A Generic and Scalable Architecture for a Large Acoustic Model and Large Vocabulary Speech Recognition Accelerator Using Logic on Memory

Ojas A. Bapat, Paul D. Franzon, Fellow, IEEE, and Richard M. Fastow

Abstract—This paper describes a scalable hardware accelerator for speech recognition, which uses a two pass decoding algorithm with word dependent N-best Viterbi Beam Search. The observation probability calculation (Senone scoring) and first pass of decoding using a Bigram language model is implemented in hardware. The word lattice output from the first pass is used by software for the second pass, with a trigram language model. The proposed design uses a logic-on-memory approach to make use of high bandwidth NOR flash memory to improve random performance for Senone scoring and first pass decoding, both of which are memory intensive operations. The proposed HW/SW co-design achieves an overall speed up of 4.3X over a 2.4-GHz Intel Core 2 Duo processor running the CMU Sphinx speech recognition software, while consuming an estimated 1.72 W of power. The hardware accelerator provides improved speech recognition accuracy by supporting larger acoustic models and word dictionaries while maintaining real-time performance.

Index Terms—Accelerator, beam search, embedded, hardware software co-design, logic on memory, multipass decoding, N-best, speech recognition, sphinx.

I. INTRODUCTION

M AJORITY of continuous speech recognition algorithms use Hidden Markov Models (HMMs). Excessive memory bandwidth and computing power required to obtain high recognition accuracy in real time are the two main factors keeping speech recognition from being mass adopted on the embedded platform. Use of smaller acoustic models and word dictionaries to maintain real time performance induces inaccuracy in recognition. The high computational requirement uses up most of the resources on a general purpose CPU and the acoustic and language models use most of the cache and dynamic RAM (DRAM). This results in resource contention and leaves the CPU unable to do any other task along with speech recognition.

Manuscript received October 18, 2012; revised June 10, 2013; accepted December 3, 2013. Date of publication January 21, 2014; date of current version November 20, 2014. This work was supported by the Research under Contract 5582374 through Spansion Inc. and NC State University.

O. A. Bapat was with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695 USA. He is now with Spansion Inc., Sunnyvale, CA 94085 USA (e-mail: oabapat@ncsu.edu).

P. D. Franzon is with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695 USA (e-mail: paulf@ncsu.edu).

R. M. Fastow is with Spansion Inc., Sunnyvale, CA 94085 USA (e-mail: richard.fastow@spansion.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2296526

Existing hardware solutions [1]–[6] are far from a generic processor and are optimized for use with a given set of acoustic and N-gram language models. Many of them are unusable with other models while others suffer considerable performance degradation when used with other models.

Existing hardware Software Co-Designs [7]–[10] mainly calculate the observation probability in hardware. However, it is unclear as to which tasks in the algorithm should be fixed in hardware and which parts should be software controllable through commands, to allow maximum flexibility while providing improved performance.

In this paper, a scalable and portable hardware accelerator for speech recognition has been proposed. It accelerates the acoustic modeling and decoding process of the speech recognition algorithm. A multipass decoding approach [11], which splits the speech decode process into two parts has been used. The first pass is carried out on a large search space, using a simple language model and an N-best Viterbi Search [12], [13], which works reasonably well at coarse recognition and reduces the size of the search space. The second pass is carried out on a smaller search space, using more sophisticated N-gram [14] language models. The first pass has been implemented in hardware and the second pass in software. This keeps the performance of the hardware unaffected by the use of sophisticated models without compromising on end accuracy of the recognition process. Moreover, it makes hardware design much simpler. The software used here to demonstrate the benefits of the proposed architecture is Sphinx 3.0 [15], developed at Carnegie Mellon University.

This paper is organized as follows. Section II discusses the fundamentals of HMM-based speech recognition and multipass decoding. Section III describes the trade-offs between hardware complexity, power consumption and speech recognition accuracy. Section IV describes the proposed hardware architecture. In Section V, the modeling methodology used for performance, area and power estimation is presented. Finally, Section VI discusses results and comparisons with related work.

II. BACKGROUND

This section describes the working of an HMM-based speech recognition system. A complete speech recognition system is shown in Fig. 1. It mainly consists of three parts, front end digital signal processing, acoustic modeling, and
speech decoding. In the acoustic modeling stage, the features of incoming speech are compared to pretrained acoustic models to find which phones are closer to the observed speech. The decoding stage uses the Viterbi beam search [16] algorithm to find the most likely sequence of phones for the observed speech. Each stage in the algorithm uses models, which represent the probabilities of sounds, sequences of sounds, and words and sequences of words in the language. Gaussian distributions are used to represent nature of the sounds and HMMs are used to model sequences and duration of the sounds. Word sequences and their probabilities are stored as weights, which are added during the decode process. The models commonly used are shown in Table I.

### A. Front End

The goal of a speech recognition system is to recognize the uttered sequence of words. In the front end, input speech is sampled and a spectral analysis is performed to generate feature vectors to represent this speech. These feature vectors are generated at set intervals called frames. Each such feature vector is called an observation. The duration of a frame depends on the front end. CMU Sphinx uses 10-ms frames. Thus, the input speech is now converted to an observation sequence.

### B. Acoustic Modeling

Every spoken word in the language is represented in terms of basic sounds called phones. The pronunciation of every phone is affected by its context, i.e., the phone preceding it and succeeding it. Thus, the phones are clubbed with their neighboring phones to form a context dependent units called triphones. The total number of possible triphones can be very large for any language, e.g., there are 50 phones in English language and a possible 50^3 triphones. Each triphone is represented by a statistical HMM, as shown in Fig. 2. The proposed design supports N-state HMMs. In the rest of this paper, the terms triphone and HMM will be used interchangeably.

