FIXED-POINT IMPLEMENTATIONS OF SPEECH RECOGNITION SYSTEMS

Yuet-Ming Lam
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
ymlam@cse.cuhk.edu.hk

Man-Wai Mak
Dept. of Electronic and Information Engineering
The Hong Kong Polytechnic University
Hung Hom, Hong Kong
enmwmak@polyu.edu.hk

Philip Heng-Wai Leong
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
phwl@cse.cuhk.edu.hk

ABSTRACT

Fixed-point hardware implementations of signal processing algorithms can often achieve higher performance with lower computational requirements than a floating-point implementation. However, the design of such systems is hard due to the difficulty of addressing the quantization issues. This paper presents an optimization approach to determining the wordlengths of fixed-point operators in a speech recognition system. This approach enables users to achieve the same result as in floating-point implementation with minimum hardware resources, resulting in reduced cost and perhaps lower power consumption. These techniques lead to an automated optimization based design methodology for fixed-point based signal processing systems. An object oriented library, called Fixed, was developed to simulate fixed-point quantization effects. Quantization effects during recognition were analyzed, and appropriate wordlength that can balance hardware cost and calculation accuracy were determined for the operators.

1. INTRODUCTION

Most previous hardware implementations of speech recognition systems used fixed-point arithmetic because it results in lower hardware requirements. A VLSI chip for isolated speech recognition system which can recognize 1000 isolated words per second was introduced in [1]. Some popular DSP chips such as the TMS320 series [2] have been widely used for implementing fixed-point speech recognition systems, e.g. [3] , [4] and [5].

Previous fixed-point implementations of speech recognition systems concentrated on optimizing recognition accuracy and real time performance, and quantization effects in fixed-point arithmetic were seldom directly addressed. With improvements in speech models and VLSI technology, speech recognition accuracy is much higher than in the past, and designing a speech recognition system with real time performance is not that difficult. However quantization issues remain a problem when designing a fixed-point hardware system.

This paper introduces a framework to address the quantization issues which arise in a fixed-point isolated word recognition system. A fixed-point class, called *Fixed*, was developed to simulate fixed-point arithmetic, the isolated word recognition system was described using the fixed-point class. By analyzing the quantization effects, optimal wordlengths for operators were found. These wordlengths are optimal in the sense that they can balance hardware cost and calculation accuracy. Compared with previous implementations, this approach has advantages in design effort and hardware resource utilization. For example, in [6], in order to find an optimal allocation of variables' precision, the authors need to implement several systems using different numerical formats, such as single fixed-point format, double fixed-point format and floating-point format. Our proposed approach, however is a simulation based approach and each variable can be of arbitrary precision. This leads to shorter design time and better resource utilization.

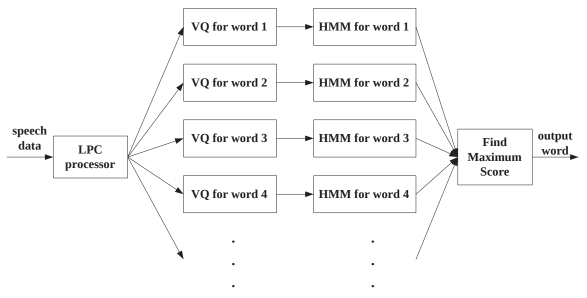

Figure 1: Isolated word recognition system

The organization of this paper is as follows. In section 2, the isolated word recognition model used in this paper is introduced. In section 3, we present the detail implementation of the fixed-point class. In section 4, we discuss the design methodology. In section 5, we present the experimental results. Section 6 is the conclusion.

2. SPEECH MODEL

Figure 1 is the block diagram of the isolated word recognition system used in this paper. The system contains three components: the linear predictive coding (LPC) processor, the vector quantizer (VQ) and the hidden Markov model (HMM) decoder [7]. In the system, a common LPC processor was used for all word models, with each word model having one VQ and one HMM decoder. All states of the HMM decoder share the same VQ codebook.

The linear predictive cepstral coefficients (LPCCs) [7] were extracted from the input speech data by an LPC processor. The LPCCs were passed to a VQ and a sequence that represents the codebook indexes having minimum distance with the LPCCs is produced. An HMM decoder was used to calculate the score for the VQ output sequence, and the word with the maximum score was chosen as the output word.

