VLSI FOR 5000-WORD CONTINUOUS SPEECH RECOGNITION

Young-kyu Choi
LG Electronics
ykchoi@dsp.snu.ac.kr

Kisun You, Jungwook Choi, and Wonyong Sung
School of Electrical Engineering, Seoul National University
{ksyou,jwchoi}@dsp.snu.ac.kr, wysung@snu.ac.kr

ABSTRACT

We have developed a VLSI chip for 5,000 word speaker-independent continuous speech recognition. This chip employs a context-dependent HMM (hidden Markov model) based speech recognition algorithm, and contains emission probability and Viterbi beam search pipelined hardware units. The feature vector for speech recognition is computed using a host processor in software in order to adopt various enhancement algorithms. The amount of internal SRAM size is minimized by moving data out to the external DRAM, and a custom DRAM controller module is designed to efficiently read and write consecutive data. The experimental result shows that the implemented system has a real-time factor of 0.77 and 0.55 using SDRAM and DDR SDRAM, respectively.

Index Terms — speech recognition, very-large-scale integration, LVCSR

1. INTRODUCTION

A VLSI chip for large vocabulary continuous speech recognition (LVCSR) can find many applications, but the implementation has not been possible because this task requires much computation and data access. Most of the current LVCSR systems use high-end personal computers, which are not only expensive but also consume much power. By taking a hardware-based approach, we can perform speech recognition with less power and smaller chip area when compared to the software-based approach.

There have been several hardware-based speech recognizers. However, some of them have been only partially implemented in hardware [1][2], and some others are not adequate for large vocabulary continuous speech recognition system [3]. The FPGA-based systems depicted in [4] and [5] can perform complete speech recognition, however they are less efficient than VLSI chip based implementations in terms of power, area, and performance.

This paper describes a 5000 word continuous speech recognizer implemented on a VLSI chip. We have employed fine-grain pipelined architecture and memory bandwidth optimization techniques to maximize the performance with a minimum chip size. Moreover, we have developed a custom DRAM controller to efficiently read and write consecutive stream of data, which contributes much for reducing the internal SRAM size. Also, we have analyzed the tradeoff between the required SRAM and the performance.

This paper is organized as follows. The speech recognition algorithm used in this implementation is explained in Section 2. Section 3 describes the architecture of the implemented system. In Section 4, the memory usage is analyzed to show the trade-off between the SRAM size and the performance. Section 5 shows the experimental results, and concluding remarks are made in Section 6.

2. SPEECH RECOGNITION SYSTEM OVERVIEW

This chip employs a context-dependent HMM-based continuous speech recognition algorithm, which is the most widely used one for LVCSR [6]. The recognizer consists of three functional blocks: feature extraction, emission probability computation, and Viterbi beam search.

The recognizer computes 39-dimensional MFCC (Mel-Frequency Cepstral Coefficient) from 30ms length of input speech waveform at every 10ms.

With the obtained MFCC feature vector, the emission probability, which represents the probability of the feature vector being generated from the corresponding HMM state, is computed. We utilize the emission probability that is approximated by the maximum Gaussian probability [7]:

$$\log(b(O_t;s)) = \max_m \{C_m - \frac{1}{2}\sum_{k=1}^{K}\frac{(x_k-\mu_{mk})^2}{\sigma_{mk}^2}\}, \qquad (1)$$

where K is the feature dimension, C_m is a Gaussian constant, and μ_{mk} and σ_{mk} are means and standard deviation of the Gaussian. Since many HMM states need to be compared, this needs not only many multiplications and additions, but also much memory access.

In the Viterbi beam search, the best state sequence up to the current frame is sought based on the emission probability of all the active HMM states. The time synchronous Viterbi beam search [8] is employed, which can be divided into two parts: one for the intra-word transition and the other for the inter-word transition.

The dynamic programming (DP) recursion for the intra-word transitions is conducted as follows:

$$\psi_t(s_j;w) = \max_i\{\psi_{t-1}(s_i;w) + \log(a_{ij})\} + \log(b(O_t;s_j,w)), \qquad (2)$$

where a_{ij} is the transition probability from the state i to j, and $\log(b(O_t;s_j,w))$ is the emission probability for the state

j of the word w in the time frame t. $\psi_t(s_j; w)$ is the accumulated likelihood of the most likely state sequence reaching the state j of the word w at time t. During the dynamic programming, any state that has a smaller accumulated likelihood value than the threshold is discarded by the beam pruning.