Each state in the HMM generates observation probabilities $B_j(Y_t)$ shown in (1). Each of the states in the HMM are represented by a multivariate Gaussian mixture models. In a multivariate mixture, vectors are used to represent the mean and variance parameters for the Gaussian distribution. The dimensionality of these vectors is governed by the dimensionality of the features used to represent incoming speech. This depends on various factors such as the type of front end filter/window functions used and input sampling rate. The parameters for these Gaussian distributions are set from training and are called the acoustic model. To avoid redundancy and to reduce training effort, states with the same Gaussian distributions are combined together to one state called a Senone [17].

In the acoustic modeling (observation probability calculation) stage of the recognition process, the observation probabilities of the Senones are calculated by calculating the distance between the incoming feature vector and the Gaussian distribution of the Senone, as shown in (1). This stage is called Senone scoring

$$\log b_j(Y_t) = \sum_{m=1}^{M} C_{jm} \sum_{n=1}^{N} (Y_{tn} - \mu_{jn})^2 * V_{jm}[n].$$

### C. Speech Decoding

The objective of the decoder is to find out the most probable sequence of words from the language model, given the observation sequence. The probability of this word sequence is shown in (2). The term $P(O|W)$ in (2) is called the observation or acoustic probability and is calculated by Senone scoring. It is calculated for every HMM state, during the decoding process. $P(W)$ is obtained from the language model

$$W_{1,...,W_n} = \arg_{w} \max P(W) \frac{P(O|W)}{P(O)}$$

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

Equation (3) is used to calculate the probability for HMMs in the language model over the entire observed sequence. The term $\delta_t(j)$ is the probability that an HMM would be in state $j$ for an observation $t$. This calculation needs to be repeated for each state in the language model that is active for the given observation frame $t$. Each HMM can be a sequence

---

**TABLE I**

### STATISTICAL MODELS USED IN SPEECH RECOGNITION

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acoustic (Senone) Model</td>
<td>models individual Senones using Gaussian mixtures</td>
</tr>
<tr>
<td>HMM (Tri-phone) Model</td>
<td>models context dependent phones using sequence of Senones</td>
</tr>
<tr>
<td>Word Model</td>
<td>models words using sequence of HMMs and inter-HMM transition probabilities</td>
</tr>
<tr>
<td>Language Model</td>
<td>models probabilities of sequences of words</td>
</tr>
<tr>
<td>Unigram Model</td>
<td>models probabilities of single words (context independent)</td>
</tr>
<tr>
<td>Bigram Model</td>
<td>models probabilities of sequences of two words (context dependent)</td>
</tr>
<tr>
<td>Trigram Model</td>
<td>models probabilities of sequences of three words (context dependent)</td>
</tr>
<tr>
<td>N-gram Model</td>
<td>models probabilities of sequences of N words (context dependent)</td>
</tr>
</tbody>
</table>

---

Fig. 1. Complete speech recognition system.

Fig. 2. HMM for a triphone.
of \( N \) states. In the models used, each HMM consists of three states and is called a triphone. Viterbi Beam search [18] keeps HMMs, which are above a predetermined threshold active, since it is impractical to calculate the probability of all HMMs in the language model. Previous work [18] suggests that best possible recognition accuracy for a given language model can be obtained with approximately 10% of the HMM states in the language model kept active. Simulations with CMU Sphinx confirm that this is true for the Wall Street Journal CSR I corpus.

D. Multipass Decoding Using N-Best Search

Single pass decoding uses the Viterbi beam search algorithm and computes the most probable sequence of phones, given the observed speech. This sequence of phones usually corresponds to the most probable sequence of words. However, this assumption may not be true in case the probability path consists of a word in the dictionary with multiple pronunciations. Also, the Viterbi algorithm assumes the dynamic programming invariant, i.e., if the ultimate best path for the observation passes through state \( K \), then it must include the best path up to and including state \( K \). Higher N-gram [14] language models do provide better accuracy by attaching probabilities to sequences of words, but they also violate the dynamic invariant and need backtracking to find the best path or risk losing this information.

In multipass decoding [11], the decoding process is divided into two passes. The first pass of decode follows N best paths at every node [11], [12] and uses a simple unigram/bigram model. In the proposed design, the first pass is implemented in hardware using a word-dependent N-best search [13], which follows multiple paths at word level. This approach has less complexity compared to other approaches [13] and is very generic as it does not make use of multiple word sequences. A unigram/bigram language model is used. This model stores probabilities for individual words and pairs of words only and does not violate the dynamic programming invariant. This eliminates the need for backtracking and simplifies the hardware implementation. Also, it allows us to use a wide beam width and a larger vocabulary for the Viterbi search. The output of this coarse first pass decoding step is a lattice of identified words. This lattice includes multiple paths and not just the best path. This lattice is of a much lower order, compared with the entire word vocabulary. The second decoding pass, which is implemented in software uses this word lattice as the input. The second pass rescores the word lattice using a sophisticated N-gram language model, e.g., trigram, to obtain the best hypothesis.

The proposed hardware software partitioning scheme is shown in Fig. 3. Use of multipass decoding with N-best search on a unigram/bigram model language model makes this implementation very generic and usable as the coarse first pass for speech recognition in any application. The beam width for the first pass has to be wide enough, so that it always includes the best possible hypothesis in the word lattice. The second pass of decode is in software and can be very application specific. For example, it can have an N-gram for, Call XYZ from company ABC on cell phone. The multipass decoding approach is very well suited for a hardware software co-design because it helps keep the first decode pass generic and does not require any feedback from the second decode pass. This allows the speech recognition front end, Senone scoring and both decode passes to work completely in parallel.

