System-on-Chip Architecture for Speech Recognition

GIN-DER WU AND KUEI-TING KUO
Department of Electrical Engineering
National Chi Nan University
Puli, 545 Taiwan

This paper proposed a system-on-chip (SOC) architecture for speech recognition which is speaker dependent. The feature extraction bases on LPC (linear predictive coefficient)-cepstrum coefficients, and template matching employs Hidden Markov Models (HMM). It does not aim to offer a sophisticated solution but rather a high speed solution. This SOC architecture includes an ASIC of LPC-cepstrum and a Dual-ALU processor. The proposed ASIC of LPC-cepstrum can reduce the calculation load of processor in the speech recognition system. To reduce the area of this ASIC, the resource sharing method is adopted into our design. In addition, this paper also proposed the Dual-ALU processor which provides parallel calculation capability. Hence, it can run more complicated algorithm of speech recognition. For the consideration of chip size, the area of the second ALU is only half of the first ALU. From the experiments, the speech recognition system can provide a high speed solution.

Keywords: system-on-chip, LPC-cepstrum, HMM, ASIC, dual-ALU, speech recognition

1. INTRODUCTION

In recent years, many researchers proposed speech recognition structure to satisfy the requirement of portable device. The implementation of the speech recognition can be classified as software design and hardware-software codesign method. Shi et al. [1] proposed an 8051 single chip to implement the speech recognition. However, its computation ability is not adequate to run more complicated robust speech recognition algorithms. Therefore, the high-performance DSP is used to speed up the execution of speech recognition [2]. However, since the general-purpose DSP is not a dedicated design for speech recognition, the cost may be higher than the dedicated hardware. ASIC implementation is another useful method to implement the speech recognition. [3] proposed the speech recognition IC with double mixtures. [4-7] proposed the hardware architecture of HMM architecture. Although these methods can speed up speech recognition, the flexibility is relatively low. Hence, [8, 9] proposed the application-driven digital signal processor (LASP24) to implement speech/audio algorithm. However, the power consumption of LASP24 is not suitable for portable device. In order to solve the above problems, this paper proposed a system-on-chip architecture for speech recognition which contains an ASIC of LPC (linear predictive coefficient)-cepstrum and a high speed Dual-ALU processor. It implements isolated-word and small-vocabulary speech recognition. This ASIC of LPC-cepstrum can reduce the computation load of processor. To reduce the area of a chip, the resource sharing method is adopted into the design of LPC-cepstrum. In addition, since the proposed high speed Dual-ALU processor has the parallel calculation capability, it can run more complicated algorithm of speech recognition.
The paper is organized as follows. The implementation of LPC-cepstrum is proposed in section 2. In section 3, the Dual-ALU processor is proposed for HMM. Finally, the experiments and conclusion are presented in sections 4 and 5 respectively.

Fig. 1. The flowchart of the proposed speech recognition system.

2. ASIC OF LPC-CEPSTRUM

The flowchart of the proposed speech recognition is shown in Fig. 1. First, we focus on the speech recognition feature. The LPC-cepstrum is commonly adopted because it can provide an accurate estimation of speech. Besides, it has relatively less computation load. Hence, we proposed an ASIC of LPC-cepstrum to implement the speech feature extraction.

\[ y[n] = x[n] - ax[n-1] \]  

(1)

where \( x[n] \) is input speech signal, \( y[n] \) is the result of output, and \( a \) is 0.95 for pre-emphasis. The corresponding circuit of the pre-emphasis is shown in Fig. 2. Then we adopt Hamming window in Eq. (2).

\[
w(n) = \begin{cases} 
0.54 - 0.46 \cos \left( \frac{2n\pi}{N-1} \right), & 0 \leq n \leq N-1 \\
0, & \text{otherwise}
\end{cases}
\]

(2)

Fig. 2. The circuit of pre-emphasis.
where \( N \) is the window length, which is 240 points. The corresponding circuit is shown in Fig. 3. This circuit operates in two different clock domains. One is 8 KHz for sampling 16-bit speech signal, and the other is 100 MHz for internal signal processing.

