ABSTRACT

CHANDRA, DHRUBA. Speech Recognition Co-processor. (Under the direction of Professor Paul D. Franzon.)

With computing trend moving towards ubiquitous computing propelled by the advances in embedded mobile processors and battery technoloqy, speech recognition is becoming an essential part of embedded processor I/O device. Speech recognition is also used in command and control and automated customer service. Real time speech recognition application is both computation and memory intensive and it overwhelms even a high end multi-gigahertz processor to achieve real time performance. An embedded mobile device cannot support real time large vocabulary speech recognition application as the processors are less aggressive because of tighter power budget. Hardware solution to speech recognition, in the past, have mainly concentrated on building specialized hardware or ASIC accelerators to run software speech application faster but have largely ignored design for large vocabulary and power reduction.

In this work, we propose a hardware-software co-design for real time large vocabulary speech recognition. Our design has custom ASIC blocks and RAM memories and a low power processor. The processor maintains a high level control over the blocks and processes parts of speech recognition application which is not computation and memory intensive. The custom ASIC computes the Gaussian probability and performs word search in the dictionary. The RAMs are used for storing the intermediate values and states. The design can handle large vocabulary speech recognition in real time on a mobile embedded device. Our word search uses innovative dictionary word layout in memory which reduces bandwidth by a factor of 11 compared to software implementation and by a factor of 4 compared to other ASIC implementation. One unit of our proposed design can perform 4x and 20x better than other proposed design of specialized hardware design for software speech application in computing the Gaussian probability and word search, respectively.

Speech Recognition Co-processor

by

Dhruba Chandra

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina

2007

Approved By:

Dr. E. Rotenberg	Dr. Robert Rodman
Dr. Paul D. Franzon Chair of Advisory Committee	Dr. W. Rhett Davis

Dedication

Dedicated to Ma (Mom), Baba (Dad), Didibhai (Sister), Saibalda (Brother-in-law) and Ria (Niece)

Biography

Dhruba Chandra was born and raised in Durgapur, West Bengal, India. He received a Bachelors and a Masters in Mathematics from Indian Institute of Technology (IIT), Kharagpur. Dhruba joined graduate program in Applied Mathematics at North Carolina State University in fall of 1999. He received a Masters in Applied Mathematics in fall of 2001 and a Masters in Computer Engineering in spring of 2002. Since spring of 2003 he has been enrolled in computer engineering doctoral program at North Carolina State University.

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Paul Franzon. His has unique way of advising students. He encourages free thinking and his valuable feedback keeps his students on track and always motivated. He has always impressed me and all other ECE graduate and undergradute students with his teaching. I am thankful to all my committee members. I would like to thank Dr. Rhett Davis for his valuable suggestions and for asking the critical questions. I thank Dr. Rotenberg for teaching computer architecture courses and letting me use the modified simplascalar simulator. I thank Dr. John Wilson and Dr. Steve Lipa for always helping the research group. I would like to thank my previous research group - Dr Solihin, Dr Seong beom Kim, Dr Mazen Kharbutli and Fei Guo.

I thank my parents for their love, support and constant encouragement. My sister has always been source of my inspiration. I would like to thank all my friends from my undergraduate days - Bibhudatta Sahoo, Chandrasekhar Puthilathe, Debasis Mishra, Mukul Khandelia, Ranadeb Chaudhuri, Shailendra Jha, Sunil Saini, Vinit Srivastava Vivek Gulati, Saurav Prasad, Sumon Chattopadhye, Rohith S., Shubhadip Niyogi, Pratik Chaudhury and Chandramauli Sastry. Thanks to Vikram Pendharkar, Paul Palathingal, Jaishankar, Sajid, Tanvir, Milind, Shamsheer, Udayan, and Shashank for friendship and tolerating me as their roommate. Thanks to the Volleyball gang and Kausar Hassan.

My graduate life story is incomplete without the tale of a coffee shop, millions

of gallon of Java sipped, camaraderie I developed with the fellow coffee drinkers and hundreds of hours invested - discussing world politics, NFL, college basketball and research. Thanks to Cup-A-Joe for being that coffee shop. Thanks to Ajit Rajagopal, Meeta Yadav, Nikola Vouk, Radha Venkatagiri, Salil Pant, Sarat Kocherlakota, Saurabh Sharma, Sibin Mohan, Surendra Jain for being the comrades. Time spent with these folks in - Lab / Coffee Shop/ Glenwood South- is priceless. Thanks to Nagendra Kumar and Milind Nemlekar for starting the evening coffee tradition. I would like to thank my friend Ravi Jenkal for his help in my research. Thanks to Ambarish Sule and Vimal Reddy for helping me in my research work with various tools and simulator. Thanks to Meeta again for helping me in getting post defense formalities done.

To all friends and folks in Raleigh, thank you for making my stay enjoyable over last eight years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Speech Recognition: Demand and Challenges

The advances in computing ability of processors, low power embedded processors and artificial intelligence are making Human-Computer Interaction (HCI) and ubiquitous computing a reality. Automatic speech recognition is an essential part of HCI and ubiquitous computing and has some attractive advantages when used as an I/O device for a system. Usually, a person can speak faster than he/she can type and it keeps his/her other senses free to do other tasks.

Automatic speech recognition is also useful in applications like automated customer service, interactive video games, and for controlling unmanned vehicle. With the advent of battery operated mobile devices like cell-phone, PDA, speech recognition can be used in a 'speech to text' manner for writing emails on PDA/cell-phone,

medical transcription etc. as the I/O (alphabet keyboard) of these battery operated mobile devices are not user friendly. Automatic speech recognition coupled with an automatic language translator can be used as language-to-language translation for tourism, business or military purposes.

Speech recognition can done with template matching [6] but has limitations on its usability. Any word spoken outside the pre-recorded templates are not recognizable. Also, recognition may be speaker's accent dependent. We now look at desirable/ideal characteristic of a speech recognition system. A speech recognition system should be as accurate as possible. Speech is different from the other I/O devices, such as a computer key board, since in most cases it needs to be speaker independent. For example, an automated customer service application may have callers with different accents. Speaker independent speech recognition cannot be done using template matching and is best handled using Hidden Markov Model (HMM), explained later in the text.

When used as an I/O (speech to text) for emails, making documents or for medical transcription the vocabulary needs to be fairly large since an ideal recognizer should identify all words spoken, which may span the entire language dictionary. The recognition should be done real-time/finite-time as otherwise the recognized speech may lose its significance. Therefore, an ideal speech recognition system should be a high accuracy real-time speaker independent Large Vocabulary Automatic Speech Recognition (LVASR) system. It should work for multiple languages and support

different dialects.

Research in academia and industry have long pursued and developed high accuracy LVASR. The popular commercial systems available in market today are Dragon naturally speaking [38], IBM's Via Voice [4], Phillips-Freespeech98 [5] etc and systems developed by academia are CMU-Sphinx [1], Cambrige-HTK [2] etc. These systems are all software solutions, available as software packages and are run on desktop/server platforms.

Speech recognition involves complex mathematical calculations involving floating point numbers and has large memory requirement. Software solutions for speech overwhelm a powerful desktop/server system because of their computational needs, large memory and high memory bandwidth requirement.

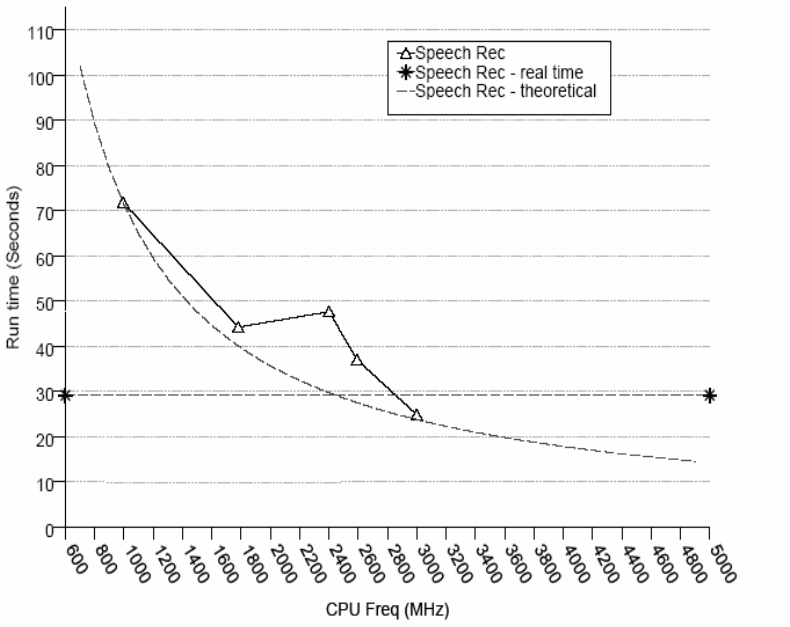

Figure 1.1: Sphinx-III Performance [27]

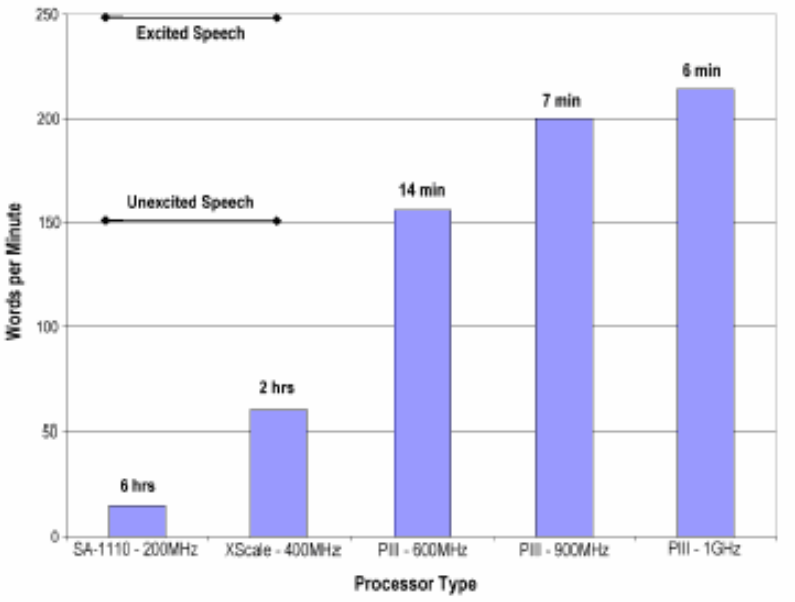

Figure 1.2: Battery Life-Words/min vs.Processor [24]

Sphinx-III developed by CMU runs 1.8x slower than real-time on a 1.7 GHz AMD Athelon [27]. The Figure 1.1 shows the processor speed and the recognition performance for 29.3 sec of speech. In theory [1], 2.4GHz is required to achieve real-time performance. These recognition systems are memory intensive, have large memory footprint, high L2 miss-rate and memory bandwidth requirement. The miss-rate of L2 cache and bandwidth requirement are shown in Table 1.1. The recognition system uses key resources of a system (execution units, caches, bus bandwidth) because of its high computational needs and high miss-rates of speech recognition and other applications running on the desktop/server system may freeze because of resource starvation. Moreover speech recognition is not scalable with processor frequency and

[1] Sphinx on Simplescalar simulator

architectural changes are required. Figure 1.1 shows the battery life of two 'AA' batteries if they power different processors running speech application.

Table 1.1: Cache size Miss-rate and Bandwidth

L2 Cache Size	2MB	4MB	8MB
Miss Rate (%)	34.39	29.42	17.9
Bandwidth (MB/s)	1502	1261	773

It implies that the speech recognition application cannot be ported to battery operated mobile devices as embedded processors (SA-1110 or X-scale) do not have the required processing power and it is impractical to have a a Pentium / Athlon like processor on a mobile device as it cannot be powered by a battery.

To make real time speech recognition work on embedded mobile enviroment, we propose a system with a general purpose microprocessor, dedicated ASICs and RAMs. We identify the computationally heavy and memory intensive parts of the speech recognition and provide a solution with dedicated ASIC and RAM structures, respectively. The microprocessor maintains a high-level control over the ASIC blocks and performs computationally non-intensive tasks of the speech recognition.

We now take a look at speech recognition in somewhat more details [2] to quickly identify the critical parts that cuases performance bottleneck (computationally heavy and memory intensive parts) and solution proposed (key contributions).

[2]Speech Recognition theory is explained in in greater details in the next chapter

1.1.1 Recognition Basics

Any words spoken in a language is composed of a sequence of basic sounds. Sound features of these basic sounds are captured and are represented as a collection of multi-dimensional statistical models. The following figure 1.3 shows the word "ONE" as sequence of basic sounds "HH-W-AH-N". These basic sounds are also called *phones*. All words in a language dictionary are sequence of basic sounds.

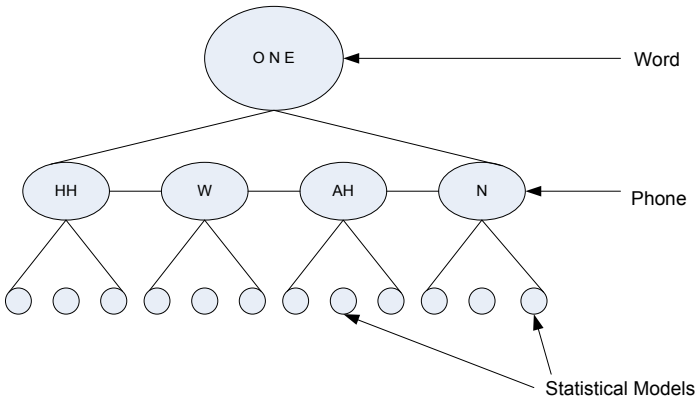

Figure 1.3: Dictionary Word, Basic Sounds and Statistical Models

The following Figure 1.4 shows how a speech recognition system works. It is divided in three broad functional blocks. Spoken words goes through Digital Signal Processing (DSP) *Frontend* and sound features of the speech are extracted. To have a real-time recognition features are extracted every 10ms. These features are then probabilistically compared with the huge database of all possible known basic sound features to identify the most likely spoken sequence of sounds. (This database is a collection of statistical models of basic sound features and is created by speech training data offline). The sequence is put together using Viterbi decoder. As each

of the dictionary words are mapped as sequence of sounds, the first part of the word search block identifies the word(s) from dictionary.

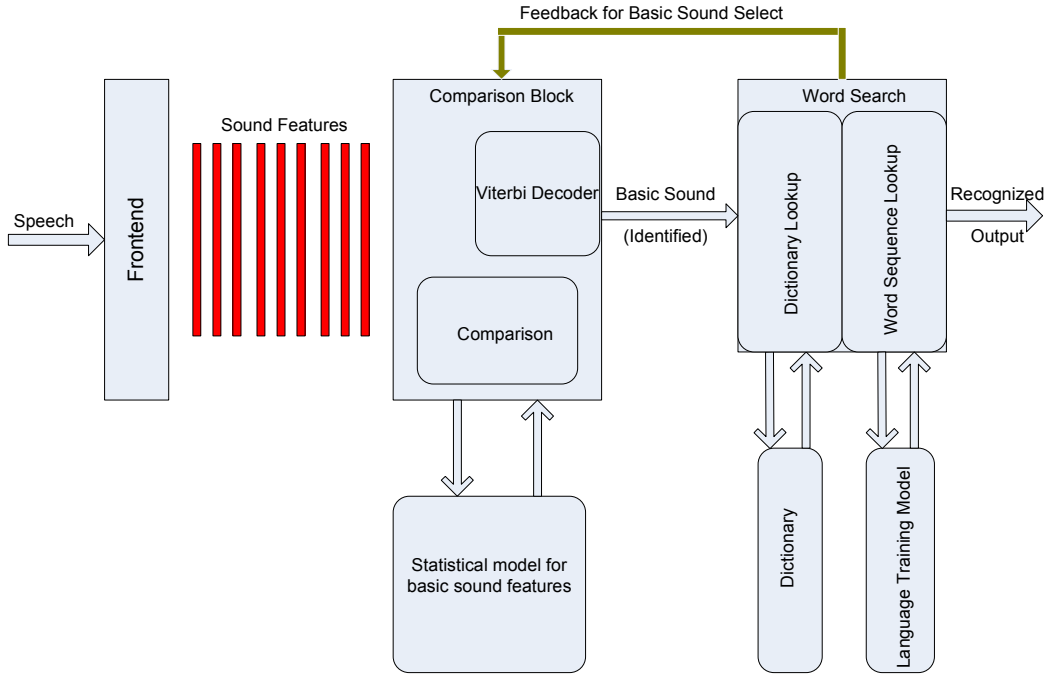

Figure 1.4: Speech Recognition (Functional Blocks)

There are many close sounding words and many different words may sound similar because of speaker's accent. It often happens that sequence of sound produced matches closely with more than one word in the dictionary for one word spoken at the frontend. All the close matching words are considered as poteintial candidate for the actual word spoken. So a sentence with several words spoken at the frontend results in a trellis of words identified as poteintial candidate for words spoken. The second part (word sequence search) in word search block picks the most likely sequence of words spoken from the trellis. This is done by baising certain sequences of words over others. The baises are formed based on commonly appearing word sequences in the

language training model.