III. DESIGN SPACE EXPLORATION

A SystemC model of the hardware was developed and used with CMU Sphinx 3.0 [15] for design space exploration and performance analysis of the speech recognition system. The parameterized SystemC Model allowed investigation of the tradeoffs for various HW/SW partitioning schemes and the impact of various hardware configurations on the overall performance of the system. Both the hardware and the communication interface were modeled in SystemC to observe the end-to-end latency for any operation. The parameters of the hardware model for each HW/SW split investigated were chosen such that the hardware performs in real time for each split. The results of the simulation for each split represent the impact of software and interface latency for the HW/SW Co-design. The details of the SystemC Model are discussed in Section V.

The first step in design space exploration is to estimate the CPU and memory bandwidth requirement for the speech recognition algorithm (Table II). The Senone scoring process is the most CPU and memory intensive as it involves calculating two summations and reading the entire acoustic model. This phase can greatly benefit from hardware acceleration. The decode stage is not as CPU and memory intensive, however, the random nature of memory accesses and data dependency between them slows down this phase of the algorithm. The second pass of decode works on the word lattice, which is very small compared with the entire dictionary. Hence, it does not require much CPU and memory bandwidth. There are multiple hardware/software partitions possible for an HMM-based speech recognition system. One is to accelerate only the observation probability calculation stage [1], [2], [9] in hardware while the word search is performed in software (Split1). The next step is to accelerate the processing of HMMs (triphones). In this split (Split2), the HMM scores are sent back to the software every frame. The software performs the transitions from one HMM to the next. In Split3, the hardware performs transitions for HMMs within a single word, and the inter word transitions are performed by software. The hardware sends back a list of identified words to the software.
TABLE II
CPU AND MEMORY BANDWIDTH REQUIREMENTS FOR VARIOUS STAGES IN THE SPEECH RECOGNITION PROCESS

<table>
<thead>
<tr>
<th>Stage Description</th>
<th>CPU</th>
<th>Memory BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senone Scoring (8000-senones, 8-mixtures, 39-features)</td>
<td>1.1 GOPs</td>
<td>2.05 GBps</td>
</tr>
<tr>
<td>Decode Pass I (64K-Bigram)</td>
<td>192 MOPS</td>
<td>0.46 GBps</td>
</tr>
<tr>
<td>Decode Pass II (64K-Trigram)</td>
<td>15.3 MOPS</td>
<td>24.9 MBps</td>
</tr>
</tbody>
</table>

TABLE III
HARDWARE SOFTWARE PARTITIONS INVESTIGATED

<table>
<thead>
<tr>
<th>Partitioning Scheme</th>
<th>Split 1</th>
<th>Split 2</th>
<th>Split 3</th>
<th>Split 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senone Scoring</td>
<td>HW</td>
<td>HW</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>HMM Scoring</td>
<td>SW</td>
<td>HW</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Word Scoring</td>
<td>SW</td>
<td>SW</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Word Transitions</td>
<td>SW</td>
<td>SW</td>
<td>SW</td>
<td>HW</td>
</tr>
<tr>
<td>Second Pass of Decoding</td>
<td>SW</td>
<td>SW</td>
<td>SW</td>
<td>SW</td>
</tr>
</tbody>
</table>

Fig. 4. Performance comparison with sphinx 3.0 running on desktop PC with various HW/SW splits for acceleration.

TABLE IV
COMMUNICATION BANDWIDTH REQUIREMENT FOR VARIOUS HW/SW SPLITS

<table>
<thead>
<tr>
<th>Partitioning Scheme</th>
<th>Communication Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split 1</td>
<td>3.22 MBps</td>
</tr>
<tr>
<td>Split 2</td>
<td>84 MBps</td>
</tr>
<tr>
<td>Split 3</td>
<td>0.76 MBps</td>
</tr>
<tr>
<td>Split 4</td>
<td>0.136 MBps</td>
</tr>
</tbody>
</table>

Fig. 5. WER for different language models in pass I and trigram in pass II.

and receives a new list of words to be activated every frame.
The next step is to bring in the inter word transitions into hardware (Split4). Table III shows various HW/SW splits investigated in this paper.

The software used was Sphinx 3.0 running on a 2.4-GHz Intel Core 2 Duo processor with 4-GB RAM. The acoustic model used had 8000 Senones, eight Gaussian mixtures per Senone [19], a 64-K word dictionary, a bi-gram language model for first pass of decode and trigram language model [20] for the second pass. The goal was to observe the performance improvement and communication bandwidth requirements for all possible HW/SW partitions, while keeping the word error rate (WER) constant. Fig. 4 and Table IV show the results of simulations. It can be observed that implementing more and more tasks into hardware, achieves better performance improvement. Offloading of inter word transitions (Split4) into hardware provides a huge performance benefit, since it involves lot of memory accesses. The improved performance translates to better end accuracy of speech recognition by use of larger acoustic and language models in real time.

