Flexible Low Power Probability Density Estimation Unit For Speech Recognition

Ullas Pazhayaveetil, Dhruba Chandra and Paul Franzon
Department of Electrical and Computer Engineering, North Carolina State University
Email: {ucpazhay,dchandr,paulf}@ncsu.edu

Abstract— **This paper describes the hardware architecture for a flexible probability density estimation unit to be used in a Large Vocabulary Speech Recognition System, and targeted for mobile platforms. The speech recognition system is based on Hidden Markov Models and consists of two computationally intensive parts – the probability density estimation using gaussian distributions, and the viterbi decoding. The power hungry nature of these computations prevents porting the application successfully to mobile devices. We have designed a flexible probability estimation unit that is both power efficient and meets real time requirements while being flexible enough to handle emerging speech recognition techniques. The flexible nature of the design allows it to utilize emerging power and computation reduction techniques (at the algorithm level) to achieve up to an 80% power reduction as compared to conventional designs.**

I. INTRODUCTION

By their very nature applications such as speech are likely to be most useful in mobile embedded systems. A fundamental problem that plagues these applications is that they require significantly more performance than current embedded processors can deliver. Most embedded and low-power processors, such as the Intel XScale, do not have the hardware resources and performance that would be necessary to support a full-featured speech recognizer without limiting its availability for other tasks. The energy consumption that accompanies the required performance level is often orders of magnitude beyond typical embedded power budgets [1]. The conflicting requirements of low power and high-speed real time recognition cannot be successfully realized using software solutions alone for mobile domains.

FPGA solutions [2] also fail to meet the increasing demands of Large Vocabulary Continuous Speech Recognition (LVCSR) neither completely meeting low power requirements nor real time recognition. Dedicated hardware in the form of an ASIC meets these but offers low degree of flexibility.

In this paper, we propose a VLSI architecture for part of a speech recognition system – the observation probability estimation unit – for HMM based speech recognition. Our architecture offers a high degree of flexibility and adaptability to emerging techniques, thus allowing power consumption reduction at the algorithm level.

II. SPEECH RECOGNITION THEORY

The Observation Probability Unit (OPU) evaluates the observation probability (or senone scores) of the input observation vector - the likelihood of each senone (state) model producing the observed input. The computation of the senone score is an intensive task typically making up between 40-80% of the total computation in a large vocabulary system. The senone score is given by $b_j(O_t)$, and represents the probability that state j emits observation feature vector O_t for time frame t. The senone score [3] is given by

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

where c_{jm} is the weight of the mixture component m in state j, and $N(O_t;\mu_{jm};\sigma_{jm})$ is the multidimensional (multivariate) Gaussian distribution with mean μ and covariance σ^2, and M is the total no. of mixtures per senone.

$$N(O_t; \mu_j; \sigma_j) = K.\exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(O_t[d]-\mu_j[d])^2}{\sigma_j^2[d]}\right) \quad (2)$$

where (3)

$$K = \left(\frac{1}{\sqrt[D]{2\pi}}\right)\frac{1}{\sqrt{\prod_{d=1}^{D}\sigma_j^2[d]}}$$

D is the dimensionality of the feature vector considered. This computation can be performed in the log domain to reduce the floating-point operations with negligible loss in performance and accuracy. C_{jm} is the new weight in the log domain. δ is the inverse of the covariance.

$$\log(b_j(O_t)) = \sum_{m=1}^{M} C_{jm}\sum_{d=1}^{D}\left((O_t[d]-\mu_j[d])^2 \times \delta_j[d]\right) \quad (4)$$