1.1.2 Identifying Performance Bottlenecks

We now look at the performance bottlenecks in each block (as shown in Figure 1.4) of a speech recognition application run on a general purpose microprocessor. Our speech recognition application is Sphinx-III [1]. We use Simplescalar [3] [3] simulator for the underlying processor architecture, is similar to present day application processor on a mobile device. It is a 233MHz, 2-way out-of-order with 32KB L1 instruction cache, 32KB L1 instruction cache, 256KB unified L2 cache (instruction and data). Since the features are extracted every 10ms, we look at how the following parts perform in processing the features extracted in 10ms. If each block takes 10ms or less (*i.e.* 2.33 million cycles or less) in processing, a simple pipelined architecture with 3 stages would suffice.

The Frontend is a lightweight process and the software implementation for extracting features from speech signal take less than 1% [27, 24] of the execution time of speech recognition application on a general purpose microprocessor.

In the comparison block, there may be as many as 6000 (or more) different statistical models representing basic sound features. The number of comparisons can be reduced with a feedback mechanism from the word search block. Figure 1.5 shows the performance of the comparison part of the comparison block. Even one thousand comparison takes around 16 million processor cycles. Usually, with feedback, the

[3] EricScalar - SimpleScalar modified by Dr. Eric Rotenberg for ECE 721

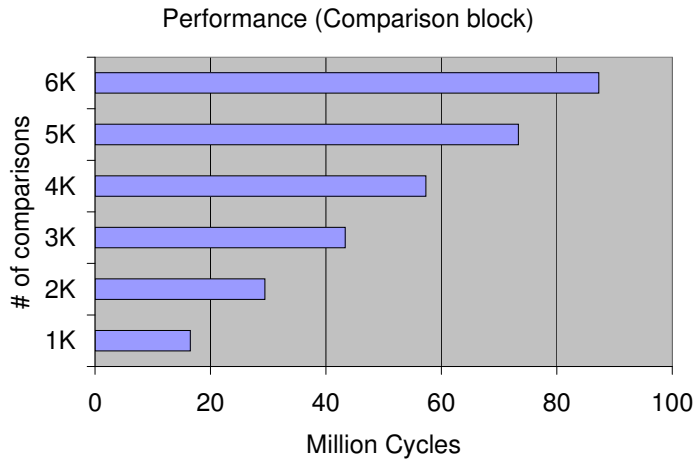

Figure 1.5: Comparison Computation Performance

number of comparisons reduced to around two thousand. All comparisons must be done without feedback. Therefore, the comparisons are a bottleneck for a real time speech recognition.

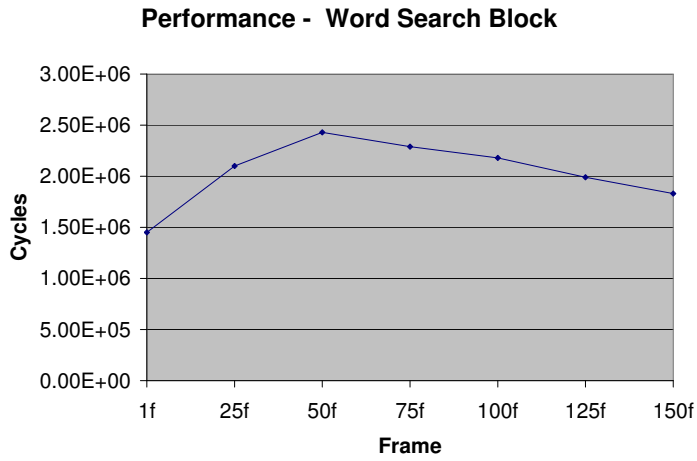

Figure 1.6: Word search block Computation Performance

We now look at the performance of word search functional block (Figure 1.6). The performance of this functional block is dependent on size of the vocabulary. We use a 11000+ dictionary. The performance shows that word search block exceeds the

number of cycles needed for real time recognition for many frames. A dictionary for large vocabulary is usually consists of hundred-thousand words. With those many words in the dictionary, the number of frames not processed within 10ms is much more than the number shown in the figure. Therefore, this gives us another bottleneck that needs to be considered.

The performance of speech application clearly shows that real time recognition is not possible on a present day embedded processor. In this work, we propose a software-hardware co-design of the speech recognition system such that real time recognition can be handled on a mobile device.

1.2 Proposed Solution and Key Contributions

- We propose a new design to traverse through lexical tree dictionary in hardware and suggest an ASIC for word search, which uses this hardware tree traversal to look through the words in the dictionary. It recognizes individual words and provide information for determining the possible word sequences in the input speech. One unit of this design performs about 20 times better than other implementation of specialized hardware design to run speech application.

- Large vocabulary may have more than hundred thousand dictionary words. The dictionary is stored in a DRAM memory. We propose a new way to store the dictionary words in the DRAM which facilitates lexical tree traversal and reduces memory bandwidth by a factor of 11 when compared to software speech

application and by a factor of 4 when compared to other hardware implementation of speech recognition.

- In the previous section, we saw that the comparison of a feature vector with the database is a performance bottleneck. The comparison involves complicated floating point calculations. We propose a dedicated ASIC for these computations. The ASIC can perform the required number of comparison within 10ms (*i.e.* meets real time goal) and can operate in much lower frequency than the processor, therefore saves power. This solution also enables the system to do more exhaustive search and computation than a software speech recognition application tailored for embedded enviroment. The design is flexible to algorithmic changes in speech recognition theory which reduces processing requirement and bandwidth, thus gives opportunity to reduce frequency and hence reduce power. One ASIC unit performs 4 times better than one unit of specialized hardware (processing element) for running speech application. The ASIC power consumption is lower by a factor of eight.

- The feedback from the word search enables the comparison block to make smaller number of relevant comparison in the database than comparing the entire database. We design a mechanism which implements the feedback using three small embedded SRAMs (1KB each). This design also priortizes some comparisons over others depending on their relevance.

1.3 Dissertation Outline

The remainder of the dissertation is organized as follows:

- Chapter 2 introduces the HMM based speech recognition theory in detail. The Markov process is explained with a simple example of weather forcasting. The hidden Markov model is introduced with the coin tossing example. Basic definitions and and an understanding of speech recognition system is presented.

- In chapter 3, we take look at the characteristics of speech recognition applications. Various performance bottlenecks are analyzed in greater detail, which leads to our software - hardware co-design of the speech recognition system.

- Chapter 4 touches upons the related work in academia and industry. The CMU-Sphinx system (opensource) is explained in detail as we use Sphinx system for speech recognition *Frontend* and SphinxTrain for creating Acoustic and Language Models. Known speech recognition implementations in hardware is then discussed.

- Chapter 5, presents our design, its various components and an understanding of how the system works.

- Chapter 6 presents the performance and power results of our system and compares it with the other related work.

- Chapter 7 concludes the thesis with summary of our work.

Chapter 2

Hidden Markov Model and Speech Recognition Theory

As mentioned earlier, speech recognition is usually efficient and accurate using Hidden Markov Model (HMM). We start by understanding Markov process (chain) and a hidden Model model and its application to speech recognition.

2.1 Markov Process

A Markov process is a stochastic process with Markov property. The Markov property states that conditional probability of appearance of future state of a process, given the present and all the past states only depends on the present state and not on any of the past states. In other words, the future state is independent of the path of the process (past states of the process). If $X_1, X_2, X_3, ..., X_N$ are the random variables which represents the occurrence of states $s_1, s_2, s_3, ..., s_M$ then

$$P(X_i/X^{i-1}) = P(X_i/X_{i-1}) = P(X_i = s_p/X_{i-1} = s_q) \tag{2.1}$$

where $X^{i-1} = X_{i-1}, ..., X_3, X_2, X_1,\ i < N$ and the random variable X_i and X_{i-1} represent the occurrence of state s_p and s_q, where $p, q \epsilon [1, M]$ respectively. The probability $P(X_i = s_p / X_{i-1} = s_q)$ is called the transition probability between the states p and q and is represented by a_{pq}. The other properties associated with Markov proccess are

$$\sum_{j=1}^{M} a_{ij} = 1; \tag{2.2}$$

$$\sum_{j=1}^{M} P(X_1 = s_j) = 1; \tag{2.3}$$

Both the equations represent basic principles of probablity theory that the sum of the transition probabilities from one state to all of its possible next state is one and the sum of the probabilities of the initial state being in any one of the M states is also one. The Markov process above is called an observable Markov model. The output of the process is a set of states, the occurence of each of which corresponds to an event represented by the random variables. There is a one-to-one mapping between observable event sequence X and Markov chain state sequence S.

The following example gives a good understanding of the Markov process and its applicability. The summer weather in Raleigh, NC can be considered to have the following three states.

- Really hot and dry (s_1)
- Really hot and little humid (s_2)
- Really hot and very humid (s_3)

Let $P(s_i)$ represent the probability of a certain day (initial day) being in state s_i. Since the summer days are categorized into the above three states, any particular day will be either dry or little humid or very humid. Therefore, probability that the intial day is either dry, little humid or very humid (equaton 2.3) follows:

$$P(s_1) + P(s_2) + P(s_3) = 1; \tag{2.4}$$

Assume, $P(s_1) = 0.4$, $P(s_2) = 0.4$ and $P(s_3) = 0.3$. Now, if the transition probability from dry, little or very humid weather on a particular day to dry(d), little humid($l.h$) or very humid($v.h$) next day is given by the following matrix:

$$a_{ij} = \begin{array}{c} \\ d \\ l.h \\ v.h \end{array} \begin{array}{c} \begin{array}{ccc} d & l.h & v.h \end{array} \\ \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.3 & 0.3 & 0.4 \\ 0.1 & 0.3 & 0.6 \end{pmatrix} \end{array}$$

The above matrix is called transition probability matrix for this system. The summing transition probabilities to all possible weather states for next days we get one (equation 2.2). In above probability transition matrix this corresponds to the sum of the row elements. With above information we can forecast the probability of a particular weather pattern using Markov process. Probability of a weather pattern $-l.h- > d- > l.h- > v.h- > l.h- > d- > v.h- > v.h-$ is evaluated as follows:

$$\begin{aligned} P(O|M) &= P(s_2, s_1, s_2, s_3, s_2, s_1, s_3, s_3) \\ &= P(s_2)P(s_1|s_2)P(s_2|s_1)P(s_3|s_2)P(s_2|s_3)P(s_1|s_2)P(s_3|s_1)P(s_3|s_3) \end{aligned}$$

$$= (0.4)(0.3)(0.6)(0.4)(0.3)(0.3)(0.1)(0.6)$$

$$= 0.00015552$$

2.2 Hidden Markov Model

The above Markov model is observable, *i.e* states are observed when an event occurs. This observable Markov model is too restrictive and cannot be applied to many problems. Alternately, Markov model can be extended such that the observations are probabilistic functions of the state. In other words, it is a double-embedded stochastic process with an underlying stochastic process not directly observable (hidden). This underlying stochastic process can only be probabilistically associated with another observable stochastic process producing the sequence of features we can observe. The hidden Markov model concept is well understood through the well known coin toss example [34]. Assume that there is a coin tossing experiment where a person (say person A) is performing coin tosses from two different sets of coins: one set has two coins in it and the other one has three coins in it. Person A can select any coin from just one set for the toss. Now another person (say person B) can't see how A chooses a coin but just observes the outcome of sequence of head and tail.

The observation O is sequence of head or tail tossed by person A and seen by person B. Each set can produce an observation and each individual coin can be considered a state. Within a set, the probability that another coin (or the same coin)

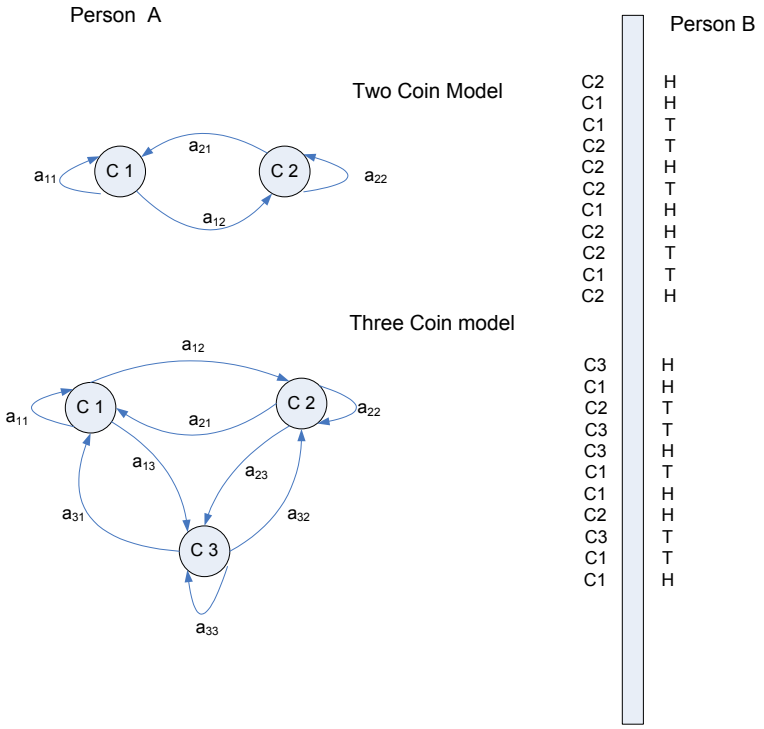

Figure 2.1: Hidden Markov Model Experiment

will be tossed is the transition probability a_{ij} (the j^{th} coin will be tossed immediately after the i^{th}). The person B can now probabilistically determine whether the two coin model or the three coin model was most likely used for tossing.

HMM based speech recognition uses the same concept. Any utterence (speech) is composed of sequence of basic sounds. Any word in dictionary is composed of sequence of basic sounds. Each of these basic sound have statistical models. Therefore, a word can be expressed as sequence of statistical models. Speech (to be recognized) is sampled by the the speech recognition system to form a sequence of speech feature vectors (numerical parameters). This sequence of speech feature vector is the observation sequence. The recognizer then probabilistically determimes which model(s)

most likely would produced the speech vector. Identifying the model sequence results in identifying the word spoken.

2.3 Speech Recognition Theory

A spoken word or phrase is represented as a sequence of basic sounds called *phone* or *phoneme* [39, 18, 20, 15]. For example, there are approximately 51 phonemes in the English language. A Permutation of these phones leads to a word or a phrase. The sentence "This is speech" is phonetically broken down to 'th ih s ih z s p iy sh'.

Phones are an excellent means for encoding word pronunciation but they are not ideal for recognition of speech. The contextual affects cause variation in the way the basic sounds are produced. Each of the phones along with it neighboring phones (left and right) is called a *triphone*. Therefore, in above example sound of 'ih' is effected by neighboring 'th' (left phone) and 's' (right phone). Recognizing triphone units in context tends to be more accurate than recognizing individual phones. The triphones are represented with a '-/+' sign so 'th-ih+s' represent a triphone where 'th' and 's' are the left and right phones respectively. The first phone in any utterance has 'sil' (representing silence) as the left phone. The total number of possible triphones in English is around 65000. The phones without any influencing neighbouring phones are called *context-independent* phones. In this text, phones and triphones are used interchangeably. Context-indenpendent phone refers to original phone without any influence of neighbouring phones.

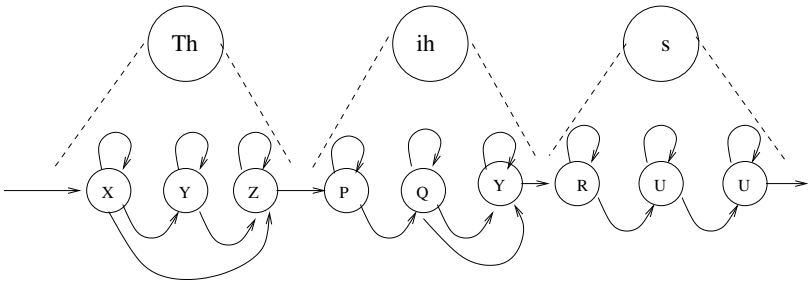

Figure 2.2: Composite HMM representing the word this (th-ih-s)

In HMM based speech recognition theory, for each phone and triphone, there is a corresponding statistical model called hidden Markov model. These HMMs are sequence of states called Markov states, connected by probabilistic transitions. Each of these states have one or more entry and exit states. Exit state of one triphone merged to entry state of another forms *composite HMM*, which allows to represent a word and then the words joined together to form a phrase or utterance.

Figure 2.2 is an example [1] of a three state HMM representing the word "this". The states in triphones are best represented by multi-variate mixture Gaussian distribution. The parameters for the statistical models which represents the states (Gaussian mixtures and transition probabilities) are obtained from the training data and is collectively called the *Acoustic Model*.

