PREFACE

State-of-the-art speech recognition, speech analysis as well as music modeling have approached Mel-Frequency Cepstral Coefficient (MFCC) and confirmed great performance in comparison to other feature extractions. Based on obtained software performance, a wide range of hardware designs are applied to highly increasing integrated systems achieving real-time performance and ability of mobility. Nevertheless, most hardware approaches witnessing certain configurations have experienced limitation of functions due to fixed-point format, strict silicon requirements or exact applications, which is reasonable for low ability of reusing and high cost of product. As regards MFCC method, there are various concerning parameters such as number of samples, range of filter bands, Fast Fourier Transform (FFT) points, number of cepstrals or even different level of delta coefficients, which significantly affect final performance of entire applications.

As a result, a dynamic ASIC-based MFCC hardware architecture is proposed in this thesis in order to meet real-time system requiring high performance as well as confirm superiorities regarding to ability of reconfiguration feasibly through an Advance High-performance Bus (AHB) interface in chip level instead of modifying at Register Transfer Level (RTL) in developed duration. Besides, have not only experiments on 130nm technology with full ASIC design flow witnessed high frequency at 500MHz but applying IEEE 754 floating-point format has also confirmed great accuracy between hardware design and software design, which apply in certain application towards Vietnam automatic speech recognition (ASR).

CONTENTS

FIGURE LISTS

TABLE LIST

CHAPTER 1: THE OVERVIEW OF RESEARCH

1.1 The overview of research

MFCC has become highly popular as regards ASR systems or music modeling methods due to high performance of audio feature extraction, which has experienced in Table 1.1 with conducted survey from [1] to [7]. However, most approaches have applied software-based designs to be able to achieve flexible configurations, which is also reasonable for low performance of speech, reducing ability of integration or even witnessing limitation of real-time systems as considerable disadvantages. Indeed, in [1] and [3] indicated that MFCC was the most classic hardware architecture for ASR systems. Another design proposed by [6] also confirms that dynamic MFCC increases 5 to 6 percent of the recognition rate than a fixed one. Additionally, [10] and [11] have same problem posing a trade-off between energy consumption and system performance, which need to be solve as future work.

Although the contrasting trend performing on hardware-based designs can tackle such issues, other concerning problems associating with ability of reconfiguration, silicon requirement as well as how to achieve high accuracy based on hardware view point with a vast number of operations are considered carefully.

Table 1.1 Statistics Towards MFCC And Applications

Author	Implementation	Method	Identification Algorithm	Conclusion	Application
[1]	Hardware (FPGA) And Software (C++)	MFCC And LPC And ENH-MFCC	HMM	MFCC better than LPC	Automatic Speech Recognition system (ASR system)
[2]	Software	LPC And MFCC	ANN	MFCC better than LPC	Automatic Language Identification(Arbic, English, French)
[3]	Software	MFCC	HMM	MFCC is more	ASR system

				dominant used	
[4]	Software (Matlab)	MFCC	DTW	-	Voice Identification (Security system)
[5]	FPGA	MFCC	HMM	MFCC is extensively used	ASR system
[6]	Software	MFCC (Dynamic)	GMM	Dynamic MFCC better than 5% to 6%	ASR system with real noisy environment
[7]	-	General	General	MFCC is used mostly	ASR system with Marathi Language

Table 1.2 Different MFCC configurations on hardware

Author	**Implement on**	**Sample in frame**	**FFT point**	**Mel filter**	**Cepstral**	**Delta's order**
[8]	FPGA (Xilinx)	1024	1024	24	24	-
[9]	ASIC (0.6µm)	256	256	20	12	-
[10]	ASIC (130nm) And FPGA (Xilinx)	256	256	32	13	2
[11]	ASIC (0.18 µm)	256	256	-	12	1 & 2
[12]	FPGA	160	256	33	12	2

[13]	FPGA (Xilinx Spartan)	512	512	24	12	-
[14]	FPGA	128-256	128-256	24	12	-
[15]	FPGA (Xilinx)	256	256	-	17	1

Table 1.3 Range of parameters in proposed MFCC

Parameter	**Proposed MFCC**	**Reference MFCCs [8-15]**
Pre-emphasis coefficient	Any value	0.97, 0.975
Sample In Frame	25 to 1024	128, 160,256, 512, 1024
Overlap rate (%)	30 to 70	50
FFT Point	25 to 1024	128, 160,256, 512, 1024
Mel Filter number	1 to 63	20, 24,32, 33
Number of cepstrals	1 to 31	12,13,17,24
Delta's Order	Level 1 and Level 2	Level 1, level 2

Particularly, Table 1.2 presents collected data relating to not only performance of hardware-based MFCC but also limitation of applications. Most approaches in Table 1.3 present fixed configurations, which is compatible with exact applications. Another aspect that needs to be concerned that such hardware targets are usually built on FPGA to be able to reconfigure feasibly that is cause of low performance in comparison to ASIC-based designs. From such circumstances, a high performance dynamic ASIC-based MFCC architecture in 130nm technology is proposed in this work so to halt almost issues relating to hardware approaches. To be more specific, utilizing diverse range of techniques such as IEEE 754 floating-point format, Booth algorithm, internal memories, multi state machines, ability of setting parameters through AHB interface and pipeline technique makes effort to achieve high performance in comparison to other designs. Besides, ASR system integrated proposed MFCC hardware is also completed to confirm high performance of such architecture.

According to tables above, the characteristic configuration of MFCC method can be generalized with following features:

- MFCC configurations as overlap rate, FFT point, cepstral or delta's order are different for each purpose of system.
- Different languages as well as number of word can change MFCC configurations.
- Almost of MFCC hardware architecture researches often built fixed configurations.
- MFCC hardware architecture researches used fixed bit number, which can be more results that are inaccurated.
- MFCC architecture with limited configurations is applied only to recognition systems that have small rate at number of word.

With the trend of research and implementation of hardware architecture as we concerned. There are two way to design exist in the current. Firstly, the flexibility in MFCC configurations is bound closely by specific applications for recognition of different patterns or different voices in many countries. Secondly, the accessible method for hardware design is limited by problems, which related to the accuracy and resources.

In order to solve these two issues, a purposed MFCC hardware architecture with the ability to set the value of a number of sampling points on one frame, frame length, overlap rate, FFT points, number of cepstral and delta's order is implemented. Last but not least, a dynamic hardware architecture can solve effectively most of voice recognition systems.

CHAPTER 2: THE HARDWARE ARCHITECRURE OF MFCC

In Chapter 1, the accessible MFCC method has already proved the creating successful platform for the entire voice recognition system because this method has proven its effective performance in most applications as well as research surveyed. The MFCC theory will be presented in this chapter with its algorithm and configuration, so that we can see how the important of dynamic hardware architecture is in this research.

2.1 The MFCC Mathematic Model

Block diagram of MFCC

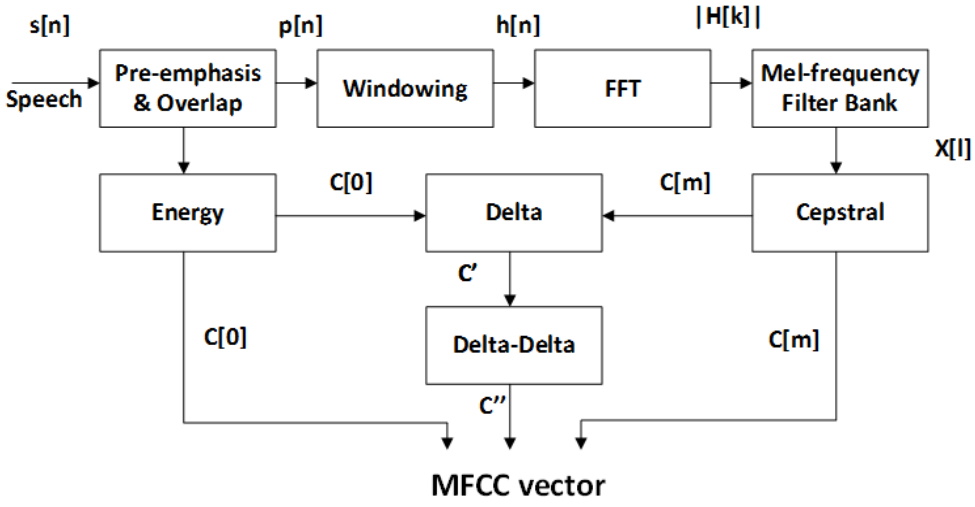

Figure 2.1 Block diagram of MFCC extraction method

Theoretical model in Figure 2.1 shows the basic characteristic blocks of MFCC algorithm. Each function block will be analyzed and built hardware architecture from the theoretical descriptions.

2.1.1 Pre – emphasis

Pre_emphasis block is used to amplify the signal at high frequencies. In time domain, relationship between output and input is shown in (1), with $s[n]$ is the nth sample of speech signal and $p[n]$ is the nth sample after going through Pre- emphasis filter. After filtering by Pre- emphasis, n samples are separated into many frames with different number of sample per frame and different percentage of overlapping which based on setting data.

$$p[n] = s[n] - 0.97 \cdot s[n-1] \qquad (1)$$

Frame Blocking

Because voice signals change slowly over time, in a voice recognition system, voice will be segmented into short period, which called frame. In order to frame voice signals the overlap parameters change, usually with 50% overlap between mutual frames.

In voice recognition system of software, voice is divided into 20ms frame length with 10ms overlap. When the voice is sampled at 8 kHz, there are 160 samples per frame and 80 samples overlap between two mutual frames. However, 50% overlap rate is not always used, the value of 40%, 60% or even 70% are also considered to increase recognition results effectively. On the other hand, some systems do not use the 8 kHz sampling frequency because the studies show that some characteristics located in the higher frequency domain.

2.1.2 Windowing

Window filter are applied to increase the continuity characteristic between mutual frames. One of the most window filter commonly used in speech recognition is Hamming window, where N is the length of the window and it is equal to the length of the frame and $h[n]$ is defined as the result of this block.

$$h[n] = p[n] \cdot \left\{0.54 - 0.46 \cdot cos\left(\frac{2\,\pi\,n}{N-1}\right)\right\} \qquad (2)$$

However, when the number of point on a frame change, the value of the filter coefficient window will also change. Therefore, the fixed values of the filter coefficients in the hardware window filter means that a fixed number of points on frames.

2.1.3 FFT algorithm

Fast Fourier Transform (FFT) is used to transfer signal from time domain to frequency domain. Actually, it is Discrete Fourier Transform (DFT) which achieves high performance with certain conditions that spectrum are evaluated at discrete frequencies. FFT algorithm only requires computational complexity ratio of $N \log N$, whereas DFT calculations require the bigger ratio of N^2. FFT transform is one of the most time consuming phases as well as the resources of the system. Hence, FFT hardware approach will effect highly to the entire MFCC architecture. At final state of FFT, DFT results in $H[k]$ from input $h[n]$, then $|H[k]|$, magnitude of $H[k]$ in each frame, is calculated as (3).

$$H[k] = \sum_{n=0}^{N-1} h(n) \cdot e^{j\frac{2\pi n}{N}}$$

$$|H[k]| = \sqrt{\left(Re(H[k])\right)^2 - \left(Im(H[k])\right)^2} \quad (3)$$

FFT transform is one of the most time consuming steps as well as the resources of the system. Hence, the FFT hardware approach will effect highly to the entire MFCC architecture. Some hardware FFT architecture approaches are mentioned in Table 2.1.

Table 2.1 Research paper list of FFT hardware architecture configurations

Author	Configuration	Implement on	Algorithm	Latency
GIN-DER WU [17]	256 points Radix 2	ASIC (0.18µm)	Butterfly	10,4 µs
Chin-Teng Lin [18]	256 points Radix 16	ASIC (0.13µm)	Pipelined	-
Dongsuk Jeon [19]	1024 points Radix 4	ASIC(65 nm)	Pipelined	6,7 µs
Lihong Jia [20]	128 points Radix 2/4/8	ASIC (0.6µm)	Pipelined	3 µs
Atin Mukherjee [21]	8 points Radix 2	FPGA (Xilinx Virtx-6)	Butterfly	19,598 ns
Jungmin Park [22]	64 – 8K points Radix 8	FPGA (Xilinx Virtex-5)	Butterfly	0,33 µs (64 points) 96,20 µs (8K points)
K. Umapathy [23]	128 points Radix 2/4	ASIC(90 nm)	Pipelined	40 µs
Ediz Çetin [24]	256 points Radix 2	ASIC (0.7µm)	Butterfly	102,4 µs

2.1.4 Mel - Scale Filter

A tape of digital filters used to model the original story of the transformation of the human auditory system for two reasons. Firstly, the position of maximum displacement along the vibrating membrane in the human ear are in proportion to the logarithm of the voice frequency. Secondly, the complex voice frequencies inside a determined band include a few nominal frequency, which cannot be realized individually.

Bandwidth filter of conventional Mel scale in voice recognition includes many triangular band-pass filters which are distributed within a bandwidth signal. The logarithmic power spectrum on the Mel-scale is computed in order to use a filter bank consisting of L Mel filters. Where l = 0, 1, . . . , L − 1; $W_l[k]$ is the lth triangular filter; k_{ll} and k_{lu} are the lower and upper bounds of the l[th] filter, respectively.

$$X[l] = \log\left(\sum_{k=k_{ll}}^{k_{lu}} |H[k]| \cdot W_l[k]\right) \quad (4)$$

For speech recognition, spectral boundary is more useful than spectral components, which is reasonable for using inverse Fourier transform to find the boundary of spectrum.

The system of human auditory frequency is nonlinear with voice received, so that a Mel scale is used to map the audio frequency received into linear scale. Scale frequencies are defined by (5) and is illustrated in Figure 2.2. It approximated as linear scale ranging from 0 to 1000 Hz and approximated as the logarithm of frequency, which is over 1000 Hz.

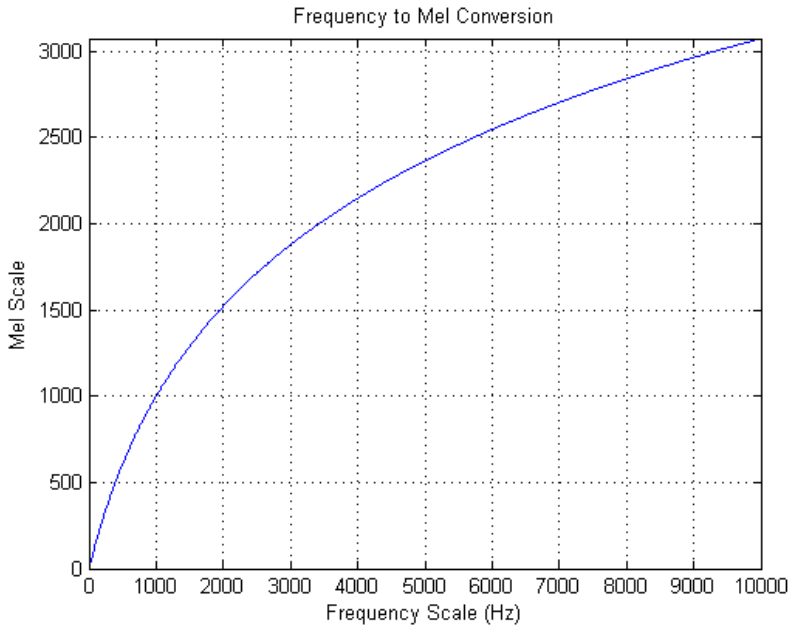

Figure 2.2 Mel scale with voice frequency signal

(https://courses.cit.cornell.edu/ece576/finalprojects/f2008/pae26_jsc59/pae26_jsc59)

$$Mel(f_{mel}) = 2595\ log_{10}\ (1 + \frac{f_{Hz}}{700}) \quad (5)$$

The bandwidth filter of conventional Mel scale in voice recognition includes several triangular bandpass filters, which are distributed within a bandwidth signal. They are distributed equally in Mel scale and their bandwidth is designed to 3db point and must be located between two adjacent filters as shown in Figure 2.3.

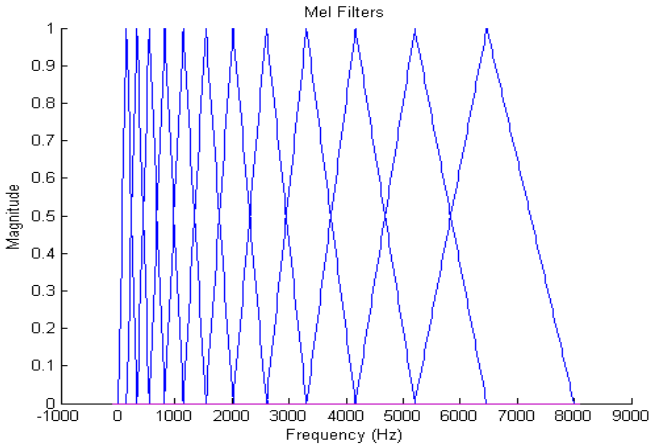

Figure 2.3 Mel frequency filter

(https://courses.cit.cornell.edu/ece576/finalprojects/f2008/pae26_jsc59/pae26_jsc59)

When the number of filters decreases, the values at higher frequencies is not calculated through the filter. Fixed or flexible filter coefficients are important issues in hardware approach because the changing these parameters will effect seriously to the system performance. Another important issue is the accuracy of the filter, triangle filters used with decimal coefficient will reflected performance issues when using the approximate value method.

2.1.5 Cepstral - analysis

The response of the vocal bunch decide boundary of spectrum, whereas started spectrum of the signal component representing the spectrum of voices. For voice recognition, spectral boundary is more useful than spectral components, so we can use inverse Fourier transform to find the boundary of spectrum.

Cepstral is defined as the inverse Fourier transform of the power factor after taking the logarithm. It can be simplified as DCT transformation as (6).

$$C[m] = \sum_{l=1}^{L} X[l] cos\left(\frac{\pi\, m\, (l - 0.5)}{L}\right) \quad (6)$$

The low cepstral coefficients' degree reflected bundle sound information of voice signals. In the analysis of spectrum for voice recognition, we normally used between 8

and 16 low cepstral coefficients' degree, but in the majority of applications we have already used 12 cepstral coefficients.

The problem occurs when designing hardware approach like all detailed blocks above, the calculating of logarithm approximation methods as well as the coefficient of cos functions are matters of concern in this research.

2.1.6 Energy computation

Energy of each frame is also a typical component of MFCC. It is calculated as the logarithm of the signal power before pre-emphasis as (7), where $C[0]$ is the first component of *C*.

$$C[0] = \log\left(\sum_{n=0}^{N-1} s^2[n]\right) \tag{7}$$

2.1.7 Delta and Delta-Delta

The quality of the voice recognition system can be improved by adding more derivative features over-time to obtain the basic static parameters. In digital signal processing, first and second order derivative over time can be approximated by (8)

$$\begin{aligned} C'_i &= C_{i+2} + C_{i+1} - C_{i-1} - C_{i-2} \\ C''_i &= C'_{i+2} + C'_{i+1} - C'_{i-1} - C'_{i-2} \end{aligned} \tag{8}$$

Some system applied to add the second order delta formula to increase the system's accuracy. The chosen delta's order effected significantly to the quality of the voice recognition systems.

2.1.8 Conclusion

Model of theory as well as specific of MFCC algorithm are analyzed in detail to realize the advantages and disadvantages in the transform process of hardware architecture approach model. The problem is how the hardware architecture can

implemented with such flexibility in order to be able to fit most of the changes related to the number of voices, the sampling frequency as well as the characteristics of different region voice domain.

2.2 The Perspective of MFCC Architecture

2.2.1 Overall architecture

MFCC architecture can be completed like IP with ARM core as the core controls. MFCC hardware architecture to communicate with ARM core through common protocols such as APB in Figure 2.4.

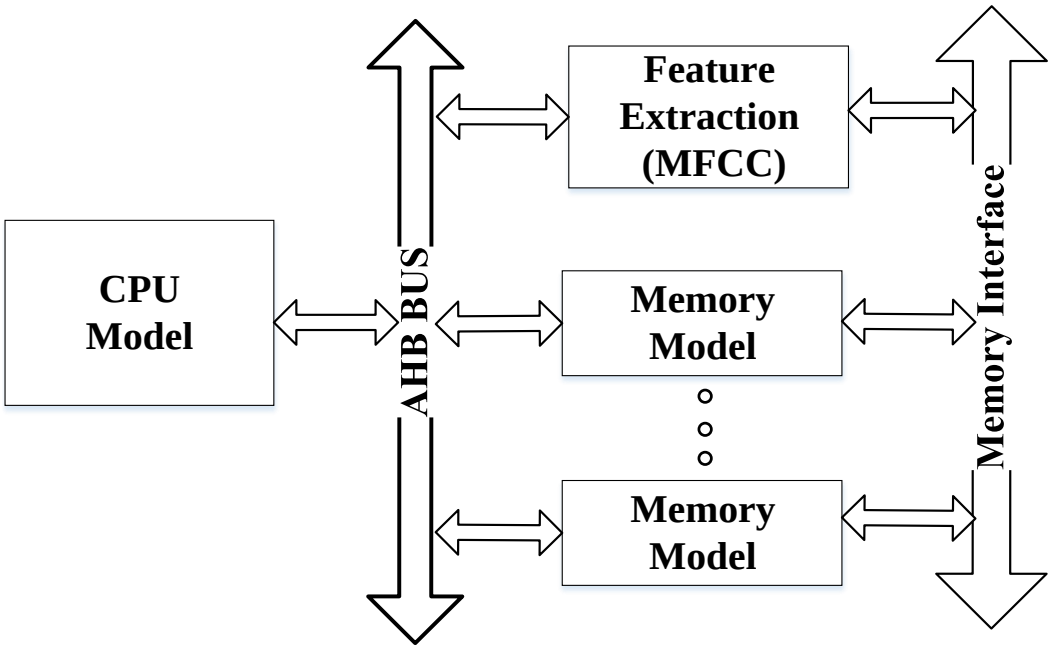

Figure 2.4 The MFCC's interface of voice recognition system

With APB bus interface, MFCC block will be set before the operation. The following Table 2.2 shows the desired parameters set for specific MFCC's configurations as well as the limits of these parameters.

Table 2.2 The dynamic configuration of MFCC and parameters limitation

Number of sample per frame (point)	Maximum is 1024
Overlap rate (%)	30-70
The "a" value	Arbitrary value
Number of sample per window filter (point)	Maximum is 1024
Number of FFT point (point)	Maximum is 1024
Number of Mel filter (filter)	Maximum is 63
Number of Cepstrals	Maximum is 31
Delta's order	Second order
Number of feature per MFCC vector	Maximum is 63

With the proposed configurations shows in Table 2.2, these parameters can be set for MFCC architecture block with a wide range of values can cover all the others have been reported in recent papers. Ability to set via the APB bus protocol allows us to set up all parameters of MFCC blocks before operating. On the other hand, hardware architecture approach based on 32-bit floating point in IEEE 754 standard allow us to set up any constant values which are decimal and can achieve the highest accuracy.

To be able to change the configuration of the MFCC hardware architecture, we need more intermediate memories to storage essential data. Since the operations in the proposed MFCC block used 32-bit floating point, Table 2.3 will describes the memory with the corresponding storage purpose.