Another factor, which is important in a HW/SW co-design is the communication bandwidth requirement between the hardware accelerator and CPU. Table IV shows the communication requirements for various HW/SW splits. Split1, which is implemented in many systems today [1], [2], [8], [9] requires nominal communication bandwidth. This system provides good performance improvement for large acoustic models but not for large language models, as most of the time is spent by software in doing the word search. Split2 requires large communication bandwidth and requires a very fast interface. Split3 is very efficient in terms of bandwidth requirements, however, interword transitions, which are memory intensive and can benefit from hardware acceleration, are still done in software. Moreover, in this partitioning scheme, to maintain sequential DRAM access in hardware (this provides huge performance boost), the incoming list of words to be activated has to be sorted by software to match the order of the active word list in hardware. This results in additional software overhead. Split4 offers both low communication bandwidth and high performance improvement. Moreover, this scheme allows the hardware to run in parallel to the software, without any feedback from software. The output of the hardware is a reduced search space (word lattice) on which the software can work in the second decode pass, to find the best hypothesis. The HW/SW split chosen for the proposed design is Split4.

For a high accuracy real time speech recognition, it is necessary to support large acoustic models and large word vocabularies. Another factor, which affects accuracy is the use of sophisticated language models, which model probabilities of sequences of words, rather than individual words. As observed in Fig. 5, a trigram language model performs better than a bi-gram model for a multipass decoder with one best algorithm. It is difficult to build hardware, which is optimized to work equally well with any N-gram language model. Working with larger N-gram models increases number of random accesses to memory. It also greatly complicates the architecture as the history of last N words and back track of the best path are now necessary. In addition, searching or
hashing techniques are needed to determine if an N-gram is present in the model for a given active word sequence. Instead, a bi-gram model with an N-best algorithm with a wider beam width in the first pass can be used to improve WER. This is evident from simulation results shown in Fig. 5. The WER obtained using bi-gram models with increased beam width for the first pass and trigram models in the second pass, is statistically similar to WER obtained using a trigram model in both passes.

In this paper, the hardware is designed to work with unigram/bigram language models, to reduce hardware complexity. Unigrams can be indexed directly using word indices (IDs) and Bigrams can be indexed using either the source or destination word ID. Direct indexing keeps memory accesses very simple. The multipass decode approach [11] is implemented, where the first decode pass is a word dependent approximate N-best time synchronous Viterbi beam search algorithm [12], [13], which returns a word lattice representing multiple best paths rather than a single path. The beam width for the first pass, which uses a bigram model is increased, making sure that the best hypothesis is not omitted from the output of the first pass. The second pass using trigram language model works on the word lattice generated in the first N-best pass and chooses the best hypothesis.

Design space exploration was started with a baseline design for the hardware, as shown in Fig. 6(a) where the Senone score unit (SSU) and Viterbi decoding unit (VU) share the same memory and do not work at the same time. The decoding stage provides feedback to the Senone scoring stage, which activates Senones which need to be scored for the next frame. As seen in Fig. 9, for a system, which uses a wide beam width (larger than 10%) and a large vocabulary (64-K words), 92.2% of the Senones are always active, as compared to 25% for a smaller vocabulary of 5 K with a narrow beam width. The activation of Senones for each frame not only introduces added complexity and memory requirement, but also introduces dependency between the decode stage and the acoustic modeling stage. Breaking this feedback loop allows the Senone scoring to be performed independently and in parallel to the decode stage with little overhead of scoring 8% more Senones. While the decode stage works on frame N, the Senone scores for next frames can be calculated.

This architecture is shown in Fig. 6(b). This offers significant performance improvement for a modest increase in power consumption (see Fig. 8).
For the senone scoring stage, the entire acoustic model has to be read for each frame. For the decode stage, the HMM data and language model weights need to be read for each active HMM. Table V shows the distribution of data read in this stage. It turns out that 86% of the data accesses are non-volatile. Hence, the proposed design uses the logic on memory approach with an on-chip high bandwidth NOR flash memory [Fig. 7(a)]. A combination of on-chip static RAM (SRAM) and off-chip DRAM was used for storing the non-volatile data like senone scores and active HMM scores. Simulations were performed to determine the access efficiency of multiple memory configurations (see Table VI). Memory efficiency is defined as the ratio of the actual bandwidth achieved in simulation and the theoretical maximum memory bandwidth for the given memory configuration. For Senone Scoring, since the entire acoustic model is read sequentially, the DRAM provides good access efficiency. The Flash memory performs slightly better because it does not need activation, refresh and precharge. For the Viterbi decoder, the access to word models, HMM models and Bigrams are random. Hence, memory access efficiency is greatly improved by storing models in a high bandwidth NOR flash memory, which supports fast random access.

Two factors make the architecture in Fig. 7(b) beneficial for this design. For the decode phase, the word model/HMM model can be read only once per word and cached, so that it can be reused for the remaining active instances of the word. Fig. 10 shows that with increase in the number of instances for a word, the cache hit rate increases dramatically. The optimal size of the cache was found to be as small as five HMMs within a word. The cache is purged moving from one word to the next. Higher cache hit rate means that the available flash bandwidth can be used to prefetch multiple rows of bigrams for word transitions. This masks the bigram fetch latency and improves performance. Based on the factors discussed in this section, the architecture shown in Fig. 7(b) and HW/SW Split4 were chosen for this design and are discussed in detail in upcoming sections.

Simulations were performed with 400 utterances from the Wall Street Journal CSR I corpus, using the proposed multipass N best decode approach. The results of these simulations are shown in Fig. 11. It was observed that a large 64 K vocabulary and an acoustic model with 8 Gaussian mixtures is the optimal design point (see Fig. 11). Increasing the number of Gaussian mixtures beyond 8 did not provide much improvement in WER for the 64 K vocabulary. Hence, the design point chosen was an acoustic model with 8000 senones, 8 Gaussians and a bi-gram language model with a 64 K word dictionary. The hardware was designed to meet this requirement in real time, including the communication between hardware and CPU.