In absence of sufficient training data or to avoid redundancy of data the states of different triphones can be sometimes represented by same distribution [21, 40], and these are called *senones* or *tied mixtures*. Therefore, a combination of senones

[1] this example is just for understanding

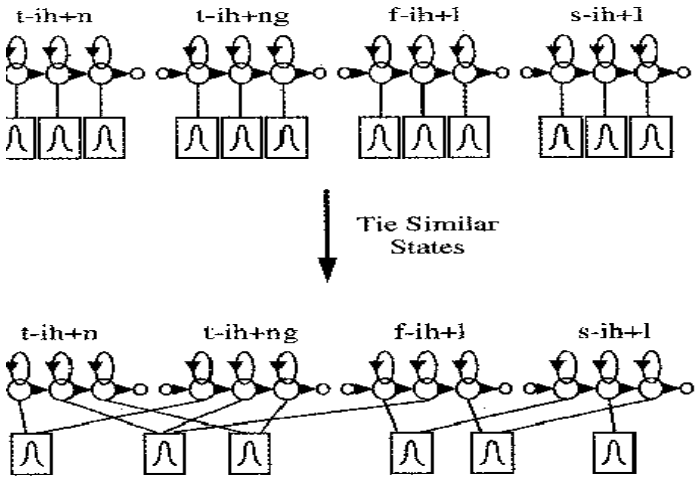

Figure 2.3: Tied mixtures or Senones

forms triphones, which, in turn, combine to form words and words, themselves, come together to form a sentence or an utterance. Number of senones in English language is around 6000.

The collection of all words is called the *dictionary*. The dictionary words are phonetically broken down and are stored in either *flat* form or *lexical tree* form [14, 36]. Figure 2.4 shows two form of dictionary arrangement representing words - Start, Starting, Started and Startup.

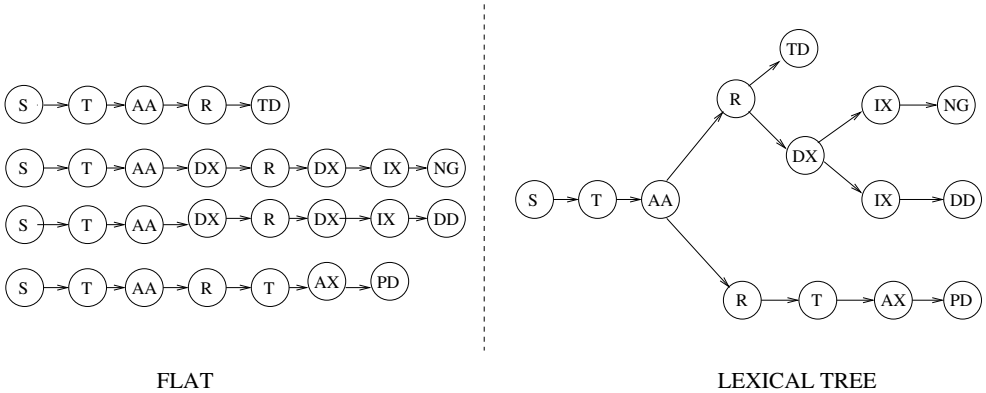

Figure 2.4: Dictionary Arrangements

In a speech recognition, speech or the utterance (to be recognized) is a sequence of words. It goes through the *Frontend*, where spectral analysis is done to extract the *feature vectors* (*input acoustic vectors*). These feature vectors are sampled at fixed intervals (10 ms). The sequence of vectors thus obtained is called the *observation sequence* . If the observation sequence is $O = O_1, O_2, O_3, ..., O_T$, where O_i represents a feature vector sampled at fixed interval, the speech recognition decoder (the decoder is the part which does the actual recognition) finds the sequence of words (i.e. sequence phones or triphones) which is most likely to produce the observation O. If the sequence of n words is represented by $W = W_1, W_2, W_3...., W_n$, the decoder maximizes $P(W|O)$, where P represents the probability. We want to maximize $P(W|O)$, applying Bayes' theorem

$$P(W) = arg_W max P(W|O) = arg_W max \frac{P(W).P(O|W)}{P(O)} \tag{2.5}$$

Where $P(W)$ represents the probability of a word sequence and is available from language model. $P(O)$ is the probability of the observation sequence. $P(O|W)$ is the probability of observation sequence O, given the word sequence W. It is computed using a composite hidden Markov model for W, constructed from simple HMM phone or triphone models joined in sequence (according to word pronunciations stored in a dictionary).

In speech recognition, HMM generates speech feature vector sequences. It is a finite state machine which changes state every time unit and each time t that a state j is entered, an acoustic speech vector O_t is generated with probability $b_j(O_t)$,

called the *observation probability*. As mentioned earlier, each of the transition is probabilistic. The probability of transition from state i to state j is given by discrete probability, called the *transition probability* and is represented by a_{ij}.

The joint probability of observation of vector sequence O and state sequence $X = x_o, x_1, x_2...x_{T-1}$, given some model M is calculated as product of the transition probabilities and the output probabilities.

$$P(O, X|M) = b_0(O_0)\Pi_{t=1}^{T-1} b_{x_t}(O_t) a_{x_{t-1}x_t} \tag{2.6}$$

assuming the HMM is in state 0 at time $t = 0$. In practice, only observation sequence O is known and underlying state sequence is hidden. The probability $P(O|M)$ is determined by the probability associated with the state sequence which maximizes $P(O, X|M)$. This is calculated using *Viterbi decoder*. We define $\delta_t(j)$ as the maximum probability that the HMM is in state j at time t. The value $\delta_t(j)$ is computed, as follows

$$\delta_t(j) = max_{0 \leq i \leq N-1}[\delta_{t-1}(i) a_{ij}].b_j(O_t) \tag{2.7}$$

where i is the previous state (at time $t - 1$). The observation probability $b_j(O_t)$, where O_t represents the probability that state j emits observation feature vector O_t for observation sequence number t. The observation probability [40] is mixture multivariate Gaussian distribution is given by

$$b_j(O_t) = \sum_{m=1}^{M} c_{jm} N(O_t; \mu_{jm}, \sigma_{jm}) \tag{2.8}$$

where c_{jm} is the weight of the mixture component, m in state j and $N(O_t; \mu_{jm}, \sigma_{jm})$ is the multivariate Gaussian with mean μ and covariance σ

$$N(O_t; \mu_j, \sigma_j) = (\frac{1}{\sqrt[L]{2\pi}}) \frac{1}{\sqrt{\Pi \sigma_{ji}}} . e^{-\frac{1}{2} \sum \frac{(O_{ji} - \mu_{ji})^2}{2\sigma_{ji}^2}} \tag{2.9}$$

L is the dimension of the feature vector considered (summation and multiplication is done from 1 through L). Calculation of observation probability is computationally very intensive and is usually calculated in logarithm domain to reduce the complexity. Moreover, it is calculated against large number of senones per frame.

2.4 Language Model

The language model is used by large vocabulary speech recognition system to increase recognition accuracy [8, 9, 31]. It helps in identifying the most probable sequence of words uttered from a large pool of words recognized as candidate for actual words spoken. The Language model selects a sequence of words attaching more weight to certain word sequence out of pool of alternative word sequences produced by the acoustic model for possible match. Weight are priori probability of sequence of words. Inclusion of language model also helps in increasing efficiency of speech recognition system by eliminating (pruning) the unlikely path for word sequence search. All the present day speech recognition systems have some sort of language model. The language model also enables the recognition system to have correct syntax and semantics of the language. It helps in forming meaningful and grammatically well formed sentences. Language modeling can be done using Chomsky's formal language

theory [29] [2] or by using probabilistic language model (Stochastic language model). Most of the recognition system uses Stochastic Language Model (SLM) [23].

2.4.1 Stochastic Language Models

Stochastic language modeling is a probabilistic approach to language modeling. The most commonly used SLM is the N-gram model. It provides adequate probabilistic information so that likely word sequence has the maximum probability among all other possible word sequence.

N-gram Models

Revisiting equation 2.5, $P(W)$ is the probability of word sequence. $P(W)$ is mathematically expressed in the following form.

$$\begin{aligned} P(W) &= P(w_1, w_2, w_3, ..., w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_n|w_1, w_2, w_3, ...w_{n-1}) \\ &= \Pi_{i=1}^{n} P(w_i|w_1, w_2, ...w_{i-1}) \end{aligned} \tag{2.10}$$

where $P(w_i)$ is the probability of word w_i appear following the occurence of word sequence w_1, w_2, w_3,...,w_{i-1}. The probability evaluation is managable with small vocabulary size since the complexity of the language model increases exponentially with the vocabulary size. If the vocabulary size is V, the word w_i will have V^{i-1} histories and it is practically impossible to evaluate all the values even for a moderate

[2] Not discussed in this text

value of i. To reduce the complexity of the $P(W)$ a variant of above probability formulation is considered. Instead of $P(w_i|w_1, w_2, w_3...., w_{i-1})$ an equivalance class $P(w_i|w_{i-N+1}, w_{i-N+2}, w_{i-N+3}...., w_{i-1})$ is considered. This is called the N-gram language model [17, 33]. When $N = 1$ we have $P(W) = \Pi P(w_i)$ - *unigram* language model, the probability of ith word does not depend upon any previously spoken words but depends on probability of a word appearing in training data. When $N = 2$ we have $P(W) = \Pi P(w_i|w_{i-1})$ - *bigram* language model. Therefore, bigram language model suggests that probability of appearence of a word in an utterence depends on the previous word uttered. When $N = 3$ we have $P(W) = \Pi P(w_i|w_{i-1}, w_{i-2})$ - *trigram* language model - probability of word appearing in an utterence depends on two previous words uttered. The three language models listed above are most commonly used in speech recognition systems. In some of the systems $N = 5$ is also used. The probabilities of all these stochastic language models are predefined for speech recognition process, and the values are obtained from training the language model using training data. The probabilities for N-gram($N = 3$) model are usually calculated using the following formula.

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{F(w_{i-2}, w_{i-1}, w_i)}{F(w_{i-2}, w_{i-1})} \tag{2.11}$$

$F(w_{i-2}, w_{i-1}, w_i)$ refers to the frequency of occurrence of the trigram (w_{i-2}, w_{i-1}, w_i) in the training text and $F(w_{i-2}, w_{i-1})$ refers to the frequency of occurrence of the bigram (w_{i-2}, w_{i-1}). The training data however may not cover entire vocabulary or have all possible N word sequences for a N-gram model. Therefore, a speech recognition

system using N-gram model may bias against a sequence of words which has maximum likelyhood according to acoustic model of the system. In such cases M-gram ($M < N$) probabilities are used in the place of N-gram probabilities after reducing the probability by a back-off weight which accounts for the fact that the next higher N-gram has not been seen and therefore has a lower chance of occurring [12]. Alternatively, smoothing technique is used to get the N-gram probabilitiies. In N-gram smoothing, the low probabilities are raised and the high probabilities are lowered. The idea is to keep the language model with extreme values (N-gram probability of 0 or 1)totally overriding the the selection of word sequence even though acoustic model may suggest otherwise. A simple smoothing technique is to assume any trigram (sequence of three words) appears one more than actually observed in the training data as shown below.

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{1 + F(w_{i-2}, w_{i-1}, w_i)}{\sum_{w_i}(1 + F(w_{i-2}, w_{i-1}, w_i))} \tag{2.12}$$

With above equation 2.12, the probability of any unseen trigram in the training data is not zero but instead a very small value.

Chapter 3

Evaluation and Analysis of Speech Recognition Application

In this chapter, we look at how speech recognition application performs on a general purpose microprocessor. Our goal is to achieve real time large vocabulary speech recognition on a battery operated mobile device. We first evaluate the performance of speech recognition software application in processing one frame of speech on a general purpose microprocessor. One frame of speech is speech input in 10ms (sampling interval). In order to have a real time recognition one frame of speech input must be processed within 10ms or less.

We use Sphinx-III [1] for our evaluation and analysis. This is because, Sphinx is open source, free and achieves good recognition (fast and low error rate). In chapter 1, we mention that the speech recognition has three major independent functional parts - the Frontend, the Comparison Block (probability computation) and the Word Search. We evaluate and analyze each of these parts separately. The performance evaluation and power consumption of the entire application is already shown in chap-

ter 1, figure 1.1 and figure 1.2, respectively. Our analysis based on different functional parts, helps us understand various performance bottlenecks of the application when run on a general purpose microprocessor. It also helps us decide, hardware software partition in for a hardware-software system design for speech recognition.

Any speech recognition system implemented in software or hardware needs large memory for the acoustic model, dictionary and language model storage. The acoustic model is looked up numerous times every frame as senones are evaluated in each and every frame in the recognition process. The dictionary is also looked up every frame for the matching triphones. The language model is looked up less frequently than the acoustic model and dictionary. For a software speech recognition application on a general purpose microprocessor these memory lookups are converted to loads and stores (generated because of storing various scores computed in the process of recognition). In case of hardware (hardware accelerator) implementations the lookups and stores are memory reads and writes.

3.1 Performance Analysis of the Functional Blocks

We saw a brief performance evaluation in chapter 1. We now present detailed performance analysis of the functional blocks. Since speech recognition is very memory intensive, we present the complete analysis of the application's memory behavior along performance.

3.1.1 Frontend

The Frontend converts the speech input to acoustic vectors. The speech signals undergo digital signal processing and acoustic vectors [25, 26] are obtained. The Frontend only counts for 1-2% of the processing time of speech recognition application [24, 27]. Though, Frontend is a light weight process (low processing time), we still look at its memory characteristic since its memory behavior may effect the performance of other jobs running simultaneously. The following figures 3.1 and 3.2 shows the miss rates of the Frontend in L1 and L2 cache.

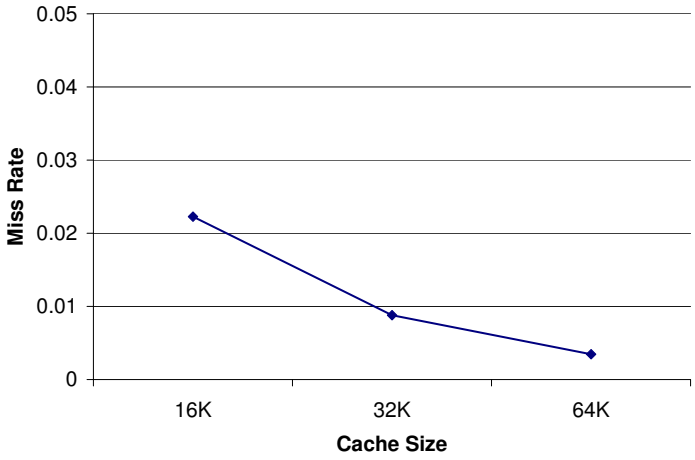

Figure 3.1: Frontend L1 Cache Miss Rate

3.1.2 Observation Probability Computation

In chapter 1, we have seen the time taken to evaluate senone scores for different number of senones. The amount of time taken to score a group of senones is linearly depended on the number of senones present in that group. This linear trend is

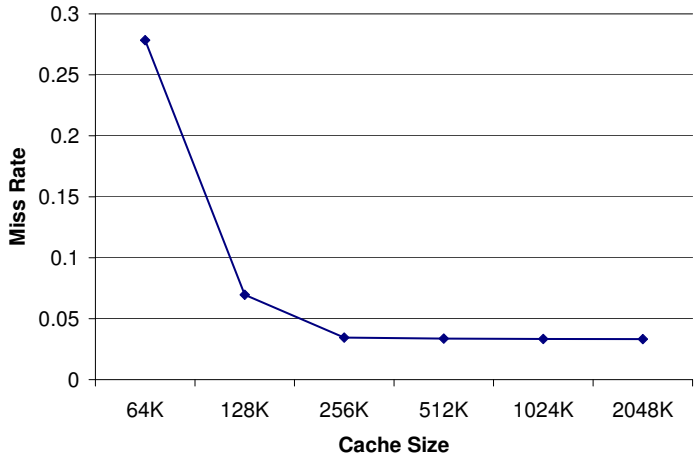

Figure 3.2: Frontend L2 Cache Miss Rate

expected and is observed in Figure 1.5. The reason for this linear trend is the fact that the senone score (observation probability) is evaluated using equation 2.8 and each senone has different acoustic parameters (mean, variance and constant cofficients). The acoustic parameter for each senone is around 2.5KB. Once a senone score is computed its 2.5KB of data is never reused. So N senones would require 2.5*N KB of data.

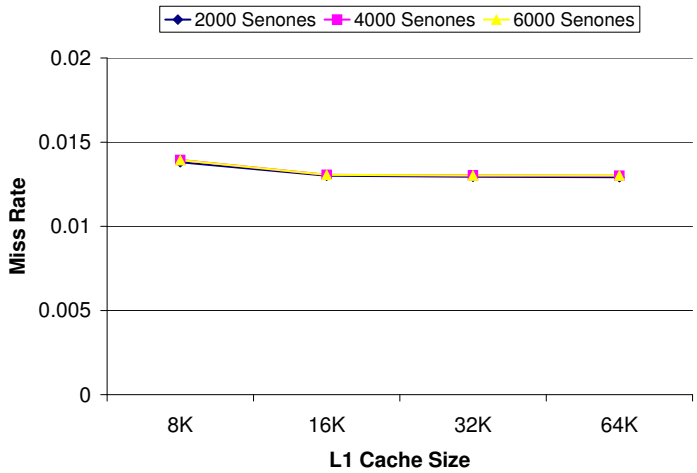

Figure 3.3: L1 Cache Miss Rate for Observation Probability Calculation

Figure 3.3, shows the L1 cache miss rates of the observation probability calculation of 2000, 4000 and 6000 senones on different cache sizes. Figure 3.4 shows ratio of number of hits per way in L1 cache. The high hit rate in most recently used cache line imply of a good spatial locality. The observation probability calculation is expected to show high spatial locality since each of the parameters in acoustic model is 4 byte (32 bits) wide and are packed together in cache blocks and each of the senone is evaluated one by one.

