A 6 mW, 5,000-Word Real-Time Speech Recognizer Using WFST Models

A 6-mW, 5,000-word Real-time Speech Recognizer using WFST Models

Michael Price, *Student Member, IEEE,* James Glass, *Fellow, IEEE,* and Anantha Chandrakasan, *Fellow, IEEE*

***Abstract*—We describe an IC that provides a local speech recognition capability for any electronic device. We start with a generic speech decoder architecture that is programmable with industry-standard WFST and GMM speech models. Algorithm and architectural enhancements are incorporated in order to achieve real-time performance amid system-level constraints on internal memory size and external memory bandwidth. A 2.5 × 2.5 mm test chip implementing this architecture was fabricated using a 65 nm process. The chip performs a 5,000 word recognition task in real-time with 13.0% word error rate, 6.0 mW core power consumption, and a search efficiency of approximately 16 nJ per hypothesis.**

***Index Terms*—CMOS digital integrated circuits, speech recognition, weighted finite-state transducers (WFST), Gaussian mixture models (GMM), low-power electronics**

I. Introduction

Speech recognition has applications in energy-constrained devices that are not well served by existing software or hardware decoders. This paper shows how modern speech recognition techniques can be mapped to digital circuits, and presents a test chip with design techniques that reduce power consumption and memory bandwidth to levels appropriate for embedded systems.

We envision cooperation and interchangeability between cloud-based (software) and circuit-based (hardware) implementations of speech recognition. For Internet-connected devices, cloud-based decoders can be provisioned to run in real-time with decoding adding little to the latency of the network. Hardware speech recognition becomes useful when the Internet connection is slow or unreliable, when the overhead of using a cloud-based decoder is prohibitive (e.g. wearable electronics with limited battery capacity), or for local operations that do not require Internet capabilities (e.g. appliances and industrial equipment). As hardware speech decoders improve, client devices will be able to seamlessly transition between cloud-based and local decoding.

Speech recognition is an area of active research. While usefully accurate algorithms and models for domain-constrained tasks are well known, practical limitations have prevented the widespread adoption of hardware recognizers:

- Fixed-function digital circuits cannot easily be reprogrammed to keep up with developments in algorithms and statistical modeling techniques.
- Circuits developed so far require complex external components and interfaces to realize complete speech-to-text decoding.
- The memory size and bandwidth requirements of speech decoding (often similar to those of a general-purpose computer) are excessive.

The remainder of this paper explores and addresses these problems. We first discuss background and previous efforts that lay the foundation for this work (section II) and a circuit/system architecture (section III) for complete audio-to-text speech recognition. Section IV describes architectural enhancements that make real-time decoding feasible in a low-power system. Section V presents and characterizes a 65 nm test chip used to evaluate this architecture, and section VI concludes.

II. Background

Speech recognition technology evolved from dynamic time warping (DTW) in the 1980s to hidden Markov models (HMM) that are still used today [1]. The HMM is a graphical model expressing a relationship between hidden variables x_t (representing the underlying word sequence) and observed variables y_t (representing the acoustic observations). The regular structure of connections in this graph (Markov property) is exploited to allow approximate inference in linear time (Viterbi algorithm), making statistical speech recognition tractable.

A. HMM Framework

Figure 1 shows the high-level architecture of a speech recognition system. Training components are implemented in software and decoding components are implemented on an IC. The decoder includes a front-end which transforms audio (typically 16-bit data at 16 kHz) into the feature representation used by the models, and a search component which generates hypotheses about the underlying speech production process.

The HMM framework requires modeling the dependencies between variables, specifically the transition model $p(x_{t+1}|x_t)$ and the emission model $p(y_t|x_t)$. The transition model incorporates information about the language, including its vocabulary and grammatical constraints; the hidden states x_t are discrete variables. The emission model describes how observations y_t vary depending on the unobserved state x_t; in speech recognition, this is called the acoustic model. These observations of the speech signal are influenced by the speaker's vocal tract, the microphone, and the acoustic environment.

Properly accounting for multiple levels of knowledge (context, pronunciation, and grammar) compounded the complexity of early HMM decoders. The weighted finite-state transducer (WFST) is a state machine representation that allows each component to be described separately [2], [3] and then composed into a single WFST encompassing the entire speech process [4]. All possible transitions between states are expressed as weighted, labeled arcs. A WFST-based decoder can