2.1. LPC feature analysis

In an LPC model, speech sample $s(m)$ at time m can be approximated as a linear combination of the previous t speech samples:

$$s(m) = \sum_{n=1}^{t} a_n s(m-n) + Gu(m) \tag{1}$$

where $Gu(m)$ is an excitation term. Transformed into the z-domain, we obtain the transfer function:

$$H(z) = \frac{1}{1 - \sum_{n=1}^{t} a_n z^{-n}} \quad . \tag{2}$$

As shown in Figure 2, the normalized excitation source $u(m)$ is scaled by the gain G. In speech recognition, the LPC feature analysis finds the filter coefficients a_n that can best model the speech data, which are used for further processing in the recognition process.

To calculate a_n, LPC feature analysis involves the following operations [7] for each speech frame:

1. Preemphasis and Windowing

$$\begin{aligned} p(m) &= s(m) - \alpha s(m-1), \\ & \qquad where \quad \alpha = 0.9375 & (3) \\ \tilde{p}(m) &= (0.54 - 0.46\cos(\frac{2\pi m}{N-1}))p(m), \\ & \qquad where \quad 0 \le m \le N-1 & (4) \end{aligned}$$

2. Autocorrelation Analysis

$$r(k) = \sum_{m=0}^{N-1-k} \tilde{p}(m)\tilde{p}(m+k), \qquad where \quad 0 \le k \le t \tag{5}$$

3. LPC Analysis

$$\begin{aligned} E^{(0)} &= r(0) & (6) \\ k_i &= \frac{r(i) - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} r(|i-j|)}{E^{(i-1)}}, \\ & \qquad where \quad 1 \le i \le t & (7) \\ \alpha_i^{(i)} &= k_i & (8) \\ \alpha_j^{(i)} &= \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)}, \quad 1 \le j \le i-1 & (9) \\ E^{(i)} &= (1-k_i^2)E^{(i-1)} & (10) \\ a_n &= \alpha_n^{(t)} \qquad where \quad 1 \le n \le t & (11) \end{aligned}$$

4. Conversion to cepstral coefficients

$$c_n = a_n + \sum_{k=1}^{n-1} (\frac{k}{n}) c_k a_{n-k}, \qquad where \quad 1 \le n \le t \tag{12}$$

The LPC coefficients a_n were converted to cepstral coefficients, which are the Fourier transform representation of the log magnitude spectrum. Using cepstral coefficients as features in speech recognition have been shown to be more reliable [7].

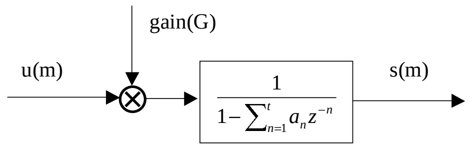

Figure 2: LPC model for speech recognition

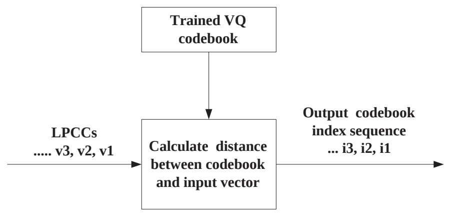
Trained VQ codebook
LPCCs
..... v3, v2, v1
Calculate distance between codebook and input vector
Output codebook index sequence
... i3, i2, i1

Figure 3: Vector quantizer

2.2. Vector Quantization

The VQ codebook is a discrete representation of speech. The VQ will find a codebook index corresponding to the vector that best represents a given spectral vector [7]. Figure 3 shows the vector quantization processing, for an input vector sequence $V\{v(1), v(2), v(3), ..., v(N)\}$, VQ will calculate the vector distance between each vector in codebook $C\{c(1), c(2), c(3), ..., c(P)\}$ and each input vector $v(n)$, and the codebook index with minimum distance will be chosen as output. After VQ, a sequence of codebook indexes $I\{i(1), i(2), i(3), ..., i(N)\}$ will be produced. The vector distance between an input vector $v(n)$ and each vector in codebook are calculated as follows:

```
for p from 1 to codebook_size {
  distance(p) = 0;
  for k from 1 to vector_length {
    temp = (v(n)(k) - c(p)(k))*(v(n)(k) - c(p)(k));
    distance(p) = distance(p) + temp;
  }
}
i(n) = arg min_p (distance(p));
```