### IV. Proposed Hardware Accelerator

This section describes the basic blocks of the proposed hardware accelerator. The system consists of three main units (see Fig. 12). The Interface Control Unit decodes the commands and data obtained from software and controls the functioning of the SSU and the Viterbi Unit (VU). It is also responsible for sending the word lattice back to the software when it requests for it. The SSU calculates the observation probabilities, i.e., senone scores, for the entire library of senones. The VU calculates the state probabilities i.e., state
scores, for all active states in the language model. In addition to this, it applies pruning thresholds to the states and activates new states if needed. It also adds recognized words to the word lattice as and when they are recognized.

A. Senone Score Unit

The SSU is shown in Fig. 13. It calculates the scores for each Senone using (1). Once the start command is received, the Senone ID incrementer loops through the entire list of Senones in the library. This module is highly pipelined to provide sustained throughput. Multiple distance calculation units are used in parallel to consume all the data provided by high bandwidth NOR flash.

1) Block Senone Scoring: The CPU sends a block of feature vectors for two consecutive frames to the hardware. The SSU computes the Senone scores for both the frames simultaneously while reading the acoustic model just once. The calculation of scores is split across the entire length of two frames and hence does not require increased parallelism. The memory bandwidth for the NOR flash is reduced by a factor of two. The size of the Senone score SRAM increases by a factor of 2, since storage is required for scores of two consecutive blocks, for the decode stage to use, as shown in Fig. 14.

Scoring of all Senones for each frame implies that the feature vector is the only changing component every frame. This can be used to share acoustic model reads across frames. This technique is a variation of subvector clustering used in [21]. Scoring of the entire Senone library is spread across multiple frames. Fig. 15 shows the effect of block size on the power consumption of the SSU. This design uses a block size of two because at the 180-nm technology node, the design is limited by SRAM size.

The data pipeline is shown in Fig. 14. The decode stage is delayed by two frames. The Senone scoring stage works on two sets of feature vectors simultaneously over the entire two frames while reading the acoustic model only once. This reduces total memory reads, peak bandwidth requirement and subsequently the read power consumption. It adds a latency of two frames but still has real-time throughput. The size of the Senone score SRAM is increased by a factor of 2 since scores for the previously scored block of frames are still being used.

2) Flash Control and Memory Structure: The Flash Control unit translates the acoustic library offset and Senone ID into the first memory address of the Senone entry. A packed data structure is used for Senones in the library, to ensure that the performance is limited only by the physical bandwidth of the memory. At the beginning of every Senone, the length of each Senone record is stored. This helps us identify the end of a Senone. At the beginning of each library, the number of Senones is stored, which helps identify the end address of the last Senone in the library. The data structure used to store the acoustic model (Fig. 16) allows use of multiple acoustic models with different number of Senones, Gaussian mixtures or feature vector dimensions.
3) Distance Calculation: This module computes the inner summation for (1). It has four parallel units for the subtraction-square-multiply operations. The output of these units is used by two stages of addition. Scalability for larger acoustic models can be easily achieved by adding more flash memories and distance calculation units in parallel, as shown in Fig. 13.

4) Logarithmic Addition: Since, all the operations are in logarithmic domain, a logarithmic addition is required for the outer summation of (1). This involves calculating the value of $\log(A + B)$ from $\log(A)$ and $\log(B)$. This is done using a lookup table similar to the one used by CMU Sphinx [15]. This unit needs pipelining as it has to access the lookup table. However, no parallelism is required since this operation takes place only once for $N$ distance calculations.

B. Viterbi Unit

The VU (Fig. 17) is responsible for performing the first decode pass on the incoming speech using a simple unigram/bigram language model. It has a long pipeline, which works on each HMM from the active list, which is streamed from the DRAM sequentially. The random accesses that are required for the state score calculation in (3) are the transition probabilities $\log a_{ij}$ and Senone scores $\log b_j(Y_t)$. These accesses are divided between the language model stored in flash and the Senone scores stored in the SRAM by the SSU. The pipelining of these accesses provides further performance benefit.

1) Flash Control and Memory: The bigram language model, word structures and probabilities and the HMM structures are stored in the flash memory. The word dictionary stores the sequence of HMMs, which form the word. The HMM dictionary stores the Senones and transition probabilities within an HMM. The bigram model stores the probabilities going from one source word to multiple destination words. Each word in the dictionary occupies a line in a 256 bit wide line of memory and can be accessed in a single read. Similarly, an HMM up to three states (a triphone) can be accommodated in a line of memory and accessed in a single read. All the word and HMM models that are larger than one line in memory are stored in the form of a linked list (Fig. 18). The each node of the linked list is one line in flash memory. The first nodes of the linked list can be accessed directly by translating the word ID or HMM ID. The each subsequent node stores the pointer to the next node. The word model and triphone structures are accessed only once per word and stored in a cache. The structure of the active list (Section IV-B.4) allows reuse of this data for multiple instances of the same word. The bigrams are indexed by source word ID. Multiple lines of bigrams for each source word are prefetched into the SRAM. The prefetch SRAM is divided equally to store multiple bigrams of all possible source words in that frame. The bigrams are read from this SRAM in the order of destination word IDs as each word from the active list is being processed. For larger language models, multiple parallel flash memories and DRAMs can be used to improve memory bandwidth.

2) HMM Scoring: This block calculates (3) for each HMM. The last state scores are available from the HMM active list entry in DRAM. Transition probabilities and Senones IDs are obtained from the flash. The Senone IDs are used to read the corresponding Senone scores from the SSU SRAM. A simple add-compare select unit adds the last state scores, transition probabilities and Senone scores, compares them and then selects the best.

