HARDWARE ACCELERATOR FOR HMM BASED SPEECH RECOGNITION USING APPROXIMATE COMPUTING TECHNIQUES

A Thesis

by

HARIHARAN BHAGAVATHEESWARAN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Committee Members,	Jiang Hu
	Duncan M. Walker
Head of Department,	Miroslav M. Begovic

December 2015

Major Subject: Computer Engineering

ABSTRACT

This thesis presents a hardware design for recognizing speech using phoneme-level Hidden Markov Models (HMMs) and proposes two alternative designs using approximate computing techniques for area and energy optimizations.

An initial hardware design is proposed to model a speech recognition system using the log-Viterbi algorithm approach. Two more hardware designs using various approximate computing techniques and modifications to the log-Viterbi algorithm are also proposed, that are shown to consume lesser area and power.

The work also presents the performance analysis in terms of recognition accuracy and hardware evaluations in terms of area, switching and leakage power and energy dissipation of all three designs. The results prove that the usage of approximate computing helps reduce area and power, with a minor compromise on accuracy. The design using approximate computing is also capable of running at a higher frequency with quicker execution time and lesser energy consumption.

For applications where accuracy is vital, the thesis also proposes an adaptive system which can operate in two modes – one at a higher frequency, with slightly lesser accuracy and another at a lower frequency, with better accuracy and capable of dynamically switching from one mode to another.

ACKNOWLEDGEMENTS

I would like to acknowledge the support of my advisor and committee chair, Dr. Peng Li. He has been extremely patient and helpful through the entire process of this work. Right from the day of meeting him to discuss the prospects of the work to final stages of its completion (and I am sure beyond that as well), he has always been encouraging me. He motivates me to push further and strive for further improvement at every stage of the research. This work was possible largely due to his expert guidance and I feel proud and privileged to have worked with him.

Dr. Li's influence spreads to his research group as well, especially to fellow graduate student, Qian Wang. He has been my go-to man for various tips, advice and inspiration. Without his help, I would have faced a lot more difficulties during the course of my research. I thank him for all the support and guidance as well.

I would also like to thank my committee members Dr. Jiang Hu and Dr. Duncan M. Walker, for their patience and support through the process. Their inputs also helped fine-tune my work and the results.

My expertise on speech processing and recognition (which is an important subject of this thesis) was limited at the start of this work. Hence, the experience gained through a course on Speech Processing by Dr. Ricardo Gutierrez-Osuna was immensely useful in helping me throughout this process and I express my gratitude to him. The course clarified

a lot of concepts which till then had appeared blurred and vague. It also gave hands-on experience on common tools and benchmarks used in the field.

I also acknowledge the support of the staff in the Department of Electrical and Computer Engineering at Texas A&M University for providing several resources and helping at various stages of this research work.

My sincere thanks to my friends and all others who have had a role to play in this work and for motivating me.

Last but as always not the least, I would like to thank my parents Mr. Bhagavatheeswaran H and Mrs. Latha V for all the support they showed in me for completing this work. Their belief and encouragement went a long way in helping me complete my research successfully.

NOMENCLATURE

CLA	Carry Look Ahead
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
DC	Design Compiler
DFT	Discrete Fourier Transform
HMM	Hidden Markov Model
HTK	Hidden Markov Model Toolkit
MFCC	Mel-frequency Cepstral Coefficients
S-o-C	System on Chip
STFT	Short Time Fourier Transform

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

Speech recognition is a popular application in today's world. Speech forms an integral part of the space occupied by gesture and finger touch, as means of interaction with electronic gadgets, be it mobile phones or cars. It is used as means of searching the web, texting or emailing friends and colleagues, to control the multimedia or for navigation in a car. Speech recognition also finds a place in the treatment of speech impairment, especially in children suffering from dysarthria (Rosen & Yampolsky, 2000).

Current smartphones and other gadgets have speech recognition applications built into them, which are software-based (Schuster, 2010). Apart from these, there are various systems built entirely on software, which do speech recognition or provide a toolkit to develop software that can recognize speech. HTK (Hidden Markov Model Toolkit) (Young et al., 1997) and Sphinx-4 (Walker et al., 2004) are two such software currently in use.

To make the speech recognition accurate and speaker independent, recognition models require sufficient training and use various algorithms for recognition, both of which are computationally intensive (Lee, 1988). With speech recognition being widely implemented in many devices currently, the processor is under heavy utilization performing the recognition apart from its own other tasks. On its own, the processor is undergoing various enhancements with respect to performance, resulting in multi-core and multi-threaded systems being prominent. A corollary of the results of (Esmaeilzadeh,

Blem, St Amant, Sankaralingam, & Burger, 2011) shows that having dedicated processors or hardware accelerators for computationally intensive tasks are better ways to enhance performance and offload the main CPU.

In a way similar to having dedicated hardware for signal processing, this thesis proposes a hardware design for implementing the speech recognition task. This allows the main CPU in a System-On-Chip to be responsible only for invoking the hardware for the recognition and can be offloaded from the actual recognition process. The motivation for such a design arose from the fact that hardware implementations can be faster than software.

While implementing a new hardware design, area and power are of utmost concern. The implementation of a complex software task on the hardware is expected to consume a large area. A computationally intensive algorithm translates to large power dissipation. Hence the challenge is to find an optimal balance between the area and power consumption of the hardware and the speed and accuracy of the implementation.

This work discusses an implementation of the hardware design and looks at a couple of optimization techniques for reduction in area and power. The techniques used for optimization revolve around the concept of approximate computing. Approximate computing involves techniques by which deliberate errors are introduced in the computation task to reduce energy consumption and enhance speed, without affecting the accuracy of the final output significantly. This is a popular design paradigm being used

for various applications (Hegde & Shanbhag, 2001), (Kim, Zhang, & Li, 2013), (Shao & Li, 2014), (Kim, Zhang, & Li, 2014) & (Shao & Li, 2015).

This thesis is organized in the following way; details of speech recognition are explained in Section-2, covering aspects of speech recognition, feature extraction and basic structure of a typical speech recognition unit. Section-3 discusses Hidden Markov Models, their usage in the speech recognition process and the Viterbi algorithm for obtaining the optimal state transition sequence. Section-4 describes necessary modifications to the standard Viterbi algorithm to facilitate easier implementation on hardware and discusses previous work that has been done related to hardware implementation of speech recognition. Section-5 details the proposed design with schematic level details of each arithmetic or logical unit used in the process. Section-6 covers customized optimization techniques for improving area and power dissipation of the proposed design. The optimization techniques are along the lines of basic approximate computing. Section-7 implements approximate computing techniques to various elements within the proposed design. Performance analysis and hardware evaluation results are presented in Section-8, with a description of an adaptive model which is capable of supporting both the approximate computation techniques. Section-9 discusses possible enhancements to this work in the future and Section-10 provides a summarized conclusion to the thesis.

2. SPEECH RECOGNITION

The goal of speech recognition is to convert the spoken speech into text. A speech recognition system is a special case of a pattern recognition task, with speech being the pattern and the recognition task being the task of identifying the text that represents the speech, with the maximum probability out of a possible number of texts. A detailed speech recognition process is described in (Lawrence R. Rabiner & Juang, 1993).

2.1. Speech Analysis

Speech recognition is a form of speech analysis wherein the recognition of speech essentially translates into recognition of words or phonemes. *Phonemes* are the smallest meaningful unit of speech in a language (İnce, 1992) and the physical sound produced during the articulation of a phoneme is called a *phone*. A phoneme can correspond to multiple phones due to the variations that can occur during utterance.

During a speech utterance, the articulation of a phoneme is affected by its neighboring phonemes – the ones preceding and succeeding it. This phenomenon is called co-articulation (Ohala, 1993).

2.2. Basic Principle of Recognizing Speech

Given an observation speech sequence O, and a possible word sequence W, the recognition task is to maximize $P(W|O)$ or the recognized word/text ($\widehat{W}$) is the word with the maximum posterior probability.

$$\widehat{W} = \arg\max_{W} P(W|O) \tag{2.1}$$

Applying Bayes' theorem,

$$P(W|O) = \frac{P(O|W) * P(W)}{P(O)} \tag{2.2}$$

As $P(O)$ is not dependent on W, eqn. (2.2) can be incorporated in (2.1) as shown in eqn.(2.3), so that the recognition essentially involves computing $P(O|W)$ and $P(W)$.

$$\widehat{W} = \arg\max_{W} P(O|W) * P(W) \tag{2.3}$$

The first term in eqn. (2.3) $P(O|W)$, is generally obtained using statistical models, termed as acoustic models and denotes the probability of observing the sequence O for the specific word model W. Models are built for various sub-word or word units and $P(O|W)$ is computed for all such units.

The second term in eqn. (2.3) $P(W)$, is computed from language models and denotes the probability of the occurrence of the current word/sub-word unit, constrained to the syntax of the language/application in use.

In this thesis, the sub-word units used are phones. Phones are the basic units of speech analysis.

2.3. Feature Vectors

The observation sequence used in eqn. (2.3) is the input speech to the recognition system. This input speech is converted to feature vectors before they can be processed by

the system. Mel-frequency Cepstral Coefficients (MFCCs) have been proven to best represent speech in terms of its features (Davis & Mermelstein, 1980) and hence are one of the most popular features used in speech recognition.

The human auditory system contains the *Basilar Membrane* which is responsible for analyzing the frequencies of sounds. The *basilar membrane* is a coiled structure within the *cochlea* of the ear and has different properties at different points across its length, which affect the frequency to which that point is sensitive to (Bacon, Fay, & Popper, 2004). The membrane is sensitive to frequency in a non-linear fashion, with the scale being linear till 1000 Hz and logarithmic beyond. The *Mel* scale is an approximate equivalent of the human auditory system and hence is used as a standard scale from which MFCCs are derived.

To map a speech sample to its corresponding *Mel* scale, the speech signal in its frequency domain is multiplied by a triangular weighing function (Xu et al., 2005). A sample triangular weighing function is shown in Figure 1. The triangular weighing function is linear till 1000 Hz but exponential beyond. As the speech is a signal that changes continuously with time, a discrete short-time Fourier transform (DFT) of the input speech is used to convert it to its frequency domain and further map on to the *Mel* scale (Sahidullah & Saha, 2012).

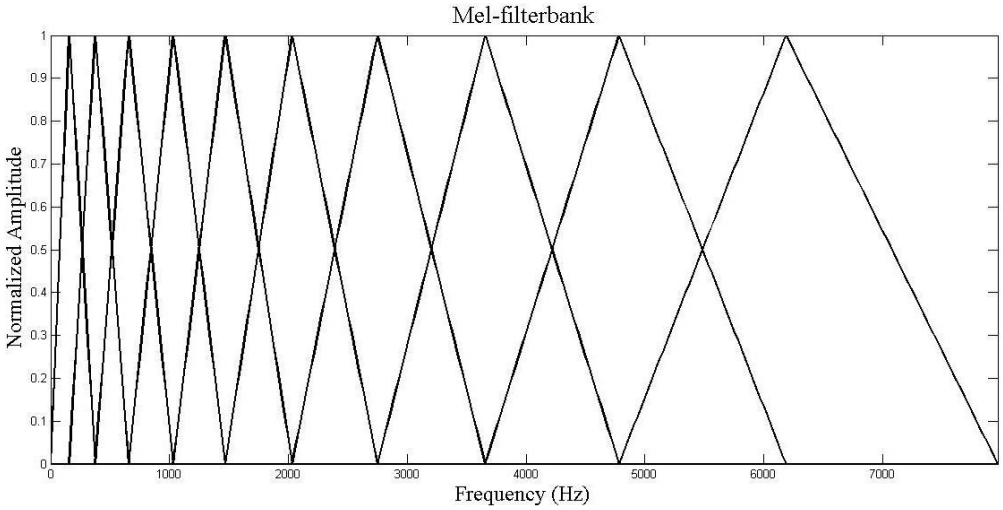

Figure 1: *Mel* filter-bank

The mel-frequency spectrum is obtained by summing the energies in each filter (Lawrence R. Rabiner & Schafer, 2007), as shown in eqn. (2.4)

$$MF[r] = \frac{1}{A_r}\sum_{k=L_r}^{U_r} |V_r[k]X(n,k)| \tag{2.4}$$

Here $V_r[k]$ is the triangular weighing function for the r^{th} filter, ranging from DFT index $L_r\ to\ U_r$, $A_r = \sum_{k=L_r}^{U_r}|V_r[k]|^2$ is a normalization factor and $X(n,k)$ is the DFT of the signal at analysis time n.

