電機與控制工程學系

博士論文

# 多媒體系統晶片平台設計與應用

# Design and Application of Multimedia System-on-Chip Platform

研 究 生：鍾 仁 峯

指導教授：林 進 燈

中 華 民 國 九十五 年 三 月

# 多媒體系統晶片平台設計與應用

# Design and Application of Multimedia System-on-Chip Platform

研 究 生：鍾仁峯　　Student：Jen-Feng Chung

指導教授：林進燈　　Advisor：Chin-Teng Lin

國 立 交 通 大 學
電 機 與 控 制 工 程 學 系
博 士 論 文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical and Control Engineering

March 2006

Hsinchu, Taiwan, Republic of China

中華民國 九十五 年 三 月

# 多媒體系統晶片平台設計與應用

學生：鍾仁峯　　　　　　　　　　　　　　指導教授：林進燈 博士

國立交通大學電機與控制工程學系

## 摘　　要

多媒體訊號處理涵蓋兩大核心領域，一為影像及視訊處理，另一為語音及音效處理。它的應用適合於家庭娛樂系統中及資訊科技產業，具體的產品包括寬頻網路影音系統、數位廣播系統、多聲道視聽系統、及高音質隨身音樂媒體等。這些系統為了滿足人類聽覺的需要，即時運算是必須的。然而，多媒體處理計算的需求是由訊號處理的工作分配，若執行高的取樣頻率也就是處理較大的資料量，則需要複雜運算。以多媒體的角度而言，它必須處理多不同類型的數據，但使得處理的工作變得複雜化。本論文針對特別利用在語音與音訊處理上的計算特性，開發出一種新型的多媒體處理的架構來解決快速驗證平台的問題。

多媒體系統晶片平台的開發以聲音為導向，針對消費性 3C 產品做整合性的開發設計，它不僅適用多聲道音源輸入與不同喇叭或耳機輸出都可以增強音效的 SoC 雛型設計環境，以便對所需求的規格來確定主系統的架構。在這個規劃中我們提出了三階段的週期:系統規劃週期、系統設計週期、及系統驗證週期。多媒體系統處理核心為模組控制，模組控制就像是一個軟體的智財 (IP) 插座，可以透過相同的電子設計自動化 (EDA) 平台環境進行整合，以微處理器來分配系統資源，提供數位訊號處理器執行所需的功能。系統匯流排依循 AMBA 匯流排的時序設計，並提供 IP 標準的的介面，充分發揮系統執行的效能。用傳統微處理器和數位訊號處理器的

相互搭配架構，是未來數位電子的趨勢，亦是降低硬體成本的考量。

本論文中的多媒體系統晶片平台提出了一個 FPGA 的設計與驗證方法，系統晶片設計部分包含了系統匯流排、微處理控制器、周邊數位 I/O、16 位元空間迴響器、及 24 位元適用於音訊系統之數位訊號處理器等。可程式化微處理控制器負責系統晶片內部的流程處理和周邊 I/O 控制；24 位元數位訊號處理器因其架構與指令集是特別針對音訊系統的主要演算法做考量；16 位元空間迴響器則是一個即時 3-D 音效處理的智財 (IP)，這兩個處理器分別並連接於高速及周邊匯流排上。系統晶片驗證部分包含語音線性估測編碼的參數求取、音高位置估測、和空間迴響器等演算法的資料測試與訊號驗證。此平台搭配處理器指令與 gated-clock 的方法，可適應性的調整算數邏輯單元的使用，具有省功率運作的特性，非常適合音訊多媒體系統中可攜式與低功率要求的應用。使用 FPGA 經過驗證與測試整個系統的執行效能平均達 80MIPS，功率消耗在 90mW。此設計是跨平台的實現方法，未來可整合到任意單一矽晶片之中。

# Design and Application of Multimedia System-on-Chip Platform

Student：Jen-Feng Chung　　　　Advisor：Chin-Teng Lin

Department of Electrical and Control Engineering
National Chiao Tung University

## Abstract

Multimedia signal processing involves two important fields: one is image and video processing; the other is speech and audio processing. It is suitable to be applied into the system of home entertainment and the industry of information technology, for example, the concrete products such as a wide-band video-audio system of networks, a digital broadcast system, a multi-channel video-audio system, a high-quality walkman, etc. To satisfy the requirement of human hearing, it is necessary for real-time processing. However, the assignment of multimedia signals depends on the requirement of computational power. If a system has to process mass data, i.e., high operating frequency, it should perform complex operations. Because data is composed of different signals in the multimedia world, the work of signal processing becomes complication. In this thesis, we make use of computational characteristics of speech and audio processing and design a new architecture of multimedia processing in order to solve the problem of verification quickly.

Based on the conception of sound processing, a multimedia System-on-Chip (SoC) platform, which can integrate 3C consumer products, is designed. It is not only suitable for multi-channel sound input or output such as speakers or headsets, but also achieves the effect of virtual sound. We are in accordance with three phase cycles as specification, design, and verification for assisting the platform design. The kernel of the SoC platform is like a module control. The module control is just like as the software socket of intelligence property (IP). Thus, we can integrate with IPs via the environment of electronic design automation (EDA). In the platform, a microprocessor is as the master to assign system resources. The system bus meets the timing of AMBA and offers the standard AMBA interface to promote performance and to reduce hardware costs. The architecture of traditional microprocessor and digital signal processor (DSP) is the trend of digital circuit design in the future.

In this thesis, we present design and verification of the multimedia SoC platform. The platform design integrates the system bus, microprocessor, memory controller, peripheral I/O, 16-bit reverberator, 24-bit DSP, etc. The programmable microprocessor manages internal data flow and digital I/Os. The 24-bit DSP is specified as its architecture and instruction set for sound algorithms. The 16-bit reverberator is 3-D virtual sound IP performed in real time. The two processors are connected to the high-performance bus (AHB) and the peripheral bus (APB), respectively. The platform verification includes the speech parameters of linear predictive coding, pitch estimation, and reverberation. These algorithms are used to test data flows and to verify functionality for the proposed SoC platform. By using the gated-clock scheme, the platform has reducing power characteristics so that it can adaptively adjust the usage of parallel ALUs. Finally, under FPGA verification and testing, on average the whole performance obtains 80MIPS, and power consumption is about 90mW. Due to a cross-platform implemented scheme, it can be applies into an embedded and portable multimedia system and can also be integrated to a single silicon chip.

# 誌　　　謝

在交大電機與控制工程系攻讀博士的期間，似乎都在和時間競逐下過日子，不但修課考試要花時間，想研究內容也要花時間，實驗與寫作更花時間。不論何時何地，都和時間在競賽，這些都在磨練我的體力與耐力，雖然心力交瘁，最後還是走完這一段人生重要的過程。

首先要感謝我的指導教授—林進燈教授，因為我遇到一個優秀的，並給了我彈性與自治，他的精闢見解與專業知識適時地修正我的方向，給予寶貴的建議，彌補我研究經驗的不足，使我省下不必要的時間浪費；他的治學態度嚴謹又不失務實，不僅在學業方面的悉心指導，更在爲人處世及求學態度上給予啟蒙及悉心指導，讓我學習到許多寶貴的知識與經驗，使得本論文能順利完成。另外也要感謝口試委員們的建議與指教，使得本論文更為完整。

其次，感謝曾提供我相當多協助實驗室的學長：勝富、得正、鶴章，與學弟妹們建志、龍吉、互志、長茂、朝暉、世安、育緯、晴慧、家昇、經翔、峻谷、紹航等，以及我身邊的好朋友們，在研究過程中所給我的鼓勵、協助與支持；其實該感謝的人很多，無法一一道盡，只有將感激收藏在心裡。

最後，更要感謝我的父母親對我的教育與栽培，給予我一切鼓勵與協助，使我能安心無負擔的完成博士學業。僅將本論文獻給我的家人所有關心我的師長與朋友們，願他們共享這份榮耀。

仁峯　於交通大學電控系資訊媒體實驗室
2006.3.15

# TABLE OF CONTENTS

**APPENDIX A LASP24 INSTRUCTION SET AND EXAMPLES ....................92**

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION TO MULTIMEDIA PROCESSING

## 1.1 Introduction

Multimedia signal processing, which represents a major part of the latter category, involves the joint processing of digital information in various representations. It covers a very broad spectrum of applications:

- Audio and speech processing: audio compression, Dolby surrounding;
- Image and video processing: resolution conversion, image enhancement, image restoration, image and video compression;
- Content-based indexing and retrieval: feature extraction, pattern recognition, face detection/recognition, fusion of multi-modality;
- 2-D and 3-D graphics: volume rendering, modeling transformation, computer-assisted animation, virtual reality, etc.

As speech, audio, image, and video are playing increasingly dominant roles in multimedia information processing, content-based retrieval has a broad spectrum of application. Processing the signal using a filter circuit can remove or at least reduce the unwanted part of the signal. Increasingly nowadays, the filtering of signals to improve signal quality or to extract important information is done by digital signal processing techniques rather than by analog electronics.

## 1.2 Motivation

Multimedia systems [1] have attracted considerable media attention because of their promise to transform ordinary personal computers into entertainment centers that also function as powerful business tools. However, applications for these systems also present major design challenges to processor developers because multimedia applications such as video games, Dolby AC-3**,** and MPEG-2 video [2] have a mix of processing requirements that go beyond the capabilities of general-purpose processors. Multimedia applications [8] must not only meet stringent specifications including real-time processing, low-power dissipation, and small die size, but also be inexpensive for access to the consumer market.

For the multimedia market, processor developers must have the utmost sensitivity toward the processors effect on the final product's price. Because of this, the processor must reach a balance where hardware utilization is maximized while at the same time allowing for enough throughputs to be achievable for several applications. A number of standards have been proposed in the field of audio and video compression [3]. Communication applications, such as video telephony, are covered by the ITU-T standards H.261 and H.263 [61]-[62]. Playback of video stored on CD-ROM, TV broadcast, and video-on-demand are applications aimed by the ISO standards MPEG-1 and MPEG-2. Other important multimedia signal processing algorithms beside compression deal with content-based indexing and retrieval, speech analysis and synthesis, 2- and 3-D image animation, or scene modeling and understanding. The growing complexity of the algorithms, often associated with real-time constraints, leads to increasing computational demands. Having to deal with multiple streams of different data types further complicates the processing in a multimedia environment.

Multimedia processing requests extreme demands on computing-, transmission-, and storage-devices. Especially video consists of large data volumes, which makes it difficult to handle the data in their raw form. Therefore, compression is a key technology for multimedia applications. Thus, it is necessary for a powerful digital signal processor (DSP). Like a general-purpose microprocessor, DSP is a programmable device, with its own native instruction code. DSP is the capable of carrying out millions of floating point operations per second, and like their better-known general-purpose cousins, faster and more powerful versions are continually being introduced. It can also be embedded within complex "system-on-chip" devices, often containing both analog and digital circuitry. To perform multimedia tasks, many companies develop dedicated hardware such as hardwired solutions customized for a given application [2]-[8], or specialized processors that appear to be hardwired solutions [9]-[10]. These designs work well for their intended purposes but their inherent inflexibility forces developers to make modifications for each new application. Furthermore, dedicated hardware means that design engineers must refamiliarize themselves with a new architecture for each new system they develop.

While complexity and sophistication of multimedia algorithms continue to grow, commercial success of multimedia applications essentially relies on efficient VLSI implementation [60]. Today's standard processing devices are generally not able to fulfill the demands of multimedia processing without special adaptation. Programmable high-end general-purpose processors, as designed for the PC and workstation market, are typically weak at signal processing and moreover too expensive and power-consuming for standalone multimedia applications. Conventional digital signal processors, although optimized for processing of speech and audio signals, still lack the required high performance for video signal processing. In consequence, special architectural approaches are required to deliver sufficient multimedia processing performance at low cost.

Currently available standard processing devices are not able to fulfill the requirements of multimedia processing without special adaptation. Architectural enhancements have therefore been introduced aiming to exploit the special algorithm characteristics. Current processors, however, mainly rely on massively available data parallelism and highly predictable program flow to achieve performance gains. While this approach is feasible for algorithms dominated by block-based processing style, as encountered, e.g., in traditional video compression schemes, it will not be sufficient for emerging applications characterized by higher complexity and decreasing computational predictability. This thesis discusses innovative architectural approaches that promise a more exhaustive exploitation of parallelism and a more flexible utilization of processing resources. First, computational characteristics of current and future multimedia algorithms are analyzed. Then, architectural enhancements employed - by state-of-the-art multimedia processors with multi-core architecture [75], mainly targeting compression schemes, are shortly reviewed. The remaining part of the thesis presents reconfigurable computing, simultaneous multi-threading, and associative controlling as three promising architectural concepts able to deal with the future demands of emerging multimedia applications.

## 1.3 Objectives

Consider the components of a typical media processing system, shown in Fig. 1-1. Here, an input source presents a data stream to a processor's input interface, where it is manipulated appropriately and sent to a memory subsystem. The processor core(s) then interact with the memory subsystem in order to process the data, generating intermediate data buffers in the process. Ultimately, the final data buffer is sent to its destination via an output subsystem.

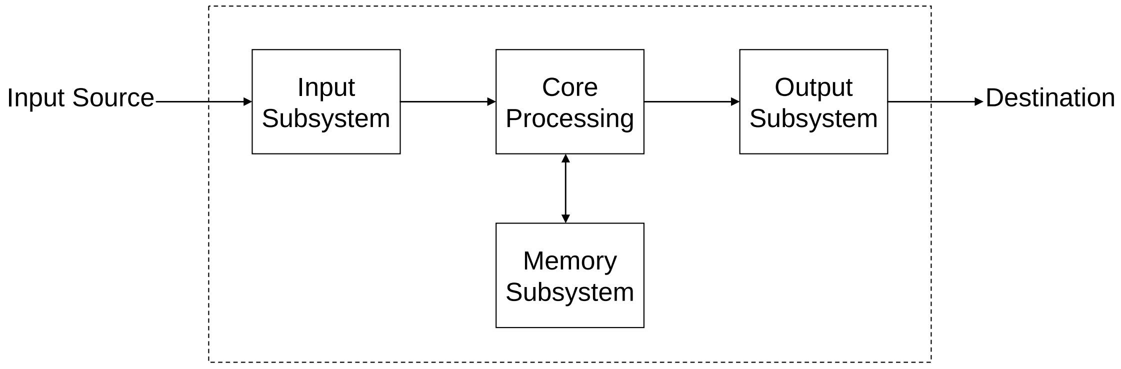


Fig. 1-1. Components of a typical media processing system.

Multimedia processing is that the actual work done by the media processor core. The input data varies widely in its bandwidth requirements. Raw audio might be measured in tens of kilobits/second (kb/s), and raw video could entail tens of megabytes per second (Mbytes/s). Then, it is clear that the media processor needs to handle different input formats in different ways

The computational requirements of multimedia processing are dominated in the first place by the signal processing tasks, requiring complex operations to be performed on large data volumes at high sample rates. Typically, real-time constraints arise from the need to satisfy human sound perception demands. Having to deal with multiple streams of different data types further complicates the processing task. State-of-the-art multimedia architectures employ a number of architectural measures in order to exploit the computational characteristics of speech and audio processing algorithms in particular. So far, the design focus has been on efficient implementation of the computation-intensive low-level parts of the algorithms-as dominating in frame-based schemes. Depending on the targeted application field, dedicated and programmable approaches can be distinguished. The design of dedicated VLSI implementations for selected multimedia processing schemes is driven by the need for inexpensive, highly integrated systems targeting the consumer market. This goal is achieved by deep adaptation of modules to special algorithms and algorithm classes. Programmable architectures, on the other hand, provide a more general platform, offering the flexibility to allow various algorithms being executed on the same hardware by only software modifications.

To achieve computing performance, the application-driven necessity to provide processors with both microprocessor and DSP functionality enforces new architectures and approaches. Simply using two cores—a microprocessor and a DSP core—is multitasking-effective as resources are often doubled. The 24-bit architecture presented here provides general purpose micro-processing as well as DSP functionality through a single-core and a unified architecture, respectively. The optimal solution is an application-oriented processor core [69], [70], having a lower cost than a general-purpose DSP. The parallel architecture of DSP can efficiently execute vector and matrix operations without extra overhead. In order to implement an application-driven DSP, we use a methodology for hardware/software (HW/SW) co-verification [71] and optimize the processor architecture and instruction sets. High flexibility in use, small area on silicon, high data throughput, and fast portability to a wide range of technologies are our main targets in the core development.

