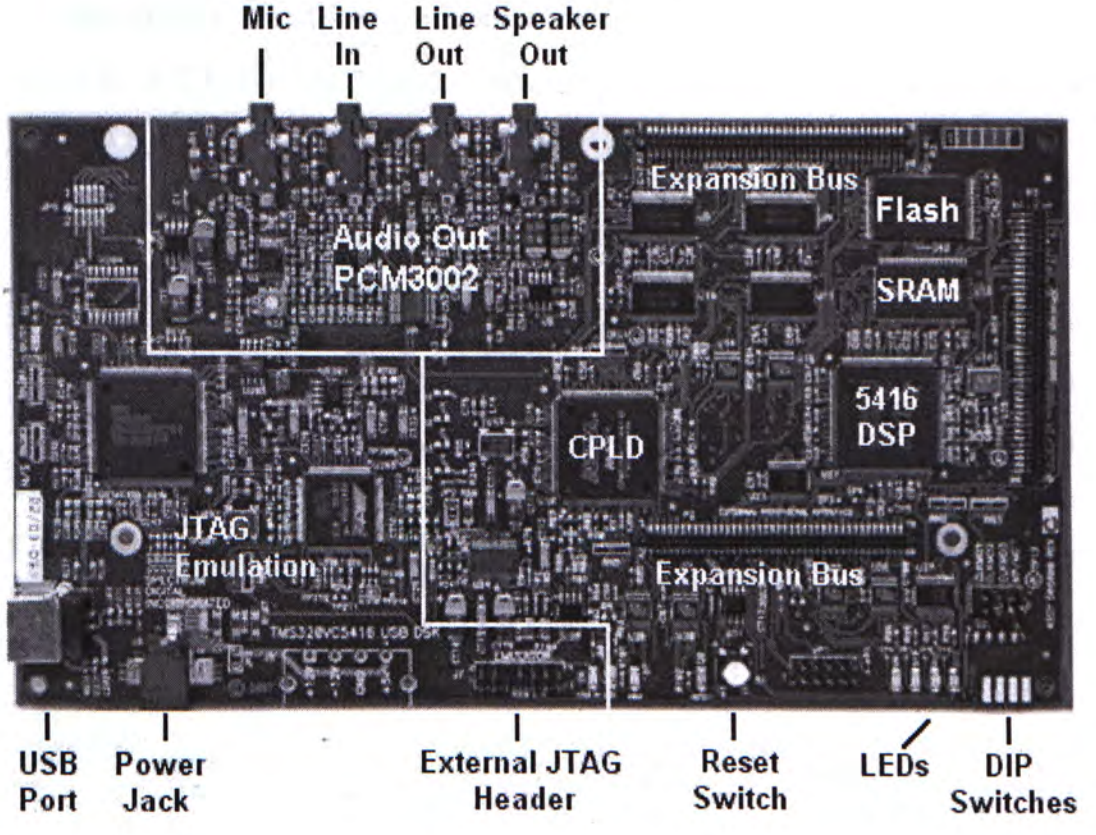


Figure 4.1: Main sections on the board

In this DSP, there are some specific blocks which are designed for some commonly-used operations, such as FFT, VQ, etc. These architectures can make the DSP run faster and more effective for these operations in speech recognition, speech coding and some other implementations.

4.3 I/O operation for real-time processing

In Fig. 4.1, there is a microphone port which can be connected to an external microphone to pass analog speech data in and also there is a speaker out port which can output the processed speech data simultaneously. Both hardware and software should cooperate to realize the real-time input/output operation. In this project, we utilize the pipeline method which is suggested by TI [47].

Fig. 4.2 is the hardware structure of I/O processing. First, the digitized speech sample from the CODEC passes through a serial port interface on the chip and then is stored in the Data Receive Register (DRR). Second, after a dedicated frame of speech data is full, a hardware interrupt, called serial port receive interrupt service routine (ISR), will be enabled and this frame of data goes to the audio software interrupt which activates the main function of the speech processing. Then when the processing of this frame is completed, the processed speech data will return to the ISR and each speech sample will serially go through the Data Transmit Register (DXR) which is connected to the output serial port.

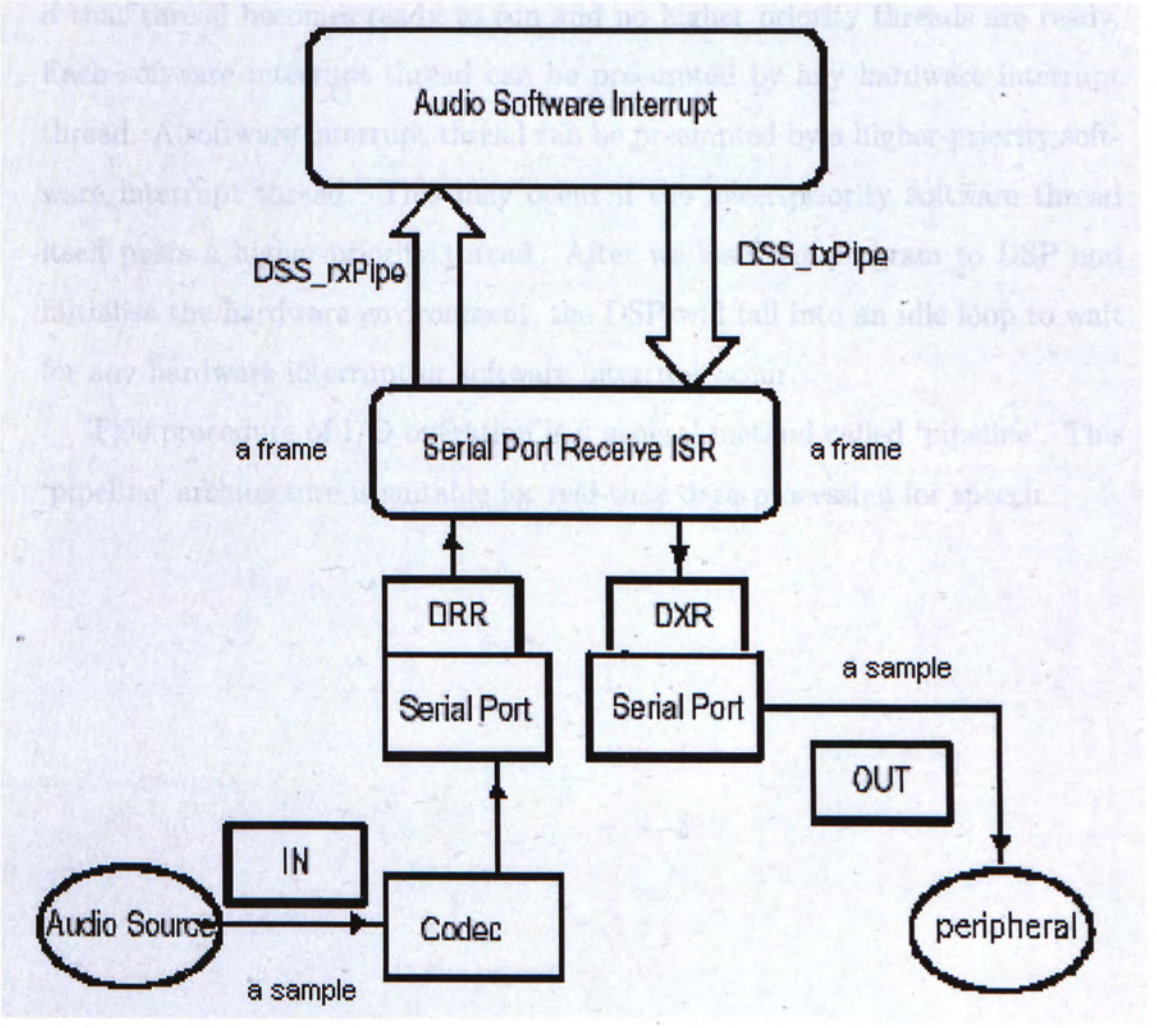


Figure 4.2: Hardware structure of I/O processing

Fig. 4.3 is the diagram showing how the interrupts operate along processing. In this diagram, HWI1, HWI2, and HWI3 are hardware interrupt (HWI) threads, and SWI1, SWI2, SWI3, and SWI4 are software interrupt (SWI) threads. The priority of threads decreases from left to right. (SWI2 and SWI3 have the same priority.) A thread can be interrupted by a thread to its left, if that thread becomes ready to run and no higher priority threads are ready. Each software interrupt thread can be preempted by any hardware interrupt thread. A software interrupt thread can be preempted by a higher-priority software interrupt thread. This may occur if the lower-priority software thread itself posts a higher-priority thread. After we load out program to DSP and initialize the hardware environment, the DSP will fall into an idle loop to wait for any hardware interrupt or software interrupt occur.

This procedure of I/O operation is a general method called 'pipeline'. This 'pipeline' architecture is suitable for real-time data processing for speech.

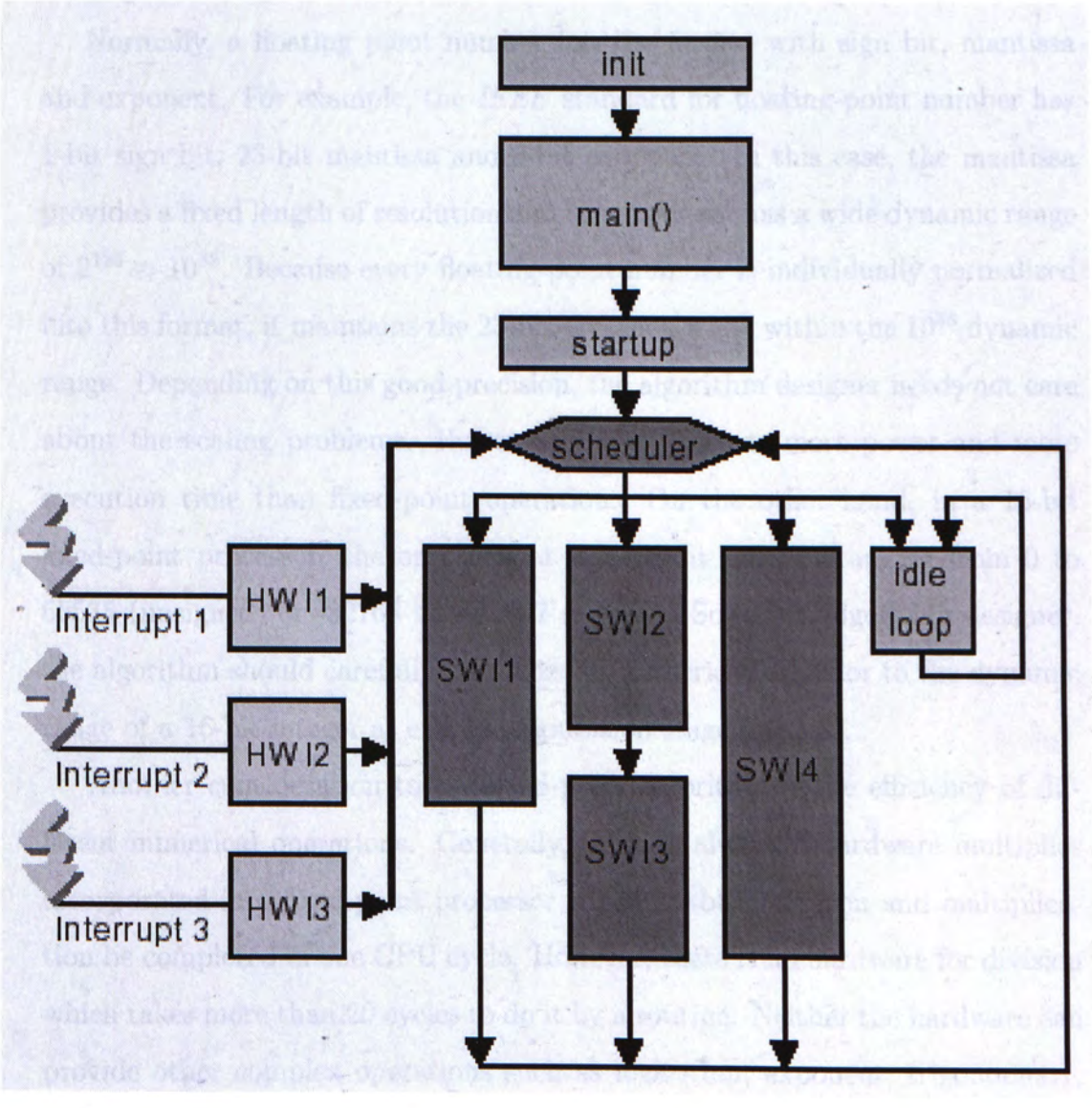


Figure 4.3: Diagram of software interrupt priorities

4.4 Fixed point algorithm on DSP

While most ASR systems for PC use are based on floating-point algorithms, most of the processors used in embedded systems are fixed-point processors. An inevitable deviation from the original design is caused by the finite amount of arithmetic precision available with fixed-point processors [48].