Before we use Durbin’s recursive procedure [10] to derive the LPC coefficients, we must calculate the autocorrelation coefficient \( R(k) \) of the speech signal as follows.

\[
R(k) = \sum_{n=k}^{N-1} x(n)x(n-k), \quad 0 \leq k \leq p
\]

(3)

where \( p \) is the value of order. Based on Eq. (3), we can use Eqs. (4)-(9) to get the LPC coefficients. This autocorrelation method leads to more stable results than covariance method. Besides, it has lower computational load than the lattice method [11].

\[
E^{(0)} = R^{(0)},
\]

(4)

\[
K_i = \frac{R(i) - \sum_{j=1}^{i-1} \alpha_j^{(i-j)} R(i-j)}{E^{(i-1)}}, \quad 1 \leq i \leq p,
\]

(5)

\[
\alpha_i^{(i)} = k_i,
\]

(6)

\[
E^{(i)} = (1-K_i^2)E^{(i-1)},
\]

(7)

\[
\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_j^{(i-1)} - 1 \leq j \leq i-1,
\]

(8)

\[
\alpha_j = \alpha_j^{(p)},
\]

(9)

where \( \alpha_j \) is the LPC coefficient, \( E \) is error value, and \( K \) is partial correlation coefficient.

For the consideration of implementation [12], we have to rewrite Eqs. (4)-(9) as follows.

\[
E = R[0];
\]

for \( i = 1; i \leq \text{LPC\_ORDER}; i++ \) {
    \[
    \text{sum} = 0.0;
    \]
    for \( j = 1; j < i; j++ \) {
        \[
        E = (1-K_i^2)E;
        \]
        // Other calculations...
    }
}
\[ \text{sum} = \alpha[j][j-1] \times R[i-j]; \]  
\[ K = (R[i] - \text{sum}) / E; \]  
\[ \alpha[i][i] = K; \]  
\[ E* = (1 - K * K); \]  
\[ \text{for } (j = 1; j \leq i - 1; j++) \{ \]  
\[ \alpha[j][i] = \alpha[j][j-1] - K * \alpha[i-j][i-1]; \]  
\[ \} \]

Finally, we use the following equation to perform cepstral coefficient extraction.

\[ C_j = \alpha_j^{(p)} + \sum_{k=1}^{j-1} \left( \frac{k}{j} \right) C_j \alpha_{j-k}^{(p)}, 1 \leq j \leq p \]  

where \( C_j \) is the coefficient of LPC-cepstrum for order \( j \). To implement LPC-cepstrum Eqs. (4)-(10), we proposed the circuit in Fig. 4. In Fig. 4 (a), the left register array saves auto-correlation coefficients, and the right register array saves LPC coefficients. First of all, Eq. (4) is initialized into Block C. For the consideration of \( \sum_{j=1}^{i-1} \alpha_j^{(j-1)} R(i-j) \) in Eq. (5), “sel_m1” selects \( R(i) \) and “sel_23” selects \( \alpha_j^{(i-1)} \) into Block A. Since Block B can do the summation/subtraction by “sel_out”, we can calculate \( \sum_{j=1}^{i-1} \alpha_j^{(i-1)} R(i-j) \) as “Sum” and get \( R(i) - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} R(i-j) \). Based on this result, we use the division circuit in Block C to get the result of Eq. (5). As far as Eq. (6) is concerned, it is just the transfer function. Since Eqs. (7) and (8) can be implemented by using Block A and Block B, Eq. (9) can be easily calculated.

To calculate “sum”, “E”, “K”, and “\( \alpha \)”, the Finite State Machine (FSM) which has 4 states is used to control the architecture in Fig. 4 (a). We use the counter to determine when to jump to the next stage. The 1st state is used to calculate “sum”. The 2nd state is used to calculate “K”. When the division finishes the calculation, “div_finish” will changes from 0 to 1 and move to the 3rd state which is used to calculate the value of “E”. The last state is used to derive “\( \alpha \)”. Finally, Eq. (10) is implemented as shown in Fig. 4 (b) [13]. We use the other Finite State Machine (FSM) which has 2 states to control this cepstral coefficient extraction. The first state is used to calculate the multiply-add function. The next state is used to calculate the division function. When the division finishes the calculation, “div_finish” will changes from 0 to 1.

For division operation, we implement it by using prune-and-search [11] as shown in Fig. 5. In order to reduce the chip area, the resource sharing method is adopted into our design. For example, we use the same multiplier circuit in Figs. 4 and 5. The ASIC of LPC-cepstrum can work on 100 MHz. Besides, since floating-point system is more complex than fixed-point system, we use fixed-point format to define the data in this hardware. The data type in the ASIC of LPC-cepstrum is shown in Fig. 6.