The modern embedded system has moved toward the target of system integration and implementation. Reuse [50] is done at the chip level called intellectual property (IP) core which represents the functions of specification domains like multimedia applications. These modules are integrated into System-on-Chip (SoC) [72] which is a typical architecture. We investigate architectural techniques to facilitate analysis and integration for heterogeneous and general/complex SoC applications in this thesis. We use a microcontroller and digital signal processor application to validate the embedded platform prototype. The concept of platform refers to a family of architectures satisfying ARM defined constraints [54], and allowing customizations and substantial re-use of hardware and software modules. We developed a base architecture with customization or parameterization options to speed-up derivative implementations while reducing the HW/SW overhead. The definition of the SoC platform IP is the result of a trade-off process involving reusability, overall SoC integration effort [67], performance [68] and power optimizations [77]. Our focus is on efficiency of the hardware and software resources in the context of a self-adapting architecture with autonomic features. The motivation for developing such IP is to facilitate integration of SoCs. Because the platform is meant to be easily customized, it is essential to meet stated resource-efficiency goals.

The SoC integrated platform provides multi-function system backbone for various multimedia applications. The new proposed SoC integrated platform which combines microprocessor, digital signal processor (DSP), memory, and other functional modules such as GPIO (General-Purpose Input/Output), $I^2S$ (Inter-IC Sound), and communication (UART) into a single IC is popular recently. To verify these functions of the proposed platform for audio and speech processing, the FPGA (Field Programmable Gate Array) rapid prototype approach will be used. FPGA were primarily used for prototyping and lower volume applications in years and custom ASICs were used for high volume, cost-sensitive designs. In the thesis, we will describe the proposed platform, IP reuse design experiment based on a methodology in [66].

The contribution of this thesis is to propose a multimedia SoC platform, which can integrate 3C consumer products, for sound processing. The platform can solve low-cost consideration and construct quickly the components of multimedia signal processing in the standard bus. The platform using two cores, a microprocessor and a DSP core, can promote computational performance. Especially DSP, it is designed as the sound signal processor for the consideration of architecture and instruction set, and has low-power characteristics with gated-clock technology [37]. Due to their high capacitive load which makes them a major contributing factor to the overall power consumption of the SoC device, we take advantage of instruction or control types to decide which component needs to be disabled. In addition, bus encoding techniques [77] can reduce the power consumption on a bus by mapping the information conveyed on the bus to a form which has less transition activity than the original. This is to reduce the consumption of the electric current. Due to cross-platform design which is not limited by any synthesis tool or FPGA, this method can be easily verified and fabricated as ASIC.

## 1.4 Organization of the Thesis

In this thesis, the rest of the dissertation is organized as follows. Chapter 2 describes in multimedia the application of sound signal processing including artificial reverberation and speech compression. These three sound processing will be applied to the SoC platform. According to the reverberation principle, we design a real-time reverberator with the pseudo-random coefficient method. Afterward, Chapter 3 would implement an application-driven digital signal processor. We consider for special addressing modes, matrix and vector processing, power optimization, and the architecture of reverberator. Continuously, the experimental results and comparison would be illustrated in Chapter 4. In order to be able to apply into the SoC platform for sound signal processing, Chapter 5 constructs a multimedia integrated platform. The platform is controlled by a built-in microprocessor, and it can capture and play sound through the standard inter-IC sound interface. Hence, the multimedia platform can call a programmable SoC platform. Finally, conclusions and future works are made the last chapter.

# CHAPTER 2

# MULTIMEDIA IN SOUND PROCESSING

## 2.1 Introduction

Sound processing is one of the many applications of digital signal processing. Three-dimensional (3-D) sound is becoming increasingly important in scientific, commercial, and entertainment systems [8] for human life. It can greatly enhance auditory interfaces to computers, improve the sense of presence for virtual reality simulations, and add excitement to computer games. Recent extensions of physical and behavioral studies have revealed that the external ear plays an important role in spatial hearing. Due to the rapid growth in computational power, many new virtual auditory systems could be implemented in real time. In the Section, we introduce two techniques of 3-D sound processing as artificial reverberation and one basic speech processing as linear prediction coding (LPC) and pitch estimation (PE), respectively.

## 2.2 Artificial Reverberation

Today, a multichannel [9] playback system has been frequently used in cinema or home video. In this thesis, a stereo-channel multiband room effect simulator [81] with friendly control interface is presented. In order to obtain different music quality, we design a new room simulator to be suitable for the multichannel surround sound system [10].

The impulse response for room simulation is the result of the many reflections of a sound that occur in a room. The response consists of direct sound, early reflection, and fused reflection. In 1961, the first room effect algorithm was proposed by Schroeder [11].

Then Schroeder's algorithm was extended by Moorer [12] in 1978. The room effect introduces a spatial dimension to a piece of recorded sound, which means that it can be used to model a specific acoustic environment in which to affect a dry unaltered signal. Long reverberation times provide the feeling of a large hall, and short reverberation times (RT) give the impression of smaller rooms. We refer to Moorer's reverberator using FIR and IIR filters to design artificial reverberation called a reverberator. The impulse response for an acoustic room is depicted in Fig. 2-1. This response includes direct sound, early reflections, and late reverberation. The main contribution of this thesis should be a specification of the requirements made on the reverberation algorithm, which will be preparing for real-time processing and multichannel outputs. The reverberation algorithm is based on an exponentially-decaying pseudo-random FIR filter [13] to represent the early reflections segment, with a feedback delay path to create the dense reverberant field. In addition, an equalizer offers the capability of both compensating for defects and fine tuning the system. With an equalizer, certain frequency ranges can be either increased or cut. We also design a 10-band equalizer as like Winamp[2] to control how finely the frequency pattern can be amplified or attenuated and setup several selective modes for selection. Finally, the output of the room simulator can be connected to modified 5.1-channel Dolby surround decoder with [14].

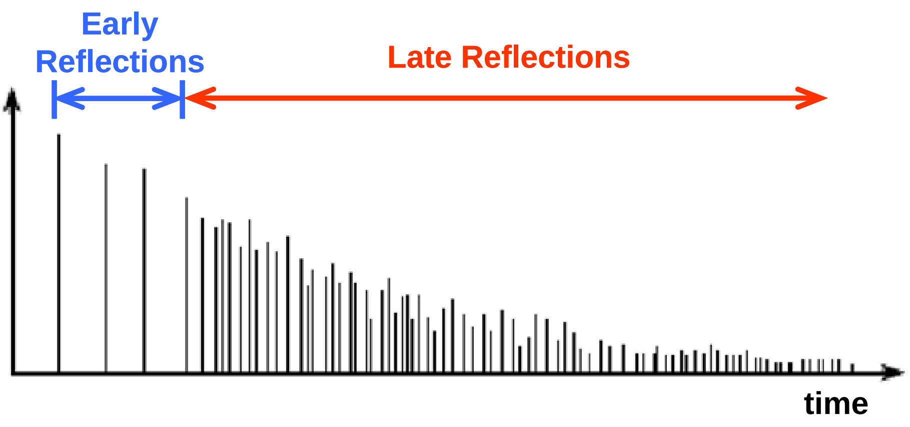


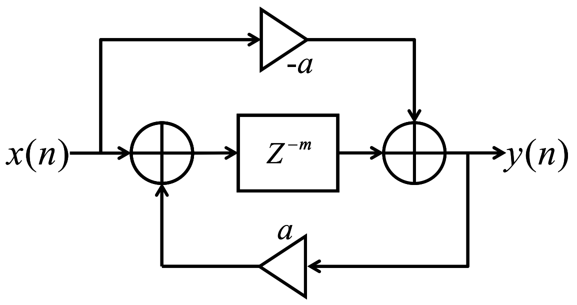
Fig. 2-1. Ideal impulse response of an acoustic room.

Reverberation [15] is probably one of the most heavily used effects in music. Reverberation effects can be achieved by using any combination of filter techniques. The FIR filter, comb filter, and all-pass filter are the basic structures that have been combined in different ways in an attempt to imitate the effects of various rooms.

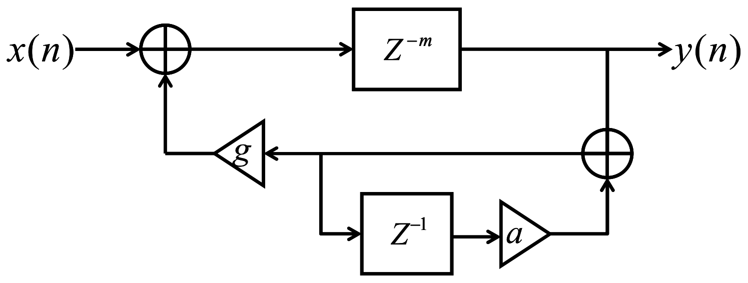
### 2.2.1 Filters

Filtering techniques are used to perform convolution with input sound sources. A FIR filter is used to model the segment of early reflection. This is because each reflected signal could be distinguished by human ears in this segment. The parallel comb filters and cascade all-pass filters are added to generate its late reverberation segment. In order to increase the echo density, the output of the parallel comb filters is fed into one or more all-pass filters (Fig. 2-2(a)) in series. Each all-pass filter has a multiplicative effect on the number of echoes, but prevents coloration due to the all-pass filter's flat frequency response.

In general, a high order FIR filter is considered to model the early reflection segment, but it would take too much execution time for computation. To improve this problem, a simple delayed feedback loop around the FIR early segment is used to reduce the FIR order. The order of the FIR filter is decided by the first comb filter delay and the reverberation length in our system. However, room impulse responses consist of very dense series of echoes that cannot be practically realized using this architecture. Since the eigenfrequencies of rooms have a rapid decay for high frequencies, and a frequency-dependent reverberation time can be implemented with a low-pass filter. Moorer suggested a modified comb filter with a low-pass filter (Fig. 2-2(b)) in feedback loop to take frequency-dependent decay into consideration to solve this problem.



(a)



(b)

Fig. 2-2. (a)All-pass filter and (b) modified comb filter, where $M$, $g$, $a$ represent the delay length, the gain factor, and the coefficient, respectively.

### 2.2.2 FIR Early Reflection

The impulse response of sound consists of direct sound, early reflections shown as Fig. 2 and exponentially decaying late reverberation (with an IIR filter to reduce computational complexity).

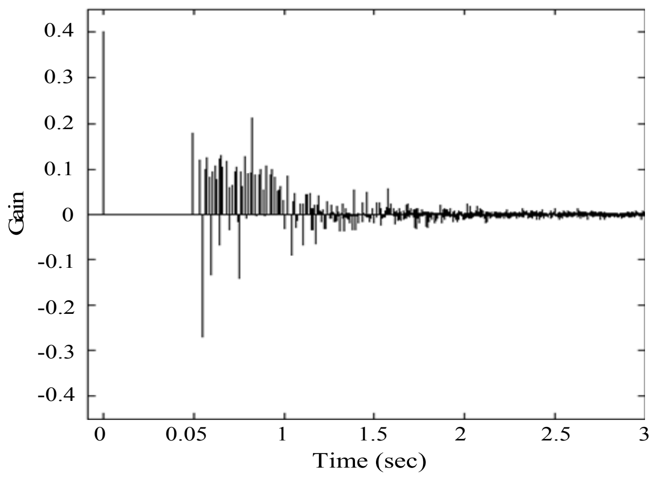


Fig. 2-3. Impulse response with early reflections.

The early reflections often derived from a room model, e.g., as reflections caused by an image source. Reflections during about 20~80ms after sound is triggered are heard together with the direct sound as one single auditory event. An FIR filter is used to generate these early reflections. Since the early reflections are relatively sparse and span a relatively short time, they can be implemented using tapped delay lines (Fig. 2-4). This idea was apparently first suggested by Schroeder in 1970 [11] and evidently first implemented by Moorer [12]. A key parameter in determining the quality of the reverberation is the echo density. In the case, to increase echo density, the FIR order will be large. Of cause, this causes long computing time. Hence, we can increase non-zero values with pseudo-random coefficients and select suitable for memory spaces to reduce the FIR order but not affect sound qualities.

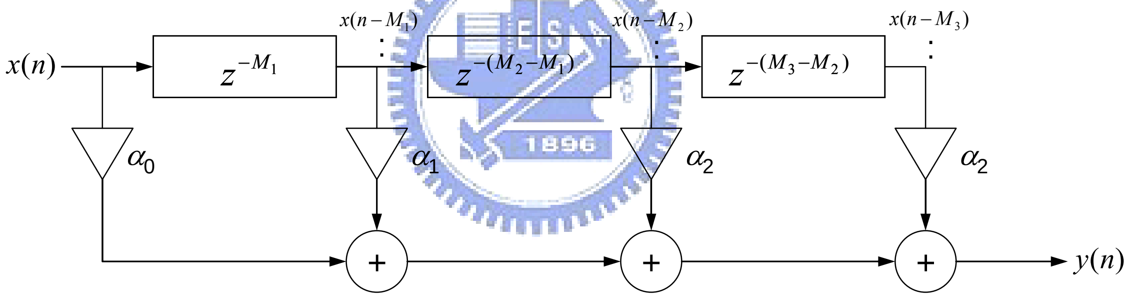


Fig. 2-4. Impulse response with early reflections.

## 2.2.3 Reverberator

Different audio effects can be performed by designing and implementing suitable filters. The proposed reverberator shown in Fig. 2-5 is composed of a FIR filter with pseudo-random coefficients (Fig. 2-6(a)), 10 parallel comb filters, 4 cascade all-pass filters, and a pair of late low-pass filters. The FIR filter models the segment of early reflection. The parallel comb filters and cascaded all-pass filters model the segment of late reverberation, and late low-pass filters are to produce the feeling of the distance from sound source. Note that the delay length of comb filters must be carefully chosen to avoid the coloring phenomenon shown as Fig. 2-6(b) and (c), respectively.

The input of the room simulation is the mono signal $x_R(n)$ and $x_L(n)$ respectively. These two mono input signals are added to the left and the right room signals after going through a delay line *Del2*, and then go through another delay line (FIR filter). The total sum of the early reflections made by FIR filter then goes to parallel circuit of comb filters and cascade all-pass filters which implements subsequent reverberation. The generated reverberant signals $e_L(n)$ and $e_R(n)$ are added to the direct signals ($x_L(n)$ and $x_R(n)$) and early reflections ($ER_L(n)$ and $ER_R(n)$).