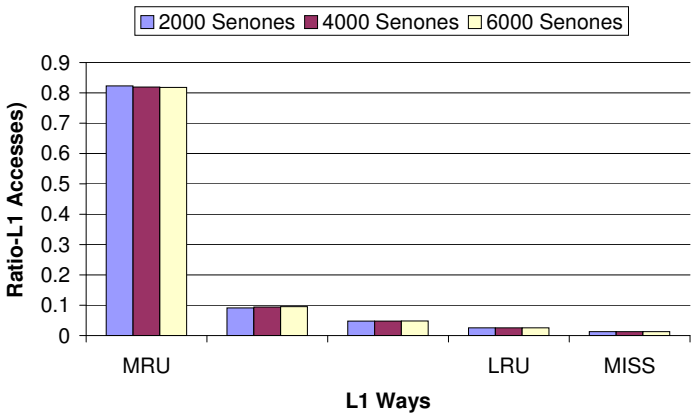

Figure 3.4: Access pattern in L1 Cache

The miss rate in L2 cache is shown in Figure 3.5, 3.6 and 3.7. The high miss rates in L2 and the pattern of hits in the L1 ways indicate that lack of temporal locality in data used to evaluate the observation probability of senones. The high number of accesses to L2 cache with high miss rate results in cache pollution and effects other application threads or different threads of the same application. It tends to increase the number of misses in L2 access for other threads possibly slowing them down [10].

High spatial and low temporal locality characteristic of the observation probability

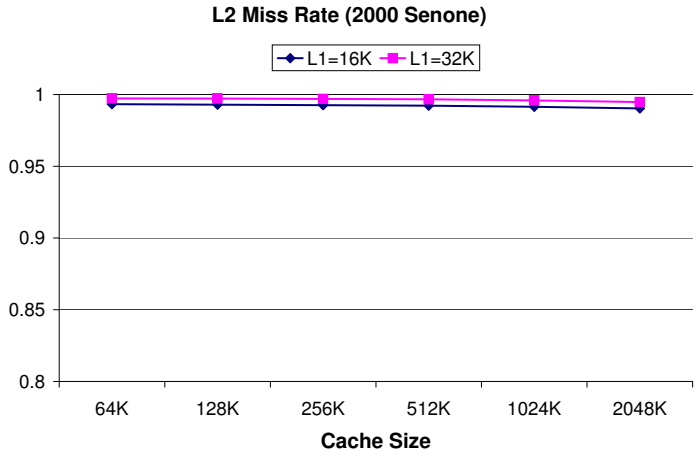

Figure 3.5: L2 Miss Rate for Observation Probability Computation (2000 Senones)

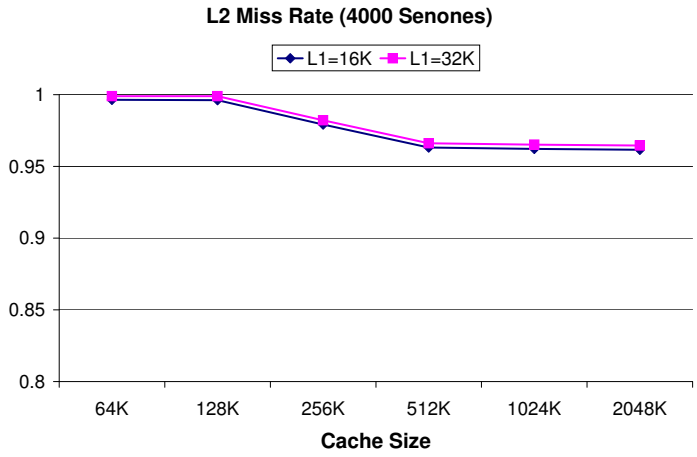

Figure 3.6: L2 Miss Rate for Observation Probability Computation (4000 Senones)

imply that observation probability of senone can be computed with one hierarchy of small cache size.

We now focus on the computational part of the senone score evaluation. This is done by assuming one cycle latency in all loads and stores, *i.e.*, every memory request (including the cold misses) is assumed to be a hit in L1 cache. Figure 3.8 shows the performance on senone computation in two different architecture platform for 2000 4000 and 6000 senones. The processor is assumed to be 2-way out of order 1 GHz

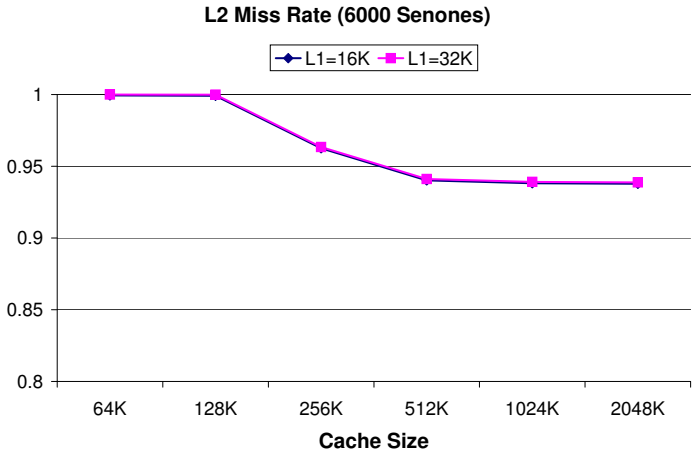

Figure 3.7: L2 Miss Rate for Observation Probability Computation (6000 Senones)

processor.

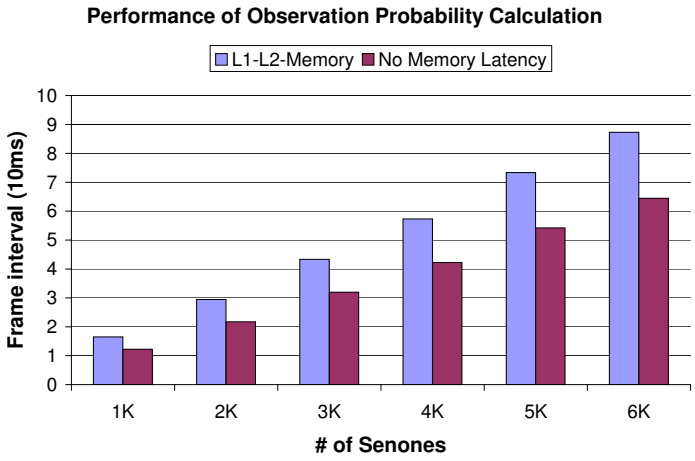

Figure 3.8: Performance of Observation Probability on 1GHz Processor

Ideally, all senones in an acoustic model used in recognition system should be scored within one frame. Therefore, for large vocabulary speech in English, having 1000 or more senones, real time recognition is not possible since it takes about 8.6 times more time to compute.

Even an aggressive processor for mobile computing enviroment does not meet real

time requirement. Therefore, either architectural changes are needed for running the speech application or ASIC accelerator is needed to to speed up the computation.

3.1.3 Word Search

The words in a speech recognition system are represented by sequence of triphones. Each triphone is represented by three senone sequence in 3-state HMM (or five senone sequence in 5-state HMM) and the transition matrix. If the senone scores are available, the triphone score can be computed using a Viterbi decoder [13, 16, 19, 22]. In each frame, the word search computes its triphone scores. A triphone (in a word) may be computed in consecutive frames, also one frame may compute more than one triphone. (shown in figure 3.9).

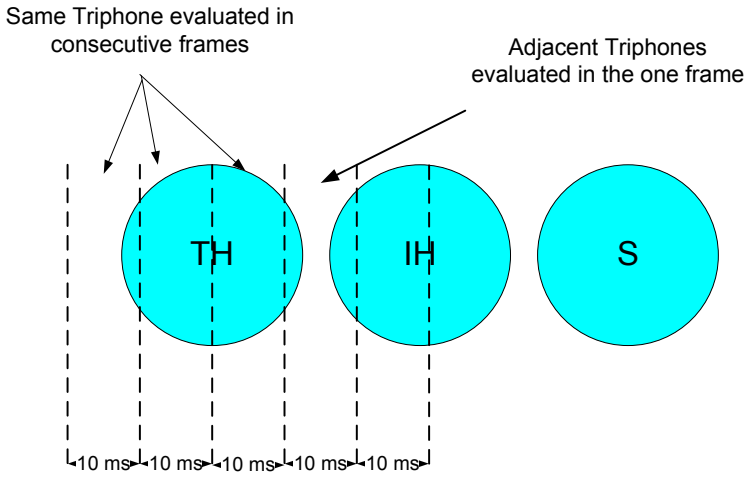

Figure 3.9: Triphone Evaluation Example

A word is idtentified by the speech recognition system as spoken word when it detects the last triphone of the word and the over all word score (sum of score of each triphone in that word) and individual score of each individual triphone in that word

is above preassigned threshold. We refer triphones in a word as *nodes* or *triphone nodes*.

For real time speech recognition, every frame needs to evaluate certain number of nodes. These nodes are called the *active nodes*. The number actives nodes changes dynamically based on algorithm implemented in the speech recognition system. Dictionary used in the speech recognition application is a lexical tree dictionary where the different words share the common nodes. Sharing reduces memory space for storing dictionary words and reduces redundant score computation. The computation of each node with available input data takes same amount of time.

With the dictionary arranged in lexical tree format the two factors effecting the performance of this stage are number of active nodes and memory latency of input data. Unlike senone evaluation stage, the cache hierarchy plays an important role in determining the performance of this stage. The reason for this is the fact that a particular node may be active in two or more consecutive frames, therefore, a high possibility of the node data structure being found in the cache. In presence of a node data structure in cache hierarchy results in quicker score evaluation because of smaller latency for data.

There are three possible categories in which we can classify the performance of the word search stage. They are 1) None or very few active nodes present in the cache hierarchy (High miss mate 2) Almost all or majority of the active nodes present in the cache hierarchy (low miss rate) 3) Neither the hits (nodes present in the cache)

nor the misses (nodes not preset in the cache) on active nodes are ignorable. The following figure (Fig. 3.10) shows first two categories. It shows the miss rates of the L2 cache accesses by the word search in frame duration. All fresh nodes signifies that score of each node is computed for the first time in this frame, therefore causing L2 cache misses. Score of a node computed in the preceding frame is a repeat node and likely to result in cache hits. The miss rates for the third category is bounded within maximum and minimum miss rates values available for these curves. It is

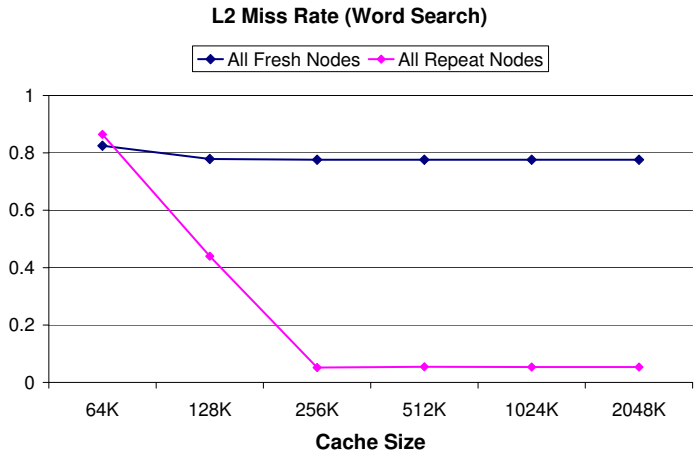

Figure 3.10: Miss Rate for All Fresh nodes, and All Repeat Nodes

difficult to estimate the performance of the word search part of speech recognition application and guarantee real time performance since the number of active triphone nodes and the repeat characteristic of the active triphones in consecutive frames are input (speech) dependent. We, therefore, bound the performance of word search between time taken to compute one triphone node score under cache hit and time taken to compute one triphone node score under cache miss. The figure (Fig. 3.11) below shows the number of triphones (nodes) that can be evaluated in one frame (*i.e.*

10 ms). (These values are not linear with processor speed. The processor speed is only used for calculation of processor memory latency in cycles)

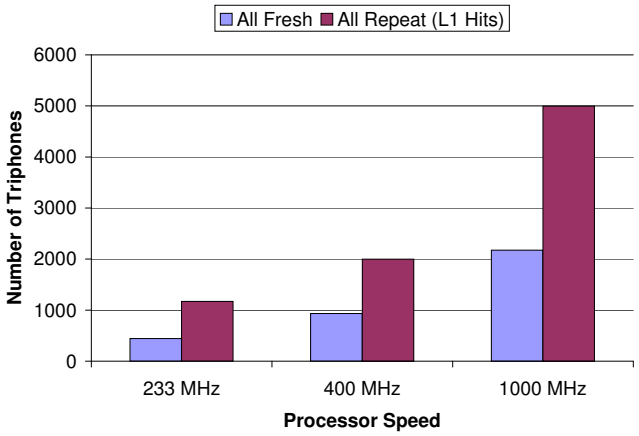

Figure 3.11: Performance of Word Search Stage - Triphones per Frame

3.2 Building a Case for our Design

We now find a solution for real time speech recognition. For real time speech recognition, input speech is sampled every 10ms and must be processed within 10ms. This involves processing by the Frontend, computing the observation probability and processing by the Word Search block.

We have seen in the previous section that a very aggressive processor (1 GHZ, 2-way, 32KB L1D, 32KB L1I, 256KB L2) cannot process the computation of observation probability even for 1000 senones within 10ms. Moreover, it can poteintially slow down other applications or other parts of speech recognition application because of its high miss rate in L2. Since the calculation of probability is same for all senones, we suggest an ASIC for the computation and a RAM memory for the acoustic model.

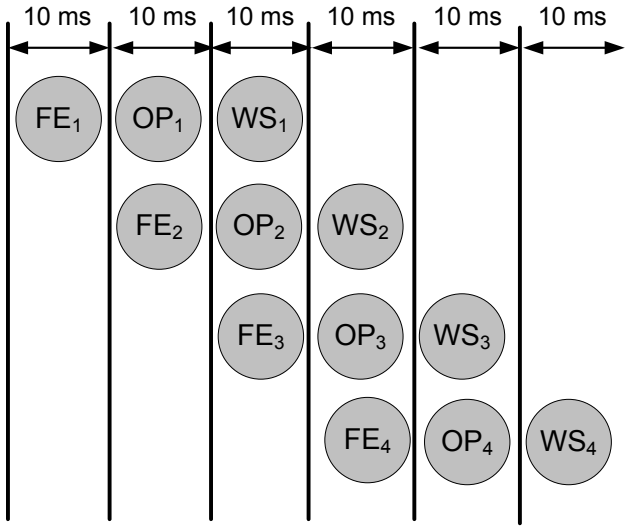

Figure 3.12: Processing of Input Speech Pipelined across three Functional Blocks

The performance of word search cannot be gauranteed since it depends on availability of cache hierarchy, size of the vocabulary (dictionary size) and input speech. We suggest an ASIC for computation and a RAM for the dictionary to gaurantee performance of based on number of triphones that can be processed.

We separate out the three functional blocks and compute each of these in parallel as shown in the figure (Fig. 3.12). This give more time to each block to process.

ASIC blocks can operate on much lower clock frequency and perform much better than the speech recognition software application on general purpose processor.

Chapter 4

Related Work

Speech recognition applications have been developed in industry for commercial purpose as well as in university for academic interests. Most of the speech recognition systems are based on HMM. The software solutions for speech recognition are available as software packages. These packages have two parts - the decoder, which does the actual recognition of the utterance and the acoustic and language models, which are used by the decoder. The software solutions are efficient implementation of speech recognition algorithms in high level language, therefore need a general purpose microprocessor. The decoder is computationally very demanding and dedicated hardware units and platforms have been built in academia and the industry. These hardware approaches usually use the acoustic and language models of the software packages. We now look at few software solutions as well as hardware approaches for speech recognition.

4.1 Software Solutions

The AT&T Watson speech recognition engine [35] is a software implementation of AT&T voice processing technology. This system is HMM based and the uses gender based triphone models. In 94 NAB [1] evaluation, the system has around 22000 context dependent phones. The language model is 5-gram. The recognition is in two passes, the first pass does the phone recognition, word recognition and build the word lattice and the second pass rescores the word graph. The system works on Pentium with Win/NT operating system. This system also implemented for shared memory multiprocessor architecture [32].

The IBM speech recognition is known as *Via-Voice*. It is different from other speech recognition system. It uses rank-based approach [2] for the computation of observation probabilities. The search technique is a combination of A* search [3] heuristic and time synchronous Viterbi decoding. A version of Via-Voice designed for embedded processors and works on PDA's and automobile GPS system and mobile devices for command and control. A detail on this version is not available in literature.