The inter-word transition is performed based on Eq. (3), where the accumulated likelihood of the last state of each word is propagated to other words. The bigram language model gives the constraint to the inter-word transition.

$$\psi_t(s_0; w) = \max_v \{\log(p(w \mid v)) + \psi_t(s_f; v)\} \quad (3)$$

where $p(w \mid v)$ is the bigram language model probability from word v to word w, s_f indicates the final state, and s_0 is the pseudo initial state.

After detecting silence, backtracking is performed to obtain the best state sequence, which shows a recognized speech sentence.

3. ARCHITECTURE

3.1. Overall Architecture

As shown in Fig. 1, the system consists of a host processor, the speech recognition chip, and an external DRAM.

For the host processor, the Colibri XScale PXA270 module is used, which contains the Intel PXA270 processor running at 520 MHz, 64MB of SDRAM, and an audio I/O [9]. The host processor performs feature extraction in software. Note that the feature extraction does not demand much computation when compared to emission probability or Viterbi beam search modules. However, this part is important because speech recognition accuracy can be increased significantly by employing noise cancelling, speaker adaptation, or beam-forming algorithms.

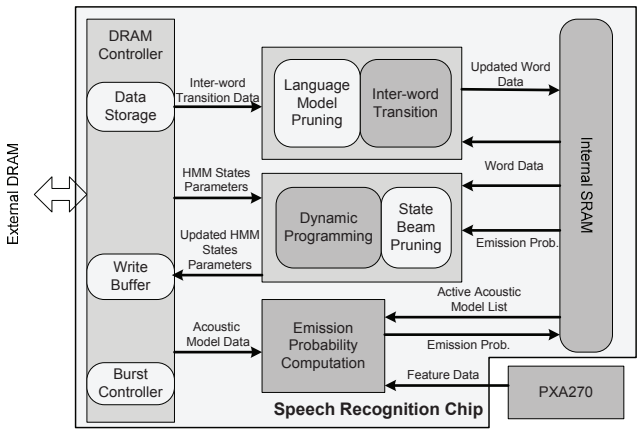

Figure 1. Overall speech recognition system architecture

As for the external DRAM, either Synchronous DRAM (SDRAM) or Double Data Rate (DDR) SDRAM can be used. The data bit-width is 16, the data capacity is 64MB, and the DRAM clock is 133MHz.

The implemented speech recognition chip is composed of four parts: emission probability computation, dynamic programming & beam pruning, language model pruning & inter-word transition, and a DRAM controller. The remainder of this section describes each part in detail.

3.2. Emission Probability Computation Unit

This unit calculates the likelihood $\log(b(O_t; s))$ of the HMM state by comparing the Gaussian parameters with the feature data.

To reduce the DRAM memory bandwidth required for reading the feature data, the emission probability of four speech frames is computed at a time [5]. Because of the correlation in adjacent speech frames, the Gaussian parameters to be loaded for the neighboring four frames overlap very much. This technique, however, introduces a small delay of 40 msec. At the end of every four frames, the emission probability for four frames is computed at a time, and the result is stored in the SRAM along with a list of loaded Gaussian parameters. For the correctness, the emission probability for the newly activated HMM states in the middle of four frames is computed instantly. After the optimization, we reduced 66.8% of DRAM access without losing any recognition accuracy. Additional SRAM of 23.4KB is needed to store the emission probability.

3.3. Dynamic Programming & Beam Pruning Unit

Dynamic programming recursion shown in Eq. (2) is performed to evaluate the accumulated likelihood of HMM states. Since the data structure representing the HMM states requires 699KB of memory, it needs to be stored in the external DRAM. In order to mitigate the effect of increased latency for accessing external DRAM, fine-grain pipelined architecture is adopted [5]. The architecture can process one HMM state every clock cycle by pipelining, and the computation time overlaps with the data access time.

In the first stage of the pipeline, the likelihood value of the current state is compared with that of the previous state. The higher likelihood between the two is selected and stored in a buffer for the next cycle. Also, the request signal for the emission probability that corresponds to the current HMM state is sent to the SRAM in this stage.

In the second stage, the emission probability requested in the first stage becomes available, and it is added to the stored likelihood from the previous stage. Then, this accumulated value is pruned away if it is smaller than the acoustic beam threshold value.