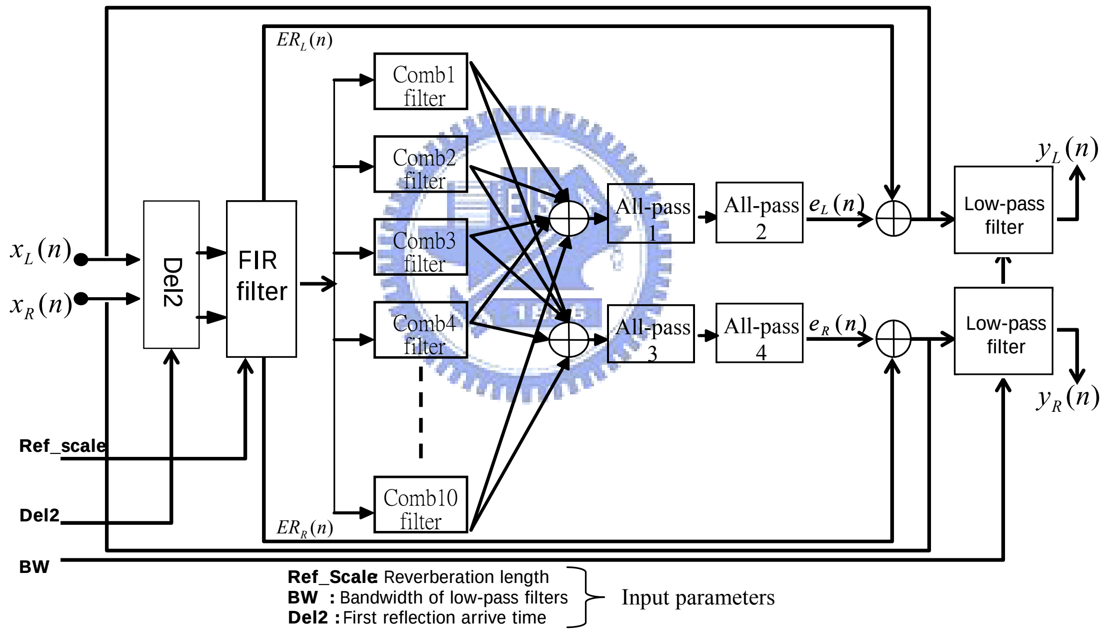
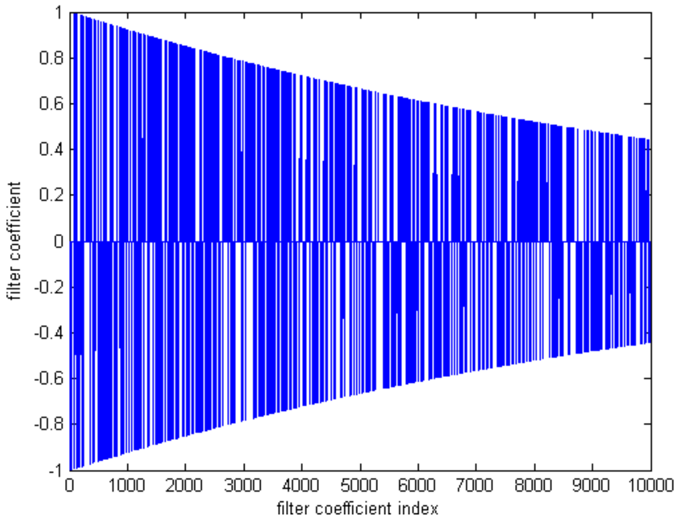
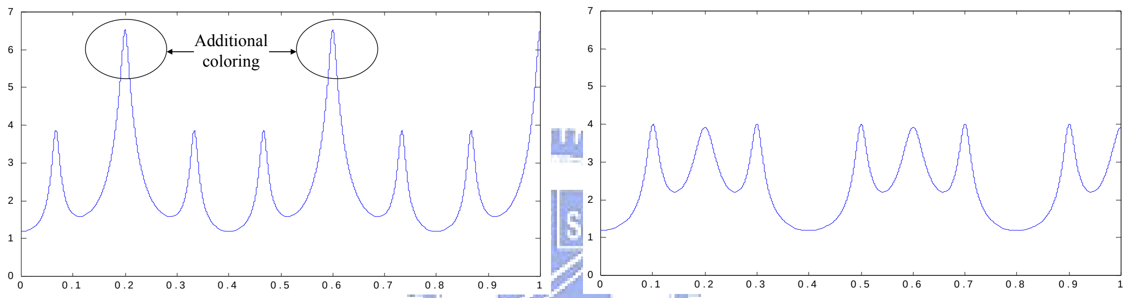


Fig. 2-5. The arcticture of reverberator [24].

In order to obtain a high quality spatial impression, it is not necessary to correlate the room signals $e_L(n)$+$ER_L(n)$ and $e_R(n)$+$ER_R(n)$. In Fig. 2-5, these input parameters are *Ref_Scale*, *Del2*, and *BW*. *Ref_Scale* denotes the reverberation length. It will effect the order of FIR filter and the delay lengths of comb filters and all-pass filters, *Del2* denotes the first reflection arrival time, and *BW* denotes the bandwidth of the late low-pass filters.



(a)



(b) (c)

Fig. 2-6. (a) FIR modeling with exponentially-decaying pseudo-random coefficients; (b) Additional coloring phenomenon of a comb filter; and (c) Coloring reduction.

The identification process is based on the knowledge of the input $x(n)$ and the output $y(n)$ of the 1024-tap FIR as

$$y(n) = \sum_{i=0}^{1023} h(i) \cdot x(n-i), \tag{2.1}$$

where $n$ is the number of infinite sequences, and $h(i)$ represents pseudo-random coefficients. The FIR requires 1024 MAC operations. We use two circular buffers for input and output sequences to perform 1024-tap operations. According to the function of FIR early reflections, Eq. (2.1) can be changed as

$$y(j) = y(j-1) - \left[h(2i)\cdot x(delay)\right],\ \ 0 \le i \le 1024, \qquad (2.2)$$

$$delay = \begin{cases} block-(2i-j), & \text{if } j<2i \\ j-2i, & \text{otherwise} \end{cases}, \qquad (2.3)$$

where *block* represents the length of a spatial circular buffer, and *j* is equal to *block*. The spatial buffer is larger than the FIR order. Due to random coefficients, sound reflections can be represented by impulse responses from any direction in a room. The pseudo-random coefficient is generated by

$$h(i){=}floor\,((1+p)\cdot rand\,(1,Le){-}p/2), \qquad (2.4)$$

where the parameter *p* is equal to 2×(the density of non-zero filter taps)/(sampling rate (Hz)) and *Le*=ceil(sampling rate × reverb time). These two signals (*p* and *Le*) indicate the probability of non-zero filter taps and the length of FIR filter (in samples), respectively. We use two different densities per second of non-zero filter taps, 1.92s reverb time, and 44,100Hz sampling rate to generate the FIR coefficients shown in Fig. 2-7. The generated sound quality and effect using coefficients in Fig. 2-7(right) are better than those of Fig. 2-7(left). These coefficients are only 0, 1, and -1. For a multi-tap FIR implementation, it can reduce MAC operations and even not need multiplication. To avoid too large accumulated values, a shifting operation is added into Eq. (2.2) after multiplication as

$$y(j) = y(j-1) - \left[h(2i)\cdot x(delay)\right]/32,\quad 0 \le i \le 1024\,. \qquad (2.5)$$

In order to simply design complexity for implementing reverberation, hence, we reduce FIR orders and the numbers of comb filters. The FIR filter uses pseudo-random coefficients sequence of 1's, 0's, and -1's, and the tap is less than 1024 orders. We recommend that Schroeder proposed the architecture of four-parallel comb filters and two-cascade all-pass filters as the implementing method on hardware.

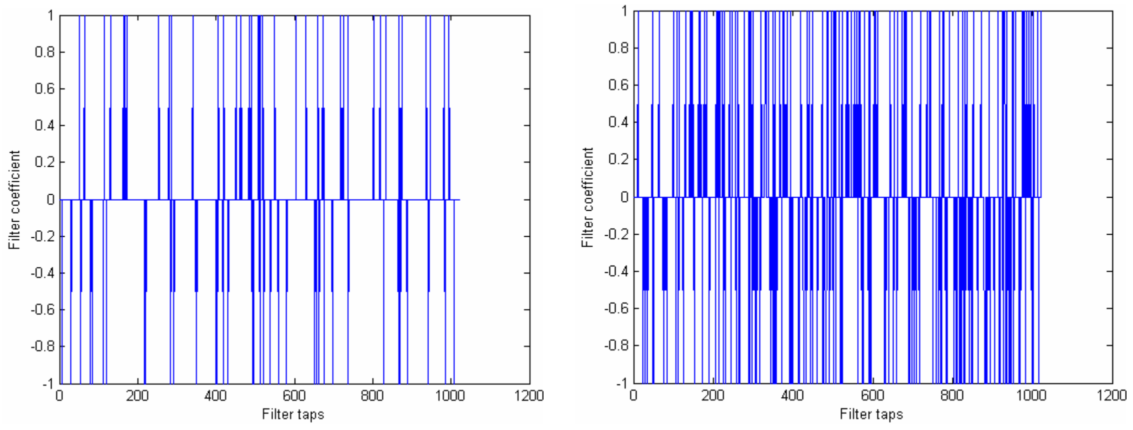


Fig. 2-7. The un-weighted pseudo-random FIR sequence of 1's, 0's, and -1's with 4,000 (left) and 14,400 (right) densities per second of non-zero filter taps, respectively.

## 2.3 Speech Processing

One of the powerful speech analyses is Linear Prediction Coding, or LPC analysis as it is commonly referred. In the LPC analysis, the short-term correlation between speech samples (formants) is modeled and removed by a very efficient short-order filter. Another equally powerful and related method is pitch estimation (PE). The long-term correlation of speech samples are analyzed in PE. A vocal tract model, as described in [30] can be estimated using LPC analysis and approximated by an all-pole filter. We shall describe briefly the ways for finding LPC coefficients and pitch information as follows.

### 2.3.1 Linear Predictive Coding

In order to model the time-varying nature of the speech signal whilst staying within the constraint of our LPC analysis, i.e., stationary signal, it is necessary to limit our analysis to short-time blocks of speech. This is achieved by summations over finite limits, i.e.,

$$\begin{aligned}\phi_n(i,j) &= E\{s(n-i)s(n-j)\} \\ &= \sum_m S_n(m-i)S_n(m-j), \qquad (2.6)\end{aligned}$$

where $E$ is the mean squared error, the waveform segment, $Sn(m)$, is assumed to be zero outside the interval $0 \le m \le N-1$, and $N$ is the length of the sample sequence. We use the auto-correlation method to approach the interpretation of Eq. (2.6). Since, for $N \le m \le N+p$, we are trying to predict zero sample values (which are not actually zero), the prediction error for these samples will not be zero. Assuming that we are interested in the future prediction performance, the limits for Eq. (2.6) can then be expressed as

$$\phi_n(i,j) = \sum_{m=0}^{M+p-1} S_n(m-i)S_n(m-j), \qquad (2.7)$$

or

$$\phi_n(i,j) = \sum_{m=0}^{N-1-(i-j)} S_n(m)S_n(m+i-j), \qquad (2.8)$$

for $1 \le i \le p$ and $0 \le j \le p$. Equation (2.8) can be reduced to the short-time auto-correlation function, as given by

$$\phi_n(i,j) = R_n(|i-j|), \text{ for } i = 1,\cdots,p, \quad j = 0,\cdots,p, \qquad (2.9)$$

where $R_n(j) = \sum_{m=0}^{N-1-j} S_n(m)S_n(m+j)$. Using the auto-correlation method, $\sum_{j=1}^{p} \alpha_j \phi_n(i,j)$ can therefore be expressed as

$$\sum_{j=1}^{p} \alpha_j R_n(|i-j|) = R_n(i), \ 1 \le i \le p, \qquad (2.10)$$

or in normal matrix form given by

$$\begin{bmatrix} R_n(0) & R_n(1) & \cdots & R_n(p-1) \\ R_n(1) & \cdots & \cdots & R_n(p-2) \\ \vdots & \vdots & \vdots & \vdots \\ R_n(p-1) & \cdots & \cdots & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix}, \tag{2.11}$$

where $\alpha_j$ represents the estimate parameters on $a_j$ , $j$ = 1,...,$p$. The above matrix has the property that it is symmetrical and all elements along a given diagonal are equal, i.e., it is a Toeplitz matrix [18]. Equation (2.10) can be solved by the simple inversion of the $p \times p$ matrix; however this is not usually performed since computational errors such as finite precision tend to accumulate. By exploiting the Toeplitz characteristic, however, very efficient recursive procedures have been devised. The Levinson-Durbin's algorithm [33] is used to compute the prediction coefficients for LPC analysis of the auto-correlation sequence of samples. It provides solutions to the linear equations through recursive procedure that exploits the symmetry property.

### 2.3.2 Pitch Estimation

Accurate estimation of the pitch period or the lag $\tau$ in the speech coding is very important. The direct distance measurement is the most popular criterion, examining the similarity between two waveforms which can be expressed as

$$E(\tau) = \frac{1}{N}\sum_{n=0}^{N-1}\left[s(n) - \beta s(n+\tau)\right]^2 , \tag{2.12}$$

where $\beta$ is a scaling factor, or the pitch gain, controlling the changes in signal level. Under the assumption that the signal is stationary, the error criterion of Eq. (2.12) can be written as

$$E(\tau) = \left[R(0) - R(\tau)\right], \textbf{ where } R(\tau) = \sum_{n=0}^{N-1} s(n)s(n+\tau). \tag{2.13}$$

Speech in the long term is a non-stationary signal, and the direct similarity criterion may exhibit large errors, implying fewer similarities in position where the shift is equal to the real pitch period. Equation (2.13) is the direct auto-correlation which indicates more similarities in triple pitch period as the amplitude is increasing. The normalized similarity criterion in Eq. (2.12) is derived under the consideration of such a non-stationary process. Setting $\partial E(\tau, \beta)/\partial\beta = 0$ of Eq. (2.12), the optimum normalization coefficient (pitch gain) can be calculated using

$$\beta = \frac{\sum_{n=0}^{N-1} s(n)s(n+\tau)}{\sum_{n=0}^{N-1} s^2(n+\tau)}. \tag{2.14}$$

By substituting the optimum gain back into the error function of Eq. (2.12), the pitch can be estimated by minimizing

$$E(\tau,\beta) = \sum_{n=0}^{N-1} s^2(n) - \frac{\left[\sum_{n=0}^{N-1} s(n)s(n+\tau)\right]^2}{\sum_{n=0}^{N-1} s^2(n+\tau)}. \tag{2.15}$$

This is equivalent to maximizing the square of the normalized auto-correlation function given by

$$R_n^2(\tau) = \frac{\left[\sum_{n=0}^{N-1} s(n)s(n+\tau)\right]^2}{\sum_{n=0}^{N-1} s^2(n+\tau)}. \tag{2.16}$$

The pitch period can be determined from Eq. (2.16). The normalized auto-correlation method shows a much better performance than direct (un-normalized) auto-correlation method.

# CHAPTER 3

# DESIGN OF APPLICATION-DRIVEN DIGITAL SIGNAL PROCESSOR

## 3.1 Introduction

The proposed application-driven digital signal processor (DSP) [85]-[87], called LASP24 (Low-cost Application-driven Speech Processor, 24-bit data width), is constructed as a reduced instruction set computer (RISC) architecture with vector and matrix operations and power optimization. An effective verification is used to subserve the hardware design and to decrease debugging time during the development of hardware and software. High performance is achieved by vector and matrix operations that are not usually supported by general-purpose DSPs. The parallel architecture of LASP24 can quickly execute vector and matrix operations without extra overhead. High flexibility in use, small area on silicon, high data throughput, and fast portability to a wide range of technologies are our main targets in the core development.

The development of the digital signal processor shown Fig. 3-1 is to meet the system demands that are based on sophisticated arithmetic algorithms and that emphasize on both hardware and software solutions. The verified tools offer the opportunity to trade off between software (for flexibility) and hardware (for performance and power optimization). The development flow consists of two parts: hardware implementation and software development. Software includes two development tools: the assembler and the emulator. The assembler can translate assembly language into binary codes (or called machine codes). Simultaneously, the initial ROM file is generated for the processor emulator and the HDL simulator. The emulator can emulate the computations of the processor hardware and verify the precision of different floating-point formats such as 32- or 24-bit. In hardware design, using the hardware description language (HDL) implements the processor and improves performance and power dissipation for speech/audio algorithms. The processor can be regarded as an embedded DSP processor.

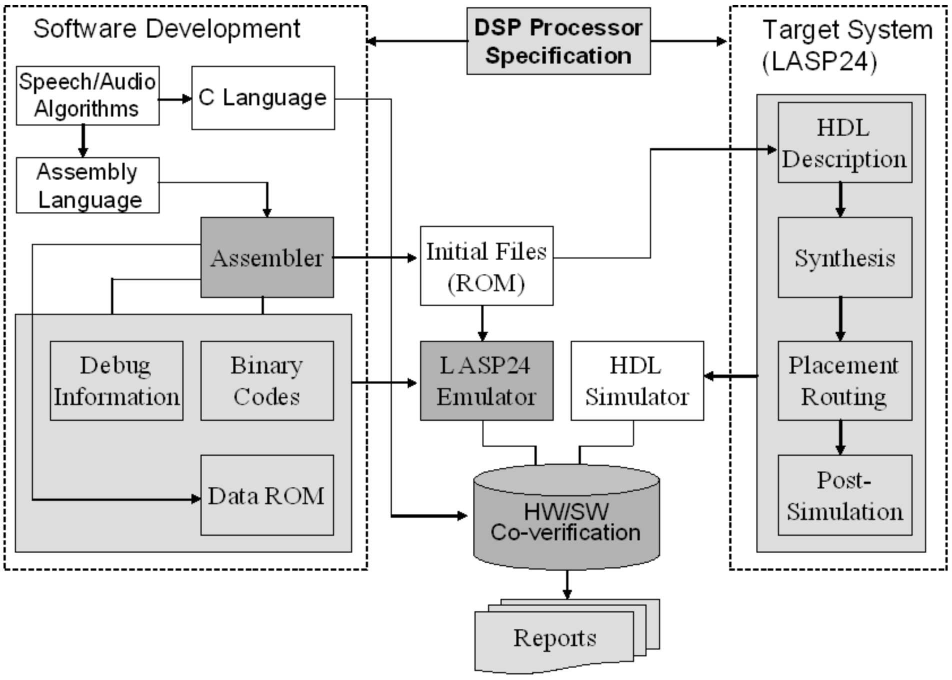


Fig. 3-1. Hardware/Software development flow for LASP24.