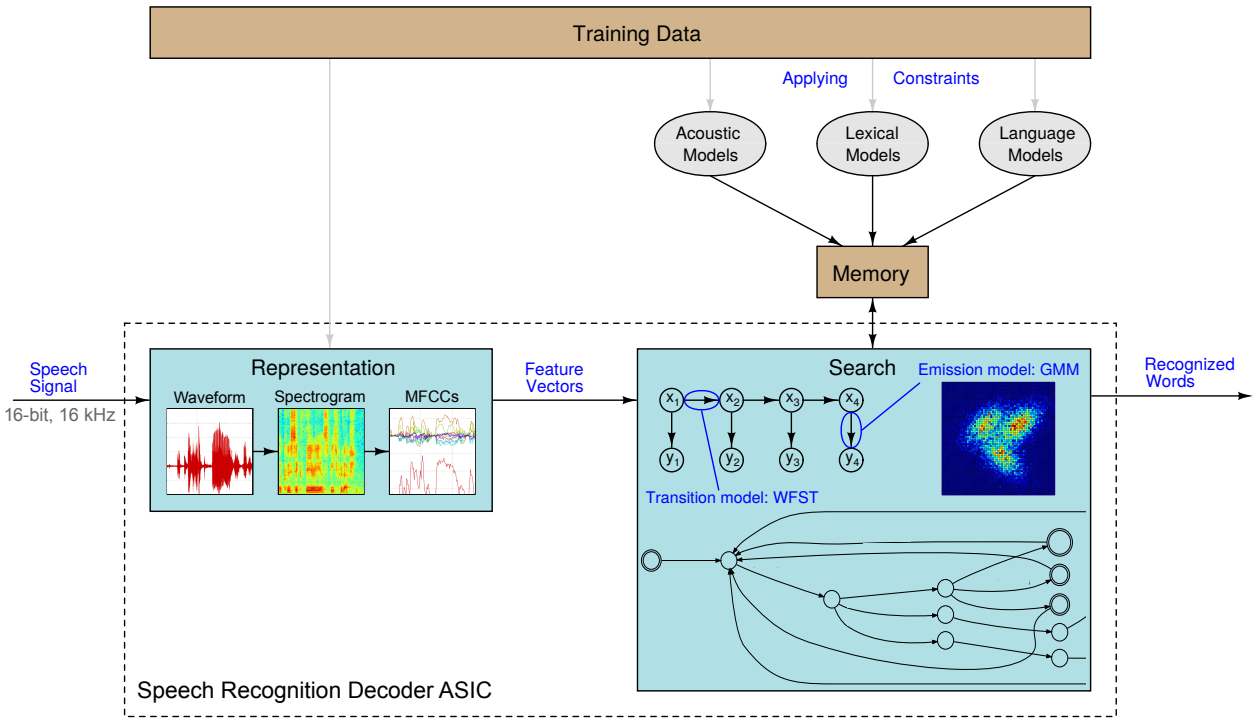


Fig. 1. High-level overview of speech recognition system. The speech recognition chip includes only decoding components (representation and search). Software tools are used to train an HMM incorporating WFST and GMM models, which are stored in an external memory.

complete tasks of varying complexity in different domains and languages by substituting the appropriate model parameters.

We use the Viterbi search algorithm, which maintains a list of active states at each time step. The forward update approximates the likelihood of all reachable states at the next time step:

$$p(x_{t+1}) = \sum_{x_t} p(x_t)\, p(x_{t+1}|x_t)\, p(y_{t+1}|x_{t+1})$$
$$\approx \max_{x_t} p(x_t)\, p(x_{t+1}|x_t)\, p(y_{t+1}|x_{t+1})$$

The existing state likelihood $p(x_t)$ is retrieved from memory. The transition probability $p(x_{t+1}|x_t)$ is the weight of the WFST arc[s] leading from state x_t to state x_{t+1}. The emission probability $p(y_{t+1}|x_{t+1})$ is an acoustic likelihood specified by the acoustic model, typically a Gaussian mixture (GMM). Rather than using a separate GMM for each state in the WFST (of which there are usually millions), the states are usually grouped into clusters (typically 1,000 to 10,000) by the similarity of their sounds; all states in each cluster share one GMM probability density [5]. Evaluating these probabilities takes up the bulk of power and I/O for the chip and is the main focus of our architectural enhancements.

B. Related Work

WFSTs are now commonplace in software speech decoders, including commercial systems [6]. Perhaps due to long development cycles, most of the hardware implementations realized so far have used non-WFST software decoders as a starting point.

ASICs for HMM-based speech recognition were reported as early as 1991 [7], followed by power-conscious implementations intended for portable devices [8]. The "In Silico Vox" project [9] created sophisticated hardware ports of Sphinx-3, a well-established software framework. Between 2006 and 2010 this project developed several hardware decoders in both single-FPGA and multi-FPGA implementations [10], [11], [12], achieving 10x faster than real-time performance using a Wall Street Journal (WSJ) dataset.

Digital circuits using WFSTs were presented in 2008 [13], contributing an important observation that pre-processing the WFST could eliminate the need to consider unlabeled arcs recursively during each frame. Taking advantage of external DRAM memory with 3.2 GB/s bandwidth, [14] illustrated how to split the decoder's active state list between internal and external memory to achieve better accuracy with limited internal SRAM.

More recent efforts have expanded decoder capabilities for general-purpose transcription applications with a 60,000 word (or larger) vocabulary. For example, [15] proposed an architecture for achieving much higher throughput (127x faster than real-time) using an ASIC. Another effort [16], [17], [18] bridged the gap between high-performance and low-power applications, applying a series of optimizations to a Japanese-language system called Julius in order to obtain just 54 mW of power consumption and 82 MB/s of memory bandwidth during real-time decoding.

This paper presents a working ASIC providing real-time audio-to-text conversion on a single chip, with 6 mW core power consumption on a 5,000-word speech decoding task.

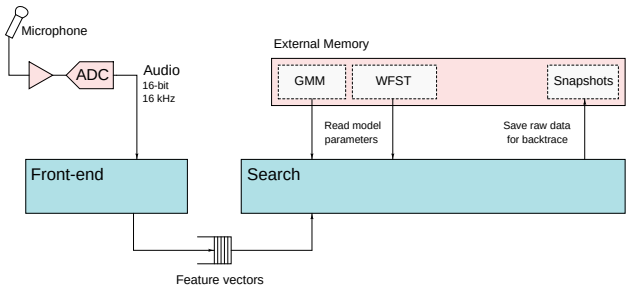


Fig. 2. High-level architecture of speech decoder.

III. Base Architecture

The high-level architecture of a hardware speech decoder is shown in figure 2. First, a front-end circuit extracts feature vectors from the audio stream. These feature vectors are treated as observations by a Viterbi search circuit, which reads model parameters from an external memory in order to update its active state list. The feature vectors are buffered in a FIFO so that the search module can accommodate varying workloads at a constant clock frequency. At the end of an utterance, the search circuit constructs a backtrace of the most likely state sequence. In a hybrid hardware/software system, multiple backtraces could be rescored using a more detailed language model to improve accuracy. Control logic translates the state sequence into words and handles communications with a host device as well as the external memory.

There are two clock domains in this system. An external clock between 20–100 MHz drives the decoder logic, and is divided by 16 internally to drive the less demanding front-end logic (as a simple energy-saving mechanism).

Our system was designed to accommodate publicly available WFST [19] and GMM [20] models trained on a *Wall Street Journal* (WSJ; Nov. 1992) data set [21], using a bigram language model with a 5,000 word vocabulary. The WFST has 2.9M states and 9.2M arcs, and the GMM has 10.2M parameters (4,002 mixtures in 39 dimensions, with 32–64 components per mixture). Both the WFST and GMM are programmable and reside in an external memory. Larger vocabularies (up to 65,535 words) could also be used, with language model pruning to obtain the desired WFST size. In this work, both the WFST and GMM models are speaker-independent.

We now describe the major components of the architecture (front-end and search) in detail.

A. Front-end (feature extraction)

Speech audio signals are sparse in the sense that time-domain sampling captures redundant information. We use mel-frequency cepstral coefficients (MFCCs) to concisely represent relevant characteristics of the signal. The MFCCs' low dimensionality simplifies classification, and channel effects (convolution) are additive in the cepstral domain [22]. Several standalone feature extraction devices have been reported in the literature, including DSPs, FPGAs, fixed-function digital circuits [23], and low-power analog front-ends [24]. The front-end could be used to provide features to an external (e.g. cloud-based) decoder or to the on-chip decoder.