Cepstral analysis involves converting the transformed signal into its logarithmic domain before it is processed. A similar approach is used during the computation of MFCCs, by converting the spectrum output $MF[r]$ to its logarithmic value and applying a discrete cosine-transformation (DCT) as shown in eqn. (2.5).

$$MFCC[m] = \frac{1}{R}\sum_{r=1}^{R} \log(MF[r]) \cos\left[\frac{2\pi}{R}\left(r + \frac{1}{2}\right)m\right] \quad (2.5)$$

In eqn. (2.5), R is the number of filters used, m is the coefficient of MFCC and $MFCC[m]$ is typically evaluated for 13 coefficients using $R = 22$ mel-bank filters.

In order to include dynamic information within speech, delta and double-delta-MFCC features are used along with standard MFCCs. Delta-MFCCs are calculated based on the difference between successive MFCC frames and double-delta coefficients are obtained by a similar procedure on delta-coefficients. In total, this generates 13 MFCCs, 13 delta and 13 double-delta coefficients, resulting in a 39-dimensional feature vector for each speech sample.

2.4. Speech Recognition System

Figure 2 displays a basic speech recognition unit. The first step in the process to extract feature vectors from the input speech. The next step is to compare these feature vectors to acoustic models to find the best "match", which results in the recognized speech output.

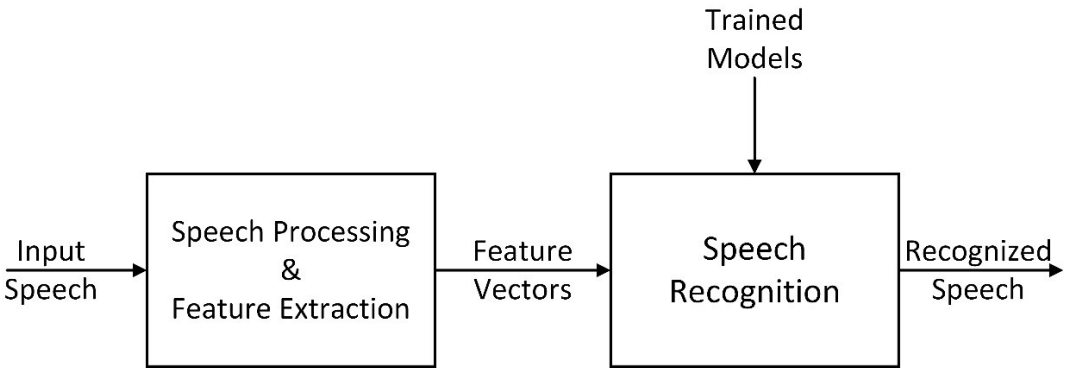

Figure 2: Basic speech recognition system

Acoustic models are statistical models to represent feature vector sequences and are trained using known speech samples. Repeated training results in pruning their parameters and these can be used for the recognition purpose. Language models are defined syntactically based on the constraints provided in the language and the grammar for recognition.

Typically, speech is recognized by recognizing smaller units that make up the speech. Words and phonemes are commonly used units while recognizing speech (Watada, 2009).

3. HIDDEN MARKOV MODELS

Markov models are stochastic models used to represent systems where the future states depends only on the present state and not on the sequence of events that preceded it. Hidden Markov Models (HMMs) are statistical Markov models with unobservable (hidden) states (Lawrence R. Rabiner & Juang, 1986).

3.1. Theory of Hidden Markov Models

HMMs are a combination of two stochastic models with one embedded in the other. The first stochastic model involves a Markov process with "hidden" states, where transitions can be made from one state to another. The states are considered hidden as they are not visible and can only be estimated based on observations made. The second model involves observations made in each state according to some probabilistic distribution that are state independent (Dymarski, 2011).

Figure 3 depicts a very basic HMM with three hidden states- z_1, z_2 and z_3 and four observations - x_1, x_2, x_3 and x_4. The solid lines represent transitions, which are either to the immediate next state or to itself. The dotted lines represent the observations made at each state.

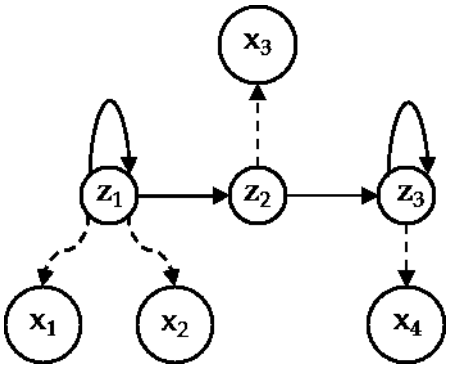

Figure 3: A basic HMM

A HMM is defined by the set of parameters $\lambda = (A, B, \pi)$. The state transition probability distribution $A = \{a_{ij}\}$, consists of the probability of transitioning from state i to state j out of a possible N states. A transition can occur from one state to a different state or to itself. The observation probability distribution $B = \{b_j(k)\}$, consists of the probability of observing an event v_k out of M possible events, while being in any state j. The initial probability distribution $\pi = \{\pi_i\}$, corresponds to the probability of being in the state i at time $t = 0$ (Lawrence R. Rabiner, 1989).

3.2. HMMs as Acoustic Models

HMMs are used in speech recognition as acoustic models from which $P(O|W)$ for eqn. (2.3) can be computed. For speech recognition, the state transitions correspond to the transitions occurring in speech during the utterance of a phoneme and the recognition task is to find the most probable state sequence given the observation. As the states are hidden, the sequence of observations are used to estimate the "most-likely" state sequence.

Acoustic models are developed for each phoneme in the language. Hence the task of maximizing $P(O|W)$ from eqn. (2.3) becomes analogous to maximizing $P(O|\lambda)$, where $\boldsymbol{\lambda}$ includes the set of all HMMs representing phonemes in the language.

The acoustic models are trained using known speech samples in order to compute the values for the transition probabilities and output probabilities.

3.3. Viterbi Algorithm

A common optimality condition to estimate the state sequence is to obtain the single best state sequence path. Mathematically, this translates to maximizing $P(Q|O,\lambda)$, where $Q = \{q_1 q_2 q_3 \dots q_T\}$, is the state sequence, $O = \{o_1 o_2 o_3 \dots o_T\}$ is the observation sequence and $\boldsymbol{\lambda}$ is the HMM model. But maximizing $P(Q|O,\lambda)$ is equivalent to maximizing $P(O|\lambda)$, which is the probability that the observation sequence can be produced by HMM λ (Lawrence R. Rabiner, 1989).

To find the best state sequence over the entire time period, we need to track the best path at each time instant t. Let $\delta_t(i)$ stand for the highest probability for a single path, at time t, ending in state i. Then,

$$\delta_t(i) = \max_{q_1 q_2 q_3 \dots q_{t-1}} P[q_1 q_2 q_3 \dots q_t = i, o_1 o_2 o_3 \dots o_t|\lambda] \tag{3.1}$$

Mathematically, it can be re-written as,

$$\delta_t(i) = \pi_{q_1} b_{q_1}(o_1).\, a_{q_1 q_2} b_{q_2}(o_2).\, a_{q_2 q_3} b_{q_3}(o_3) \dots a_{q_{(t-1)} i} b_i(o_t) \tag{3.2}$$

In eqn. (3.2), π_j is the probability of being in an initial state j, a_{ij} is the transition probability from state i to j and $b_j(o_{t+1})$ is the probability making an observation while being in state j at time $(t+1)$.

In eqn. (3.2), the most-likely state sequence is assumed to be $q_1 q_2 q_3 \dots q_T$. In practice, this is obtained by considering all possible state transitions from one state in a time frame to the next time frame, generating the corresponding observation and picking out the one with the maximum probability. Hence the current value of $\delta_t(j)$ is dependent on its previous value $\delta_{t-1}(j)$. Dynamic programming using mathematical induction can be used for the computation of the optimal state sequence and $\delta_{t+1}(j)$ can be expressed as given in eqn. (3.3), where $\delta_t(i)$ is the most probable path till time t ending in state i.

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] . b_j(o_{t+1}) \tag{3.3}$$

Eqn. (3.3) can be solved using the Viterbi algorithm (Forney, 1973), which is a dynamic programming technique, as follows:

1. Initialization

$$\delta_1(i) = \pi_i . b_i(o_1) \quad , t = 1, 1 \le i \le N \tag{3.4}$$

2. Recursion

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i) a_{ij}\right] . b_j(o_t) \quad , 2 \le t \le T, \; 1 \le j \le N \tag{3.5}$$

3. Termination

$$P(O|\lambda) = \max_{1 \le i \le N} [\delta_T(i)] \tag{3.6}$$

In eqns. (3.4) - (3.6), N stands for the number of states in the HMM model and T is the number of time frames of the observation sequence.

3.4. HMMs in Speech Recognition

HMMs are used for modeling each basic recognition unit. In this proposal, HMMs are used for each monophones. Monophones are synonymous to isolated individual phonemes. For each HMM, $P(O|\lambda)$ is computed using eqns. (3.4)-(3.6). To recognize the input phoneme, all the likelihood probability values, $P(O|\lambda)$ are compared and the HMM which yields the maximum value of the probability signifies the recognized phoneme.

Such a monophone-HMM based recognition system is represented pictorially in Figure 4 below.

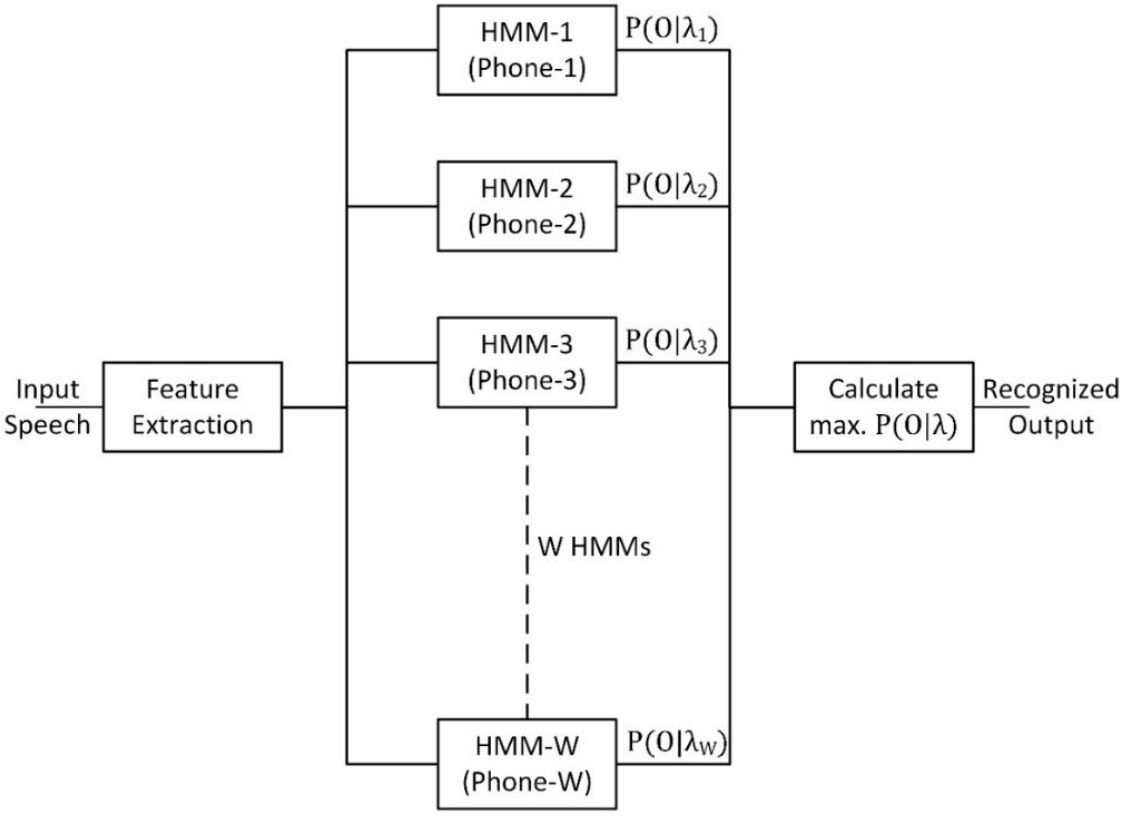

Figure 4: A monophone-HMM based recognition system

4. HARDWARE IMPLEMENTATION OF SPEECH RECOGNITION SYSTEM

A hardware design for a speech recognition system must be capable of processing the input speech, extract the probability of observing the state sequence for the input across all the trained HMM models and identify the best match to recognize the input speech. Assuming that the input speech is processed and features are extracted and available, the design must implement the Viterbi algorithm to extract the best possible states sequence and compare the likelihood probabilities generated by all the HMMs.