## 3.2 Micro-architecture

The RISC-type [31] processor has traditionally enhanced performance by the reduced instruction set to maximize the throughput, and most of them access rather a large program memory at every clock cycle to fetch each instruction. Thus, application-driven design can reduce complexity and is greatly enhanced at performance. For an embedded DSP, it is necessary that the architecture should support effective data communication between memory system and execution units, low-overhead loop control, and accumulator-based instruction set architecture.

An efficient method of data representation and a hardware implementation is proposed to utilize a smaller program memory, while maintaining other merits of the RISC, such as simple decoding, fixed instruction size, and high performance. LASP24 is a 24-bit DSP processor with a floating-point unit and is ease of use. The DSP processor has the architecture of a 24-bit single-instruction/multiple-data (SIMD) instruction set with five addressing modes, and a five-level pipeline executing engine, which is Instruction Fetch (IF), Instruction Decode (ID), Execution (EX1, EX2), and Write Back (WB). It is important to perform parallel multiplication and arithmetic operations in a single cycle. This allows instruction execution to overlap. Thus, the effective execution time for most instructions is one cycle. Some key features of LASP24 are listed below:

- 24-bit fixed length instructions which support 2- and/or 3-operand.
- Five pipeline stages to improve throughput.
- Five addressing modes and one control mode. Up to the support of 32 instructions.
- Two bank internal memories for use of vector addressing.
- 24 address stacks and 70 data stacks.
- Block repeat capability.
- Zero-overhead loops with a single-cycle branch.
- Branch conflict with hardware detection and solution.
- Power saving consideration.

Floating-point operations provide fast, accurate, and precise computations. The 24-bit floating-point format is compatible with IEEE-754 standard [32]. Specifically, LASP24 facilitates floating-point operations at high speed for speech/audio signal processing, which offers addition, subtraction, multiplication, and simulated division.

The block diagram of proposed LASP24 is shown in Fig. 3-2. LASP24 is functionally partitioned into the following major blocks: a computation unit, which indicates ALU, multiplier, and accumulators, a program control unit, an external bus control dictating LASP24 external buses, a vector address generator computing the addresses which are used in vector operations. The program control unit performs instruction fetch, decoding, exception handling, and wait state supports. The PCU generators the next address to the program memory and controls hardware loops.

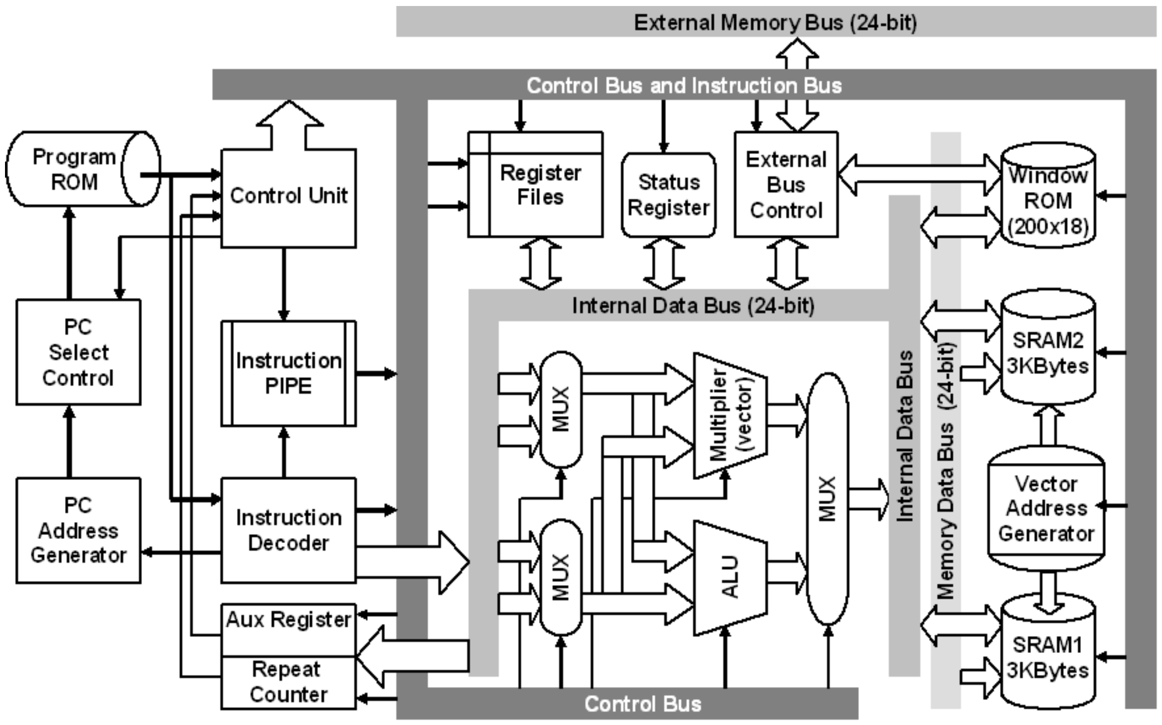
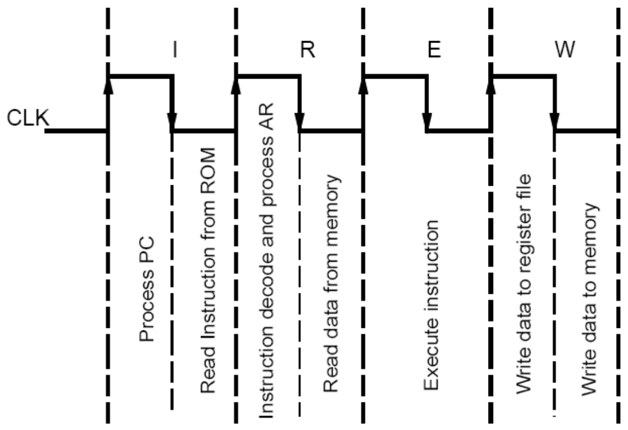


Fig. 3-2. The block diagram of the proposed digital signal processor.

LASP24 includes four register groups. The eight general-purpose registers (Register File) are capable of storing and supporting operations on 24-bit floating-point numbers. The two 8-bit auxiliary registers can be accessed by the processor and modified by the auxiliary register arithmetic unit. The primary function of the auxiliary registers is the generation of 8-bit addresses. They can also be used as loop counters or as matrix point register. The status registers contain information relating to the state of ALU and parallel multiplication. When the status registers is loaded, LASP24 sends out a busy signal, and executes the selected function. The two 8-bit repeat counters which used to specify the number of times are to be repeated when performing a block repeat.

LASP24 uses a five-stage pipelined structure, and the pipelined operation is shown in Fig. 3-3. The Instruction Fetch (I) stage fetches the instruction words from instruction ROM and updates the program counter (PC). The Read and Decode (R) stage decodes the instruction word and performs address generation. Also, it controls the modification of the AR0 and AR1 registers in the matrix and vector addressing modes, and if required, reads the operands from memory or general registers. The Execution (E) stage is divided into two stages and performs the necessary operation, such as floating-point addition, subtraction and multiplication. The Write Back (W) stage, if required, writes results to the register file and memory.


I
R
E
W
CLK
Process PC
Read Instruction from ROM
Instruction decode and process AR
Read data from memory
Execute instruction
Write data to register file
Write data to memory
Fig. 3-3. Pipelining operations.

The pipelined control exists the problems of conflicts (or hazards). The conflicts can be grouped as branch, memory, and register conflicts. The branch and register conflicts are described in [58], and the concept of its solution to these conflicts is applied to our design. The register conflicts arise when an instruction depends on the results of a previous instruction in a way that is caused by the overlapping of instructions in the pipeline. Using the forwarding way can solve the problem of register conflicts. The branch conflicts arise from the pipelining of branches and other instructions that change the PC. The condition of a branch conflict is shown in Fig. 3-4. The (*i*+2)th instruction will return to the *j*th instruction, but the pipeline register has fetched the (*i*+2)th instruction. For the branch taken, the (*i*+2)th instruction is not used and replaced by the "NOP" instruction. This

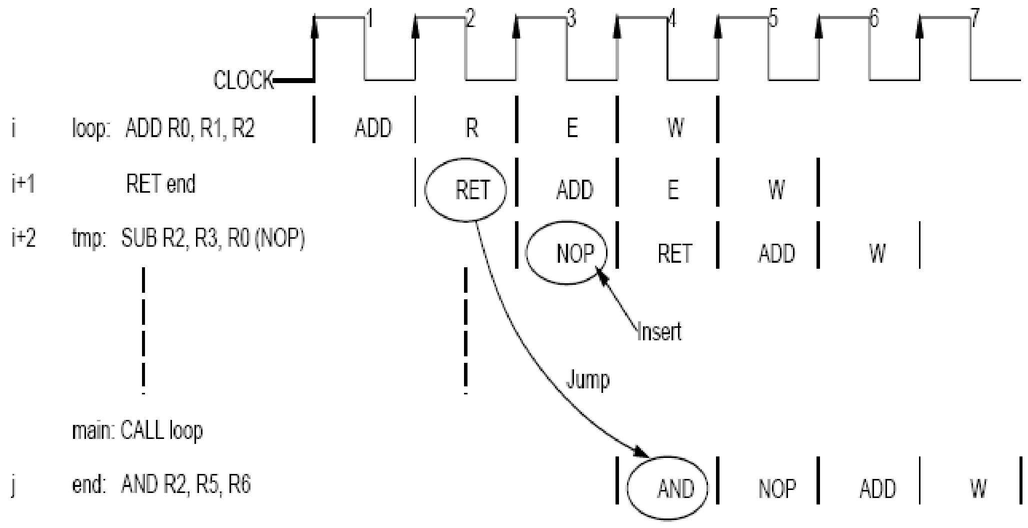


Fig. 3-4. Branch operations.

change solves the branch conflict, but the pipeline causes overhead. Hence, we modify the way of branch conflicts in Fig. 3-4 to avoid NOP operation and to reduce time overhead. The branch conflict in LASP24 does not exist because the PC is changed in the I stage and the R stage, not in the E stage. Before the next cycle, the indicated branch instruction will be ready in the I stage. That means the program control is free of branch conflicts, and there is zero overhead for a branch instruction. The memory conflicts arise from resource conflicts when the hardware cannot support all possible combinations of instructions in the simultaneous overlapping. As shown in Fig. 3-5, this type of conflicts may happen. The *i*th instruction does not yet write R1 to the location of RAM0[r], but the (*i*+1)th instruction reads data from the location of RAM0[r]. At this time, a memory hazard occurs in the pipeline. The (*i*+2)th instruction is reading data from the locations of RAM0[r] and RAM1[r], but the *i*th instruction is writing R1 to RAM0[r]. This is seriously conflicts for memory data buses. The solution is to assign the priority of writing memory higher than that of reading. The above condition similarly occurs between two internal RAMs and one external bus. In the other way, the software codes can also avoid this type of conflicts.

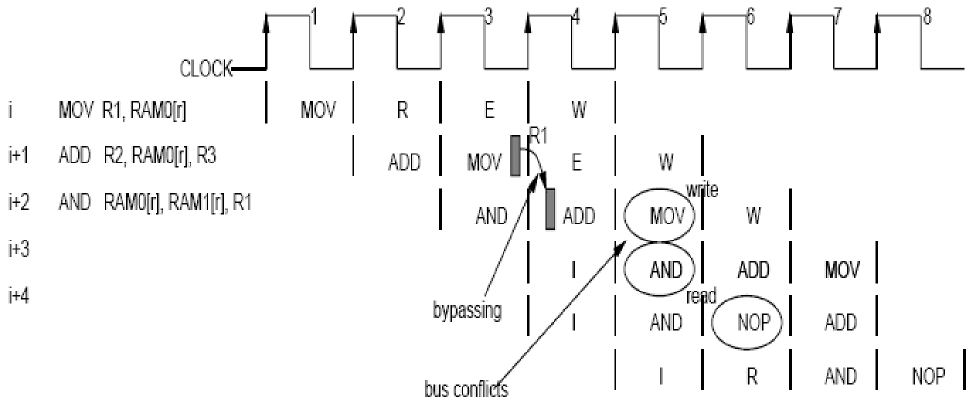


Fig. 3-5. Memory accessing operations.

## 3.3 Instruction Set

The processor instruction sets have been designed with two goals in mind: 1) to make maximum use of the processor's underlying hardware, thus increasing efficiency and 2) to minimize the amount of memory space required to store DSP programs, since DSP applications are often quite cost-sensitive and the cost of memory contributes substantially to overall chip and/or system cost. To accomplish these two goals, it is necessary to reduce the number of bits required to encode instructions and to offer fewer registers and addressing modes than other types of processors. Thus, the architecture of LASP24 is defined as a fixed instruction length at 24 bits. A 24-bit instruction uses five bits each for addressing 8 general-purpose registers. LASP24 instruction set includes five addressing modes and is classified into three groups as data transfer, arithmetic, and control instructions. The total of defined instructions is about twenty-five (see Appendix A in details). Some representative instructions are listed as follows.

| Instruction | Descriptions and Examples |
|---|---|
| | **Load and Store Instructions** |
| MOV | Load, store and move data<br>1. General data moves<br>EX: MOV RAM0[address], R0; R0=RAM0[address]<br>2. Data moves for the matrix addressing mode<br>EX: MOV RAM1[AR1L+1, AR0L], R3; R3=RAM1[AR1L+1, AR0L],<br>where AR0L and AR1L are defined as AR0[3:0] and AR1[3:0]). |
| LD | Load fixed values as follows:<br>0.0, 0.75, 1.0, and<br>2.0 - A (the floating-point value from 2.0 leaves operand A) |
| | **Arithmetic Instructions** |
| ADD | Add floating-point values<br>EX: ADD R0,R1,R2; R2=R1+R0 |
| SUB | Subtract floating-point values<br>EX: SUB R0,R1,R2; R2=R1-R0 |
| MPY | 1. General multiplication<br>EX: MPY R0,R1,R2; R2=R1×R0<br>2. Matrix multiplication<br>EX: MPY R3,RAM0[1110,AR0L-AR1L];<br>R3=RAM0[1110,AR0L-AR1L]×R3 |
| VMPY | Vector multiplication<br>EX: VMPY EXT[j],WIN[j],RAM0[j],RAM1[j];<br>{RAM0[j],RAM1[j]}=EXT[j]×WIN[j] |
| MAC | Multiplication-and-accumulation<br>EX: MAC RAM0[j], RAM1[j], R3; R3=RAM0[j]×RAM1[j]+ACC,<br>where ACC is an accumulator. |
| DIVEXP | Re-scale after division<br>EX: DIVEXP R0,R3; R3=DIVEXP(R0) |
| NORM | Normalize floating-point value<br>EX: NORM R0,R1; R1=Norm(R0) |
| | **Program Control Instructions** |
| NOP | No operation |
| LDC | Load AR0 and AR1 value<br>EX: LDC AR0,#14; load 14 to AR0 |
| RPB | Begin repeat block<br>EX: RPB RC0, 255; for (r=0; r<=254; r++) |
| RETB | Return repeat block of instruction<br>EX: RETB AR0, label; if AR0=RC0, goto label |
| END | End of programs (halt) |

## 3.3 Addressing Modes

Most of speech and audio processing is related with auto-correlation, convolution, and FIR calculation. Hence, addressing modes are to enhance the hardware computing capability for the algorithms. Five types of addressing modes allow access of data and instruction words from memory and registers: register, direct, indirect, immediate, and vector addressing modes. These detailed addressing formats are described in Appendix B.

The register addressing mode offers internal accessing operations of general-purpose registers. In this addressing mode, an ALU register contains three operands, as shown in this general operation: “RA Operation RB $\Rightarrow$ RC.” The destination operand is RC and the source operands are RA and RB. The direct addressing mode offers an immediate value as an index of memory address to access memory data. In this addressing mode, the data address is formed by 0-7 bits in the instruction. Because the length of instruction is short, the direct addressing mode only supports RAM block 0. The matrix addressing mode is designed for Durbin's algorithm [33] and used to compute matrix multiplication. For example, there is a 10×10 matrix multiplication. To access data in the matrix fast, the auxiliary registers (AR0 and AR1) are used to assist addressing the coordinate (X, Y) in the matrix. In matrix addressing, a three-operand instruction can be used in the indirect addressing mode. The vector addressing mode is used in data computation between memory and memory. This mode provides 512-data-length vector operations and can also execute parallel instructions that make auto-correlation function operate faster than the general-purpose DSPs.