Normally, a floating point number has the format with sign bit, mantissa and exponent. For example, the *IEEE* standard for floating-point number has 1-bit sign bit, 23-bit mantissa and 8-bit exponent. In this case, the mantissa provides a fixed length of resolution and the exponent has a wide dynamic range of $2^{128} \approx 10^{38}$. Because every floating-point number is individually normalized into this format, it maintains the 23-bit precision while within the 10^{38} dynamic range. Depending on this good precision, the algorithm designer needs not care about the scaling problems. However, it usually costs more power and more execution time than fixed-point operation. On the other hand, in a 16-bit fixed-point processor, the only format is a 16-bit integer, ranging from 0 to 65535 (unsigned) or -32768 to +32767 (signed). So for an algorithm designer, the algorithm should carefully normalize its numerical behavior to the dynamic range of a 16-bit integer at every computation stage [31] [49].

Another consideration to the fixed-point algorithm is the efficiency of different numerical operations. Generally, there is always a hardware multiplier incorporated in a fixed-point processor which enables addition and multiplication be completed in one CPU cycle. However, there is no hardware for division which takes more than 20 cycles to do it by a routine. Neither the hardware can provide other complex operations such as logarithm, exponent, trigonometric functions and etc. To avoid such complex operations, some strategies should be considered. For example, we often replace division operation with the precomputed inverse data so that a division can be changed to a multiplication. Another way for taking division is using bit-shift. This is a simple but effective operation on DSPs. For example, right-shift one bit for a value in binary format is equal to dividing this value by 2. And bit-shift only needs 1 cycle indeed. In the latter chapter, we will discuss a number of useful strategies of optimizing

the speech recognition algorithms for DSP applications in more detail.

Chapter 5

ASR algorithm optimization

5.1 Methodology

Despite the continual growth of the computing power of DSP, direct implementation of ASR algorithms can't offer real-time processing. In this chapter, we will discuss the optimizations of ASR algorithms for real-time processing on DSP.

When porting speech recognition algorithms to embedded platforms, the essential procedures will be included:

- Use fixed-point computation as far as possible;
- Analyze the algorithm and identify the computationally most critical parts;
- Simplify the computation by, e.g. using look-up table, avoiding complex function, etc.;
- Avoid unnecessarily repetitive computation;
- Pre-compute frequently used parameters as pre-stored constant values whenever possible;
- Use low-level language if appropriate.

In this work, we propose a set of optimizations for ASR algorithms. By analyzing each computation part in ASR, the most effective optimization methods are applied. Details are shown in the following sections.

5.2 Floating-point to fixed-point conversion

As described in Chapter 4, signal processing algorithms are often converted from floating-point to fixed-point because of the hardware requirements [50]. For the ASR algorithms as described in Chapters 2 and 3, there are three main parts that have to be converted from floating-point to fixed-point:

- Acoustic front-end;
- Computation of Gaussian probability density function;
- Viterbi search.

Regarding the front-end, fixed-point implementations of common signal processing techniques such as autocorrelation, convolution, windowing, Linear Predictive Coding (LPC), and LPC to cepstrum (LPCC) conversion have been reported in [50] and [51]. The implementation of mel-frequency cepstral coefficient (MFCC) was discussed in [31].

The fixed-point format here only has 16-bit. To avoid overflow we should pre-estimate the maximum magnitude of the input waveform and then normalize the waveform to make the maximum magnitude lower than +32767 ($2^{15} - 1$). Additionally, multiplications and summations would cause overflow too. For the development of fixed-point algorithm, these operations should be paid more attention because they are very "dangerous" and may break the whole system. With certain amount of testing data, we can roughly determine how many bits we need to shift at these operations to avoid overflow and attain the highest precision by reserving the effective bits as many as possible at the same time. For example, when we calculate the energy output of a mel-scale filterbank, it is needed to compute the summation of the power of the samples (Eq. 2.11).

In this case, the resulted filterbank energy may overflow. We need to find out how many bits should be shifted to avoid this problem.

In HMM, the computation of Gaussian distributions with fixed-point arithmetic has to be fast because it is one of the most time-consuming part of the whole recognition system (typically between 30% and 60% of the decoding time). And since its precision can greatly affect the recognition performance, this computation has to be accurate as well.

Suppose the log likelihood of an observation vector is calculated as: (Eq. 3.1)

$$\ln b\left(\vec{o}\right) = -\frac{1}{2}\left(D \cdot \ln\left(2\pi\right) + \ln\left(\prod_{d=1}^{D}\sigma_d^2\right)\right) - \frac{1}{2}\sum_{d=1}^{D}\frac{\left(o_d - \mu_d\right)^2}{\sigma_d^2} \tag{5.1}$$

The first term $-\frac{1}{2}\left(D \cdot \ln\left(2\pi\right) + \ln\left(\prod_{d=1}^{D}\sigma_d^2\right)\right)$ can be pre-computed since it is independent to the observation $\vec{o}$. Most of the computation and integer conversion errors are caused by computing the squares $\left(o_d - \mu_d\right)^2$ and the product $\sum_{d=1}^{D}\frac{\left(o_d-\mu_d\right)^2}{\sigma_d^2}$. In order to achieve high performance, many different hardware applications chose different solutions for this tough problem [46] [52]. With 16-bit fixed-point DSP, we do the followings:

1) Pre-compute the scaled factor $SCALE/\sigma_d^2$ (here $SCALE = 2^{26}$) which is a 16-bit integer so that the division operation becomes a multiplication depending on the representation of σ_d^2 [53];

2) Quantize all the other parameters in the models to 16 bits and use 32 bit multiplications;

3) Multiply $o_d - \mu_d$ by $SCALE/\sigma_d^2$ and right shift 11 bits to get $index1$;

4) Multiply $index1$ by $o_d - \mu_d$ and right shift 12 bit to get $index2$;

5) After the summation, right shift the result with 3 bit to rescale the value to the original $\sum_{d=1}^{D}\frac{\left(o_d-\mu_d\right)^2}{\sigma_d^2}$.

By performing the above steps, overflow can be avoided with high probability and precision can be maintained.

At the search level, the probabilistic score of a search path is the product of all the transition and observation probabilities along its path. Because these probabilities were small values (< 1.0), they are generally computed into logarithm to avoid underflows. But the scores still have a large range after logarithm while the speech utterance is long. We have to use long integer of 32 bit to store the recognition score to prevent overflow.

5.3 Computational complexity consideration

For real-time speech recognition based on DSP, we should try our best to reduce the computation in each part of the recognition. In the following subsections, we will describe our approaches in detail.

5.3.1 Feature extraction techniques

Feature extraction is computationally intensive because it involves many signal processing steps such as windowing, Fast Fourier Transform, Mel-scale filter banks, Discrete Cosine transform and so on. For each of these steps, different approximation techniques have been considered.

Windowing

In this work, we use the Hamming window as described in Eq. 2.3. For a frame with fixed-length N, the window shape is fixed and it is not needed to calculate the window function $h(n)$ for each incoming frame. Instead, $h(n), 0 \leq n < N$ is pre-stored in the RAM. In this way, we save N cosine operations, N multiplications and N additions which require a lot of CPU cycles. We will need N 16-bit words of RAM to store the window function which is not overburden compared to the execution time it costs.

Integer FFT Implementation

Although the Fast Fourier Transform (FFT) is much more efficient than the original Discrete Fourier Transform (DFT), it still requires the computation of many exponential or trigonometric functions and multiplications that would take a long execution time on fixed-point DSPs. Integer implementation of FFT for DSP applications is highly desirable.

In this work, 256 point radix-2 in-place FFT is used. Since speech is a real-valued signal, each input frame contains 256 real elements [54]. The result contains the first half i.e. 128 complex elements as the FFT output. This output still provides the full information because an FFT of a real sequence has even symmetry around the center or Nyquist point ($N/2$).

The output $y(k)$ is shown in the following format:

$\mathrm{Re}\{y(0)\}, \mathrm{Im}\{y(N/2)\} \rightarrow DC$

$\mathrm{Re}\{y(1)\}, \mathrm{Im}\{y(1)\}$

$\mathrm{Re}\{y(2)\}, \mathrm{Im}\{y(2)\}$

.....

$\mathrm{Re}\{y(N/2)\}, \mathrm{Im}\{y(N/2)\}$

where *Re* and *Im* mean the real and imaginary parts of one output element.

Because of the complexity of FFT calculation which contains exponential and many multiplications and divisions, it is hard to realize the 256-point FFT using high level language (like C language) with real-time constraints. The alternative method used in this work is using assembly language. The assembly language implementation of FFT is provided as standard reference library by TI. It runs considerably faster than the equivalent code in C language [55].

Here the FFT is in 16-bit format because it is accurate enough for speech recognition. So we did not have to use 32-bit FFT which costs more computation. And to avoid overflow, we need a scale down of factor 2 at each of the 8 butterfly stages. Meanwhile, the reference library allows us to remove the input frame and replace the same memory space with the output elements for convenience and saving memory [31].

Computation of spectral magnitude

For speech recognition, the phase of the spectrum after FFT is always ignored. For each complex element in frequency domain only the magnitude is needed. The magnitude of $x = s_r + j \cdot s_i$ is $|x| = \sqrt{s_r^2 + s_i^2}$. The computation includes a square root, two multiplications and an addition. Here we suppose s_r and s_i are non-negative. For a frame of N FFT points, there will be $N/2$ square roots operations which are still very expensive in computation. To get rid of the square roots, three methods can be considered:

- A simple method is shown as following:

$$|x| = \sqrt{s_r^2 + s_i^2} \approx \max(s_r, s_i) \tag{5.2}$$

  which essentially select the larger one of s_r and s_i. This approximation can be used [56], if the accuracy requirements are not very stringent;

- Better accuracy can be achieved by using a lookup table as described in [53]. This algorithm is shown in Appendix A. By using lookup table, the precision can be increased at the price of increased memory requirement;

- Another method is based on a linear combination of the real and imaginary parts,s_r and s_i, which is described by

$$\begin{aligned} |x| &= \sqrt{s_r^2 + s_i^2} \approx \alpha \times \max(s_r, s_i) + \beta \times \min(s_r, s_i) \\ &= \max(s_r, s_i) \times (\alpha + \beta \times t), \qquad 0 \le t = \tfrac{\min(s_r, s_i)}{\max(s_r, s_i)} \le 1 \end{aligned} \tag{5.3}$$

In the third method, we try to use a linear curve to fit the conic. We consider a set of data points of t between 0 and 1. By calculating the least mean square error, we find that when we set the coefficients to $\alpha = 0.9344$ and $\beta = 0.4269$, the mean square error (MSE) is minimized. Furthermore, for fixed-point DSP implementation where calculating a multiplication of decimal values also needs more execution time than bit shift, it is better to regularize the coefficients to the power 2 format. So we let $\alpha = 1$ and $\beta = 0.25$ (equivalent to right shifting two bits) which can maximally minimize $e(t)$ with minimal execution time. Table 5.1 shows the mean square error for different setting of these coefficients.

And we found that the mean square error of this setting is 1.9635 while the error is 0.8465 when the coefficients are set to $\alpha = 0.9344$ and $\beta = 0.4269$. In the experiment as described in Chapter 6, we only use the third method and let $\alpha = 1$ and $\beta = 0.25$ due to the comparison of MSE in Table 5.1.