4.1. Log-Viterbi Algorithm

The Viterbi algorithm described in Section-3.3 is a standard way of computing the most-likely state sequence for an input speech. However, for implementing the same on hardware, the equations can be simplified by taking logarithms (Bok-Gue, Koon-Shik, & Jun-dong, 2002). This yields the log-Viterbi algorithm as follows:

1. Initialization

$$\delta_1(i) = \log \pi_i + \log b_i(o_1) \qquad t = 1, 1 \le i \le N \quad (4.1)$$

2. Recursion

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) + a_{ij}] + \log b_j(o_t) \qquad 2 \le t \le T, \quad 1 \le j \le N \quad (4.2)$$

3. Termination

$$P(O|\lambda) = \max_{1 \le i \le N} [\delta_T(i)] \tag{4.3}$$

The symbols used in the equations denote the same as meant in Section-3.3. By converting the Viterbi algorithm to the logarithmic form, multiplications become additions, which are easier to implement in hardware.

4.2. HMM Parameters for Hardware Implementation

4.2.1. Sparse Transition Matrix

Speech is a signal that varies with time and as it does not go back in time, a left-right HMM can be used to model speech. Every utterance is preceded by the previous one and hence the transitions are either to the existing state or the immediate successor. Such a left-to-right HMM is shown in Figure 5. The HMM in the figure has 3 states and each state either transitions onto itself or to its successor.

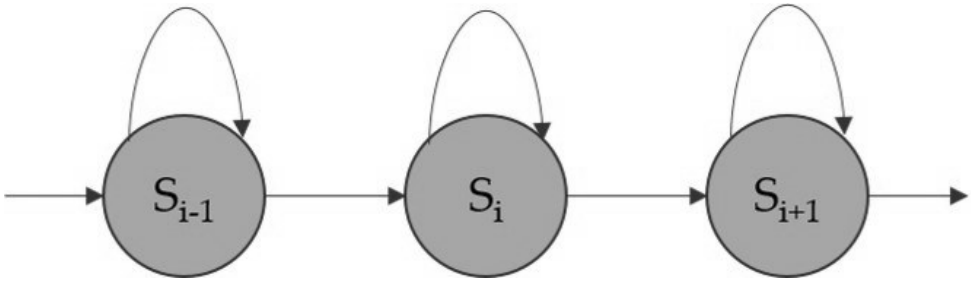

Figure 5: A left-to-right HMM

With such a HMM, the transition probability matrix becomes a sparse matrix of the form,

$$A = \begin{cases} a_{ij}, for\ j = i\ or\ i-1 \\ 0, for\ all\ other\ \ j \end{cases} \qquad 1 \le i, j \le N \qquad (4.4)$$

Using the sparse transition probability matrix from eqn. (4.4), eqn. (4.2) can be modified as

$$\delta_t(j) = \max_{i=j-1,j} [\delta_{t-1}(i) + a_{ij}] + \log b_j(o_t) \qquad (4.5)$$

4.2.2. Output Probability Distribution

The HMMs are called a continuous-HMM if the output probability distribution is continuous in nature. As speech is continuous, it is typical to assume a Gaussian distribution for the output probability and the output probability can be written as

$$b_j(o_t) = \mathcal{N}(o_t, \mu_j, \Sigma_j) \qquad (4.6)$$

$$b_j(o_t) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_j|}} \exp\left(-\frac{1}{2} (o_t - \mu_j)' {\Sigma_j}^{-1} (o_t - \mu_j) \right) \qquad (4.7)$$

Here o_t is the D-dimensional feature vector extracted from the input speech and μ_j and Σ_j are D-dimensional vectors of mean and co-variance of the output probabilities for state j. For eqn. (4.1) and (4.5), we deal with log output probabilities. Hence, by taking logarithm of eqn. (4.7), we have

$$\log b_j(o_t) = \log \frac{1}{\sqrt{(2\pi)^D |\Sigma_j|}} - \frac{1}{2}(o_t - \mu_j)' \Sigma_j^{-1} (o_t - \mu_j) \tag{4.8}$$

Additionally, the output probabilities are typically modelled using Gaussians with diagonal covariance matrix (Gales, 2000). Hence eqn. (4.8) is simplified as,

$$\log b_j(o_t) = \log \frac{1}{\sqrt{(2\pi)^D \prod_{d=1}^{D} \Sigma_{jd}}} + \sum_{d=1}^{D} -\frac{1}{2\Sigma_{jd}} (o_{td} - \mu_{jd})^2 \tag{4.9}$$

In eqn. (4.9), Σ_{jd} and μ_{jd} denote the covariance and mean for state j and dimension index d and o_{td} is the observation at time t for dimension index d.

$$\log b_j(o_t) = \omega_j + \sum_{d=1}^{D} \sigma_{jd} (o_{td} - \mu_{jd})^2 \tag{4.10}$$

In eqn. (4.10),

$$\omega_j = \log \frac{1}{\sqrt{(2\pi)^D \prod_{d=1}^{D} \Sigma_{jd}}} \tag{4.11}$$

and variance,

$$\sigma_{jd} = -\frac{1}{2\Sigma_{jd}} \tag{4.12}$$

The values of ω_j, σ_{jd} and μ_{jd} in eqn. (4.10) can be computed during the training phase and made available for use during the recognition phase. Hence for the hardware

implementation, the trained models can be stored in memory and accessed during recognition. Using eqns. (4.1), (4.3), (4.5) and (4.9), the computation of $P(O|\lambda)$ has been broken down into simpler mathematical steps, which can be implemented in hardware with much ease than could have been the case with the standard Viterbi algorithm.

4.3. Prior Work

In 2001, a speech recognition system was implemented on an FPGA by (Rodriguez-Andina, Fagundes, & Junior, 2001). The work was based on a hardware-software co-design approach. It briefly touches upon a different way of scoring the most-likely state sequence in a way which is slightly different from the standard Viterbi algorithm, using dedicated logic blocks. The performance of their design is based on a comparison of the time taken by their FPGA-based design to one that implements a classic Viterbi algorithm. The system used word-level HMMs and performance was reported for applications with 2, 3 & 4 words separately.

(Melnikoff, Quigley, & Russell, 2002) implemented a simple continuous speech recognition system in 2002 on a Xilinx Virtex FPGA, using monophone HMM models, with 3 states each. They demonstrated the superiority in speed for the hardware design over a software model, while maintaining similar accuracy results. Their implementation uses the standard Viterbi algorithm.

A scalable word-HMM based complete speech recognition system was designed in 2006 by (Yoshizawa, Wada, Hayasaka, & Miyanaga, 2006). Their implementation is based on the log-Viterbi algorithm and a continuous HMM implementation. Their system

does speech processing, robust noise reduction, speech recognition and controls data-transfer resulting in a master-slave-like model. The hardware design was done using CMOS 0.18µm technology and performance was measured by comparing the power/energy dissipation with that of the corresponding software implementation. The results were favorable for the hardware implementation.

A flowchart based on their implementation is shown in Figure 6. A similar structure is used in this thesis proposal as well. The flowchart depicts the process flow in computing the likelihood of the current input sequence for a HMM model $\boldsymbol{\lambda}$, in terms of $P(O|\lambda)$.

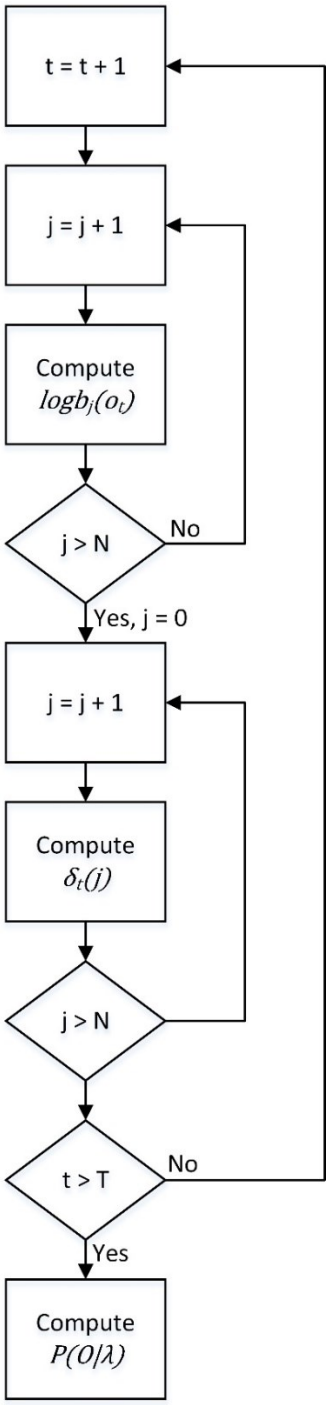

Figure 6: Flowchart for computing $P(O|\lambda)$ within a single HMM (© 2006 IEEE (Yoshizawa et al., 2006) – adapted with permission from the IEEE)

5. PROPOSED IMPLEMENTATION

The hardware was designed using Verilog and compiled using a commercial 90nm CMOS standard cell library. The proposed hardware design implements the logic for the computation of log-output probability, the log-Viterbi algorithm and the likelihood probability comparison. This implementation was for an application that used 28 monophones from the English language – *aa*, *ae*, *ah*, *ao*, *ay*, *d*, *eh*, *ey*, *f*, *ih*, *iy*, *jh*, *k*, *l*, *n*, *ng*, *ow*, *p*, *r*, *s*, *t*, *th*, *uh*, *uw*, *v*, *w*, *y* and *z* and hence consisted of 28 HMMs, with each HMM having 3 states. Feature vectors used are of 39 dimensions, using 13 MFCC vectors and their delta and double-delta coefficients.

A basic structure of the overall circuit, similar to the block diagram in Figure 4, is shown in Figure 7. Each of the 28 HMM modules are similar in design, but trained for the particular monophone that they represent.

Training was done using the HTK software (Young, 1993) using a recipe similar to the one detailed in the HTK Book (Young et al., 1997). The trained model parameters were used in the design. Hence the mean and the variance of the output probabilities, the transition probabilities and the initial state probabilities are all available and stored in memory (not shown in the figure) for each monophone. With 3 states for each HMM, the memory requirement is minimal.