Several techniques have been proposed to speed up the computation of the senone scores. These techniques have been categorized into different layers [4]. We will briefly go through these layers and techniques. Frame-layer algorithms decide whether the senone score of the current frame should be computed or skipped. The simplest technique called Simple-Down Sampling (SDS) computes the frame scores only for every other frame[5]. Conditional Down Sampling (CDS) [5] is another technique where a frame is skipped if the

feature vector is quantized to a codeword, which is the same as that of the previous frame. Algorithms that decide which senone scores need to be computed in each computed frame are placed in the Gaussian mixture model (GMM) layer. One such representative technique is the Context-Independent (CI) GMM-based selection (CIGMMS) [6]. CI GMM scores (scores of monophone models instead of triphone models) are first computed. For those scores that are within a preset threshold, the detailed context dependent (CD) GMM (senone) scores are computed. The rest are backed- off by their corresponding CI GMM score. The different techniques used to decide which Gaussians dominate the senone score computation are categorized as Gaussian-Layer techniques. One such technique is the Sub-Vector Quantization Gaussian Selection (SVQGS)[7] where a rough model computation is first used to decide which Gaussians (in the multidimensional Gaussian distribution) need to be computed.

In the next section we propose the architecture for a dedicated OPU, which achieves real-time performance with low power consumption while incorporating latest algorithmic changes in speech and linguistic research making it more flexible.

III. ARCHITECTURE

A. Implementation

The system context for our OPU is shown in Figure 1. The extracted feature vectors are fed into the OPU through an arbiter unit. The arbiter also initializes the DRAM with the acoustic models for all senones, and obtains feedback from the Viterbi Decoder. It uses this information to provide the input to the OPU. The results or senone scores are stored in an SRAM, from which the Viterbi Decoder accesses them. Figure 2 shows the block diagram of the OPU itself.

The OPU is a highly pipelined IEEE 754 32-bit floating-point unit. The data path consists of an $(a-b)^2*c$ floating point unit (FPU1) followed by an adder that completes the inner loop of Equation (4). A fused multiply-add unit (labeled SWA in Figure 2) then performs the scale and weight adjustment. A log_add unit completes the outer loop. The basic building units (adders /multipliers) for this design have a 3-stage pipeline needing three buffers at both adder units to complete the calculations. The internal control unit has a coarse grain control over most of the arithmetic units, and multiplexers (all shaded boxes in Figure 2). The different mode settings provide coarse-grain control of different stages of the pipeline, as well as control over the interaction between the different units. This will be discussed in detail. From this point on, 'a','b' and 'c' will be used to refer to the inputs to FPU1.

During the initialization and setup phases, the parallel inputs to the unit, In1 and In2, initialize the internal LUT for the log_add module, and also setup the vector length or dimension (dim), number of mixture (mix), scales, weights, thresholds and other setup information. It should be noted that many of these parameters change during the course of the

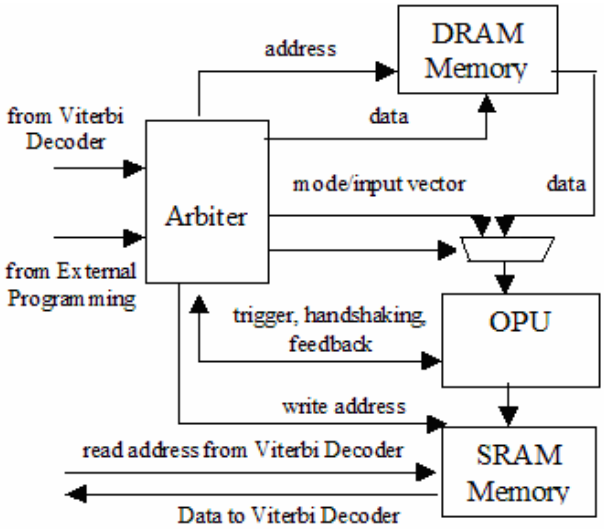

Figure 1. OPU Interfacing

computation, (for example, scale and weight values change for every GMM), and can be controlled separately and quickly. The SW register array stores the scale and weight values during the normal operation mode. The threshold array stores the different beam values, which the compare (x>y) unit uses to compare outputs at different stages of the pipeline against.

During the normal observation probability estimation process, the input feature vector, mean and variance are fed into the data path. The output of each of the completed internal loop (over the entire vector length) is a gaussian. This output is scaled and weighed and passed onto the log_add unit, which performs the outer loop calculations in the log domain (mixture of gaussians). The output (ScoreOut) of this is sent to the SRAM. A crude power save mode compares each of the individual gaussians as well as the output of the log_add module to one of four threshold values. If the observation probability falls below a particular threshold, further calculations for that particular senones are squashed and a preset 'Constant backoff' is sent to the output.