	α	β	Mean square error
1	0.9344	0.4269	0.8465
2	1	0.25	1.9635
3	1	0	6.1508

Table 5.1: MSE of linear combinations

Output of mel-frequency filterbank

Recall that the filterbank output in Eq. 2.11 are calculated as the squared magnitude. In practice, this will present some challenges since this squared number multiplied by the filter coefficients, $H_m(k)$, would easily overflow the 32-bit registers. This overflow can be avoided by using the magnitude instead of the squared magnitude:

$$S(m) = \log_{10}\left[\sum_{k=0}^{N-1} |X(k)| \cdot H_m(k)\right], 0 \leq \mathrm{m} < \mathrm{M} \tag{5.4}$$

There is a trick of calculating the energy in each filterbank [57]. As mentioned in Chapter 2, traditionally the first step is finding the samples in one filterbank. Second, the magnitude is multiplied with the corresponding weight of the filterbank function. At last, the weighted magnitudes in one filterbank are summed to get the energy and logarithm is taken. We can find that the whole procedure contains many multiplications and additions.

As shown in Fig. 2.3, if we normalize the gain of the filterbanks to 1, the weight for each sample will be in the range between 0 and 1. For each sample, it may fall into two overlapped filterbanks. We need to calculate the weight twice. Consider the example as illustrated in Fig. 5.1.

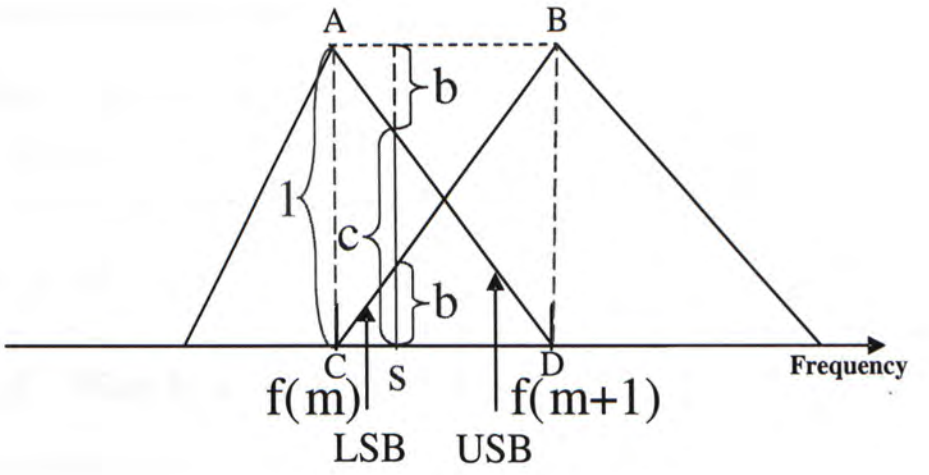


Figure 5.1: Filterbank illustration

A sample s falls in the upper side band (USB) of filter m and the lower side band (LSB) of filter $m+1$ where $f(m)$ and $f(m+1)$ are the centers of these two filters. It is obvious that $\triangle ACD \cong \triangle BDC$. Therefore

$$\begin{aligned} b + c &= 1 \\ c &= \frac{f(m+1)-s}{f(m+1)-f(m)} \end{aligned} \tag{5.5}$$

We only need to process sample s in the USB (or LSB) of the filter m (or filter $m+1$). From Eq. 5.5, the output of s from filter m can be computed as $|X(s)| \times c$. At the same time, the output of s from filter $m+1$ can be easily obtained by $|X(s)| - |X(s)| \times c$. For all the N samples in one frame, N multiplications can be reduced. We can also pre-store the N weights in RAM to further save computation.

DCT Implementation

After computing the logarithmic energy, Discrete Cosine Transformation is implemented to decorrelate these coefficients. As mentioned before, trigonometric function is computationally intensive on fixed-point DSP chips. To avoid these cosine calculations as required in Eq. 2.12, a look-up table is set up. It can be efficiently interpolated using the algorithm in Appendix A.

A lookup table with 132 parameters in 16-bit fixed-point format is used to achieve enough resolution.

Cepstral weighting

In Eq. 2.13, the weight $1 + \frac{M}{2} \cdot \sin\left(\frac{k\pi}{M}\right)$ is not related to the cepstral coefficient $C(k)$. Therefore we can pre-compute the weight sequence $\left\{1 + \frac{M}{2} \cdot \sin\left(\frac{k\pi}{M}\right)\right\}, 0 \leq k < M$ and pre-store it in the RAM so that M sine functions and M multiplications are reduced.

5.3.2 Viterbi search module

The computational load of the Viterbi search is mostly contributed by the computation of $b(\vec{o})$, which has to be calculated for each frame and each state. Suppose that $b(\vec{o})$ is a mixture of Gaussian density functions, i.e.

$$b(\vec{o}) = \sum_{m=1}^{M} b^{(m)}(\vec{o}) \tag{5.6}$$

where M is the number of mixtures and $b^{(m)}(\vec{o})$ is the mth mixture component given as,

$$b^{(m)}(\vec{o}) = c_m \cdot \frac{1}{\sqrt{(2\pi)^D \prod_{d=1}^{D} \sigma_{m,d}^2}} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(o_d - \mu_{m,d})^2}{\sigma_{m,d}^2}\right) \tag{5.7}$$

where o_d is the dth component of $\vec{o}$, $\mu_{m,d}$ and $\sigma_{m,d}$ denote the mean and variance of the dth dimension for the mth mixture component, D is the dimension of the feature vector, c_m denotes the weight for the mth mixture component and $\sum_{m=1}^{M} c_m = 1$.

For dynamic range reduction, the logarithm of $b(\vec{o})$ is often used. The direct computation of $\ln b(\vec{o})$ based on Eq. 5.6 is computationally prohibitive and approximation is needed.

Consider M=2, then we have $\ln b(\vec{o}) = \ln\left(b^{(1)}(\vec{o}) + b^{(2)}(\vec{o})\right)$ which can be re-written as

$$\tilde{b} = \max\left(\tilde{b}^1, \tilde{b}^2\right) + \ln\left[1 + \exp\left(min\left(\tilde{b}^1, \tilde{b}^2\right) - max\left(\tilde{b}^1, \tilde{b}^2\right)\right)\right] \tag{5.8}$$

where $\tilde{b} = \ln b(\vec{o})$, $\tilde{b}^1 = \ln b^{(1)}(\vec{o})$ and $\tilde{b}^2 = \ln b^{(2)}(\vec{o})$.

We propose the following approximation:

if $(\max(\tilde{b}^1, \tilde{b}^2) - \min(\tilde{b}^1, \tilde{b}^2)) < threshold$, use Eq. 5.8

else, let $\tilde{b} \approx \max(\tilde{b}^1, \tilde{b}^2)$.

That is, if the two components have a large difference, we simply ignore the smaller one. Otherwise, Eq. 5.8 would be used for accurate calculation.

From Eq. 5.7, $\tilde{b}^1$ and $\tilde{b}^2$are computed as

$$\ln b^{(m)}\left(\vec{o}\right) = \ln\left\{c_m \cdot (2\pi)^{-D/2} (\prod_{d=1}^{D} \sigma_{m,d}^2)^{-1/2}\right\} - \frac{1}{2}\sum_{d=1}^{D} \frac{(o_d - \mu_{m,d})^2}{\sigma_{m,d}^2}$$

where $\ln\left\{c_m \cdot (2\pi)^{-D/2} (\prod_{d=1}^{D} \sigma_{m,d}^2)^{-1/2}\right\}$ can be pre-computed from the trained model parameters. The factor $1/\sigma_{m,d}^2$ can also be pre-computed and the division operation would become a multiplication [53].

The method we proposed can be easily extended to more than 2 mixtures. For example, if $M = 3$, we have three mixture components. We first find out the smallest two components and apply the above approximation technique. The result is then regarded as a new Gaussian component. It will be combined with the remaining component to produce the final log output probability.

5.4 Memory requirements consideration

Embedded DSP system has limited memory resource. For a particular recognition task, the memory storage required depends on the vocabulary size and the complexity of the HMMs. For the storage of HMM parameters, means and variances of the Gaussian components require most of the memory. The memory size required is approximately equal to $2 \times M \times n \times m \times D$, where M is the number of models, n is the number of active states for each model, m is the number of mixtures at each state and D is the dimension of feature vector. In an HMM, there are other parameters needed to be stored, such as weights of mixtures, GCONST and transition probability matrix. The number of mixture weights for one HMM is $n \times m$. 'GCONST' is a value calculated by multiplying the determinant of the covariance matrix by $(2\pi)^D$. For an HMM, GCONST needs $n \times m$ words. We don't need to save all the values in the transition matrix

because in this research the model is assumed to be a left-right word model with no skipped state. Only $2 \times n$ values in the transition matrix are stored. So there are $2 \times n \times m \times D + 2 \times n \times m + 2 \times n$ words stored for one HMM model.

For the front-end part, the major memory requirement comes from storage of look-up table for DCT and gains of filterbanks, and some pre-computed parameters such as the weights of Hamming windows, the center frequencies of each filterbank and cepstral weights. Table 5.2 shows the memory storage for these parameters.

Lookup table	Pre-computed parameters			
DCT	Hamming window	cepstral weights	filterbank center	cepstral lifter
M_0	N_0	N/2	M_1	K

Table 5.2: Memory requirements (in words) for front-end

where M_0 is the size of cosine lookup table for DCT, N_0 is frame size, N is FFT points, M_1 is the number of filterbanks and K is the number of first-order cepstral coefficients. The total memory size for front-end is about $N/2 + M_0 + N_0 + M_1 + K$.

Chapter 6

Experimental results and performance analysis

Using the optimization techniques described in Chapter 5, two recognition systems have been implemented for experiments. The two systems are for Cantonese isolated word recognition system and connected English digits recognition respectively. Both of them use mel-frequency cepstral coefficients (MFCCs) as the acoustic feature extracted from the front-end part and continuous density hidden Markov models (CDHMMs) as the acoustic model. For the search engine, they both use the Viterbi search algorithm.

For each system, we first compare the computation complexity and required memory size between the original un-optimized and optimized recognition algorithms. Then the recognition performance will be analyzed to illuminate that optimization methods can reduce the computation complexity without noticeable degradation of recognition performance. From the relation analysis between optimization and performance, a framework is established to assist system designers to realize a particular speech recognition system on fixed-point DSP within their limited hardware resources.

6.1 Cantonese isolated word recognition (IWR)

The vocabulary contains 11 Cantonese words. Each word has two syllables. This vocabulary is shown in Appendix B. Training data include 2200 utterances from 5 male and 5 female native speakers. 898 utterances from another 10 males and 10 female speakers are used for performance evaluation. Both training and testing utterances are recorded in the same laboratory environment at about 30 dB SNR.

The signal analysis configuration used in the front-end part is given as in Table 6.1.

A/D & Sampling	8 kHz, 16-bit linear PCM
Pre-emphasis	y[n] = x[n] - 0.97 × x[n-1]
Frame size	20 msec.
Frame shift	10 msec.
Window	Hamming window
FFT	256 points
Feature parameters	13 MFCCs+13 Δ MFCCs

Table 6.1: Analysis configuration at the front-end

Each of the 11 words is modeled by one HMM. The HMM has 8 active states and each state has two mixtures. The HMMs are trained offline by the speech recognition toolkit HTK [58].

6.1.1 Execution time

Without optimization, the speech recognition algorithms can't be run in real-time on DSP. Fig. 6.1 shows the computation load analysis of front-end before optimization. From this figure, we find that the front-end alone would require about 11 msec which exceeds the 10 msec upper bound of real-time processing.

It is also shown that FFT, computation of spectral magnitude and DCT cost computation time significantly. With the illustrated optimization procedure, the execution time of the front-end can be greatly reduced as shown in Table 6.2. We can purposefully illustrate the improvement of computation time from Fig. 6.2, which shows the computation time ratio of the front-end part after optimization. It is obvious that the execution time can be reduced to 5% by applying the optimization techniques to the front-end.