There are 3 major logical components in the design:

1. Computation of $\log b_j(o_t)$ for each state and time frame, within each HMM.
2. Computation of $\delta_t(j)$ for each state and time frame and obtaining $P(O|\lambda)$, the maximum value across the 3 states at time $t = T$, for each HMM.
3. Computation of the maximum $P(O|\lambda)$ obtained from all 28 HMMs, with the HMM yielding the maximum value corresponding to the recognized phoneme.

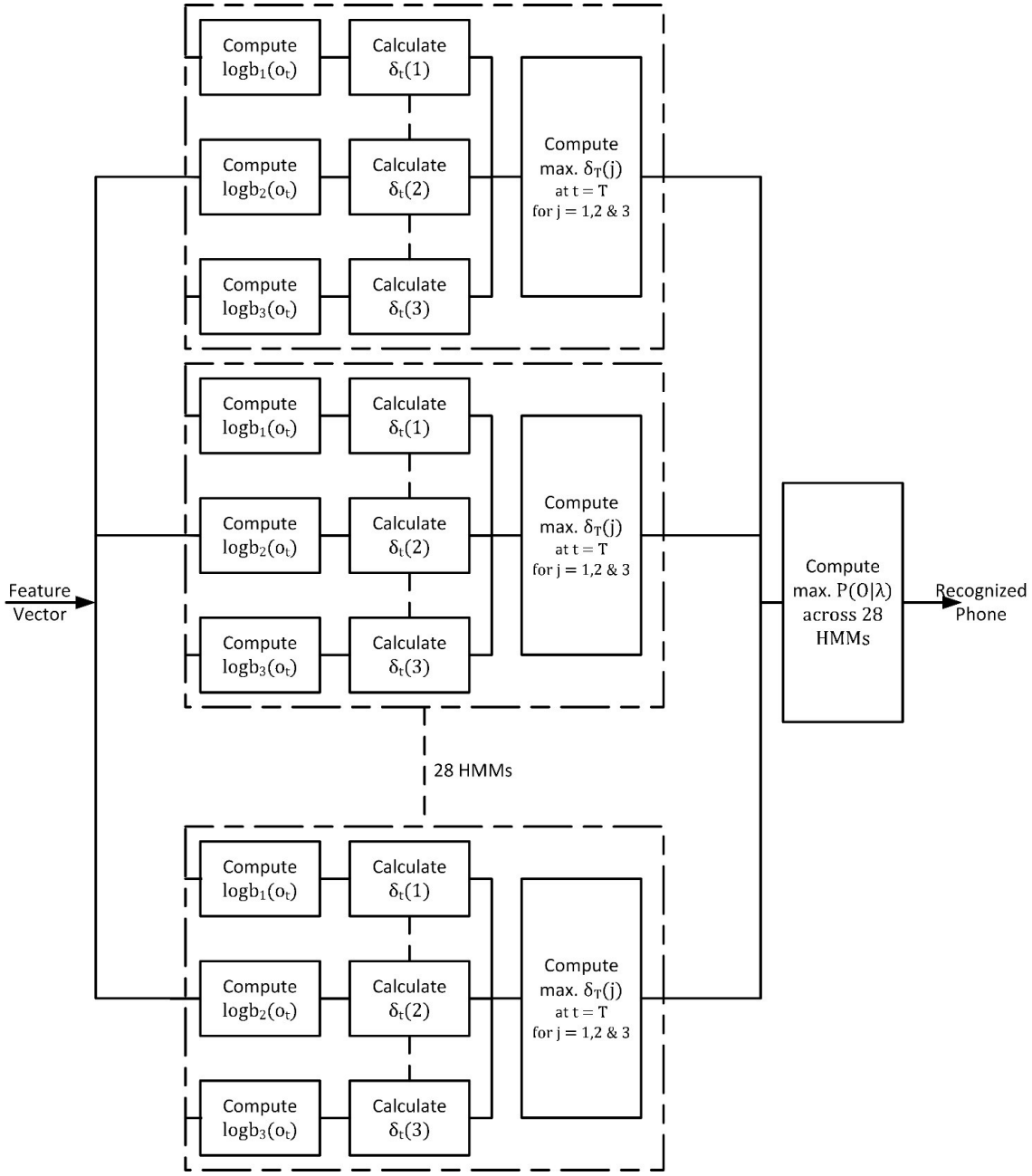

Figure 7: Basic structure of the proposed system

5.1. Output Probability Computation

As seen in eqn. (4.10), the computation of $\log b_j(o_t)$ involves one subtraction and two multiplications iterated 39 times. The accumulated sum undergoes another addition before $\log b_j(o_t)$ is generated for each state in a time period. As the subtraction can be implemented using an adder with the second operand reversed in sign, the output probability computation unit is implemented using 2 adders and 2 multipliers and a register for accumulating the sum.

A parallel implementation of the same can be designed, but this would require 39 instances of the adder and the 2 multiplier units and a 39-operand adder for their cumulative sum. As the computation of $\log b_j(o_t)$ is a critical step and is used for each state and time frame, the sequential method of implementation is preferred over the parallel, to conserve area.

The input to this unit are the feature vectors. The feature vectors' mean and variance are a combination of signed and unsigned floating point numbers covering a wide range. Hence, for a uniform implementation and to prevent loss of precision, they are scaled to a 16-bit integer value. The scaling is done by multiplying all three parameters by a fixed constant K. Hence, eqn. (4.10) is modified as

$$\log b_j(o_t) = \omega_j + \frac{1}{K^3}\sum_{d=1}^{D} \sigma'_{jd}\left(o'_{td} - \mu'_{jd}\right)^2 \tag{5.1}$$

In eqn. (5.1), $\sigma'_{jd} = K.\sigma_{jd}, o'_{td} = K.o_{td}$ and $\mu'_{jd} = K.\mu_{jd}$. For this implementation, the value of K is chosen as 100. A consequence of scaling these parameters is that to maintain equality, all other terms need to be scaled accordingly. Hence, the remaining terms in the computation of $\log b_j(o_t)$ and in the log-Viterbi algorithm are all scaled by K^3.

Figure 8 provides a schematic description of the circuit used for computing $\log b_j(o_t)$. The bit-width of operands for each operation is also shown.

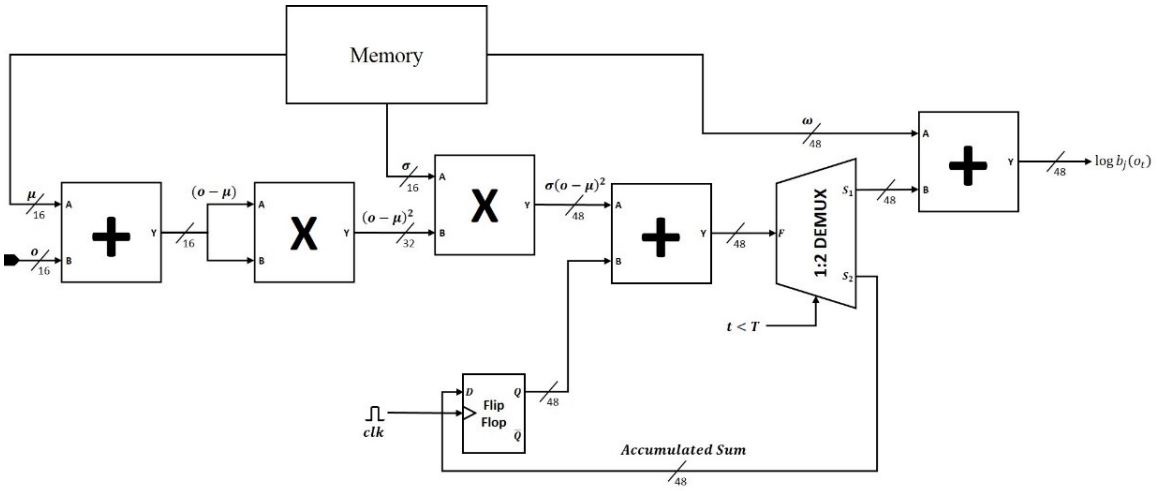

Figure 8: Output probability computation

The above design is instantiated thrice within each HMM to compute $\log b_j(o_t)$ for each state. The $\log b_j(o_t)$ computed is used for the calculation of $\delta_t(j)$.

5.2. Log-Viterbi Algorithm Implementation

The implementation of the log-Viterbi algorithm is split into two parts – firstly the computation of $\delta_t(j)$ and secondly, using that to evaluate $P(O|\lambda)$. The computation of $\delta_t(j)$ is done differently based on the time-frame. For $t = 0$, it is a direct addition of $\log \pi_i$, which is stored in the memory, with the value of $\log\ b_j(o_t)$, which was computed previously. For all other values of t, $\delta_t(j)$ depends on $\delta_{t-1}(j)$, transition probability a_{ij} and $\log\ b_j(o_t)$.

Figure 9 displays a schematic representation of the $\delta_t(j)$ computation. Since the generation of $\log\ b_j(o_t)$, all computations are on operands which are 48-bit wide. As mentioned, each $\delta_t(j)$ utilizes the $\delta_{t-1}(j)$ computed in the previous time-frame (shown as dotted lines in the figure). It is implemented by delaying the current $\delta_t(j)$ through a D-Flip-Flop. The design generates $\delta_t(j)$ for the initialization and the recursion steps of log-Viterbi algorithm and the final value is multiplexed based on the time-frame.

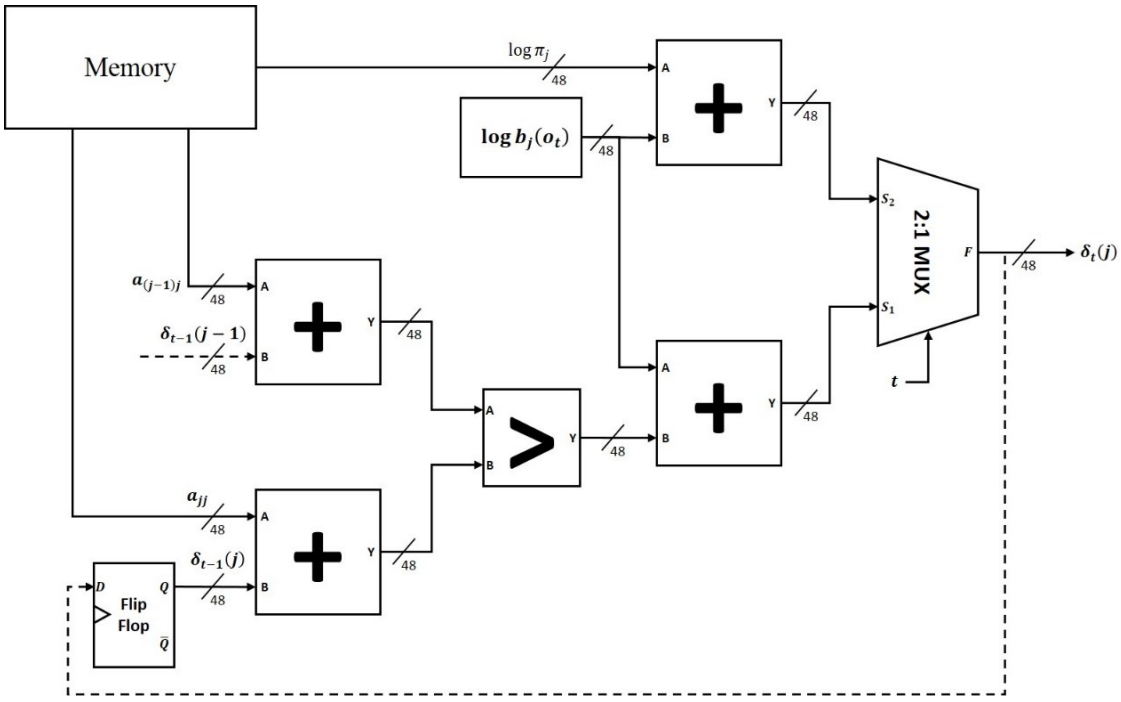

Figure 9: Implementation of log-Viterbi initialization & recursion

This design is also instantiated 3 times, one for each state within a HMM. The output of the 3 units are passed onto to a cascaded comparator design to obtain $P(O|\lambda)$ at time $t = T$, as shown in Figure 10.