3) Adaptive Pruning: Adaptive pruning [22] is used to limit the size of the search space for the first pass of decode. The initial pruning threshold ($T_0$) is set by the software and then modified for each frame using equation (4). Here, $N_{set}$ is the maxhmmmpf parameter set by the user. $N_t$ is the number of states that are active in the current frame. This equation represents a closed-loop system, which adjusts $T_{t+1}$ to keep $N_{t+1}$ (number of Senones active in the next frame) as close to $N_{set}$ as possible. The value of $\alpha$ is set to 0.2 to dampen the response of this system. A 10% tolerance is added to $N_{set}$ to make sure that the number of HMMs passed is always more than $N_{set}$. This tolerance value was obtained empirically, by running 400 sentences from the Wall Street Journal CSR I Corpus. For any frame, the adaptive threshold is never used if it is wider than the initial beam threshold set by software.
Also, the adaptive threshold is not calculated unless the value of $N_t$ is larger than $N_{set}$. A similar equation is used for the word thresholds and pruning of N-best paths as well. This threshold is subtracted from the best HMM score for the previous frame $t$ and compared with every state score in frame $t + 1$ for pruning. For the word threshold, the same procedure is repeated with the word score. For N-best path pruning, the same technique is applied to multiple active list entries of the same word. Shown in Fig. 19 are the number of active states, which pass pruning for an example utterance, which lasts 381 frames.

\[ T_{t+1} = T_t + \alpha (1.1 \cdot N_{set} - N_t). \]  

4) **Active List Generation**: The active list is maintained at word level. There is a separate active list entry for each combination of a word and its predecessor. All the entries for one word are stored together. Each word entry contains entries for all the active HMMs within that word. For each active HMM entry, the scores of all states in the HMM are stored. For each word entry, its word ID, the word ID of the previous word it transitioned from, start frame of the first HMM of the word, the ID of the first HMM state in the word and the start frame of the last HMM of the word are stored. After the HMM is scored, the best of its state scores is compared with every state score in frame $t + 1$ for pruning. For the word threshold, the same procedure is repeated with the word score. For N-best path pruning, the same technique is applied to multiple active list entries of the same word. Shown in Fig. 19 are the number of active states, which pass pruning for an example utterance, which lasts 381 frames.

5) **Word Lattice Generation**: Whenever the last HMM of a word gets deactivated, it is added to the word lattice. For each word in the lattice, its word ID, previous word ID, score, start frame for the first HMM of the word, start frame for the last HMM of the word, and end frame for the last HMM of the word are stored. This is the same format as used by CMU Sphinx and is discussed in [11]. It provides both, path and time informations for each recognized word. Such a lattice can easily be converted into a word graph, N-best sentence list, N-best word list or any other lexical tree notation, which is required by the software.

6) **New Word and HMM Activation**: For activation of new HMMs within a word, the score of the last state of an HMM is checked against the HMM exit threshold. If this threshold is passed, the next HMM within the word is activated. The method for activating new words is not straightforward. Here, a transition has to be made, from a word that has exited, to all other possible words. This task is performed using a word activation map. This map represents the new entries that need to be activated for an exit word by checking the possible sounds it can transition to. As existing word entries are read from the active list, their previous word IDs are compared against the word lattice entries from the last frame. If entries already exist, the best of the existing and new entry is chosen. After all the existing entries for a word have been processed, the remaining new entries for the word are appended to the list. The word/unigram model in flash memory is grouped by the starting Senones of words and the same insertion order is maintained in the active list and word lattice. This ensures the lists never lose order and can be processed sequentially.

C. **Interface Between Software and Hardware**

The interface and control unit (Fig. 12) decodes the commands received from software. The SSU and VU are used to service these commands. A variety of commands have been defined to set configurable parameters for the proposed hardware. Some of the configurable parameters are as follows.

1) **ACOUSTIC_MODEL_OFFSET**: This is the memory offset for the acoustic model used. This lets the user switch acoustic models on the fly.
2) **LANGUAGE_MODEL_OFFSET**: This is the memory offset for the language model used. This lets the user switch language models.
3) **HMM_MODEL_OFFSET**: This is the memory offset for the HMM model used. This lets the user switch HMM models.
4) **HMM_INIT_BEAM**: This is the threshold value used to define the widest beam width to be used for pruning HMMs.
5) **WORD_INIT_BEAM**: This is the threshold value used to define the widest beam width to be used for pruning words.
6) **MAXHMPF**: This sets the maximum number of HMMs to be kept active for any frame. Adaptive pruning [22] is used to calculate the pruning threshold using the number of active HMMs from previous frames. The threshold is adjusted to keep the number of active HMMs close to $maxhmmpf$.
7) **MAXWPF**: This is similar to $maxhmmpf$ except it controls number of word exits.
8) **MAX_N_BEST**: This parameter sets the maximum instances of a single word to be kept active. Adaptive pruning is used to prune the N-best list as well. The pruning threshold for each word is different and is stored in the active list along with the word entry. This threshold is used only if the HMM and word pruning threshold are not able to keep the number of instances of a word below the N-best parameter.
9) **FEATURE_LENGTH**: This parameter is used to dynamically change the dimensions of the incoming feature...
vector from software. This allows us to be compatible with multiple front ends. It has been observed in [23] and [24] that for lower input sampling rates, lower dimensional feature vectors can be used without sacrificing accuracy.

10) HMM_LENGTH: This is the number of states for each HMM in the language model.

