CHAPTER 4 ARCHITECTURE FOR LOW CALCULATION COMPLEXITY OF TONE RECOGNITION

This chapter introduces a low calculation complexity tone recognition and its parallel architecture. We describe how to convert series computation of tone recognition to parallel computation. Subsequently, we explain parallel architecture of each tone recognition process.

Many research has develop high performance tone recognition system. However, the research is achieved by software implementation. This thesis aims to design automatic tonal speech recognition (ATSR) amenable for portable equipment.

In order to achieve tone recognition system in hardware, we discuss the gap between software implementation and reconfigurable computing. Software implementation uses general purpose microprocessor (GPP) such as personal computer CPU for computing software instructions. That makes users easily to interface the GPP and to re-design instruction code. Cost of development is thus low. However, GPP can cause high power consumption because all functionality must be executed from a set of instructions. If we run complex instructions, consuming very high power trades off the GPP advantages. In addition, GPP is not suitable to develop portable equipment because the personal computer size is too big.

In order to design tone recognition for hardware, we consider the use of programmable hardware such as GPPs, DSPs, ASICs, and FPGAs. Figure 4.1 compares the advantages and disadvantages of GPPs, DSPs, FPGAs and ASICs [66]. This provides an overview and highlights the general trends and characteristics.

architecture	performance	power	flexibility	design time / NRE	chip cost
GPP	- -	- -	++	++	- -
DSP	-	-	+	++	-
FPGA	+	0	++	+	0
ASIC	++	++	- -	- -	++

Figure 4.1 Comparison of architectures for programmable hardware [66]

GPPs and DSPs performance are limited because they are not generally a variable solution as a stand-alone system. FPGAs can offer similar performance to ASIC. Modern FPGAs also support dedicated DSP cores which help to accelerate arithmetic functions. In term of power consumption, although FPGAs consume more power than in ASIC, the FPGAs can be designed and also programmed/tested into the FPGA immediately. In term of design time, FPGA is more complicated than GPP and DSP while ASIC design requires extensive testing before tape-out which no changes can be made. The costs for the individual chip are very low for ASICs while DSPs are too expensive for most consumer applications. FPGAs lack competitive pricing because they are usually not sold in very high volumes. However, the same FPGA can be used in several different designs so that inventory cost is reduced.

Reconfigurable computing hardware, FPGAs, is therefore considered. FPGAs can be designed in granularity level so that it can limit to be a smallest function for hardware executing. Pre-design circuit of FPGAs can design optimized chip. That makes FPGAs lead to wasted hardware and also meet low power consumption. Moreover, the hardware can calculate some algorithms in parallel while software can run only sequential computation. The hardware can execute the functional algorithms faster than software simulation.

The challenges of FPGA implementation [67] are described as follows;

- FPGA design requires many level of detail when we verify an idea. We have to design the architecture especially for particular functions and then convert the architecture into a logic design.
- The architecture should be designed for reducing cost. Therefore, designing low power consumption and device performance should be considered.

In order to design low power tone recognition suitable for portable device, this thesis reduces tone recognition calculation complexity as described in Chapter 3. While continuous tone recognition is more applicable in natural communication than isolated tone recognition [68]-[69], it requires high computation cost. We therefore reduce the calculation cost by using only estimated vowel parts as an input of tone recognition. In addition, we apply vowel magnitude different function (V_{MDF}) for feature extraction process. The V_{MDF} can be operated without multiplication, which requires high time consuming.

We therefore compare the number of operations using in feature extraction process, the highest calculation cost. The proposed method applies V_{MDF} for feature extraction process, while the conventional methods use autocorrelation (*AC*) [54] and average magnitude difference function (*AMDF*) [49]. Table 4.1 shows the number of operations used in a frame where frame length *N* is 256. The V_{MDF}, *AC*, and *AMDF* can be defined by:

$$V_{MDF}(j,n) = \sum_{i=1}^{2M} \left| s(x+i) - s(x+i+n) \right|$$

where $s(i)$ is an input vowel signal at the *i*-th sample, $x = (j-1)M$, and *n* is a lag value $n = [1, 2,..., N]$

$$AMDF(j,n) = \frac{1}{N} \cdot \sum_{i=1}^{2M} \left| s(x+i) - s(x+i+n) \right|$$

where $s(i)$ is an input vowel signal at the *i*-th sample, $x = (j-1)M$, and *n* is a lag value $n = [1, 2,..., N]$.