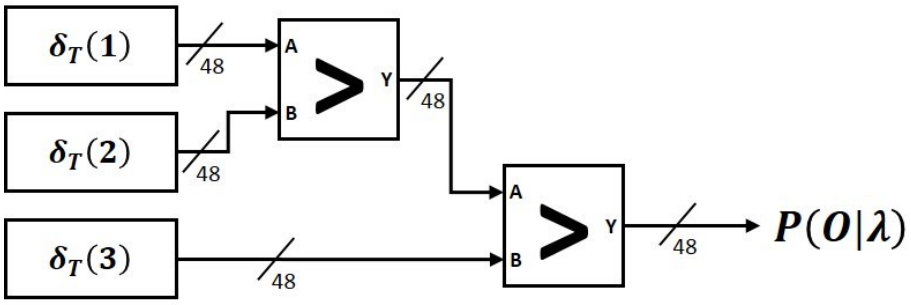

Figure 10: Termination of Viterbi algorithm

The schematics given in Figure 8, Figure 9 and Figure 10 are connected together to implement a single HMM on hardware.

5.3. Output Probability Comparison

This is the final step in the recognition of a phoneme. Each HMM is constructed as mentioned above, and 28 such modules are instantiated, one for each phoneme. Each HMM yields a likelihood probability, $P(O|\lambda_v)$, where λ_v corresponds to the v^{th} HMM and $1 \leq v \leq 28$. The recognized phoneme is the one with maximum $P(O|\lambda_v)$.

Hence the final step in the implementation consists of a cascaded comparator structure as shown in Figure 11.

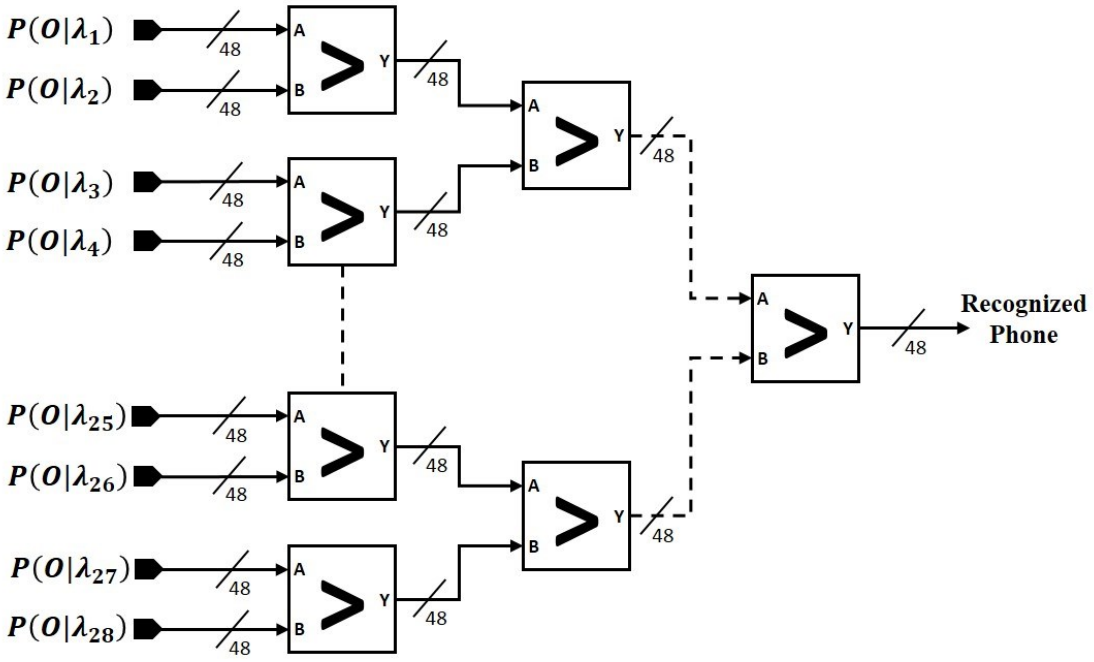

Figure 11: Final probability comparison

Each comparator has two 48-bit wide inputs and outputs the greater of the two inputs. There are 28 comparators at the first level and are cascaded down each level till the final comparison is made. The index of the final output corresponds to the index of the recognized phoneme.

5.4. Modifications to the log-Viterbi Algorithm for Implementation

Implementing logarithmic computations on hardware require the additional overhead of validating the input to the logarithmic function to be non-zero as $\log 0 = \infty$. To prevent this scenario from occurring, only absolute values of $\log b_j(o_t)$ were used. Any occurrence of $\log 0$ was assigned the highest possible value in the fixed point implementation.

By using absolute values in the implementation, the log-Viterbi algorithm has to be modified by changing all max functions in the actual algorithm to min functions.

6. CUSTOM OPTIMIZATION OF THE PROPOSED HARDWARE DESIGN

The design described in Section-5 is a direct translation of the log-Viterbi algorithm into hardware. A closer examination reveals that the system can be optimized further with respect to area and power.

6.1. Optimizing the log-Viterbi Initialization Step

As seen in Section-5.2, the computation of $\delta_t(j)$ is done twice – one for the initialization step and for the recursion step. The value of $\log b_j(o_t)$ is required for both the steps. But the initialization step consists of only the addition of the computed $\log b_j(o_t)$ value with the stored $\log \pi_j$ values. Hence by merging this addition in the computation of $\log b_j(o_t)$, three 48-bit additions can be avoided (one for each state in the initialization step).

Furthermore, since we are considering a left-to-right HMM, all computations start with the first state. Hence the probability of the HMM being in state-1 at time $t = 0$, is 1 and that of being in any of the other states is 0.

Eqn. (4.1) can be expanded using eqn. (4.10) as

$$\delta_1(j) = \log \pi_j + \omega_j + \sum_{d=1}^{D} \sigma_{jd}\left(o_{td} - \mu_{jd}\right)^2 \tag{6.1}$$

$$\log \pi_j = \begin{cases} 0, & for\ j = 1 \\ 0x7FFF, & for\ j \neq 1 \end{cases} \tag{6.2}$$

Hence combining (6.1) and (6.2),

$$\delta_1(j) = \omega_j' + \sum_{d=1}^{D} \sigma_{jd}\left(o_{td} - \mu_{jd}\right)^2 \tag{6.3}$$

, where

$$\omega_j' = \begin{cases} \omega_j, & for\ j = 1 \\ \omega_j + 0x7FFF, & for\ j \neq 1 \end{cases} \tag{6.4}$$

So for $t = 0$, the value of ω_j changes to ω_j' and these can be stored in the memory as well and used for computation.

With this modification, the initialization step can be overlapped with the computation of output probabilities.

$$\delta_t(j) = \begin{cases} \log b_j(o_t), & for\ t = 0 \\ \max\limits_{1 \le i \le N}\left[\delta_{t-1}(i) + a_{ij}\right] + \log b_j(o_t), & for\ t \neq 0 \end{cases} \tag{6.5}$$

The schematic in Figure 9 is modified as per eqn. (6.5) and shown in Figure 12.

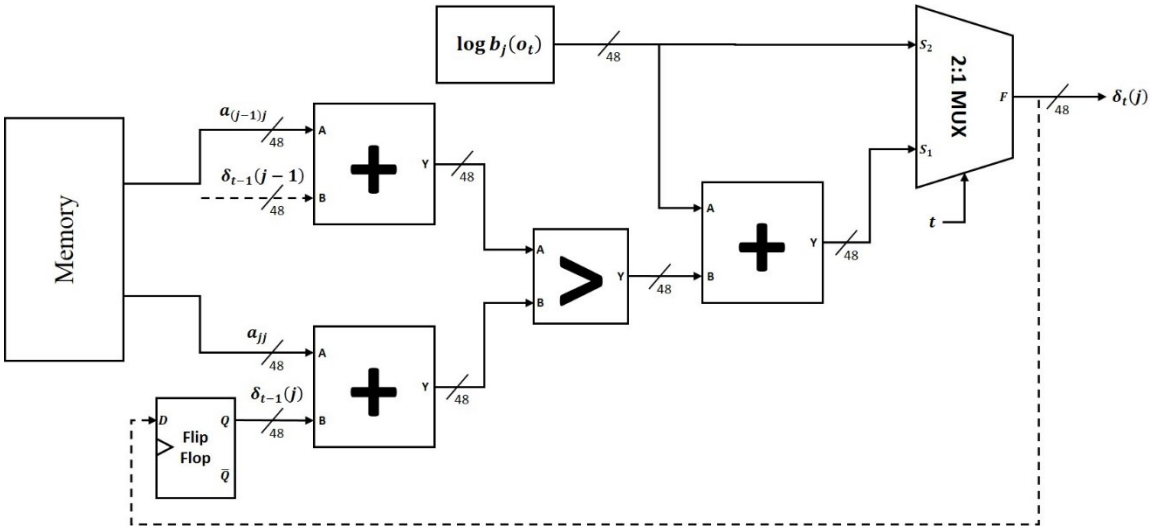

Figure 12: Optimized log-Viterbi algorithm

6.2. Optimizing the Output Probability Computation

As seen in Section-5.1, the computation of $\log b_j(o_t)$ involves 39 computations sequentially and then the accumulated sum is added on to ω_j. At this instant, the addition is between two 48-bit values. A closer observation of the data reveals that the value of the accumulated sum is much greater than the value ω_j. The scaling discussed in Section – 5.1 further magnifies this difference to the extent that the higher-order bits of ω_j do not carry much information at all.

Hence during the computation of $\log b_j(o_t)$, only the 24 lower order bits are added to the accumulated sum. The 24 more significant bits of $\log b_j(o_t)$ are connected directly to those of the accumulated sum.

In terms of eqn. (4.10), it becomes,

$$\log b_j(o_t)\,[47:24] = \left\{\sum_{d=1}^{D} \sigma_{jd}\left(o_{td} - \mu_{jd}\right)^2\right\}[47:24] \tag{6.6}$$

$$\log b_j(o_t)\,[23:0] = \omega_j[23:0] + \left\{\sum_{d=1}^{D} \sigma_{jd}\left(o_{td} - \mu_{jd}\right)^2\right\}[23:0] \tag{6.7}$$

where the numbers within square brackets signify the bits.

With this modification in the computation of output probability, three 48-bit additions are converted to 24-bit additions. This is significant considering that the output probability computation is the primary and critical step in the entire process.

6.3. Optimizing the log-Viterbi Recursion Step

The recursion step in the log-Viterbi algorithm involves two 48-bit additions between previous $\delta_t(j)$ values and transition probabilities, for each state. In a manner similar to the one discussed in Section-6.2, the operands in these additions vary extremely in magnitude.

The transition probability value a_{ij} does not carry much information in its higher order bits. Hence during the addition of $\delta_{t-1}(j)$ and a_{ij}, the more significant 24 bits of the sum are connected directly to those of $\delta_{t-1}(j)$ and actual addition is carried out between the lower 24 bits of $\delta_{t-1}(j)$ and a_{ij} to obtain the corresponding bits of the sum.