MFCCs are derived from an audio timeseries via a chain of conventional signal processing blocks including an FFT, mel-scale bandpass filter bank, and DCT. Our front-end, shown in figure 3, emulates the behavior of the commonly used `HCopy` utility from HTK [5]. The audio is split into a series of overlapping frames 25 ms long with a 10 ms pitch. In addition to computing 12 cepstral coefficients and the log power for each frame, we extract first- and second-order time differences and concatenate them into a 39-dimensional feature vector at 16-bit resolution. Cepstral mean normalization is approximated by subtracting a 10-second moving average from the feature vectors.

Architectural optimizations shown at right in figure 3 reduced the front-end's circuit area and power consumption. The FFT exploits the real-valued nature of the input signal to operate on half as many points [25], and the bandpass filter bank employs two multipliers which are reused across the 26 bands. The circuit has an area of 51.8k gates (plus 107 kb of SRAM and 66 kb SROM) and requires a clock speed of at least 625 kHz to process a 16 kHz waveform in real-time.

B. Viterbi search

Viterbi search begins with an empty hypothesis for the utterance text and incorporates a stream of information from the feature vectors to develop a set of active hypotheses, represented by states in the HMM [26]. Each of these hypotheses is modeled using the WFST and GMM; if its likelihood is sufficiently high, it will be saved. Figure 4 shows the architecture of this subsystem.

The forward pass of Viterbi search propagates a set of hypotheses forward in time, from the active state list of frame t to that of frame $t+1$. We represent this operation as a pipeline with the following stages, corresponding to blocks in figure 4.

1) Hypothesis fetch: Each hypothesis is read from the active state list for the current frame (frame t). The active state list is implemented as a hash table in SRAM with open addressing and collisions resolved by chaining [27]. We store accepted states for the next frame in a separate SRAM and swap the two SRAMs using multiplexers at the end of each frame. In these hash tables, the key is a unique state ID (the memory address of the state in the WFST model) and the value is a structure describing the state. Hash table operations become slower as more states are stored (increasing the number of collisions), but this latency is masked by the much longer time required by WFST and GMM operations that access external memory. To reduce the need for memory accesses (which require additional logic and create pipeline stalls), arc labels and other metadata are carried through the pipeline along with state information.

2) Arc fetch: Each state ID is an index into the WFST model, from which we can retrieve the parameters of all outgoing arcs. These include all of the information (except a final score) that will be stored in the active state list if the hypothesis is accepted.

While this operation requires far less memory bandwidth than fetching GMM parameters, the arcs retrieved during a typical frame are distributed sparsely across a large memory

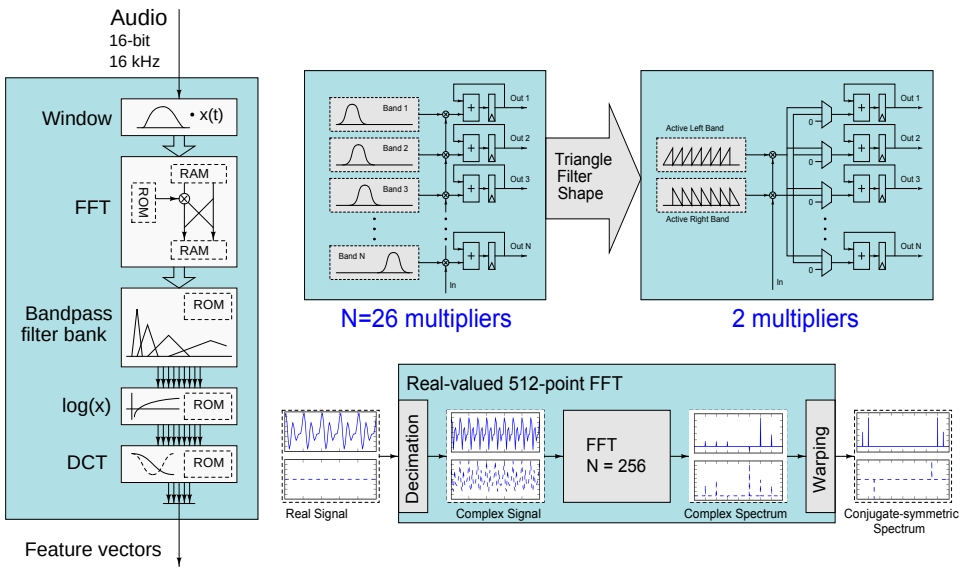


Fig. 3. Block diagram of MFCC frontend (left); filter bank and FFT optimizations applied to reduce circuit area and power consumption (right).

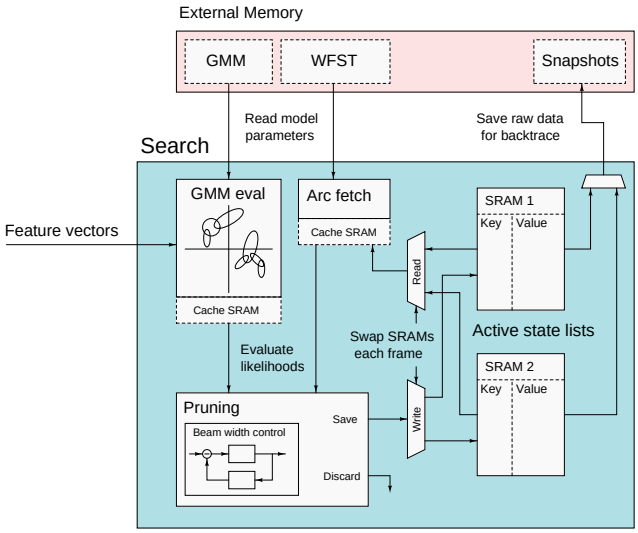


Fig. 4. Block diagram of Viterbi search module.

space: 193 MB for the 5,000 word WSJ model, versus 1 GB or larger for state-of-the-art models. The memory access pattern is sparse and depends on the hypotheses being searched.