In the third stage, the updated likelihood from the second stage is written to the DRAM write buffer. The data in the write buffer are written to the DRAM in a burst mode when the write buffer becomes full. Also, the state status is up-

dated in this stage. If the accumulated likelihood in the second stage is bigger than the beam threshold, it stays active, otherwise it becomes inactive.

3.4. Language Model Pruning & Inter-word Transition

After updating all the HMM states, the inter-word transition is processed. This unit also adopts the pipelined architecture to hide DRAM access latency [5]. The language model probability and the next word address are read from the DRAM.

In the first stage, the language model probability is added to the likelihood of the last state. Then the result is compared with the language model threshold. If it is bigger than the threshold, the likelihood value of the first state of the next word is requested from the SRAM. Also, the transition probability is stored in a buffer for the second stage.

In the second stage, the likelihood value of the first state of the next word becomes available. This value is compared with the inter-word transition probability from the first stage. If the value is smaller than the incoming transition probability, the SRAM is updated with the incoming value.

To reduce the memory bandwidth of the inter-word transition, two-stage language model pruning is used [5]. In the first stage, the best transition language model probability is compared with the language model pruning threshold. If the best transition probability does not exceed the threshold value, it means that the rest of transition probability would not exceed it, either. Therefore, the rest of the language model probability and its corresponding next node address are not fetched from the DRAM. If the best transition probability is bigger than the threshold value, the transition probability is read from the DRAM and compared with the pruning threshold in the second stage. This technique reduces the DRAM bandwidth of inter-word transition part by 46.0%.

3.5. DRAM Controller

Since the performance of the speech recognizer heavily depends on the memory access time of external DRAM, we increased the memory bandwidth by designing a custom memory controller that supports up to 32 burst memory access. Since DRAM only support burst access of length 8, we issued 4 DRAM access request consecutively without closing the opened row of a bank. This scheme reduces the delay between the memory requests, because we need to open and close a row only once during the burst access operation.

Note that the data must reside in the same row to implement this policy. In order to insure that the recognizer requests data that is placed in the same row as much as possible, we stored all DRAM data in a consecutive manner.

4. MEMORY USAGE

Although the internal SRAM provides high performance, it consumes much chip area. Therefore, we analyze the tradeoff between the area and the performance by moving some data out to the external DRAM. In order to implement the pipelined architecture explained in Section 3, the data described in Table I must be stored in the internal SRAM.

The data not listed in Table I can be moved to the DRAM for saving the chip area. Table II describes the trade-off between the performance and the internal SRAM size. The performance is measured using the DDR SDRAM. The result shows that storing only the minimum amount of required data in the SRAM reduces the performance by 6.8% (0.517RT -> 0.552RT), while lowering the SRAM usage by 59.6% (149KB -> 60.2KB). In order to reduce the area, we have chosen to use the minimum amount of SRAM in the final implementation.

Table I. SRAM usage

Description	Size (KB)	Single /Dual port
List of active Gaussian parameters	0.73	Dual
Emission probability storage	23.4	Single
List of active words	0.61	Dual
Accumulated likelihood of active words	9.77	Dual
Back-trace info. storage (time)	9.77	Single
Back-trace info. storage (prev word name)	9.77	Single
List of words that perform inter-word trans	1.22	Dual
Likelihood values to the back-off node	4.88	Single
Total	60.2	

Table II. Tradeoff between performance & SRAM size

Data moved to DRAM	Size(KB)	Real Time Factor
(None)	149	0.517
Dyn. Prog. Only	100	0.545
Inter-word Trans. Only	109	0.525
Dyn.Prog.+Inter-word Trans	60.2	0.552

5. EXPERIMENTAL RESULTS

5.1. Experimental Setup

The acoustic model of the recognition system was trained by HTK, an open-source speech recognition toolkit [10]. The speaker independent training data in Wall Street Journal 1 corpus is used. The acoustic model data consists of 3,000 shared HMM states. Each state has a mixture of 16 Gaussian distributions. For the evaluation of the system, we used the Wall Street Journal 5k-word continuous speech recognition task. The test set consists of 330 sentences spoken by several speakers. The language weight is set to 16.0

and the word error rate of the implemented system is 9.36%. No special preprocessing algorithm is used.