Additionally, a control mode is defined to control data paths in the processor design. Programmers can use this mode to control their program flow and/or to easily set of repeat counters. Through two auxiliary registers (AR0 and AR1), the processor can execute two-level nested program. The function-finishing instruction and holding status are also in the control mode. The loop control is very useful for auto-correlation function in Durbin's algorithm [33] because they are all two-level nested programs. The mode is very efficient to handle the program flow without any additional instructions, which might be necessary to other general-purpose DSPs.

## 3.4 Matrix Processing Technique

Particularly, we design an auto-index method which uses auxiliary registers to address memory data as shown in Fig. 3-6. This method called matrix addressing can easily get memory data in a single multiplier instruction. When the instruction decoder gets the vector address, the address would represent the coordinate of the matrix. Matrix multiplication is based on the operation of RAM0 and R3 (the third general-purpose register). The results are stored to the R3 register. An example for the equation of matrix multiplication is as

$$y = \sum_{j=r-1}^{0} x[k, r-j]h[j+1, r]. \tag{4.1}$$

We can replace the above with the following LASP24 micro codes:

```
     RPB j, #r-1                       // set repeat block counter
L1:  MOV WIN[j+1, r], R3;              // move a coefficient to R3
     MPY R3, RAM0[AR0, r-j], R3        // matrix multiplication
     ADD R1, R3, R1                    // R1=R1+R3
     RETB j, L1                        // if j≠0, return to L1
```

The index of a matrix coordinate is defined by auxiliary registers (AR0 and AR1). The address index can automatically increase so that the pointer indicates the next matrix address. Hence, this addressing method enables a single-instruction matrix computation so that the size of program memory and the number of program memory access can be reduced.

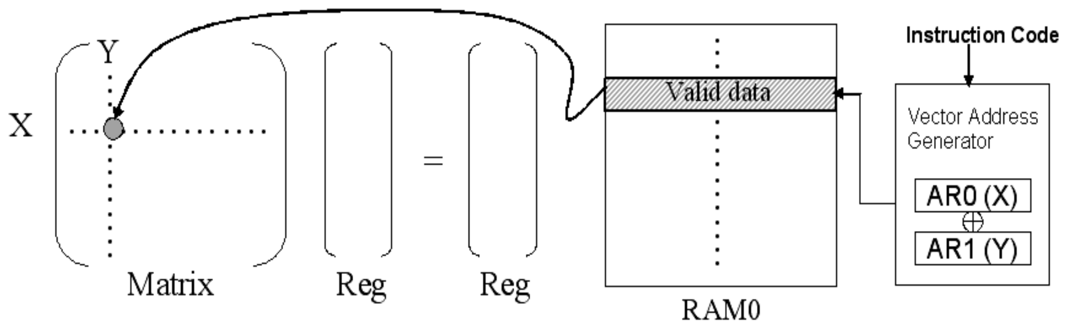


Fig. 3-6. Illustration for computing a matrix address with the vector addressing mode.

In Fig. 3-6, the instruction decoder gets the matrix position with four bits listed in Table 3-1 and then transfers them to the address processing unit. The processing unit can analyze and calculate the matrix address (X, Y) in RAM0. Table 3-1 shows the coordinate table of two matrix addressing modes. One is the indirect addressing mode as RAM0[AR0]; the other is the matrix addressing mode as RAM0[AR0L+1, AR0L+1]. The matrix coordinate is defined in AR0 and AR1. The index automatically adds one so that the pointer indicates the next matrix address. The vectors {0000, 0001} and {1110, 1111} are two special coordinates which can directly access the start and the end of row location in the matrix. Hence, the proposed matrix addressing method enables a single-instruction matrix computation so that the total number of program instructions can be reduced.

Table 3-1. The matrix coordinate for the matrix addressing mode, where AR0L and AR1L represent the lower four bits of AR0 (AR0[3:0]) and ar1 (AR1[3:0]), respectively.

| CODE | Addressing Mode | CODE | Addressing Mode |
|---|---|---|---|
| 0000 | RAM0[AR0] | 1000 | RAM0[AR0L-AR1L, AR0L] |
| 0001 | RAM0[AR1] | 1001 | RAM0[AR1L+1, AR0L+1] |
| 0010 | RAM0[AR0+AR1] | 1010 | Reversed |
| 0011 | RAM0[1111, AR0L] | 1011 | Reversed |
| 0100 | RAM0[AR1L+1, AR0L] | 1100 | RAM0[0000, AR0L] |
| 0101 | RAM0[1110, AR0L-AR1L] | 1101 | RAM0[1110, AR0L] |
| 0110 | RAM0[1110, AR0L+1] | 1110 | Reversed |
| 0111 | RAM0[AR0L+1, AR0L+1] | 1111 | RAM0[0001, AR0L] |

## 3.5 Vector Processing Technique

The SIMD-style vector processing scheme provides an approach to accelerating the processing of data streams. This technique can provide a significant speedup for communications, multimedia, and other performance-driven applications by using data-level parallelsim. In the vector processors [34], [35] the design can provide high-level operations that work on vectors — linear arrays of numbers. The vector processing unit supports both intra- and extra-memory operations. In the operation, elements work in parallel on the corresponding elements from multiple intra- or extra-memory sources and place the results in the corresponding fields in the destination operand memories. An operation example is the vector multiplication (VMPY) instruction shown in Fig. 3-7, and the instruction format and addressing representation are shown in Table 3-2.

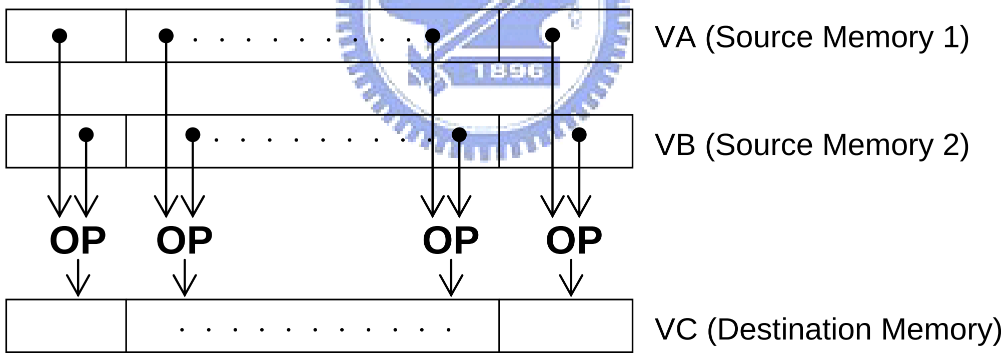


Fig. 3-7. An example of memory operations in LASP24, where OP indicates the vector multiplication. VA, VB, and VC represent different memory banks. They are defined in Table Table 3-2.

Table 3-2. The format of the vector addressing mode and the representation of vector addresses in LASP24, where OP indicates operation; VA, VB, and VC represent vector registers. The symbols, FIL, EXT, WIN, RAM0, and RAM1, are memory symbols.

| VC ⇐ VA[AR_A] OP VB[AR_B] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 23 ~ 19 | 18~16 | 15~14 | 13~12 | 11~10 | 9~8 | 7 ~6 | 5~4 | 3~2 | 1~0 |
| OPCODE | 011 | NU | FIL | EXT | RAM0 | RAM1 | VC | VA | VB |

| VL | FIL | EXT | RAM0 | RAM1 | VC | VA | VB |
|---|---|---|---|---|---|---|---|
| | 13~12 | 11~10 | 9 ~ 8 | 7 ~ 6 | 5 ~ 4 | 3 ~ 2 | 1 ~ 0 |
| 00 | FIL | EXT | AR0 | AR0 | RAM0 | RAM0 | RAM0 |
| 01 | FIL+AR0 | EXT+AR0 | AR1 | AR1 | RAM1 | RAM1 | RAM1 |
| 10 | FIL+AR1 | EXT+AR1 | AR0+AR1 | AR0+AR1 | EXT | EXT | WIN |
| 11 | FIL-AR0 | EXT-AR0 | AR1-AR0 | AR1-AR0 | R3 | - | FIL |

The vector multiplier has several important properties that solve most of the above problems as explained below.

1. The computation of each result is independent of the computation of previous results, allowing a pipelined operation without generating any data hazards.
2. A single vector instruction specifies a great deal of computation work. It is equivalent to executing an entire loop. Thus, the number of instruction fetch is reduced, and the bottleneck is considerably mitigated.
3. The vector instruction has a known memory access pattern. If the vector's elements are all adjacent, then fetching the vector from a set of heavily interleaved memory banks works very well. The high latency of initiating a main memory access versus accessing an instruction ROM is rather high, because a single access is initiated for the entire vector rather than for a single element. Thus, the cost of the latency to memory is seen only once for the entire vector, rather than once for each element of the vector.

4. Because an entire loop is replaced by a vector instruction whose behavior is predetermined, control hazards that would normally arise from the loop branch are nonexistent.

To illustrate the above features, we compare performance with a general-purpose DSP in computing the vector multiplication of 100 points. A vector multiplication instruction fetches data from RAM0 and RAM1 and feeds into ALU. ALU executes the "MAC" operation and adds the result to the accumulating register. The final results are stored to the external memory. An example of vector processing (100 points) is shown as follows.

```
L1:  MPY RAM0(r), RAM1(r), EXT(r);     // EXT(r)= RAM0(r)× RAM1(r)
     RETB r, L1                        // r=r+1. if r=100, then jump to L1
```

The total execution time is about 200 clock cycles. Hence, we use a single instruction within a repeat block to execute the parallel multiplication-and-accumulation in the auto-correlation operation. The above example demonstrates that LASP24 has higher performance in vector computation than the general-purpose DSPs such as the TI TMS320C3X series.

## 3.6 DMA and Interrupt Interface

LASP24 needs the interrupt and direct memory access (DMA) to process data transformation. The interrupt is to tell LASP24 that peripheral devices are to be active. DMA is to free the bus control and to deal with the operation of I/O-to-memory or memory-to-memory data moving. There is a control interface for interrupt and DMA. We take advantage of a simple finite state machine (FSM) to implement its circuit. Each state is described as follows in detail.

S0: monitor mode, the state is to initialize parameters and to detect interrupt and/or DMA in one instruction cycle.

S1: DMA mode.

S2: interrupt mode, in the state, the controller will save the program counter and all status into data memory. If data memory is not ready, stay in S2; otherwise go to S3.

S3: accepted mode, the state sends interrupt acknowledge and goes to S4.

S4: get vector mode, the state receives an interrupt from LASP24's data bus and sends a read signal to main memory. If main memory is not ready, stay in S4; otherwise go to S5.

S5: interrupt mode, after complete interrupt processing, the state switches the subroutine return address, and then goes to S6.

S6: rally normal mode, the state performs nothing and then goes to S0.

Polling of peripheral service requests monopolizes a significant amount of processing time. This service reduces system throughput, useful information processed or communicated during a specified time period. Therefore, it is advantageous, in terms of increasing throughput as well as reducing program complexity; if a peripheral device demands service directly from LASP24. Interrupts provide this capability. Essentially, LASP24's interrupt is a subroutine function call initiated by external hardware. A simple structure that allows a single device to interrupt LASP24 is shown in Fig. 3-8.

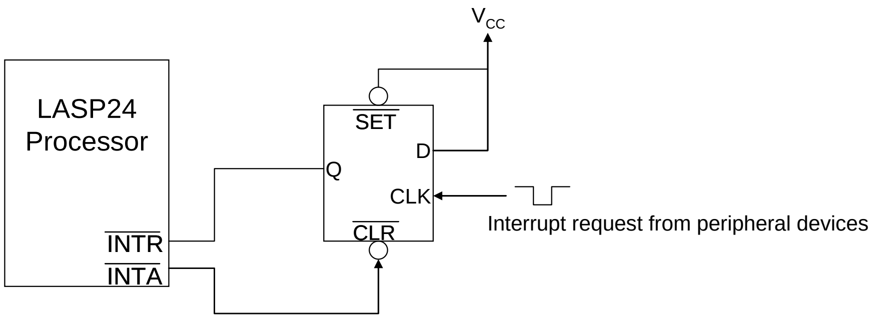


Fig. 3-8. Generation of LASP24's interrupt for a single peripheral device via interrupt request flag.

When a peripheral device requires service, the interrupt controller sets its corresponding register which is connected to an interrupt pin of LASP24. Thus, the register records the interrupt request until it is acknowledged by LASP24. Because the request is asynchronous, it may occur at any time when a program is executed. In order to resume program execution at the proper point, when the interrupt subroutine is finished, the return address is automatically restored to the program counter from data memory. Notice that when LASP24 handles an interrupt, it can not accept other interrupt requests at the time. A multiple loop interrupt mode is not supported at present. In response to an interrupt, the processing operations occur as follows.

1. The processing of the current instruction is completed.
2. An interrupt machine cycle is executed during which the program counter is saved and the flow control is transferred to an appropriate memory location.
3. A subroutine is executed.
4. When the subroutine is finished, the state of LASP24 is saved; otherwise goes to Step 3.
5. The saved state of LASP24 is restored.
6. The flow control is returned to the instruction which follows the interrupted instruction.

## 3.7 Power Optimization

For many consumer electric applications, low average power dissipation is desirable and for certain special applications low power dissipation is of critical importance. Most of power reduction techniques emphasize on reducing the level of activity in some portion of the circuit. Since LASP24 has high activity, it must include a power management mechanism. The power management means to analyze and realize substantial power saving [36] by stopping the clock during proper time period. Clock power reduction is important in a synchronous design, since as was noted earlier, it can contribute to a large portion of the overall power budget. Minimization of clock power falls in to several categories including clock distribution optimizations, clock gating, and low-swing clocking techniques. An analysis methodology operating at register transfer level (RTL) is a key factor to obtain early estimation results, while maintaining an acceptable level of accuracy in the results.

The control unit and the arithmetic unit dissipate most of the power since they are usually active. Hence, our power saving design focuses on these two units. In the control unit, a modified finite state machine (FSM) is designed for power saving. This design is to use the knowledge of the next state function to generate an activation signal only when the system control unit needs to perform a state transition. This scheme makes the modified system control unit functionally equivalent to the original system control unit, with a reduction in power dissipation and a small increase in area and critical path delay.

The arithmetic unit is designed as a parallel architecture as shown in Fig. 3-9(a). In this architecture, the three arithmetic units (integer, floating-point, and vector) have the same inputs but operate independently. Hence, when the input signals (A and B) are changed, all of these units always re-compute their outputs, and a multiplexer is then used to select the final result. This architecture wastes most of dynamic power, since the parallel architecture only processes one instruction during a cycle. Gated clocking [37] is a commonly applied technique used to reduce power by gating of clock signals to registers or latches. Gating may be done when there is no required activity to be performed by logic whose inputs are driven from a set of storage elements. Since new output values from the logic will be ignored, the storage elements feeding the logic can be blocked from updating to prevent irrelevant switching activity in the logic.