$$AC(j,n) = \sum_{i=1}^{2M} (\mathrm{sgn}[\tilde{s}(i)] \cdot \mathrm{sgn}[\tilde{s}(i+n)])$$

where *n* is a lag value $n = [1, 2,..., N-i]$ and the function of $\mathrm{sgn}[\tilde{s}(i)]$ is a center clipping which is defined as:

$$\mathrm{sgn}[\tilde{s}(i)] = \begin{cases} 1, s(i) > T_c \\ -1, s(i) < -T_c \\ 0, otherwise. \end{cases}$$

Table 4.1 The number of operations in a frame

Tone Recognition Method	Number of Operations			
	Add	**Sub**	**Mul**	**Abs**
AC	255	-	256	-
AMDF	255	256	1	256
V_{MDF}	255	256	-	256

In Table 4.1, addition (add), subtraction (sub), multiplication (mul) and absolute (abs) operations are used to compare tone recognition complexity. AMDF has one multiplication more than the number of operations of V_{MDF}. While AMDF and V_{MDF} require more operations than AC method, the add, sub, and abs operators require only a clock cycle to complete. Assume that x and y are the input signals in binary bit, where x = '0001 1010' and y = '0001 1111'. Multiplication has to take 4 cycles to process x×y, while other operations, add, sub, and abs, require only a cycle to operate.

Since multiplication requires more clock cycle than other operations, AC requires higher processing time than the other methods.

If we use the entire input syllable to extract F_0 features in *AC* and *AMDF*, they require high computation cost. V_{MDF} reduces the number of input frames using only vowel signals as the input of tone feature extraction. This thesis applies V_{MDF} to our feature extraction process. This not only reduces the number of input frames but also provides the least number of clock cycles to operate all operators.

Although V_{MDF} method provides low calculation cost, it still operates in series. This may require high computation when we apply it in large vocabularies. This results in tone recognition speed reduction.

This thesis therefore reduces tone recognition complexity using parallel architecture. We develop the tone recognition in pipeline and parallel architecture in order to achieve a high throughput and accelerate the tone recognition. The architecture is operated in 32 parallelisms of three process elements. Each process element (PE) is executed in pipelined processing. We describe a concept of series to parallel computation in Section 4.1 and explain the parallel architecture of the tone recognition in Section 4.2.

4.1 Low Calculation Complexity Tone Recognition

The proposed V_{MDF} reduces the number of frames by using only vowel signals which are the input of the tone classifier. Figure 4.2 shows a series computation flow chart of

the tone recognition method. In this flow chart, there are three loop iterations. Loop A and Loop B calculates V_{MDF} *(j,n)*. Loop A repeats $2M$ times and Loop B repeats the frame length (N). In this thesis, we assign $N = 256$ and $M = 128$. Loop C finds $P_v(j)$ and then calculates $F_0(j)$. Due to V_{MDF} *(j,n)*, the number of frames j is lower than ordinary tone classifications where j is $j = 1, 2, \ldots, k$ and k is the frame ending.

While the calculation cost of the series tone recognition can reduce the number of frames, it still requires $2M \times N \times k$ repetitions to complete all of the whole tone recognition. This thesis redesigns the series tone recognition method in Figure 4.2 to the parallel method in Figure 4.3. Figure 4.3 converts Loop A, B, and C of the series computation into 32 parallelisms providing an effective acceleration. The reason is that the features in Loop A are conducted individually using parallel computation in each i where $s(i)$ is an input vowel signal at the i-th sample.

The V_{MDF} value of each frame is executed N times. The F_0 estimation process is calculated by Loop D. In this process, V_{MDF} values are sent to find P_v values and calculate F_0. Loop D repeats Y times, where $Y = \lceil k/32 \rceil$. Loop A and Loop D can also be scaled (extended/reduced) depending on the values of Y. This makes our tone recognition method easier to apply with different input speech.