SRI international's Speech Technology And Research (STAR) groups has speech recognition system called DECIPHER. The system is HMM based and does multi-pass time synchronous Viterbi decoding. The system uses tied-mixture states.

Sphinx-II speech recognition system was developed by CMU (later versions are

[1] Benchmark
[2] Not discussed in this report
[3] Not discussed in this report

Sphinx-III and Sphinx-IV). This system is based on hidden Markov models. Sphinx-II uses normalized feature representation, multiple-codebook semi continuous hidden Markov models [20]. It has multi-pass search architecture and unified acoustic and language models. For the HMM states it uses senones based approach tying mixtures of similar Gaussian distribution. It uses scaled-integer for computation of observation probability. The multi-pass search architecture is mainly used when the vocabulary is large. Viterbi algorithm is efficient but less optimal compared to A* search algorithm. In first pass, Viterbi decoding is done to reduce the search space for A* algorithm. In second pass the A* search combines the result of Viterbi decoding and the language model to effectively do the recognition. Sphinx uses a unified stochastic engine which combines acoustic model and the language model and jointly optimizes both the models. The newer versions of Sphinx are built on the older version. The Sphinx-III and Sphinx-IV uses lexical trees rather than flat dictionary. The searches in the higher version are efficient using beam searchs and various pruning strategies. The acoustic models consist of 3-state or 5-state HMMs. Sphinx is only good for academic interest as it requires extraordinary memory and high-end workstation for speech recognition.

Microsoft Whisper is built on the basics of CMU-Sphinx system is more applicable in real world. It works on common desktop machines. Whisper uses speaker adaptation and noise cancellation. Whisper is memory efficient as the acoustic model is compressed and has no decompression overhead effect on speech recognition and uses around 7000 senones.

Binu et.al. [27] worked on Sphinx-III and made it more efficient. They targeted on reducing the bandwidth requirement for calculating observation probability. Since observation probability for a single senone requires lot of data corresponding to the mean, variance and the weight mixtures and the calculations are done every frame, their scheme calculated probability of a senone for ten consecutive frames before moving on to the next senone. This reduces the bandwidth requirement by ten times, if all the senones are to be evaluated every frame. Moreover, calculating senone score for ten consecutine frames leads to more cache hits therefore has better performances.

4.1.1 Problems with Software Solution

Before going into hardware solutions for speech recognition, we re-visit the problems software implementation has with HMM based speech recognition.

- The calculation is **slow** because of complicated mathematical operation even on high performance platform. (Floating point multiplication, logarithm calculation or exponent calculation)
- To have simplicity, *fixed point* or *scaled-integer* arithmetic is used instead of floating point, therefore it **induces inaccuracy**.
- To speed up recognition, number of computation is reduced by introducing *threshold values*. This also **induces inaccuracy**.
- The computational need of speech recognition algorithm freezes most of the core resources in a general purpose processor.

- Speech Recognition requires large amount of memory (Table 1.1) for storing the parameters of statistical model (example Mean, Variance, Weights etc), language model and dictionary.
  - It has large working set which causes problem when some other application runs on the same platform. They share the same cache hierarchy resulting in **contention** and therefore affecting each others performance.
  - The **memory bandwidth** requirement is a bottleneck.
- Software solution is **power hungry**.

4.2 Hardware Solutions

Software solution fails to achieve the real time speech recognition, which is accurate and has tight power budget needed for mobile devices. Hardware-software approaches have been tried by many researchers. Mainly hardware accelerators [27, 28] for calculating the observation probability were implemented in academia. There are several IC's for speech recognition but it is not clear whether they do anything other than the frontend DSP and the probability calculation.

Low power accelerator [27] implements the computation of observation probability. The design acts as accelerator for Sphinx-III speech recognition system. The implementation has floating-point arithmetic. The floating point arithmetic, however, is not same as the IEEE-754 standards, instead has only 12-bit mantissa. Smaller size of mantissa reduces the bandwidth requirement by almost one third. The design

buffers the feature vectors from ten consecutive frames and then evaluates the acoustic score (probability) with respect to each senone. This enhances the performance with more cache hits (if there is a cache type fast memory buffer in the embedded environment) and reduced bandwidth requirement. This design (0.25μ process) shows 29-fold improvement in power consumption over the software implementations used by Sphinx-III on Pentium-4 (0.13μ process). Though, this design optimizes Sphinx-III for embedded environment, it evaluates senone score for ten consecutive frames together, therefore restricting any algorithmic optimization using the Viterbi decoder feedback. Other than computation of probability, the Viterbi algorithm for optimal path and the search of optimal path in word lattice is computationally demanding and therefore accelerator is not a complete solution for enhancing real-time performance.

Hardware implementation speech recognition system, with FPGA acting as a co-processor [28] for speech recognition is an alternative to ASIC implementation of real-time recognition. This implementation however, is not designed for power efficiency.

Low power device for speech recognitions implemented by Sergiu et.al. [30] does real-time speech recognition. Unlike other hardware recognition systems which are usually accelerators for certain function in the software, this design is a complete recognition system. It has its own language and acoustic model. This implementation does not use the n-gram language model, instead uses regular grammar based language model to ease the search space. It exploits parallelism existing in speech recognition algorithm with multiple Processing Elements (PE). Parallel execution

helps in reducing clock frequency resulting in reduced power usage. The acoustic and language models are kept in FLASH memory and SRAM is used for dynamic memory required for intermidiate stages of recognition. The design however is good for a very small vocabulary and is a phoneme based, therefore it is not good for real life use.

Chapter 5

System Architecture

The speech recognition system that we design has components shown in Figure 5.1. The input speech goes through the Frontend, where spectral analysis is done to extract acoustic feature vectors.

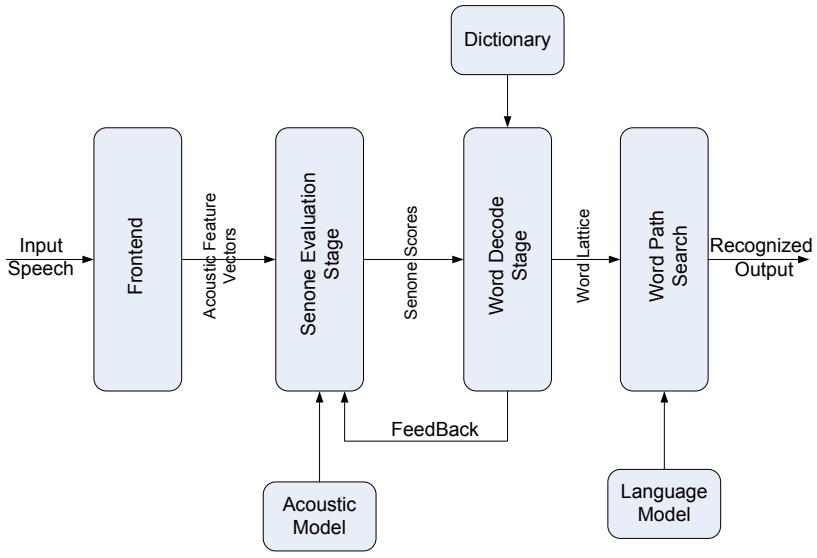

Figure 5.1: Speech Recognition System

These acoustic feature vectors then go through the *senone evaluation stage*, where the observation probability of each senone is calculated. This probability is also called *senone score*. With the senone scores now available, triphone scores can be calculated

easily (triphones are collection of senones) and is done by the *word decode stage*. This stage calculates the triphones scores of the dictionary words. Also, the word score is calculated, summing all the triphone scores of the component triphones. The probable spoken word is identified based on word scores. Over time, it generates a word lattice over time. The word lattice is then evaluated by the *Word Path Search* for the most likely spoken word sequence in the input speech.

The word decode stage also provides the senone evaluation stage with information about the relevant senones whose score must be evaluated in the next frame. This is shown by 'Feedback' in the figure.

As shown in the previous chapters, an acoustic vector is easily processed within 10ms and the native execution takes only 1% of the computation time. We, in our system, have the Frontend in software which is run on the general purpose microprocessor. Our design proposes ASIC and RAM structures for the senone evaluation and word decode stages since the real time processing (of these two stages) is not possible using a general purpose microprocessor.

We now give detail of the Frontend, then a brief understanding of how the senone evaluation and the word decode works using ASIC blocks we design along with RAM structures we propose.

5.1 Frontend

The system audio device converts the spoken words into the speech signals. The speech signals are sampled over an interval of few milliseconds. The spectral characteristic of speech is assumed to be constant within the interval. Frontend takes the sampled speech signals as input and generates the speech feature vectors as output. It performs a spectral analysis of speech signals. It is assumed that the speech signals correspond only and only to the spoken words on the audio device. i.e, all background words/noise does not contribute to the speech signal. Linear Predictive Coding (LPC) is used for the spectral analysis. We go through the brief detail of each of the steps involved. Figure 5.2 shows the block diagram of a speech recognition frontend. The overall block action is a frame of N samples is processed and a speech feature vector O_t is computed.

The intervals are typically spaced 10 msecs. Blocks are overlapped to give a longer analysis window, typically 25 msecs. The raw signal is then pre-emphasized by applying high frequency amplification to compensate for the radiation from the lips.

Spectral estimates can be computed via linear prediction or discrete Fourier analysis or cepstrum analysis, and the coefficients, *i.e.*, the final acoustic vectors can be obtained via a number of transformations. The most typical method of modern LVR systems is to use the mel-frequency cepstral coefficients (MFCCs). The processing is mainly done in order to satisfy constrains in the acoustical modeling component.

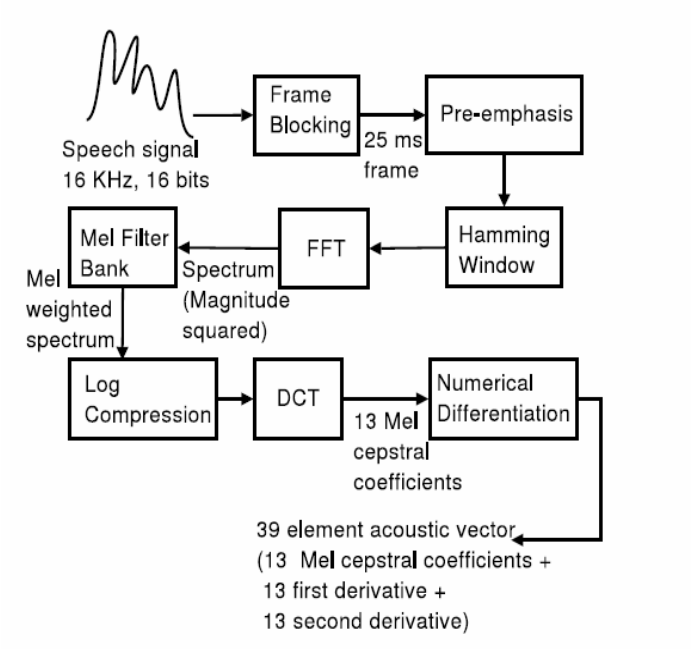

Figure 5.2: Speech Recognition Frontend

The Fourier spectrum of each speech block is smoothed by a mel-scale filter-bank that consists of 24 band-pass filters that simulate the human cochlea processing. The mel-scale is linear up to 1000 Hz and logarithmic thereafter, creating a so-called perceptual weighting to the signal.

From the output of the filter-bank a squared logarithm is computed, which discharges the unnecessary phase information and performs a dynamic compression making the feature extraction less sensitive to dynamic variations. This also makes the estimated speech power spectrum approximately Gaussian. Finally, the inverse DFT is applied to the log filter-bank coefficients, which actually is reduced to a discrete cosine transformation (DCT). DCT compresses the spectral information into lower-order coefficients, and it also decorrelates them allowing simpler statistical modeling.

The acoustic modeling assumes that each acoustic vector is uncorrelated with its neighbors. Due to human articulatory system, this requirement is not well satisfied; there is continuity between consecutive spectral estimates. Second and third order differentials greatly reduce this problem. A linear regression is fitted over two preceding and two following vectors resulting the final acoustic vector with 39 components (each 32 bits wide).

5.2 Senone Evaluation Stage and Word Decode Stage

The observation probability calculated in the senone evaluation stage is not calculated for all senones, but is restricted by the dictionary used in the speech recognition and probability values calculated. The senones are classified in two categories 1) *active* senone, 2) *not-active* senone. The senone for which the observation probability is calculated in the present or the very next frame is an active senone. Else, its a not-active senone. The senones for the context independent phones are always calculated in every frame, *i.e.* the senones composing the phones are always active. The word search logic in the word decode stage interfaces with the dictionary and feeds back to the senone evaluation stage with a set of senones to be evaluated in the next or future frames (active senones) and a set of senones which may be set to not-active. Figure 5.3, shows the basic operation of the system with snapshots of first couple of frames. The utterance is "this".

Now, each stage is processed in one frame interval (10ms). The acoustic feature

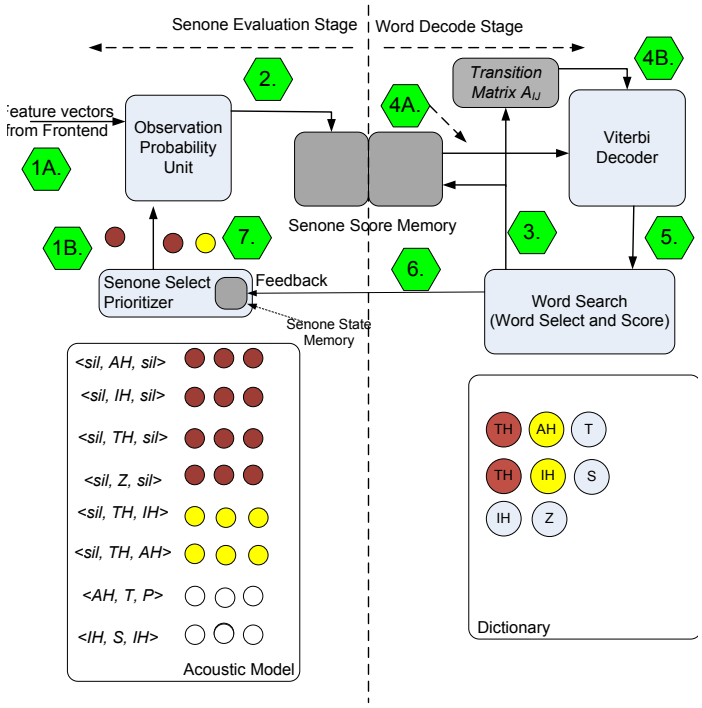

Figure 5.3: Senone Evaluation and Word Decode Stage for first couple of frames

vectors are obtained from frontend processing of speech signals. These vectors are input (path-1A) to the observation probability unit. Since the senone corresponding to context independent phones are always active, (in the figure, this corresponds to the brown coloured nodes in the Acoustic Model) these senone parameters are the inputs (path-1B) to observation probability unit. The observation probability unit calculates the senone score and stores it in the senone score memory (path-2). This repeats till all the active senones are evaluated against the acoustic vector. When 10ms frame interval expires, the senone scores are moved over to the word decode stage for further processing, while a new acoustic vector fills in the senone evaluation stage.

In the word decode stage, the word search logic does the word search with the

active triphones of the word (in the figure, it is brown coloured 'TH' nodes in the dictionary). To evaluate the triphone score, it uses the Viterbi decoder. It parses the triphone and gets the senone composition of the triphone and the requests senone scores from the senone score memory and the corresponding transistion matrix (path-3). The senone scores and the transition matrix is fed to the Viterbi decoder (Path-4A and path-4B). The Viterbi decoder gives out the triphone score back to word search logic (path-5). Depending in the evaluated triphone score it sets the next triphone active (in figure it is yellow coloured triphone nodes in the dictionary). The next triphone is the one next to the active triphone in each word. Word search logic parses the newly set active triphones to get the senones and gives a feedback to the senone evaluate stage to set these senones active (path-6). This concludes the word decode stage.

The word decode stage identifies the words spoken. However, for the same set of input frames the word decode stage may identify multiple similar sounding words. This results in formation of word lattice. The word path search re-evaluates the the HMM states of the identified words for the best path and uses the language model to identify the best word sequence.

5.3 Basic Blocks

5.3.1 Observation Probability Unit

The observation probability is given by equation 2.8 and 2.9. To reduce computational overhead, the calculations are done in logarithmic domain [37]. Therefore, applying logarithm to equation 2.9, we have equation of the following form.

$$log(N(O_t; \mu_j, \sigma_j)) = log(K_1) - \frac{1}{2}\sum(O_{ji} - \mu_{ji})^2 \times \delta_{ji} \tag{5.1}$$

Applying logarithm to equation 2.8 we get

$$log(b_j(O_t)) = log(A_1 + A_2 + ... + A_n) \tag{5.2}$$

The logarithm on addition in the right hand side is evaluated as follows, assuming $A_1 \geq A_2$,

$$\begin{aligned} log(A_1 + A_2) &= log(A_1(1 + \frac{A_2}{A_1})) \\ &= log(A_1) + log(1 + \frac{A_2}{A_1}) \end{aligned} \tag{5.3}$$

Now, $1 \leq 1 + \frac{A_2}{A_1} \leq 2$, so $0 \leq log(1 + \frac{A_2}{A_1}) \leq 0.693$ and is obtained from a lookup table. A Dedicated hardware unit is designed to calculate observation probability with above simplifications. This unit handles 32-bit floating point operations as opposed to fixed point arithmetic in most of the systems. Figure 5.4 shows the block diagram of the unit which calculates observation probability.