B. Adapting to the four layer techniques

1) Frame Layer

For the SDS [5] implementation, the arbiter simply feeds the GM unit every other frame, and in turn updates senones score value every other frame. The arbiter contains a counter that can be externally set and triggered. For the SDS, this counter is activated and its last bit is monitored to find out which frames are to be skipped.

For the CDS implementation, we first calculate the senone scores $b_j(O_t)$ for lookahead HMM models[5]. Typically these are small in number allowing this computation to be fast enough to be performed for every input frame t and every state j of the lookahead HMM models. Two sets of scores are maintained in the SRAM memory, one for the previous calculated frame, and one for the current frame. Scores of consecutive frame models (the Euclidean distance – $D(t)$) are compared and recorded.

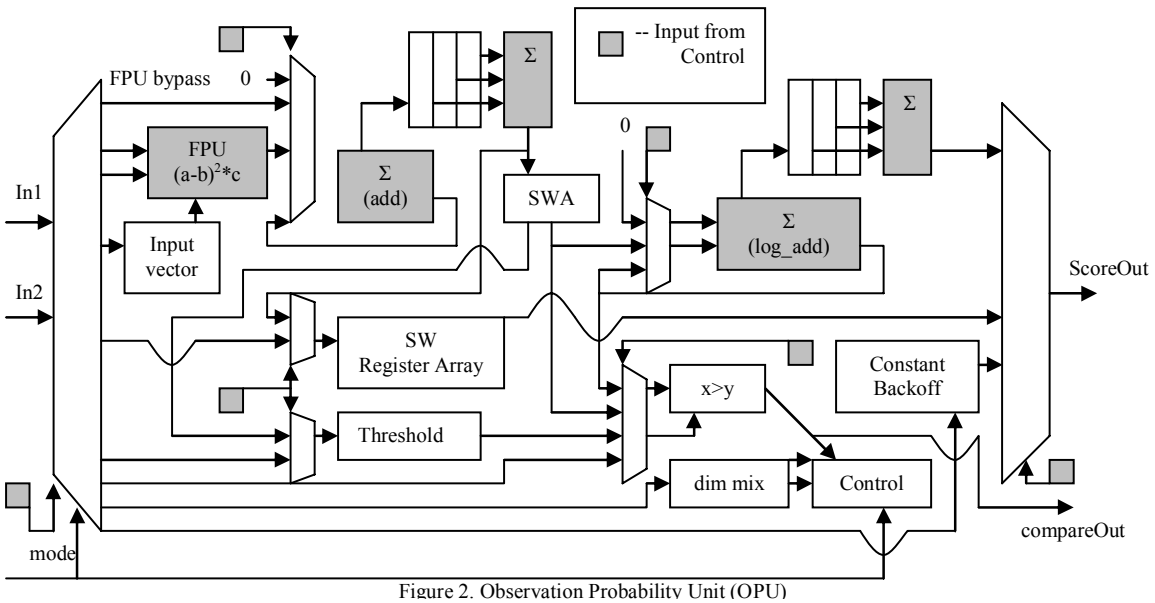

Figure 2. Observation Probability Unit (OPU)

$$D(t) = \sum_{j=0}^{J} \left(b_j(O_t) - b_j(O_{t-1})\right)^2 \quad (5)$$

$$D_{norm}(t) = \frac{D(t)}{\max_{0 \le i \le t} D(i)} \quad (6)$$

$$0.0 < D_{norm}(t) \le 0.3 \text{ : skip 2 frames}$$

$$0.3 < D_{norm}(t) \le 0.6 \text{ : skip 1 frame}$$

$$0.6 < D_{norm}(t) \le 1.0 \text{ : skip no frames} \quad (7)$$

The maximum score $D_{max} = \max_{(0<i<t)} D(i)$ is also recorded. To determine whether or not a signal is currently changing and in turn gives us information on how many frames to skip the following equations are used. For the *D(t)* calculations, 'dim' is first modified to fit the required vector length, and then $b_j(O_t)$ is fed to 'a', $b_j(O_{t-1})$ is fed to b, and '1' fed to c.