To apply tone recognition for portable equipment, the proposed method aims to reduce the calculation complexity of tone recognition system. The lower the number of repetitions, the less calculation complexity. The reduction of each process is described as follows:

1. In feature extraction process, V_{MDF} reduces the number of repetitions from $2M \times N \times k$ to $N \times k$ times. Loop A checks whether input data *s(i)* is loaded to all 32 V_{MDF} calculation processes. Note that V_{MDF} is calculated separately in V_{MDF} (j). Loop B repeats *N/32* repetitions to calculate V_{MDF} in each frame. Afterwards, Loop C repeats V_{MDF} calculation until frame ending k.

2. In the F_0 estimation process, P_v, and F_0 are calculated within $Y = \lceil k/32 \rceil$ times where $\lceil\ \rceil$ is the ceiling of the value of Y. They are concurrently processed in parallel. Therefore, the repetition is reduced from k to $\lceil k/32 \rceil$

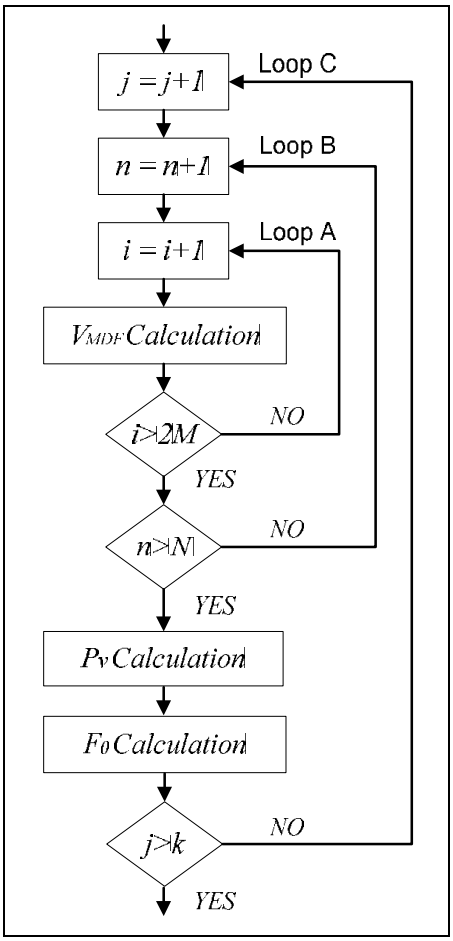

Figure 4.2 Series computation of tone recognition

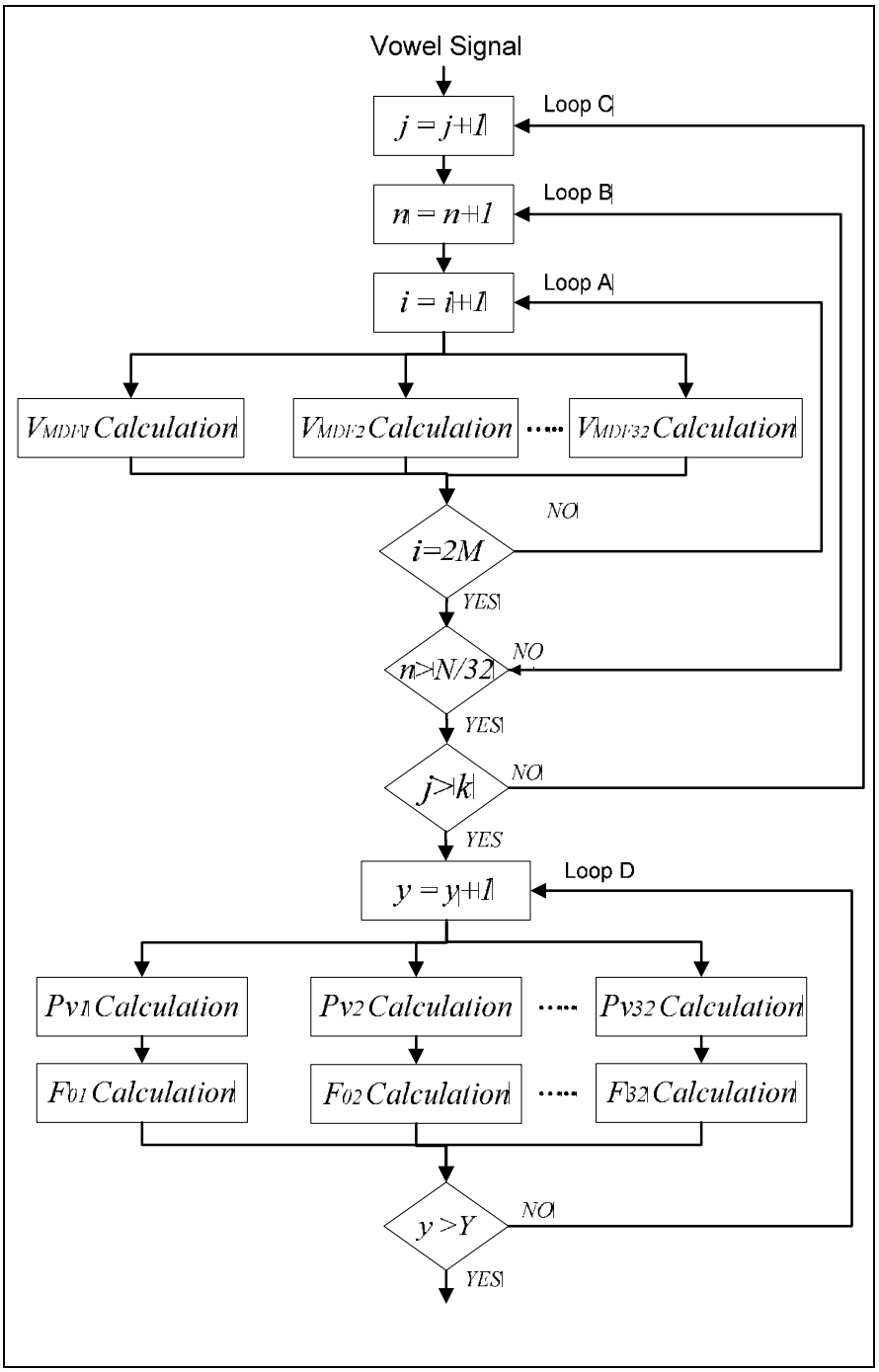

Figure 4.3 Parallel computation of tone recognition

In this thesis, j is evaluated between 6 and 104. Due to the proposed parallel computation, the tone recognition reduces the complexity of tone recognition, in the feature extraction process, from 256×256×102 to ((256/32)×102)+256 repetitions, assuming k is the maximum number of frame 102. This is a 99.9% reduction. In the F_0 estimation process, the proposed tone recognition reduces the number of frames from 102 to 4 repetitions which is a 96.1% reduction.

4.2 Parallel Architecture of Tone Recognition

Parallel architecture of a low calculation tone recognition is designed as 3 process elements shown in Figure 4.4. Each process element (PE) is executed in pipeline processing. A data allocation allocates input data *x(n)* to *s(i)*. PE1 executes V_{MDF} (1) to V_{MDF} (32), given by Equation (9), in a 4-stage parallel and pipeline processing and then stores the V_{MDF} values in the buffer. Subsequently, PE2 finds P_v, defined by Equation (10) to Equation (12), by using the 3-stage pipeline process. Finally, the F_0, given by Equation (19) to Equation (20), is calculated by a 2-stage pipeline process in PE3.

Figure 4.5 shows the internal structure of V_{MDF} calculation in PE1. There are 32 V_{MDF} calculations. In each V_{MDF} calculation, the input data of input data is allocated to a data allocation process. The arithmetic operations (*O*) generally require the 2 input data *x(n)* to execute each operation. In this thesis, the proposed data allocation (DA) allocates the input *s(i)* from the data *x(n)*. In PE1, *x(n)* and *s(i)* are of size 8 bits while the output accumulation of V_{MDF} is of size 14 bits. The difference of 6 bits between the input and output sizes is due to the number of inputs *s(i)* being 2^6.