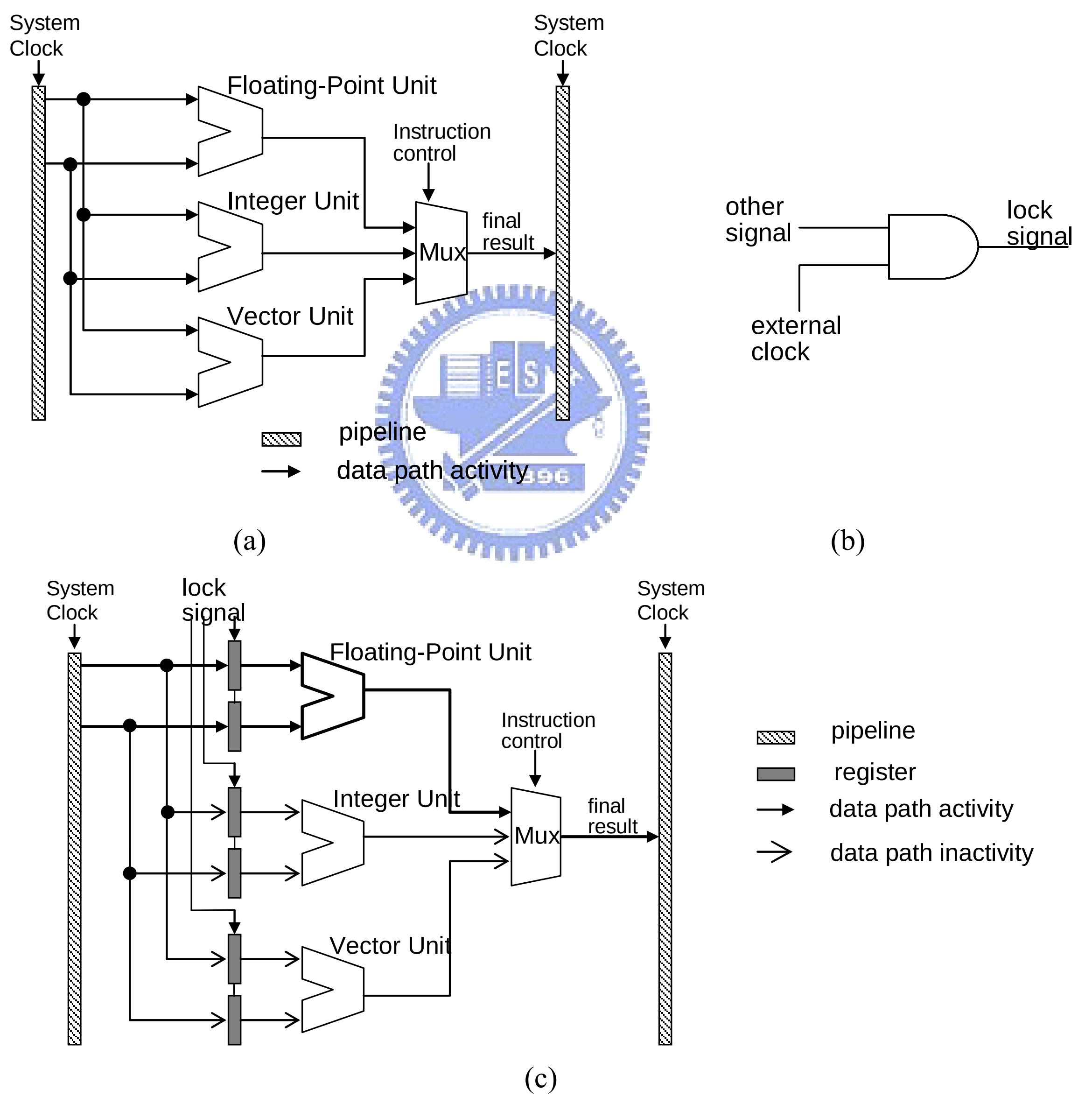


Fig. 3-9. The gated-clock scheme in (a) Parallel arithmetic unit, (b) Clock gating, and (c) Power saving parallel architecture.

To reduce dynamic power by the system clock, it is important to minimize the switching activity by powering down the arithmetic unit when they are not performing “useful” operations. Thus, we apply the gated-clock scheme to reduce the dynamic power. Fig. 3-9(c) shows the parallel architecture controlled by the gated-clock scheme.

## 3.8 Coprocessor: Reverberator

Based on the description of the multi-tap FIR filter in Chapter 2.2, we propose the pseudo-random FIR architecture using two-stage pipeline to perform convolution computation. The two-stage FIR shown in Fig. 3-10 can independently perform different channels (left and right) at the same time. The resulting signals are summated together and stored into the circular buffer.

In Fig. 3-10, real-time audio signals via the $I^2S$ (Inter-IC Sound) interface are stored into the FIR circular buffer (FCB) in order. After one cycle latency, the first signal is loaded and fed into the first stage (MAC1). In next cycle time, the second signal is loaded and fed into the second stage (MAC2), and the first stage has begun to perform convolution using the input signal from FCB and a random coefficient from the pipeline register. Each MAC block performs the total of 256 loops at most due to generating non-zero and zero coefficients. The FIR state machine can search a non-zero value beforehand and quickly load it to the MAC block during the same cycle. Although input and output signals are all 16 bits, to avoid the overflow condition in accumulation, the input signal is scaled. All adders are set to 20-bit complement operations.

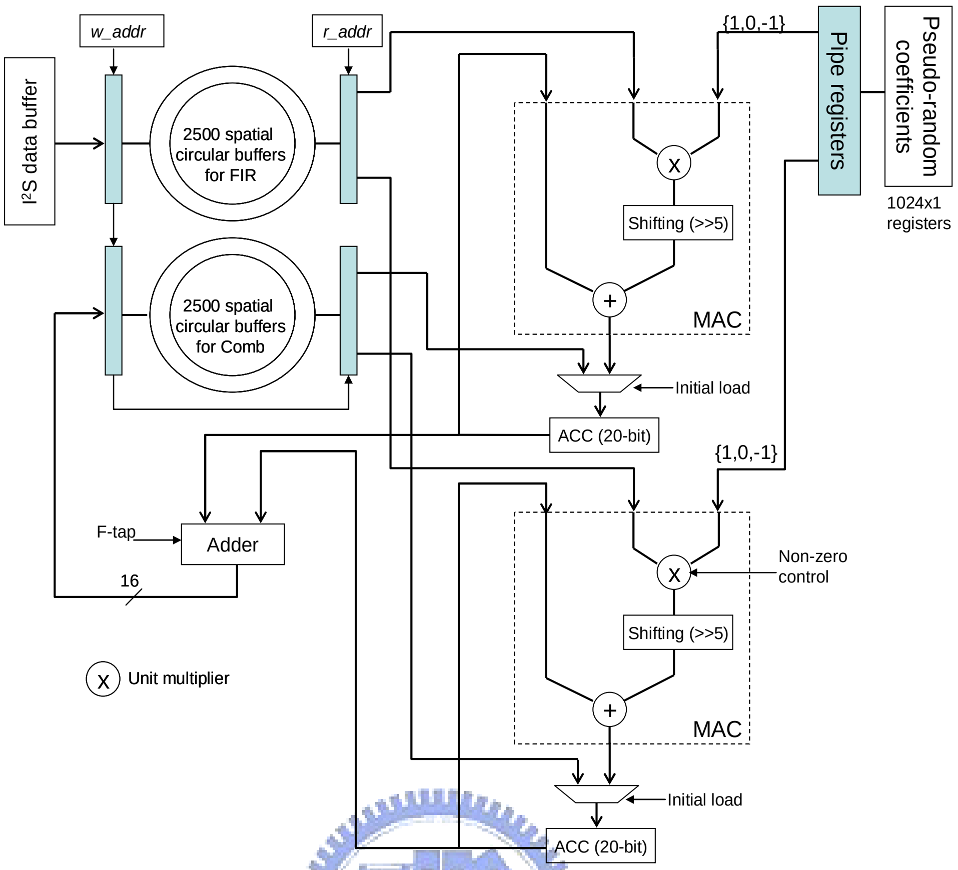
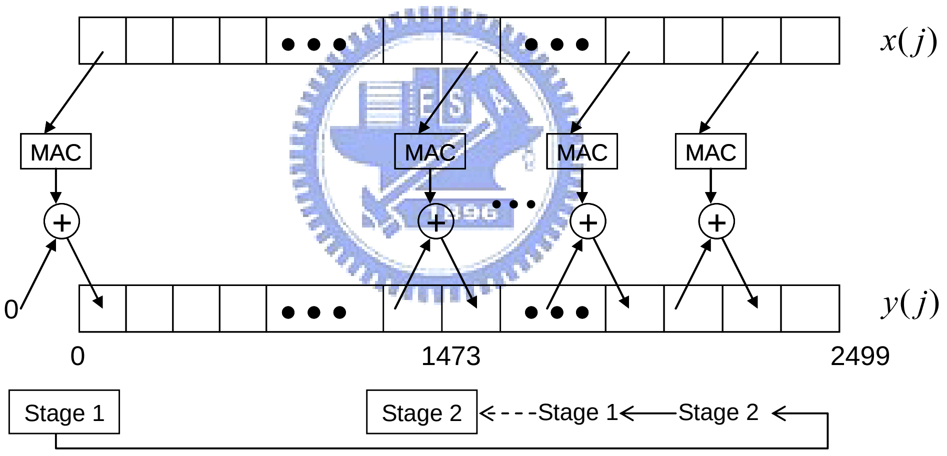


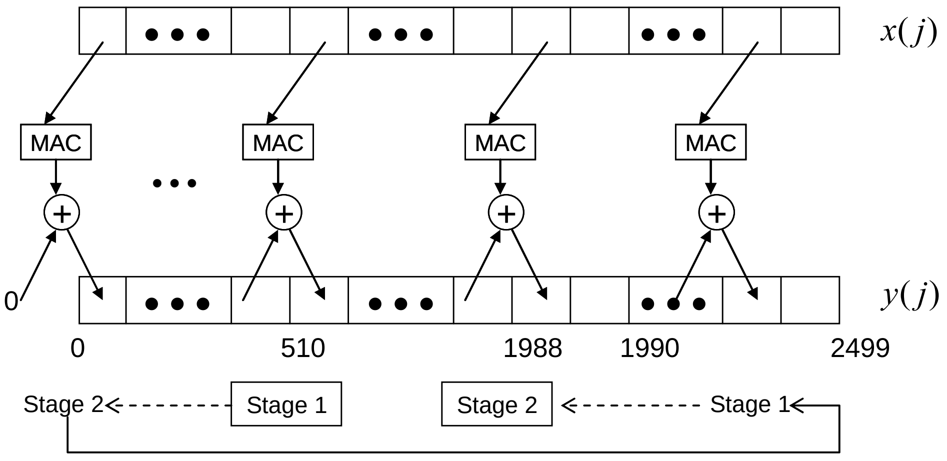
Fig. 3-10. The proposed pseudo-random FIR architecture with two-stage pipeline. The adder is a 2'sc addition operation. The multiplier limited by the non-zero control is a unit multiplication operation.

The delay effect of early reflections is based on Eq. (2.3) to generate the memory address. The address is used to access FIR and comb circular buffers. Each value in the comb circular buffer (CCB) will be accumulated with the previous computational result. Fig. 3-11 shows the operation diagram in the circular buffer when the block index is equal to 0, 510, and 2000, respectively. The convolution operation is based on Eq. (2.2). Each input signal $x(n)$ has to perform 256-tap FIR (possible less than 256 times), then the result of adding to $y(n-1)$ is stored to $y(n)$.

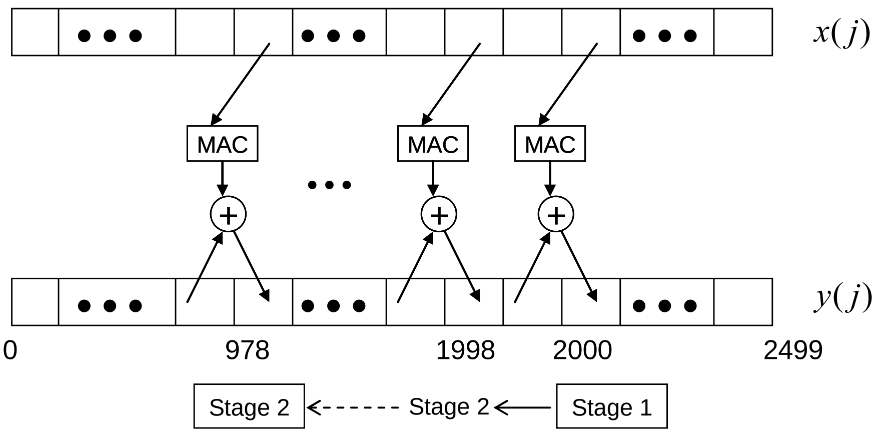
The two-stage architecture as Fig. 3-12 uses the finite state machine (FSM) to control active flows. There are five states shown in Fig. 3-12 and described as follows. First the sequence of 1's, 0's, and -1's is generated by the random coefficient generator, and then they are loaded into 1024 by 1 registers. There is a parameter *Finish* which determines to turn on or off the FSM operation. When *Finish*=1 (Step 5), FIR results are restored into CCB. Step 2 can set a loop counter and calculate the delay address for two circular buffers (FIR and comb). The calculation of CCB address is based on Eq. (2.3). At the same time, the state does not check coefficient registers unit finding a non-zero value. Note that the operation should be finished in one machine cycle. In Step 3, the state can get the input signal and previous MAC result from two circular buffers respectively. Finally, MAC and acclamation operations are performed by Eq. (2.2) in Step 4. The whole flow will repeatedly perform convolution operations of 512 times under the half of a sampling time (i.e., 44.1 kHz in one cycle per 22.68ms).



(a)



(b)



(c)

Fig. 3-11. The computational diagram of the circular buffer when the addressing index indicates (a) *j*=0, (b) *j*=510, and (c) *j*=2000.

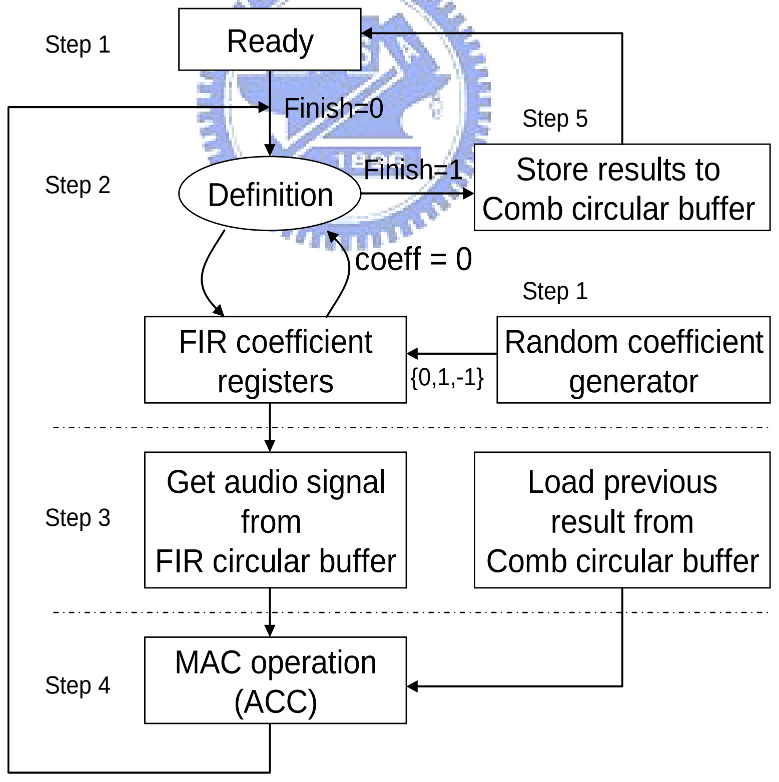
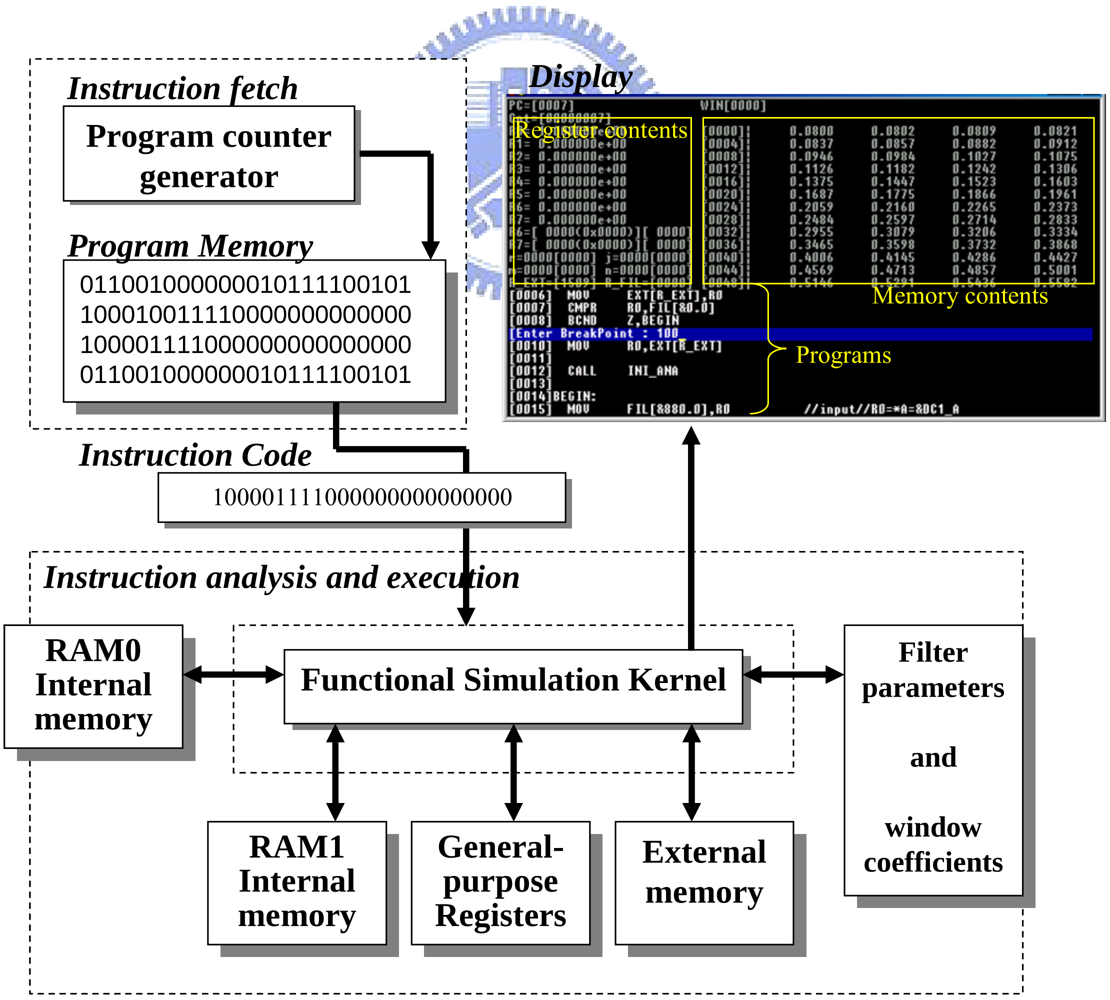

Step 1
Ready
Finish=0
Step 5
Step 2
Definition
Finish=1
Store results to Comb circular buffer
coeff = 0
Step 1
FIR coefficient registers
{0,1,-1}
Random coefficient generator
Step 3
Get audio signal from FIR circular buffer
Load previous result from Comb circular buffer
Step 4
MAC operation (ACC)

Fig. 3-12. Fully FSM control flows for two-stage architecture.