To evaluate the performance, we analyze the autocorrelation, LPC and LPC-Cepstrum in Table 1. In autocorrelation, it includes the operation of hamming window. Hence, we apply the “addition” for 2640 times, and the “multiplication” for 2880 times. In LPC part, the “addition” and “multiplication” are applied 100 times. Besides, the division is executed by 10 times. In the circuit of cepstral coefficient extraction (LPC-Cepstrum),
Fig. 4. (a) The hardware architecture of the proposed LPC-cepstrum; (b) The circuit of cepstral coefficient extraction.
Fig. 5. The division circuit.

Fig. 6. The data type in the ASIC of LPC-cepstrum.

Table 1. Analysis of the circuit in autocorrelation, LPC and LPC-cepstrum.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Autocorrelation</th>
<th>LPC</th>
<th>LPC-Cepstrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition/Subtraction (times)</td>
<td>2640</td>
<td>100</td>
<td>45</td>
</tr>
<tr>
<td>Multiplication (times)</td>
<td>2880</td>
<td>100</td>
<td>90</td>
</tr>
<tr>
<td>Division (times)</td>
<td>0</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>ROM size (bits)</td>
<td>120 * 16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total Clock Cycles</td>
<td>2880</td>
<td>230</td>
<td>207</td>
</tr>
</tbody>
</table>

Table 2. Performance comparison among ADI DSP-2191, KOINDOL and LPCC ASIC.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Architecture</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPCC</td>
<td>ADI DSP-2191</td>
<td>1376943</td>
</tr>
<tr>
<td></td>
<td>KOINDOL [14]</td>
<td>4000</td>
</tr>
<tr>
<td></td>
<td>Proposed LPCC ASIC</td>
<td>3317</td>
</tr>
</tbody>
</table>
the “addition” and “multiplication” need to be executed for 45 and 90 times respectively. In addition, the division is executed for 9 times. In this table, the execution of division needs 13 clock cycles.

To give more comparisons on its performance with other designs, we use ADI DSP-2191 to implement the LPC-cepstrum (LPCC) by C compiler in Table 2. Since this C compiler is not very efficient, it totally needs 1376943 cycles. It is obvious that the proposed LPCC ASIC can reduce huge computation cycles. In addition, Table 2 also compares with the other design of LPCC ASIC in [14] (KOINDOL). Since it needs 400 µs to compute LPC-cepstrum of each frame at 10 MHz, the corresponding computation is 4000 cycles. We can see that our proposed LPCC ASIC is still the best solution among these three algorithms.

3. DUAL-ALU ARCHITECTURE FOR SPEECH RECOGNITION

After the feature extraction procedure, we use the HMM algorithm for speech recognition. To implement the HMM algorithm, we proposed a Dual-ALU processor and adopt its instruction set to implement HMM algorithm. We adopt the instruction set of this processor to run HMM algorithm. The block diagram of this processor is shown in Fig. 7. It can be divided into the control unit, data register file, computational unit, memory address generator, embedded data memory and program sequencer. The computational unit performs the numeric processing of speech recognition. This unit gets the data from registers in the data register file. In addition, computational instructions provide fixed-point operations. Each instruction can completely execute in one clock cycle. To increase the performance, this processor uses four-level pipeline hardware design. It comprises four stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX) and Write (W). A pipeline conflict happens when an instruction reads from or writes to a
place (such as register, memory or bus) which is not ready to be read or written. The hardware design should supervise this pipeline conflict and make some corresponding operation. In other words, the bypass and forwarding will be activated when the hardware detects a place (such as register, memory or bus) which is not ready to be read or written.

![Dual-ALU architecture diagram](image)

Fig. 8. Dual-ALU architecture.

The detail architecture of the Dual-ALU is shown in Fig. 8. In this computational unit, the first ALU performs arithmetic, logic, multiplication, shift, execute multiply/add and multiply/subtract operations. For the consideration of cost, the second ALU only performs arithmetic and multiplication operations. Hence, the area of the second ALU is only half of the first ALU. The Dual-ALU architecture supports parallel execution ability so that this processor can execute two fixed-point arithmetic operations in one clock period. Hence the computational unit can increase execution efficiency especially for repeat loop and nested loop of program. Besides, the register array is shown in Fig. 8, and each register is 32-bit. It can be viewed as a pair of independent 16-bit registers. Data flow paths through the computational units are arranged in parallel. By using the parallel data paths within Dual-ALU architecture, any output of computational unit may serve as any input of computational unit on the next instruction cycle. We can use single instruction to get two operation results from the Dual-ALU structure. The example of usage is shown as follows.

```assembly
MOV  #32,R0.L
MOV  #23,R0.H
MOV  #42,R1.L
MOV  #9, R1.H

ADAD R0.HL, R1.HL, R2.H, R2.L  ;R2.H = R0.H + R1.H = 31
ADSB R0.HL, R1.HL, R3.H, R3.L  ;R3.H = R0.H + R1.L = 31
       ;R4.L = R0.L * R1.L = 1444
```
Since the instruction set is the interface between the hardware and the software, the performance of the processor depends on the instruction set. We define seven instruction types including Data-transfer, Boolean, Add/Sub, Mac, Dual-ALU, Shift/Rotate, and Control. They are 32-bit format. However, the implementation of HMM algorithm needs some divisions. For the consideration of chip size, it is not economical to design extra division hardware. Hence we use software coding to implement the division. We use the reciprocal of divisor to replace divisor and change division to multiplication as follows.