2.3. HMM decoder

The Viterbi algorithm [7] is used to find the most likely state sequence and the likelihood score for a given observation sequence. Using the Viterbi algorithm, computation is reduced since the main operations are addition rather than multiplication as in the traditional HMM decoding algorithm [7]. The score δ for the VQ codebook index sequence is calculated as:

$$\delta_t(j) = \max_{1 \le i \le M} [\delta_{t-1}(i) + \log(a_{ij})] + log(b_j(o_t)) \tag{13}$$

where $\delta_t(j)$ is the maximum score along a single path ending at time t, a_{ij} is the probability of state transaction from i to j and $b_j(o_t)$ is the density function of the input symbol o_t's at state j.

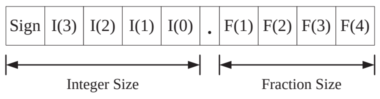

Figure 4: Fixed-point number representation

3. FIXED-POINT CLASS

In order to analyze quantization effects and utilize variable precision fixed-point arithmetic in the isolated word recognition system, a fixed-point class, called *Fixed*, was developed in the C++ language. The detailed implementation of the fixed-point class will be described later. The isolated word recognition system was also described in the C++ language using this fixed-point class. In this way, all operators in a floating-point implementation can be represented by fixed-point objects. A fixed-point object contains the following attributes:

```
<integer_size, fraction_size>
```

By defining different a $integer\_size$ and $fraction\_size$ for each fixed-point operator, all operators can be of arbitrary wordlength.

3.1. Fixed-point representation

Floating-point numbers can be represented in fixed-point format, and the format used in this work is shown in Figure 4. All fixed-point numbers are represented in 2's complement format. The Integer Size is the number of bits used to represent the integer part, and the Fraction Size is the number of bits used to represent the fraction part.

3.2. Fixed class

C++ was used because it offers faster execution than other object oriented languages such as Perl or Java. Several operators such as $+$, $-$, $*$ and $/$ are overloaded by the *Fixed* class.

- *+/-*: These operators can perform addition/subtraction of two *Fixed* objects and the result is also a *Fixed* object. The two input *Fixed* operands can have different integer size and fraction size, and the addition/subtraction result will choose the maximum integer and fraction size of the two input operands. For example, if the integer and fraction size for the first operand are 4 and 3 respectively and those for the second operand are 3 and 9, the result has integer size 4 and fraction size 9. The two operand are aligned, and addition/subtraction are done in fixed-point.
- ***: This operator can perform multiplication of two *Fixed* objects. The product's wordlength will be the summation of the two input operands' wordlength minus one.
- */*: This operator can perform division of two *Fixed* objects. To avoid loss of precision, the result will have the maximum integer and fraction size of the two input operands.
- *=*: If the input operand is a floating number, this operator will convert the floating point number into a fixed-point object at a precision specified by the target object. If the input operand is a fixed-point object, this operator will round it up to the target object's precision. Furthermore, the maximum value involved in the calculation is stored in this object, and the maximum value will be used to determine the minimum integer size of this operand.

The following method is provided by *Fixed*:

- *getIWL()*: The class can monitor the dynamic range of the calculation, and record the minimum and maximum values of each variable. This method will get the minimum integer size for this variable, the minimum integer size being calculated from the maximum value.

The fixed-point class also supports arrays, and saves the maximum value of the array. When calculating the minimum integer size, if this object is an array, the size will be calculated using the maximum value of all elements in the array.

3.3. Operator overloading

Converting a floating-point program into fixed-point implementation can be done by replacing the variable definition. The rest of the program is unchanged. For example, the following floating-point program:

```
float a;
float b;
float c;

a = 1.23;
b = 4.56;
c = a + b;
```

can be transformed into the fixed-point implementation:

```
Fixed a(4, 5);// integer size 4, fraction size 5
Fixed b(5, 6);// integer size 5, fraction size 6
Fixed c(5, 5);// integer size 5, fraction size 5

a = 1.23;
b = 4.56;
c = a + b;
```

In the above fixed-point program, in statements $a = 1.23$ and $b = 4.56$, since the " $=$ " operator is overloaded, floating-point values 1.23 and 4.56 will be converted into fixed-point format and stored in *Fixed* objects a and b. Statement "$a + b$" will be handled by the overloaded operator " $+$ " and the result will be a *Fixed* object. When assigned to c, the sum will be round up to the precision of c.