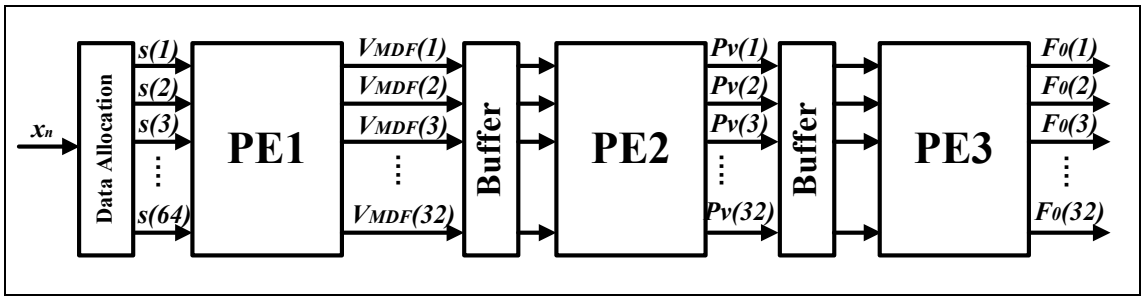

Figure 4.4 Parallel architecture of proposed tone recognition

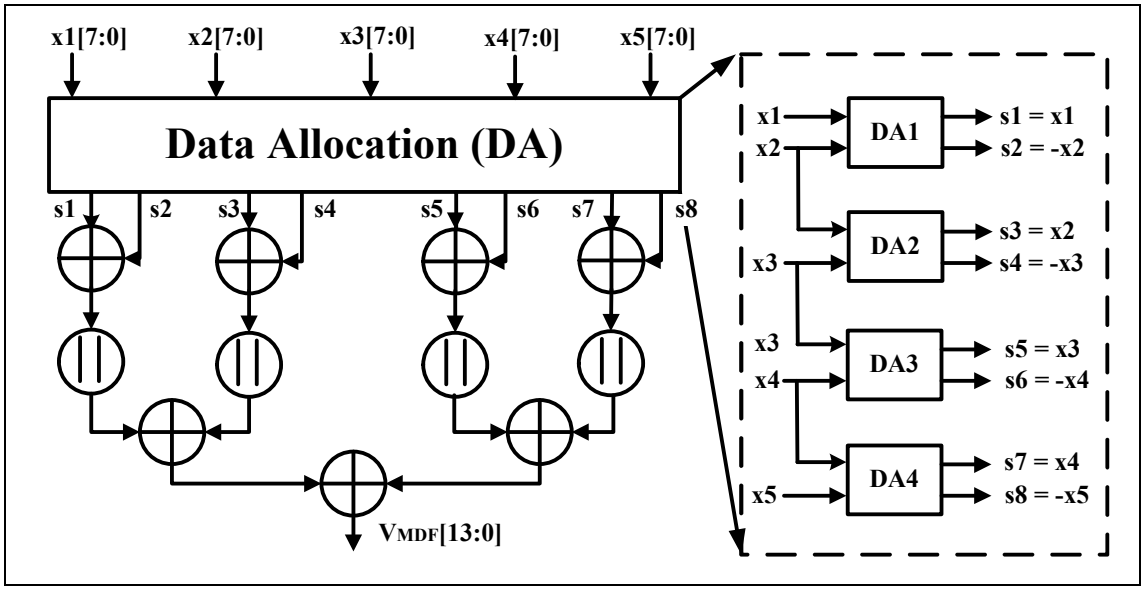

Figure 4.5 Structure of process element 1 (PE1) and data allocation

Table 4.2 Processing order of PE1

Times (ns)	Input	Stage1	Stage2	Stage3	…	Stage6	Stage7
0	0						
10	s1:s64	A1=(s1-s2) : A32=(s63-s64)					
20	s65:s128	A1=(s65-s66) : A32=(s127-s128)	|A1| : |A32|				
30	s129:s192	A1=(s129-s130) : A32=(s191-s192)	|A1| : |A32|	B1=(|A1|+|A2|) : B16=(|A31|+|A32|)			
40	s193:s256	A1=(s193-s194) : A32=(s255-s256)	|A1| : |A32|	B1=(|A1|+|A2|) : B16=(|A31|+|A32|)			
50			|A1| : |A32|	B1=(|A1|+|A2|) : B16=(|A31|+|A32|)			
60				B1=(|A1|+|A2|) : B16=(|A31|+|A32|)		E1=(D1+D2) E2=(D3+D4)	
70						E1=(D1+D2) E2=(D3+D4)	sum1=E1+E2