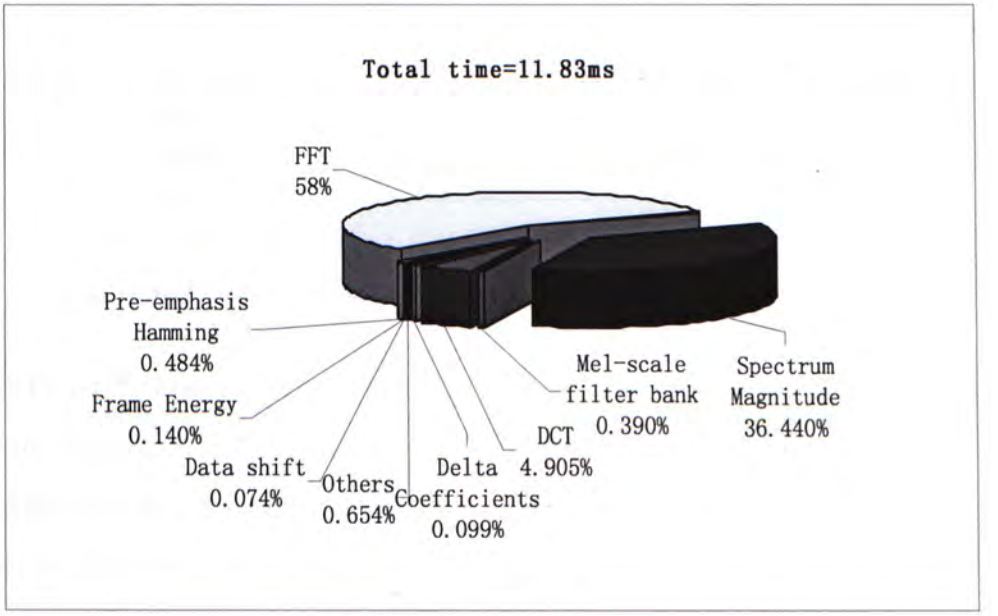


Figure 6.1: Computation time ratio of front-end before optimization

	FFT	$\|s\|$	DCT	Others	Total
Un-optimized	6.72	4.31	0.58	0.22	11.83
Optimized	0.05	0.09	0.31	0.22	0.67

Table 6.2: Execution time (in msec.) for main parts in front-end process

Consider the computation required at a particular time frame in the Viterbi search. Let the time for computing a mixture component be P and the time for path extension be Q. The theoretical estimate of the execution time for processing one frame is given by $ET_{estimated} = P \times m \times n \times M + Q \times n \times M$. Knowing that m=2, $n = 8$ and $M = 11$, $ET_{estimated} = 88 \times (2 \times P + Q)$. Table 6.3 shows

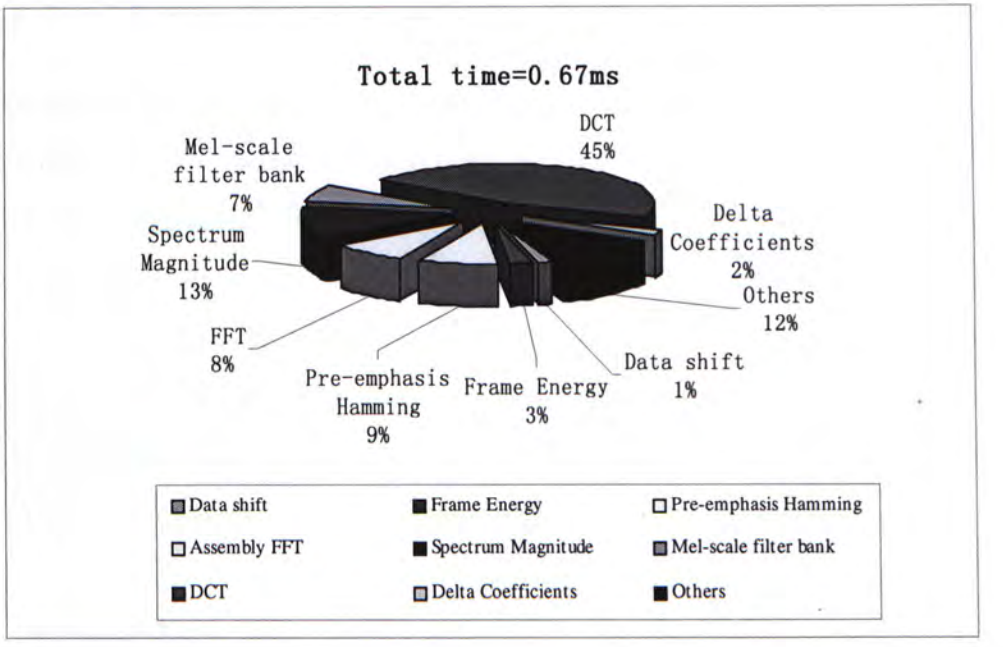


Figure 6.2: Computation time ratio of front-end after optimization

the experimentally measured values of P and Q, as well as the total execution time. The slight difference between $ET_{measured}$ and $ET_{estimated}$ is mainly due to the occasional necessity of computing add-log with Eq. 5.8. Because the measured value P is about 7 times larger than Q, we can approximately consider that $ET_{estimated}$ is equal to $P \times m \times n \times M$.

P	Q	$ETmeasured$	$ETestimated$
0.0128	0.0019	2.443	2.42

Table 6.3: Execution time (in msec.) for pattern recognition in IWR

The total execution time for one frame of speech, including both front-end and Viterbi search (estimated), is 3.090 msec. which is much shorter than the frame shift of 10 msec. The execution time would increase with the number of HMMs. To just meet the real-time requirement, the maximal affordable number of models is $(10 - 0.67)/(16 \times P + 8 \times Q) = 42$. When the number of models increases to 42, the execution time is just below 10 msec. This gives the upper bound of the vocabulary from execution time point of view.

6.1.2 Memory requirements

As described in Section 5.4, the total memory requirement for front-end is shown in Table 6.4. The memory requirement for one model is $2 \times 8 \times 2 \times 26 + 2 \times 8 \times 2 + 2 \times 8 = 880$ words. If there are 11 models, the memory for model storage will be 9680 words. It is obvious that the memory requirement for front-end is quite small compared to the model storage. And in all the experiments in this thesis, the front-end configuration is unchanged. Due to the above two reasons, we only approximately regard the model storage as the whole memory requirement in ASR systems and ignore the front-end part.

Lookup table	Pre-computed parameters				
DCT	Hamming window	cepstral weights	filterbank center	cepstral lifter	Total
132	160	128	33	12	465

Table 6.4: Memory requirements (in words) for front-end

Given the limitation of 32k words capacity of this DSK, the maximal affordable number of models is $32k/880 = 37$. Considering both the execution time and memory limitations, it is possible to extend the vocabulary size of IWR to about 37 for this particular DSP platform.

6.1.3 Recognition performance

Recognition performance depends on the complexity of the HMMs [16]. As shown earlier, the number of HMM states and the number of Gaussian mixture components at each state determine the execution time and memory requirements for the DSP implementation. We have carried out a series of recognition experiments on the IWR task with different complexity levels of HMM.

As discussed in Subsections 6.1.1 and 6.1.2, both the execution time and the memory requirements are approximately determined by the total number of mixtures $n \times m$ in each model. Generally, the more complex the model is,

the more accurate the recognition system can achieve. This is at the assumption that we have infinite training data. Practically, due to the constraints of CPU speed and limited storage, the fairly complex model can not be realized. And different combination of m and n may lead to different recognition performance. In the isolated word recognition system, if the vocabulary size is fixed, the largest number of mixtures affordable is about 66. Table 6.5 shows the recognition performance using different combination of m and n.

In this table, 'Front-end type for training' illustrates the type of feature vectors for training by HTK tool. For example, 'DSP' means that the features are obtained from the fixed-point DSP platform and they are in integer data format. And 'Front-end type for testing' means the format of features from testing utterances. Take an example, 'HTK' accounts for the HTK tool with floating-point, original algorithm. From this table, we see a clear trend of increase of recognition accuracy when the number of mixture components increases. And we can also find that the DSP-based speech recognition system attains good recognition performance with a degradation of about 2%.

The relation between model complexity and recognition accuracy can be plotted as in Fig. 6.3. This diagram serves as a useful reference for a DSP application engineer to design and implement the recognition system for a particular vocabulary and a particular hardware platform. Since the execution time is proportional to the total number of mixtures, the highest recognition accuracy among the models with the same execution time but different combination of states and mixtures per state is shown in this figure. Therefore this is also useful for an application engineer to determine the number of mixtures and states for the highest recognition performance based on their specific requirements. As a rule of thumb, given a DSP processor with 100 MIPS and 32 K word memory, the optimal selection of the model parameters are 8 states and 4 mixtures. The recognition accuracy can reach about 98% according to this figure.

no. of state	Front-end type for training	Front-end type for testing	1 mixture	2 mixtures	3 mixtures	4 mixtures
4	DSP	HTK	76.06	81.96	87.19	90.76
	HTK	HTK	78.73	88.42	88.98	92.98
	DSP	DSP	74.83	79.06	87.86	90.65
8	DSP	HTK	92.09	95.55	97.44	97.55
	HTK	HTK	91.31	96.66	97.44	98.78
	DSP	DSP	92.54	95.32	96.33	97.77
12	DSP	HTK	95.43	96.88	96.55	96.77
	HTK	HTK	97.66	98.11	98.00	98.00
	DSP	DSP	95.21	96.55	96.66	97.10
16	DSP	HTK	96.33	97.10	96.99	97.88
	HTK	HTK	97.22	98.55	98.66	98.78
	DSP	DSP	96.55	97.44	97.33	98.33
20	DSP	HTK	97.44	97.44	98.00	97.44
	HTK	HTK	98.00	98.78	99.01	99.33
	DSP	DSP	97.10	97.66	97.88	97.66
24	DSP	HTK	97.88	97.44	98.00	97.44
	HTK	HTK	98.44	98.78	99.22	99.22
	DSP	DSP	97.88	97.66	98.22	97.77
28	DSP	HTK	98.11	98.00	98.11	97.55
	HTK	HTK	98.11	98.66	99.11	99.44
	DSP	DSP	98.00	98.00	98.33	97.88
31	DSP	HTK	98.33	98.00	98.11	97.66
	HTK	HTK	98.44	99.00	99.22	99.11
	DSP	DSP	98.66	98.22	98.33	98.11

Table 6.5: Recognition accuracy (in percentage) evaluation for IWR

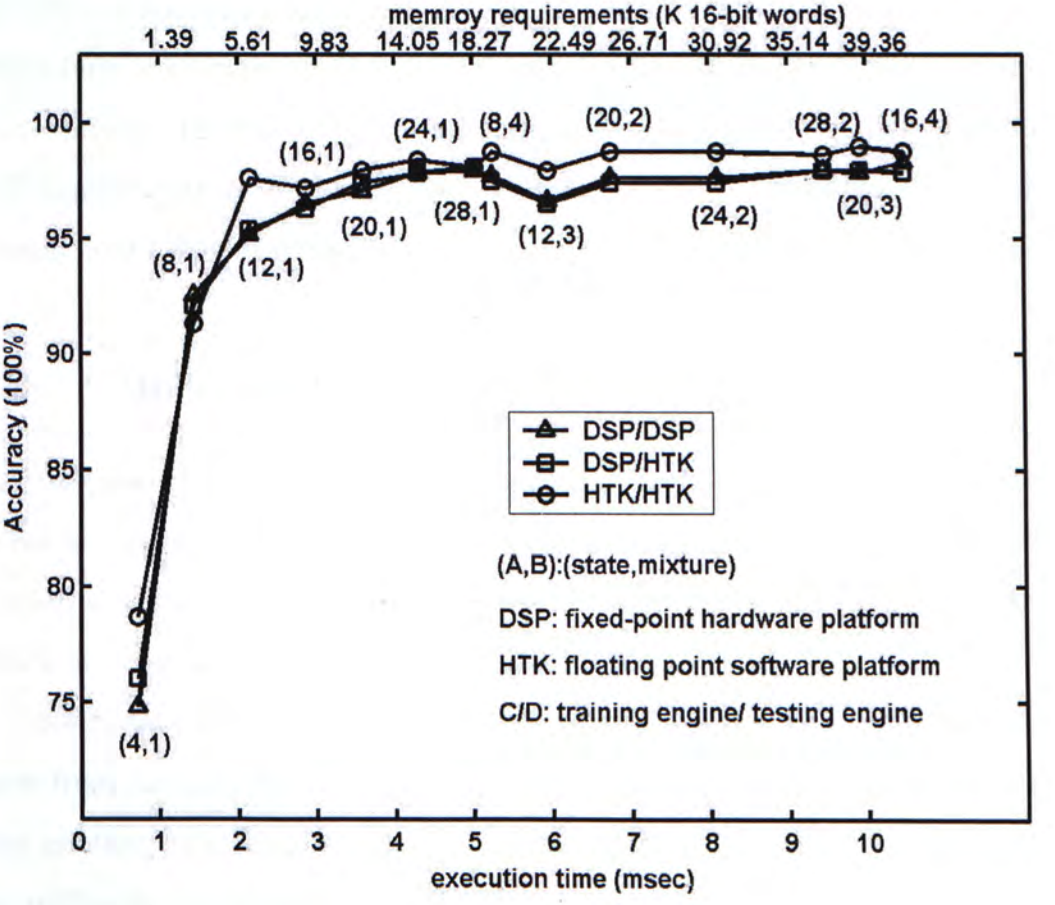


Figure 6.3: Recognition performance of the IWR

In Fig. 6.3, the curve with the circle mark illustrates the recognition accuracy of floating-point software simulation based on HTK. It can be regarded as the performance upper bound of our DSP implementation. The recognition performance represented by the rectangular mark was also obtained by HTK simulation, but the features parameters for HMM training were obtained from the fixed-point DSP implementation. The performance with such an arrangement indicates whether the proposed optimization of the Viterbi search algorithm would degrade the recognition accuracy. Lastly, the curve with the triangular mark represents the recognition accuracy attained with full DSP implementation. For the task of Cantonese isolated word recognition, the full DSP implementation attains an accuracy that is only 1% lower than that of the original floating-point algorithms.