Table 2.3 The configuration of memory with corresponding function

Quantity	Function	Capacity (word)
2	Memory storage the output of Pre-emphasis	4 K
2	Memory storage the output of Window	4 K

1	Memory storage the coefficient of Window	4 K
4	Memory storage the output of FFT	4 K
4	Internal memory	4 K
2	Memory storage the coefficient of FFT	4 K
2	Memory storage the output of Magnitude	4 K
2	Memory storage the output of Mel	4 K
4	Memory storage the coefficient of Mel	4 K
1	Memory storage the coefficient of Log	4 K
1	Memory storage the output of Cepstral	4 K
4	Memory storage of the MFCC vectors	4 K

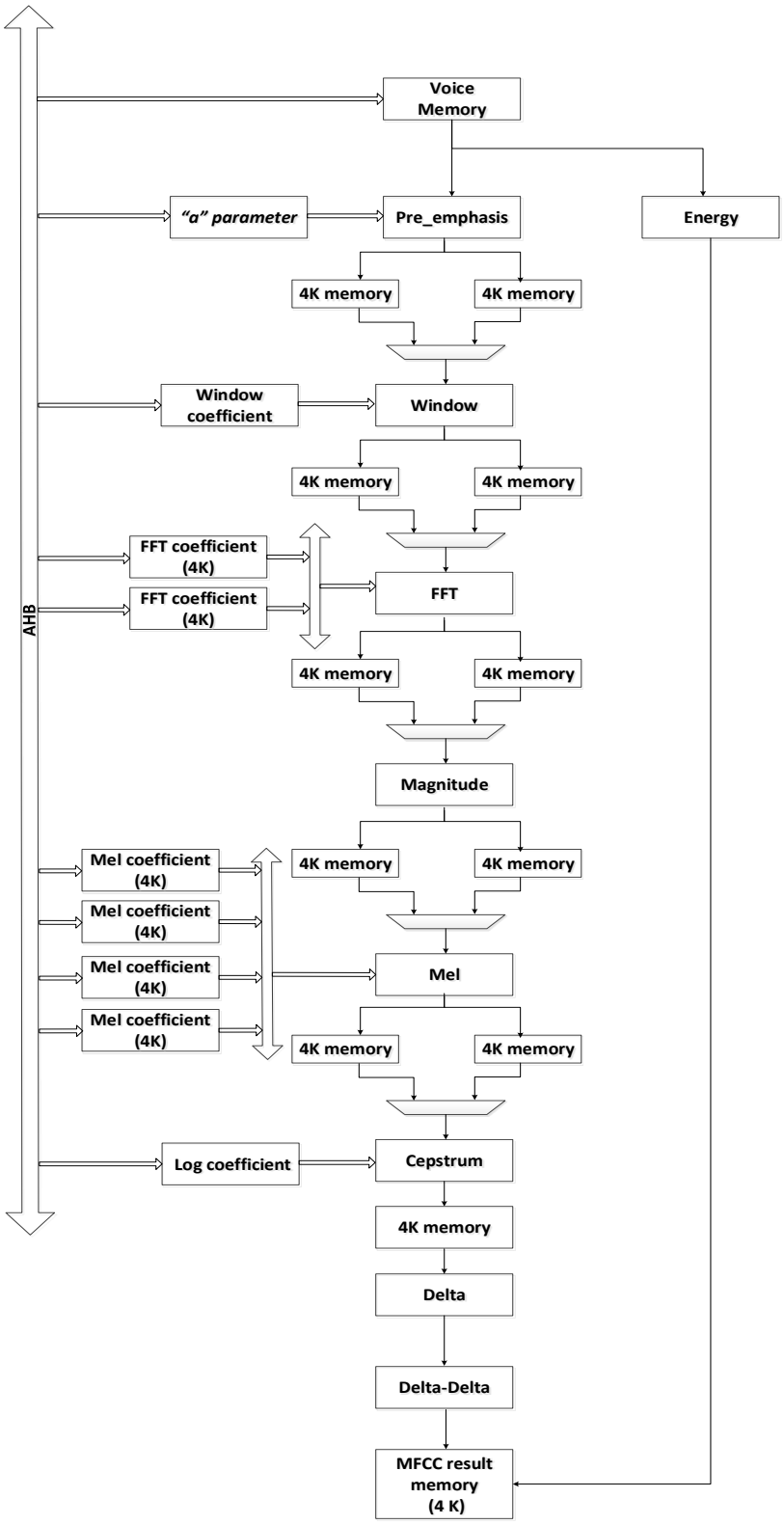

Figure 2.5 The detailed connection between all memories and blocks.

2.2.2 Data path and main control

2.2.2.1 Main control block

Main Control block control tasks keep all the calculation process. Here we use techniques to reduce the time pipeline calculations. Purposed architecture is divided into 6 stages of pipeline: Pre-emphasis, Window, FFT, Amplitude, Mel, Cepstral. In each block, separative state machines are activated to implement its algorithm and WAIT state to stop after each calculation is completed, and will be restarted that level when the enable signal appeared.

Main control block has been implemented via these tasks:

- Export the enable signal for all stage of this purposed architecture.
- Decide which memory of each stage should be at read or write mode. Each stage has at least 2 memories to stored the data, which has been calculated and provided data for next block in the process. Furthermore, Main Control provides the enable signal as the switch of all memories to make sure that the designed architecture can operates successgully between two consecutive stages.

The rule of the enable signals of the main control is very important because it synchronizes the system but some problems appeared during the process as follow:

Consider the example: At the P_W_F state, this state will implement simultaneously 3 stages pre-emphasis, Window and FFT so that all enable signal of them the is activated during P_W_F state. When the main control at P_W_F, it pre-emphasis‘s enable pin is 1, transfers the state from WAIT to the START at a new calculation stage. When, pre-emphasis is completed for 1 frame, but its enable signal is still 1 because the main control is currently in P_W_F state, so that it start once again with the same data calculation as before. This is the same problem with Window and FFT, so that 3 stage will run endlessly and separative state machine of each stage will not be executed follow the discretion of the main control.

To solve this problem, we design state machines like this one: the excited state activity levels only exist in one clock, mean that the enable signal for each stage is as the trigger. After an arbitrary trigger, the main control will move to WAIT, WAIT_N or WAIT_M to wait the over counter, it will go to an next stage and after 1 clock, come back to the one of waiting states, which have mentioned above.

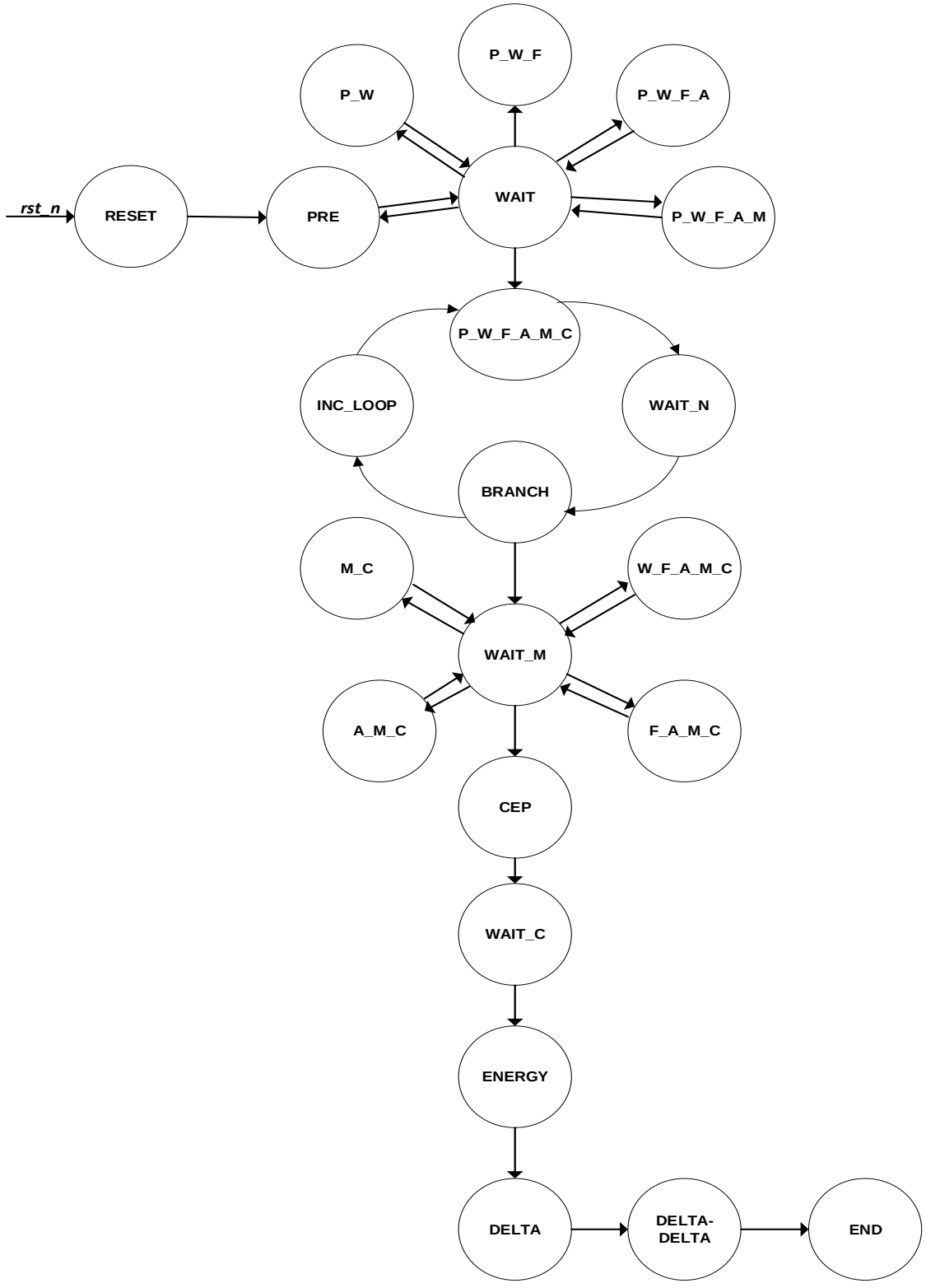

Figure 2.6 The state machine of purposed MFCC architecture

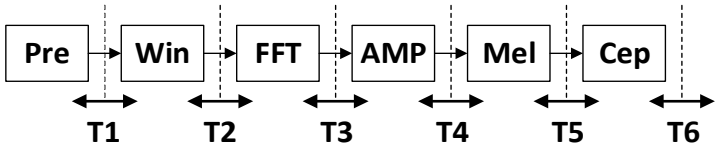

Figure 2.7 Timing consumption without pipeline

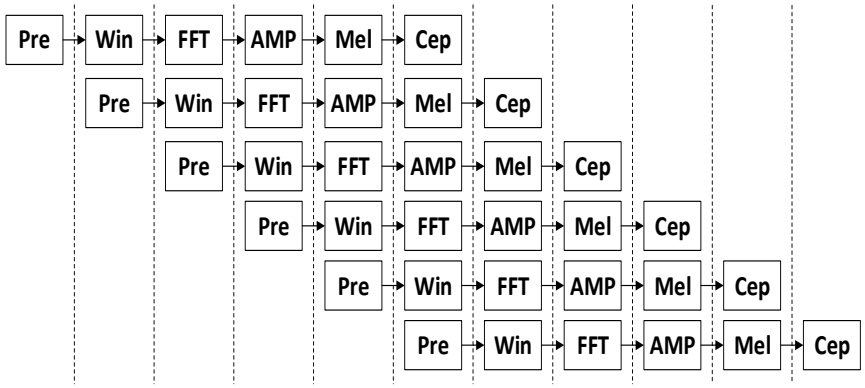

Figure 2.8 Pipeline technique

Based on MFCC model, if samples are transferred to the Pre-emphasis block firstly and then to next architectures, it consumes much time to solve consequentially before storing expected cepstrals to memory finally. Consequently, if we assume timing consumption per every block is from T1 to T6, total of timing consumption is calculated as Figure 2.7 and (8) with *n* frames of data, which negatively affect real-time criterion.

$$T = n * (T1 + T2 + \dots + T6) \qquad (9)$$

In order to tackle this issue, pipeline technique is applied to be able to execute all blocks at the same time. Particularly, only one timing parameter T_{Stage} established through AHB interface applied to all blocks to ensure that no block has to wait neighbor one. Moreover, because every block experiencing different formulas from section A for solving a frame of samples instead of one sample, internal memories is essential to store and load intermediate data. Hence, Figure 2.8 shows how to load and store data toward two neighbor blocks at the same time to ensure pipeline process correctly.

Table 2.4 The function of states in Main control block

	Present state	**Function**
1	RESET	Reset
2	PRE	Enable Pre-emphasis block
3	P_W	Enable Pre-emphasis and Window block
4	P_W_F	Enable Pre-emphasis, Window and FFT block
5	P_W_F_A	Enable Pre-emphasis, Window, FFT and Magnitude block
6	P_W_F_A_M	Enable Pre-emphasis, Window, FFT, Magnitude and Mel block
7	P_W_F_A_M_C	Enable Pre-emphasis, Window, FFT, Magnitude, Mel and Cepstral block
8	W_F_A_M_C	Enable Window, FFT, Magnitude, Mel and Cepstral block
9	F_A_M_C	Enable FFT, Magnitude, Mel and Cepstral block
10	A_M_C	Enable Magnitude, Mel and Cepstral block
11	M_C	Enable Mel and Cepstral block
12	CEP	Enable Cepstral block
13	DELTA	Enable Delta and Delta-Delta block
14	ENERGY	Enable Energy block
15	WAIT	General counter for all block
16	WAIT_N	General counter for all block at N loops
17	WAIT_M	General counter for all block at M loops
18	WAIT_C	General counter for final Ceptrum stage
19	BRANCH	Check the number of necessary loops for P_W_F_A_M_C state
20	INC_LOOP	Increase number of loops
21	END	Finish

Table 2.5 State machine in Main control block

	Present state	**Next state**	**Condition**
1	RESET	PRE	The top_state_en is activated
2	PRE	WAIT	After 1 clock
3	P_W	WAIT	After 1 clock
4	P_W_F	WAIT	After 1 clock
5	P_W_F_A	WAIT	After 1 clock
6	P_W_F_A_M	WAIT	After 1 clock
7	P_W_F_A_M_C	WAIT_N	After 1 clock
8	WAIT_N	BRANCH	Counter is over
9	BRANCH	INC_LOOP	After 1 clock
		WAIT_M	After 1 clock
10	INC_LOOP	P_W_F_A_M_C	After 1 clock

11	W_F_A_M_C	WAIT_M	After 1 clock
12	F_A_M_C	WAIT_M	After 1 clock
13	A_M_C	WAIT_M	After 1 clock
14	M_C	WAIT_M	After 1 clock
15	CEP	WAIT_C	After 1 clock
16	WAIT_C	ENERGY	Counter is over
17	ENERGY	DELTA	Counter is over
18	DELTA	END	Counter is over
19	WAIT	State + 1	Counter is over
20	WAIT_M	State + 1	Counter is over

Table 2.6 Output of states in Main control block

State	**Output of state**

RESET	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

WAIT	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;

	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= LOOP_WAIT;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

PRE	preem_state_en <= 1'b1;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

P_W	preem_state_en <= 1'b1;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;

	copy_energy_en <= 1'b0; finish_flag <= 1'b0;

P_W_F	preem_state_en <= 1'b1;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b1;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

P_W_F_A	preem_state_en <= 1'b1;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b1;
	amp_state_en <= 1'b1;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

P_W_F_A_M	preem_state_en <= 1'b1;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b1;
	amp_state_en <= 1'b1;

	mel_state_en <= 1'b1;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

P_W_F_A_M_C	preem_state_en <= 1'b1;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b1;
	amp_state_en <= 1'b1;
	mel_state_en <= 1'b1;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b10;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

WAIT_N	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;

	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b10;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= LOOP_WAIT;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

BRANCH	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

INC_LOOP	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b1;
	counter_value <= 20'd0;

	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

WAIT_M	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b10;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= LOOP_WAIT;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

W_F_A_M_C	preem_state_en <= 1'b0;
	window_state_en <= 1'b1;
	fft_state_en <= 1'b1;
	amp_state_en <= 1'b1;
	mel_state_en <= 1'b1;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b1;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

F_A_M_C	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b1;

	amp_state_en <= 1'b1;
	mel_state_en <= 1'b1;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b1;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

A_M_C	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b1;
	mel_state_en <= 1'b1;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b1;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

M_C	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b1;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;

	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b1;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

CEP	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b1;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b1;
	result_change_mem <= 2'b10;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

WAIT_C	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b10;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;

	counter_value <= LOOP_WAIT;
	inc_state_en <= 1'b0;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

DELTA	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b1;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b11;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= LOOP_DELTA;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b0;

ENERGY	preem_state_en <= 1'b0;
	window_state_en <= 1'b0;
	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b01;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	counter_value <= LOOP_ENERGY;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b1;
	finish_flag <= 1'b0;

END	preem_state_en <= 1'b0; window_state_en <= 1'b0;

	fft_state_en <= 1'b0;
	amp_state_en <= 1'b0;
	mel_state_en <= 1'b0;
	cep_state_en <= 1'b0;
	delta_state_en <= 1'b0;
	preem_win_change_mem <= 1'b0;
	win_fft_change_mem <= 1'b0;
	fft_amp_change_mem <= 1'b0;
	amp_mel_change_mem <= 1'b0;
	mel_cep_change_mem <= 1'b0;
	result_change_mem <= 2'b00;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	counter_value <= 20'd0;
	inc_state_en <= 1'b1;
	copy_energy_en <= 1'b0;
	finish_flag <= 1'b1;

Table 2.7 The function of states'input/output of Main control block

Pin name	**# Bit**	**Description**
preem_state_en	1	An enable signal allows Pre-emphasis block implements
window_state_en	1	An enable signal allows Window block implements
fft_state_en	1	An enable signal allows FFT block implements
amp_state_en	1	An enable signal allows Amplitude block implements
mel_state_en	1	An enable signal allows Mel block implements
cep_state_en	1	An enable signal allows Cepstral block implements
delta_state_en	1	An enable signal allows Delta block implements
preem_win_change_mem	1	An enable signal allows Pre-emphasis block read or write its results into memories
win_fft_change_mem	1	An enable signal allows Window block read or write its results into memories
fft_amp_change_mem	1	An enable signal allows FFT block read or write its results into memories
amp_mel_change_mem	1	An enable signal allows Amplitude block read or write its results into memories
mel_cep_change_mem	1	An enable signal allows Mel block read or write its results into memories
result_change_mem	2	An enable signal allows Energy, Cepstral and Delta block read or write final results into memories
counter_en	1	Enable pin of counter module

counter_loop_en	1	Enable pin of counter_loop which use to count number of loops
inc_state_en	1	An enable signal allows register of state is updated
copy_energy_en	1	An enable signal allows Energy block implements
finish_flag	1	MFCC has been finished completely

2.2.2.2 Pre-emphasis block

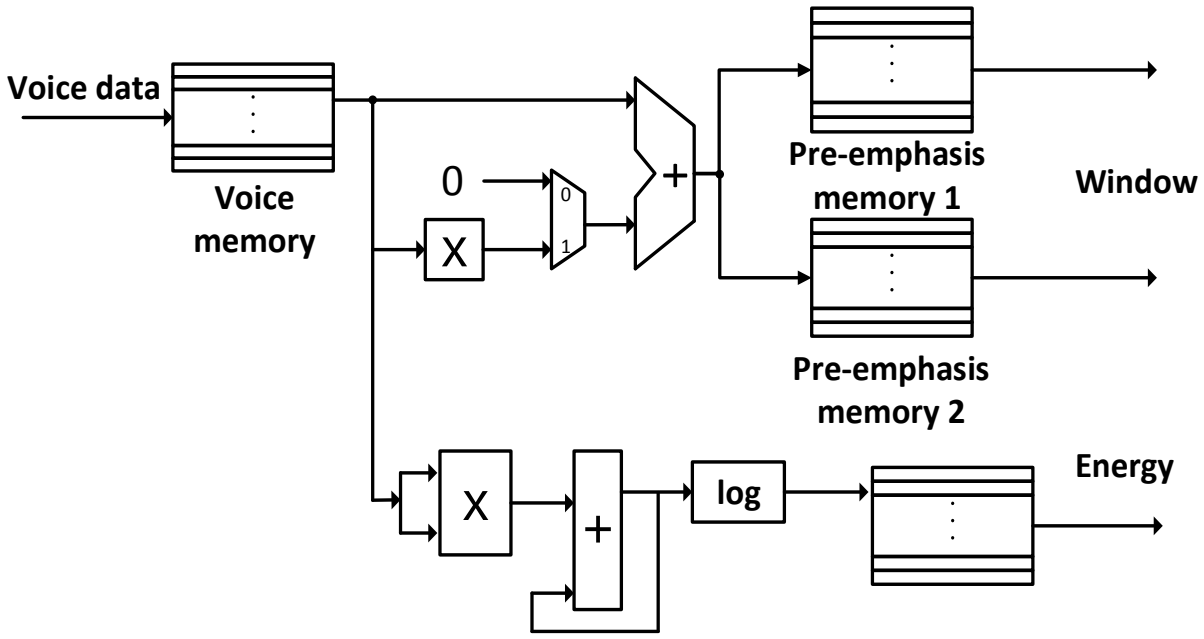

Figure 2.9 Pre-emphasis block

Pre-emphasis block is implemented with the sub-frames by the overlap ratio as shown in the theory. Due to the characteristic of this dynamic architecture so that its frame blocking ratio can be set to read data from memory address,which contain the input voice data and will be changed according to the following rule: take an ascending address for reading until the first frame has been completed and then the address will be subtracted by a overlapped value to read a next frame. For example, the frame has 160 samples, an overlapping ratio is 50%, the reading address will be increased from 1 to 160, then be subtracted by 80 (50% of overlap) to read from 80-240. In fact the second frame contains the values from 81-160, but to calculate the value of S_{81}, it must has a theoretical value S_{80} so that this is the reason why we should read from the address of second frame from 80 to serve for its calculation. Similarly, we have reading address

for following frames 80-240 (2nd frame), 160-320 (3rd frame), 240-400 (4th frame) respectively.

Subtraction for reading address of each configuration setting is transformed into the addiontion of 2nd compensation number by using the formula:

$$com\_2\_ovl = 2^{nd}\ compensation\ (Overlap\ ratio\ \times Number\ of\ sample\ per\ frame)$$

This formula wil be difficult to implement in hardware by the overlap ratio so we will calculate the com_2_ovl in software and send it via bus to configure.

Both pre-emphasis and energy blocks all need to use frame's data, which is retrieved from the voice input memory, so we perform the calculation of them in the one state machine.

The calculation of energy block need Logarithm operation to minimize energy values because when the number of sample per frame is too large, the cumulative pattern of squares of each sample will be become an enormous result. Energy is considered to be one of the characteristic of a data frame.

From above Figure.2.1 and mathematic equations, it is acknowledged that if most arithmetic operations including addition, multiplication and logarithm apply fixed-point numbers, accuracy is affected significantly. As a result, in order to tackle this issue, floating-point format is proposed in this thesis. Indeed, floating-point format is widely applied to variety of hardware designs so as to improve accuracy in [16]. To be more specific, Figure 2.10 presents such standard with total of 32 bit comprising of 1 sign bit, 8 exponent bits and 23 mantissa bits.

However, applying floating-point format has to face high cost of 23 bit array multiplication. Consequently, Booth algorithm confirmed the high performance compared with array multiplication in [17] is applied to enhance speed of calculation.

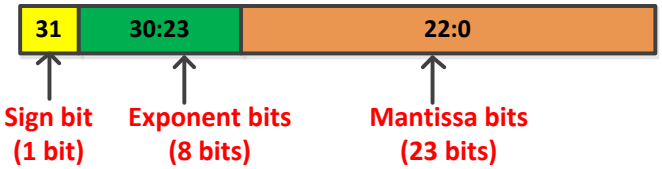

Figure 2.10. IEEE 754 floating-point data format

Logarithm's implementation is based on "Efficient floating-point design for FPGAs logarithm of Nikolaos Alachiotis Unit, Alexandros Stamatakis":

The format of 32-bit floating point IEEE - 754 is of these components: SIGN (1 bit), Exponent (8 bits) and mantissa (23 bits). Decimal value (Value) of this type are calculated as follows:

$$\text{Value} = \text{SIGN} \times 2^{\text{EXPONENT}} \times \text{MANTISSA}$$

In order to calculate logarithmic operation with the requirements of this proposed architecture, the idea is based on the following chain of transformations:

$$\begin{aligned} \log(\text{Value}) &= \log(2^{\text{EXPONENT}} \times \text{MANTISSA}) \\ &= \log(2^{\text{EXPONENT}}) + \log(\text{MANTISSA}) \\ &= \text{EXPONENT} \times \log 2 + \log(\text{MANTISSA}) \end{aligned} \quad (9)$$

In addition, to deal with the requirement for the calculation of logarithm with any base *x*, it will also be solved with the replacement of constant " $\log(2)$" by " $\log_x(2)$ " in (9).