3) GMM evaluation: The acoustic likelihood of the current feature vector y_t (given each hypothesis for the state x_t) is approximated using a GMM. GMMs used in speech recognition are typically limited to diagonal covariance matrices to reduce the number of parameters, using more mixture components to make up for the shortfall in modeling accuracy. Diagonal GMMs can be efficiently evaluated in the log domain, using a dot product for each component followed by a log-sum across components [28]:

$$p(y_t|x_t) = \sum_{c=1}^{C} \frac{w_c}{|\Sigma_c|^{\frac{1}{2}}(2\pi)^{\frac{K}{2}}} \exp\left[-\tfrac{1}{2}(y_t-\mu_c)^T\Sigma_c^{-1}(y_t-\mu_c)\right]$$
$$= \sum_{c=1}^{C} A_c \exp\left[\sum_{k=1}^{K} -\tfrac{1}{2}\frac{(y_{t,k}-\mu_{c,k})^2}{\sigma_{c,k}}\right]$$
$$\log p(y_t|x_t) = \log \sum_{c=1}^{C} \exp\left[g_c \sum_{k=1}^{K}(y_{t,k}-\mu_{c,k})^2 \tfrac{1}{\sigma_{c,k}}\right]$$

In this computation, mixture component c has parameters w_c (weight), μ_c (mean) and Σ_c (covariance); the dimensionality of y_t (and hence μ_c and Σ_c) is K. The values

$$g_c = \log A_c - \tfrac{1}{2} = \log \frac{w_c}{|\Sigma_c|^{\frac{1}{2}}(2\pi)^{\frac{K}{2}}} - \tfrac{1}{2}$$

are pre-computed in software and stored along with the mean variance parameters in external memory. The variances are stored as inverses $\frac{1}{\sigma_{c,k}}$ in order to use multiplication rather than division.

We implemented the log sum operator $\log\sum\exp$ by repeating a binary log add operation in an area-efficient manner as in [29]. Assuming that $a > b$ (the inputs can be swapped as necessary),

$$\log(e^a + e^b) = \log e^a(1 + \tfrac{e^b}{e^a})$$
$$= a + \log(1 + e^{b-a})$$

The scalar $\log(1+e^{b-a})$ operator, with a range of $[0, \log 2]$, can then be implemented using a lookup table without appreciable loss of accuracy.

The speed of GMM evaluation is limited by the rate at which parameters can be fetched from memory. Section IV-D presents a parameter compression approach to reduce the memory bandwidth demands at some expense in accuracy.

4) Pruning and storage: The state space of a Viterbi search grows exponentially unless a beam or histogram method is used to prune unlikely hypotheses from being stored and considered. A beam pruning stage tests each hypothesis against a threshold (relative to the highest likelihood encountered on the previous frame). Only hypotheses that pass the test are stored into the active state list for frame $t + 1$. Once all hypotheses for a given frame have been processed, a snapshot of the active state list is saved to the external memory.

C. Control

A control module accepts commands via a bytewise host interface (UART or USB) to actuate the front-end, search, and modeling components. These commands can program models into external memory, send audio data to the front-end, send feature data directly to the search system, and adjust configuration parameters. The chip is also instrumented with registers that collect a variety of statistics at the per-frame and per-utterance level. The statistics include the number of active states, number of WFST states expanded, and amounts of memory access required for WFST and GMM parameters. The host can be a portable or embedded device since its computational demands are trivial.

IV. Search Architecture Enhancements

Using a software prototyping environment, we determined that the original WFST/GMM architecture would require large on-chip memories and place great demands on the external memory, resulting in system power consumption well over 100 mW. An external DRAM configured to provide the necessary bandwidth of nearly 3 GB/s would dissipate most of this power [30].

Due to the large storage requirements of the WFST model, read-heavy workload, and low duty cycle of portable applications, it is desirable to use non-volatile memory such as NAND flash. Flash memory has limited bandwidth compared to SDRAM, and also has a significant out-of-order access penalty, on the order of 20–40 μs per 2–8 kB page [31]. The limitations of flash memory informed the architectural enhancements described in this section. These changes improve decoding performance with limited on-chip memory, reduce memory bandwidth (improving decoding speed and I/O power), and make memory access more sequential.

A. Beam width control

When beam pruning is used in Viterbi search, the number of active states fluctuates over time. There are more possible states in between words, when little acoustic information is available to constrain the branching of the language model.

Especially in large-vocabulary tasks, this fluctuation can quickly encounter on-chip memory limitations. If the beam width (pruning threshold) were constant, it would have to be set fairly low to avoid overflowing the active state list, and desirable hypotheses would be rejected. [14] demonstrates that the overflow could be stored in off-chip DRAM, but the bandwidth required is prohibitive in our application.

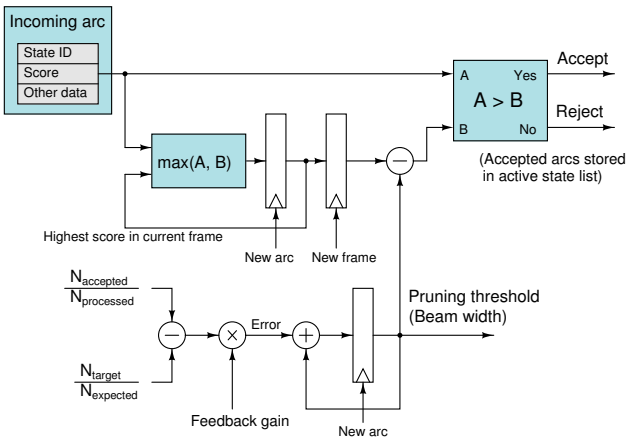


Fig. 5. Block diagram of feedback-based beam width controller.

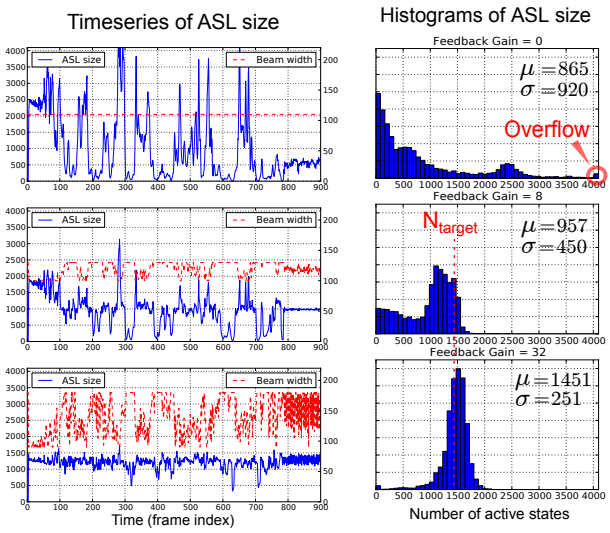


Fig. 6. Timeseries of beam width and active state list size (left) and histogram of active state list size (right) under varying levels of feedback.