6.2 Connected word recognition (CWR)

This task concerns the recognition of connected English digits. The vocabulary contains 11 words, including the digits "zero" to "nine", and "oh" as a pronunciation variant of "zero". And there is still a silence model to represent the silence or noise before and after the speech segment.

Speech data are from the Aurora 2 database. We use 4004 training utterances from 52 male and 52 female speakers. There are 1688 testing utterances from another 55 male and 55 female speakers. Both training and testing data are artificially corrupted with 4 different noise types in 10 dB SNR [59]. There are 1001 training and 422 testing utterances for each kind of noise in one SNR level.

The implementation of CWR shares the same front-end and HMM configuration as the IWR. Therefore the execution time for the front-end and memory requirement for HMM parameters are about the same as described in the previous subsection. The main difference between CWR and IWR is on the execution time for the pattern recognition process.

6.2.1 Execution time consideration

For the CWR, extra search time is needed at the entry state of an HMM to determine whether the path extension comes from the same entry state or the exit state of another HMM. For IWR, this is not required because the path extension must be within the same model.

Table 6.6 shows the execution time for the pattern recognition process in the CWR task, where $Q1$ denotes the execution time of path extension for an entry state and $Q2$ is the execution time of path extension for a non-entry state. The theoretical estimate of the execution time for processing one frame is $ET_{estimated} = P \times m \times n \times M + Q1 \times M + Q2 \times (n-1) \times M$. In contrast with Table 6.3, CWR requires only slightly longer execution time than IWR.

P	$Q1$	$Q2$	$ETmeasured$	$ETestimated$
0.0128	0.0027	0.0019	2.642	2.650

Table 6.6: Execution time (in msec.) for pattern recognition in CWR

6.2.2 Recognition performance

We also build up the relation between model/computational complexity and recognition performance for the connected word recognition task. Fig. 6.4 is the relation between model complexity and recognition accuracy.

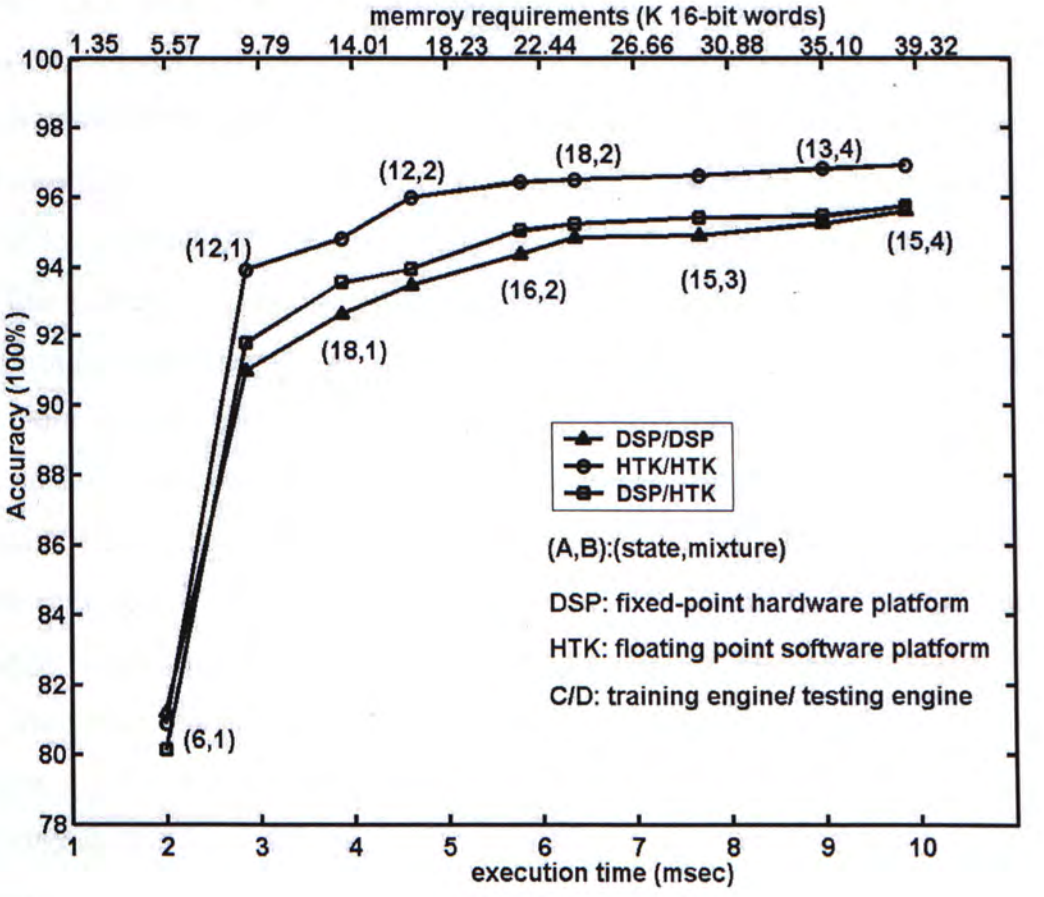


Figure 6.4: Recognition performance of the CWR

It shows that if we push the execution time per frame to be less than 3 msec., the recognition accuracy would drop dramatically due to the insufficient representation of the acoustic property in the model. On the other hand, the recognition accuracy remains high and can not be further improved if the required execution time exceeds 5 msec. Thus if one want to set up a recognition system for a similar task, 12 states per model and 2 mixtures per state or 16 states per model and 2 mixtures per state would be an appropriate choice. It can retain high recognition accuracy and save computation time for other usage like incorporating other robust speech processing algorithms. It also allows the use of lower-speed CPU or smaller memory for commercial considerations.

The 'performance saturation' phenomena is mainly due to the limited training data for a practical system. Theoretically, if there are infinite training data, the more complex the models are, the higher recognition accuracy the system can achieve. In practice, the recognition accuracy will not grow unlimitedly because the amount of training data is always limited. With this limitation, the mixture components can not be effectively trained if the number of components is large. Thus, the recognition accuracy can not be increased any more.

From Fig. 6.4, we find that there is about 2% degradation of recognition accuracy between the floating-point SR and the optimized fixed-point SR. This 2% degradation is considered not very critical for real-time speech recognition system in real-world operation.

To better understand the contributions of individual optimization techniques, a series of simulation experiments have been carried out and the key findings are summarized as in Table 6.7. We have the following observations:

- The major degradation is caused by the floating-point to fixed-point conversion. This degradation is inevitable for implementation on fixed-point processor;

- By using the assembly code to optimize the FFT algorithm, the recognition accuracy can be increased by 0.12% compared with the un-optimized fixed-point *C* language one. The assembly code not only runs faster, but also gives better precision than the un-optimized fixed-point *C* language

FFT code;

- When applying the optimization of spectral magnitude calculation to the fixed-point un-optimized SR system, the recognition accuracy will have 0.07% degradation due to the approximation of square root and square operations;
- When optimizing the output probability calculation in the back-end, the recognition accuracy will decrease for about 0.07%.

From this analysis, it is shown that for the computation optimization methods, the optimization for spectral magnitude calculation mostly affects the recognition accuracy. This result is based on the implementation of optimization *Method2* in magnitude computation. We compare the relation between the computation time and the recognition accuracy for the different methods of magnitude calculation discussed before in Table 6.8. From this table, we can verify that using *Method2* can not only remain recognition accuracy, but also save the computation time.

	Computation optimization			
	Front-end		Back-end	Accuracy
	FFT	Mag.		
floating-point	no	no	no	96.44
fixed-point	no	no	no	94.78
	yes	yes	yes	94.40
	no	no	yes	94.71
	no	yes	no	94.71
	yes	no	no	94.90

Table 6.7: Performance identification (in percentage) for each major individual part in SR (yes: optimized, no: unoptimized)

	Computation time (msec)	Recognition accuracy (%)
unoptimized	2.6	94.78
optimized using method 1	0.46	94.75
optimized using method 2	0.08	94.71
optimized using method 3	0.075	94.54

Table 6.8: Performance comparison of different optimization methods for spectrum magnitude computation

6.3 Summary & discussion

With a thorough and in-depth understanding of the ASR algorithms, we optimize the complex parts of these algorithms under the considerations of execution time and memory requirement on DSP. It shows that the computation cost of both the front-end and the pattern recognition algorithms can be reduced significantly so that real-time ASR is achievable without noticeable degradation of recognition performance. Furthermore, an IWR and a CWR system are successfully implemented and a series of recognition experiments are carried out to reveal the tradeoff between resources requirement and recognition performance. A reference framework is therefore provided for a simple and quick design of ASR applications on DSP.

Chapter 7

Implementation of practical techniques

In this chapter, we will mainly discuss about how to implement two practical techniques, end-point detection (EPD) and spectral subtraction (SS), in real-time applications. Some modifications will be made to enhance the performance of these techniques. A set of experiments are carried out to evaluate the improvement of recognition performance in detail. At last, some results and conclusions will be given.

7.1 End-point detection (EPD)

As explained in Chapter 2, energy-based end-point detection is computational effective to be implemented on low-cost embedded systems. In order to achieve a better detection of the speech, a set of parameters should be adjusted for a specific recognition task. For example, what are the energy threshold levels to judge the beginning and ending of speech activities? How many high-energy frames does it take to qualify for utterance starting? How many low-energy frames for utterance end?