To execute the requirement of solving this logarithm operation with 32 bit floating point number, this ideal was done by the method of LUT (Look Up Table), based on the use of internal ROMs, which contain the value of these components: EXPONENT and log(MANTISSA). All of them can be retrieved data and performed by simple operation of floating point number below:

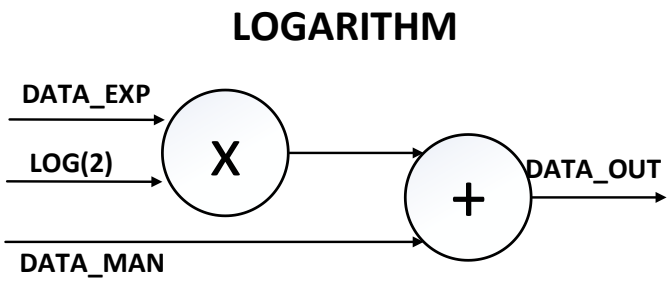

Figure 2.11 The datapath of Logarithm module

With LUT method, the data-path to perform logarithm calculations is presented as figure:

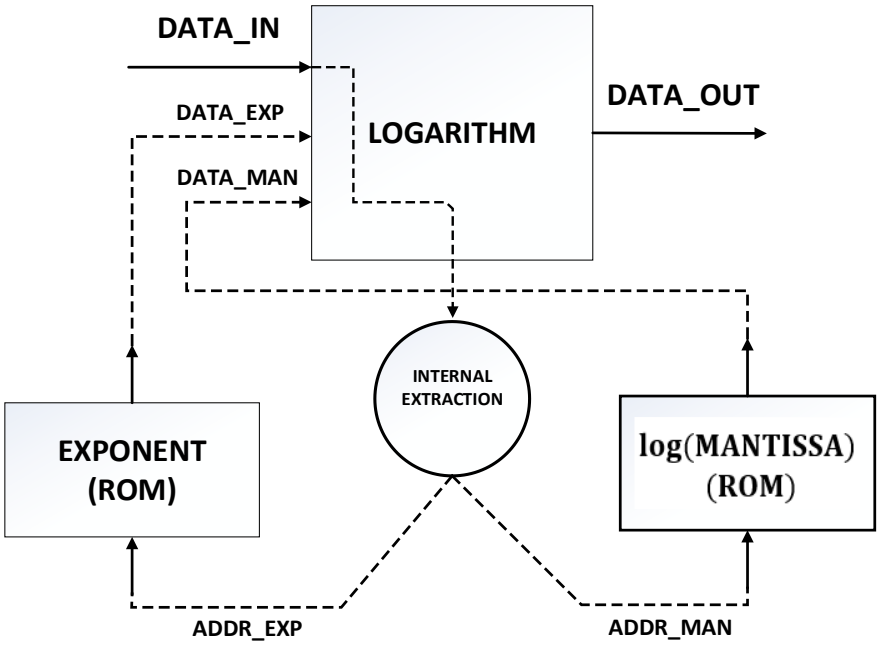

Figure 2.12 Logarithm module with its ROMs

Based on a mentioned mathematical model, data flow of architecture is depicted in Figure 2.11. The implemmentation of its operations is detailed in the following state diagrams and tables below.

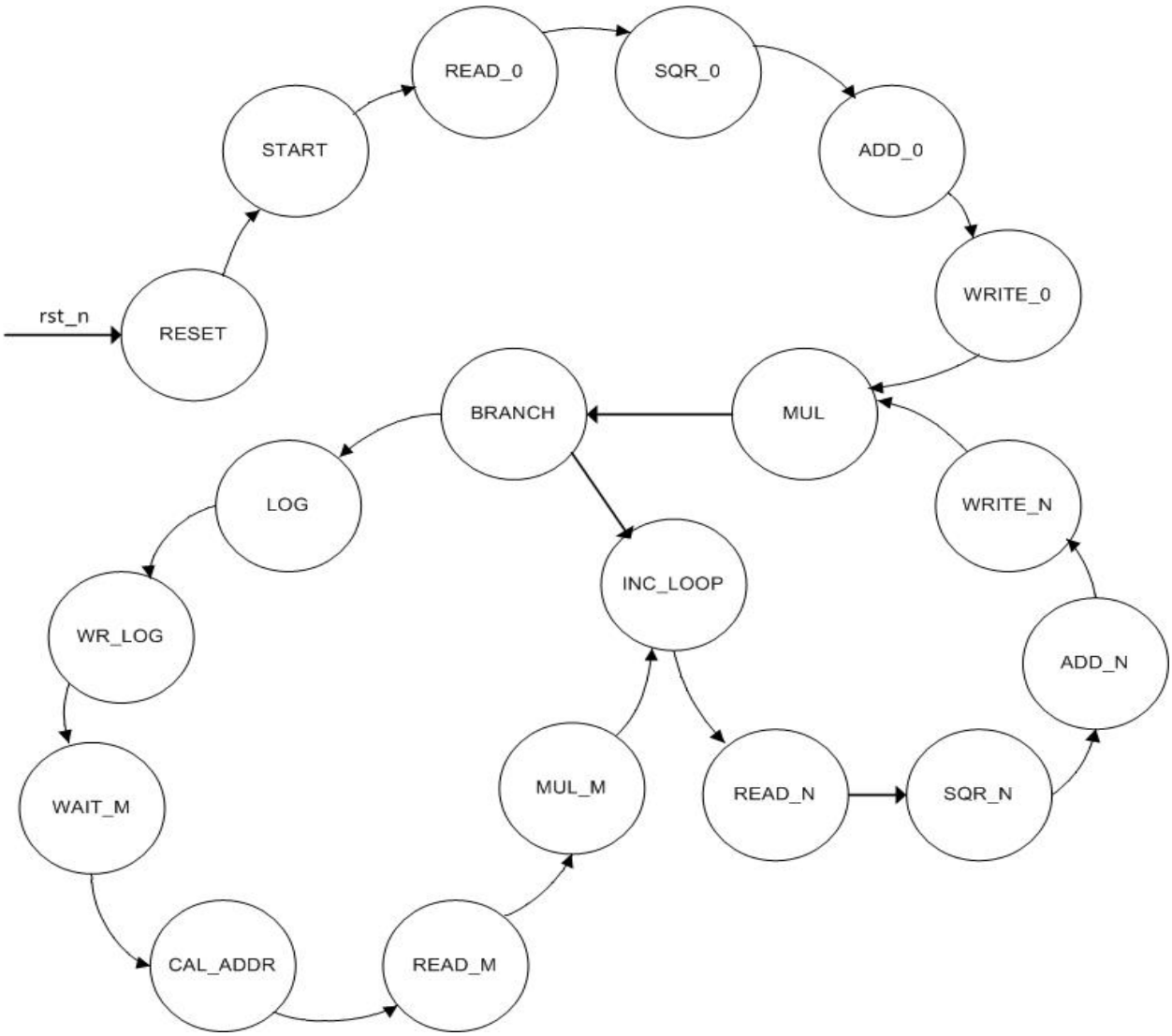

Figure 2.13 The state machine of Pre-emphasis block

Table 2.8 The function of states in Pre-emphasis block

	Present state	**Function**
1	RESET	Reset a system
2	START	Start a system
3	READ_0	Read the first element of memory
4	SQR_0	Take the square of data which has been read (For the purpose of Energy computation)
5	ADD_0	Implement two tasks: Plus first element with zero Take the sum of all square results (For the purpose of energy computation)
6	WRITE_0	Write the first element into Pre-emphasis memories
7	MUL	Multipli the element which has been read from voice memory with an alpha value
8	BRANCH	If this frame has already done then wait the permission from main control to repeat for others. If has not yet finished then continue this task
9	LOG	Computation of Energy
10	WRITE_LOG	Write Energy value into Energy memories
11	WAIT_M	Wait the enable signal from Main control to activate a new frame
12	CAL_ADDR	Decrease counter value to release new address which is adapted with overlap rate
13	READ_M	Read first element of a new frame (With overlap)
14	MUL_ADD	Implement 2 task: Multiply first element of frame (Not initial frame) with alpha Reset register which contents the sum of all square results
15	INC_LOOP	Increase counter of sample in one frame
16	READ_N	Read data from Voice memory
17	SQR_N	Take the square of data which has been read
18	ADD_N	Implement two tasks: Plus first element with the product of alpha and previous element Take the sum of all square results (For the purpose of Energy computation)
19	WRITE_N	Write the element which has been computed into Pre_emphasis memory

Table 2.9 State machine in Pre-emphasis block

	Present state	**Next state**	**Condition**
1	RESET	START	When receive an enable signal from Main Control
2	START	READ_0	After 1 clock
3	READ_0	SQR_0	The counter is over
3	SQR_0	ADD_0	The counter is over
4	ADD_0	WRITE_0	Add function has already done (The counter is over)
5	WRITE_0	MUL	After 1 clock
6	MUL	BRANCH	Mul function has already done (The counter is over)
7	BRANCH	LOG	1 frame finishes (The counter is over)
		INC_LOOP	1 frame has not yet finished
8	INC_LOOP	READ_N	After 1 clock
9	READ_N	SQR_N	The counter is over
10	SQR_N	ADD_N	The counter is over
11	ADD_N	WRITE_N	Add function has already done (The counter is over)
12	WRITE_N	MUL	After 1 clock
13	WAIT_M	CAL_ADDR	When receive an enable signal from Main Control
14	CAL_ADDR	READ_M	After 1 clock
15	READ_M	MUL_M	After 1 clock
16	MUL_M	INC_LOOP	Mul function has already done (The counter is over)
17	LOG	WRITE_LOG	The counter is over
18	WRITE_LOG	WAIT_M	After 1 clock
19	WAIT_M	CAL_ADDR	When receive an enable signal from Main Control
		WAIT_M	Has not yet received enable signal from Main control

Table 2.10 Input and output of Pre-emphasis block

Pin name	**Type**	**#Bit**	**Description**
clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
preem_state_en	Input	1	An enable signal allows Pre-emphasis block implements
sample_in_frame	Input	11	Number of sample in 1 frame
com_2_ovl	Input	12	Use to release a new address for each new frame

			(Follow the formula above)
voice_data_in	Input	32	Voice data are read
alpha	Input	32	Alpha parameter
preem_mem_write_addr	Output	12	The address of Pre –emphasis result for memory writing
preem_data_out	Output	32	The result of Pre-emphasis
preem_mem_read_addr	Output	12	The address of input data from Voice memory which use to read into Pre-emphsis
write_preem_en	Output	1	An enable signal allows Pre-emphasis results are written
energy_log_data_out	Output	32	Logarithm of energy results
energy_log_frame_num	Output	12	The frame serial will corresponding with an logarithm of energy result's address
write_energy_en	Output	1	An enable signal allows to write logarithm of energy results
exp_data_out	Input	32	Results from " exponent " table for logarithm computation
man_data_out	Input	32	Results from " mantissa " table for logarithm computation
exp_addr	Output	12	Address of exponent table for logarithm computation
man_addr	Output	12	Address of mantissa table for logarithm computation

Table 2.11 The function of states'input/output of Pre-emphasis block

Pin name	**# Bit**	**Description**
sel	1	Choose the input for addition module
change_addr_sel	2	Control the variation of address
add_en	1	Enable pin of addition module
mul_en	1	Enable pin of multiplication module
counter_en	1	Enable pin of counter module
counter_loop_en	1	Enable pin of counter_loop which use to count number of loops
num_wait	5	Value of counter
change_addr_value	1	Change address (increase/decrease)
addr_preem_en	1	Enable pin which release a new address
write_preem_en	1	An enable signal allows Pre-emphasis block write its results into memories
energy_log_en	1	Enable pin of logarithm of energy results
sel_log	2	Enale pin for taking a sum of square results
sqr_en	1	Enable pin of square module

write_energy_log_en	1	An enable signal allows to write logarithm of energy results into memories
frame_count_en	1	Enable pin of frame count module (Address for logarithm of energy writing)
log_add_en	1	Enale pin for taking a sum of square results (use for logarithm of energy)

Table 2.12 Output of states in Pre-emphasis block

State	**Output of state**

RESET	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd0;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

START	sel <= 1'b0;
	change_addr_sel <= 2'b01;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b1;
	num_wait <= 5'd0;
	change_addr_value <= 5'd1;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

READ_0	sel <= 1'b0; change_addr_sel <= 2'b11;

	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd1;
	change_addr_value <= 5'd1;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

SQR_0	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_MUL;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b1;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

ADD_0	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b1;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_ADD;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;

	frame_count_en <= 1'b0; log_add_en <= 1'b1;

WRITE_0	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WRITE;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b1;
	write_preem_en <= 1'b1;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

MUL	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b1;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_MUL;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

BRANCH	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= 0;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;

	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

LOG	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_LOG;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b1;
	sel_log <= 2'b0;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

WRITE_LOG	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WRITE;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b1;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

WAIT_M	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;

	num_wait <= 0;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

CAL_ADDR	sel <= 1'b1;
	change_addr_sel <= 2'b10;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_CAL;
	change_addr_value <= com_2_ovl;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b1;
	log_add_en <= 1'b0;

MUL_ADD	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b1;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_MUL;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b1;

READ_M	sel <= 1'b1; change_addr_sel <= 2'b11;

	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd1;
	change_addr_value <= 5'd1;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b00;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

INC_LOOP	sel <= 1'b1;
	change_addr_sel <= 2'b01;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b1;
	num_wait <= 0;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

READ_N	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd1;
	change_addr_value <= 5'd1;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b10;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;

	frame_count_en <= 1'b0; log_add_en <= 1'b0;

SQR_N	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_MUL;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b1;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

ADD_N	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b1;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_ADD;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b1;

WRITE_N	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WRITE;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b1;
	write_preem_en <= 1'b1;

	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

WAIT	sel <= 1'b1;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WAIT;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

END	sel <= 1'b0;
	change_addr_sel <= 2'b11;
	add_en <= 1'b0;
	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd0;
	change_addr_value <= 5'd0;
	addr_preem_en <= 1'b0;
	write_preem_en <= 1'b0;
	energy_log_en <= 1'b0;
	sel_log <= 2'b11;
	sqr_en <= 1'b0;
	write_energy_log_en <= 1'b0;
	frame_count_en <= 1'b0;
	log_add_en <= 1'b0;

2.2.2.3 Window block

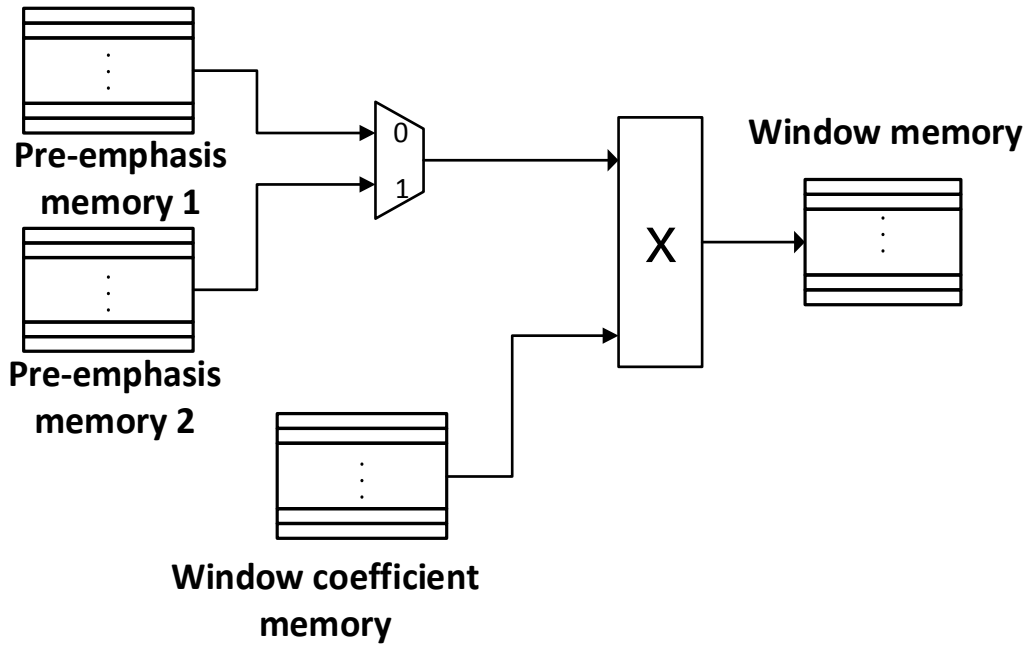

Figure 2.14 Window block

Window block is computed basically by multiplication of data after through the Pre-emphasis with window filter coefficient. Therefore, a multiplication can be carried out in the block diagram above. Because of after Windows, the next step is FFT calculation, which requires some FFT point is an exponential of 2. So that we add the state enter a value of 0 array in the memory to make sure the number of output data per each frame will be equally with the FFT point, which has been considered at the initial setting. The formula for calculating number of needed 0 values is:

$$Number\ of\ needed\ 0\ value = FFT\ point - Number\ of\ sample\ frame$$

The detailed state diagrams is described by the following figure:

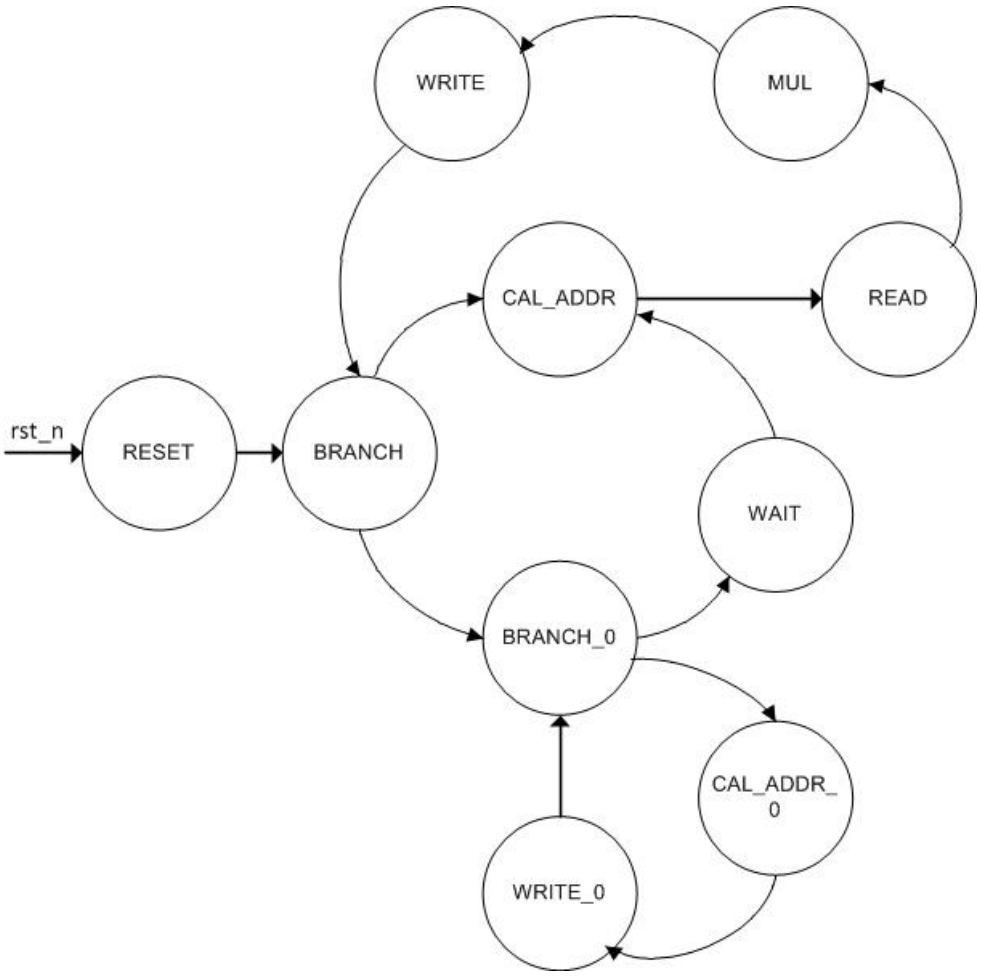

Figure 2.15 The state machine of Window block

Table 2.13 The function of states in Window block

	Present state	**Function**
1	RESET	Reset a system
2	BRANCH	If this frame has already done then the next state is BRANCH_0. If it has not yet finished then continue this task
3	CAL_ADDR	Increase the address in one frame to read a next element
4	READ	Read data from Pre-emphasis memory
5	MUL	Multipli the element which has been read with the corresponding window coefficient
6	WRITE	Write the element which has been computed into Window memory

7	BRANCH_0	If this result has already filled with needed zero-range then wait the permission from main control to repeat for others. If has not yet finished then continue this task
8	CAL_ADDR_0	Increase the address in one frame to write needed zero-range into momories
9	WRITE_0	Write zero value into Window memories
10	WAIT	Wait the enable signal from Main control to activate a new frame

Table 2.14 State machine in Window block

	Present state	**Next state**	**Condition**
1	RESET	BRANCH	When receive an enable signal from Main Control
2	BRANCH	CAL_ADDR	1 frame has not yet finished
3		BRANCH_0	1 frame finishes (The counter is over)
4	CAL_ADDR	READ	After 1 clock
5	READ	MUL	After 1 clock
6	MUL	WRITE	Mul function has already done (The counter is over)
7	WRITE	BRANCH	After 1 clock
8	BRANCH_0	CAL_ADDR_0	1 frame has not yet filled with needed zero-range
		WAIT	1 frame finishes the zero-range (The counter is over)
9	CAL_ADDR_0	WRITE_0	After 1 clock
10	WRITE_0	BRANCH_0	After 1 clock
11	WAIT	CAL_ADDR	When receive an enable signal from Main Control

Table 2.15 Input and output of Window block

Pin name	**Type**	**#Bit**	**Description**
clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
fft_num	Input	12	FFT point
window_en	Input	1	An enable signal allows Window block implements
window_data_in	Input	32	Output data from Pre-emphasis which use to read into Window block
sample_in_frame	Input	12	Number of sample in 1 frame
window_cof_in	Input	32	The Window coefficient

win_mem_read_addr	Output	12	The address of Window input for memory reading
window_mem_write_addr	Output	12	The address of Window output result for memory writing
window_data_out	Output	32	The results of data after calculated through Window and filled with needed zero-range
write_win_en	Output	1	An enable signal allows to write Window results into memories

Table 2.16 The function of states'input/output of Window block

Pin name	**# Bit**	**Description**
mul_en	1	Enable pin of multiplication module
counter_en	1	Enable pin of counter module
counter_sample_en	1	Enable pin of sample per frame counter (The counter is over when 1 frame finishes)
counter_value	4	Value of counter
write_win_en	1	An enable signal allows Window block write its results into memories
counter_addr_en	1	Enable pin of counter module for data writing which has been consisted of Window results ans need zero-range.
fill_zero	1	An enable signal allows Window block write into memories

Table 2.17 Output of states in Window block

State	**Output of state**

RESET	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b0;

READ	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;

	fill_zero <= 1'b0;

MUL	mul_en <= 1'b1;
	counter_en <= 1'b1;
	counter_sample_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b0;

WRITE	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b1;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b0;

CAL_ADDR	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b1;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b1;
	fill_zero <= 1'b0;

BRANCH	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b1;

CAL_ADDR_0	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b1;
	fill_zero <= 1'b1;

WRITE_0	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	write_win_en <= 1'b1;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b1;

WAIT	mul_en <= 1'b0;

	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b1;

BRANCH_0	mul_en <= 1'b0;
	counter_en <= 1'b0;
	counter_sample_en <= 1'b0;
	counter_value <= 4'd0;
	write_win_en <= 1'b0;
	counter_addr_en <= 1'b0;
	fill_zero <= 1'b1;

2.2.2.4 FFT block

Theoretical basis for algorithm of this block is the radix 2 decimation-in-time FFT with N points. A N-point DFT, where $N = 2^m$, is implemented through m stages of computation, each has its own input and output. The output from a certain stage is the input to the next stage. The sum of x(n) can be decomposed into two terms, one containing n-even terms with $n = [0,2,4,\ldots, N-2]$, the other n-odd terms with $n = [1,3,5,\ldots, N-1]$ such as equations below:

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad ,k = 0,1,2,\ldots,N-1$$

$$\text{với } W_N^{nk} = e^{-j\left(\frac{2\pi}{N}\right)nk} \qquad (1)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(j\frac{2\pi\times(2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(j\frac{2\pi\times(2n+1)k}{N}\right)} \qquad (2)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(j\frac{2\pi nk}{\frac{N}{2}}\right)} + e^{-\left(j\frac{2\pi k}{N}\right)} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(j\frac{2\pi nk}{\frac{N}{2}}\right)} \qquad (3)$$

$$= DFT_{\frac{N}{2}}\left[\left[x(0), x(2), \ldots, x(N-2)\right]\right] +$$
$$W_N^k DFT_{\frac{N}{2}}\left[\left[x(1), x(3), \ldots, x(N-1)\right]\right] \qquad (4)$$

$$= DFT_{\frac{N}{2}}\left[x_{chan}(n)\right] + W_N^k DFT_{\frac{N}{2}}\left[x_{le}(n)\right] \qquad (5)$$

X[k] is written as the sum of two $\frac{N}{2}$ DFTs, apply this algorithm to each of the $\frac{N}{2}$ point DFTs also. And the algorithm can be applied recursively to $\frac{N}{4}, \frac{N}{8}, \frac{N}{16} \ldots$ until $\frac{N}{2^p} < 2$.