By adopting a feedback scheme shown in figure 5, we are able to obtain a 13.0% word error rate (WER) with an active state list capacity of only 4096. Our data structure for a WFST arc includes counts of outgoing arcs from its destination state. By accumulating these counts, the decoder knows how many arcs (hypotheses) it expects to consider in the next frame; we call this N_{expected}. The feedback logic (shown in the lower half of the figure) adjusts the beam width by comparing the proportion $\frac{N_{\text{accepted}}}{N_{\text{processed}}}$ of arcs accepted so far to the proportion $\frac{N_{\text{target}}}{N_{\text{expected}}}$ needed for the desired active state list size of N_{target}. A gain parameter controls how quickly these changes take effect, and the beam width can also be clamped within a specified range. Registers store the highest score in the previous frame, from which the beam width is subtracted in order to test the incoming arc score and make an accept/reject decision.

Figure 6 shows how the number of active states varies over time with three different levels of beam width feedback. The best performance is obtained with a moderate amount of feed-

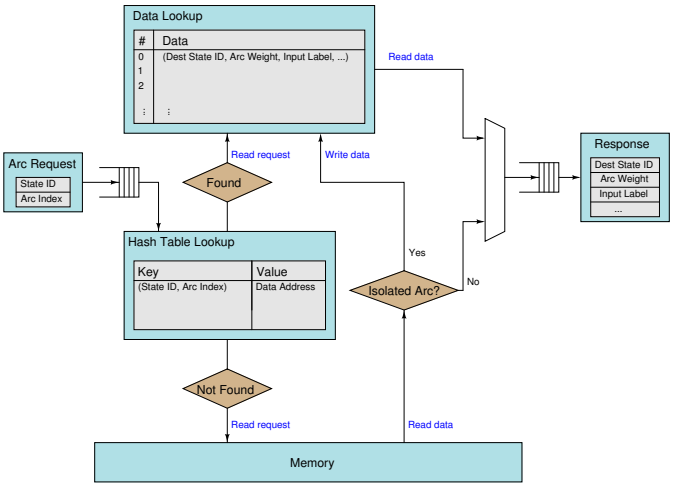


Fig. 7. Block diagram of WFST arc cache.

back (middle plots), which compresses but does not eliminate the natural variation occurring throughout the utterance.

B. WFST arc caching

Google has shown that the set of active states changes slowly during decoding, with an average of just 8,000 new states being explored per second in a voice search application [32]. If we could cache all WFST parameters fetched so far for an utterance, then arc fetch would require little memory bandwidth: we would only need to look up arcs reaching previously unseen states. There is not enough on-chip memory to do this within a reasonable core area, but we can still achieve bandwidth and memory latency reductions by prioritizing the most valuable WFST parameters to cache.

WFST arc fetch and caching operations are implemented as shown in figure 7. During each Viterbi update, we need to retrieve information about arcs leaving all of the active states. Some of these active states are located close together in the WFST model memory; we can encourage this behavior by ordering the WFST states according to a breadth-first search. We cache only "isolated" arcs that are not on the same page as others being evaluated in each frame, since these incur the most amortized latency. Furthermore, different states have different numbers of outgoing arcs; most have one or two, but some have thousands. To get the most out of our limited cache, we only consider states that have 0, 1, or 2 outgoing arcs.

The data table SRAM contains one entry for each arc. The pseudo-least-recently-used (PLRU) algorithm [33] was selected based on a comparison of hit rates between different cache eviction algorithms under typical use. The PLRU tree is fully associative (4096-way), and we use a hash table to map from keys to data addresses. This cache hit rate is 30.4% and WFST memory bandwidth is reduced from 8.01 MB/s to 5.73 MB/s; hit rates are limited by the large working set. The average number of distinct (non-sequential) 2 kB page accesses is reduced by a factor of 3, from 554 to 180 per frame, as shown in figure 8. This is sufficient for real-time

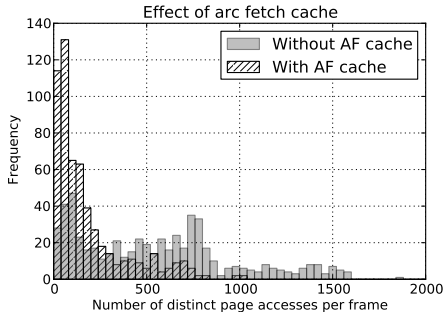


Fig. 8. Histogram of distinct page accesses per frame with and without WFST arc caching.

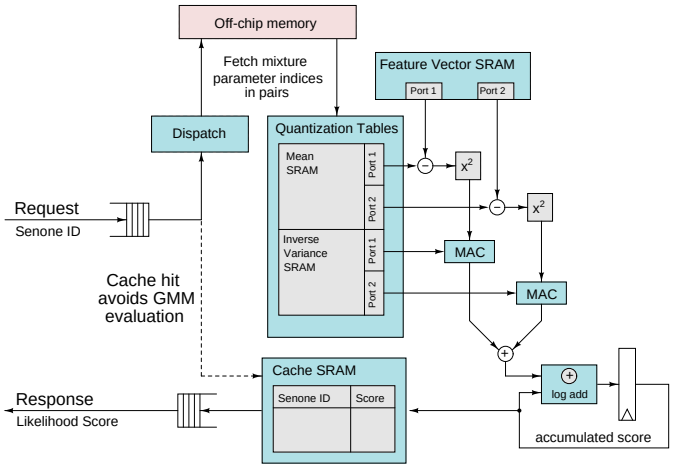


Fig. 9. Block diagram of GMM evaluator incorporating caching and parameter quantization.

operation when the WFST is read from SLC NAND flash memory. The total area of this block is 73.1k gates, plus 966 kb of SRAM.

C. GMM score caching

While the GMM acoustic model occupies less memory than the WFST, reads of GMM parameters are highly sequential and make up the bulk of data volume in any speech decoding task. We now present two architectural enhancements which together reduce the bandwidth needed for GMM parameters by a factor of 55.

A typical decoding workload in our tests required 2,900 GMM evaluations per frame, each with 32 mixture components. In a 39-dimensional feature space, using single-precision parameters, this would require 2.9 GB/s of memory bandwidth. However, the same GMM density is often evaluated multiple times per frame (for different arcs with the same input label). By caching each distinct GMM log-likelihood and adding the weight and previous likelihood of each candidate arc, we avoid repeated calculations (and parameter reads) with a hit rate of 86%.