The proposed energy-based end-point detection procedure consists of four steps:

1) Estimate the background noise energy by averaging the energies from the

first several frames;

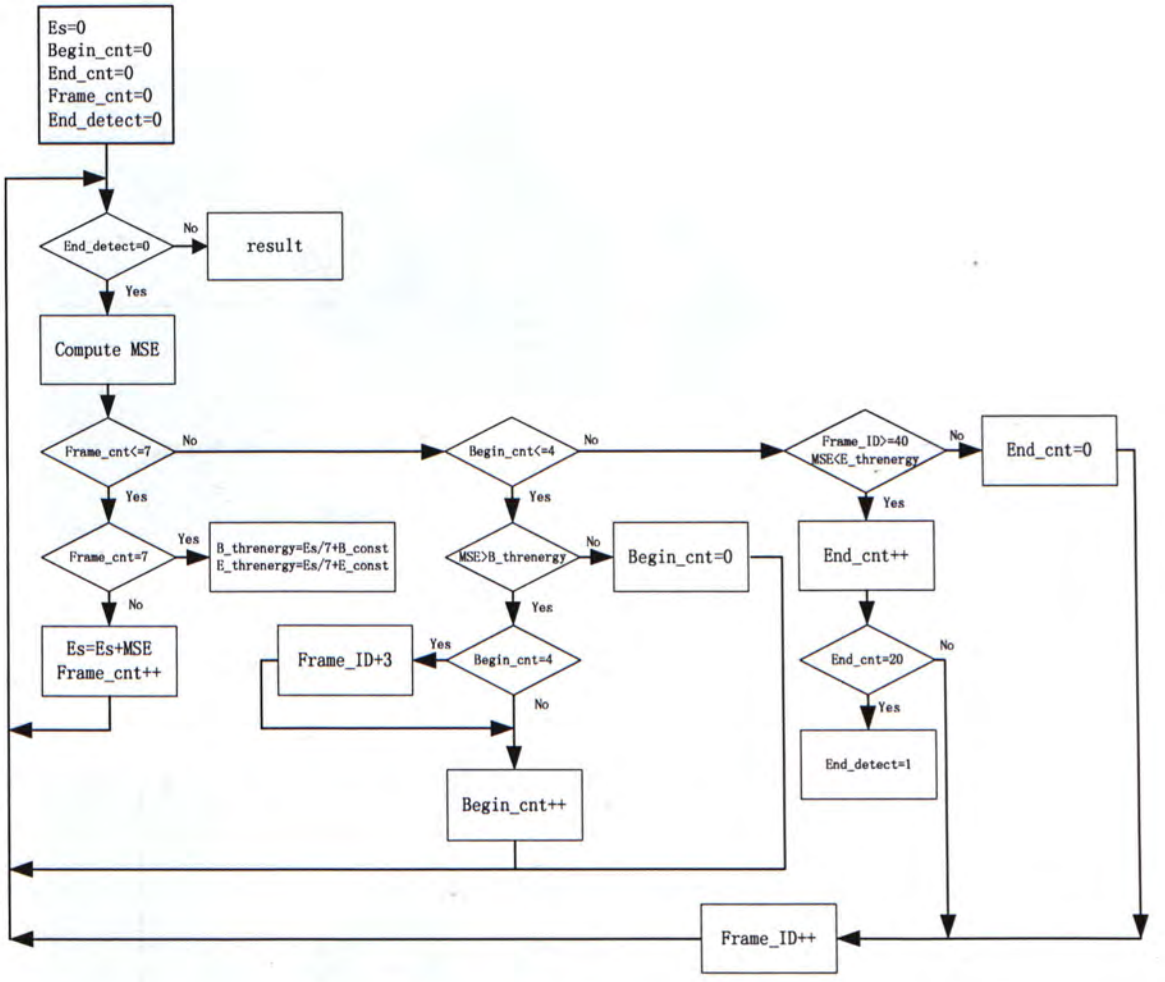
2) The beginning and ending threshold are defined by multiplying different factors to the estimated noise energy;

3) The beginning of the real speech will not be detected until the energies from a set of successive frames are bigger than the beginning threshold;

4) The ending of the real speech will be declared if the detected speech utterance is longer than a certain period and the energies from a series of successive frames are smaller than the ending threshold.

The detailed EPD procedure employed in this research is shown in Fig. 7.1.

In this figure, energies are represented in logarithm to reduce the dynamic range. After computing the reference noise energy Es from the first 7 frames which are assumed to be non-speech, the beginning energy threshold ($B\_threnergy$) and the end energy threshold ($E\_threnergy$) are obtained by multiplying Es with different constants B_const and E_const. After a certain number of successive frames pass the energy threshold, the recognizer notifies to start recognizing the speech signal. In order to better estimate the averaged noise power $Es/7$, a certain number of frames should be used as noise at the beginning. The counter $Begin\_cnt$ should be carefully determined. Too long to determine the beginning of real speech will cause long delay which is harmful to real-time processing. On the other hand, too little counter will introduce more judging error because short clip noise may be decided to be real speech. The ending determination counter End_cnt can not be set too large or too little either. Too large End_cnt will include more noise in the detected speech while too little End_cnt may lose real speech if there is a long silence or pause period during the whole speech part. The same methods should be considered when adjusting two energy thresholds, $B\_threnergy$ and $E\_threnergy$. If the thresholds are set too high, the important speech information will be lost (Fig. 7.2).



Es	:	accumulated MSE for noise estimation
Begin_cnt	:	counter for detecting the beginning of speech
End_cnt	:	counter for detecting the ending of speech
Frame_cnt	:	frame counter for noise estimation
End_detect	:	judgment for completion of end-point detection
B_threnergy	:	energy threshold for beginning of speech
E_threnergy	:	energy threshold for ending of speech
B_const	:	constant for weighting the beginning threshold
E_const	:	constant for weighting the ending threshold
Frame_ID	:	number of frames detected as speech

Figure 7.1: End-point detection algorithm

Otherwise, more useless information (noise) will be included in if the thresholds are too low (Fig. 7.3).

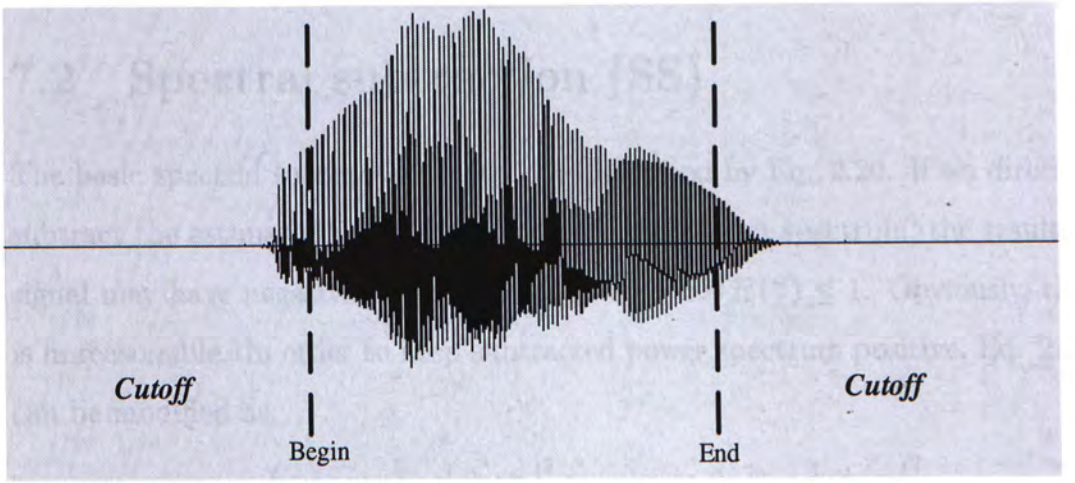


Figure 7.2: Overvalued energy threshold for EPD

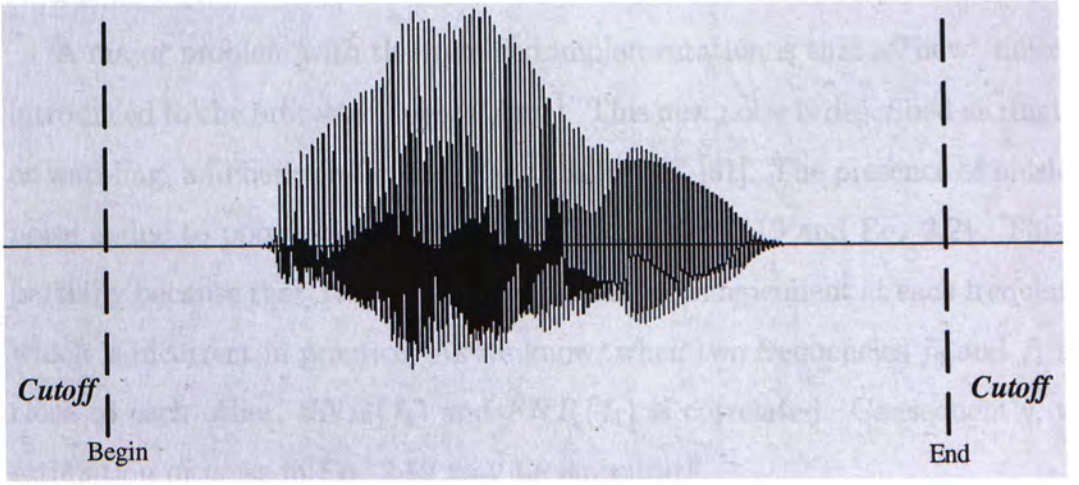


Figure 7.3: Undervalued energy threshold for EPD

Since it is not an absolute requirement that the end-point detection offers very accurate end-point in this research, the parameters (such as *B_threnergy*, *E_threnergy* and *End_cnt*) can be adjusted loosely to keep a low rejection rate (i.e., speech segments should not be detected as silence/noise segments). Any false rejection leads to an error to the recognizer. On the other hand, a possible

false acceptance (i.e., the EPD interprets noise segments as speech segments) may be rescued by the speech recognizer later if the recognizer has appropriate noise models.

7.2 Spectral subtraction (SS)

The basic spectral subtraction method is described by Eq. 2.20. If we directly subtract the estimated noise spectrum from the speech spectrum, the resulted signal may have negative power spectrum when $SNR(f) \leq 1$. Obviously, this is unreasonable. In order to keep subtracted power spectrum positive, Eq. 2.20 can be modified as:

$$\left|\hat{X}(f)\right|^2 = \begin{cases} |Y(f)|^2 - \left|\hat{N}(f)\right|^2, & if\ |Y(f)|^2 > \left|\hat{N}(f)\right|^2 \\ a, & otherwise \end{cases} \tag{7.1}$$

where $a \geq 0$ is the floor of spectral value. This is also known as half-wave rectification (HWR) [60].

A major problem with the modified implementation is that a "new" noise is introduced to the processed speech signal. This new noise is described as ringing or warbling, and hence referred to "musical noise" [61]. The presence of musical noise is due to poor estimate of $SNR(f)$ from Eq. 2.19 and Eq. 2.21. This is partially because that $SNR(f)$ is supposed to be independent at each frequency which is incorrect in practice. As we know, when two frequencies f_0 and f_1 are close to each other, $SNR(f_0)$ and $SNR(f_1)$ is correlated. Consequently, the estimation of noise in Eq. 2.19 may be inaccurate.

There are several methods to reduce musical noise. Full-wave rectification (FWR) is one of methods described as the following:

$$\left|\hat{X}(f)\right|^2 = \begin{cases} |Y(f)|^2 - \left|\hat{N}(f)\right|^2, & if\ |Y(f)|^2 > \left|\hat{N}(f)\right|^2 \\ \left|\hat{N}(f)\right|^2 - |Y(f)|^2, & otherwise \end{cases} \tag{7.2}$$

In the following experiments, we use both HWR and FWR. Comparative results show that better recognition performance in noisy environment can be achieved by using FWR than using HWR [62] [63].

7.3 Experimental results

In this section, two speech recognition tasks, namely, isolated English digit recognition (IWR) and connected English digit recognition, are experimented. The speech data are all extracted from the Aurora 2 database [59]. We will mainly analysis and discuss the effect of speech enhancement technique to the speech recognition performance at different SNR. Different parameters will be chosen in each task.

7.3.1 Isolated word recognition (IWR)

For the isolated word recognition task, we use the single-digit utterances from Aurora 2 for both training and testing. There are 1144 training utterances from 52 male and 52 female native speakers, and 474 testing utterances from another 55 male and 55 female speakers. Both training and testing data are artificially corrupted with 4 different types of noise at 10dB SNR.

- Suburban train;
- Crowed of people (babble);
- Car;
- Exhibition hall.

The signal analysis configuration used in the front-end part is exactly the same as the one in Table 6.1. For the recognition part, Hidden Markov Models and Viterbi search engine are used for decoding. Each HMM model contains 8 active states. In the following experiments, the number of mixtures in each state is changed from one to four.

Performance evaluation of end-point detection

As discussed in previous sections, implementing the end-point detection algorithm needs to adjust a number of parameters. In this task, we set these parameters as in Table 7.1

Frame_cnt	7
B_const	2
E_const	12
Begin_cnt	4
End_cnt	20

Table 7.1: EPD Parameters for IWR

These values are not very strict because they are obtained from experiments. Practically, this is one of the best settings we have tested. By setting these values, we can retain the full speech segment without losing useful information, especially for the clean speech environment as seen in Fig. 7.4.