With the range of $k = 0, 1, 2, \ldots, N-1$, and the property of twiddle factor : $W_N^{k+\frac{N}{2}} = W_N^k W_N^{\frac{N}{2}} = -W_N^k$, the computation of butterfly is achieved as below:

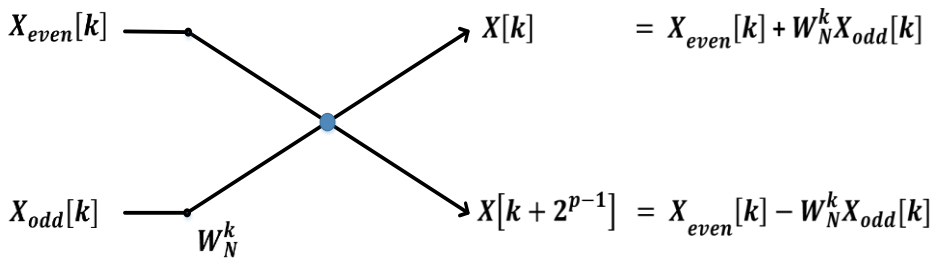

Figure 2.16 The model butterfly computation

In the p-th stage of computation, where $1 \le p \le m$, with $m = log_2 N$, the elements of the input sequence are arranged into $\frac{N}{2^p}$ groups, each consists of 2^p elements and 2^{p-1} butterfly computations. The two elements participating in the butterfly computation in the p-th stage are spaced in index 2^{p-1}.

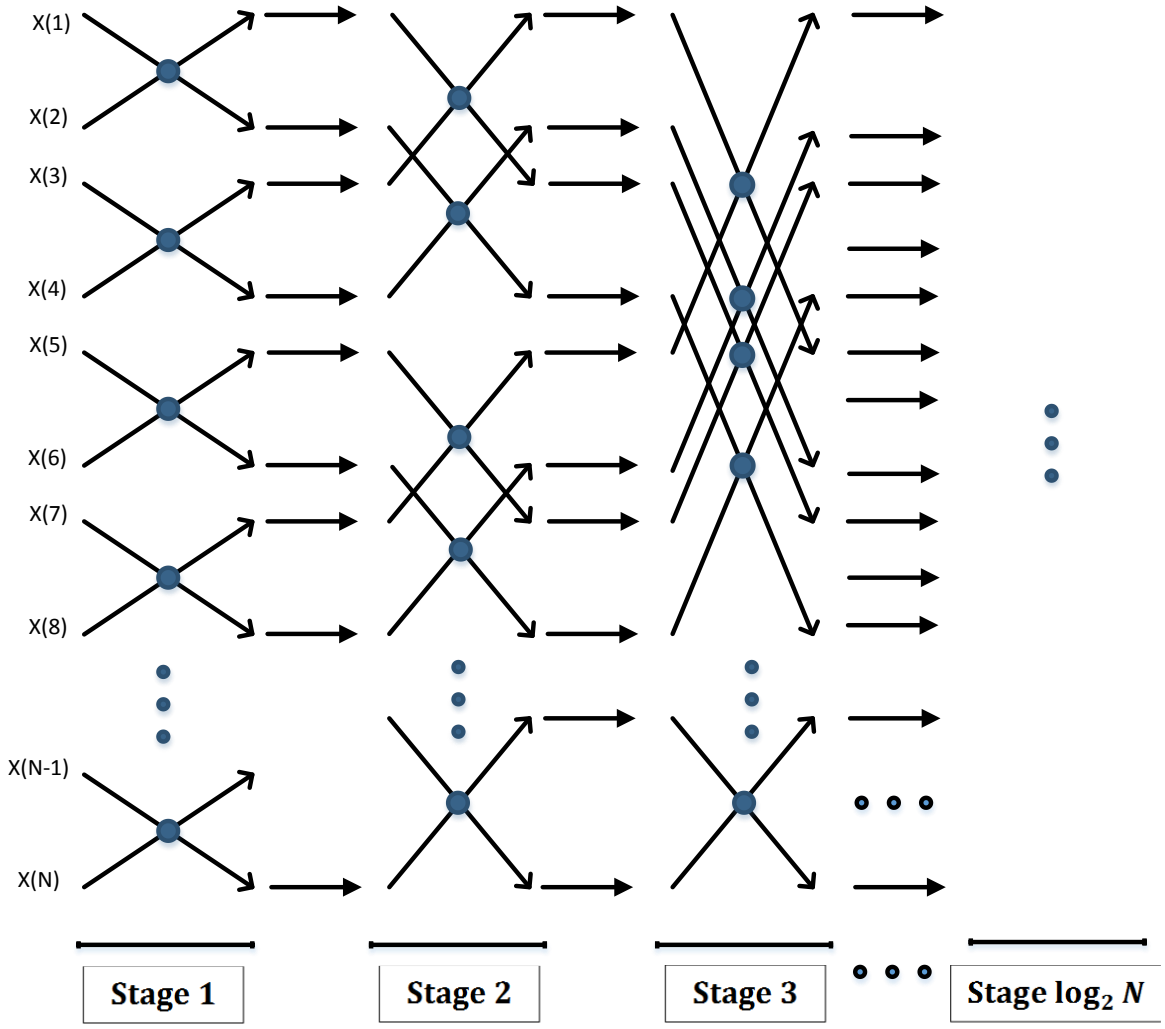

Figure 2.17 The model butterfly computation with N points.

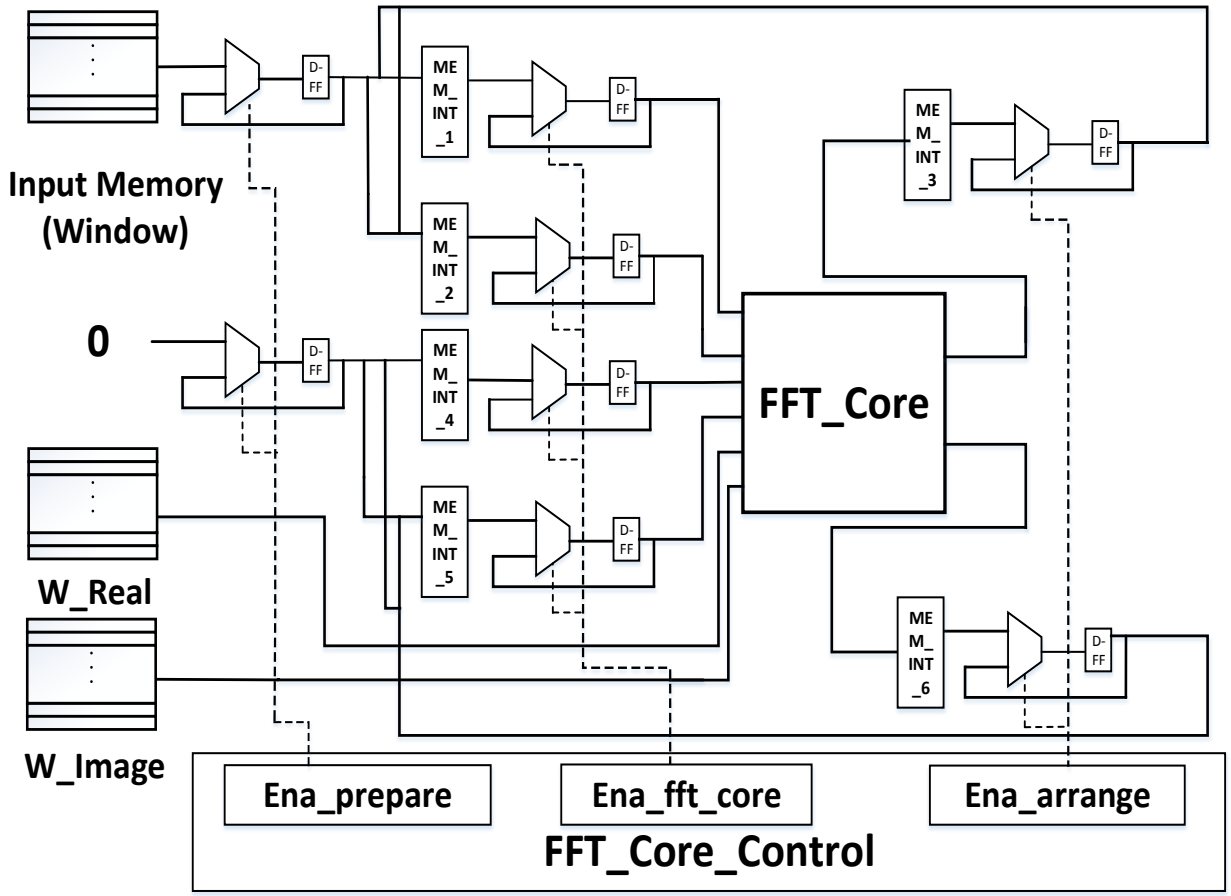

Figure 2.18 FFT with N points block.

The implementation of FFT according to the ideal of dynamic configuration for FFT point such as : 128, 256, 512 or 1024 and so on has been completed successfully. This achievement can be seen as the outstanding point of MFCC architecture reseach due to the the meaningful ability of reconfiguration, which allow to face with lots of problems while the fixed MFCC architecture did not perform before. Furthermore, dynamic FFT also be the key to slove the issuses that appear when deal with the requirement of word or sentence recognition system.

Author	**Technology**	**FFT points**	**Frequency (MHz)**	**Power (mW)**	**FFT latency**
GIN-DER WU [18]	ASIC (0.18µm)	256	100	89,18	10.4 µs
Chin-Teng Lin [19]	ASIC (0.13µm)	256	100	22.37	-
Dongsuk Jeon [20]	ASIC (65 nm)	1024	19	-	6,7 µs
Lihong Jia [21]	ASIC (0.6µm)	128	50	400	3 µs
Atin Mukherjee[22]	FPGA (Xilinx Virtx-6)	8	51	-	19.598 ns
K. Umapathy [23]	ASIC (90 nm)	128	40	-	40 µs
Ediz Çetin [24]	ASIC (0.7µm)	256	40	-	102,4 µs
Purposed FFT	**ASIC (130nm)**	**8-4096**	**500**	**3.44**	**1.5µs-1.969ms**

This wonderful work has been verified and recognized in "Proceedings of The 2015 National Conference on Electronics, Commuications and Information Technology, ECIT 2015", with the title of "Integrated ASIC-Based Dynamic FFT architecture with radix 2 in 130nm technology" in [25].

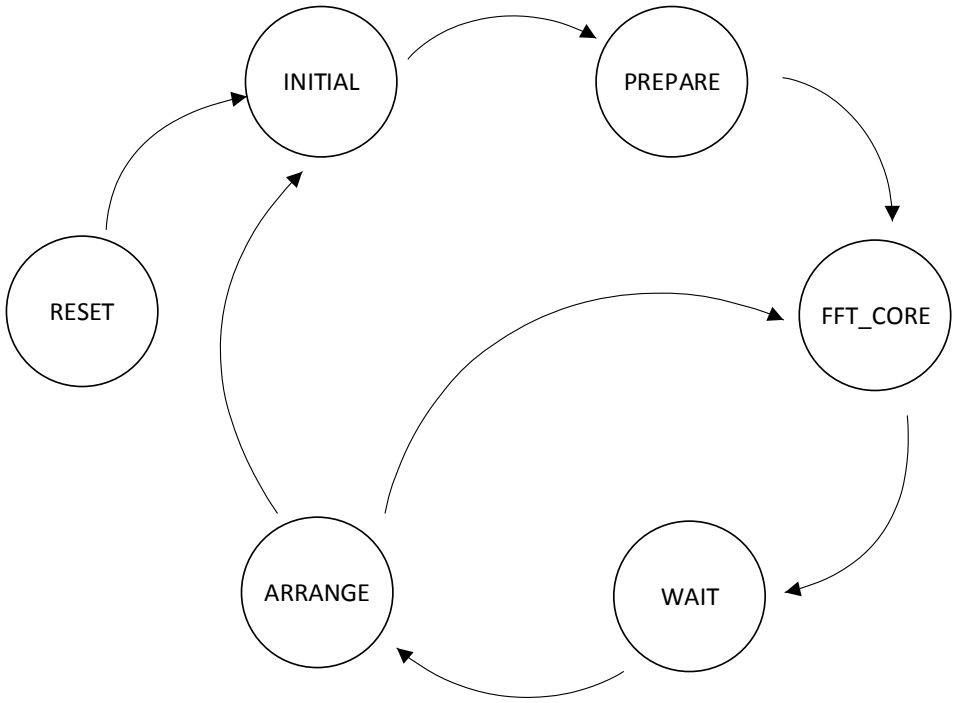

Figure 2.19 The state machine of FFT with N points.

Table 2.18 The function of states in FFT block

	Present state	**Function**
1	RESET	Reset a system
2	INITIAL	Initial state of internal memories reading
3	PREPARE	Implement the bit-reserved algotithm for the input data
4	FFT_CORE	Implement the butterfly computation
5	WAIT	Until a butterfly computation has been finished
6	ARRANGE	Implement the arrangement of output data of butterfly computation

Table 2.19 State machine in FFT block

	Present state	**Next state**	**Condition**
1	RESET	INITIAL	When receive an enable signal from Main Control
4	INITIAL	PREPARE	When receive an enable signal from Main Control
5	PREPARE	FFT_CORE	When receive an enable signal from Main Control

6	FFT_CORE	WAIT	Butterfly computation has already done (The counter is over)
7	WAIT	ARRANGE	After 1 clock
8	ARRANGE	FFT_CORE	1 stage has not yet done
		INITIAL	Final stage finishes (The counter is over)

Table 2.20 The function of internal memories in FFT block

Interal Memory	Description
MEM_INPUT	Contain the value of input data (just has the real part)
0	The imaginary part of input data is zero at initial sate
MEM_INT_1	Contain the real part of the first complex number participates in butter fly computation
MEM_INT_2	Contain the real part of the second complex number participates in butter fly computation
MEM_INT_4	Contain the imaginary part of the first complex number participates in butter fly computation
MEM_INT_5	Contain the imaginary part of the second complex number participates in butter fly computation
MEM_INT_3	Contain the real part of the complex result number after butter fly computation
MEM_INT_6	Contain the imaginary part of the complex result number after butter fly computation
W_REAL	Contain the real part of a twiddle factor
W_IMAGE	Contain the imaginary part of a twiddle factor

Table 2.21 Output of states in FFT block

State	Output of state

RESET	ena_prepare <= 1'b0;
	ena_fft_core <= 1'b0;
	ena_fft_wait <= 1'b0;
	ena_arrange <= 1'b0;
	w_01 <= 1'b0;
	w_02 <= 1'b0;
	w_03 <= 1'b0;
	w_04 <= 1'b0;
	w_05 <= 1'b0;
	w_06 <= 1'b0;

INITIAL	ena_prepare <= 1'b0; ena_fft_core <= 1'b0;

	ena_fft_wait <= 1'b0;
	ena_arrange <= 1'b0;
	w_01 <= 1'b0;
	w_02 <= 1'b0;
	w_03 <= 1'b0;
	w_04 <= 1'b0;
	w_05 <= 1'b0;
	w_06 <= 1'b0;

PREPARE	ena_prepare <= 1'b1;
	ena_fft_core <= 1'b0;
	ena_fft_wait <= 1'b0;
	ena_arrange <= 1'b0;
	w_01 <= 1'b1;
	w_02 <= 1'b1;
	w_03 <= 1'b0;
	w_04 <= 1'b1;
	w_05 <= 1'b1;
	w_06 <= 1'b0;

FFT_CORE	ena_prepare <= 1'b0;
	ena_fft_core <= 1'b1;
	ena_fft_wait <= 1'b0;
	ena_arrange <= 1'b0;
	w_01 <= 1'b0;
	w_02 <= 1'b0;
	w_03 <= 1'b1;
	w_04 <= 1'b0;
	w_05 <= 1'b0;
	w_06 <= 1'b1;

WAIT	ena_prepare <= 1'b0;
	ena_fft_core <= 1'b0;
	ena_fft_wait <= 1'b1;
	ena_arrange <= 1'b0;
	w_01 <= 1'b0;
	w_02 <= 1'b0;
	w_03 <= 1'b1;
	w_04 <= 1'b0;
	w_05 <= 1'b0;
	w_06 <= 1'b1;

ARRANGE	ena_prepare <= 1'b0;
	ena_fft_core <= 1'b0;
	ena_fft_wait <= 1'b0;
	ena_arrange <= 1'b1;
	w_01 <= 1'b1;
	w_02 <= 1'b1;

	w_03 <= 1'b0;
	w_04 <= 1'b1;
	w_05 <= 1'b1;
	w_06 <= 1'b0;

2.2.2.5 Amplitude block

Amplitude block received the output data of previous FFT, which includes the real and imaginary part of each FFT result. To be able to apply the pipeline method, 4 internal memories store the value of complex number, the first two memories contain a real part, and other corresponding with the imaginary values. Approximate formula:

$$|a + jb| = \max(|a|, |b|) + 1/4\min(|a|, |b|)$$

and the corresponding state diagram will implement the calculation of this block in hardware as follows.

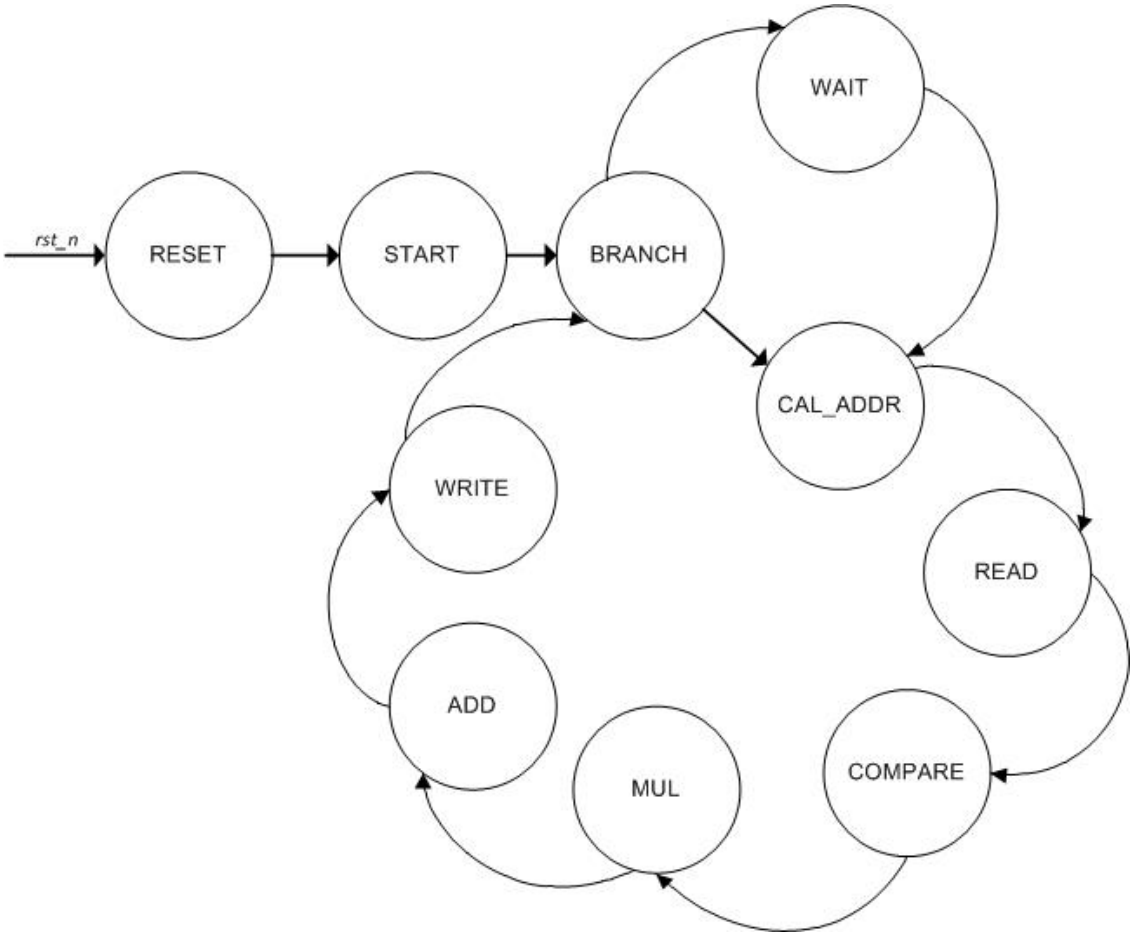

Figure 2.20 The state machine of Amplitude block

Table 2.22 The function of states in Amplitude block

	Present state	**Function**
1	RESET	Reset a system
2	START	Activate a system
3	BRANCH	If this frame has already done then wait the permission from Main control to repeat for others. If has not yet finished then continue this task
4	CAL_ADDR	Increase the address in one frame to read a next element
5	READ	Read data from FFF memory
6	COMPARE	Max and Min value has been chosen by comparision module

7	MUL	Multipli the Min value with ¼
8	ADD	Plus the Max value with a result of MUL module above
9	WRITE	Write the element which has been computed into Magnitude memory
10	WAIT	Wait the enable signal from Main control to activate a new frame

Table 2.23 State machine in Amplitude block

	Present state	**Next state**	**Condition**
1	RESET	START	When receive an enable signal from Main Control
2	START	BRANCH	After 1 clock
3	BRANCH	CAL_ADDR	1 frame has not yet finished
4		WAIT	1 frame finishes (The counter is over)
5	CAL_ADDR	READ	After 1 clock
6	READ	COMPARE	After 1 clock
7	COMPARE	MUL	COMPARE function has already done (The counter is over)
8	MUL	ADD	MUL function has already done (The counter is over)
9	ADD	WRITE	Add function has already done (The counter is over)
10	WRITE	BRANCH	After 1 clock
11	WAIT	CAL_ADDR	When receive an enable signal from Main Control

Table 2.24 Input and output of Amplitude block

Pin name	**Type**	**#Bit**	**Description**
Clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
fft_num	Input	12	FFT point
ampli_state_en	Input	1	An enable signal allows Amplitude block implements
quarter	Input	32	¼ parameter (can changed to get the high accuracy)
fft_real_data_in	Input	32	Real value from FFT
fft_image_data_in	Input	32	Imaginary value from FFT
ampli_mem_write_addr	Output	12	The address of Amplitude result for memory writing

ampli_data_out	Output	32	The result of Amplitude
ampli_mem_read_addr	Output	12	The address of input data (real and imaginary value) from FFT memory which use to read into Amplitude
write_ampli_data_en	Output	1	An enable signal allows Amplitude results are written

Table 2.25 The function of states'input/output of Amplitude block

Pin name	**# Bit**	**Description**
change_addr_sel	2	Control the variation of address
add_en	1	Enable pin of addition module
mul_en	1	Enable pin of multiplication module
counter_en	1	Enable pin of counter module
counter_loop_en	1	Enable pin of counter_loop which use to count number of loops
num_wait	5	Value of counter
change_addr_value	1	Change address (increase/decrease)
ena_max_min_fp_clk	1	Enable pin of comparision module
write_ampli_data_en	1	An enable signal allows Amplitude block write its results into memories
write_ampli_addr_en	1	Enable pin which release a Amplitude results address into memories

Table 2.26 Output of states in Amplitude block

State	**Output of state**

RESET	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd0;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

START	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;

	num_wait <= 5'd0;
	change_addr_sel <= 2'b00;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

BRANCH	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd1;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

PRE_CALC	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= 5'd1;
	change_addr_sel <= 2'b00;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

CAL_ADDR	add_en <= 1'b1;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b1;
	num_wait <= 5'd0;
	change_addr_sel <= 2'b01;
	change_addr_value <= 5'd1;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

READ	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_READ;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;

	write_ampli_addr_en <= 1'b0;

COMPARE	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b1;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_COMPARE;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

MUL	add_en <= 1'b0;
	mul_en <= 1'b1;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_MUL;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

ADD	add_en <= 1'b1;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b1;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_ADD;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b1;

WRITE	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;
	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WRITE;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b1;
	write_ampli_addr_en <= 1'b1;

WAIT	add_en <= 1'b0;
	mul_en <= 1'b0;
	ena_max_min_fp_clk <= 1'b0;

	counter_en <= 1'b0;
	counter_loop_en <= 1'b0;
	num_wait <= LOOPS_WAIT;
	change_addr_sel <= 2'b11;
	change_addr_value <= 5'd0;
	write_ampli_data_en <= 1'b0;
	write_ampli_addr_en <= 1'b0;

2.2.2.6 Mel filter block

Mel block is more complex than other algorithms because of its ability to apply flexibility lots of triangle filters. The logarithm is also used to calculate the mel filter. Therefore, the total value mel filter corresponding to the maximum configuration case so that it should contain 8 memory (4K memory for each) and up to 15 bit address decoder. Details of the state diagram is shown in the following table.