The summation within the max function in eqn. (4.2) can be re-written to include this modification as

$$temp\_sum[47:24] = \delta_{t-1}(i)[47:24] \tag{6.8}$$

$$temp\_sum[23:0] = \delta_{t-1}(i)[23:0] + a_{ij}[23:0] \tag{6.9}$$

where $temp\_sum$ holds the temporary sum of both the operands and the numbers within the square brackets indicate bit positions.

6.4. Optimizing Final Comparator Logic

Section-5.3 described the cascaded comparator logic for obtaining the most-likely phoneme from all the probability values computed. Each HMM outputs a 48-bit value, which is fed into the comparator. It has been observed that the data being compared is not too relatively close and that the comparisons can be made just by observing the more-significant bits.

Hence the comparator logic has been designed for 24-bit inputs. Only the higher order 24-bits from each of the HMMs are used for the comparison. This reduces the complexity of the required comparisons significantly.

7. APPROXIMATE COMPUTING

The optimization techniques discussed in Section-6 involve modifying actual computations in order to save computational area and time. Such techniques tend to "approximate" actual computation but not affect the final output significantly. Though the methods discussed so far are simple in terms of complexity and implementation, it is in line with a contemporary design methodology to face the challenge in managing chip power consumption. This methodology is categorized as approximate/soft computing and is targeted at tackling energy efficiency challenges (Chippa, Mohapatra, Raghunathan, Roy, & Chakradhar, 2010).

Such techniques are especially useful for systems were the inherent algorithms are error-resilient. In the proposed design, the recognition is based on comparisons on likelihood probabilities generated by all the HMMs. In other words, the relative probability values are more important than the absolute value of each. This property generates a scope for the use of approximate computing techniques.

The usage approximate computing with respect to the proposed design refers to introducing deliberate errors in computation to enhance the speed without significantly affecting accuracy.

(Kim et al., 2013) discusses an error-resilient adder design applied for neuromorphic VLSI systems. The speed determining step in an adder is the carry propagation from LSB to MSB. But the work proposes a carry-prediction design wherein

the carry is predicted for sub-blocks of the operands in a parallel fashion. This results in a significantly improved hardware performance albeit with a slight compromise on accuracy.

In this thesis too, a similar approach is undertaken. As stressed previously, the computation of $\log b_j(o_t)$ is critical for the entire algorithm. The steps involved in this computation are repeated 39 times for each of the 3 states during all the time frames. The first step in computation of $\log b_j(o_t)$ is the summation of o_{td} and $-\mu_{jd}$. This is a 16-bit addition, normally implemented using a Carry-Look-Ahead adder.

The implementation is shown in Figure 13. The 16-bit addition is divided into sub-blocks of 4 and 2-bits. The 8 most significant bits are divided into two blocks of 4-bits each and the lower 8-bits are divided into four 2-bit blocks, resulting in six blocks totally. Each block of input from both operands is connected to a Carry-Generator and an adder.

'Generate', 'propagate' (both not shown in figure) and carry-out signals are generated from each of the six Carry-Generator blocks as follows,

$$g_i = a_i \circ b_i \tag{7.1}$$

$$p_i = a_i \oplus b_i \tag{7.2}$$

$$C_{out} = g_{i-1} + g_{i-2}p_{i-1} + \cdots + g_0 \prod_{j=1}^{i-1} p_j \tag{7.3}$$

where g_i and p_i are the 'generate' and 'propagate' signals for the i^{th} bit in a block, and C_{out} is the carry-out signal for that block.

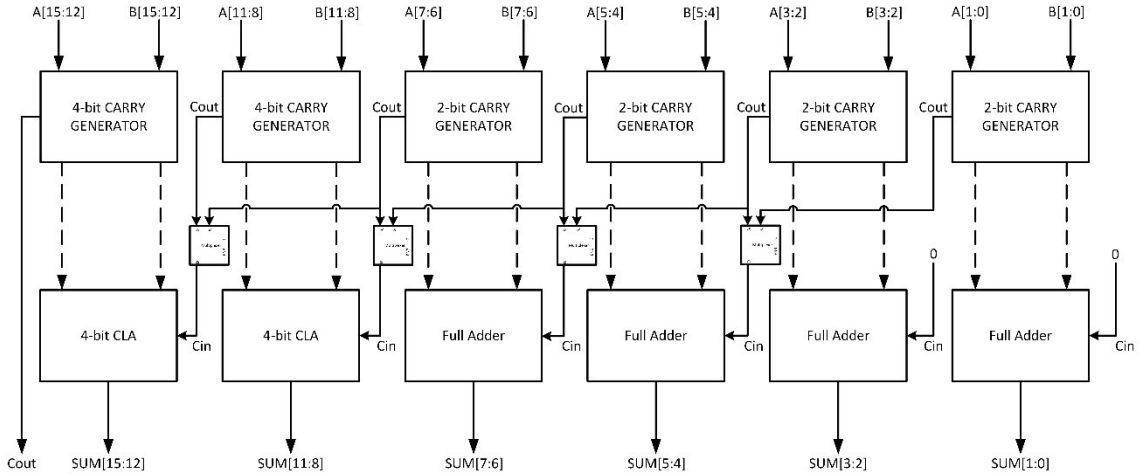

Figure 13: 16-bit approximate adder design

Instead of each Adder unit depending on carry to be propagated from all its preceding units, a carry-in is predicted for each Adder unit using the carry-out signal generated by the previous two Carry-Generator units. A carry-out signal from $(k-1)^{th}$ block is propagated to $(k+1)^{th}$ block if all the propagate signals in the k^{th} block are 1. Hence the carry-out signals from block k and $(k-1)$ are multiplexed using the propagate signal from the k^{th} block.

$$C_{in}^{k+1} = \overline{P^k} C_{out}^{k} + P^k C_{out}^{k-1} \tag{7.4}$$

, where $P^k = \prod_{j=0}^{i-1} p_j$ for a block of i bits.

The Adder unit operates on the inputs and the carry-in and produces the sum. For the 4-bits adders, a Carry-Look-Ahead adder is used. For 2-bit, a full adder is used. The carry-in for the first 2 adders are set to 0.

After the output probability is computed, it is used in the computation of $\delta_t(j)$. As discussed in Section-6.3, this step also involves a 24-bit addition of previous $\delta_t(j)$ value and the transition probability. A similar approximate adder is used for this addition as well. In place of a 16-bit adder, now a 24-bit adder is used with six blocks, each with 4-bit operands. The full adders get replaced with 4-bit Carry-Look-Ahead adders. The implementation is shown in Figure 14.

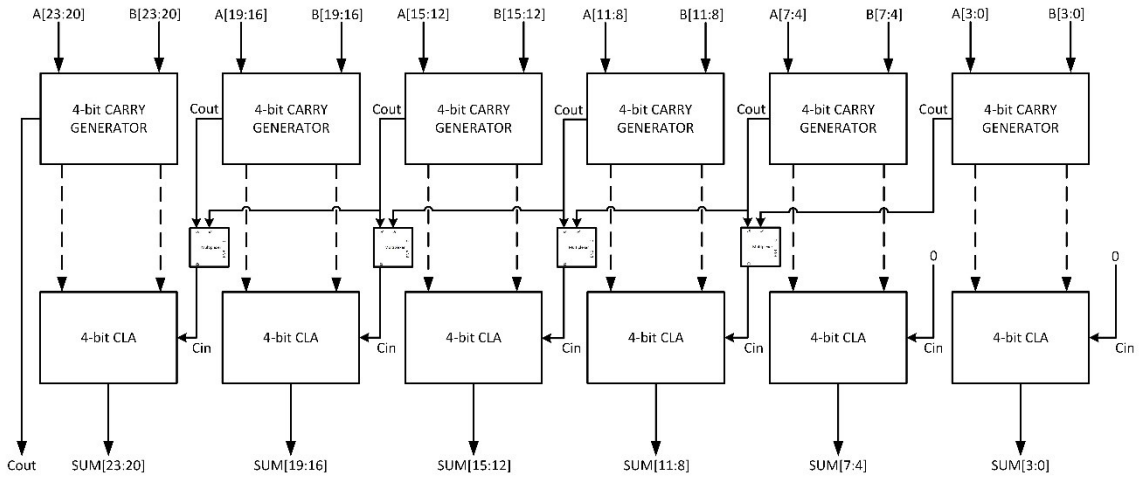

Figure 14: 24-bit approximate adder design

8. PERFORMANCE ANALYSIS AND RESULTS

A basic model of a recognizer was built and its performance analyzed. The results were analyzed separately for the actual full-precision design, with the custom optimizations suggested and incorporating approximate computing techniques and are presented below.

The entire hardware design was coded in Verilog. It was compiled in Synopsys Design Compiler (DC) (Bacon et al., 2004) using a commercial 90nm CMOS standard cell library designed for a typical part at 25C operating temperature and a voltage level of 1.2V.

The Design Compiler takes in the Verilog hardware description and produces a netlist and estimates for area, timing and power consumption for the clock frequency input. The area report presents the total area of the design which is directly related to the total area of all the cells in the design. The timing report contains the slack for the input clock frequency. Based on the slack, the frequency can be adjusted and the maximum frequency of operation is the frequency that produces zero slack. The power reports provide estimates on dynamic power and leakage power.

For all the implementations, an adder which performed actual addition was designed as a Carry-Look-Ahead Adder. For the 2 multiplication tasks, DC's standard multiplier was used. All comparisons were made using custom-built CLA adders, whose second input is the negative of the second operand and the smaller (or greater) number is

selected based on the sign of the sum. These were the three combinational arithmetic and logic units used in the design.

Hidden Markov Toolkit (HTK) (Young, 1993) was used to train the acoustic models. HTK is a toolkit for working with HMMs mainly for speech recognition. Model parameters obtained after the training process were used in the design. This design does not discuss memory storage for these parameters as currently the requirement is not significant.

Testing samples were recorded and its MFCC vectors extracted using MATLAB scripts in a way similar to the MFCC extraction done using the HTK tool. These vectors act as input to the system.

The initial raw implementation will henceforth be referred to as the 'Actual Implementation', the custom-optimized version will be referred to as 'Approximate Technique-1' and the implementation using approximate adders on top of the custom optimization will be referred to as 'Approximate Technique-2'.

8.1. Performance Evaluation

The system shows a recognition accuracy of 46% with both the Actual Implementation and the Approximate Technique-1. Hence, the custom optimization does not affect the accuracy but while using Approximate Technique-2, the accuracy slightly lowers to 43%. The overall low accuracy is attributed to using monophone HMMs and the use of single Gaussians for modeling output probability distribution. Commercial speech

recognizing software use tri-phone HMM models and Gaussian mixture models to incorporate contextual information and can achieve high accuracy. A tri-phone-based recognizer built using HTK with similar training showed an accuracy of 80%. Reported accuracy using monophones tend to be around 56.8% (Melnikoff et al., 2002).

8.2. Hardware Evaluation

Hardware design evaluation for the entire system is given in Table 1. The table reports area, dynamic power, leakage power, execution time, energy dissipated (which is the product of the total power and the execution time) and an Energy-Area-Product (EAP) to portray the combined effect of area, power and execution time.

The system is clocked at 95 MHz, which is the highest common frequency that all the three designs can be run at. As all three designs are running at the same frequency, their execution times are the same.

Table 1: Hardware evaluation of the entire system at 95 MHz

	Area (mm^2)	Dynamic Power (mW)	Leakage Power (mW)	Execution Time (ns)	Energy Dissipated (nJ)	Energy-Area Product (EAP) (mm^2-nJ)
Actual Implementation	3.2442	36.7378	4.4599	425	17.5090	56.8035
Approximate Technique-1	2.8729	37.0367	3.9147	425	17.4043	50.0011
Approximate Technique-2	2.8930	37.1014	3.9269	425	17.4370	50.4448

Both the Approximate Techniques show a decrease in the area of the design compared to the actual implementation. The reduction is almost 12%. Dynamic power is

almost similar across all three designs as they are running at the same frequency and operating at the same voltage level. But leakage power shows a reduction of almost 13% in both the approximate designs compared to the Actual Implementation. As the execution time is same, the energy value is dependent on the total power alone and is similar across all the three designs as the total power does not differ much. However, the Energy-Area-Product (EAP) shows a significant decrease of almost 12% for both the approximate designs. Figure 15 displays a comparison of the normalized area, power, energy, and EAP numbers across the 3 designs.