4. DESIGN METHODOLOGY

Figure 5 shows the design flow employed in this paper. VQ and HMM training were done using floating-point arithmetic. The isolated word recognition system was developed in C++ using the fixed-point class. After the calculation, the minimum integer size for each operand can be obtained. Minimum circuit area implementation can be found by further performing fraction size optimization.

4.1. Arithmetic circuit area calculation

Since this paper focuses on analyzing the quantization effects and finding a minimum wordlength implementation which will result in minimum circuit area, in the estimation of hardware costs, only arithmetic components are taken into account . Furthermore, in the isolated word recognition system, the main operations are addition, subtraction, multiplication and division, the circuit area calculation will only consider these four operations.

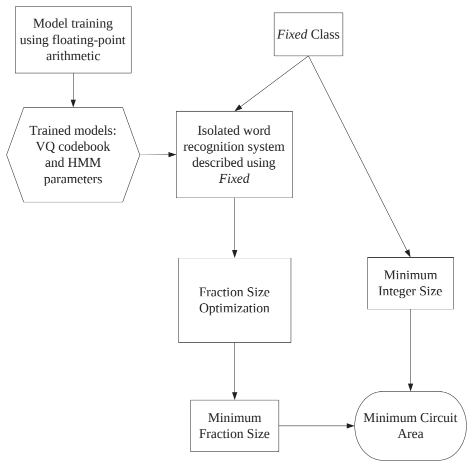
Model training using floating-point arithmetic
Fixed Class
Trained models: VQ codebook and HMM parameters
Isolated word recognition system described using *Fixed*
Fraction Size Optimization
Minimum Integer Size
Minimum Fraction Size
Minimum Circuit Area

Figure 5: Design flow

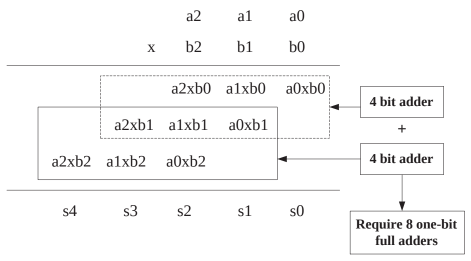

Figure 6: Standard multiplication algorithm

The cosine operation in the LPC feature analysis and logarithm operation in the HMM decoding were implemented using lookup tables. In Equation 4, only N cosine results need to be calculated, and in Equation 13, the logarithm function is required only for the calculation of state transaction probability.

When estimating the arithmetic circuit area, since addition, subtraction, multiplication and division can be implemented using adders, circuit area is counted as the number of one-bit full adders used for each arithmetic operation. The following descriptions detail the arithmetic circuit size estimation for each operation.

- *addition/subtraction*: Since the result chooses the maximum integer and fraction size of the two input operands, the number of full adders (i.e. the circuit size) used is: $Max(I1, I2) + Max(F1, F2)$, where ($I1$ and $F1$)/($I2$ and $F2$) are the integer size and fraction size for the first/second operand.
- *multiplication*: Different implementations of multipliers have different circuit sizes. In this paper, we used the parallel multiplier shown in Figure 6, In this example, two 4-bit adders are required and the total number of one-bit full adders required is 8. The circuit size of an n bit $\times$ m bit multiplier is: $n \times m - 1$
- *division*: A basic division algorithm, restoring-division [8], was used in this paper. Using this algorithm, only addition is used and the circuit size will be: $Max(I1, I2) + Max(F1, F2)$, where ($I1$ and $F1$)/($I2$ and $F2$) are the integer size and fraction size for the first/second operand.