\[
\text{dividend} \times \frac{1}{\text{divisor}} = \text{quotient} + \text{residue}
\]

To clarify the reciprocal of divisor, the examples are described as follows.

\[
\frac{1}{3} = 0.333333 \rightarrow 0x2AAA \rightarrow 0010-1010-1010-1010
\]

Dividend: 0000-0000-0000-0001
Divisor: 0000-0000-0000-0011
Dividend – Divisor < 0
Quotient: 0000-0000-0000-0000
Residue: 0000-0000-0000-0001

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0010 – 0000-0000-0000-0011 < 0
Quotient: 0000-0000-0000-0000
Residue: 0000-0000-0000-0010

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0100 – 0000-0000-0000-0011 > 0
Quotient OR 0000-0000-0000-0001 = 0000-0000-0000-0001
Residue: 0000-0000-0000-0001

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0010 – 0000-0000-0000-0011 < 0
Quotient: 0000-0000-0000-0000
Residue: 0000-0000-0000-0010

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0100 – 0000-0000-0000-0011 > 0
Quotient OR 0000-0000-0000-0001 = 0000-0000-0000-0001
Residue: 0000-0000-0000-0001

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0010 – 0000-0000-0000-0011 < 0
Quotient: 0000-0000-0000-0000
Residue: 0000-0000-0000-0010

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0100 – 0000-0000-0000-0011 > 0
Quotient OR 0000-0000-0000-0001 = 0000-0000-0000-0001
Residue: 0000-0000-0000-0001

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0010 – 0000-0000-0000-0011 < 0
Quotient: 0000-0000-0000-0000
Residue: 0000-0000-0000-0010

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0100 – 0000-0000-0000-0011 > 0
Quotient OR 0000-0000-0000-0001 = 0000-0000-0000-0001
Residue: 0000-0000-0000-0001

Quotient left shift 1bit & Residue left shift 1bit
0000-0000-0000-0010 – 0000-0000-0000-0011 < 0

For the consideration of testability in the circuit, there are two auxiliary circuits. First, we created Built-In-Self-Test (BIST) circuit for the embedded memory of SRAM. The BIST controller uses the test algorithm to write test patterns into embedded memory. Then it reads each data from embedded memory in order to compare with the test pattern. Hence we can detect any error in the embedded memory. Secondly, one scan chain is inserted into each module of the processor by using DFT-compiler. All the sequential logics including the RAM BIST circuit are connected as a chain for serial-in-serial-out scan test. To implement the speech recognition system, we link up this Dual-ALU processor with ASIC of LPC-cepstrum in Fig. 9 (a). We build the protocol to connect the Dual-ALU processor with the ASIC of LPC-cepstrum. Fig. 9 (b) shows the waveform of this protocol. When both “Busy” and “INT” are “Hi”, ASIC of LPC-cepstrum will send the 16-bit data to Dual-ALU processor. After receiving the 10th transmission data, the
Fig. 9. (a) Structure of the dual-ALU processor connects with the ASIC of LPC-cepstrum; (b) Waveform of the protocol.

port of “Busy” becomes “Low”. Then the Dual-ALU processor begins to process these data by HMM algorithm.