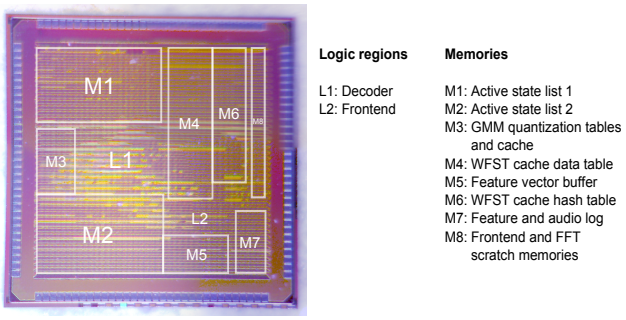


Fig. 10. Annotated die photo of speech decoding chip.

Specification	Value
Process	TSMC 65nm
Die size	2.5 x 2.5 mm
Package	128-pin LQFP
Logic gates	340k (NAND2 equiv.)
SRAM	2.4 Mb
Supply voltage	0.8 – 1.2 V
Power consumption	5 – 23 mW (core)
Clock frequency	40 – 110 MHz
# WFST states	2.9M
# GMMs	4k x 32 Gaussians

Fig. 11. Summary of test chip specifications.

D. GMM parameter quantization

Compressing the GMM model itself allows a further reduction in memory bandwidth into the sub-100 MB/s range that is feasible with flash memory. With an appropriate choice of quantizer one can eliminate many bits of resolution from the mean and variance values while incurring just 1–2% higher WER [34]. We use 32 quantization levels (5 bits) for the mean and 8 levels (3 bits) for the inverse variance; these sizes were identified by [34] as a good trade-off between accuracy and bandwidth. Our feature representation is not normalized or whitened, so the parameters have very different empirical distributions in different dimensions; thus, we need a different quantizer for each of the 39 dimensions.

Figure 9 demonstrates how caching and quantization are incorporated into the GMM evaluator. A dual-port SRAM totaling 41 kb is used to store the quantization tables at 16-bit resolution; these are loaded from the external memory at startup. The two ports of the SRAM feed two arithmetic units in parallel, such that evaluating one 39-dimensional mixture component requires 20 clock cycles. Parallel evaluation would increase throughput but require corresponding increases in the width of the external memory bus.

Parameter quantization results in a further 7.2x memory bandwidth reduction; typical GMM parameter bandwidth with both caching and quantization is 53.7 MB/s. The GMM evaluator has an area of 37.7k gates, plus 143 kb of SRAM (cache and quantization tables) and a 64 kb SROM for the log adder.

V. Results

In order to evaluate this architecture, we designed a digital IC using TSMC's 65 nm low power logic process. The die photo and specifications of this chip are shown in figures 10 and 11. The chip's generic host and memory interfaces are translated by an FPGA, and supporting board-level circuitry allows us to control and monitor its power supplies (figure 12). This system recognizes real-time audio input from an electret microphone; however, in the following tests we supplied audio samples digitally to ensure repeatability and consistency with the models.

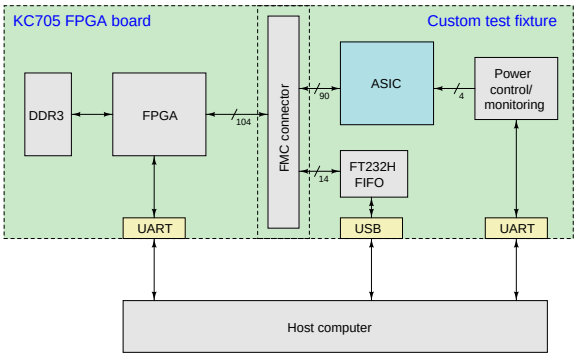
We verified correct operation of the entire system at a variety of operating voltages and clock frequencies. Figure 13 shows the maximum working frequency of the system as a function of the chip's core supply voltage. Scaling above 100 MHz and 0.9 V is limited by the FPGA logic and interfaces. Characteristics of the SRAMs (which use high-density 6T bit cells) prohibit operation at very low voltages. Alternative SRAM designs such as [35] could allow lower-voltage operation in exchange for increased chip area.

As with software decoders, the beam width can be used to trade between decoding speed and accuracy. Figure 14 shows the decoding time and WER across the 40-minute speaker-independent WSJ dataset with a variety of beam width settings; figure 15 shows how the workload of GMM and WFST model evaluations is affected by beam width. By employing voltage/frequency scaling, we can also trade between speed and power consumption, or accuracy and power consumption.

Real-time performance at the best word error rate of 13.0% was obtained at a clock speed of 50 MHz, allowing the supply voltage to be reduced to 0.85 V for both the logic and the embedded SRAMs. Under these conditions the average core power consumption (including both logic and SRAMs) was 6.0 mW while decoding with a 100% duty cycle. The front-end (feature extraction) circuit consumed an average of 110 μW. External clock gating results in an idle (leakage) power of 42 μW; clock gating can be activated between utterances to reduce power consumption at lower duty cycles.

Figures 16 and 17 show measurements collected from the chip while decoding a test utterance, "bids totaling five hundred twenty-five million dollars were submitted." The memory bandwidth required during each frame is shown in figure 16; there is significant variation due to the changing set of hypotheses over time. More than 85% of the bandwidth is used reading GMM parameters (with caching and quantization enabled). Memory writes due to state snapshots average 1–2 MB/s and are sequential within each utterance, allowing hundreds of hours of endurance when using a large circular buffer in flash memory. Memory bandwidth tracks overall decoding time very closely because nearly all on-chip computation involves parameters fetched from memory.

A timeseries of core logic and memory power consumption is shown in figure 17; note that the horizontal axis of this graph is clock time, rather than utterance time as in figure 16. The current draw is essentially constant except when the decoder has processed all available feature vectors and is waiting for further input. Memory is responsible for 77% of the core power consumption when operated at the same voltage as the logic. This is important because conventional high-density memory designs do not scale to lower voltages as well as standard-cell logic.