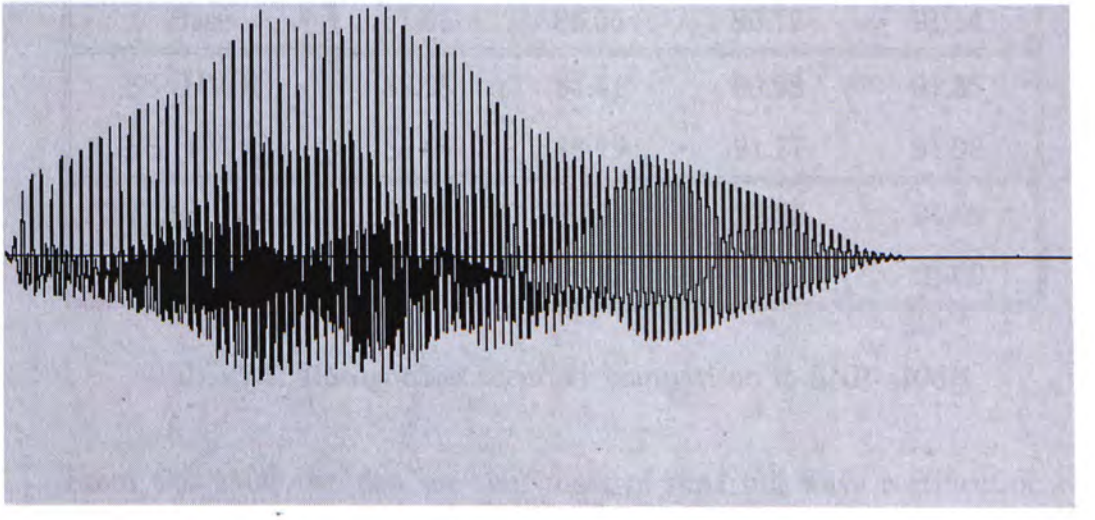


Figure 7.4: EPD with proper energy threshold

For high SNR, this end-point detection algorithm may be efficient. But when SNR is lower than 10dB, this algorithm is becoming poor. This is a challenge in the end-point detection algorithms. Here we only implement this simple one.

Table 7.2 shows the comparison of recognition performance between the base line system (without EPD) and the one with EPD.

	1 mixture	2 mixtures	3 mixtures	4 mixtures
Base	81.01	85.65	90.72	91.14
EPD	87.35	90.72	91.35	93.25

Table 7.2: Recognition accuracy (%) in SNR=10dB

Performance evaluation of spectral subtraction

In spectral subtraction, we assume that the first 7 frames be non-speech segments which can be used for estimating $\left|\hat{N}(f)\right|^2$. Half wave rectification (HWR) and full wave rectification (FWR) are used in this experiment. Comparison of these two methods is shown in Table 7.3.

	1 mixture	2 mixtures	3 mixtures	4 mixtures
Base	81.01	85.65	90.72	91.14
SS (HWR)	82.28	84.81	90.93	91.35
SS (FWR)	85.65	88.19	91.77	91.98
EPD_SS (HWR)	89.03	90.93	92.83	94.09
EPD_SS (FWR)	88.61	90.72	93.04	94.30

Table 7.3: Recognition accuracy comparison in SNR=10dB

From this table, we can see that most of time full wave rectification can achieve higher recognition performance than half wave rectification. We also find that the performance is much better when implementing both end-point detection and spectral subtraction than only implementing spectral subtraction. If spectral subtraction is implemented to a long period of noise, more residual noise will be introduced by wrongly subtracting the estimated noise from the actual noise spectrum.

7.3.2 Connected word recognition (CWR)

The connected word recognition task also uses Aurora 2 database. It is exactly the same as the task in Chapter 6 except that the SNRs here cover from 5 to 20dB and as well as the clean data. In this task, we implement the end-point detection and spectral subtraction techniques in the front-end part of the recognition system.

The parameters in EPD are set as in Table 7.4.

Frame_cnt	7
B_const	2
E_const	8
Begin_cnt	4
End_cnt	50

Table 7.4: EPD Parameters for CWR

In connected word recognition, there are one or more (up to 7) digits in an utterance. Between each pair of two adjacent digits, there may exist a short pause. It will make the end-point detector wrongly judge the end of an utterance if the parameter *End_cnt* is shorter than a short pause. We found that the short pause between two digits in Aurora database is always shorter than 45 frames. Therefore, we set the *End_cnt* to 50 to satisfy that no digit is lost after end-point detection. However, this long *End_cnt* will also cause the tail of an utterance contains longer noise which will reduce the recognition performance after spectral subtraction. Other people's work showed that a short pause model should be used between two digits [59]. But in our current implementation, we restrict that all the models have the same number of states and mixtures. Therefore it does not allow the use of a short pause model. In the future work, we may improve the implementation by using short pause model.

Evaluation of recognition performance

In order to evaluate the robustness of the speech recognition system with speech enhancement techniques, we use clean speech data to train a clean model by HTK and use multi-condition speech data to evaluate the recognition performance. The recognition accuracy is listed in Table 7.5 for recognition with or without spectral subtraction (SS). The result shows that spectral subtraction can increase the recognition accuracy while the SNR is lower than 10dB. Performance measure for the whole test set has been introduced as average over all noises and over SNRs between 5 and 20dB plus clean. This average accuracy is 87.09% for the whole test of spectral subtraction.

SNR/dB	subway(n1)		babble(n2)		car(n3)		exhibition(n4)		Average	
	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS
clean	98.90	98.54	98.55	97.78	98.92	98.49	98.95	98.57	98.83	98.35
20	95.51	96.22	96.93	96.59	98.10	97.63	96.74	95.90	96.82	96.59
15	89.67	90.47	92.79	91.53	95.26	93.84	91.78	90.90	92.38	91.69
10	77.33	79.56	82.92	82.50	84.72	84.95	80.05	79.78	81.26	81.70
5	64.23	66.38	67.97	67.29	68.22	70.83	61.85	64.05	65.57	67.14
Average	85.13	86.23	87.83	87.14	89.04	89.15	85.87	85.84	86.97	87.09

Table 7.5: Recognition accuracy (in percentage) of clean training CWR (no SS/SS)

The recognition result for spectral subtraction plus end-point detection is listed in Table 7.6. This table shows that for the noises that are approximately statistically stationary, end-point detection and spectral subtraction methods can enhance the recognition performance. For non-stationary noises, these methods can not effectively estimate the background noise so that the recognition accuracy will drop. And EPD is not good in noisy speech recognition. Overall speaking, the accuracy is slightly higher than the one without

spectral subtraction.

SNR/dB	subway(n1)		babble(n2)		car(n3)		exhibition(n4)		Average	
	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS	$\overline{SS}$	SS
clean	98.8	98.78	97.96	98.03	98.60	98.58	98.54	98.59	98.48	98.50
20	91.64	94.86	95.43	94.82	97.06	96.67	94.37	94.51	94.63	95.22
15	84.10	89.90	89.02	86.84	92.19	91.36	87.86	88.61	88.29	89.18
10	69.07	76.54	74.15	71.64	78.11	76.53	72.32	73.01	73.41	74.43
5	54.18	58.34	55.14	49.32	55.24	52.41	51.58	49.34	54.04	52.35
Average	79.56	83.68	82.34	80.13	84.24	83.11	80.93	80.81	81.77	81.93

Table 7.6: Recognition accuracy (in percentage) of clean training CWR (EPD, no SS /EPD with SS)

7.4 Results

For a practical implementation of speech recognition system, the type of noise and the noise level are always unknown or not what we have expected. Therefore, noise robust techniques are essentially useful in such conditions. Spectral subtraction, one kind of speech enhancement techniques, has been implemented in this work, which is possible for real-time operation on DSP due to its simple computation. The experiment results shown in this chapter reveal that spectral subtraction with full wave rectification can increase the recognition accuracy in different tasks.

In this chapter, we also describe a simple end-point detection technique based on frame energy. To get the right beginning and ending place of the speech part, some parameters should be adjusted. Although this end-point detection can work well in high SNR environment, the performance will drop down if the SNR is too low. This can be found in the experiments of this chapter.

Chapter 8

Conclusions and future work

8.1 Summary and Conclusions

Speech recognition technology has many applications on embedded systems, such as stand-alone devices and single-purpose command and control systems. The development of such applications is not straightforward. The highly complex ASR algorithms have to be optimized to meet the limitations in computing power and memory resources. The optimization, which typically involves simplification and approximation, inevitably leads to the loss of precision and the degradation of recognition accuracy. This thesis describes the exploitation of state-of-the-art ASR techniques for DSP-based embedded applications. The complex algorithms have been optimized for real-time processing on a specific DSP platform. An isolated-word recognition system and a connected-word recognition system have been successfully developed. More importantly, the thesis provides a thorough and detail analysis for individual computational steps and suggests many optimization methods for computation reduction. The use of these methods reveals the underlying principles of optimization for different kinds of computation. The suggested methods are applicable not only to DSP, but also to other embedded platforms. It is anticipated that this thesis can serve as a useful reference for the design and implementation of embedded ASR applications.

Two speech recognition systems have been successfully realized on

TMS320VC5416 DSP. For the recognition of 11 Cantonese isolated words, the optimized algorithms can be run about three times faster than real time with only 2% degradation of recognition accuracy. For a 10 msec. speech frame, only 3 msec. processing time is needed. For the connected English digits recognition, similar performance can be achieved. Only 3.1 msec. is needed within a 10 msec. frame.

Two practical techniques, namely end-point detection (EPD) and spectral subtraction (SS), are implemented in the ASR systems. Experimental results show that the recognition performance with EPD can be increased by 5.07% for IWR task with 2 mixtures per state and 0.12% for CWR task. When both EPD and SS are applied, the recognition performance shows a 5.28% absolute improvement for IWR with 2 mixtures per state and 0.16% improvement for CWR.

Performance analysis shows that the execution time or memory requirement is approximately proportional to the number of mixtures in HMMs, which determines the model complexity. Theoretically, the recognition accuracy would keep increasing with the increase of model complexity, if we have infinite amount of training data. In practice, the recognition accuracy will not grow unlimitedly because training data are always limited. With limited training data, the mixture components can not be efficiently trained if the number of the components is too large. For example, in the IWR task, the recognition accuracy will stop growing when the execution time reaches to 4 msec. and above.

By optimizing the speech recognition algorithms, the computation time for both front-end and pattern recognition has been efficiently reduced. We investigated the front-end and back-end parts in detail and found out the elements that cost most of the computation. In the front-end, FFT, spectral magnitude, DCT and Hamming window cost a lot of computation. Experimental results show that the optimization methods have reduced the computation time for front-end part to 0.67 msec., which is about 20% computation load of the entire recognition system.

On the other hand, the execution time for the back-end is proportional to

the complexity of model. The more complex the model is, the more execution time and memory will be required. Experimentally, 2.5 - 4 msec. of computation cost for the back-end is adequate to give good recognition accuracy. The memory requirement for storing the model parameters is the most intensive load of the whole system, and it is approximately proportional to the total number of states and models which are crucial to the recognition performance. We have investigated the relation between the model complexity and recognition accuracy. Experimental result shows that if the total number of mixtures is larger than 16, the recognition performance will not increase very much for the isolated word recognition system.

8.2 Suggestions for future research

There is potential work to be done in the future:

1) To make the system more robust to the noisy environment, especially for low SNR environment, some new modifications to the spectral subtraction method may be researched for low-cost applications.

2) As we can see, the end-point detection used in this work is only based on the frame energy which is not good for a noisy environment with low SNR. The error rate of determining the beginning and ending of speech segments will greatly increase which directly influence the recognition performance at the pattern recognition part. So, we should try to use some effective way to do end-point detection. One of these methods we think may work is to use the statistical way to find a distribution which can separate the noise and speech from each other.

3) Although we have proposed many optimization methods to the complex speech recognition technology, there maybe some other optimizations that we can apply. For example, when the vocabulary size increases, we can use pruning technique to reduce the computation time for path extension.

4) In this work, we assume that the number of states and mixtures are same

for each model. This limitation is too strict. So in the future work, we should make the number different for each model, which is more flexible for use.

Appendix A

Interpolation of data entries without floating point, divides or conditional branches

Let $f()$ be the function being approximated, b be the number of bits in the unsigned input number x, and t be the number of bits used to index the table. The table will have values of $f(x)$ for $x = n \cdot 2^{b-t}$ for $n = 0, 1, 2, ..., 2^t - 1$. The index for the table entry $j \cdot 2^{b-t}$ is just j left shifted by (b - t). Thus to approximate $f(x)$ using the table we just right shift x by (b - t) bits and then index into the table.