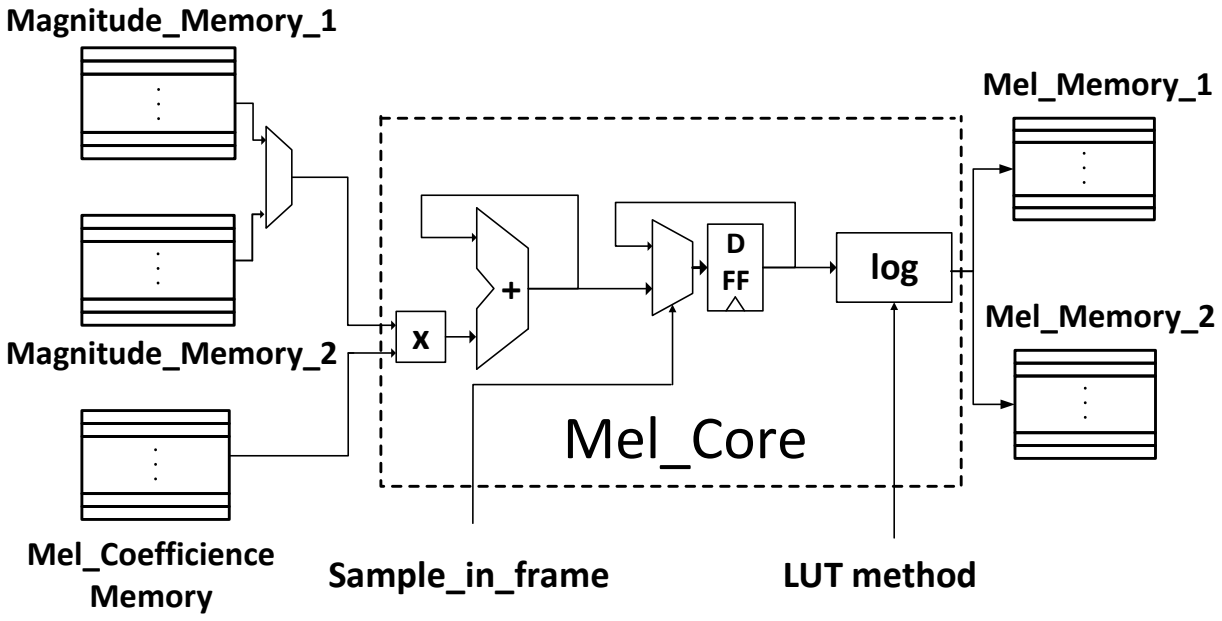

Figure 2.21 Mel block

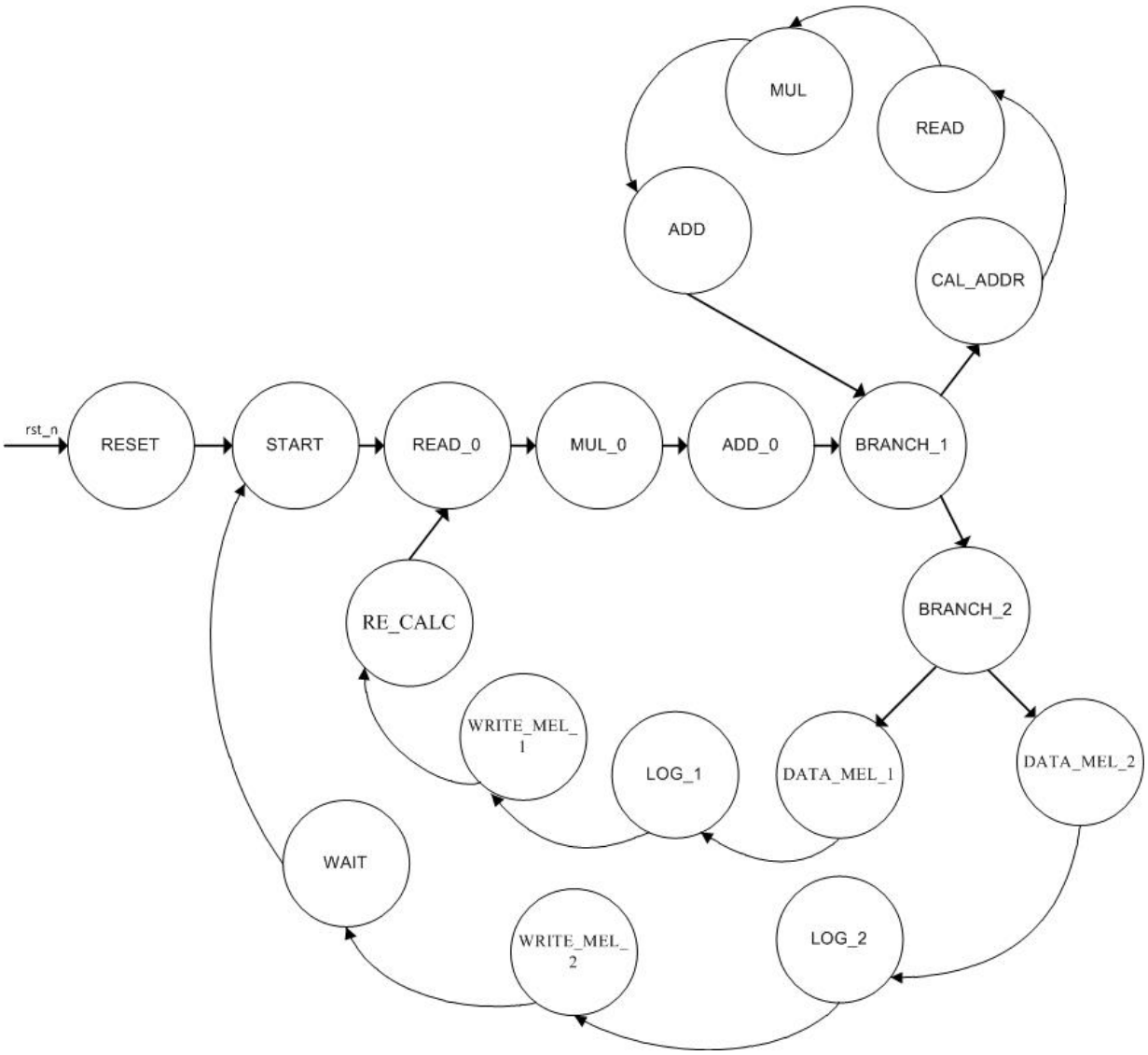

Figure 2.22 The state machine of Mel block

Table 2.27 The function of states in Mel block

	Present state	**Function**
1	RESET	Reset the system
2	START	Activate a new frame
3	READ_0	Read the first element of Amplitude memory
4	MUL_0	Multipli first element with the corresponding coefficient of Mel filter
5	ADD_0	Plus zero with the value which has been calculated

6	BRANCH_1	Check the process of calculation for one mel filter corresponding with one frame
7	BRANCH_2	Check the process of calculation for rest of mel filter corresponding with one frame
8	DATA_MEL_1	Count the number of mel filter for 1 frame
9	DATA_MEL_2	Count the number of final mel filter for 1 frame
10	LOG_1	Take the logarithm of mel value which has been calculated
11	LOG_2	Take the logarithm of final mel value of one frame.
12	WRITE_MEL_1	Write the mel value of each frame into Mel memories
13	WRITE_MEL_2	Write the final mel value of each frame into Mel memories
14	RE_CALC	Increase the reading address of Mel filter coefficient (When the calculation of new mel value is activated)
15	WAIT	Wait the enable signal from Main control to activate a new frame
16	CAL_ADDR	Release the reading address of Mel filter coefficient
17	READ	Read the Amplitude value for 1 frame
18	MUL	Multipli the element which has been read from Amplitude memory with Mel coefficient
19	ADD	Plus the recent element with previous element

Table 2.28 State machine in Mel block

	Present state	**Next state**	**Condition**
1	RESET	START	When receive an enable signal from Main Control
2	START	READ_0	After 1 clock
3	READ_0	MUL_0	After 1 clock
4	MUL_0	ADD_0	Mul function has already done (The counter is over)
5	ADD_0	BRANCH_1	Add function has already done (The counter is over)
6	BRANCH_1	BRANCH_2	A frame in stage of Mel calculation has already done
		CAL_ADDR	A frame in stage of Mel calculation has not yet done
7	BRANCH_2	DATA_MEL_1	Mel calculation for one frame has not yet done
		DATA_MEL_2	Mel calculation for one frame has already done
8	CAL_ADDR	READ	After 1 clock
9	DATA_MEL_1	LOG_1	After 1 clock

10	DATA_MEL_2	LOG_2	After 1 clock
11	LOG_1	WRITE_MEL_1	Logarithm function has already done (The counter is over)
12	LOG_2	WRITE_MEL_2	Logarithm function has already done (The counter is over)
13	WRITE_MEL_1	RE_CALC	After 1 clock
14	WRITE_MEL_2	WAIT	After 1 clock
15	RE_CALC	READ_0	After 1 clock
16	WAIT	START	When receive an enable signal from Main Control
17	CAL_ADDR	READ	After 1 clock
18	READ	MUL	After 1 clock
19	MUL	ADD	Mul function has already done (The counter is over)
20	ADD	BRANCH_1	Add function has already done (The counter is over)

Table 2.29 Input and output of Mel block

Pin name	**Type**	**#Bit**	**Description**
clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
mel_state_en	Input	1	An enable signal allows Mel block implements
mel_num	Input	6	Number of Mel filters
fft_num	Input	12	FFT point
mel_data_in	Input	32	The results of Amplitude block are used for Mel block
mel_cof_in	Input	32	Mel filter coefficient
mel_cof_read_addr	Output	15	Address for read Mel filter coefficient
mel_mem_read_addr	Output	12	Address for read from input data
mel_mem_write_addr	Output	12	Address for write results of Mel into memories
mel_data_out	Output	32	The result of Mel block
write_mel_en	Output	1	An enable signal allows Mel results are written

Table 2.30 The function of states'input/output of Mel block

Pin name	**# Bit**	**Description**
sel_add	1	Choose the input for addition module
change_addr_sel	2	Control the variation of address
add_en	1	Enable pin of addition module
mul_en	1	Enable pin of multiplication module
counter_en	1	Enable pin of counter module
counter_mel_en	1	Enable pin of counter_mel module

counter_value	5	Value of counter
addr_mel_en	1	Enable pin which release a new address
write_mel_en	1	An enable signal allows Mel block write its results into memories
counter_half_frame_en	1	Enable pin of frame counter module (Just a half of FFT point)
log_en	1	Enable pin of logarithm module

Table 2.31 Output of states in Mel block

State	**Output of state**

RESET	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

START	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b1;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

READ_0	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

MUL_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

ADD_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

BRANCH_1	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

BRANCH_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

DATA_MEL_1	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b1;
	addr_mel_en <= 1'b1;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

DATA_MEL_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b1;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

WRITE_MEL_1	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b1;
	log_en <= 1'b0;

WRITE_MEL_2	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b1;
	log_en <= 1'b0;

LOG_1	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_LOG;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b1;

LOG_2	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_LOG;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b1;

WAIT	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

RE_CALC	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

CAL_ADDR	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

READ	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 10'd0;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

MUL	sel_add <= 2'b01;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

ADD	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_mel_en <= 1'b0;
	addr_mel_en <= 1'b0;
	write_mel_en <= 1'b0;
	log_en <= 1'b0;

2.2.2.7 Ceptral block

The low cepstral coefficients' degree reflected bundle sound information of voice signals. In the analysis of spectrum for voice recognition, we normally used between 8 and 16 low cepstral coefficients' degree, but in the majority of applications we have already used 12 cepstral coefficients.

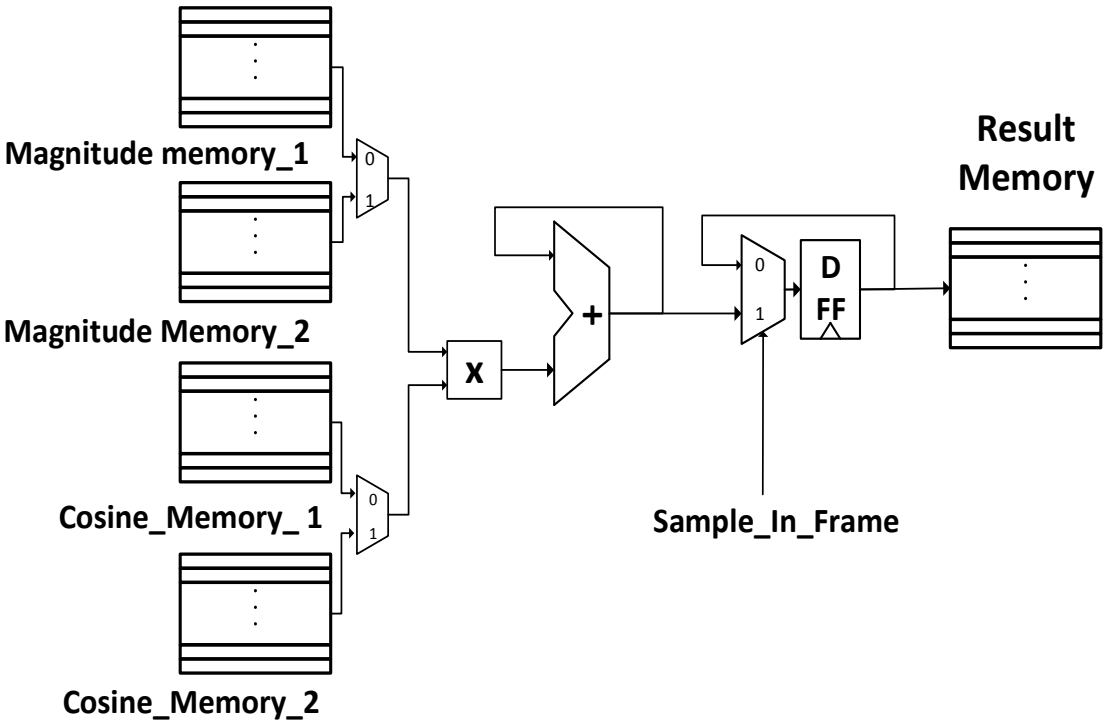

Figure 2.23 Cepstral block

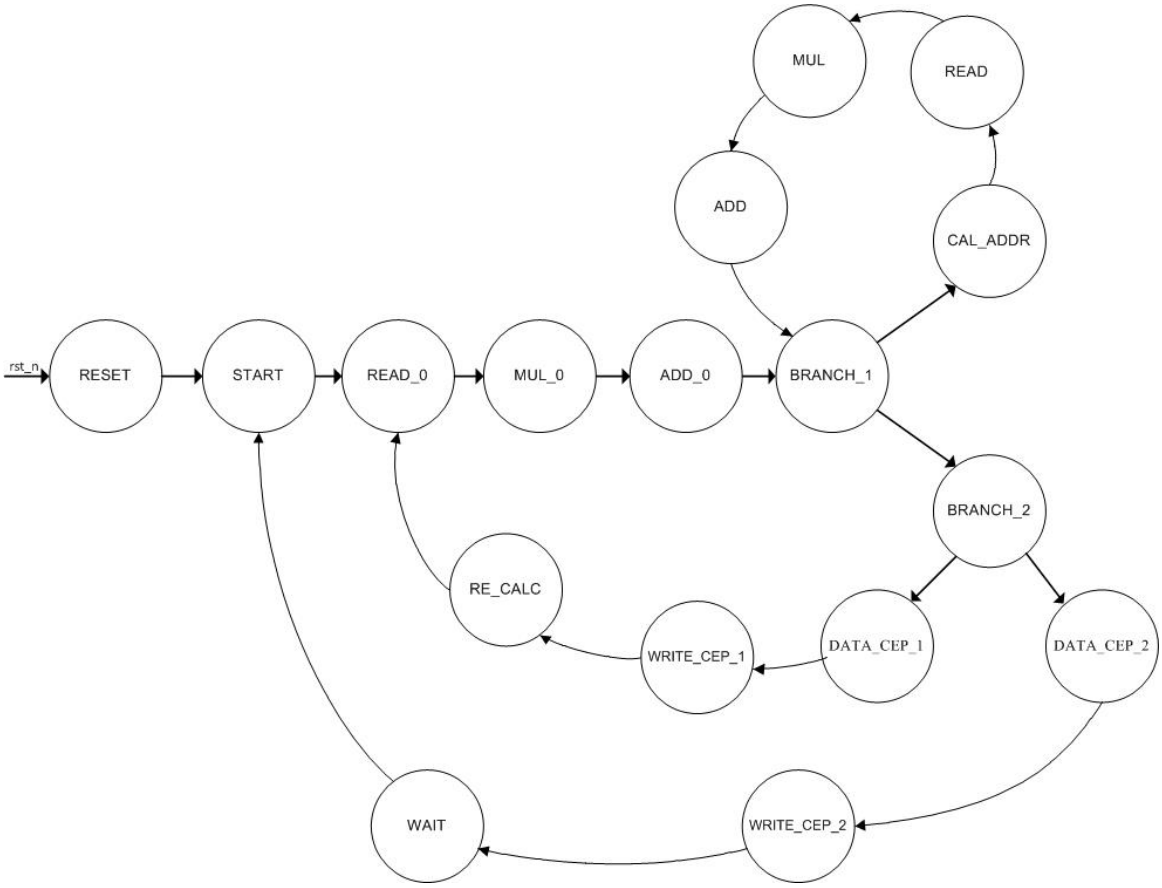

Figure 2.24 The state machine of Cepstral block

Table 2.32 The function of states in Cepstral block

	Present state	**Function**
1	RESET	Reset the system
2	START	Activate a new frame
3	READ_0	Read the first element of Mel in a frame
4	MUL_0	Multipli first element with the corresponding coefficient of Cepstral
5	ADD_0	Plus zero with the value which has been calculated
6	BRANCH_1	Check the process of calculation for one cepstral filter corresponding with one frame
7	BRANCH_2	Check the process of calculation for rest of cepstral filter corresponding with one frame

8	DATA_CEP_1	Count the number of cepstral filter for 1 frame
9	DATA_CEP_2	Count the number of final cepstral filter for 1 frame
10	WRITE_CEP_1	Write the cepstral value of each frame into Cepstral memories
11	WRITE_CEP_2	Write the final cepstral value of each frame into Mel memories
12	RE_CALC	Increase the reading address of Cepstral filter coefficient (When the calculation of new cepstral value is activated)
13	WAIT	Wait the enable signal from Main control to activate a new frame
14	CAL_ADDR	Release the reading address of Cepstral filter coefficient
15	READ	Read the Mel value for 1 frame
16	MUL	Multipli the element which has been read with Ceptral coefficient
17	ADD	Plus the recent element with previous element

Table 2.33 State machine in Cepstral block

	Present state	**Next state**	**Condition**
1	RESET	START	When receive an enable signal from Main Control
2	START	READ_0	After 1 clock
3	READ_0	MUL_0	After 1 clock
4	MUL_0	ADD_0	Mul function has already done (The counter is over)
5	ADD_0	BRANCH_1	Add function has already done (The counter is over)
6	BRANCH_1	BRANCH_2	A frame in stage of Cepstral calculation has already done
		CAL_ADDR	A frame in stage of Cepstral calculation has not yet done
7	BRANCH_2	DATA_CEP_1	Cepstral calculation for one frame has not yet done
		DATA_CEP_2	Cepstral calculation for one frame has already done
8	CAL_ADDR	READ	After 1 clock
9	DATA_CEP_1	WRITE_CEP_1	After 1 clock
10	DATA_CEP_2	WRITE_CEP_2	After 1 clock
11	WRITE_CEP_1	RE_CALC	After 1 clock
12	WRITE_CEP_2	WAIT	After 1 clock
13	RE_CALC	READ_0	After 1 clock
14	WAIT	START	When receive an enable signal from Main Control
		WAIT	Doesn’t receive an enable signal from Main Control
15	READ	MUL	After 1 clock

16	MUL	ADD	Mul function has already done (The counter is over)
17	ADD	BRANCH_1	Add function has already done (The counter is over)

Table 2.34 Input and output of Cepstral block

Pin name	**Type**	**#Bit**	**Description**
Clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
cep_state_en	Input	1	An enable signal allows Cep block implements
mel_num	Input	6	Number of Mel filters
cep_num	Input	6	Number of Cepstral filters
cep_data_in	Input	32	The results of Mel block are used for Cepstral block
cep_cof_in	Input	32	Cepstral filter coefficient
cep_cof_read_addr	Output	12	Address for read Cepstral filter coefficient
cep_mem_read_addr	Output	12	Address for read from input data
cep_mem_write_addr	Output	14	Address for write results of Mel into memories
cep_data_out	Output	32	The result of Cep block
write_cep_en	Output	1	An enable signal allows Cepstral results are written

Table 2.35 The function of states'input/output of Cepstral block

Pin name	**# Bit**	**Description**
sel_add	1	Choose the input for addition module
change_addr_sel	2	Control the variation of address
add_en	1	Enable pin of addition module
mul_en	1	Enable pin of multiplication module
counter_en	1	Enable pin of counter module
counter_cep_en	1	Enable pin of counter_cep module
counter_value	5	Value of counter
addr_cep_en	1	Enable pin which release a new address
write_cep_en	1	An enable signal allows Cepstral block write its results into memories
counter_half_frame_en	1	Enable pin of counter_frame module
frame_count_en	1	Enable pin of counter_frame serial module

Table 2.36 Output of states in Cepstral block

State	**Output of state**

RESET	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

START	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b1;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b1;

READ_0	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

MUL_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	counter_cep_en <= 1'b0;

	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

ADD_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

BRANCH_1	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

BRANCH_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b1;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

DATA_CEP_1	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;

	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b1;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

DATA_CEP_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b1;
	frame_num_out_en <= 1'b0;

WRITE_CEP_1	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b1;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

WRITE_CEP_2	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b1;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

WAIT	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

RE_CALC	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

CAL_ADDR	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

READ	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;

	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

MUL	sel_add <= 2'b01;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

ADD	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

2.2.2.8 Energy block

Since the calculation of energy block has already done simultaneously in the pre-emphasis stage, with the pipeline technique, if we save energy results immediately in the same memory with the results, it will be disputed memory because the cepstral block also use that memory. So we must save energy results into a separate memory, and after finishing cepstral calculations for all the frames, the main control will copy this energy results in the final memory to calculate delta function. The implementation of this block are shown in the following tables.