5.2. Execution Time

For execution time measurement, we have constructed the cycle-accurate models of the SRAM, the external DRAM, and the processor. Table III shows the performance comparison between the developed system with two DRAM versions and the FPGA based implementation result in [5]. Note that the developed system employs a 16bit wide 133MHz SDRAM or DDR SDRAM while the FPGA based system in [5] uses a 32bit wide 100MHz DDR SDRAM.

The comparison result shows that the developed SDRAM version is 1.17 times slower than [5], and the DDR SDRAM version is 1.20 times faster than [5] in terms of the execution speed. This is due to several reasons. First, we should consider that the FPGA based system in [5] uses a 32bit wide 100MHz DDR SDRAM and thus has approximately 3 times and 1.5 (=2.0/1.3) times more memory bandwidth than the SDRAM and DDR SDRAM versions, respectively. Second, by minimizing the SRAM usage, there was a performance degradation of 6.6% and 6.8%, respectively. The FPGA based system in [5] consumes the internal memory size of approximately 274 KB. The improved performance is mainly due to the new DRAM controller that supports the open-row policy.

Table III. Comparison of execution time

	FPGA[5] (Mcycles/s)	SDRAM (Mcycles/s)	DDR (Mcycles/s)
Emission Prob.	26.9	45.2	31.1
Dynamic Prog.	30.0	40.8	31.0
Interword Trans	9.1	16.4	11.4
Total	66.0	102.4	73.5
Real Time Fac.	0.66	0.77	0.55

Table IV Area estimation after synthesis

	SDRAM(μm^2)	DDR(μm^2)
Area of Comb. Logic	1,287,213	1,686,295
Area of Non-Comb. Logic	6,772,583	7,099,528
Total Cell Area	8,059,796	8,785,823

5.3. Area Estimation

Synopsys design compiler A-2007.12-SP5 with the Chartered 0.18μm process was used to synthesize the design. Table IV shows the area estimation after synthesis. The area of the DDR version is 9% larger than that of the SDRAM version. After the logic synthesis, placement & routing was performed using Synopsys Astro tool.

6. CONCLUDING REMARKS

We have implemented a VLSI chip for 5000 word continuous speech recognition. We try to reduce the internal SRAM memory size, and hide the latency of external DRAM memory access by employing fine-grain pipelined architecture and a custom DRAM controller. Two hardware versions, one uses a 16bit SDRAM and the other employs a 16bit DDR SDRAM, have been developed, and achieve the execution speed of 1.30 (with SDRAM) and 1.82 (with DDR SDRAM) times faster than real-time.

7. ACKNOWLEDGEMENTS

This work was supported in part by the Brain Korea 21 Project and ETRI SoC Industry Promotion Center Human Resource Development Project for IT SoC Architect.

8. REFERENCES

[1] U. Pazhayaveetil, D. Chandra, and P. Franzon, "Flexible low power probability density estimation unit for speech recognition," *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 1117–1120, 2007.

[2] B.Mathew, A. Davis, and Z. Fang, "A low-power accelerator for the SPHINX 3 speech recognition system," *Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES),* pp. 210–219, 2003.

[3] S. Nedevschi, R. Patra, and E. Brewer, "Hardware speech recognition for user interfaces in low cost, low power devices," *42nd Annual Conf. on Design Automation (DAC),* pp. 684–689, 2005.

[4] E. Lin, Y. Kai, R. Rutenbar, and T. Chen, "A 1000-word vocabulary, speaker independent, continuous live-mode speech recognizer implemented in a single FPGA," *ACM/SIGDA 15th Int. Symp. on FPGA,* pp. 60–68, 2007.

[5] Y. Choi, K. You, and W. Sung, "FPGA-based implementation of a real-time 5000-word continuous speech recognizer," *16th European Signal Processing Conf., 2008.*

[6] X. Huang, A. Acero, and H.W. Hon, *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, New Jersey, 2001.

[7] B. Pellom, R. Sarikaya, and J. Hansen, "Fast likelihood computation techniques in nearest-neighbor based search for continuous speech recognition," *IEEE Signal Processing Letters,* vol. 8, no. 8, pp. 221–224, August 2001.

[8] H. Ney and S. Ortmanns, *"Dynamic programming search for continuous speech recognition,*" *IEEE Signal Processing Magazine,* pp. 64–83, 1999.

[9] Toradex, *Colibri XScale PXA270 Datasheet*, http://www.toradex.com, 2006.

[10] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P.Woodland, *The HTK Book Version 3.3*, 2005.