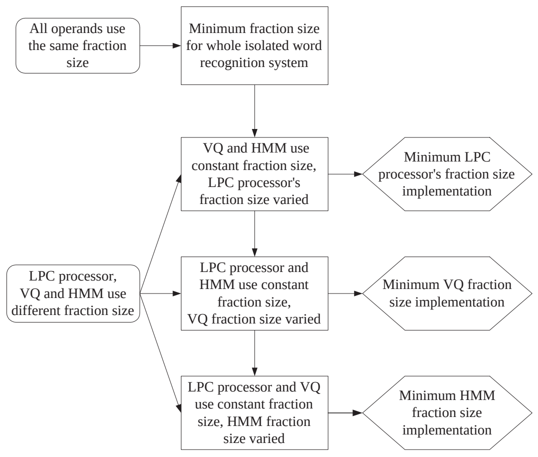
All operands use the same fraction size
Minimum fraction size for whole isolated word recognition system
VQ and HMM use constant fraction size, LPC processor's fraction size varied
Minimum LPC processor's fraction size implementation
LPC processor, VQ and HMM use different fraction size
LPC processor and HMM use constant fraction size, VQ fraction size varied
Minimum VQ fraction size implementation
LPC processor and VQ use constant fraction size, HMM fraction size varied
Minimum HMM fraction size implementation

Figure 7: System level optimization

The total arithmetic circuit size is calculated by summing the circuit size of all addition, subtraction, multiplication and division operators for the entire isolated word recognition system.

4.2. Fraction size optimization

After finding the minimum integer size for each operand, the optimization stage will find the minimum fraction size for each operand, which will result in minimum circuit size while achieving the same recognition accuracy as that of a floating-point system.

One difficulty in performing wordlength optimization is that the searching space is very large. For example, in a speech system containing 50 variables, and assuming an average wordlength of 16-bits, the total searching space is 16^{50}. Consequently, an alternative approach was used to limit the search space during wordlength optimization.

The approach taken for fraction size optimization will be introduced below. It is divided into two stages, in the first stage, it optimizes the fraction size from the entire system's point of view, and then in the second stage, it further optimizes the fraction size of the LPC processor.

4.2.1. System level optimization

In this stage, the isolated word recognition system was divided into three parts, namely the LPC processor, VQ and HMM decoder, and the fraction size of each part was minimized independently via a brute force search. The optimization steps are shown by Figure 7.

1. Optimize whole speech system's fraction size
2. Optimize LPC processor's fraction size
3. Optimize VQ's fraction size
4. Optimize HMM decoder's fraction size

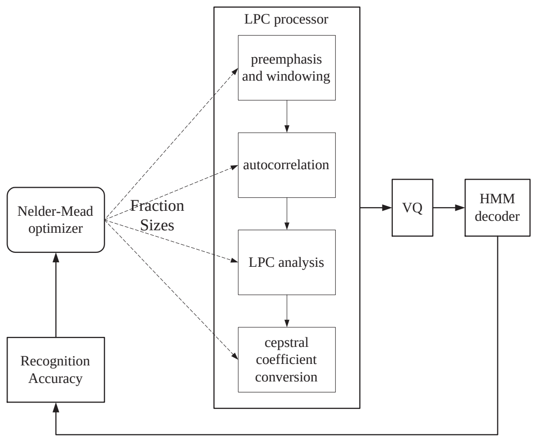

Figure 8: LPC processor's fraction size optimization flow

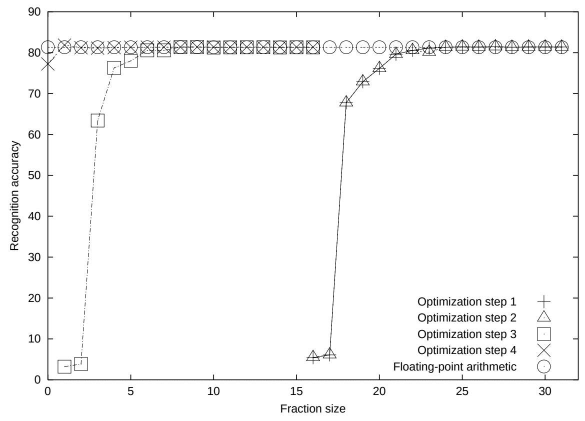

Figure 9: Recognition accuracy for system level optimization

4.2.2. *LPC processor's fraction size optimization*

Since the LPC processor occupies most of the hardware resources, the LPC processor was further divided into four parts, namely preemphasis and windowing, autocorrelation analysis, LPC analysis, and cepstral coefficient conversion. Optimization was done to minimize these four fraction sizes as shown in Figure 8. An optimizer using the Nelder-Mead algorithm [9] was used to minimize the hardware cost of the LPC processor based on a user defined cost function which takes recognition accuracy and implementation cost into account, finding an implementation which balances hardware cost and recognition accuracy. The following cost function was used:

$$cost = \alpha * LPC\_CircuitSize + \beta * RegAcy \quad (14)$$

where *RegAcy* is the recognition accuracy calculated using fixed-point arithmetic, and α and β are the weights of circuit area and recognition accuracy respectively. Note that α and β can be adjusted for different weightings of circuit area and recognition accuracy. For example, if circuit area is very important, α can be a large value and β can be small.

In this work, the following condition was added to compute the cost:

```
if (RegAcy < expectRegAcy) {
    penalty = expectRegAcy - RegAcy;
    cost = cost + VeryLargeValue * penalty;
}
```

where *expectRegAcy* is a user defined expected recognition accuracy. In this work, the recognition accuracy of a floating-point system was used. The optimizer will try different combinations of fraction size for the four parts in the LPC processor, cost is increased when recognition accuracy is smaller than the user specified recognition accuracy, and the optimizer will recognize that it must try other fraction sizes to obtain higher recognition accuracy.

5. RESULTS

The isolated word recognition system uses 12th order LPCCs , a codebook with 64 code vectors, and 20 HMMs, each with 12 states. One set of utterances from the TIMIT TI 46-word database [10] containing 20 words from 8 males and 8 females were used for both training and recognition. There were 26 utterances for each word, 10 were used for training and 16 were used for recognition. The recognition accuracy calculated using floating-point was 81.33%.

The isolated word recognition system was developed using the *Fixed* class. Since fixed-point arithmetic was simulated using objects, the execution time is much longer than a floating-point implementation using primitive float type. Specifically, 7.35 hours were required to perform recognition for all words using fixed-point simulation on an Intel Pentium 4 2.2GHz processor. Parallel computing was used carry out the simulation such that different fraction size implementations were executed in parallel using a Linux cluster with 32 processors.

Figure 9 shows the result obtained after optimization step described in Section 4.2.1, which is system level optimization. In this diagram, the line with marker "$\circ$" represents the recognition accuracy using floating-point arithmetic, and other lines are the recognition accuracy using fixed-point arithmetic for different optimization step.

The line with marker "$+$" in Figure 9 represents the recognition accuracy obtained after system level optimization step 1. All operators in the isolated word recognition system use the same fraction size in this step. It can be seen that at fraction size 24, the fixed-point calculation reaches the same recognition accuracy as floating-point arithmetic.

The line with marker "$\triangle$" in Figure 9 represents the recognition accuracy obtained after system level optimization step 2. In this step, the VQ and HMM decoder have fraction size 24, and the LPC processor's fraction size is varied. It can be seen that when the LPC processor's use fraction size 24, the fixed-point calculation can reach floating point calculation's recognition accuracy.

The line with marker "$\Box$" in Figure 9 represents the recognition accuracy obtained after system level optimization step 3. In this step, the LPC processor and HMM decoder have a fraction size of 24, and the VQ fraction size is varied. It shows that, at a VQ fraction size of 8, the fixed-point calculation has the same recognition accuracy as floating-point arithmetic.

The line with marker "$\times$" in Figure 9 represents the recogni-

Table 1: System level fraction size optimization

	LPC processor fraction size/ LPC circuit size	VQ fraction size/ VQ circuit size	HMM decoder fraction size/ HMM circuit size	Total Circuit Size
Before optimization	24/9076	24/842	24/195	10100
After optimization	24/9076	8/186	1/57	9306

Table 2: LPC processor's fraction size optimization

	Before LPC optimization	After LPC optimization
Preemphasis and windowing fraction size	24	20
Autocorrelation fraction size	24	31
LPC analysis fraction size	24	20
Cepstral conversion fraction size	24	19
LPC processor circuit size	9076	6857
Total circuit size	9306	7100

tion accuracy obtained after system level optimization step 4. In this step, the LPC processor's fraction size is fixed at 24, VQ fraction size at 8, and the HMM decoder's fraction size is varied. In Figure 9, it can be seen that using a fraction size of 1 for the HMM decoder is sufficient.