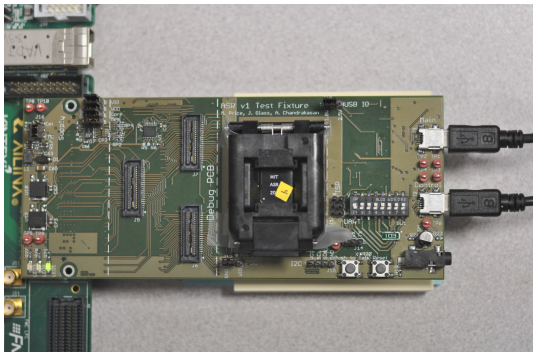


Fig. 12. Block diagram and photograph of test setup.

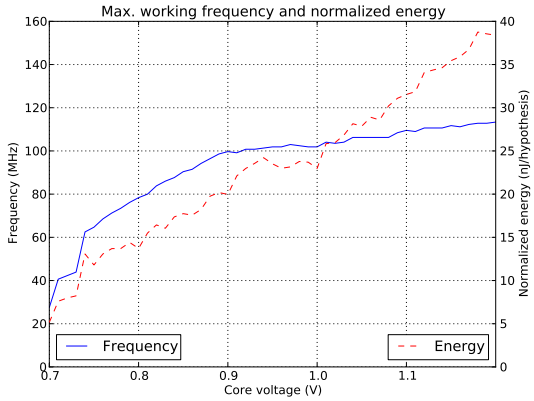


Fig. 13. Voltage/frequency scaling performance.

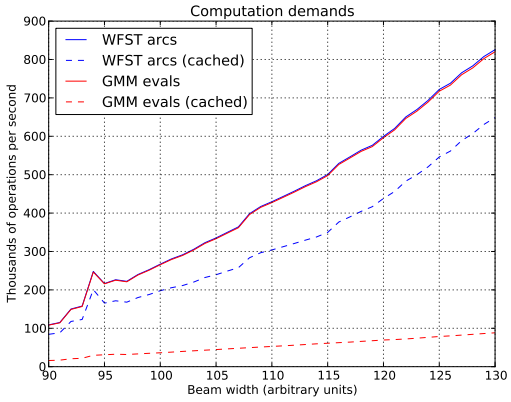


Fig. 15. Computational workload as a function of nominal beam width.

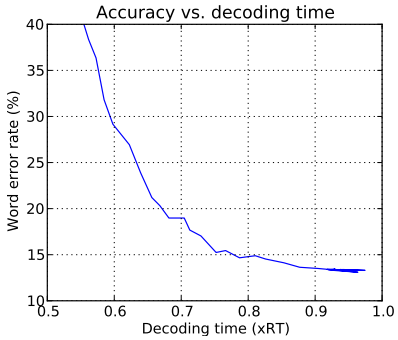


Fig. 14. Accuracy vs. decoding time tradeoff at 50 MHz.

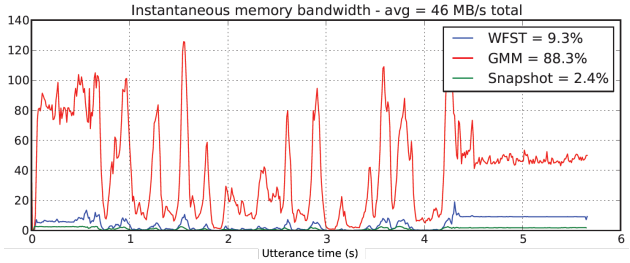


Fig. 16. Time-dependent breakdown of measured memory bandwidth requirements for an example utterance.

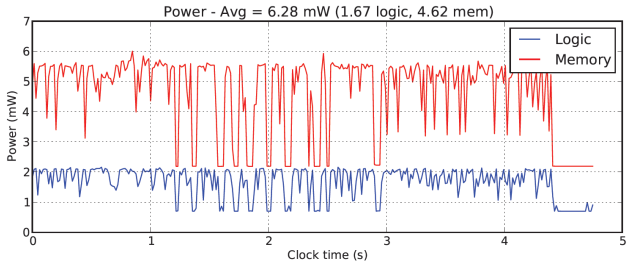


Fig. 17. Timeseries of core logic and memory current draw during an utterance. The timescale differs from figure 16 since decoding is faster than real-time.

Power consumption is closely correlated with the number of active states. Figure 18, a heat map of 13,530 decoding results using a variety of different utterances and configurations, shows the correlation between the energy per utterance and the number of active states stored during the utterance. At this operating point (0.9 V and 50 MHz), the coefficient of correlation is 81 nJ per stored active state, or 16 nJ per hypothesis (80% of hypotheses were pruned and not stored).

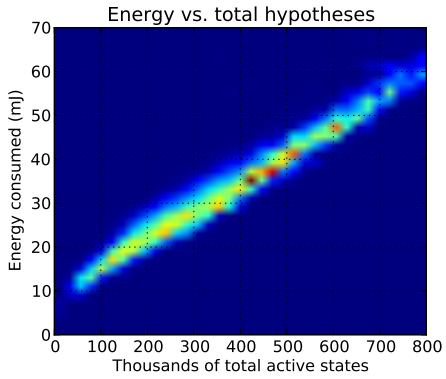


Fig. 18. Scatter plot of energy consumption and cumulative number of active states.

VI. Conclusion

Computational problems such as speech recognition admit a spectrum of system design approaches. While our research has focused on the extreme low-power results that can be obtained using ASICs, commercial systems may choose to use other approaches including FPGA, GPU and DSP architectures (or some combination, such as SoC hardware accelerators). The architectural changes described above can also be applied to other types of systems (including embedded software) if memory bandwidth reductions are desired.

This work has demonstrated that general-purpose speech recognition tasks can be performed in hardware with modest core power (6.0 mW) and memory bandwidth (58 MB/s) requirements. We presented an ASIC speech decoder that uses industry-standard WFST models and performs end-to-end recognition (audio in, text out). Compared to state-of-the-art hardware speech decoders, we achieved an order of magnitude lower core power consumption.

Acknowledgment

This work was funded by Quanta Computer (via the Qmulus Project) and an Irwin and Joan Jacobs fellowship. The authors would like to thank the TSMC University Shuttle Program for providing chip fabrication; Xilinx for providing FPGA boards; and Synopsys, Mentor Graphics, and Cadence for providing CAD tools.

References

[1] R. Lawrence, *Fundamentals Of Speech Recognition*. Pearson Education, 2008. [Online]. Available: http://books.google.com/books?id=4PQeabgwfX8C

[2] M. Mohri, "Weighted Finite-State Transducer Algorithms. An Overview," in *Formal Languages and Applications*. Springer, 2004, pp. 551–563.