11) MAX_MIXTURES: This is the maximum number of Gaussian mixtures used for any Senone.

V. HARDWARE MODELING METHODOLOGY

A parameterized transaction level SystemC model of the proposed hardware architecture was developed. The SystemC model was transaction based at the boundary between the hardware and CPU. The model was integrated with CMU Sphinx. Sphinx was modified to send commands to the SystemC model of the hardware, which runs on a separate thread. The communication between the Sphinx and SystemC was done using a shared memory structure. This allowed observation of the effects of different hardware configurations on the end accuracy of the recognition process. An example of the interaction between the Sphinx and SystemC simulation threads is shown in Fig. 20. This model was used for design space exploration and for estimation of performance, area and power.

A. Timing, Area and Power Estimation

For area estimation, the total number of building blocks, which would be required for each operation modeled in SystemC was estimated. The individual components in synthesized Verilog were stitched together using glue logic. The total logic area was calculated by adding the areas of each of the basic block. The glue logic within the basic blocks was seen to add 20% overhead to the area. An additional 20% area overhead was added to account for similar glue logic on the top level.

For calculating the power, counters were used in the SystemC model to count the number of arithmetic operations on each basic building block and memory read/write operations for SRAM and flash. The power consumption for each basic block was obtained from synthesis and used to obtain energy per operation. The total power consumption for logic was obtained by multiplying the energy per operation with the total number of operations obtained from the SystemC simulation. This number was padded by another 20% to match the area overhead added for top level glue logic. For the SRAM and Flash, the read/write energy per operation was obtained from Spansion, Inc and multiplied with the total number of memory operations obtained from simulation. DRAM power was obtained by performing simulations using DRAMSim2 [26] memory system simulator. A SystemC wrapper was built around DRAMSim2. This wrapper passed commands and data between the hardware SystemC model and DRAMSim2. The DRAMSim2 simulation provided the time spent by the DRAM in each modes, i.e., read/write/activate/refresh/precharge. The data obtained from the SystemC/DRAMSim2 co-simulation was verified by performing a trace-based simulation with DRAMSim2 for the same data. The energy consumption was calculated using formulas provided by DRAM manufacturers to calculate power consumption for the DRAM in each operating mode [27]. The total power consumption for the hardware was calculated by adding the energy consumption for logic, SRAM, flash, and DRAM.

VI. RESULT

This design is estimated to run at a clock speed of 100 MHz, using Spansion CS239LS 180-nm standard cell library. The SSU supports processing of an Acoustic Library of 8000 Senones, eight Gaussian Mixtures, and a 39-D feature vector in 0.85x real time. It uses a 768 bit wide flash memory to store the acoustic model. The VU supports an N-best search using an unigram/bigram language model of 64-K words. Using a maximum N-best parameter value of 10, 0.78x real time performance was achieved for the VU. However, this number varies largely with input. Numbers as high as 0.92x were

<table>
<thead>
<tr>
<th>Task</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senone Scoring</td>
<td>867 mW</td>
</tr>
<tr>
<td>HMM Scoring</td>
<td>512 mW</td>
</tr>
<tr>
<td>HMM Transitions</td>
<td>138 mW</td>
</tr>
<tr>
<td>Word Transitions</td>
<td>206 mW</td>
</tr>
</tbody>
</table>
observed for one of the utterances in the test data. The VU uses a 256 bit wide flash memory to store bigram, word and triphone models. The flash memories used have a random read latency of 80 ns. An on-chip SRAM of 14 KB is needed for DRAM read/write buffers, log-add lookup tables, HMM dictionary cache, bigram prefetch cache, and word activation map. The SRAM requirement for storing Senone scores and word lattice depends on the size of acoustic and word models. In this design, 64 KB is needed for Senone scores and 16 KB for the word lattice. The estimated area is 200-K gate equivalents for SSU logic and 120-K gate equivalents for VU logic.

The SSU consumes an estimated 422 mW in the data path and 384 mW for memory reads. The VU consumes 230 mW in the data path and 359 mW for memory reads. The total estimated on-chip SRAM power consumption for SSU and VU is 143 mW. A comparison with previous work is shown in Tables VIII and IX. All the designs in these tables meet the real time performance criteria. The figure of merit chosen for comparison of SSU is the power consumed per Gaussian, as Gaussian is the lowest unit of the acoustic modeling stage. The work done in this stage can be easily represented in terms of number of Gaussians processed per frame. The figure of merit chosen for the VU is the power consumed per word in the dictionary. The amount of work done in this stage is proportional to the number of words in the vocabulary and the language model used. Since prior work uses different language models, only the number of words in the dictionary is used for comparison. It should also be noted that prior work does not implement the N-best algorithm.

Table VII shows the power consumed by different stages of the recognition algorithm. Senone scoring is the most power hungry as the entire acoustic model needs to be read once every two frames. The power consumption for other stages can be reduced by caching and using a NOR flash memory, which provides fast random accesses. The proposed hardware architecture provides high memory access efficiency using NOR Flash for random accesses (see Table VI). The random nature of word and HMM dictionary reads greatly reduces memory access efficiency when stored in a DRAM. The memory read power for VU is greatly reduced by caching HMM dictionary reads on an SRAM till all entries or a single word has been processed. An average cache hit rate of 65.74% was achieved for HMM dictionary reads. The second major factor in improving the performance and power efficiency of a VU is the use of a simplified unigram/bigram language model. The proposed Hardware/Software co-design provides an overall 4.3X performance improvement compared with a software only solution running on an Intel Core 2 duo 2.4-GHz processor with a 4 GB 667-MHz DDR2 SDRAM, while consuming an estimated 1.72 W. The proposed partition, which uses multipass decoding with word N-best search provides a generic hardware architecture, which can be used with software that runs any front end and N-gram language model.