Table 1 shows the circuit size before and after system level fraction size optimization and optimal fraction sizes of 24, 8 and 1 were found for LPC processor, VQ and HMM decoder respectively. Circuit size before optimization was 10100. After optimization, it is reduced to 9306, a 7.9% improvement.

Table 2 shows the circuit size before and after the LPC processor's fraction size optimization. After optimization, fraction sizes of 20, 31, 20 and 19 were found for preemphasis and windowing, autocorrelation analysis, LPC analysis and cepstral coefficient conversion respectively. LPC circuit size after optimization is 6857, and the total circuit size after optimization is 7100. Compared with the original circuit size of 10100, an improvement of 29.7% was achieved.

Fraction size 1 is sufficient for the HMM decoder. As shown in Figure 1, in HMM decoding, the scores for all words are sorted and the word with highest score will be chosen as output. Although there are some errors for smaller fraction sizes, the correct score sequence for all words still can be calculated. Furthermore, since the HMM state transaction probabilities are computed in the log domain, the integer part is the most important factor that affects the scores.

6. CONCLUSION

A framework involving a fixed-point library was developed to address the quantization issues of fixed-point systems. An object oriented library, called *Fixed*, was developed to simulate fixed-point arithmetic such as addition, multiplication and division where each operator can use wordlength of arbitrary precision.

This framework was applied to an isolated word recognition system based on a vector quantizer and a hidden Markov model decoder using LPCCs as features. Utterances from the TI 46-word TIMIT database were used for both training and recognition. Training was done in floating-point format to find the VQ codebook and maximum likelihood estimates of the HMM parameters.

Recognition accuracy are used as optimization constraints, such that the wordlengths for the LPC processor, VQ and HMM decoder were found. The optimized wordlengths result in a 29.7% reduction in circuit area. Such an approach leads to clear advantages both in design effort and hardware resource utilization over the traditional approaches where the same wordlength is used for all operators.

7. REFERENCES

[1] Sung-Nam Kim, In-Chui Hwang, Young-Woo Kim, and Soo-Won Kim, "A VLSI chip for isolated speech recognition system," in *IEEE Transactions on Consumer Electronics, Volume: 42, Issue: 3*, pp. 458–467, 1996.

[2] Kun-Shan Lin, Gene A. Frantz, and Ray Simar, Jr., *The TMS320 Family of Digital Signal Processors.* Texas Instruments, 1997.

[3] Yifan Gong and Yu-Hung Kao, "Implementing a high accuracy speaker-independent continuous speech recognizer on a fixed-point DSP," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 3686–3689 vol.6, 2000.

[4] K.K. Shin, J.C.H. Poon, and K.C. Li, "A fixed-point DSP based Cantonese recognition system," in *IEEE International Symposium on Industrial Electronics*, pp. 390–393 vol.1, 1995.

[5] Nishida, Y., Nakadai, Y., Suzuki, Y., Sakurai, T., Kurokawa, T., and Sato, H., "Voice recognition focusing on vowel strings on a fixed-point 20-MIPS DSP board," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 137–140 vol.1, 1999.

[6] Guanghui Hui, Kwok-Chiang Ho, and Zenton Goh, "A robust speaker-independent speech recognizer on ADSP2181 fixed-point DSP," in *1998 Fourth International Conference on Signal Processing*, pp. 694–697 vol.1, 1998.

[7] Lawrence Rabiner and Biing-Hwang Juang, *Fundamentals of Speech Recognition*. Prentice Hall PTR, 1993.

[8] Carl Hamacher, Zvonko Vranesic, and Safwat Zaky, *Computer Organization, fifth edition*. McGraw-Hill Companies, Inc, 2002.

[9] J. Nelder and R. Mead, "A simplex method for function minimization," in *Computer Journal*, pp. 308–313, 1965.

[10] LDC, *http://www.ldc.upenn.edu*.