D(t) is temporarily stored in the SW register array. D_{max} is recorded by continually feeding the result of the first sumer unit to the compare unit and updating the threshold if the current output value is greater than the recorded D_{max} till now. In our implementation, instead of the normalization (Equation (6)), we scale the max score to obtain the thresholds ($0.3*D_{max}$, $0.6*D_{max}$). This is performed in 2 separate runs of the OPU, where 0.3 and 0.6 are fed into 'a', 0 into 'b', and D_{max} into 'c'. The outputs (for both values) from the first sumer unit as well as D_{max} are sent to three of the four threshold buffers. Finally the values of *D(t)* are compared to these values using the compare_unit(labeled 'x>y' in Figure 2). The output of the compare unit (compareOut) is sent to the arbiter to set its internal counter in turn setting how many frames to skip.

2) GMM layer

CI observation probabilities [6] are first calculated, and compared to a threshold value using the compare unit. The output 'compareOut' sets a bit in memory signaling whether the corresponding context dependent phones need to be computed or not. Else the CD scores are backed off by the CI score (ScoreOut).

3) Gaussian Layer

Codeword and cluster definitions are done offline. A pre-calculated threshold value (θ) is sent to the threshold register array. The output of the first adder unit identifies the codeword and neighborhood of the input vector [7].

$$\frac{1}{D} \sum_{d=1}^{D} \frac{\left(O_t[d] - \mu_j[d]\right)^2}{U_j[d]} > \theta \quad (8)$$

The output of the compare unit compareOut is sent to the arbiter and is used to select the codeword and its neighbors using a LUT. Now each GMM is made up of a reduced set of mixtures. Finally the mix parameter is varied for each senone and the observation probability of each one is calculated using the reduced set of mixture values.

IV. RESULTS

The HMM based OPU design was implemented in Verilog and synthesized using the Synopsys Design Analyzer tool in a .18m CMOS technology. The Synopsys PrimePower tool was used to obtain the power numbers. Feature vectors are extracted from the input speech waveforms using the Sphinx-3 front-end. An operational frequency of 50MHz was achieved. Three of these units were sufficient to support real-time speech recognition for about 6000 senones. The die size was about 2.168 mm^2.

The design was evaluated for real time performance and power consumption for six configurations – Baseline system, crude threshold check (CTC), SDS, CDS, CIGMMS and SVQGS. Figure 3 shows the real time performance of each of these techniques on our design. Real-time performance is measured as a fraction of the sampling frame rate (every 10ms). Real time performance is achieved if all required senone scores are computed before the next set of input observation vectors is available (10ms later). The frequency of operation for SDS was cut down to 25MHz so as to reduce power consumption, which explains its 1xRT performance. Figure 4 shows the power consumption for each of these techniques with our design.

The power consumption values reported are for a single OPU. We achieve a power consumption reduction of up to 48% over conventional designs using these techniques.