To interpolate between table entries, let the value obtained from the lookup table be f_1 and the next higher entry in the lookup table be f_2. Obviously $f(x)$ is in between f_1 and f_2. To interpolate we use standard linear interpolation and get the equation

$$f_{approx}(x) = \frac{(f_2 - f_1)(x - x_{trunc})}{2^{b-t}} + f_1$$

where x_{trunc} is x with the lower b-t bits zeroed out. Note that $x - x_{trunc}$ is just the lower *(b-t)* bits of x which can be implemented as a bitwise AND. The division is by a power of two so it is just a shift. It may be necessary to distribute the shift by *(b-t)* into two operations to prevent overflow in computing the numerator. The index for f_2 is just one more than the index for f_1.

With this method we can approximate any function with an interpolated lookup table without divides or conditional branches.

Appendix B

Vocabulary for Cantonese isolated word recognition task

Vocabulary

A-廚房	B-客廳	C-飯廳	D-睡房
E-廁所	F-電燈	G-冷氣	H-音響
I -HiFi	J-電視	K-風扇	

Bibliography

[1] B.-H. Juang and S. Furui: " Automatic recognition and understanding of spoken language - a first step toward natural human-machine communication", *Proceedings of the IEEE*, volume: 88, pages: 1142-1165, 2000.

[2] B. H. Juang and Tsuhan Chen:" The past, present and future of speech processing", *Signal Processing Magazine, IEEE*, volume: 15, issue: 3, pages: 24-48, May 1998.

[3] Sensory Technologies:" Sensory speech products", http://www.sensoryinc.com/.

[4] Conversay Speech Technology Solutions, " Conversay Advanced Symbolic Speech Interface (CASSI)".

[5] Lim Hong Swee:" Implementing speech recognition on TMS320C2xx", report of Texas Instruments.

[6] Lorenzo Cali, Francesco Lertora, Monica Besana and Michele Borgatti:" CO-design method enables speech recognition SoC", 2001.

[7] K. Davis, R. Biddulph and S. Balashek:" Automatic recognition of spoken digits", *J. Acoust. Soc. Am.*, volume: 24, pages: 637-642, 1952.

[8] J. W. Cooley and J. W. Tukey:" An algorithm for the machine computation of complex Fourier series", *Math. Comput.*, volume: 19, pages: 297-301, 1965.

[9] B. Bogert, M. Healy and J. Tukey:" The frequency analysis of time series for echos", *Proc. Symp. on Time Series Analysis*, pages: 209-243, Wiley, New York, USA, 1963.

[10] J. D. Markel and A. H. Gray Jr., *Linear Prediction of Speech*, Springer-Verlag, 1976.

[11] F. Itakura and S. Saito:" Analysis-synthesis telephone based on the maximumlikelihood method", *Proc. 6th Int. Cong. Acoust.*, Tokyo, Japan, 1968.

[12] B. Atal and S. Hanauer:" Speech analysis and synthesis by prediction of the speech wave", *J. Acoust. Soc. Am.*, volume: 50, pages: 637-655, 1971.

[13] H. Sakoe and S. Chiba:" Dynamic programming algorithm optimization for spoken word recognition", *IEEE Transactions on Audio, Speech and Signal Processing*, volume: 26, pages: 43-49, 1978.

[14] L. E. Baum and T. Petrie:" Statistical inference for probabilistic functions of finite state Markov chains", *Ann. Mathemat. Stat.*, volume: 37, pages: 1554-1563, 1966.

[15] C. Tappert, N. Dixon, A. Rabinowitz and W. Chapman:" Automatic recognition of continuous speech utilizting dynamic segmentation, dual classification, sequential decoding, and error recovery", IBM Tech. Rep. RADC-TR-71-146, Yorktown Heights, NY, USA, 1971.

[16] L.R. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Englewood Cliffs Publisher, NJ, 1993.

[17] K. F. Lee, *Automatic speech recognition - the development of the Sphinx system*, Kluwer, 1989.

[18] F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, 1997.

[19] Oliver Deroo:" A short introduction to speech recognition", *http://www.babeltech.com/download/SpeechRecoIntro.pdf.*

[20] S. Yong, M. Adda-Dekker, X. Aubert, C. Dugast, J. L. Gauvain, D. J. Kershaw, L. Lamel, D. A. Leeuwen, D. Pye, A. J. Robinson, H. J. M. Steeneken and P. C. Woodland:"Multilingual large vocabulary speech recognition: the European SQALE project", *Computer Speech and Language*, volume: 11, pages: 73-89, 1997.

[21] J. D. Markel and A. H. Gray Jr., *Linear Prediction of Speech*, Springer-Verlag Publisher, 1976.

[22] S. B. Davis and P. Mermelstein:" Comparison of parametric representations for monosyllabic word recognition in continuous spoken sentences", *IEEE Transactions on Audio, Speech and Signal Processing*, volume: 28, pages: 357-366, 1980.

[23] H. Hermansky:" Perceptual linear predictive (PLP) analysis of speech", *J. Acoust. Soc. America*, pages: 1738-1752, 1990.

[24] C. Myers and L. R. Rabiner:" Connected word recognition using a level building dynamic time warping algorithm", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume: 6, pages: 951-955, Apr 1981.

[25] L. R. Rabiner:" A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE* , volume: 77, issue: 2, pages: 257-286, Feb. 1989.

[26] R. C. Moore:" Using natural-language knowledge sources in speech recognition", *Computational Models of Speech Pattern Processing*, pages: 304-327, Ed. K. Ponting, Springer-Verlag, 1997.

[27] J. C. Junqua and J. P. Haton, *Robustness in Automatic Speech Recognition*, Kluwer Academic Publishers, 1996.

[28] B. Delaney, N. Jayant, M. Hans, T. Simunic and A. Acquaviva:" A low-power, fixed-point, front-end feature extraction for a distributed speech

recognition system", *IEEE Transactions on Audio, Speech and Signal Processing*, volume: 1, pages: 793-796, May 2002.

[29] K. K. Shin, J. C. H. Poon and N. C. Li:" A fixed-point DSP based Cantonese recognition system", *ISIE '95*, volume: 1, pages: 390-393, Jul. 1995.

[30] Bernhard Obermaier and Bernhard Rinner:" A TMS320C40-based speech recognition system for embedded applications", *Proceedings of The 2nd European DSP Education & Research Conference* , Paris, France, September 1998.

[31] Yifan Gong and Yu-Hung Kao:" Implementing a high accuracy speaker-independent continuous speech recognizer on a fixed-point DSP", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume: 6, pages: 3686-3689, June 2000.

[32] Xuedong Huang, Alex Acero and Hsiao-Wuen Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, Carnegie Mellon University, pages: 230-231, April 2001.

[33] S. Furui:" Speaker independent isolated word recognition using dynamic features of speech spectrum", *IEEE Transactions on Audio, Speech and Signal Processing*, volume: 34, pages: 52-59, 1986.

[34] K. Elenius and M. Blomberg:" Effects of emphasizing transitional or stationary parts of the speech signal in a discrete utterance recognition system", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages: 535-538, 1982.

[35] C. H. Lee, E. Giachin, L. R. Rabiner, R. Pieraccini and A. E. Rosenberg:" Improved acoustic modeling for speaker independent large vocabulary continuous speech recognition", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume: 1, pages: 161-164, 14-17 April 1991.

[36] J. G. Wilpon, C. H. Lee, and L. R. Rabiner:" Improvements in connected digit recognition using higher order spectral and energy features", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1991.

[37] Sadaoki Furui:" Recent advances in robust speech recognition", *ESCA-NATO Workshop on Robust Speech Recognition for Unknown Communication Channels*, page: 11-20, April 1997.

[38] Guanghui Hui, Kwok-Chiang Ho and Zenton Goh:" A robust speaker-independent speech recognizer on ADSP2181 fixed-point DSP", *International Conference on Signal Processing Proceedings*, pages:694 - 697, 12-16 Oct. 1998.

[39] Xuedong Huang, Alex Acero and Hsiao-Wuen Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, Carnegie Mellon University, page: 473-482, April 2001.

[40] G. S. Ying, C. D. Michell and L. H. Jamieson:" Endpoint detection of isolated utterances based on a modified Teager energy measurement", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages: 732-735, 1993.

[41] P. Thumpothong:" Multispeaker recognition system", *Master's thesis*, Chulalongkorn University, 1990.

[42] T. Prathumthan:" Thai speech recognition using syllable unit", *Master's thesis*,Chulalongkorn University, 1987.

[43] N. Jittiwarangkul, S. Jitapunkul, S. Luksaneeyanavin, V. Ahkuputra and C. Wutiwiwatchai:" Thai syllable segmentation for connected speech based on energy", *IEEE Asia-Pacific Conference on Circuits and Systems*, pages:169 - 172, 24-27 Nov. 1998.

[44] Xuedong Huang, Alex Acero and Hsiao-Wuen Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, Carnegie Mellon University, pages: 510-512,April 2001.

[45] Wayne A.Lea, *Trends in Speech recognition*, Prentice Hall,1980.

[46] Jean-Claude Junqua, *Robust Speech Recognition in Embedded Systems and PC Applications*, Kluwer Academic Publishers, 2000.

[47] " Reference frameworks for eXpressDSP software: RF3, a flexible, multi-channel, multi-algorithm, static system", Texas Instruments.

[48] N. Hataoka, H. Kokubo, Y. Obuchi and A. Amano:" Development of robust speech recognition middleware on microprocessor", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages: 837-840, 1998.

[49] J. J. Labrosse:" Fixed-point airthmetic for embedded systems", *C/C++ Users Journal*, pages: 21-28, 1998.

[50] Batlle and E. Fonollosa, J.:" Computational cost and memory requirements for real-time speech recognition systems", *Proceedings of International Conference on Signal Processing Applications and Technology*, pages: 1730-1734, Boston, USA, 1996.

[51] R. Boman:" Fixed point implementation of common signal processing algorithms", *International Conference on Signal Processing Applications and Technology*, pages: 716-720, 1997.

[52] Wei Han, K. W. Hon, C. F. Chan, Tan Lee, C. S. Choy, K. P. Pun and P. C. Ching:" An HMM based speech recognition IC", *Proceedings of the International Symposium on Circuits and Systems*, pages: 744-747, Bangkok, May 2003.

[53] M. J. Flaherty and T. Sidney:" Real-time implementation of HMM speech recognition for telecommunications applications", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages: 145-148, Apr. 1994.

[54] "TMS320C54x DSP applications guide reference set volume 4", Texas Instruments, 1996.

[55] "Optimized DSP library for C programmers on the TMS320C54x", Texas Instruments, 2000.

[56] Robert C. Boman:" Integer implementation of a perceptual based acoustic front-end for robust speech recognition in additive and convolutional noise", *International Conference on Signal Processing and Application Technologies*, 1997.

[57] Wan Chun Kit:" Implementation of HMM based speech recognition on a general purposed DSP chip", *undergraduate's thesis*, 1999.

[58] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev and P. Woodland, *The HTK book version3.1*, Entropic, 2002.

[59] H. G. Hirsch and D. Pearce:" The AURORA experimental framework for the performance evaluations of speech recognition systems under noisy conditions", *Proc. ISCA ITRW ASR2000*, Sept. 2000, Paris.

[60] J. Ortega-Garcia and J. Gonzalez-Rodriguez:" Overview of speech enhancement techniques for automatic speaker recognition", *International Conference on Spoken Language Proceedings*, volume: 2, pages: 929-932, Oct. 1996.

[61] M. Berouti, R. Schwartz and J. Makhoul:" Enhancement of speech corrupted by acoustic noise", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages: 208-211, 1979.

[62] Laengler, A. Gruhler, G. Sovka, P. Pollak and P. Dav i dek, V.:" Real-time fixed-point DSP-implementation of spectral subtraction algorithm for speech enhancement in noisy environment", *9th International Czech-Slovak Scientific Conference*, pages: 186-189, 1999, Brno: University of Technology.

[63] V. Dav i dek:" Implementation of spectral subtraction modified by Wiener filtering of fixed-point DSP", *Conference Proceedings of Radioelektronika*, volume: 1, pages: 177-180, ISBN 80-214-2383-8, 2003.