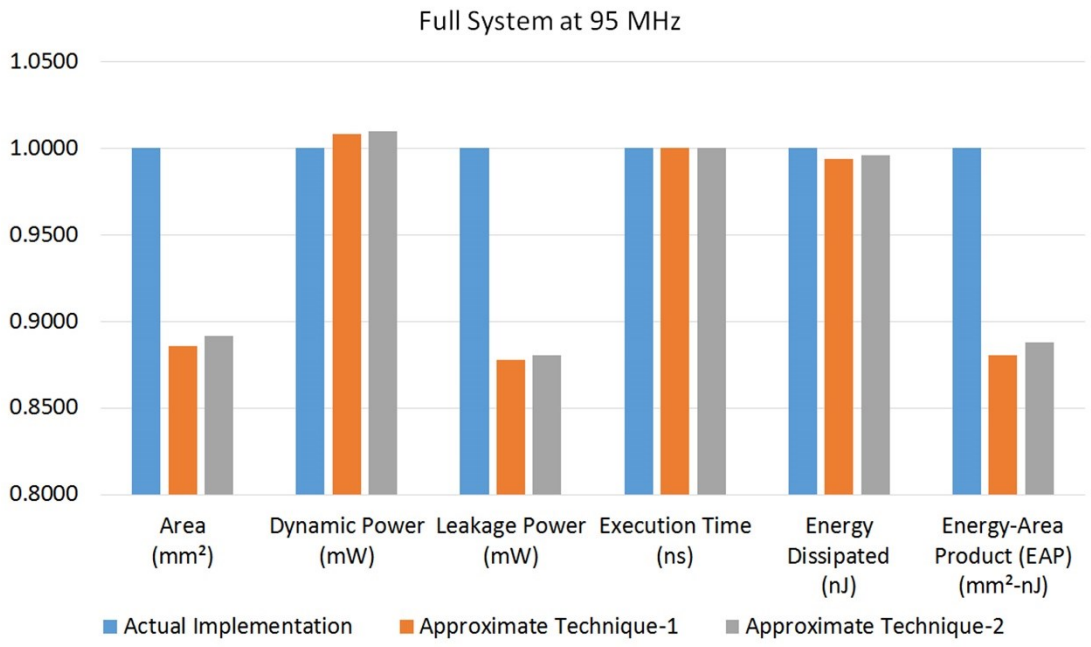

Figure 15: Hardware evaluation of the full system at 95 MHz

A single HMM can be considered as the basic building block of the entire system. As different applications require different number of phonemes, such systems would require different number of HMMs. Hence, similar performance numbers are reported for a single HMM and tabulated in Table 2.

Table 2: Hardware evaluation of a single HMM at 95 MHz

	Area (mm^2)	Dynamic Power (mW)	Leakage Power (mW)	Execution Time (ns)	Energy Dissipated (nJ)	Energy-Area Product (EAP) (mm^2-pJ)
Actual Implementation	0.1080	1.2577	151.3901	425	598.8633	64.6966
Approximate Technique-1	0.0986	1.2674	135.8360	425	596.3753	58.8118
Approximate Technique-2	0.0993	1.2641	136.3319	425	595.1836	59.1058

The trend is similar for a single HMM as it is for the entire system. The same is displayed graphically in Figure 16.

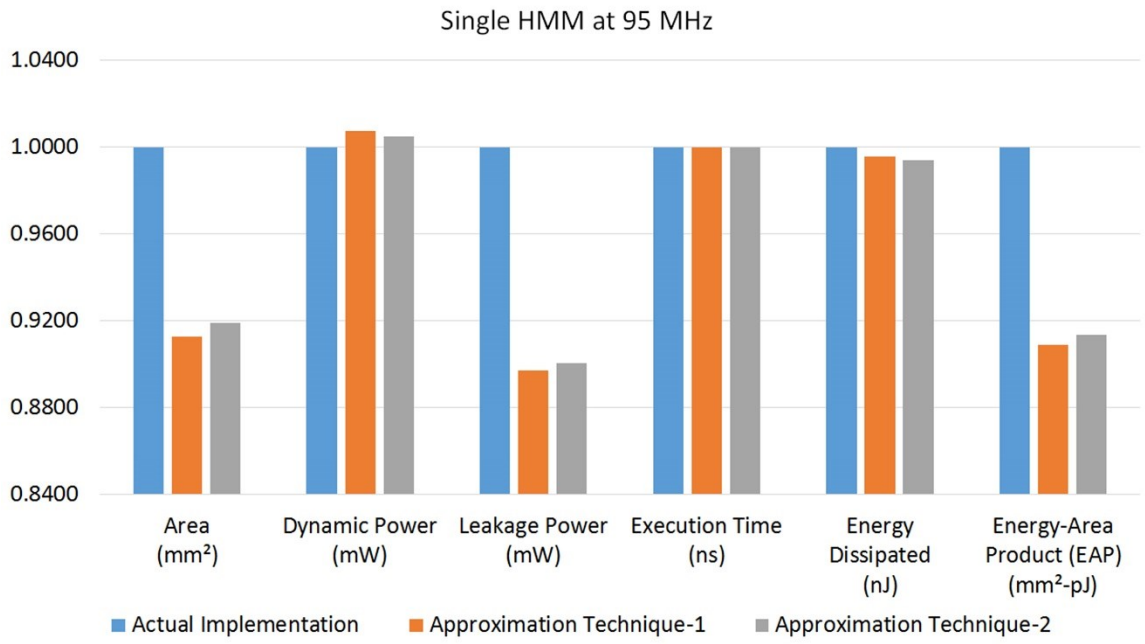

Figure 16: Hardware evaluation of a single HMM system at 95 MHz

8.3. Frequency Scaling

The system clock rate of 95 MHz was the maximum possible with the Actual Implementation and the Approximate Technique-1. But the use of approximate computing facilitates a frequency scaling and Approximate Technique-2 can run at a slightly higher clock rate.

Table 3 provides the evaluation of all three designs at their respective maximum frequencies. Approximate Technique-2 is capable of running at a frequency of 110 MHz whereas the maximum frequency of the other two designs are 95 MHz. The variation in area and leakage power is similar as mentioned in Section-8.2. A higher frequency for Approximate Technique-2 translates to lesser execution time but more dynamic power

dissipation compared to the other two designs, as is shown in the table. But the energy dissipation is lower for Approximate Technique-2 by around 5% than the Actual Implementation and Approximate Technique-1. The combination of Area and Energy as depicted by the Energy-Area Product (EAP), is also lower for Approximate Technique-2 by around 15% than the Actual Implementation and by around 4% from Approximate Technique-1. In short, Approximate Technique-2 can run faster and dissipate less energy compared to both the Actual Implementation and the Approximate Technique-1 and consume lesser area compared to the Actual Implementation.

Table 3: Full systems, each at their respective maximum frequencies

	Frequency (MHz)	Area (mm^2)	Dynamic Power (mW)	Leakage Power (mW)	Execution Time (ns)	Energy Dissipated (nJ)	Energy-Area Product (EAP) (mm^2-nJ)
Actual Implementation	95	3.2442	36.7378	4.4599	425	17.5090	56.8035
Approximate Technique-1	95	2.8729	37.0367	3.9147	425	17.4043	50.0011
Approximate Technique-2	110	2.8925	41.5635	3.9244	370	16.8305	48.6817

Figure 17 graphically depicts the area, power, execution time, energy dissipated and the EAP across all three designs running at their respective maximum frequencies. All values are normalized over those of Actual Implementation.

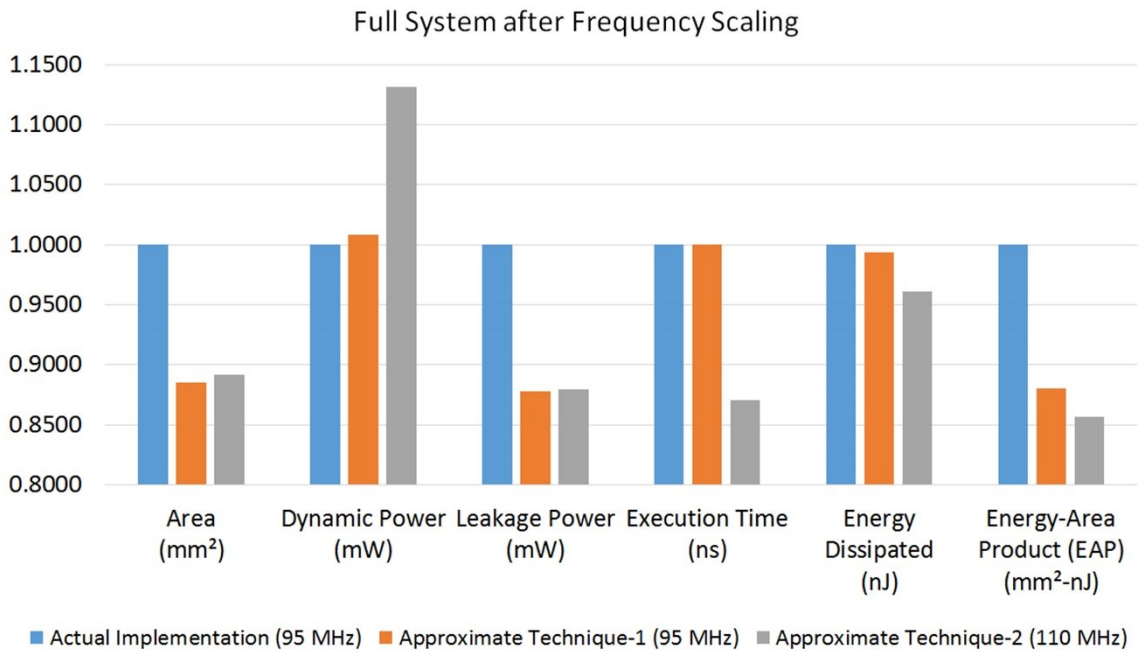

Figure 17: Hardware evaluation of the full system after frequency scaling

Similar results are provided for a single-HMM subjected to frequency scaling, in Table 4. The trend observed across the three designs is similar to the one for the full system, with the Approximate Technique-2 being superior in frequency, execution time, energy dissipated and the EAP.

Table 4: Single HMMs, each at their respective maximum frequencies

	Frequency (MHz)	Area (mm^2)	Dynamic Power (mW)	Leakage Power (mW)	Execution Time (ns)	Energy Dissipated (nJ)	Energy-Area Product (EAP) (mm^2-pJ)
Actual Implementation	95	0.1080	1.2577	151.3901	425	598.8633	64.6966
Approximate Technique-1	95	0.0986	1.2674	135.8360	425	596.3753	58.8118
Approximate Technique-2	110	0.0993	1.4241	136.3162	370	577.3540	57.3315

Figure 18 graphically depicts the area, power, execution time, energy dissipated and the EAP for a single HMM running at its maximum frequency across all three designs.