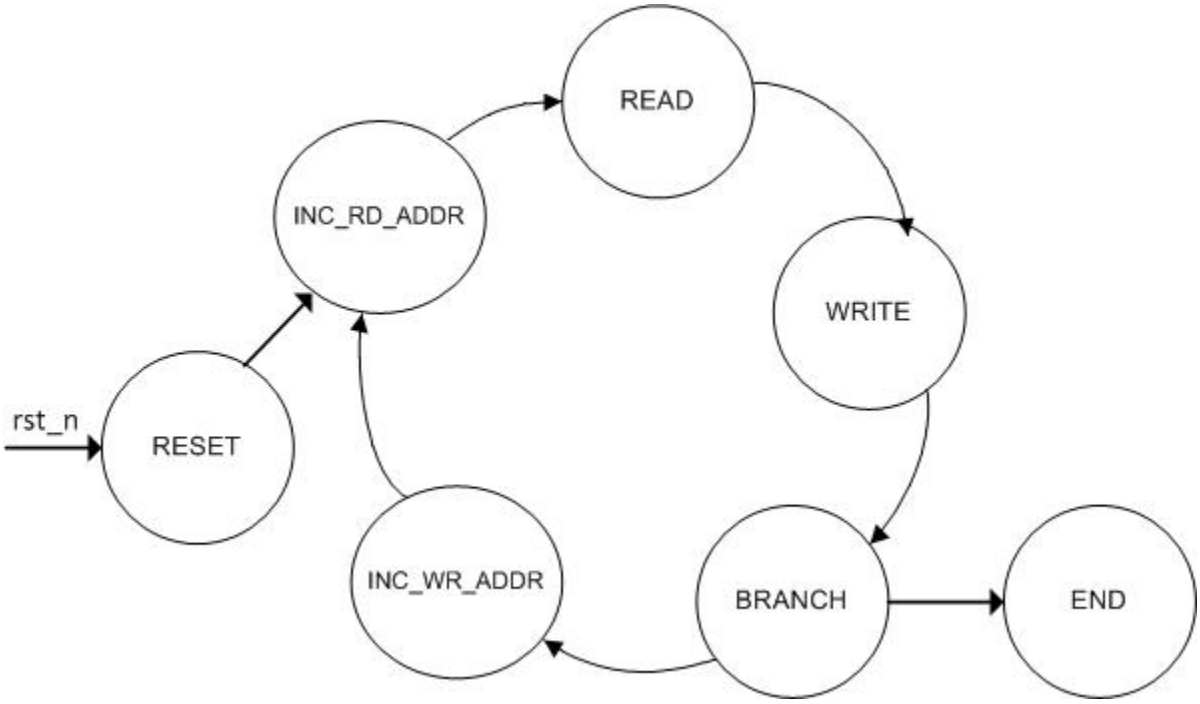

Figure 2.25 The state machine of Energy block

Table 2.37 The function of states in Energy block

	Present state	**Function**
1	RESET	Reset system
2	INC_RD_ADDR	Increase the address in one frame to read the logarithm of energy results
3	READ	Read the logarithm of energy result memories
4	WRITE	Write the logarithm of energy result into final memories
5	BRANCH	Check the process of energy calculation corresponding with all frame
6	INC_WR_ADDR	Increase the address to write the logarithm of energy results into final memories
7	END	Finish

Table 2.38 State machine in Energy block

	Present state	**Next state**	**Condition**
1	RESET	INC_RD_ADDR	When receive an enable signal from Main Control
2	INC_RD_ADDR	READ	After 1 clock
3	READ	WRITE	The counter is over

4	WRITE	BRANCH	The counter is over
5	BRANCH	INC_WR_ADDR	Logarithm of energy results writing for one frame has not yet done
		END	Logarithm of energy results writing for one frame has already done
6	INC_WR_ADDR	INC_RD_ADDR	After 1 clock

Table 2.39 Input and output of Energy block

Pin name	**Type**	**#Bit**	**Description**
clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
copy_energy_en	Input	1	An enable signal allows Copy_energy block implements
frame_num	Input	8	Number of frame
energy_mem_read_addr	Output	12	The address of input data from logarithm of energy results - memory
energy_mem_write_addr	Output	14	The address of logarithm of energy results for memory writing
write_energy_to_result_en	Output	1	An enable signal allows to write logarithm of energy results into final memories

Table 2.40 The function of states'input/output of Energy block

Pin name	**# Bit**	**Description**
counter_en	1	Enable pin of counter module
counter_frame_en	1	Enable pin of counter_frame module
inc_read_addr_en	1	Enable pin of address increasing module which used to read the logarithm energy result memories
write_energy_to_result_en	1	An enable signal allows to write logarithm of energy results into final memories
counter_value	4	Value of counter

Table 2.41 Output of states in Energy block

State	**Output of state**

RESET	counter_en <= 1'b0;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b0;
	write_energy_to_result_en <= 1'b0;
	counter_value <= 4'd0;

INC_WR_ADDR	counter_en <= 1'b0; counter_frame_en <= 1'b1;

	inc_read_addr_en <= 1'b0;
	counter_value <= 4'd0;
	write_energy_to_result_en <= 1'b0;

READ	counter_en <= 1'b1;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b0;
	counter_value <= LOOPS_READ;
	write_energy_to_result_en <= 1'b0;

WRITE	counter_en <= 1'b1;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	write_energy_to_result_en <= 1'b1;

BRANCH	counter_en <= 1'b0;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b0;
	counter_value <= 4'd0;
	write_energy_to_result_en <= 1'b0;

INC_RD_ADDR	counter_en <= 1'b0;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b1;
	counter_value <= 4'd0;
	write_energy_to_result_en <= 1'b0;

END	counter_en <= 1'b0;
	counter_frame_en <= 1'b0;
	inc_read_addr_en <= 1'b0;
	counter_value <= 4'd0;
	write_energy_to_result_en <= 1'b0;

2.2.2.9 Delta and Delta-Delta block

Delta block’s calculation is done by taking the deviation of cepstral results, which is based on approximation formulas of first and second order derivative. The dataphath and state diagram show sequential oeprations following basic conditions:

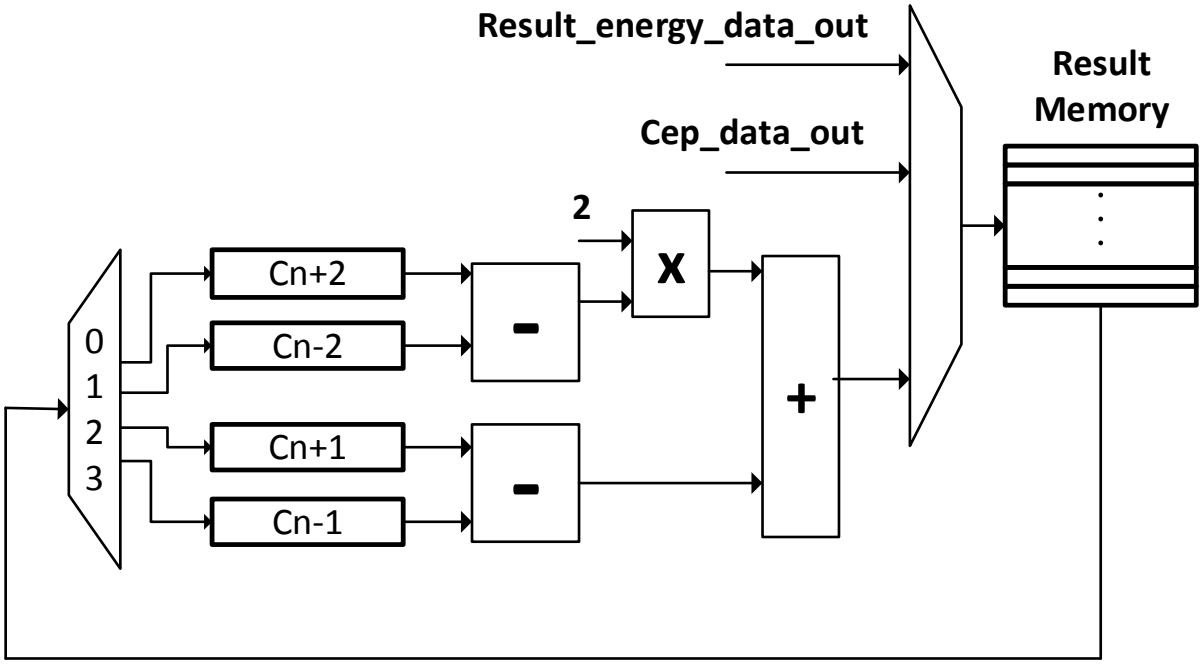

Figure 2.26 Delta and Delta-Delta block

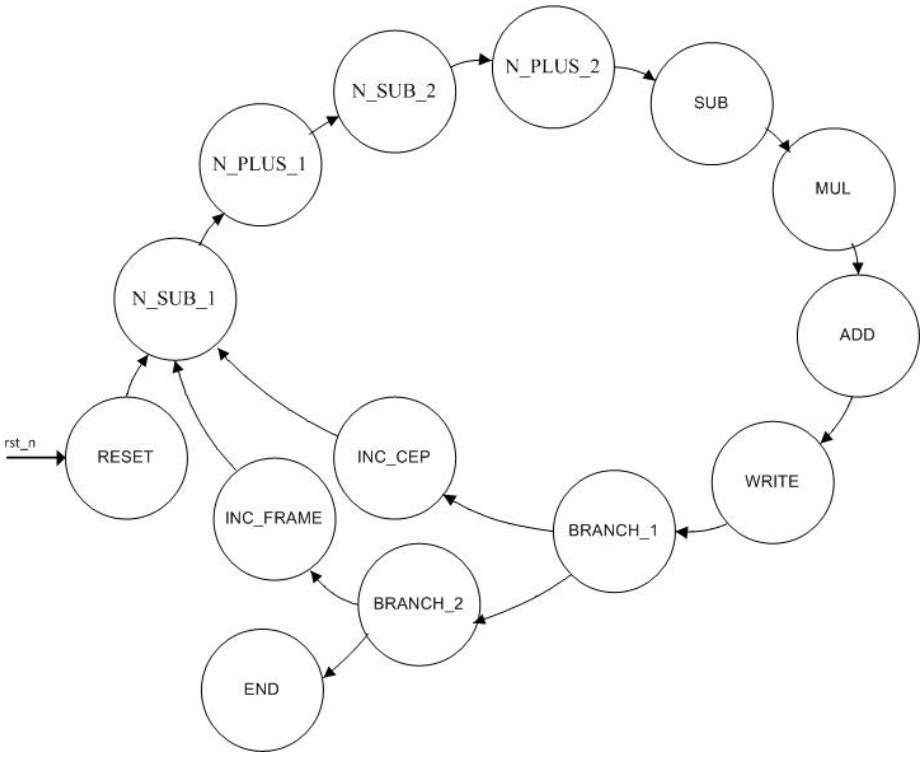

Figure 2.27 The state machine of Delta and Delta-Delta block

Table 2.42 The function of states in Delta and Delta-Delta block

	Present state	**Function**
1	RESET	Reset system
2	N_SUB_1	Read and write the value of C_{n-1} into a corresponding register
3	N_PLUS_1	Read and write the value of C_{n+1} into a corresponding register
4	N_SUB_2	Read and write the value of C_{n-2} into a corresponding register
5	N_PLUS_2	Read and write the value of C_{n+2} into a corresponding register
6	SUB	Calculate C_{n+2} - C_{n-2} and C_{n+1} - C_{n-1}
7	MUL	Calculate $2(C_{n+2} - C_{n-2})$
8	ADD	Calculate the Delta result
9	WRITE	Write the element which has been computed into result memories
10	BRANCH_1	If this frame has already done then the next state is BRANCH_2. If has not yet finished then continue this task
11	BRANCH_2	If all of frames have already done with the number of delta. If has not yet finished then continue this task
12	INC_CEP	Increase the cepstral serial for Cepstral calculation
13	INC_FRAME	Increase the frame serial for Delta calculation
14	END	Finish

Table 2.43 State machine in Delta and Delta-Delta block

	Present state	**Next state**	**Condition**
1	RESET	N_SUB_1	When receive an enable signal from Main Control
2	N_SUB_1	N_PLUS_1	The counter is over
3	N_PLUS_1	N_SUB_2	The counter is over
4	N_SUB_2	N_PLUS_2	The counter is over
5	N_PLUS_2	SUB	The counter is over
6	SUB	MUL	The counter is over
7	MUL	ADD	The counter is over
8	ADD	WRITE	The counter is over
9	WRITE	BRANCH_1	The counter is over
10	BRANCH_1	BRANCH_2	One frame has already done
		INC_CEP	One frame has not yet done
11	BRANCH_2	INC_FRAME	All of frame have not yet done
		END	All of frame have already done
12	INC_CEP	N_SUB_1	After 1 clock
13	INC_FRAME	N_SUB_1	After 1 clock
14	END	END	After 1 clock

Table 2.44 Input and output of Delta and Delta-Delta block

Pin name	Type	#Bit	Description
Clk	Input	1	A synchronous clock signal
rst_n	Input	1	An asynchronous reset signal
delta_state_en	Input	1	An enable signal allows Delta block implements
frame_num	Input	8	Number of frame
cep_num	Input	6	Number of cepstral filter
delta_data_in	Input	32	Output data from Cepstral and Energy which use to read into Delta block
delta_mem_read_write_addr	Output	12	The address of data input reading and data output writing
delta_data_out	Output	32	The result of Delta
write_delta_en	Output	1	An enable signal allows to write Delta results

Table 2.45 The function of states'input/output of Delta and Delta-Delta block

Pin name	# Bit	Description
sel_n	2	Selected pin which choose the corresponding address and register to store data in C_{n+2} , C_{n-2} , C_{n+1} and C_{n-1}
write_delta_en	1	An enable signal allows to write Delta result memories
counter_en	1	Enable pin of counter module
add_en	1	Enable pin of addition module
sub_en	1	Enable pin of subtraction module
mul_en	1	Enable pin of multiplication module
inc_frame_en	1	Enable pin of "n" counter module which used in Delta formula
inc_cep_en	1	Enable pin of cepstral number counter module which corresponding with the number of Delta value for each frame
sel_addr	1	Selected pin which decide the address of *delta_mem_read_write_addr* become read or write address
counter_value	4	Value of counter

Table 2.46 Output of states in Delta and Delta-Delta block

State	Output of state

RESET	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

START	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b1;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b1;

READ_0	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

MUL_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;

	counter_value <= LOOPS_MUL;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

ADD_0	sel_add <= 2'b00;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

BRANCH_1	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

BRANCH_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b1;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

DATA_CEP_1	sel_add <= 2'b11; counter_en <= 1'b0;

	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b1;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

DATA_CEP_2	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b1;
	frame_num_out_en <= 1'b0;

WRITE_CEP_1	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b1;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

WRITE_CEP_2	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= LOOPS_WRITE;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b1;

	frame_count_en <= 1'b0; frame_num_out_en <= 1'b0;

WAIT	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b00;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

RE_CALC	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

CAL_ADDR	sel_add <= 2'b00;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b1;
	change_addr_sel <= 2'b01;
	mul_en <= 1'b0;
	add_en <= 1'b0;
	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

READ	sel_add <= 2'b11;
	counter_en <= 1'b0;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b0;

	counter_value <= 4'd0;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

MUL	sel_add <= 2'b01;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b1;
	add_en <= 1'b0;
	counter_value <= LOOPS_MUL;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

ADD	sel_add <= 2'b11;
	counter_en <= 1'b1;
	counter_half_frame_en <= 1'b0;
	change_addr_sel <= 2'b11;
	mul_en <= 1'b0;
	add_en <= 1'b1;
	counter_value <= LOOPS_ADD;
	counter_cep_en <= 1'b0;
	addr_cep_en <= 1'b0;
	write_cep_en <= 1'b0;
	frame_count_en <= 1'b0;
	frame_num_out_en <= 1'b0;

2.3 Conclusion

The combination of pipelined method in controls and 32-bit floating point numbers throughout the system, so that proposed MFCC architecture has achieved those expectations:

- Ability to communicate with APB interface standards adapted to all systems with the support from ARM cores.
- Be able to set dynamic configurations with a large limitation and covers most of the architecture proposed in the current paper research.
- The high ability to reuse this design from independent proposed algorithm of the separate state machines.

- In this architecture controller, the application for constant voice sentence can be feasible with the support by the software in cutting continuous data into small one, which be enough to store in internal memory.

CHAPTER 3: EXPERIMENTS AND RESULTS

3.1 Simulation Environment

The purposed design is implemented on the Linux operating system with VCS compliler. Folder tree is described in the figure below. In the *01_Design* folder that contains all Verilog files of architecture, *02_Simulate* include the required configuration of tescases *.04_Script* generates the Perl file to create memory file, which contains the necessary coefficients as Mel coefficients, Window coefficients, LUT table and so on.

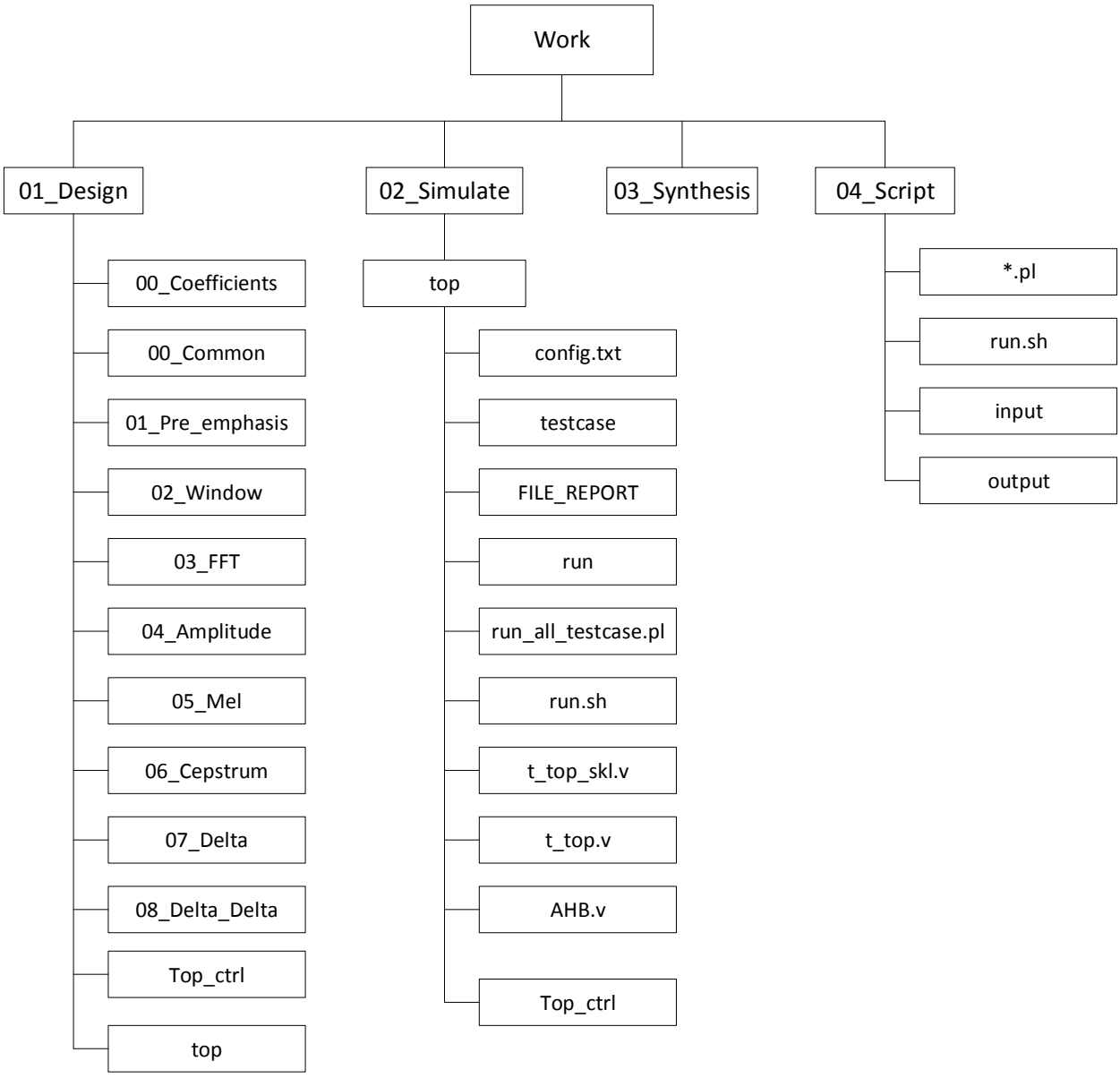

Figure 3.1 Tree folder of design in Linux environment

3.2 Testcase and Configurations

As well as MFCC architectural configuration parameters are described in chapter 1. Thus these testcases will focus on the accuracy of MFCC hardware results, which compared with the results was calculated by Matlab when the configuration parameters is changed.

Table 3.1 MFCC configuration for testcases

Testcase	Sample in frame	Overlap ratio	Mel filter	Cepstral
1	320	50%	63	31
2	320	50%	12	22
3	320	50%	50	21
4	400	40%	63	31
5	160	50%	30	12

3.3 Accuracy Estimation

In order to satisfy essential silicon requirements correctly, full ASIC design flow with 130nm technology is applied in this work, in which function of MFCC is compared to Matlab model with many different level such as RTL, gate level netlist (pre_layout and post_layout). Final performance such as area, timing report or power is confirmed after finishing physical design step. Nonetheless, Design For Test (DFT) step is not inserted into this flow due to unnecessary requirement with MFCC architecture as IP level. As regards verifying environment, a multi-language environment comprising of Perl, Bash-Shell, Verilog are established. To be more detail, a setup file as Figure 3.2 containing necessary parameter as Table 3.1 is read firstly. Following every line of such file, such parameters are called by a CPU model, which transfers them through ABH to internal registers in MFCC architecture. Based such convenient environment as Figure 3.3, a wide range of MFCC are verified automatically and compared with Matlab model exactly.

```
          #Frame_num    Sample_in_frame    FFT_num      FFT_Stage_Number    Mel_num    Cep_num
Config_1    7'd24          11'd320         11'd512          4'd9             6'd63      7'd31
Config_2    7'd24          11'd320         11'd512          4'd9             6'd12      7'd22
Config_3    7'd24          11'd320         11'd512          4'd9             6'd50      7'd21
Config_4    7'd16          11'd400         11'd512          4'd9             6'd63      7'd31
```

Figure 3.2. Some MFCC configurations in configured file

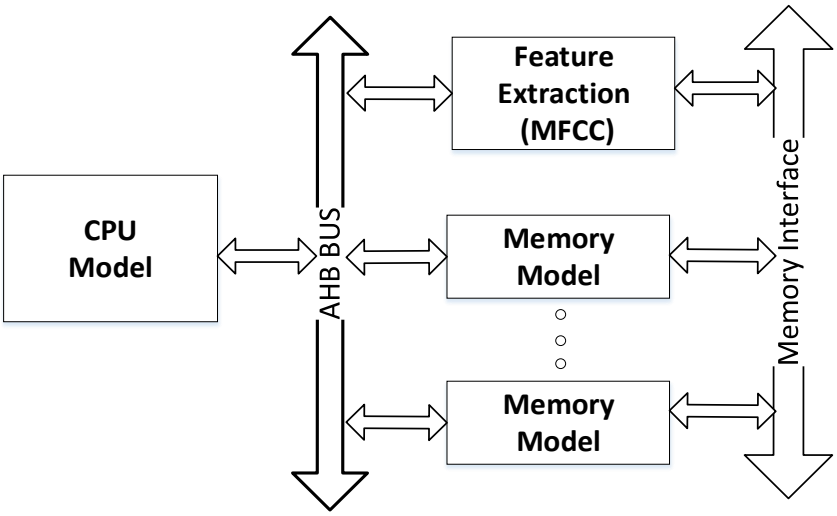

Figure 3.3. MFCC architecture and AHB interface

The table below presents the biggest error of the results for each testcase.