The data path consists of an $(X - Y)^2 \times Z$ floating point unit followed by an adder that completes evaluation of equation 5.1. A fused multiply-add unit then

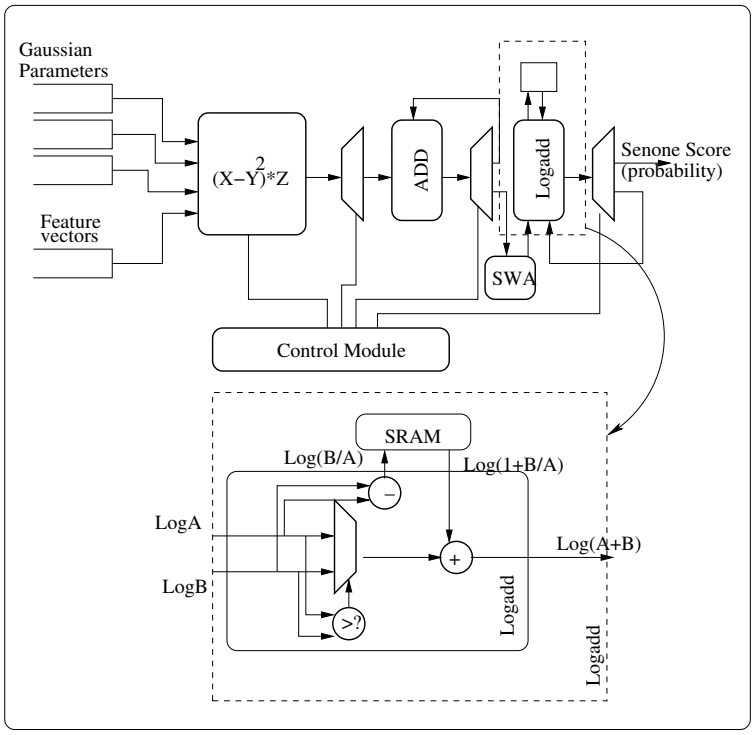

Figure 5.4: Observation Probability (OP) Unit

performs the scale and weight adjustment. The 'logadd' unit completes evaluation of equation 5.2 in logarithm domain. The control unit has a coarse-grain control over most of the arithmetic units and multiplexers. The different mode settings provide coarse-grain control of different stages of the pipeline.

5.3.2 Senone Score Memory

All senones are numbered and this number, called the *senone ID*, is used to access a senone. The senone scores are 32-bit (4 bytes) floating point numbers. Therefore, N senones require $4N$ bytes of storage for the senone score. These scores are evaluated every frame and are stored in a RAM memory. Since, memory sizes are usually in powers of two, we assume the memory to be of size $N'=2^k$, k is an integer, where

$N' > N$. With 6000 senones we have $N' = 8192$. Therefore, the system can support upto 8192 senones. The RAM memory has a 13-bit address line and has 32-bit

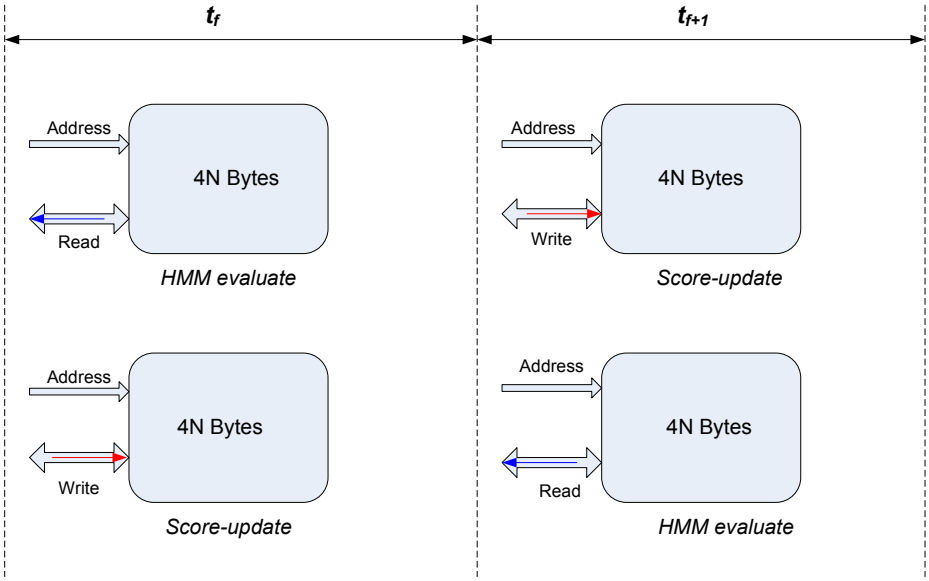

Figure 5.5: Senone Score Memory

output. Two such structures are needed for real time speech recognition. Each structure operates in two different modes, one is *score-update* mode and the other one is *HMM-evaluate* mode. Also, the mode changes after every frame duration ($10ms$). All memory locations (4 byte words) are asserted with value zero immediately after changing from HMM-evaluate mode to score-update mode. The structure in score-update mode stores the senone scores (evaluated by the Observation Probability Unit) of the feature vector. The other structure in HMM-evaluate mode, contains the senone scores of previous frame and these scores are used calculations of triphone HMMs.

5.3.3 Acoustic-Model RAM

Acoustic model is the collection of all the parameters of the senones. The parameter for a senone includes mean variance and all the constants involved in equation 2.9. Each parameter in a senone is a 32-bit (4 bytes) floating point number. 2528 bytes are required for one senone. All senones are similar in structure and have same number of parameters. So an acoustic model memory size is a multiple of number senones it has. A system with 6000 senone requires around 14.5 MB of storage memory for the acoustic model. We, in our design, suggest having the acoustic model in a DRAM for speech recognition. The DRAM is loaded with the acoustic model data from a data flash card memory during the device switch on.

5.3.4 Senone Prioritizer

If the relevant senones fall within the pool of senones calculated per frame, the recognition works as fine as if all senones were calculated. Section 5.2 (path-6 in Fig: 5.3) shows how we can speculate on the which senones have to be set active and calculated first so that the recognition is more likely to be successful. The Senone Prioritizer structure changes the order of sequential computation of the senones and prioritizes the active senones.

The structure contains three small identical RAMs of size 1KB each, small FIFO and a control unit. This can support acoustic models with 8192 or less senones. Each bit position in the RAM represents the state (active /not active) of one senone. The

bit representing the state of a senone is indexed by the the senone number itself. The following figure explains how the SRAMs are used.

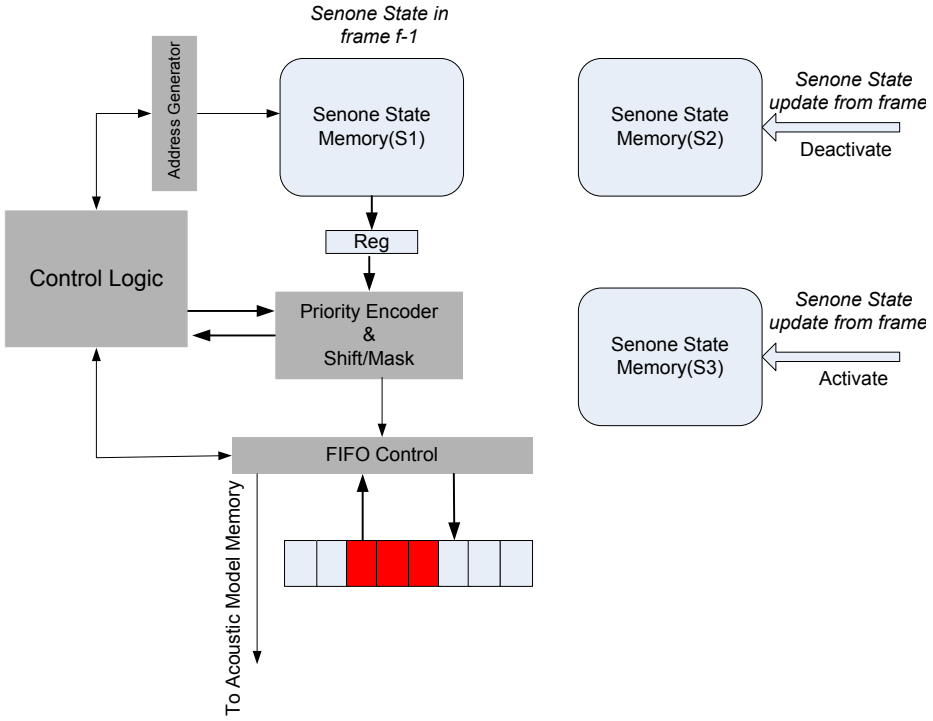

Figure 5.6: Senone State Memory and Prioritizer

$S1$ is holds the active / not-acitve state of senones whose scores are to be calculated first by Observation Probability Unit in the current frame f and is not updated. The SRAM $S2$ has exactly same memory image as $S1$ at the start of the frame but the bits in $S2$ are updated during the frame. The update is setting active to not-active for senones depending on the word decode feedback.

$S3$ contains all zeros at the start of the frame and sets certain senones active depending on word decode feedback. After the current frame ends, *i.e.*, at the start of next frame the $S2$ and $S3$ are merged and stored in $S1$ and $S2$. Then $S3$ is set

to all zeros. The 'active to not-active' and 'not-active to active' is done in separate RAMs since senone state can be set active or not-active many number of times by the word decode feedback. The senone state updated by the feedback should be active if it is set active atleast once by the feedback.

Each block in the FIFO (First In First Out) structure contains the senone id number. It decides the order in which the senones are evaluated, *i.e.*, active senones before the not-active senones. The senone number from the head of FIFO is input to the Acoustic-Model RAM.

The prioritizer Control Unit manages the three RAMs and the FIFO. At the start of a frame it activates RAM image 'merge and copy' It then sweeps $S1$ rows for active senones and puts the senone number in the FIFO locations. Once done with active senones and still in current frame it puts the not-active senone numbers in the FIFO. If the frame time expires it flushes the FIFO.

5.3.5 Viterbi Decoder

The viterbi decoder finds most likely path by solving equation 2.7. This equation is solved in logarithm domain, as there are two advantages, one the floating point multiplications are now additions and second the input $b_j(O_t)$ is readily available in logarithm domain. Applying logarithm, we get

$$log(\delta_t(j)) = max_{0 \leq i \leq N-1}[log\delta_{t-1}(i) + log(a_{ij})] + log(b_j(O_t)) \tag{5.4}$$

Therefore, the viterbi decoder in our system is a 32-bit adder and comparator

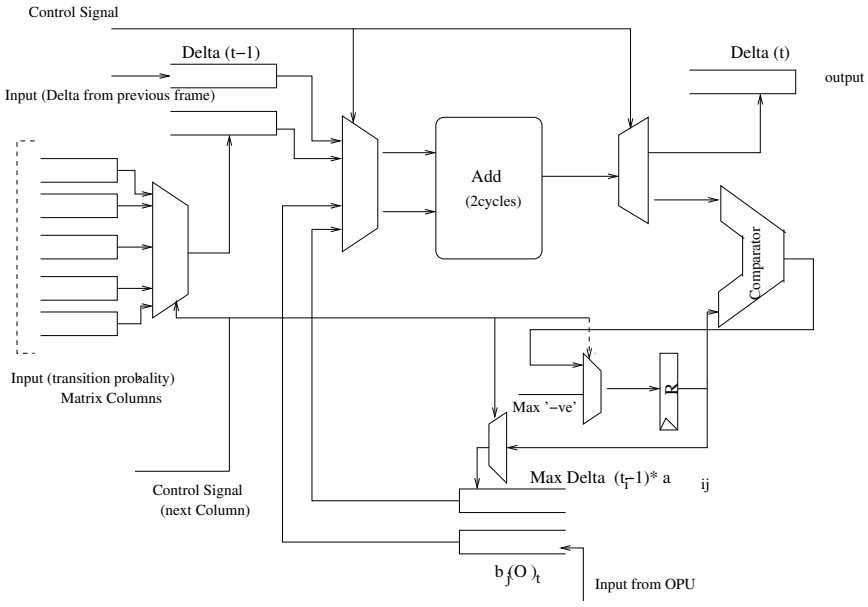

Figure 5.7: Viterbi Decoder (δ = Delta)

multiplexers etc. The design of adder and the comparator, for finding the maximum, are pipelined. The decoder is able to handle multiple state (3, 5, 7) HMMs and therefore, can handle different acoustic models. Figure 5.7 shows the block diagram of the viterbi decoder.

5.3.6 Dictionary Storage

In chapter 2, we have seen that the dictionary words are sequence of triphones. This dictionary is stored in a non-volatile memory (ex. flash memory). A word in the dictionary is identified numerically by its position in the dictionary and is called the *word ID*. Similarly, each of the senones is identified by a unique number called *senone-ID*. Triphones are not numbered, they are identified by the order of senone-ID of the senones in the triphone. Since, a senone-ID is 13 bits in length, a triphone with

3-state HMM is 39 bits wide. Similarly, 5-state HMM and 7-state HMM are 65 bits and 91 bits wide, respectively. Figure 5.8 shows triphone representation of a 3-state HMM using senone-ID.

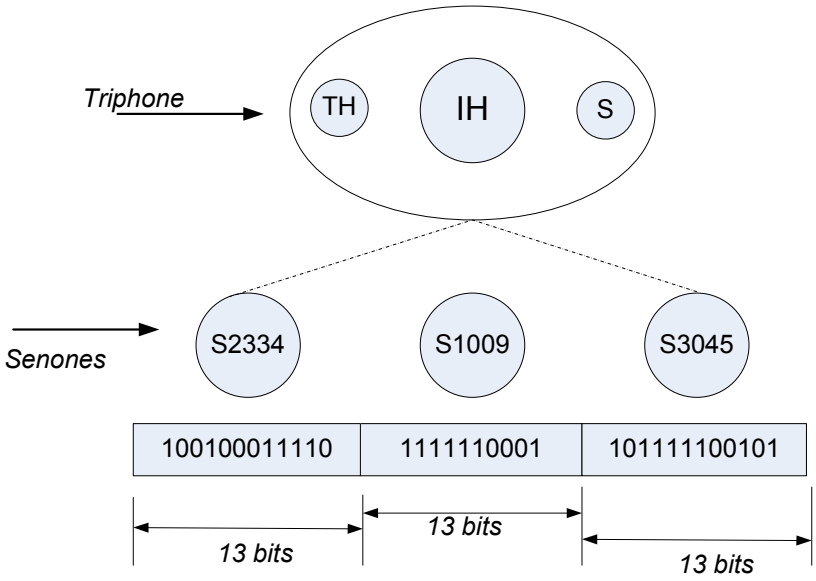

Figure 5.8: Triphone Representation using Senone ID

Now, we use a lexical tree dictionary (section 2.3) where each node is a triphone. In our construct, each triphone of a word is packed with more information about the word and the lexical tree containing the word. We call this a *triphone packet*. This information is stored in four different fields, *Tree_lvl*, *Num_Child*, *Kid_num*, *Leaf_info* and *W_Id* .

The *Num_child* field is 8-bit wide and each bit is for a child triphone node. Therefore, our construct allows a maximum of eight child nodes in a lexical tree dictionary. The *Kid_num* field is 3-bit wide and indicates which numbered child it is (of it's parent node). Therefore, a value of $b'011$ imply that it is the fourth child of its parent

node. The 4-bit *tree_lvl* stores the tree level of the triphone in the lexical tree. There is 13-bit *Leaf_info* field for each child node, which contains the number of leaf nodes resulting from the corresponding child. These four fields are very easily evaluated by modifying the lexical tree build code in Sphinx-III. The *W_Id* is the word-ID field (18 bits wide). Only the last triphone packet of a word has valid *W_Id*, all other triphone blocks within a word have zeros in the *W_Id* field. Figure 5.9 shows an example of word and tree information packed in a triphone packet.