In order to execute all operations O_n, the DA allocates input s_i from only $(I/2)$ - 1 where I is the number of operands which is assigned to 64 for calculating n = 32 operations. The DA process is shown in the dashed box of Figure 4.5. As a result of the proposed data allocation, the data allocation process reduces the number of clock cycles from 64 to 33 per a repetition in Loop A shown in Figure 4.3. There are 4 pipeline processing stages: addition, absolute, and accumulation in PE1. The PE1 thus processes the number of operations $O(n + n/2 - 1)$ in the number of clock cycles $T(\log_2 n + 1)$. This thesis assigns frame length $N = 256$. Since we propose 32 parallelisms, in each frame, we need 8 repetitions for Loop B.

Table 4.2 shows processing order of PE1. Since PE1 needs 6 stages of pipelining, we have to wait for 6 clock pluses for the output to manifest at V_{MDF}. If each clock pulse takes 10 ns to achieve, then the output is available after 60 ns. This delay is referred as the latency which is 60 ns in this process element. PE1 therefore receives the output V_{MDF} of each parallel at 70 ns.

PE2 calculates P_v from Equation (10) to (12) as shown in Figure 4.6. In order to calculate $\overline{P(j)}$, there are 32 parallelisms used for calculating PE2. Each frame requires 256 V_{MDF} samples for calculating $\overline{P(j)}$. We therefore need $O(n\text{-}1)$ by using $\log_2 n + 1$ cycles. Then we find T_p using the number of operations $O(n + n/2 + 2)$ with $\log_2 n + 4$ cycles. Finally, we compare the input V_{MDF} with the value of $T_p(j)$. If $V_{MDF}(i)$ is less than $T_p(j)$, its position is decided to be $Pv(j)$. $P1$ and $P2$ are subsequently assigned to PE3.

In Figure 4.7, PE3 executes F_0 (j) given by Equation (20) from 2 stages. The first stage finds l_0 of each input frame from 2 operands P_1 and P_2. Then F_0 (j) feature is calculated from PE3 stage 2. The PE3 is executed $O(3 + 6 \cdot (n-1))$ in $6 \cdot \log_2 n + 3$ cycles.

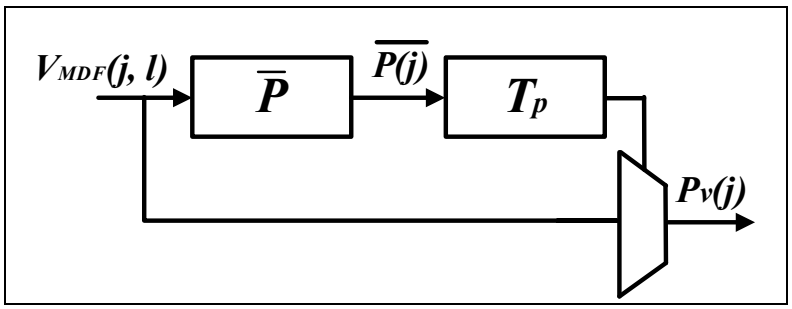

Figure 4.6 Process element 2 (PE2)

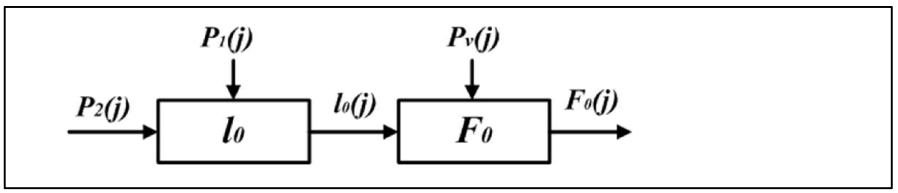

Figure 4.7 Process element 3 (PE3)

Total number of clock cycles in each process is shown in Table 4.3. In Table 4.3, we compare the number of cycles processed in a frame between series and parallel computation, where frame length is equal to 256. The architecture presents 14,656 total number of clock cycles per frame. This results in 97.8% reduction in the number of cycle per frame.

Table 4.3 Number of cycles processed in a frame

Process Element	**Number of Cycles**	
	Parallel V_{MDF}	**Series V_{MDF}**
PE1	14,336	458,752
PE2	136	4352
PE3	184	5888
Total	**14,656**	**468,992**