Table 3.2 The largest error from each testcase

Testcase	Cepstral Error	Energy Error	Delta Error	Delta-Delta Error
1	4.31E-4	4.17E-4	0.0013	0.0054
2	1.64E-4	4.17E-4	8.32E-4	0.0026
3	3.03E-4	4.17E-4	0.0012	0.0041
4	4.41E-4	3.15E-4	0.0013	0.0052
5	2.77E-4	6.33E-4	0.0012	0.0041

The average error of each testcase is calculated by using the formula:

$$E_i = \frac{\sum_{j=1}^{n} \left| x_{matlab_j} - x_{hardware_j} \right|}{n} \quad (3.1)$$

Where:

E_i is an average error of MFCC vector for ith frame

x_{matlab} is the MFCC calculated by Matlab

$x_{hardware}$ is the MFCC calculated by Verilog

n is the number of components for MFCC results in each frame, including the combination of energy, cepstral, delta and delta-delta.

Table 3.2 The average error in each testcase

Frame	Testcase 1	Testcase 2	Testcase 3	Testcase 4	Testcase 5
1	2.245E-04	9.508E-05	2.330E-04	2.112E-04	2.114E-04
2	2.007E-04	9.452E-05	2.051E-04	2.047E-04	1.600E-04
3	2.433E-04	8.446E-05	1.958E-04	1.920E-04	1.296E-04
4	1.932E-04	6.334E-05	1.954E-04	1.857E-04	9.593E-05
5	2.130E-04	1.024E-04	1.540E-04	2.039E-04	1.434E-04
6	1.987E-04	8.808E-05	1.935E-04	2.282E-04	1.312E-04
7	1.874E-04	1.029E-04	1.613E-04	2.325E-04	1.364E-04
8	1.831E-04	1.052E-04	1.883E-04	2.180E-04	1.413E-04
9	2.207E-04	9.237E-05	1.602E-04	2.397E-04	9.086E-05
10	2.282E-04	8.929E-05	1.819E-04	1.908E-04	1.480E-04
11	2.024E-04	9.720E-05	2.003E-04	1.867E-04	1.601E-04
12	2.555E-04	7.680E-05	1.993E-04	1.960E-04	1.361E-04
13	1.990E-04	7.880E-05	1.535E-04	2.112E-04	1.130E-04
14	1.676E-04	9.244E-05	2.261E-04	2.047E-04	1.265E-04
15	2.426E-04	7.605E-05	2.030E-04	1.920E-04	1.569E-04
16	2.553E-04	7.044E-05	1.872E-04	1.857E-04	1.627E-04
17	2.714E-04	7.380E-05	1.797E-04		1.369E-04
18	1.911E-04	1.170E-04	2.166E-04		1.738E-04
19	2.144E-04	1.043E-04	1.689E-04		1.188E-04
20	2.565E-04	7.219E-05	2.169E-04		1.421E-04
21					1.808E-04
22					1.213E-04
23					1.176E-04
24					1.839E-04
25					1.750E-04
26					9.737E-05
27					1.512E-04
28					1.254E-04
29					1.616E-04
30					1.653E-04
31					1.646E-04
32					1.284E-04
33					1.734E-04
34					1.299E-04
35					1.374E-04
36					1.523E-04
37					1.727E-04
38					1.970E-04
39					1.014E-04

40					1.499E-04
41					1.255E-04
42					1.329E-04
43					1.714E-04
44					1.541E-04
45					1.185E-04

Table 3.4 Maximum MFCC configuration

Sample in frame	**Overlap rate (%)**	**FFT num**	**Mel filters**	**Cep**	**Delta order**
1024	50	1024	63	31	Two levels

Through above verifying environment, largest configuration as Table 3.4 has experienced maximum measurement uncertainty with $4.17x10^{-4}$ for Energy, $4.31x10^{-4}$ for Cepstral and 0.0013 for Delta in comparison to one in [13] with an average of 10^{-3}. This configuration has also witnessed delay 2.348 ms with maximum achieved frequency at 500 MHz, which adapt real-time requirement.

CHAPTER 4: PHYSICAL PERFORMANCE

4.1 Gate Level Netlist Results From RTL

In order to synthesize MFCC architecture from RTL level to gate level netlist, the environmental setting and selection tools are needed. The following table summarizes the details of the platform, which serves the synthesis of this purposed architecture:

Table 4.1 Systhesis platform of MFCC architecture

Operation System	Red Hat Enterprise Linux 5
Design Language	Verilog
Environment Language	Bash Shell, Perl, Tcl
Editor tool	Vi
Synthesis Tool	Design Compiler (Synopsys)
Compiler Tool	VSC (Synopsys)
Synthesis Library	TSMC 130 nm
Constraint	• Input delay • Output delay • Clock frequency • Internal memory • Asynchronous Reset • Clock delay

TSMC 130nm library of synthesis is structured as follows:

```
library/
`-- TSMC_130nm
    `-- sc-x
        |-- cell_list
        |-- symbols
        |   |-- cadence
        |   |-- edif
        |   `-- synopsys
        |-- synopsys
        |   |-- README
        |   |-- scx2_tsmc_cl013g_ff_1p32v_0c.db
        |   |-- scx2_tsmc_cl013g_ff_1p32v_0c.lib
        |   |-- scx2_tsmc_cl013g_ff_1p32v_m40c.db
        |   |-- scx2_tsmc_cl013g_ff_1p32v_m40c.lib
        |   |-- scx2_tsmc_cl013g_ss_1p08v_125c.db
        |   |-- scx2_tsmc_cl013g_ss_1p08v_125c.lib
        |   |-- scx2_tsmc_cl013g_tt_1p2v_25c.db
        |   `-- scx2_tsmc_cl013g_tt_1p2v_25c.lib
        `-- verilog
            |-- README
            |-- tsmc13.v
```

Synthesis environmental structure is shown as follows:

```
./
|-- formality.log
|-- README
|   |-- README.DC-RM.txt
|-- reports
|   |-- vr.check_design.rpt
|   |-- vr.fmv_unmatched_points.rpt
|   |-- vr.mapped.area.rpt
|   |-- vr.mapped.clock_gating.rpt
|   |-- vr.mapped.power.rpt
|   |-- vr.mapped.qor.rpt
|   `-- vr.mapped.timing.rpt
|-- results
|   |-- vr.compile_ultra.ddc
|   |-- vr.elab.ddc
|   |-- vr.mapped.ddc
|   |-- vr.mapped.sdc
|   |-- vr.mapped.svf
|   `-- vr.mapped.v
|-- rm_dc_scripts
|   |-- dc.dft_autofix_config.tcl
|   |-- dc.dft_occ_config.tcl
|   |-- dc.mcmm.scenarios.tcl
|   |-- dc.tcl
|   |-- dc_top.tcl
|   |-- fm_gate2gate.tcl
|   |-- fm.tcl
```

Table 4.2 Specification of Synthesis Environment For MFCC Architecture

Name	**Description**
README	Contain the instruction of all running file in design
Reports	Contain all report files after simulation
Results	Contain all MFCC results
rm_dc_scripts	Contain all executive file, which is generated by TCL script
rm_setup	Contain all constraint files of this design
rtl	Contain all RTL files
run_dc	Contain all executive file, which is used for synthesis
run_fm	Contain all executive file, which estimate RTL and gate level netlist
run_fm_gate	Contain all executive file, which estimate the process of pre layout
wire_load	Contain all evaluation parameter for total delay and datapath delay

The constraints are expressed as follows:

```
create_clock  -name clk [get_ports clk] -period 2 -waveform {0 1}
set_clock_uncertainty 0.1 clk
set_clock_latency 0.1 clk
set_clock_transition 0.1 clk
set_dont_touch_network clk

#create_clock  -name clk_sys [get_ports clk_sys] -period 1 -waveform {0 0.5}
#set_clock_uncertainty 0.1 clk_sys
#set_clock_latency 0.1 clk_sys
#set_clock_transition 0.1 clk_sys
#set_dont_touch_network clk_sys

set_dont_touch rst_n
set_ideal_network rst_n

#set_max_fanout 50 [all_inputs]
#set_max_fanout 50 [all_outputs]
set_input_delay -clock clk  0.1  [all_inputs]
set_output_delay -clock clk 0.1  [all_outputs]
set_wire_load_model -name zero-wire-load-model
#set_load 10 [all_outputs]
```

In order to tackle with these constraints, the purposed design is synthesized with T = 2 ns or f = 500 MHz and receive the reports about timing, power, area in table below:

Table 4.3 The relationship between frequency and gate count

Frequency (MHz)	**Total equivalent gate count (# cells)**
100	29684971
200	29803786
250	29701747
500	30186098

The table above shows the relationship between the size and frequency. When changing the frequency of the synthesis, the area will change to meet the constraint. When the signal path is too long, which cause the negative slack index, this issues require to modify the design at the register level (RTL). Some improved methods are listed:

- Change the algorithm of design to get the lowest delay on the data path (RTL step).
- Insert the DFF (D Filp-Flop) on the data path to subdivide it into the smaller delay to satisfy the requirement of frequency.
- Arrange the padded cells in the physical design step (Place and Route) to remove the delay of cell components, which located on the data path.

4.2 Place and Route

Place and route is the final step in ASIC design process and the results of this step is illustrated in some figures below:

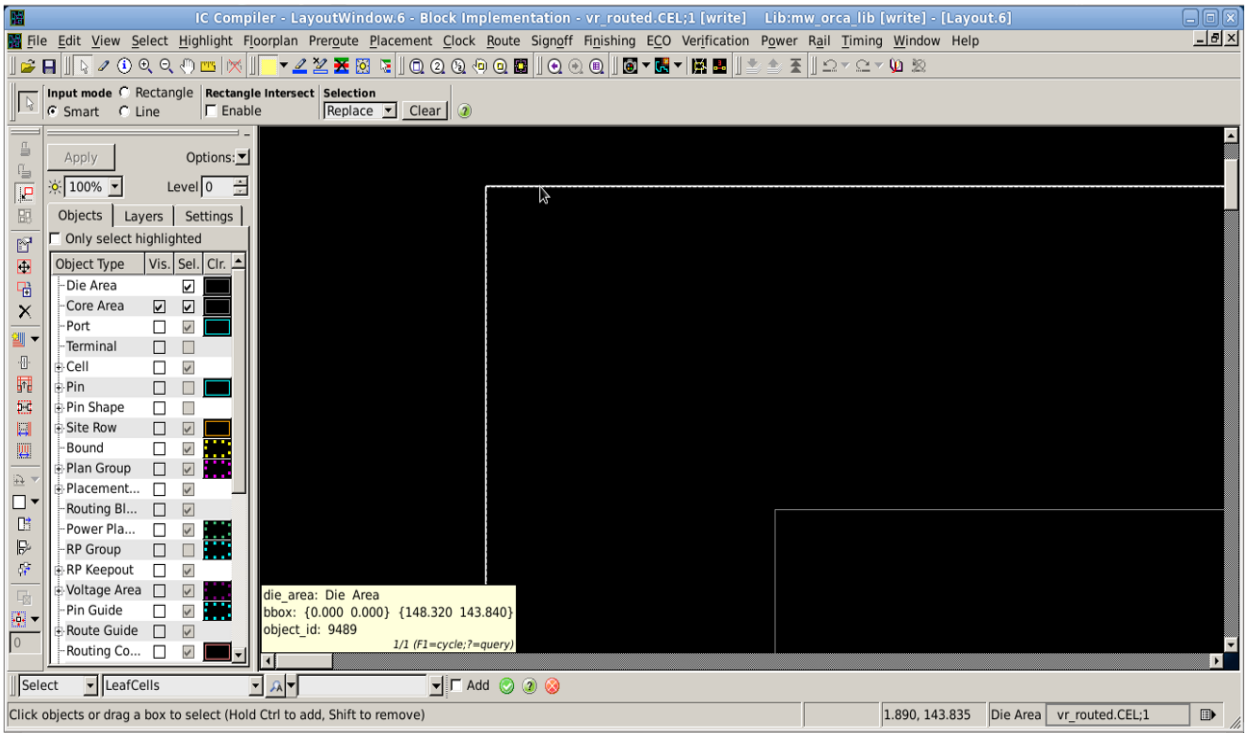

Figure 4.1 Die area of MFCC architecture

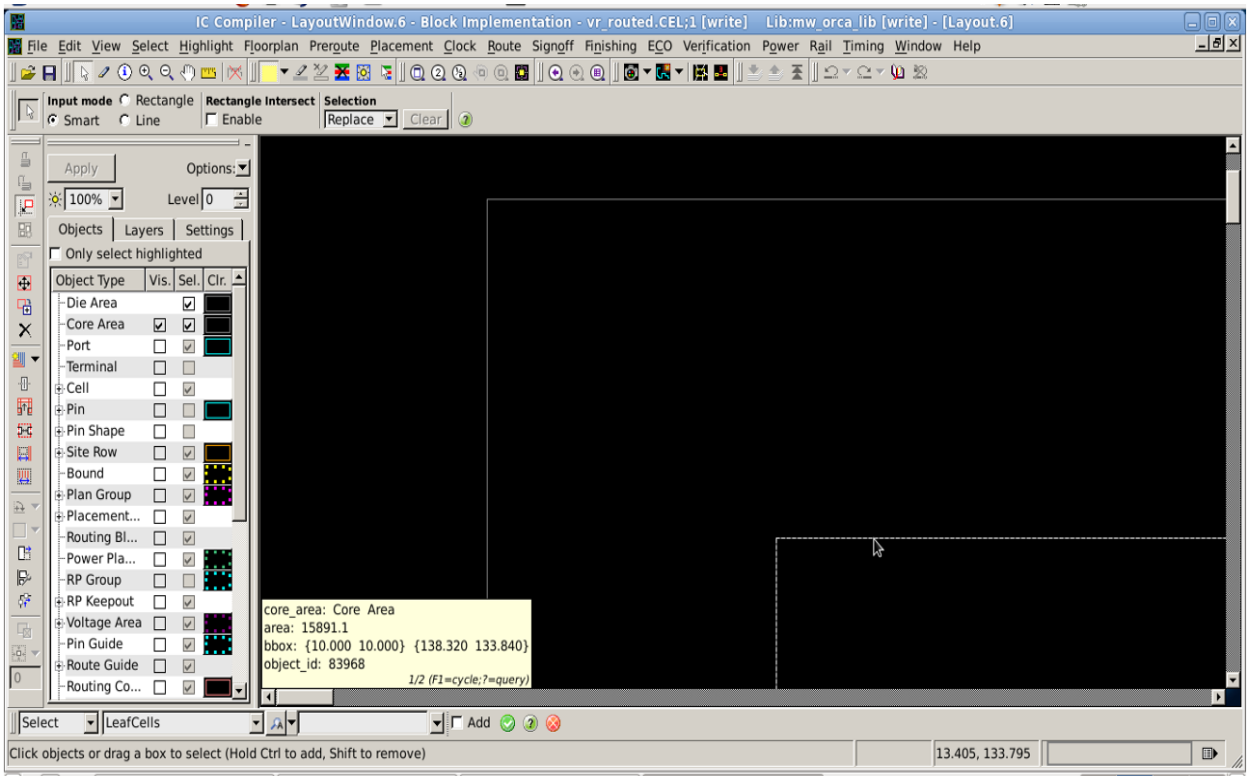

Figure 4.2 Core area of MFCC architecture

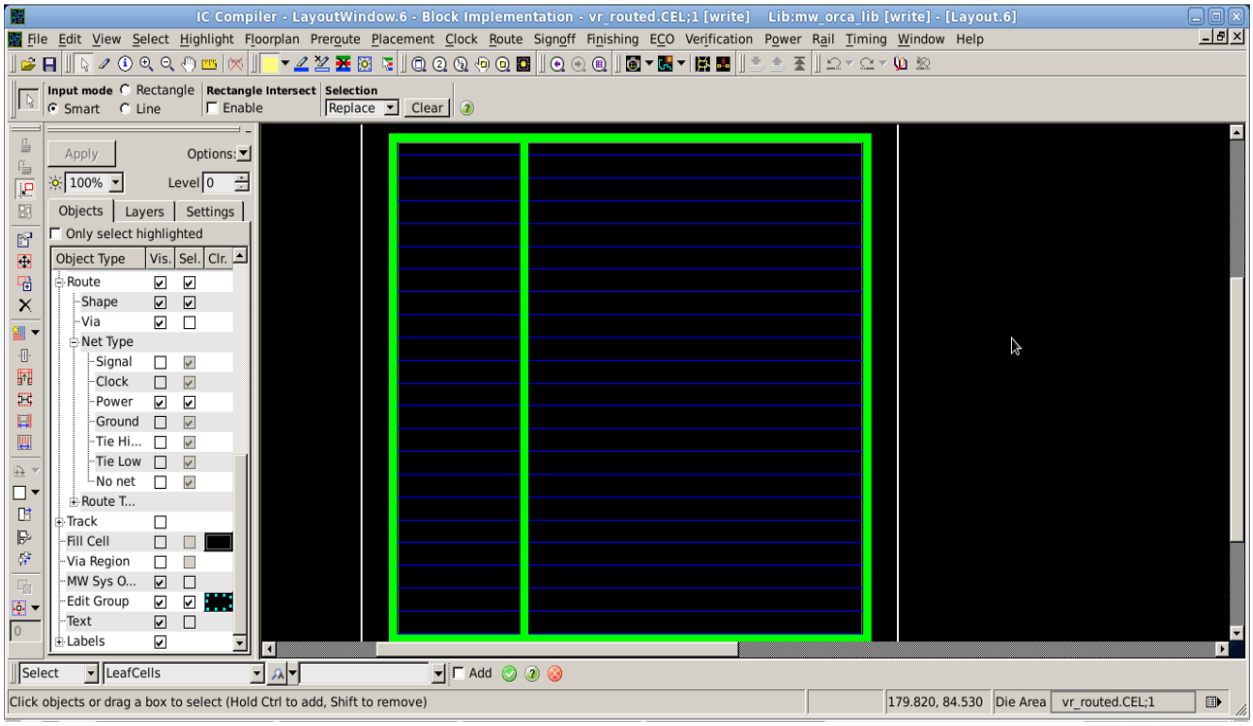

Figure 4.3 VCC and source routing result

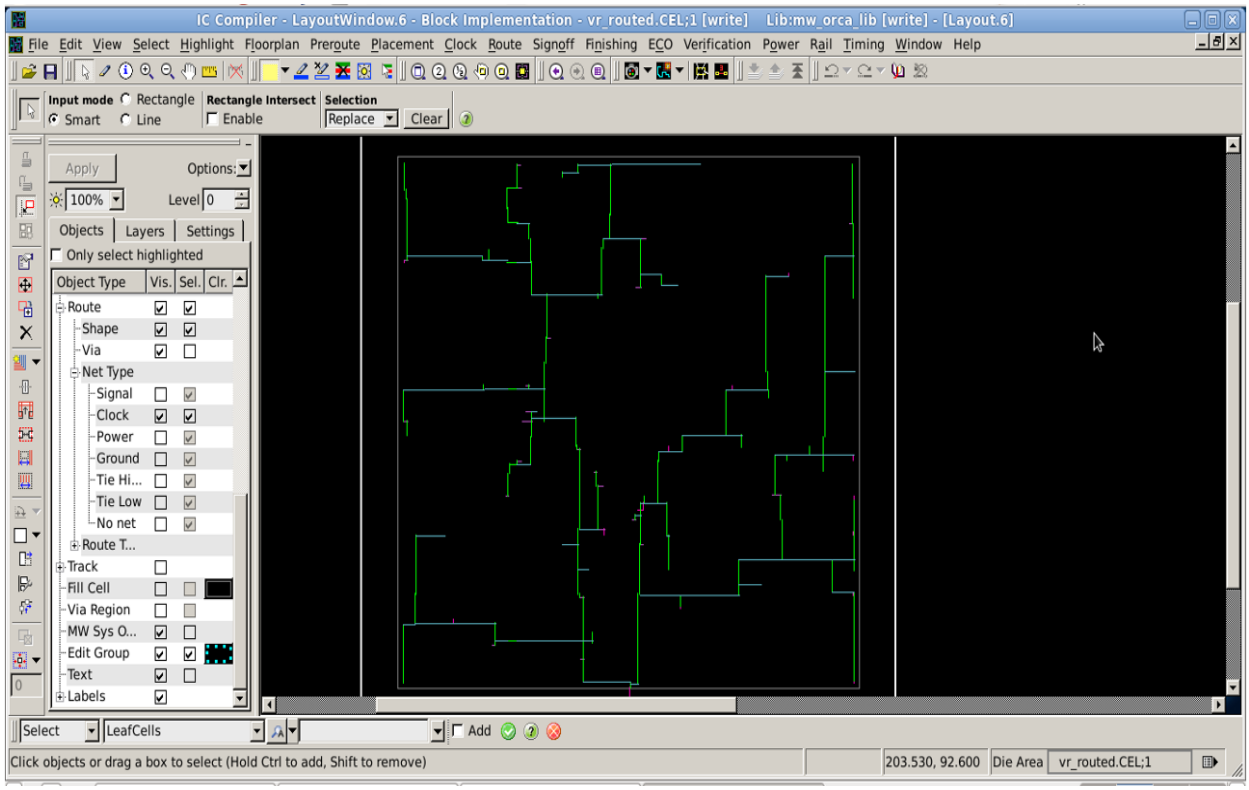

Figure 4.4 Clock routing result

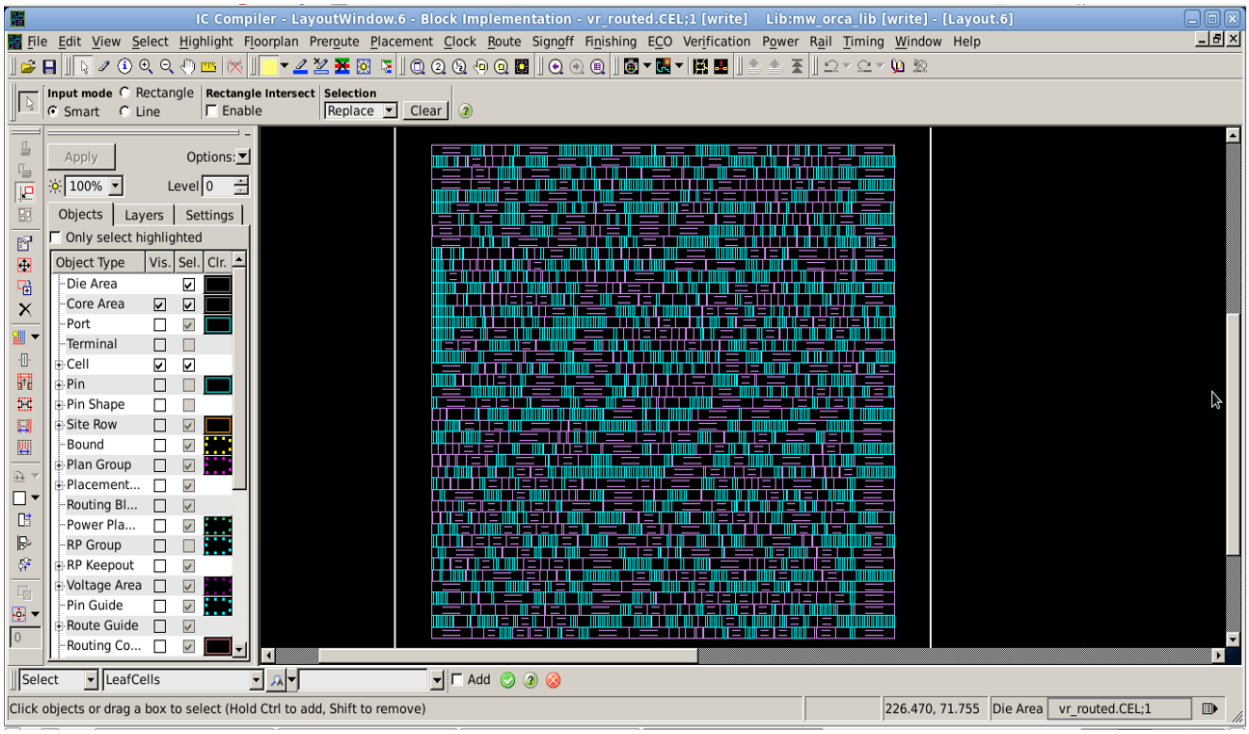

Figure 4.5 Result of Placement processing

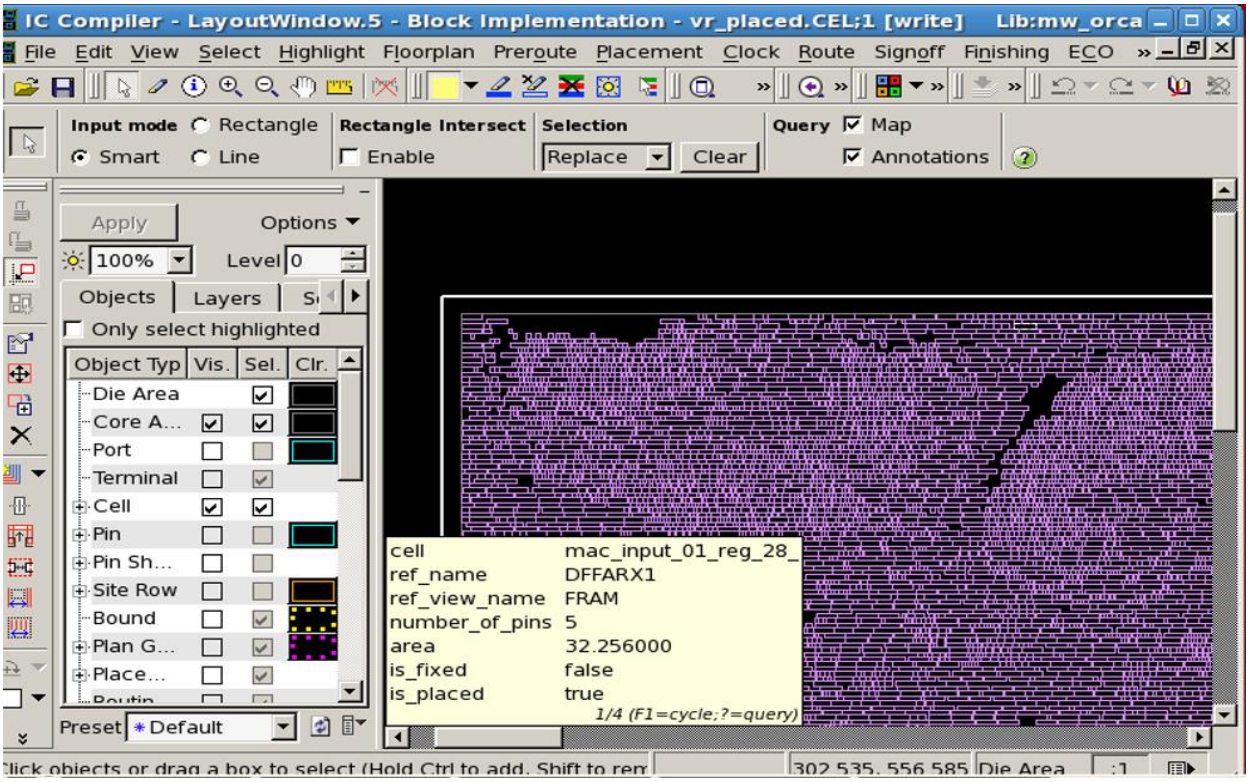

Figure 4.6 All cells in MFCC architecture

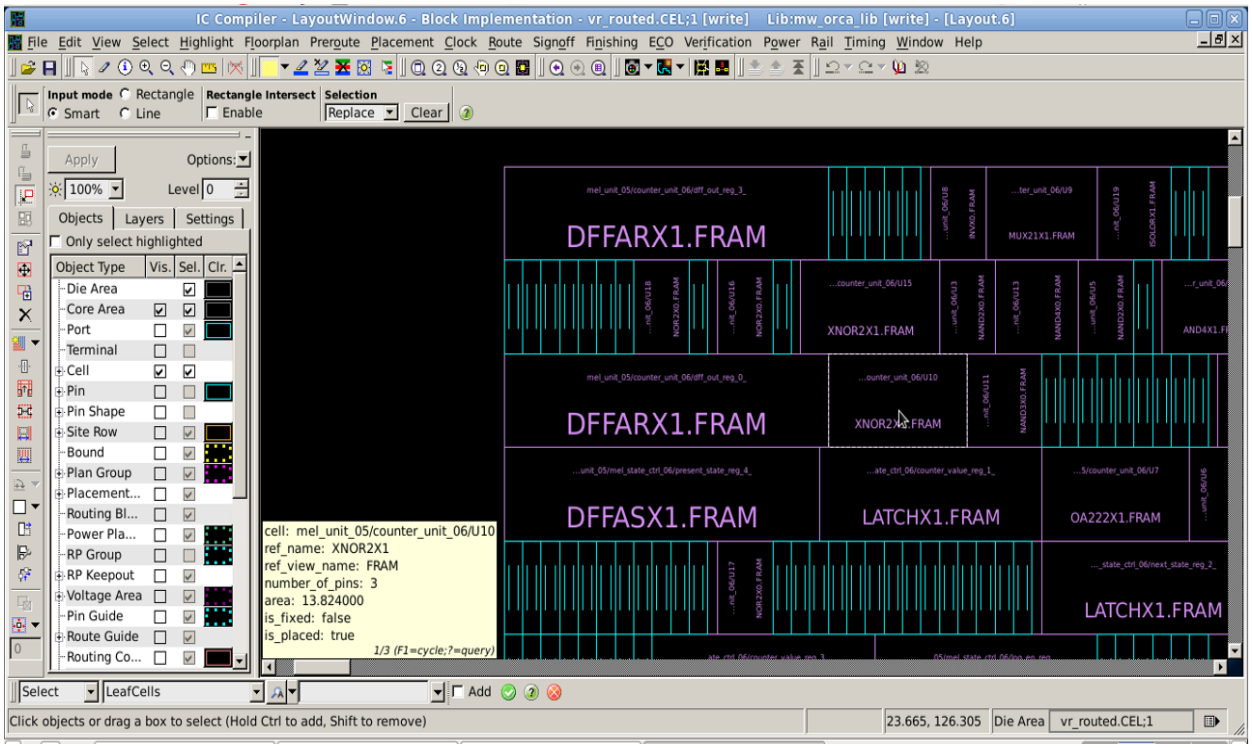

Figure 4.7 XNOR cells in MFCC architecture

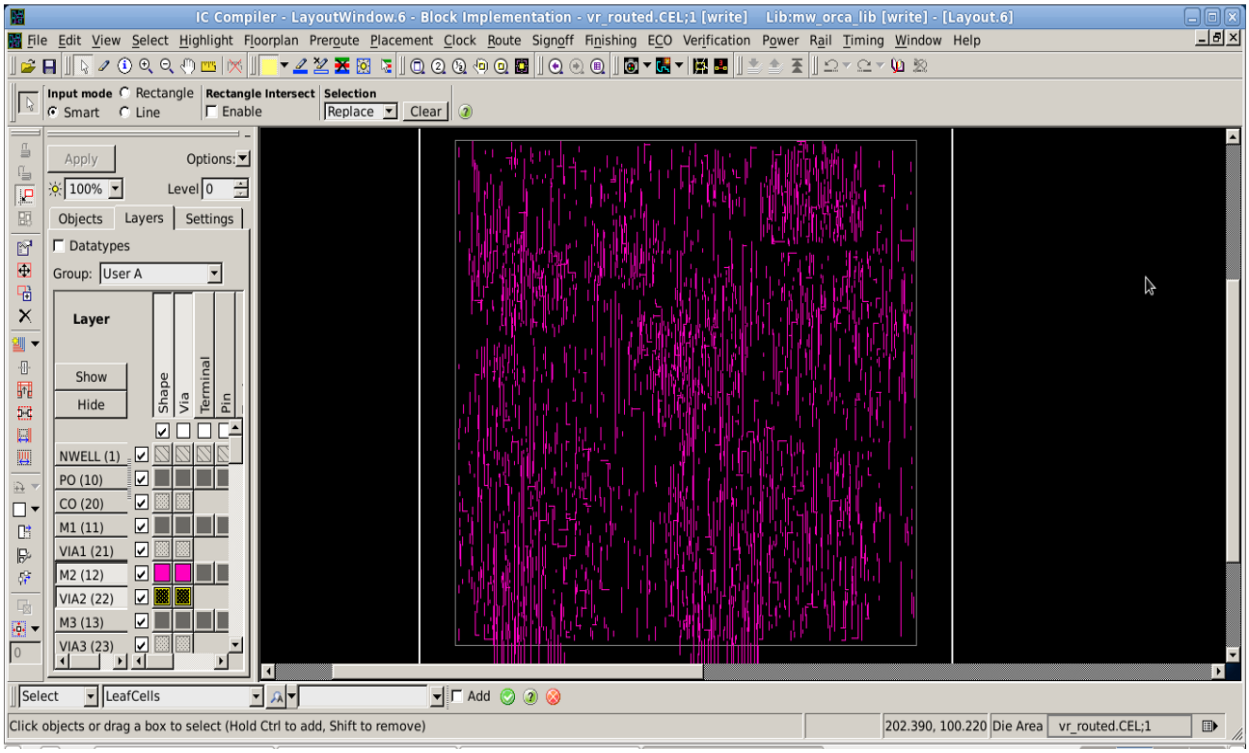

Figure 4.8 Second metal of MFCC architecture

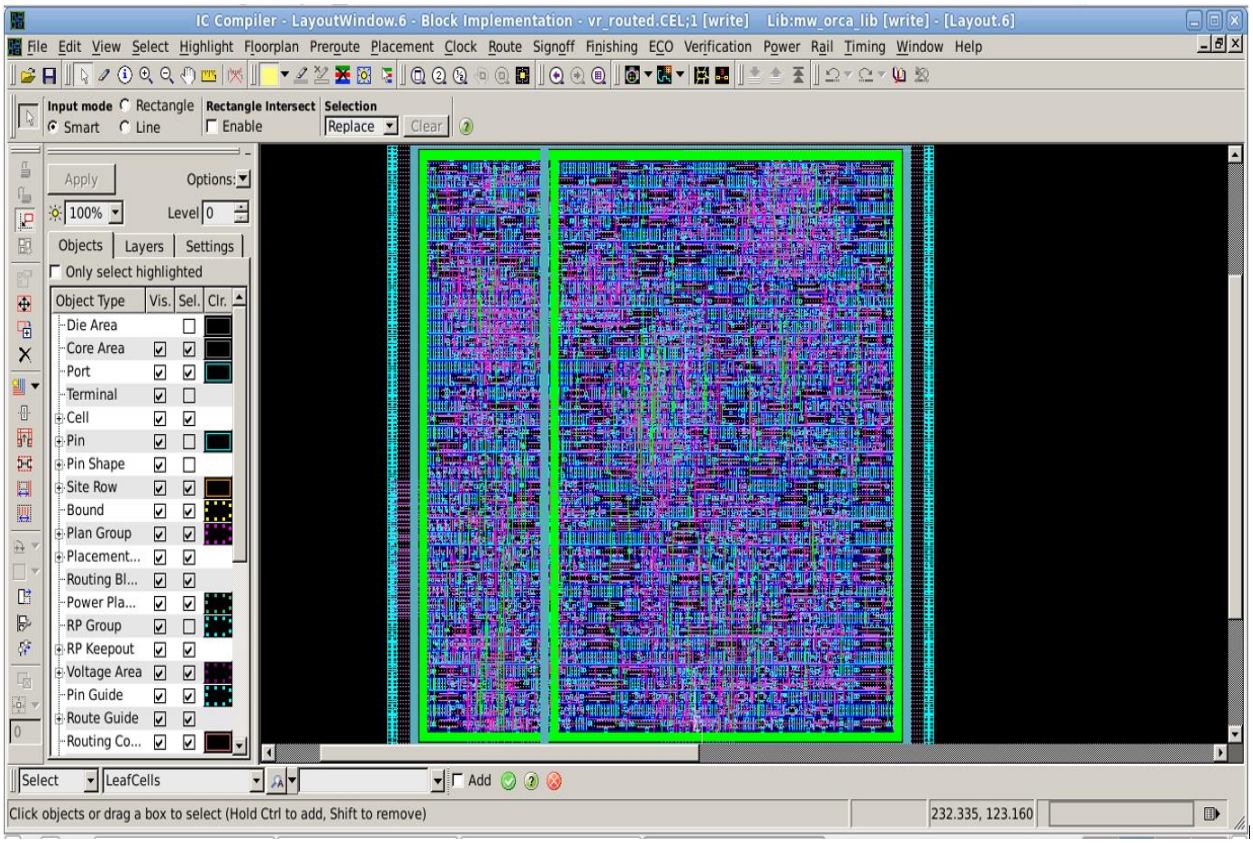

Figure 4.9 The final layout of MFCC architecture

CHAPTER 5: CONCLUSION AND FUTURE WORK

After accomplishing all steps of ASIC design flow in Figure 5.1, final results of physical performance extracted from layout results such from Figure 4.7 to Figure 4.9 are presented and compared with other references as Table 5.1. Such this table indicates that high efficiency of proposed architecture is experienced through maximum index of feature number at 96 opposed to only 12 and 48 in [9] and [11], which is reasonable for ability of reconfiguration feasibly. In addition to this, high silicon performance as regards frequency at 500 MHz as well as little area at 12.9x12.9 mm^2 is witnessed in comparison to other references.

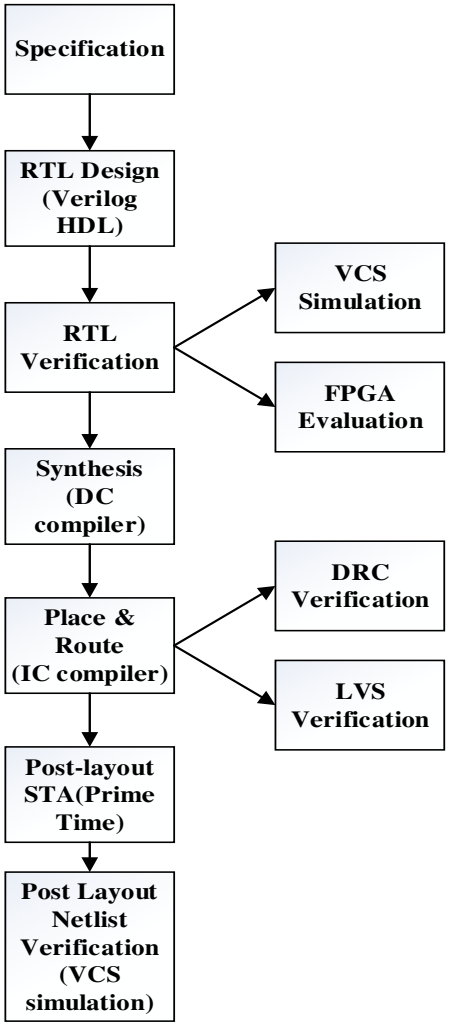

Figure. 5.1. ASIC design flow

Table 5.1 MFCC Perpormance Comparison

Architecture	Proposed Architecture	[9]	[11]
Implementation	ASIC (130nm)	ASIC (0.6µm)	ASIC (0.18µm)
FFT points	8 - 1024	256	256
Mel	1 - 63	20	32
Cep	1 - 31	12	13
Feature Number	96 (Maximum configuration)	12	48
Area (mm^2)	12.9x12.9	3.2x3.3	6.5x3.5
Frequency (MHz)	500	50	30

In this thesis, a dynamic ASIC-based MFCC design is proposed to confirm the effective performance compared with other schemes. The promoted MFCC hardware architecture not only solves many applications basing on adapt a wide range of requirement such as accuracy problem, real-time issues, ability of reconfiguration feasibly but also adapts silicon requirements such high frequency matching AHB standard, reused ability as well as resource limitation. The obtained experimental results validated on 130 nm technology have also experienced low cost design for final fabrication and ability of integration towards entire systems. As regards future works, a complete hardware-based ASR system is our target to achieve powerful integrated circuit. Moreover, other applications integrating proposed MFCC will be also implemented to estimate the MFCC hardware performance obviously.

References.

[1] Veton Z. Këpuska, Mohamed M. Eljhani, Brian H. Hight, "Wake-Up-Word Feature Extraction on FPGA," *World Journal of Engineering and Technology,* 2014.

[2] Eslam Mansour mohammed,Mohammed Sharaf Sayed, "LPC and MFCC Performance Evaluation with Artificial Neural Network for Spoken Language Identification," *Signal Processing, Image Processing and Pattern Recognition,* vol. 6, 2013.

[3] Ibrahim Patel, Dr. Y. Srinivas Rao, "SPEECH RECOGNITION USING HMM WITH MFCC- AN ANALYSIS USING FREQUENCY SPECRAL DECOMPOSION TECHNIQUE," *Signal & Image Processing : An International Journal(SIPIJ),* vol. 1, 2010.

[4] A. BALA, "VOICE COMMAND RECOGNITION SYSTEM BASED ON MFCC AND DTW," *International Journal of Engineering Science and Technology,* vol. 2, no. 12, pp. 7335-7342, 2010.

[5] Anand Mantri, Mukesh Tiwari, Jaikaran Singh, "Development of FPGA based Human Voice Recognition System with MFCC feature," *International Journal of Engineering Trends and Technology (IJETT),* vol. 8, 2014.

[6] Wang Yutai, Li Bo, Jiang Xiaoqing, Liu Feng, Wang Lihao , " Speaker Recognition Based on Dynamic MFCC Parameters," *Image Analysis and Signal Processing IEEE,* pp. 406 - 409, 2009.

[7] Santosh K.Gaikwad, Bharti W.Gawali, Pravin Yannawar, "A Review on Speech Recognition Technique," *International Journal of Computer Applications,* vol. 10, no. 3, 2010.

[8] Mohammed Bahoura, Hassan Ezzaidi, "HARDWARE IMPLEMENTATION OF MFCC FEATURE EXTRACTION FOR RESPIRATORY SOUNDS ANALYSIS," vol. Signal Processing and their Applications, pp. 226 - 229, 2013.

[9] Jia-Ching Wang, Jhing- Fa Wang, Yu-Sheng Weng, "CHIP DESIGN OF MEL FREQUENCY CEPSTRAL COEFFICIENTS," *Acoustics, Speech, and Signal Processing,* vol. 6, pp. 3658 - 3661, 2000.

[10] Jihyuck Jo, Hoyoung Yoo, In-Cheol Park, "Energy-Efficient Floating-Point MFCC Extraction Architecture for Speech Recognition Systems," *TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS,* vol. PP, no. 99, p. 1, 2015.

[11] E. Cornu, "An ultra low power, ultra miniature voice command system based on Hidden Markov Models," in *Acoustics, Speech, and Signal Processing, IEEE*, Orlando, FL, USA, 2002.

[12] Wei HAN, Cheong-Fat CHAN,Chiu-Sing CHOY, "An Efficient MFCC Extraction Method in Speech," in *IEEE International Symposium*, Island of Kos, 2006.

[13] Ngoc-Vinh Vu, Jim Whittington, Hua Ye, John Devlin, "Implementation of The MFCC Front-end for Low-cost Speech Recognition Systems," in *IEEE International Symposium*, Paris, 2010.

[14] S.M. Ahadi, H. Sheikhzadeh, R.L. Brennan, G.H. Freeman, "AN EFFICIENT FRONT-END FOR AUTOMATIC SPEECH RECOGNITION," in *10th IEEE International Conference on Electronics, Circuits and Systems*, 2003.

[15] Phaklen EhKan, Timothy Allen, Steven F. Quigley, "FPGA Implementation for GMM-Based Speaker Identification," in *International Journal of Reconfigurable Computing*, 2011.

[16] Damak A, Krid M, Masmoudi D.S, "Neural Network Based Edge Detection with Pulse Mode Operation and Floating Point Format Precision," *Design and Technology of Integrated Systems in Nanoscale Era,* pp. 1-5, 2008.

[17] Shaifali, Ms. Sakshi, , "Comparison of IEEE-754 Standard Single Precision Floating Point Multiplier's," *International Journal of Emerging Trends in Electrical and Electronics (IJETEE),* vol. 1, pp. 900-904, March 2013.

[18] GIN-DER WU, YING LEI, "A Register Array Based Low Power FFT Processor," *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING,* pp. 981-991, 2008.

[19] Chin-Teng Lin, Yuan-Chu Yu, Lan-Da Van, "Cost-Effective Triple-Mode Reconfigurable Pipeline," *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS,* vol. 16, no. 8, pp. 1058-1071, 2008.

[20] Dongsuk Jeon, Mingoo Seok, Chaitali Chakrabarti, David Blaauw, Dennis Sylvester, "ENERGY-OPTIMIZED HIGH PERFORMANCE FFT PROCESSOR," *ICASSP,* pp. 1701-1704, 2011.

[21] Lihong Jia, Bingxin Li, Yonghong Gao, Hannu Tenhunen, "Implementation of A Low Power 128-Point FFT," in *ICSICT*, Beijing, 1998.

[22] Atin Mukherjee, Amitabha Sinha, Debesh Choudhury, "A Novel Architecture of Area Efficient FFT," *arXiv,* 25 February 2015.

[23] K. Umapathy, D. Rajaveerappa, "Low Power 128-Point Pipeline FFT Processor using Mixed Radix 4/2 for MIMO OFDM Systems," *International Journal of Soft Computing and Engineering (IJSCE),* vol. 2, no. 5, pp. 177-179, November 2012.

[24] Ediz Çetin, Richard C. S. Morling, Izzet Kale, "An Integrated 256-point Complex FFT Processor for Real-time Spectrum Analysis and Measurement," in *IEEE Proceedings of Instrumentation and MeasurementTechnology Conference*, Ottawa, Canada, May 19-21, 1997.

[25] Hoang Trang, ThanhDat Ngo, Du Nguyen, Lam Pham "Kiến Trúc Vi Mạch FFT Cơ Số Hai Với Số Điểm Linh Động Và Độ Chính Xác Cao Với Công Nghệ 130nm," in *The 2015 National Conference on Electronics, Communications and Information Technology*, Ho Chi Minh, 2015.

[26] Hoang Trang, ThanhDat Ngo, ChiTam Nguyen, Lam Pham, GiaBao Bui "A High Performance Dynamic ASIC-Based Audio Signal Feature Extraction (MFCC)," in *International Conference on Advanced Computing and Application ACOM 2016*, Cantho, 2016.