## 3.9 Development Software

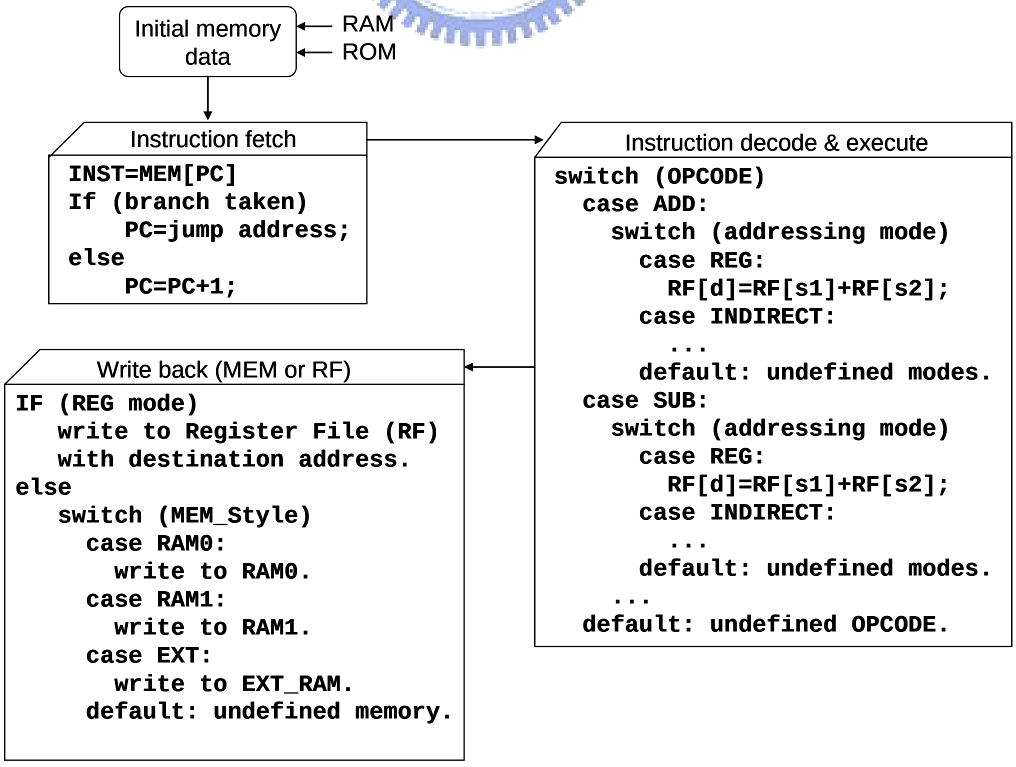
In order to verify and debug the DSP programs, a tool called DEFY-I is developed for functional emulation. The DEFY-I is an instruction-set-level hardware emulator for the processor core. With the emulator, the instructions could be taken out from the program memory and put into the instruction register for instruction analysis and execution. Finally, the execution results are written back to the register file or data memories. The flowchart of DEFY-I is shown in Fig. 3-13. The whole emulator is constructed as the functional simulation kernel and could connect to other peripheral devices to perform the memory and display functions.



```
PC=[0007]                  WIN[0000]
  =[00000007]               [0000]|     0.0800     0.0802     0.0809     0.0821
R1= 0.000000e+00            [0004]|     0.0837     0.0857     0.0882     0.0912
R2= 0.000000e+00            [0008]|     0.0946     0.0984     0.1027     0.1075
R3= 0.000000e+00            [0012]|     0.1126     0.1182     0.1242     0.1306
R4= 0.000000e+00            [0016]|     0.1375     0.1447     0.1523     0.1603
R5= 0.000000e+00            [0020]|     0.1687     0.1775     0.1866     0.1961
R6= 0.000000e+00            [0024]|     0.2059     0.2160     0.2265     0.2373
R7= 0.000000e+00            [0028]|     0.2484     0.2597     0.2714     0.2833
R6=[ 0000(0x0000)][ 0000]   [0032]|     0.2955     0.3079     0.3206     0.3334
R7=[ 0000(0x0000)][ 0000]   [0036]|     0.3465     0.3598     0.3732     0.3868
 =0000[0000] j=0000[0000]   [0040]|     0.4006     0.4145     0.4286     0.4427
 =0000[0000] n=0000[0000]   [0044]|     0.4569     0.4713     0.4857     0.5001
R_EXT=[1509] R_FIL=[0000]   [0048]|     0.5146     0.5291     0.5436     0.5582
[0006]  MOV     EXT[R_EXT],R0
[0007]  CMPR    R0,FIL[&0.0]
[0008]  BCND    Z,BEGIN
[Enter BreakPoint : 100_
[0010]  MOV     R0,EXT[R_EXT]
[0011]
[0012]  CALL    INI_ANA
[0013]
[0014]BEGIN:
[0015]  MOV     FIL[&880.0],R0          //input//R0=*A=&DC1_A
```

Fig. 3-13. The structure of DEFY-I for LASP24.

The high-level algorithm model with the LASP24 assembly language is translated into the machine language by the developed translator, and then a conversion table of mnemonic and an operation code is generated. For the software development, the tool of an effective functional simulation supports software developers so that the software application can be embedded into the tool to verify its function. The tool, named hardware emulator, can help software developers to simulate and debug developing applications. The emulator is an instruction-set-level hardware emulator based on an application-specific speech processor. With the emulator, the basic operation of LASP24 is to take out the instruction from the program memory first, set it to the INST register, decode the instruction, execute the decoded instruction, and finally write back the operation results to the register file (RF) or data memories (RAM0, RAM1, EXT RAM). The operation flow of the hardware emulator is shown in Fig. 3-14 with C pseudo codes, and its structure is shown in Fig. 3-13. The whole emulator is constructed as the functional simulation kernel and connects to other peripheral devices as memories.



```
INST=MEM[PC]
If (branch taken)
    PC=jump address;
else
    PC=PC+1;
```

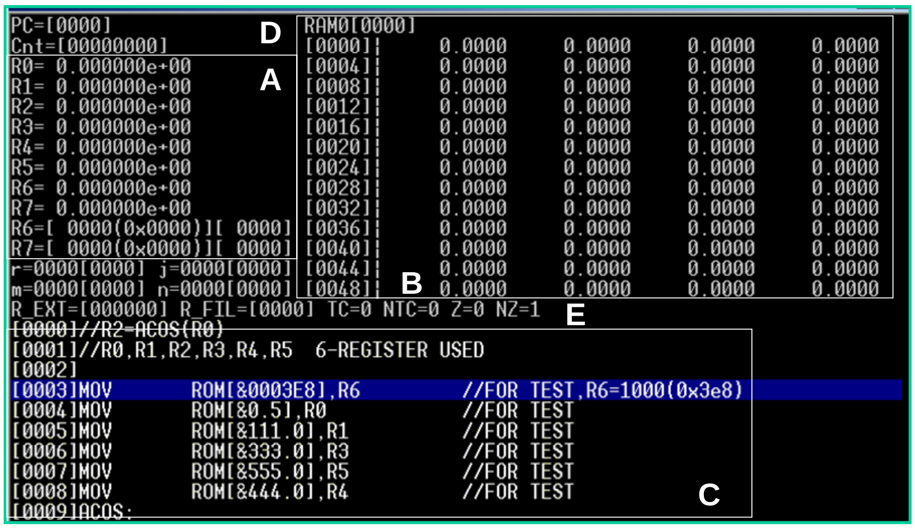
```
switch (OPCODE)
  case ADD:
    switch (addressing mode)
      case REG:
        RF[d]=RF[s1]+RF[s2];
      case INDIRECT:
        ...
      default: undefined modes.
  case SUB:
    switch (addressing mode)
      case REG:
        RF[d]=RF[s1]+RF[s2];
      case INDIRECT:
        ...
      default: undefined modes.
    ...
  default: undefined OPCODE.
```

```
IF (REG mode)
   write to Register File (RF)
   with destination address.
else
   switch (MEM_Style)
     case RAM0:
       write to RAM0.
     case RAM1:
       write to RAM1.
     case EXT:
       write to EXT_RAM.
     default: undefined memory.
```

Fig. 3-14.Operation ows of the emulator.

For the hardware emulator to be useful for effectively improving the flow of software development, we identify the following functions and requirements:

- ***Step execution***: The emulator can execute one-by-one instruction so that the programmer can trace the execution result in an instruction or clock cycle.
- ***Free run***: When a program prototype is finished, we can use the free-run way to simulate the program. Through this way, an expected result will be estimated.
- ***Set breakpoint***: Users can press the breakpoint value based on the program counter. Until the program count is equal to the breakpoint value, the program always runs.
- ***Displays***: The screen of the emulator is shown in Fig. 3-15. It can display information as the program counter (PC) in the region D, general-purpose registers (R0_R7) in the region A, a source program in the region C, auxiliary registers (R, J, M, N, R EXT, R FIL), status registers (TC, NTC, Z, NZ) in the region E, and the contents of memory banks (RAM0, RAM1, ROM, EXT RAM) in the region B, where RAM1 and RAM2 indicate the internal memory, ROM indicates filter and window ROMs, and EXT RAM indicates the external (or on-chip) memory.



```
PC=[0000]                          RAM0[0000]
Cnt=[00000000]                     [0000]|    0.0000    0.0000    0.0000    0.0000
R0= 0.000000e+00                   [0004]|    0.0000    0.0000    0.0000    0.0000
R1= 0.000000e+00                   [0008]|    0.0000    0.0000    0.0000    0.0000
R2= 0.000000e+00                   [0012]|    0.0000    0.0000    0.0000    0.0000
R3= 0.000000e+00                   [0016]|    0.0000    0.0000    0.0000    0.0000
R4= 0.000000e+00                   [0020]|    0.0000    0.0000    0.0000    0.0000
R5= 0.000000e+00                   [0024]|    0.0000    0.0000    0.0000    0.0000
R6= 0.000000e+00                   [0028]|    0.0000    0.0000    0.0000    0.0000
R7= 0.000000e+00                   [0032]|    0.0000    0.0000    0.0000    0.0000
R6=[ 0000(0x0000)][ 0000]          [0036]|    0.0000    0.0000    0.0000    0.0000
R7=[ 0000(0x0000)][ 0000]          [0040]|    0.0000    0.0000    0.0000    0.0000
r=0000[0000] j=0000[0000]          [0044]|    0.0000    0.0000    0.0000    0.0000
m=0000[0000] n=0000[0000]          [0048]|    0.0000    0.0000    0.0000    0.0000
R_EXT=[000000] R_FIL=[0000] TC=0 NTC=0 Z=0 NZ=1
[0000]//R2=ACOS(R0)
[0001]//R0,R1,R2,R3,R4,R5  6-REGISTER USED
[0002]
[0003]MOV      ROM[&0003E8],R6        //FOR TEST,R6=1000(0x3e8)
[0004]MOV      ROM[&0.5],R0           //FOR TEST
[0005]MOV      ROM[&111.0],R1         //FOR TEST
[0006]MOV      ROM[&333.0],R3         //FOR TEST
[0007]MOV      ROM[&555.0],R5         //FOR TEST
[0008]MOV      ROM[&444.0],R4         //FOR TEST
[0009]ACOS:
```

Fig. 3-15. The hardware emulator.

- ***Debug information***: When the emulator loads the object codes, the related debug information is read as well. At the same time, the emulator can show the executing instruction located in the source code to suit debugging for programmers.

- ***Emulator initialization***: When the emulator is enabled, it can search related initial files in the current working directory. If these initial files including the filter parameters, window coefficients, and initial values of the external memory exist, the emulator can auto-load them and finish initialization.

When design is completed, we check them against the specifications for completeness and correctness. The co-verification method is created, and a script file is described as follows:

```
load (analyzed sources);
load (target library);
load (debugging information);
while ( (read (instruction) != NULL) or (!finish) )
    execute the instruction from HW simulator;
    check (debugging information);
    match the results;
    if ( mismatch )
        printf (show messages and different values);
        errcount++;
    endif
end
if ( errcount != 0 )
    printf (“Here are %d errors between HW and SW”, errcount);
else
    printf (“Maching is finished. No error found.”);
endif
```

The automatic verification can help us to check whether the specifications of the hardware/software co-design are correct. If any violation, the output information show immediately the location which indicates the error. Thus the debugging time can be reduced, and the functional design can quickly meet our requirements.

# CHAPTER 4

# SIMULATION RESULTS

These two algorithms, speech coding and audio enhancement processing of reverberation, are performed on the proposed digital signal processor, LASP24. They are implemented with LASP24's assembly language and can be performed in real time. Finally, the performance result is compared with TI TMS320C3X.

## 4.1 Speech Processing

### 4.1.1 LPC and pitch estimation

Fig. 4-1 shows the microprogramming flow for performing three kernel functions (LPC, PE, and test mode) analyzed in Chapter 2. The C program was used to verify the speech processing algorithms and to test the floating-point precision. According to the experimental results, Table 4-1 shows the 10-order LPC coefficients in different bit numbers (24-bit and 32-bit) of floating-point precision. The maximal error occurred at the frequency 18.52 Hz, and the error of the two different bit numbers of floating-point precision in Table 4-1 is maximal when the LPC order is equal to 4. When precision or iterations of divider were not high enough, the reconstructed speech signals would be unnatural. After we listened to the synthesized speech, the 24-bit floating-point precision appeared to be good enough.

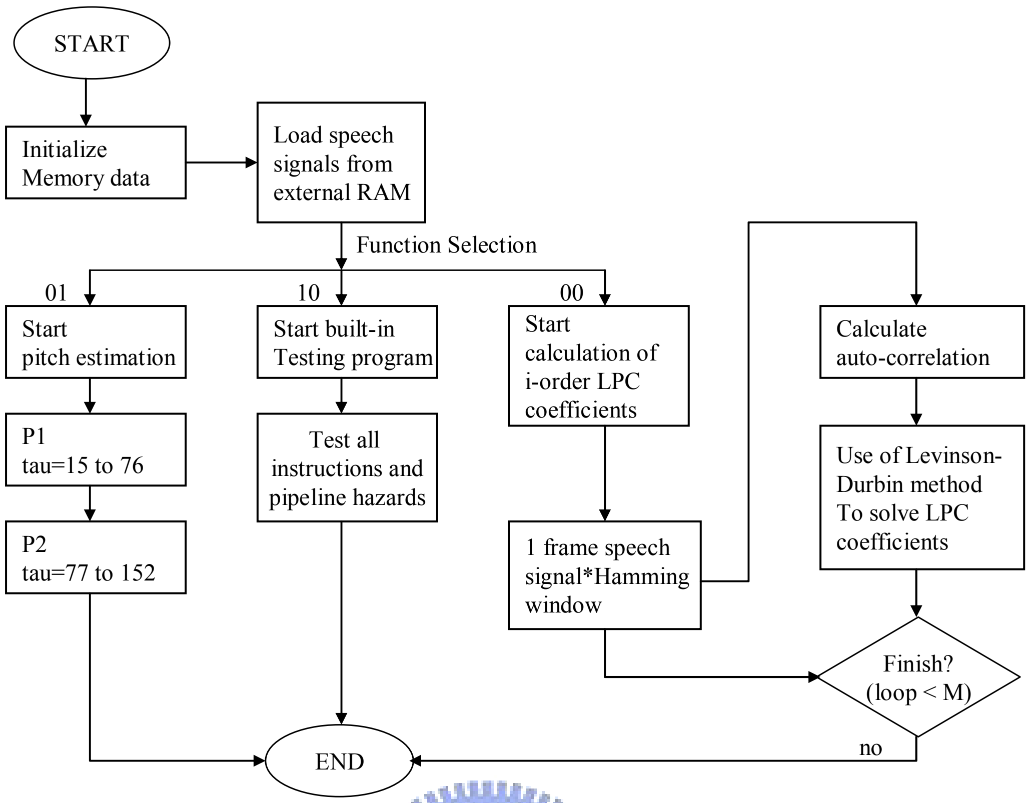

START
Initialize Memory data
Load speech signals from external RAM
Function Selection
01
10
00
Start pitch estimation
Start built-in Testing program
Start calculation of i-order LPC coefficients
Calculate auto-correlation
P1 tau=15 to 76
Test all instructions and pipeline hazards
Use of Levinson-Durbin method To solve LPC coefficients
P2 tau=77 to 152
1 frame speech signal*Hamming window
Finish? (loop < M)
no
END

Fig. 4-1. The microprogramming flow in the program ROM of LASP24.