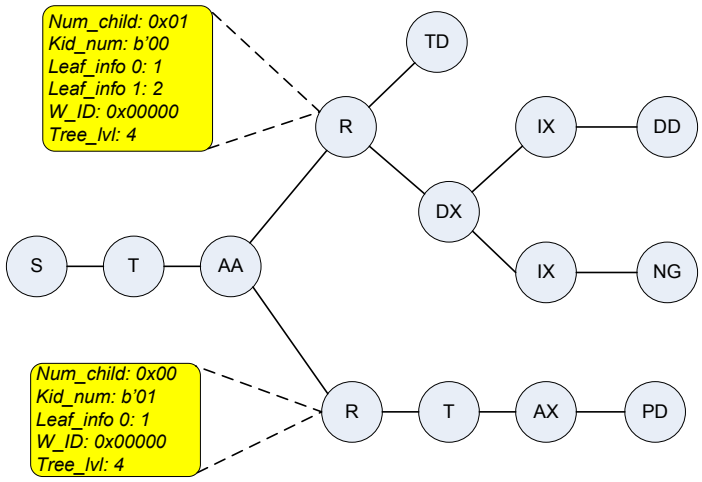

Figure 5.9: Word and Lexical Tree information packed in a triphone packet

Flash memories have low read bandwidth of around 24μs per byte (random access) and cannot support the data rate required for word search in a dictionary. Moreover, word search process requires numerous evaluation and storage (write operation) per frame (10ms) and the flash memories are not good for frequent writes. Therefore, to support the word search a DRAM memory is used. The lexical tree dictionary from the flash memory is unfurled to a flat dictionay using *Tree_lvl* and *Leaf_info* in each triphone packet of triphone composing the tree nodes and copied into the DRAM

memory (Fig 5.10). The triphone packets in this dictionary contains more additional fields as shown in figure 5.11

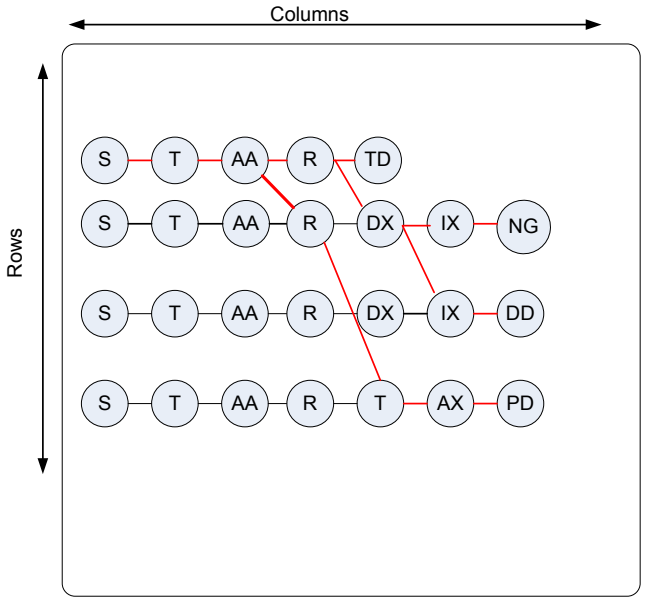

Figure 5.10: Unfurled Flat Dictionary in RAM

These fields are used in calculating and storing various scores during the word search process. The *Sibling_leaf_info* field contains the total number of leaf nodes out of the sibling nodes with *Kid_num* less than the node being considered. This information can be calculated and filled in while unfurling the tree dictionary. The *St_Score* is the score of each state. So for a N state HMM, it is $4N$ bytes wide. *T_Score*(triphone score, 4 bytes) is the maximum of N state scores of the triphone. The field *E_Score* (4 bytes) stands for the exit score of the triphone. *Last_T_Score* field is for storing the triphone score in previous frame processing the last triphone of a word. (4 bytes)

Kid_num, *T_Score*, *Tree_lvl* and *Sibling_leaf_info* are used for deactivating the tri-

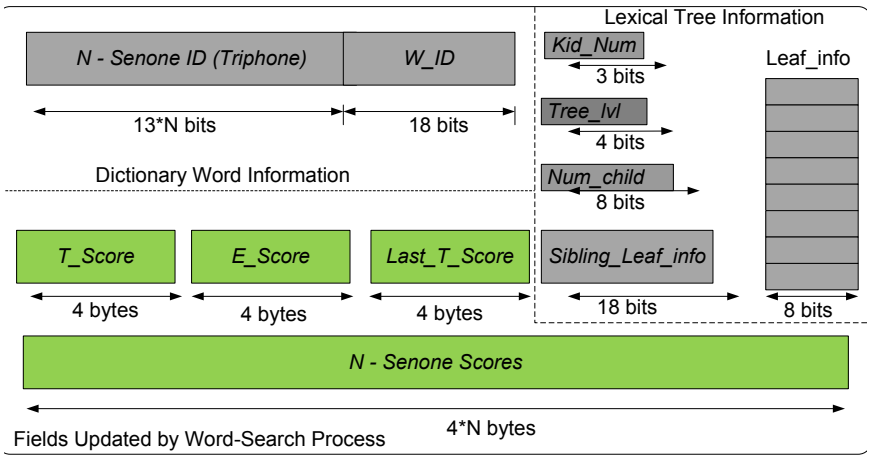

Figure 5.11: Triphone data block in Dictionary RAM

phone packet. *Num_child* field along with *E_score* are used in activating the next triphone packet in the word(s).

Though, we have a flat dictionary in the RAM, to calculate word scores and identify words, we traverse through the dictonary in lexical tree fashion. The flat structure of the dictionary and the word and lexical tree information embedded in the triphone packets makes it very easy to traverse the triphone nodes by calculating the address of child and parent nodes when required.

Lexical Tree Traversal using Flat Dictionary

We can effectively traverse a tree from any node, if we can calculate the addresses of it's parent and child nodes. Let the Row_addr and Col_Addr be the row and column addresses respectively, of a node. Let Row_addr_parent and Col_Addr_parent be the row and column address of it's parent. The parent node address can be calculated by

using the following equations.

$$Row\_Addr\_parent = Row\_Addr - Sibling\_Leaf\_info \quad (5.5)$$

$$Col\_Addr\_parent = Col\_Addr - 1 \quad (5.6)$$

Similarly, addresses of the child nodes can be found

$$Row\_Addr\_child[k] = Row\_Addr + \sum_{i=0}^{k-1} Leaf\_info[i] \quad (5.7)$$

$$Col\_Addr\_child[k] = Col\_Addr + 1 \quad (5.8)$$

Where k is the kth child.

5.4 Word Search Mechanism

The word search mechanism in the word decode stage finds the most likely spoken word in the dictionary. The mechanism involves evaluating the triphones of words in

Figure 5.12 shows the block diagram of the word search logic with the RAM structures. One of the RAM structures contains the dictionary. The dictionary is flat and the words are sequence of triphone packets. The other two RAMs are similar in structure and each is managed as a FIFO. One FIFO maintains a list of indexes (row and column address in dictionary RAM) of active triphone packets in the current frame. The other FIFO builds the list of indexes of triphone packets active for the next frame.

Each element of the list has 18-bit row address, 4-bit column address, 18-bit *P_W_ID* field, 8-bit *Link* field 32-bit *WScore* and 3-bit *W_CNT*. The *Link* field,

P_W_ID, *WScore* and the *W_CNT* field are used for generating the word sequences (section 5.4.1).

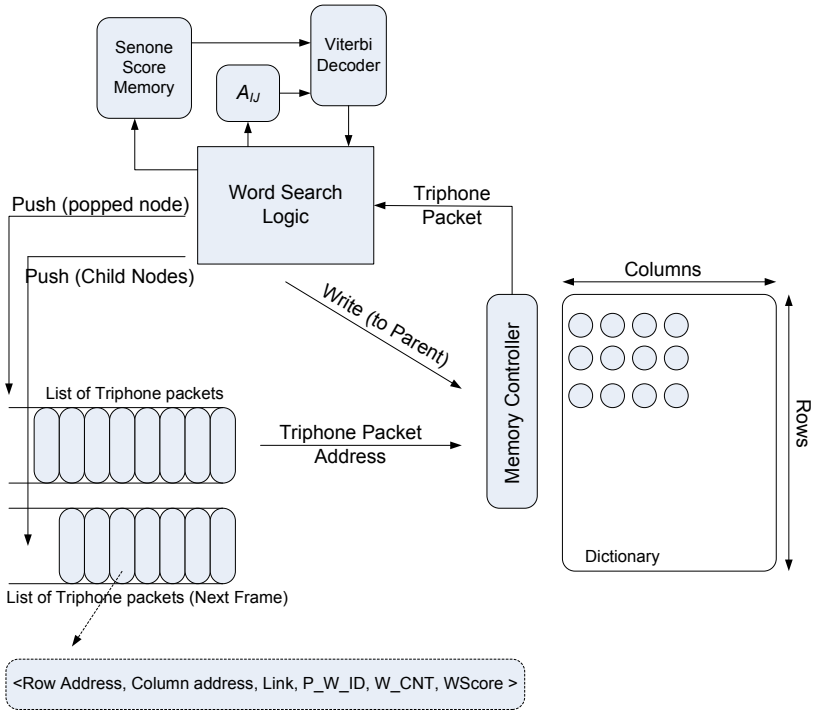

Figure 5.12: Word Search Mechanism

The list containing indexes of active triphone packets for the current frame is popped. The row and column address is used to get the triphones packet from the dictionary. This triphone packet in sent to the word search logic for processing.

The triphone packet is parsed by the word search logic to get the senone IDs. These senone IDs are used to look up the corresponding senone scores and transition matrix. The senone scores and transition matrixis fed to the Viterbi decoder to get the triphone score (T_Score) and the exit score(E_Score).

These two scores and two thresholds $TH1$ (triphone score threshold) and $TH2$ (exit score threshold), determine what action is taken. The action are as follows:

- $T\_Score < TH1$ & $E\_Score < TH2$

  (Action: Present triphone node is deactivated. If triphone is the last triphone of a word and triphone score was above threshold in previous frame then output word)

  - Write Parent$[Num_Kid] = 0$
  - If ($W\_ID > 0$ and $Last\_T\_Score > TH1$) Output W_ID

- $T\_Score > TH1$ & $E\_Score < TH2$

  (Action: Present triphone node active in next frame.)

  - IF ($W\_ID > 0$) $Last\_T\_Score = T\_Score$
  - Push $< Row\_addr, Col\_addr >$ in the other FIFO

- $T\_Score > TH1$ & $E\_Score > TH2$

  (Action: Present triphone node active in next frame. Child triphones are set to active in current frame)

  - Push $< Row\_addr, Col\_addr >$ in the other FIFO
  - IF ($W\_ID > 0$) $Last\_T\_Score = T\_Score$
  - Push $< Row\_addr\_child[k], Col\_addr\_child[k] >$ in the same FIFO

- $T\_Score < TH1$ & $E\_Score > TH2$

  (Action: Present triphone node is deactivated and child triphones are activated

for current frame If triphone is the last triphone of a word and triphone score was above threshold in previous frame then output word)

- Write Parent$[Num_Kid] = 0$
- If ($W\_ID > 0$ and $Last\_T\_Score > TH1$) Output W_ID
- Push $< Row\_addr\_child[k], Col\_addr\_child[k] >$ in the same FIFO

The elements are popped until the list is empty. Since each of the elements in the list represent active triphones in a frame the list should be emptied within one frame interval (10ms) to meet real time recognition. A word is identified when the word search logic detects $W\_ID > 0$ in the triphone packet and it outputs the Word ID.

5.4.1 Word Sequences, Language Model and Word Path Search

The number of words spoken in a speech vary from 150 words per second to 250 words per minute (Fig. 1.2). So for an excited speech four to five words are spoken per second and assuming real time word recognition with the three functional blocks, the Word Path Search must process four or five words per second. The processing involves forming word sequences based on word scores and language model biases. We, in our system, have the Word Path Search block implemented in software and assume late binding of the language model, where the N-gram language model biases are calculated after the Nth word is recognized. We use the language model used in Sphinx. This is a software which does a table look up based on the word sequences and reports the bias score.

We now look at how the Word Path Search in interfaced with Word Decode and it forms the word sequences. The triphone packet engine detects a word when it processes the last triphone of the word having end of a word information ($W\_ID > 0$). It flags the detected word as a possible spoken word, when $T\_score < TH1$ and $Last\_T\_Sccore > TH1$. End of a word is first detected when the recognition process enters the last triphone of the word (*i.e.* $E\_Score > TH2$ for the last but one triphone and $T\_score > TH1$ for the last triphone) . This same 'end of the word' may be detected multiple times over next few consecutive frames. Therefore, both T_Sccore and $Last\_T\_Score$ are checked to avoid flagging of the same word multiple times over adjacent frames as possible spoken word.

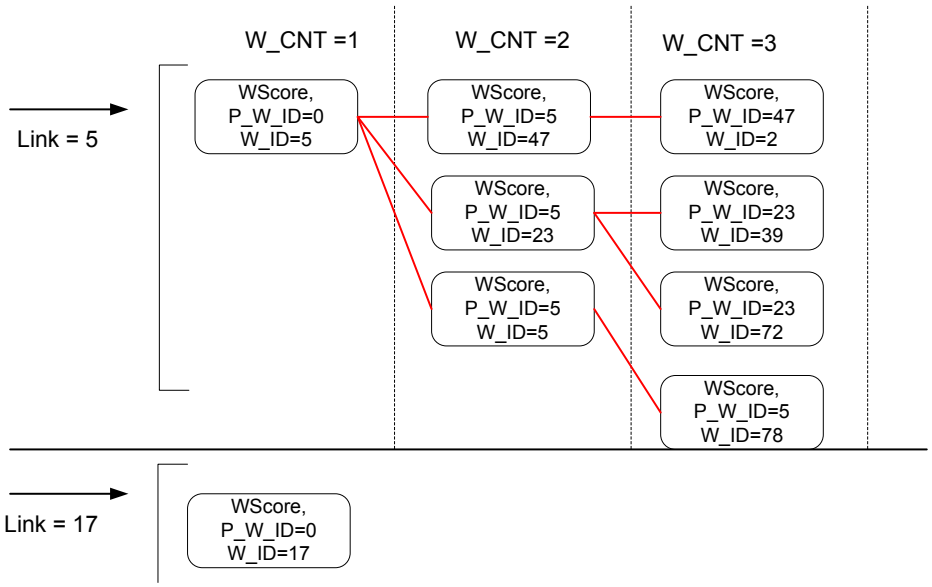

Figure 5.13: Word Path Search Table

When a word is flagged as possible spoken word while processing its last triphone, the ASIC dumps the *WScore*, *Link*, *P_W_ID* and *W_CNT* from the corresponding

FIFO element. These fields are read by the Word Path Search block and the values are entered into a table (Fig. 5.13), maintained in software. The *Link* and *W_CNT* are used for indexing into the table. The *Link* is *W_ID* of the first word recognized by the recognition system. Since the recognition system can detect more than one first word (with different word scores), there can be more than one *Link* in the table. At every word position there can be more than one word for same sequence of previous word. The field *P_W_ID* (previous word ID) helps in tracing back the word sequence.

Language model bias is attached for each of the word sequences. Score of a sequence of words is determined by adding the word scores and the language model score with (equal or different weight coeficients). The highest scoring sequence of words is the recognized spoken sequence of words.

Chapter 6

Results

In this chapter, we evaluate our proposed speech recognition system design. Our system has a general purpose microprocessor and two ASIC blocks and few RAM structures. The general purpose microprocessor executes the software parts of speech recognition and also maintains overall control of the speech recognition system. We present the performance, area and power of the design and then compare our design with related work. We revisit our goals before listing the results of our design.

Our goal is to find a solution for large vocabulary real time speech recognition on a mobile device. As described in previous chapter, we divide the speech recognition in four functional parts. These parts are the Frontend, Senone Evaluation Stage, Word Decode Stage and the Word Path Search. The frontend and the word path search are implemented in software and Senone Evaluation and the Word Decode stage are implemented as a co-processor.

6.1 Performance and Power Evaluation

The amount of computation involved for different input speech in a HMM based speech recognition, given the same acoustic and language models, is input speech dependent. Therefore, we first look at the minimum requirement for a speech recognition system. We assume that the minimum requirement for large vocabulary speech recognition is the size of the dictionary used by Ney et al. [19]. This dictionary contains 12306 words. The following table 6.1 shows the average number of active nodes for different sets of inputs. These active nodes must be scored within one frame. The number active nodes also give us information about worst case number of senones that must be evaluated within one frame. This case arises when, in a N-state HMM, none of the active triphone nodes share even a single senone and the number of senone evaluated is *MIN (Number of senone in the acoustic Model; N*number of active nodes)*

Table 6.1: Phoneme arcs and Active arcs in 12306 word Lexical Tree Dictionary

Levels	1	2	3	4	5	6	≥7
Phoneme nodes	28	331	1511	3116	4380	4950	29.20
Average Active nodes	23	233	485	470	329	178	206

So the worst case number of senones is 1455 (485x3). We show that our design can easily handle computation for above number of senones and triphone nodes. Our design can handle more number of triphone nodes and senones therefore can support larger dictionary (*i.e.* larger vocabulary).

6.1.1 Senone Evaluation Stage

The senone evaluation stage is both computation and memory intensive, and is a bottleneck for real time speech recognition. We propose a dedicated ASIC connected to RAM through a DMA and having small embedded memories for the senone evaluation stage. The ASIC calculates the observation probability and the senone prioritizer logic.

The Observation Probability Unit takes 400 cycles to compute a senone score. Therefore, within the duration of one frame (10ms), with clock frequency 50MHz, score of 1250 senones can be computed. We now evaluate the worst performance senario, which may arise when each of the triphone being scored in a particular frame share no senone with the other senones. In such situation, assuming 3-state HMM 1250 senone is 412(=1250/3) triphones. Two such units can handle the required number of triphones as shown the Table 6.1.