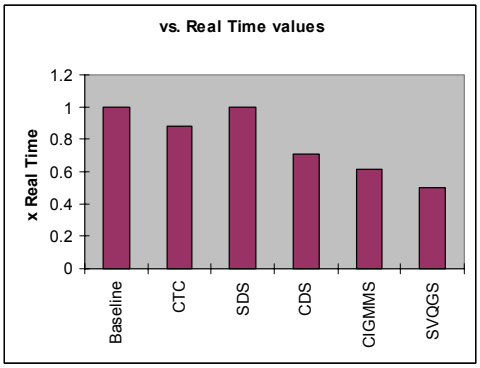

Figure 3. Performance (speed) vs. real-time

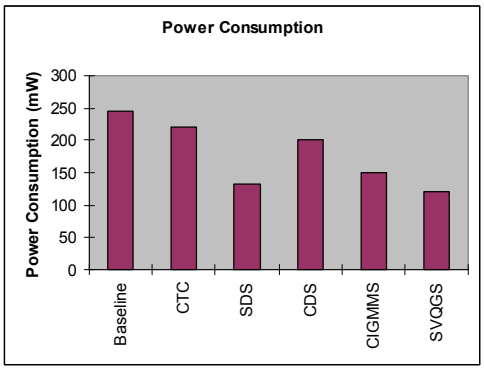

Figure 4. Power Consumption for a single OPU

V. DISCUSSION AND RELATED WORK

Several approaches have been taken towards finding solutions to the problems of speed and power consumption in speech recognition. Software solutions running on a desktop PC can barely achieve real-time high accuracy speech recognition, and completely consumes the resources of the PC if it does. This coupled with its power hungry nature prevents porting it successfully to mobile devices. The Sphinx3 baseline system runs 1.8x slower than real time on a 2.4 GHz Pentium4 system [8] and consume about 52.3 watts. A dedicated hardware accelerator has been proposed to speed up the Sphinx3 software implementation [8]. The design achieved real-time recognition and consumed about 1.8 watts of power. Hence, even with the worst case power performance of 3x245mW, we achieve about 60% power reduction over this design. Adapting our design to the new techniques (assuming the best case of 3x121mW) leads to about 80% power reduction over the previous design. Another design for a VLSI speech processor developed at the National Cheng Kung University, integrates a programmable core and specific recognition core to achieve high performance and flexibility. No power readings were reported. The use of reconfigurable logic and FPGA devices is another common approach [2]. The inherent reconfigurability of FPGAs provides a level of specialization while retaining significant generality. However the reconfiguration time is relatively long, and FPGAs have a significant disadvantage both in performance and power when compared to either ASIC or CPU logic functions.

While the power savings itself is a step towards porting our design to mobile domains, what is more important is perhaps the degree of flexibility that our design incorporates. With new techniques emerging continually in speech recognition, it is important that any hardware accelerator built will be able to incorporate these techniques at least to some extent, and take advantage of the savings they provide. Our design has been able to incorporate new techniques in speech recognition and use it to reduce power consumption at the algorithm level. The accuracy achieved with the hardware implementation of these techniques are consistent with the software implementations and hence have not been reported here.

VI. CONCLUSION

We have designed and implemented a flexible low-power VLSI architecture for observation probability estimation for a large vocabulary HMM speech recognition system. The design is capable of adapting to new and emerging techniques in speech recognition. It uses these new techniques to achieve power consumption reduction at the algorithm level. We have shown that the design achieves real-time recognition while reducing power consumption by up to 80% over comparable designs. This allows the application to be successfully ported to power constrained domains such as mobile devices.

REFERENCES

[1] Rajeev Krishna, R., Mahlke, S. and Austin, "Architectural Optimizations for Low-Power, Real-Time Speech Recognition". Proc. 2003 Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems(CASES '03), 2003, 220 -231.

[2] Melnikoff, S. Quigley, S.F. "Performing speech recognition on multiple parallel files using continuous hidden Markov models on an FPGA", Proc. IEEE International Conference on Field Programmable Technology (FPT 2002), 2002, pp.399-402.

[3] Young, S., "Large vocabulary continuous speech recognition: A review". IEEEWorkshop on Automatic Speech Recognition and Understanding, Snowbird, Utah, December 1995, 3-28.

[4] Chan, A., Mosur, R., Rudnicky, A., and Sherwani, J., "Four layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems". Intl. Conf. on Spoken Language Processing, 2004, 689-692.

[5] Woszczyan, M., "Fast Speaker Independent Large Vocabulary Continuous Speech Recognition". Universitat Karlsruhe; Institut fur Logik, Komplexitat and Deduktions system,1998.

[6] Lee, A., Kawahara, T., and Shikano, K., "Gaussian mixture selection using context-independent HMM" In IEEE ICASSP, 2001.

[7] Bocchieri, E., "Vector quantization for efficient computation of continuous density likelihoods". In ICASSP '93, Volume II, 1993, II 692-695.

[8] Mathew, B., Davis, A. and Fang, Z., "A Low-Power Accelerator for the SPHINX 3 Speech Recognition System".