### VII. Conclusion

In this paper, an architecture for generic speech recognition accelerator has been proposed. The goal was to achieve performance improvement using hardware acceleration while keeping the hardware architecture generic and portable. This involved accelerating the observation probability calculation stage of the algorithm while supporting multiple front ends. This was achieved with SIMD architecture for SSU, which supports multidimensional feature vectors and variable number of Senones and Gaussian mixtures. For the acceleration of the search phase of the algorithm, one of the biggest challenges was to decouple the hardware implementation from the software stack and the end application. Sophisticated N-gram language models provide accuracy improvements in speech recognition, especially if they can be made application specific. However, building a hardware architecture that works equally well with any N-gram model is a difficult task. In this design, a multipass decode algorithm was used, with simpler (unigram/bigram) language models in the first pass. The word-dependent N-best algorithm, which keeps track of N-best paths at each node, was used to compensate for the reduction in accuracy due to simpler language models. The second pass of decode was kept in software and could use any higher N-gram language model.

This paper implements the observation probability calculation stage and the word N-best decoder in hardware. The next step would be to look at any bottlenecks in the front end stage. The front end stage can be computationally intensive if it includes modules to combat noise, echo, reverberation

---

### TABLE VIII

**SSU Comparison with Related Work**

<table>
<thead>
<tr>
<th>Technology</th>
<th>Data Format</th>
<th>Feature Dimensions</th>
<th>Num. Mixtures</th>
<th>Num. Senones</th>
<th>Total Power</th>
<th>Power / Mixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8]</td>
<td>250nm</td>
<td>24 bit</td>
<td>39</td>
<td>8</td>
<td>6000</td>
<td>1.8W</td>
</tr>
<tr>
<td>[2]</td>
<td>180nm</td>
<td>32 bit</td>
<td>39</td>
<td>8</td>
<td>6000</td>
<td>7.35mW</td>
</tr>
<tr>
<td>[10]</td>
<td>130nm</td>
<td>32 bit</td>
<td>27</td>
<td>3</td>
<td>358</td>
<td>15.2mW</td>
</tr>
<tr>
<td>This Work</td>
<td>180nm</td>
<td>32 bit</td>
<td>39</td>
<td>8</td>
<td>8000</td>
<td>8671nW (est.)</td>
</tr>
</tbody>
</table>

### TABLE IX

**VU Comparison with Related Work**

<table>
<thead>
<tr>
<th>Technology</th>
<th>Data Format</th>
<th>Num. Words</th>
<th>Language Model</th>
<th>N-best?</th>
<th>Total Power</th>
<th>Power / Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6]</td>
<td>90nm</td>
<td>32 bit</td>
<td>5000</td>
<td>Trigram</td>
<td>No</td>
<td>190nW</td>
</tr>
<tr>
<td>[20]</td>
<td>130nm</td>
<td>32 bit</td>
<td>20000</td>
<td>Trigram</td>
<td>No</td>
<td>3.5W</td>
</tr>
<tr>
<td>This Work</td>
<td>180nm</td>
<td>32 bit</td>
<td>64000</td>
<td>Bigram</td>
<td>Yes</td>
<td>852mW (est.)</td>
</tr>
</tbody>
</table>
and other ambient conditions. It would also be interesting to observe how other more complex N-best algorithms and A* decoding algorithms perform in hardware.

ACKNOWLEDGMENT

O. A. Bapat would like to thank R. Eliyahu, J. Olson and the Spansion Design team for providing power consumption data for the on-chip NOR Flash Memory and SRAM and for access to their standard cell library.

REFERENCES


Ojas A. Bapat received the master’s and Ph.D. degrees from North Carolina State University, Raleigh, NC, USA, in 2009 and 2012, respectively.

He is currently a Systems Hardware Engineer with Spansion Inc., Sunnyvale, CA, USA. He was involved in a variety of topics, including speech recognition, in-situ calibration of circuits using an on-chip coprocessor, and DDR2 controller design. His current research interests include speech recognition algorithms, development of HW/SW co-designs and co-processors, and hardware performance modeling and design using SystemC/C++ and RTL.

Paul D. Franzon (SM’99–F’06) received the Ph.D. degree from the University of Adelaide, Adelaide, S.A., Australia, in 1988.

He is currently a Professor of electrical and computer engineering with North Carolina State University, Raleigh, NC, USA. He was with the AT&T Bell Laboratories, Murray Hill, NJ, USA, DSTD Australia, Australia Telecom and two companies, and he co-founded Communica and LightSpin Technologies. He has lead several major efforts and published over 200 papers. His current research interests include the technology and design of complex systems incorporating VLSI, MEMS, advanced packaging, and nano-electronics.

Dr. Franzon received the National Science Foundation Young Investigators Award in 1993 and the Alcoa Award in 2005. He was selected to join the NCSU Academy of Outstanding Teachers in 2001. He was selected as a Distinguished Alumni Professor in 2003.

Richard M. Fastow received the B.S. degree in physics from the Massachusetts Institute of Technology, Cambridge, MA, USA, and the Ph.D. degree in physics/material science from Cornell University, Ithaca, NY, USA.

He is currently the Director of speech technology and algorithms with Spansion, Inc, Sunnyvale, CA, USA. He was involved in nonvolatile memory at AMD and Intel. He holds 65 patents and has 31 publications. His current research interests include chip architectures for memory intensive tasks, such as speech recognition.