4. EXPERIMENTS

We adopt chinese digit (0-9) words from 10 speakers. Everyone speaks 10 times. The 16-bit speech signal is sampled in 8 KHz. The feature extraction bases on LPC-cepstrum coefficient which comes from Fig. 4. To see the effect of computation load, we implement two different algorithms of the template matching. They are Dynamic Time Warping (DTW) and Hidden Markov Models (HMM) respectively. Table 3 shows the performance comparison of speech recognition among single-ALU, Dual-ALU and Dual-ALU with LPCC ASIC. Based on this table, the HMM has higher speech recognition rate than DTW. In addition, the Dual-ALU structure outperforms the single-ALU structure. For an example of HMM in Fig. 10, we can see the effect of Dual-ALU design. Since one single instruction can get two operation results in one clock cycle, we use one Dual-ALU instruction instead of two successive arithmetic instructions in Fig. 10. It can effectively reduce execution cycles. With the ASIC of LPC-cepstrum, we can further reduce the processing time of each frame to be 2421742 cycles (24.2 ms) in 100MHz. Since each frame size is 240 points (30 ms), it can satisfy the requirement of real time processing. For the consideration of complexity and cost, we don’t need to choose more than two core architecture. This is the reason why we adopt Dual-ALU architecture.

The main source of power consumption in a typical CMOS logic gate is due to the switching power, $P_{SW}$.

$$P_{SW} = \frac{1}{2} k C_{load} V_{dd}^2 f$$  \hspace{1cm} (12)

Table 3. Performance comparison among single-ALU, Dual-ALU and Dual-ALU with LPCC ASIC.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Architecture</th>
<th>ROM</th>
<th>SRAM</th>
<th>Cycles</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTW</td>
<td>Single-ALU</td>
<td>16</td>
<td>8</td>
<td>234189</td>
<td>91.8%</td>
</tr>
<tr>
<td></td>
<td>Dual-ALU</td>
<td>16</td>
<td>8</td>
<td>135476</td>
<td>91.8%</td>
</tr>
<tr>
<td></td>
<td>Dual-ALU with LPCC ASIC</td>
<td>14</td>
<td>8</td>
<td>111026</td>
<td>91.6%</td>
</tr>
<tr>
<td>HMM</td>
<td>Single-ALU</td>
<td>23.7</td>
<td>8</td>
<td>3693930</td>
<td>95.0%</td>
</tr>
<tr>
<td></td>
<td>Dual-ALU</td>
<td>23.7</td>
<td>8</td>
<td>2464172</td>
<td>95.0%</td>
</tr>
<tr>
<td></td>
<td>Dual-ALU with LPCC ASIC</td>
<td>21</td>
<td>8</td>
<td>2421742</td>
<td>94.8%</td>
</tr>
</tbody>
</table>
Fig. 10. HMM implementation by using single-ALU and Dual-ALU instructions.
where $V_{dd}$ is the supply voltage, $f$ is the clock frequency, $C_{load}$ is the load capacitance of the gate, and $k$ is the switching activity factor which is defined as the average number of times that the gate makes an active transition in a single clock cycle. To calculate the power consumption, we used the design compiler which is developed by SYNOPSYS Company. Table 4 shows the chip features comparison. Compared with Chung et al. [8, 9], the Dual-ALU processor can reduce power consumption to only 50.58 mW. In addition, the die size of this Dual-ALU only has 4.3mm$^2$. Therefore, it provides a low-power and high speed solution for portable device. The layout view of the proposed architecture is shown in Fig. 11.

### Table 4. Chip features comparison.

<table>
<thead>
<tr>
<th>Processor</th>
<th>LASP24 [8, 9]</th>
<th>Proposed Dual-ALU processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process technology</td>
<td>UMC 0.18µm 1P6M COMS</td>
<td>UMC 0.18µm 1P6M COMS</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>100MHz</td>
<td>100MHz</td>
</tr>
<tr>
<td>Supply voltage</td>
<td>1.8V</td>
<td>1.8V</td>
</tr>
<tr>
<td>Power consumption</td>
<td>81.2mW</td>
<td>50.58mW</td>
</tr>
<tr>
<td>Die size</td>
<td>6.5mm$^2$</td>
<td>4.3mm$^2$</td>
</tr>
</tbody>
</table>

Fig. 11. The layout view of the proposed architecture.

For the consideration of data processing, there is an 8 Kbyte SRAM embedded memory in Dual-ALU processor. This on-chip SRAM is optimized for UMC’s 0.18-micron, 1.80-volt, and six-layer metal COMS process technology. To compare with ARM which is the commercial general purpose core, it does not support so large embedded memory. This is a drawback when ARM runs complicated speech algorithm. Besides, our Dual-ALU processor can support parallel calculation capability. Although ARM can provide lower power consumption, our parallel calculation capability makes the Dual-ALU based program to be more efficient than ARM-based program. For example, we implement the hamming window (2) by using ARM and the proposed Dual-ALU processor. The program instructions are shown in Table 5. It is obvious that the clock cycles of Dual-ALU processor is less than the clock cycles of ARM. In other words, our proposed Dual-ALU processor can run more complicated algorithm of speech recognition at the same clock frequency. Hence, our Dual-ALU is better than ARM core in the application of speech recognition.
Table 5. Performance comparison between ARM and Dual-ALU processor.