The following table shows the power and area estimates for the ASIC and the memories. ASIC power and area are obtained by synthesizing (Synopsys Design Compiler) the RTL (Verilog) written for the module. The power and area for the memories estimated using CACTI 4.0 [11]

6.1.2 Word Decode Stage

The word decode stage includes the Viterbi decoder and the Word Search Logic. The Viterbi decoder is capable of handling upto 7-HMM states and takes 50 cycle

Table 6.2: Power and Area of the Observation Probability Unit (0.18μ Tech)

Design Unit	Area	Power
ASIC (OPU+Priority Logic)	1.2 mm^2	32 mW
2 Senone Score Memory - 6000 senones Each 24KB, 1 read/write port, each entry 8 bytes	3.4 mm^2	62 mW
2 Senone State Memory - 6000 senones Each 1KB, 1 read/write port, each entry 8 bytes	0.23 mm^2	33 mW
Log Table 1KB, 1 read port, each entry 8 bytes	0.12 mm^2	16.5 mW

to compute a triphone score. Rest of the word search logic takes another 50 cycle - to read the triphone packet (before triphone score is calculated) from the dictionary, update the FIFOs and write back the triphone packet back in the dictionary. So score of one triphone is computed in 100 cycles. With an operating frequency of 50MHz this design is capable of scoring 5000 triphones (nodes) within 10ms. Peak requirement for number of nodes with 12306 word dictionary [19] is less than 500 nodes (Table 6.1). Therefore, the performance of our design for the Word Decode Stage is ten times better than the performance required for real time recognition. The following table shows the power and area figures for the word search and Viterbi decoder logic

Table 6.3: Power and Area of the Word Decode Stage (0.18μ Tech)

Design Unit	Area	Power
ASIC (Viterbi + Word Search + FIFO logic)	0.7 mm^2	11mW
2 Word Search FIFO for 1500 triphones Each 1KB, 1 read/write port, each entry 8 bytes	0.7 mm^2	52 mW

6.2 Comparison with Related Work

We now compare our design (D1) with other implementations [27, 24] of hardware speech recognition. Binu et al.(design D2) designed accelerator for probability calculation. Krishna et al. (design D3) and Ullas [7] (design D4) proposed more complete speech recognition system.

The accelerator proposed by Binu et al., speedup up the probability computation (senone score evaluation) of speech recognition application. It assumes that the acoustic model to be in the computer system memory hierarchy (Disk/RAM/Cache). The design proposes that instead of computing the score for each senone one by one for every input acoustic vector, one senone is scored for ten consecutive input acoustic vectors. So the acoustic parameters for a senone to be scored are required every 10 frame instead of every frame. This reduces the bandwidth requirement by a factor of ten. This design considers this reduced bandwidth requirement as the key feature [27]. We classify this design as 'D2 (Intended)'. However, the drawback of the design (not considered by Binu et al.) is that the design breaks the feedback loop which sets smaller set of senones active based on the triphone score of the words in the dictionary. To have a correct recognition without the feedback loop, all senones must be evaluated. This in turn increases the bandwidth requirement back up. We classify this as 'D2 (Actual)'. Figure 6.1, shows the bandwidth required for computing senone scores using an acoustic model with 1500 senones. The bandwidth required is linearly dependent on the number of senones evaluated except for the design D2 (labeled -

'Actual') since it computes the senone scores of all 1500 senones.

Design proposed by Krishna et al., has multiple processing elements to exploits the parallelism in speech recognition application. The senones are independent of each other and they can be computed in parallel. Look up for the words in the dictionary (for lexical tree dictionary - subtree corresponding to each child) can be done in parallel. Each processing element is seven stage pipeline with a small cache (L1 cache) to exploit spatial locality. The design also proposes as additional cache (L2 cache) to improve performance (design D3 with L2).

The design proposed by Ullas, is an ASIC design for the senone score evaluation, word decode and the word path search. The senone score evaluation block is called the Gaussian estimator block in his design and the word decode and word path search are both grouped together and is labeled as Viterbi decoder block. The Gaussian estimator block has features to control the input acoustic model parameters and are used in skipping frames, probability distributions input vector length etc. The word search and word decode are handled using a RAM memory to store the senone scores, triphones sequences, words formed and various word paths formed during the recognition process. This involves multiple lookup to senone score memory, dictionary and the language model.

Design D1, D2 and D4 interface with memory through DMA, so bandwidth calculation is amount of data read per sec. For design D3, bandwidth is measured by using the number of L1 misses (L2 misses for D3 with L2 cache), cacheline size, number of

execution cycles and the processor frequency.

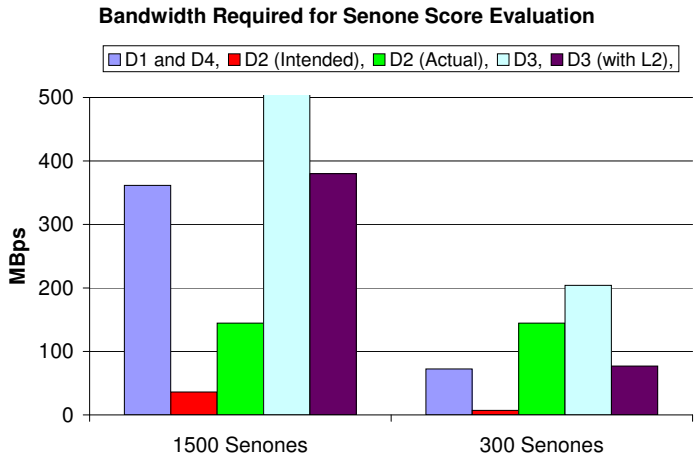

Figure 6.1: Bandwidth Comparison Senone Evaluation

In HMM based speech recognition, computation of the senones followed by computation of the triphones is the key components in looking up a word in the dictionary for a possible match. We, therefore, compare the designs based on performance per senone and triphone. Table 6.6 compares different design performances for calculating the observation probability. Though, design D2 performs as well as our system,

Table 6.4: Performance per Senone and Senones Evaluated Per Frame

Design	Cycles per senone	Num of Senones Evaluated per frame (Clk. Freq.)	Triphones (MIN)
D1	400 cycles	1250(50 MHz)	416
D2	400 cycles	1250(50 MHz)	416
D3	13000 cycles	312 (400 MHz)	104
D4	400 cycles	1250(50 MHz)	416

the design is unable to exploit the feed back from the word search. So all senones computed by design D2 may not be useful, whereas, our design computes all useful senones. The following table 6.5 compares performance per triphone. For design D3,

if all accesses to a triphone data structure are L1 cache hits, it results in minimum number of cycles in processing one triphone (corresponds to maximum number of triphones supported by the design). Similarly, if all accesses miss in the cache hierarchy, it results in maximum number of cycles in processing one triphone (corresponds to minimum number of triphones supported by the design) Our word search, which uses an ASIC to traverse through the lexical tree performs, is 20 to 40 times better than one unit of design D3. The following figure shows the bandwidth requirement

Table 6.5: Performance per Triphone and Triphones Evaluated Per Frame

Design	Cycles Per Triphone	Maximum Triphones Supported
D1	100	5000 (50 MHz)
D3	2001(Min) - 4286(Max)	1999(Max) - 933(Min) (400 MHz)

of each design when 1500 senones and 6000 thousand triphones are evaluated. The bandwidth for triphone evaluation is linearly dependent on the number of triphones evaluated per frame. The bandwidth required for D3 is 11 times and design D4 is 4 times more than our design.

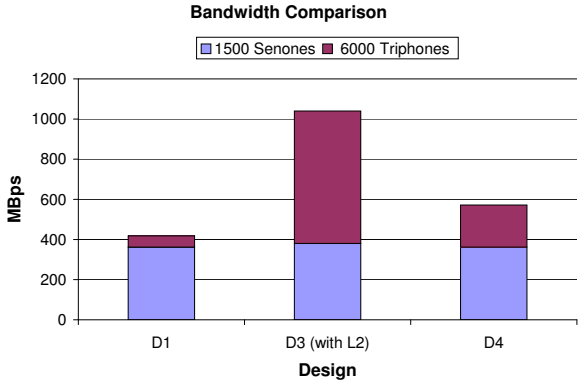

Figure 6.2: Bandwidth Comparison - Senone and Triphone Evaluation

We now look at the power consumption comparison of our design (D1) and the

other designs (D2, D3 and D4). The following shows the power consumption comparison of the designs. The power reported is power consumed by one unit multiplied by number of units required to handle 6000 senones (maximum number of senones).

Table 6.6: Comparison of Power Consumption - Senone evaluation

Design	Units	Power	Tech.
D1	5	160 mW (50 MHz)	0.18μ
D2	5	1.8 W	0.25μ
D3	20	1.4 W	0.18μ
D4	3	300 mW - 750 mW (50 MHz)	0.25μ

The following table shows the power comparison for triphone evaluation. The "Units" column shows the number of units required to have same performance as one unit of our design.

Table 6.7: Comparison of Power Consumption - Triphone evaluation

Design	Units	Power	Tech.
D1	1	11 mW (50 MHz)	0.18μ
D3	5	350 mW	0.18μ
D4	1	80 mW (200 MHz)	0.18μ

One unit of design D4 for Gaussian estimator is 2.856 mm^2 (0.25μ Tech.) and the Viterbi decoder unit is 2.278 mm^2 (0.18μ Tech.). The corresponding units in our design (D1), are senone observation probability unit - 1.2 mm^2 (0.18μ Tech.) and word search logic - 0.7 mm^2 (0.18μ Tech.).

The performance, area and power consumption comparisons clearly indicate that our design is better than other proposed designs for real time speech recognition on a mobile embedded device.

Chapter 7

Summary

This thesis proposed a design for large vocabulary continuous real time speech recognition on handheld mobile devices. Hidden Markov models (HMM) are used for modern day speech recognition system. In HMM based recognition, the recognition system assumes that the dictionary words are sequence of basic sounds. Speech is sequence of words and hence sequence of basic sounds. The basic sounds in a speech are assumed to follow a Markov process. The basic sound models are available with the recognition system. In recognition process, permutations of these basic models are probabilistically compared with the input speech. The best matching sequence of basic sounds which also creates sequence of words is recognized as the input speech.

We evaluate the speech recognition software application (CMU-Sphinx) on an aggressive processor for mobile domain and observe that the application is not good for real time recognition. The speech application is broken down into three subparts (based on functionality) to analyze in greater detail and it is observed that the application is computation as well as memory intensive. Although, the application does

real time recognition when run on a multi-GigaHertz desktop or server with large cache and other resources, in mobile domain we are limited by power budget.

We propose a co-processor, having low power ASIC structures interfaced with RAMs, for the computation and memory intensive parts of the speech recognition application along with a processor to execute the software part. We compute word triphone scores by developing a new method to traverse through the lexical tree dictionary and our design enables probability computation of only the active senones.

The ASIC structures are optimized for power and performance at system level design (are fully pipelined and share resources) as well as gate level (uses clock gating). The real time large vocabulary speech recognition is achieved. The total power usage for the ASIC blocks, required SRAM memories and a 32MB DRAM memory (for 60000 dictinary words) is around 321 mW (0.18μ Tech.). Therefore, a single AA battery (1500mA) can support more the eight hours of real time large vocabulary speech recognition.

Bibliography

[1] http://cmusphinx.sourceforge.net/ .

[2] http://htk.eng.cam.ac.uk/.

[3] http://simplescalar.com/.

[4] http://www-4.ibm.com/software/speech/ .

[5] http://www.speech.philips.com.

[6] http://www.voxtec.com.

[7] Hardware Implementation of a Low Power Speech Recognition System. In *Ph.D. thesis, North Carolina State University, Raleigh, N.C*, 2007.

[8] Suhm B. and Waibel A. Towards Better Language Models for Spontaneous Speech. In *ICSLP*, pages 831–834, Vol 2. 1994.

[9] P. Baggia, J.L. Gauvain, Kellner A., Perennou G., Popovici C., Sturm J., and Wessel F. Language Modelling and Spoken Dialogue Systems - The ARISE Experience. In *Proc. Sixth European Conference on Speech Communication and Technology*, Sep. 1999.

[10] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA*, pages 340–351, 2005.

[11] Tarjan D., S. Thoziyoor S., and Jouppi N. CACTI 4.0: An Integrated Cache Timing, Power and Area Model. In *Technical report*. HP Laboratories, Palo Alto, June 2006.

[12] D.Jurafsky. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. 1962.

[13] Alleva F., X. Huang, and Hwang M.-Y. Improvements on the Pronunciation Prefix Tree Search Organisation. In *ICASSP*, pages Vol 1, 133–136, 1996.

[14] Fissore, L. Laface, P. Micca, and R. G. Pieraccini. Lexical access to Large Vocabularies for Speech Recognition. In *IEEE Transactions on Signal Processing*, pages 1197–1213, Vol 37, Issue 2, Aug 1989.

[15] E Fosler-Lussier. Dynamic Pronunciation Models for Automatic Speech Recognition. In *Ph.D. thesis, University of California, Berkeley*, 1999.

[16] Antoniol G., Brugnara F., Cettolo M., and Federico M. Language Model representations for Beam-Search Decoding. In *International Conference on Speech, and Signal Processing*, pages 588–591, 1995.

[17] J. Goodman and S.F. Chen. An Empirical Study of Smoothing Techniques for

Language Modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, 310-318 1996.

[18] Vishwa N. Gupta. Phoneme based Speech Recognition. In *The Journal of the Acoustical Society of America*, Vol 98, Issue 3, Sep. 1995.

[19] Ney H., Haeb-Umbach, R. Tran, and Oerder M. Improvements in Beam Search for 10000-Word Continuous Speech Recognition. In *Proceedings of the IEEE Int. Conf on Acoustics, Speech and Signal Processing*, pages 9–12, 1992.

[20] X. Huang, F. Alleva, H. W. Hon, M. Y. Hwang, K.F. Lee, and R.Rosenfeld. The SPHINX-II Speech Recognition System: An Overview. In *Computer Speech and Language*, pages 137–148, 1993.

[21] Mei Huh Hwang and Xuedong Huang. Subphonetic Modeling with Markov States - Senone. In *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages I 33– 36, March 1992.

[22] Jyh-Shing Roger JANG and Shiuan-Sung LIN. "Optimization of Viterbi Beam Search in Speech Recognition. In *ISCSLP*, 2002.

[23] F. Jelinek. *Self-organized language modeling for speech recognition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[24] Rajeev Krishna, Scott A. Mahlke, and Todd M. Austin. Architectural optimiza-

tions for low-power, real-time speech recognition. In *CASES*, pages 220–231, 2003.

[25] Biing-Hwang Juang Lawrence Rabiner. In *Fundamentals of Speech Recognition*. Prentice Hall signal processing series, 1993.

[26] Ronald W. Schafer Lawrence Rabiner. In *Digital Processing of Speech Signals*. Prentice Hall signal processing series, 1978.

[27] Binu Mathew, Al Davis, and Zhen Fang. A Low-Power Accelerator for the SPHINX 3 Speech Recognition System. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '03)*, pages 210–219, October 2003.

[28] S.J. Melnikof, S.F.Quigley, and M.J.Russell. 10th ieee symposium on field-programmable custom computing machines (fccm 2002), 22-24 april 2002, napa, ca, proceedings. In *FCCM*. IEEE Computer Society, 2002.

[29] N.Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, 1965.

[30] Sergiu Nedevschi, Rabin K. Patra, and Eric A. Brewer. Hardware speech recognition for user interfaces in low cost, low power devices. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 684–689, New York, NY, USA, 2005. ACM Press.

[31] Clarkson P. and Rosenfeld. R. Statistical Language Modeling using the CMU-Cambridge Toolkit. In *Eurospeech*, 1997.

[32] S. Phillips and A. Rogers. Parallel speech recognition. In *EUROSPEECH*, pages 242–245, 1997.

[33] Kneser R. and Ney H. Improved Backing-off for M-gram Language Modeling. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 181–184, May 1995.

[34] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[35] R. Sharp, E. Bocchieri, C. Castillo, S. Parthsarath, C. Rath, M. Riley, and J. Rowland. The Watson speech recognition engine. In *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*, May 1997.

[36] J.K.C Soong and F.K. Lin-Shan. Large Vocabulary Word Recognition based on Tree Trellis Search,. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 137–140, Vol, April 1994.

[37] S.Srivastava. Fast Gaussian Evaluations in Large Vocabulary Continuous Speech Recognition. In *M.S. Thesis, Department of Electrical and Computer Engineering, Mississippi State University,*, Oct. 1999.

[38] Dragon Systems. Dragon NaturallySpeaking SDK, Integrating a Runtime into Your Speech-Enabled Application. 1999.

[39] A. Waibel and K. Lee. . In *Readings in Speech Recognition*. Morgan Kaufmann Publishers Inc., 1990.

[40] S. Young. Large vocabulary continuous speech recognition: A review. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding, pages 3–28, Snowbird, Utah, December 1995. IEEE.*, 1995.