[3] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook on Speech Processing and Speech Communication*, 2008.

[4] C. Allauzen, M. Mohri, M. Riley, and B. Roark, "A generalized construction of integrated speech recognition transducers," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, vol. 1, 2004, pp. I–761–4 vol.1.

[5] S. J. Young, "The HTK hidden Markov model toolkit: Design and philosophy," Cambridge University, Tech. Rep., 1994.

[6] M. Riley, A. Ljolje, D. Hindle, and F. Pereira, "The AT&T 60,000 word speech-to-text system," in *Fourth European Conference on Speech Communication and Technology*, 1995.

[7] A. Stolzle, S. Narayanaswamy, H. Murveit, J. Rabaey, and R. Brodersen, "Integrated circuits for a real-time large-vocabulary continuous speech recognition system," *Solid-State Circuits, IEEE Journal of*, vol. 26, no. 1, pp. 2 –11, jan 1991.

[8] A. Burstein, "Speech Recognition for Portable Multimedia Terminals," Ph.D. dissertation, University of California, Berkeley, CA, USA, 1996.

[9] E. C. Lin, K. Yu, R. A. Rutenbar, and T. Chen, "Moving Speech Recognition from Software to Silicon: The In Silico Vox Project," in *INTERSPEECH-2006*, 2006.

[10] ——, "A 1000-word vocabulary, speaker-independent, continuous live-mode speech recognizer implemented in a single FPGA," in *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, ser. FPGA '07. New York, NY, USA: ACM, 2007, pp. 60–68. [Online]. Available: http://doi.acm.org/10.1145/1216919.1216928

[11] E. C. Lin and R. A. Rutenbar, "A multi-fpga 10x-real-time high-speed search engine for a 5000-word vocabulary speech recognizer," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '09. New York, NY, USA: ACM, 2009, pp. 83–92. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508141

[12] P. J. Bourke and R. A. Rutenbar, "A High-Performance Hardware Speech Recognition System for Mobile Applications," in *Proceedings of Semiconductor Research Corporation TECHCON*, August 2005.

[13] J. Choi, K. You, and W. Sung, "An FPGA implementation of speech recognition with weighted finite state transducers," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, march 2010, pp. 1602 –1605.

[14] Y. kyu Choi, K. You, J. Choi, and W. Sung, "A Real-Time FPGA-Based 20,000-Word Speech Recognizer With Optimized DRAM Access," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, no. 8, pp. 2119 –2131, aug. 2010.

[15] J. Johnston and R. Rutenbar, "A High-Rate, Low-Power, ASIC Speech Decoder Using Finite State Transducers," in *Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on*, 2012, pp. 77–85.

[16] G. He, T. Sugahara, Y. Miyamoto, T. Fujinaga, H. Noguchi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A 40 nm 144 mW VLSI Processor for Real-Time 60-kWord Continuous Speech Recognition," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 8, pp. 1656–1666, 2012.

[17] G. He, T. Sugahara, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A 40-nm 168-mW 2.4x-real-time VLSI processor for 60-kWord continuous speech recognition," in *Custom Integrated Circuits Conference (CICC), 2012 IEEE*, 2012, pp. 1–4.

[18] G. He, Y. Miyamoto, K. Matsuda, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A 40-nm 54-mW 3x-real-time VLSI processor for 60-kWord continuous speech recognition," in *Signal Processing Systems (SiPS), 2013 IEEE Workshop on*, Oct 2013, pp. 147–152.

[19] J. R. Novak, N. Minematsu, and K. Hirose, "Open Source WFST tools for LVCSR cascade development," in *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, July 2011, pp. 65–73.

[20] K. Vertanen, "Baseline WSJ Acoustic Models for HTK and Sphinx: Training Recipes and Recognition Experiments," http://www.keithv.com/pub/baselinewsj, Cavendish Laboratory, University of Cambridge, Tech. Rep., 2006.

[21] D. B. Paul and J. M. Baker, "The design for the Wall Street Journal-based CSR corpus," in *Proceedings of the workshop on Speech and Natural Language*, ser. HLT '91. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 357–362. [Online]. Available: http://dx.doi.org/10.3115/1075527.1075614

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

[23] J.-C. Wang, J.-F. Wang, and Y.-S. Weng, "Chip design of mel frequency cepstral coefficients for speech recognition," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, vol. 6, 2000, pp. 3658–3661 vol.6.

[24] M. Baker, T.-T. Lu, C. Salthouse, J.-J. Sit, S. Zhak, and R. Sarpeshkar, "A 16-channel analog VLSI processor for bionic ears and speech-recognition front ends," in *Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003*, 2003, pp. 521–526.

[25] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 1, pp. 310–319, 2005.

[26] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260 –269, april 1967.

[27] G. Panneerselvam, G. Jullien, S. Bandyopadhyay, and W. Miller, "Reconfigurable systolic architectures for hashing," in *Databases, Parallel Architectures and Their Applications,. PARBASE-90, International Conference on*, Mar 1990, pp. 543–.

[28] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic likelihood computations," in *INTERSPEECH*, 2008, pp. 964–967.

[29] S. J. Melnikoff and S. F. Quigley, "Implementing log-add algorithm in hardware," *Electronics Letters*, vol. 39, no. 12, pp. 939–940, 2003.

[30] "DDR3 SDRAM System-Power Calculator," available online: http://www.micron.com/products/support/power-calc.

[31] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: anomalies, observations, and applications," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 24–33.

[32] C. Allauzen and M. Riley, "OpenFst: A General and Efficient Weighted Finite-State Transducer Library. Part II: Applications," in *2010 IEEE Workshop on Spoken Language Technology*, 2010, available online: http://www.openfst.org/twiki/pub/FST/FstSltTutorial/part2.pdf.

[33] Q. Wang, "WLRU Cache Replacement Algorithm," Ph.D. dissertation, University of Western Ontario, London, Ontario, Canada, 2006.

[34] I. L. Hetherington, "PocketSUMMIT: Small-Footprint Continuous Speech Recognition," in *INTERSPEECH-2007*, 2007, pp. 1465–1468.

[35] M. E. Sinangil, N. Verma, and A. P. Chandrakasan, "A reconfigurable 65nm SRAM achieving voltage scalability from 0.25–1.2 V and performance scalability from 20kHz–200MHz," in *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*. IEEE, 2008, pp. 282–285.