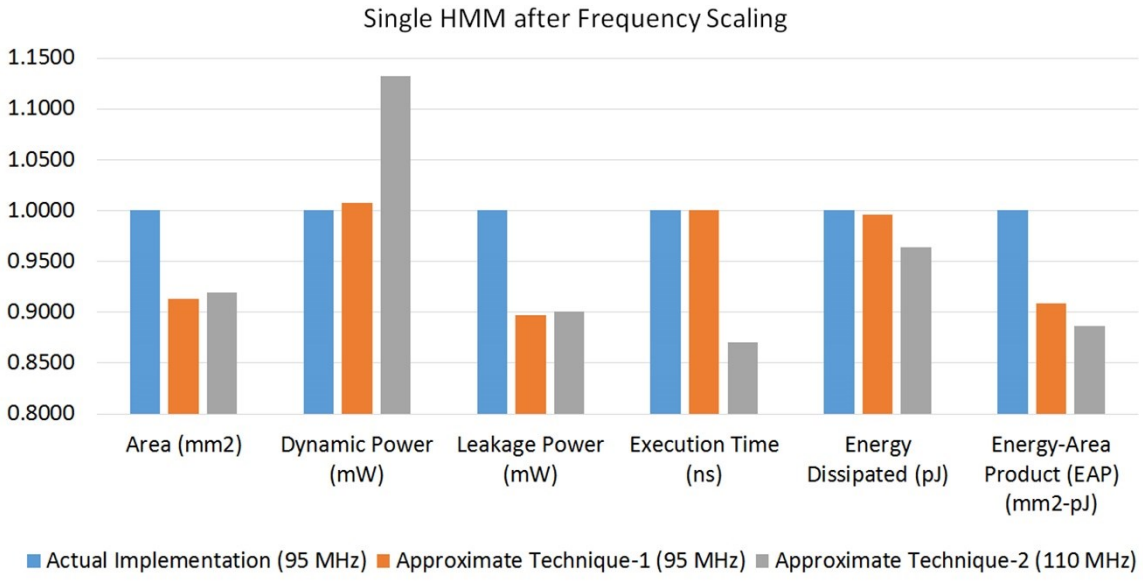

Figure 18: Hardware evaluation of a single HMM system after frequency scaling

8.4. An Adaptive System

Considering the above results, to maintain the benefits of both the Approximate Techniques, an adaptive system is proposed which is capable of operating in both the modes – one with Technique-1 and the other with Technique-2.

Approximate Technique-1 does not compromise on the accuracy of the full-precision implementation while consuming lesser area. Approximate Technique-2, although slightly less accurate, consumes similar area and power as that of Technique-1, but can be operated at a higher frequency with lesser energy consumption. Hence by incorporating the properties of both, an adaptive system can switch between having higher accuracy and running at a higher speed.

Such an adaptive system will operate in Approximate Technique-2 by default at a higher frequency. The system will shift to Approximate Technique-1 at a lower frequency, when triggered by a signal generated based on accuracy levels.

This trigger is generated at the cascaded comparator level, while comparing the probabilities generated by each HMM. If the probabilities are "too-close" based on a fixed threshold value, the system will generate a signal to change the mode of operation and shift to Approximate Technique-1.

For all the experiments, the threshold was set as the minimum difference between likelihood probabilities for an Approximate Technique-1 computation with a 10% tolerance. If the computation using Approximate Technique-2 resulted in likelihood

probabilities closer than that of Approximate Technique-1, then the trigger signal is generated and the values are re-computed.

The system runs at a frequency of 110 MHz by default and if the mode changes, the system will re-compute the current operation again at 95 MHz, thus consuming a total power equal to the sum of the power dissipated by the two modes respectively. If the mode of operation does not change, the system continues to operate at 110 MHz, dissipating only the power equivalent to that for Approximate Technique-2.

Table 5 below reports the performance of the adaptive system. The worst case scenario numbers are for a single experiment when the mode changes from Approximate Technique-1 to Approximate Technique-2. As mentioned above, in such a scenario the system re-computes the current operation, resulting in execution time, power and energy being the sum of both the modes individually. But the mode does not change frequently as is shown by the average numbers. The average numbers are obtained by conducting experiments and averaging out the results based on change of mode. A weighted average is computed by attributing the worst-case values only to the scenarios when the trigger signal is fired. When the trigger does not get fired, the values correspond to those of Approximate Technique-2.

Table 5: Performance of the adaptive system

	Worst Case	Average
Execution Time (ns)	795	497.5
Dynamic Power (mW)	2.6915	1.8043
Leakage Power (mW)	272.1522	177.0670
Energy Dissipated (nJ)	218.5007	88.9885

In comparison with the Approximate Technique-1, the adaptive model can have better speed and lesser energy consumption. In comparison with Approximate Technique-2, the adaptive model can have higher accuracy. Thus, the adaptive model includes the advantages of both the Approximate Techniques and can perform better than each Approximate Technique individually.

9. FUTURE WORK

The system needs better recognition accuracy to be used commercially. Using tri-phone models and modelling the output probabilities with tied-state Gaussian mixture models are a common practice to increase accuracy (Young, Odell, & Woodland, 1994). But considering the complexity of its implementation, the immediate scope of this work can be to increase the training samples and also to use word-level HMMs.

Another way to improve accuracy can be to have hardware and software co-exist. The hardware design would be responsible for accelerating the implementation of the Viterbi algorithm to obtain likelihood probabilities for each phoneme and the software can use grammar models and contextual information to use the phoneme likelihood probabilities and complete the recognition process for the entire speech.

In order to help make the current system standalone, trained model parameters need to be stored and fetched from memory. The current memory requirement is small and hence this is not considered in this thesis. But for commercial applications, accesses to memory would provide additional overhead and would require careful design with constraint on area and timing.

10. CONCLUSIONS

This thesis describes the design of hardware for recognizing speech using models trained offline. The design has been optimized for area and power using approximation computing techniques. Out of two such techniques proposed, both consume similar area and power (lesser than the original implementation), while differing in frequency of operation and accuracy.

For scenarios where accuracy cannot be compromised, a combined adaptive system design is proposed. Such a design works at a higher frequency with lower accuracy by default. But it is capable of operating with standard accuracy at a slower speed, and changes modes based on a trigger. The trigger can be controlled by setting the threshold limit.

This implementation can help offload the complex recognition task from the CPU when implemented as part of an S-o-C (System-On-Chip).

REFERENCES

Bacon, S. P., Fay, R. R., & Popper, A. N. (2004). *Compression: From Cochlea to Cochlear Implants*: Springer.

Bok-Gue, P., Koon-Shik, C., & Jun-dong, C. (2002). *Low Power VLSI Architecture of Viterbi Scorer for HMM-based Isolated Word Recognition.* Paper presented at the Quality Electronic Design, 2002. Proceedings. International Symposium on, San Jose, CA, USA.

Chippa, V. K., Mohapatra, D., Raghunathan, A., Roy, K., & Chakradhar, S. T. (2010). Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency, *47th Design Automation Conference. Proceedings* (pp. 555-560). Anaheim, CA, USA: ACM.

Davis, S., & Mermelstein, P. (1980). Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on, 28*(4), 357-366.

Dymarski, P. (2011). Hidden Markov Models, Theory and Applications. In: InTech Open Access Publishers.

Esmaeilzadeh, H., Blem, E., St Amant, R., Sankaralingam, K., & Burger, D. (2011). *Dark Silicon and the End of Multicore Scaling.* Paper presented at the Computer Architecture (ISCA), 2011. 38th Annual International Symposium on, San Jose, CA, USA.

Forney, G. D., Jr. (1973). The Viterbi Algorithm. *Proceedings of the IEEE, 61*(3), 268-278.

Gales, M. J. (2000). *Factored Semi-Tied Covariance Matrices.* Paper presented at the NIPS, Denver, CO, USA.

Hegde, R., & Shanbhag, N. R. (2001). Soft Digital Signal Processing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 9*(6), 813-823.

İnce, A. N. (1992). *Digital Speech Processing: Speech Coding, Synthesis and Recognition*: Boston, MA : Springer US, 1992.

Kim, Y., Zhang, Y., & Li, P. (2013). *An Energy Efficient Approximate Adder with Carry Skip for Error Resilient Neuromorphic VLSI Systems.* Paper presented at the Computer-Aided Design. Proceedings. International Conference on San Jose, CA, USA.

Kim, Y., Zhang, Y., & Li, P. (2014). Energy Efficient Approximate Arithmetic for Error Resilient Neuromorphic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, PP*(99), 1,1.

Lee, K.-F. (1988). On Large-Vocabulary Speaker-Independent Continuous Speech Recognition. *Speech Communication, 7*(4), 375--379.

Melnikoff, S. J., Quigley, S. F., & Russell, M. J. (2002, 2002). *Implementing a Simple Continuous Speech Recognition System on an FPGA.* Paper presented at the Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on, Napa, CA, USA.

Ohala, J. J. (1993). Coarticulation and Phonology. *Language and Speech, 36*(2-3), 155-170.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257-286.

Rabiner, L. R., & Juang, B.-H. (1986). An Introduction to Hidden Markov Models. *ASSP Magazine, IEEE, 3*(1), 4-16.

Rabiner, L. R., & Juang, B.-H. (1993). *Fundamentals of Speech Recognition*: PTR Prentice Hall.

Rabiner, L. R., & Schafer, R. W. (2007). Introduction to Digital Speech Processing. *Found. Trends Signal Process., 1*(1), 1-194.

Rodriguez-Andina, J. J., Fagundes, R. D. R., & Junior, D. B. (2001, 2001). *A FPGA-based Viterbi Algorithm Implementation for Speech Recognition Systems.* Paper presented at the Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). IEEE International Conference on, Salt Lake City, UT, USA.

Rosen, K., & Yampolsky, S. (2000). Automatic Speech Recognition and a Review of its functioning with Dysarthric Speech. *Augmentative and Alternative Communication, 16*(1), 48-60.

Sahidullah, M., & Saha, G. (2012). Design, Analysis and Experimental Evaluation of Block Based Transformation in MFCC Computation for Speaker Recognition. *Speech Communication, 54*, 543-565.

Schuster, M. (2010). Speech Recognition for Mobile Devices at Google. In B.-T. Zhang & M. Orgun (Eds.), *PRICAI 2010: Trends in Artificial Intelligence* (Vol. 6230, pp. 8-10): Springer Berlin Heidelberg.

Shao, B., & Li, P. (2014). *A Model for Array-based Approximate Arithmetic Computing with Application to Multiplier and Squarer Design.* Paper presented at the Low Power Electronics and Design, 2014. Proceedings. International Symposium on, La Jolla, CA, USA.

Shao, B., & Li, P. (2015). Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design. *Circuits and Systems I: Regular Papers, IEEE Transactions on, 62*(4), 1081-1090.

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., et al. (2004). *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*: Sun Microsystems, Inc.

Watada, J. (2009). *Intelligent Systems and Technologies: Methods and Applications*: Springer Berlin Heidelberg.

Xu, M., Duan, L.-Y., Cai, J., Chia, L.-T., Xu, C., & Tian, Q. (2005). HMM-Based Audio Keyword Generation. In K. Aizawa, Y. Nakamura & S. i. Satoh (Eds.), *Advances in Multimedia Information Processing - PCM 2004* (Vol. 3333, pp. 566-574): Springer Berlin Heidelberg.

Yoshizawa, S., Wada, N., Hayasaka, N., & Miyanaga, Y. (2006). Scalable Architecture for Word HMM-based Speech Recognition and VLSI Implementation in Complete System. *Circuits and Systems I: Regular Papers, IEEE Transactions on, 53*(1), 70-77.

Young, S. J. (1993). *The HTK Hidden Markov Model Toolkit: Design and Philosophy*: Citeseer.

Young, S. J., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., et al. (1997). *The HTK Book* (Vol. 2): Entropic Cambridge Research Laboratory Cambridge.

Young, S. J., Odell, J., & Woodland, P. C. (1994). Tree-based State Tying for High Accuracy Acoustic Modelling, *Human Language Technology. Proceedings. Workshop on* (pp. 307-312). Plainsboro, NJ, USA: Association for Computational Linguistics.