<table>
<thead>
<tr>
<th>Instruction code</th>
<th>ARM</th>
<th>Dual-ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop: mov r0,#17694; mov r1,#15073; mov r2,#265; mov r3,#496; mov r4,#239 mov r5,[r2] mul r1,r5,r5 sub r0,r5,r5 mov r5,[r3] add r2,#1,r2 add r3,#1,r3 sub r4,r4,#1 bne loop</td>
<td>/mov #0.54,r0 /mov #0.46,r1 /r2=addr.cos() /r3=addr.w[n] //N=240</td>
<td>mov #17694,r2h; /mov #0.54,r2h mov #15073,r21; /mov #0.46,r21 mov #265,r3; /r3=addr.cos() mov #496,r3h; /r3h=addr.w[n] mov [r31],r01; mpyi r21,r01,r0h rpb #239 inc r31 mov [r31],r01 sbmpy r2h1,r0h1,r31,r0h; mov[r3h],r31; inc r3h retb</td>
</tr>
</tbody>
</table>

For Single Instruction Multiple Data (SIMD), not only the number of loop iterations can be reduced, but also the multiple operations can be reduced to a single action. This concept is the same as our proposed Dual-ALU architecture. However, the major character of Dual-ALU is easy design. In computational unit, the first ALU performs arithmetic, logic, multiplication, shift, execute multiply/add and multiply/subtract operations. The second ALU only performs arithmetic and multiplication operations which have high utility-rate in speech recognition. For the consideration of flexibility in design, you can change the second ALU according to different application. The corresponding assembler can be changed by slightly modifying the format of operation-code. Since only high utility-rate operators are selected into the second ALU, the area of the second ALU is only half of the first ALU. In other words, Dual-ALU is a very flexible and efficient architecture. The concept of SIMD does not address this character.

The Dual-ALU processor works at 100MHz. In theory, the maximum peak performance can reach 200 MIPS (Million Instructions Per Second). In Table 3, the Dual-ALU with LPCC ASIC only needs 2421742 clock cycles (2.42 MIPS) to run HMM. It only occupies very small portion of the computation ability. Therefore, the proposed architecture can run more complicated algorithm of speech recognition at the same clock frequency.

5. CONCLUSION

This paper proposed a system-on-chip (SOC) architecture for speech recognition. It contains an ASIC of LPC-cepstrum and a Dual-ALU processor. We use hardware-software codesign method to implement the speech recognition. It does not aim to offer a sophisticated solution but rather a high speed solution. The proposed ASIC of LPC-cepstrum can reduce the calculation load of processor in the speech recognition system. To reduce the area of this ASIC, the resource sharing method is adopted into our design. In addition, this paper also proposed the Dual-ALU processor which provides parallel calculation capability. Hence, it improves the performance of speech recognition. For the consideration of chip size, the area of the second ALU is only half of the first ALU.
From the experiments, the speech recognition system can provide a high speed solution for portable device.

REFERENCES


**Gin-Der Wu** (吳俊德) received the B.S. degree in Engineering Science from the National Cheng Kung University, Tainan, Taiwan, R.O.C. in 1996 and the Ph.D. degree in Electrical and Control Engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C. in 2000. In 2002, he joined VIA Technologies, Inc., as a senior engineer in Chipset R&D Division. In 2003, he joined ALI Corporation, as a senior engineer in Digital AV Product Business Division. Since February 2004, he joined the National Chi Nan University, Nantou, Taiwan, R.O.C., where he is currently an Assistant Professor of Electrical Engineering. His current research interests are digital signal processing, neural fuzzy networks, and VLSI chip design.

**Kuei-Ting Kuo** (郭桂廷) received the B.S. degree in Department of Electronic Engineering from Kun-Shan University of Technology, Taiwan, R.O.C. in 2004 and the M.S. degree in Electronic Engineering from the M.S. degree in Electronic Engineering from the National Chi Nan University, Taiwan, R.O.C. in 2006. She is currently pursuing Ph.D. degree in the National Chi Nan University. Her research interests are speech recognition and image signal processing.