Table 4-1. Simulation results of LPC calculations in different floating-point precision.

| LPC Order | 32-bit Floating Point | 24-bit Floating-Point |
|---|---|---|
| 1 | -1.948923 | -1.954345 |
| 2 | 0.923492 | 0.913543 |
| 3 | -0.052776 | 0.017284 |
| 4 | 0.841343 | 0.730545 |
| 5 | -1.204289 | -1.122589 |
| 6 | -0.476735 | -0.426231 |
| 7 | -0.280020 | -0.223022 |
| 8 | -0.945771 | -0.904251 |
| 9 | -0.968852 | -0.966308 |
| 10 | 0.296600 | 0.302504 |

The RTL codes were written by Verilog language and simulated. Design Compiler was used to transfer the RTL codes to gate-level codes. In RTL simulation, we obtained the execution time of the realized speech processing algorithms in Table 4-2, where Pitch 1

(P1) performs $\tau$=15 to 76 and Pitch 2 (P2) performs $\tau$=77 to 152 in Eq. (2.13).

Table 4-2. Timing simulation results. The time unit of execution is microsecond (ms), and the total time of execution is the sum of LPC and PE computation time.

| Algorithms | Execution (cycles) | Vector operation Rate (%) | Execution Time (ms) 25 MHz | 33 MHz | 40 MHz |
|---|---|---|---|---|---|
| LPC | 3,298 | 2,348 (71) | 0.13 | 0.1 | 0.08 |
| P1 | 14,346 | 13,698 (95.5) | 0.57 | 0.43 | 0.35 |
| P2 | 17.424 | 16,680 (95.7) | 0.70 | 0.52 | 0.44 |
| Total | 35,068 | 32,736 (93.3) | 1.40 | 1.05 | 0.87 |

These simulations were executed with the operating frequency of 25 MHz, 33 MHz, and 40 MHz, respectively. The time for vector and matrix operations was about 93.3% of the whole algorithm; that is, the rate of chip running at optimal condition was 93.3%. The chip's internal driving ability between cells to cells was simulated in gate level simulations, too.

After the timing simulation, the post-layout simulation was performed. Final power dissipation and maximal operating frequency could be estimated at this stage. The LASP24's performance in typical (33 MHz), best (40 MHz), and worst (25 MHz) cases had also been simulated. In the typical case, LASP24 can provide the computation capability of 66.6 MFLOPS (Million Floating-point Operations per Second) and 33.3 MIPS (Million Instructions per Second). The best condition was achieved at 80 MFLOPS and 40 MIPS in a single cycle. In the worst case, the computation power is 50 MFLOPS and 25 MIPS. At the room temperature 23 (25◦C ~27◦C) and 5 V, the current requirement was 4 mA, about 20 mW, and the maximal frequency is 28.5 MHz which was lower than the gate level simulation result. At the worst case, 85◦C and 4.5 V, the current requirement is 3.2 mA, about 14.4 mW, and the maximal frequency was 20 MHz. Even in the worst case, LASP24 still could provide 50 MFLOPS and 25 MIPS computation power that was higher than that of TMS320C30.

We compared the performance of the LASP24 processor to that of TMS320C3x series, which are floating-point general-purpose DSPs. Fig. 4-2 shows the floating-point operation ability of each processor and the comparisons of vector operation ability. At the best case, LASP24 at 40 MHz provided 80 MFLOPS that was much better than TMS320C31 at 50 MHz did. In the vector operation mode, we set the vector processing ability of LASP24 at 25 MHz as index 100 and compared it with other processors. In the figure, higher value indicates higher performance. At the best case, LASP24 at 40 MHz was about 4.75 times higher than TMS320C30 and about 3.2 times higher than TMS320C31.

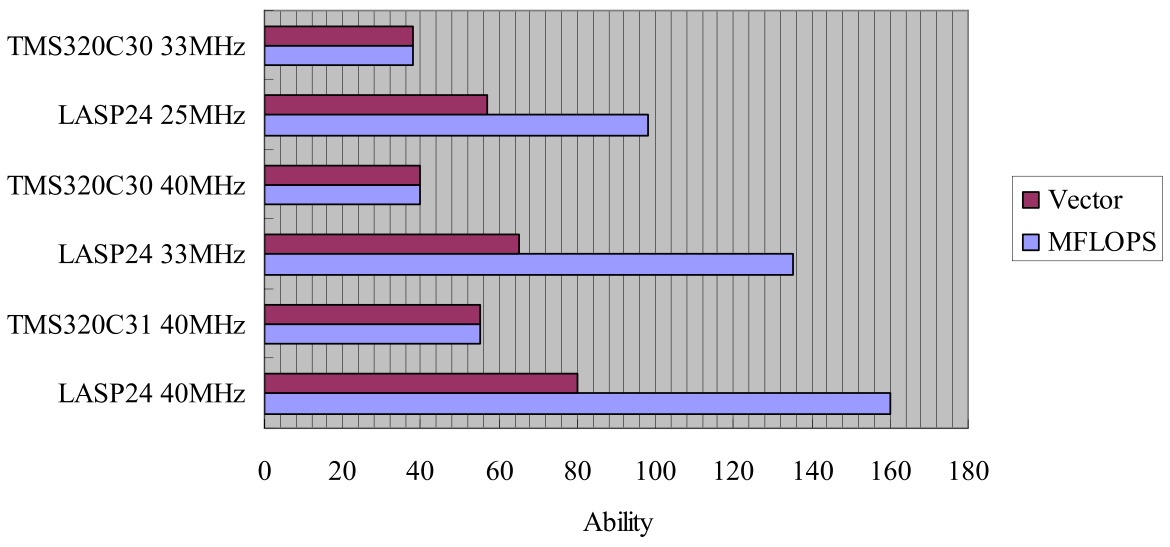


Fig. 4-2. Performance comparisons of LASP24 and TMS320C3x.

### 4.1.2 MELP Coding

The MELP coder is divided into an encoder and a decoder module. The frame size is 22.5ms (180 samples) with a sampling frequency of 8000Hz. The MELP coder is based on the traditional Linear Prediction Coding (LPC) parametric model, but also includes five additional features: mixed excitation, aperiodic pulses, adaptive spectral enhancement, pulse dispersion, and Fourier magnitude. The encoder uses 10th order LPC coefficients, which are transformed into line spectral frequencies or quantization and transmission. For each voiced and unvoiced frame, the parameters computed are listed in Table 4-3.

Line spectral frequencies are computed from the prediction coefficients, which uses Chebyshev polynomials. A fast numerical method is used for implementation on the proposed processor. Final pitch is computed using an autocorrelation analysis on the low passed residual signal:

$$r(\tau)=\frac{c_\tau(0,\tau)}{\sqrt{c_\tau(0,0)c_\tau(\tau,\tau)}}, \tag{4.1}$$

and

$$c_\tau(m,n)=\sum_{-\lfloor \tau/2 \rfloor-80}^{-\lfloor \tau/2 \rfloor+79} s_{k+m}s_{k+n} \ , \tag{4.2}$$

where $\tau$ is the lag. The computation of the autocorrelation sequence is centered on the last sample of the past frame. Band pass voicing strengths are computed using autocorrelation analysis about the pitch lag for each of the bands. Gain is computed twice per frame using an adaptive window size, which is a multiple of the pitch period. A residual signal is obtained by filtering the input speech using the set of de-quantized LPC coefficients. An FFT is performed on this residual signal and a search is performed selecting 10 Fourier magnitudes.

Table 4-3. Bit allocation for the MELP coder

| Parameters | Per Frame | Voiced | Un-voiced |
|---|---|---|---|
| LSFs | 10 | 25 | 25 |
| Pitch | 1 | 7 | 7 |
| Band pass voicings | 5 | 4 | - |
| Aperiodic flag | 1 | 1 | - |
| Fourier magnitudes | 10 | 8 | - |
| Gain | 2 | 8 | 8 |
| Error protection | | - | 13 |
| Sync bit | | 1 | 1 |
| Total | | 54 | 54 |

We provide the instruction set for matrix operations which reduce the size of program memory to 12K Bytes. For example, the autocorrelation operation of (4.2) is optimized and implemented by LASP24's instructions as follows:

```
//input R1 = s[0]
//input R2 = x                                      Variable definition
//input R4 = n-m

            FIX       R2, R7
            SHF       R7, +1
            FLOAT     R7, R2
            ADD       R2, ROM[&80.0]                Set initial address
            SUB       R2, R1, R1                    For input signals
            ADD       R3, R1, R1
            FIX       R1, R7
            LDE       R_EXT, R7
            RPB       j, #160
L1:         MOV       EXT[R_EXT+j], RAM0[j]         Data moving from external
            RETB      j, L1                         RAM to RAM0(A)
            CMPR    R4, ROM[&0.0]
            BCND    NZ, P1
            RPB     j, #160                          Data moving from external
L2:         MOV     RAM0[j], RAM1[j]                 RAM to RAM1 (B)
            RETB    j, L2                            If m=n, the A=B
            BCND    Z, P2
P1:         ADD     R4, R1, R1
            FIX     R1, R7
            LDE     R_EXT, R7                        Data moving from external
            RPB     j, #160                          RAM to RAM1 (B)
L3:         MOV     EXT[R_EXT+j], RAM1[j]            If m≠n, the A≠B
            RPB     j, L3
P2:         MOV     FIL[&0.0], R3
            RPB     j, #160                          Calculate Cx(m,n) and
COR_MAC:    MAC     RAM0[j], RAM1[j], R3             store in R3
            RETB    j, COR_MAC
```

### 4.1.3 Power Analysis

To achieve power saving, LASP24 was also designed with a gated-clock architecture. The power dissipation of the LASP24 is summarized in Table 4-4, which includes average dynamic power dissipation and power reduction. Power reduction compared the average power dissipation of the gated-clock design and the original implementations. It was expressed as a percentage by the following equation:

$$Power\ reduction = (1 - power\ ratio) \times 100, \tag{4.3}$$

where the power ratio is $P_{gatedclock}/P_{original}$, and the ratio is the average dynamic power dissipation. Table 4-4 lists the power dissipation of three parts including the ALU unit, the system control, and the whole design in different operation frequencies and processes. The results indicate that ALU unit wastes more power than the other units. The reason is possibly that the parallel processing components are all enabled in the ALU unit. After power optimization, average power reduction is about one-fourth at 33 MHz and 40 MHz, but can be reduced by 60% at 25 MHz. We find that the power dissipation rate is reduced to about 3/4 of the total power for the whole arithmetic unit shown in Table 4-4.

Table 4-4. Power dissipation analysis of LASP24 between different processes.

| Before/After gated-clock design (0.6um)<br>(5V supply voltage, unit mW) | | | | |
|---|---|---|---|---|
| Frequency | ALU unit | Control unit | Average power | Power reduction |
| 25 MHz | 61.39/30.14 | 2.63/1.52 | 46.72/16.22 | 66% |
| 33 MHz | 97.55/72.33 | 5.51/3.50 | 68.16/51.32 | 25% |
| 40 MHz | 117.02/80.17 | 4.22/3.83 | 84.95/63.51 | 24.7% |

| Before/After gated-clock design (0.35um)<br>(3.3V supply voltage, unit mW) | | | | |
|---|---|---|---|---|
| Frequency | ALU unit | Control unit | Average power | Power reduction |
| 25 MHz | 59.13/25.84 | 2.01/0.93 | 43.18/15.71 | 64% |
| 33 MHz | 82.61/70.19 | 4.83/2.87 | 66.43/49.22 | 26% |
| 40 MHz | 108.62/78.30 | 3.76/3.15 | 84.95/55.75 | 34% |
| 80 MHz | 138.47/91.05 | 9.92/8.85 | 102.40/89.56 | 12.5% |

| Before/After gated-clock design (0.18um)<br>(1.8V supply voltage, unit mW) | | | | |
|---|---|---|---|---|
| Frequency | ALU unit | Control Unit | Average power | Power reduction |
| 25 MHz | 50.37/22.31 | 2.01/1.02 | 39.92/14.83 | 63% |
| 33 MHz | 84.27/68.20 | 4.18/2.50 | 65.88/48.13 | 26.9% |
| 40 MHz | 105.85/80.28 | 4.00/2.93 | 78.05/52.60 | 32.6% |
| 80 MHz | 127.56/90.63 | 9.21/7.84 | 97.69/89.92 | 7.95% |
| 100 MHz | 181.35/137.72 | 15.02/11.50 | 166.67/128.45 | 22.9% |

| Before/After gated-clock design (Cyclone FPGA)<br>(3.3V supply voltage, unit mW) | | | | |
|---|---|---|---|---|
| Frequency | ALU<br>unit | Control<br>unit | Average<br>power | Power<br>reduction |
| 25 MHz | 58.67/26.11 | 2.24/1.07 | 40.92/17.04 | 58% |
| 33 MHz | 81.43/69.33 | 4.77/2.61 | 67.13/45.28 | 32.5% |
| 40 MHz | 100.72/75.20 | 3.15/2.53 | 83.72/56.06 | 33% |
| 80 MHz | 140.13/89.27 | 10.13/7.39 | 109.86/90.32 | 17.8% |
| 100 MHz | 172.16/112.94 | 13.76/11.53 | 138.34/98.74 | 28.6% |

## 4.2 Reverberation Algorithm

### 4.2.1 DSP Programming

Digital reverberation algorithms tried to mimic a room reverberation by using primarily two types of infinite impulse response (IIR) filters, so that the output would gradually decay. One such filter is the comb filter, which gets its name from the comb-like notches in the frequency response. The other primary filter is the allpass filter. The allpass filter has the nice property that all frequencies are passed equally, reducing a coloration of the sound.

Much of the early work on digital reverberation was done by Schroeder, and one of his well-known reverberation designs uses four comb filters and two allpass filters. More advanced algorithms can be developed to model specific room sizes. With chosen room geometry, source, and listener location, ray tracing techniques can be used to come up with a reverb pattern. By modifying Schroeder's algorithm, a finite impulse response (FIR) filter is used to create the early reflections, and then IIR filters are used to create the diffuse reverberation. Low pass filters may be used to model the air absorption. Reverberation designs can be obtained as shown in Fig. 3-12.

Performing designs and real-time prototyping of digital reverberation algorithms is based on random FIR filters, as presented in [13] to construct artificial early reflection. The four parallel comb filters and four cascade all-pass filters are to model the late reverberation and to increase echo density. Consider a modified comb filter in the frequency given by:

$$H(z) = \frac{z^{-M}}{1 - gH_1(z)z^{-M}} \tag{4.4}$$

and

$$H_1(z) = \frac{1}{1 - az^{-1}}, \tag{4.5}$$

where $M$ is the delay length, and (4.5) is a low pass filter. Combining (4.4) and (4.5), we can obtain (4.6):

$$H(z) = \frac{z^{-M}}{1 - \dfrac{gz^{-M}}{1 - az^{-1}}} = \frac{z^{-M}(1 - az^{-1})}{1 - az^{-1} - gz^{-M}}. \tag{4.6}$$

Here four cascade all-pass filters are used to increase echo density and disperse the phase. Each all-pass filter has its own delay length $D_i$ and coefficient $a_i$. Hence the total transfer function will be

$$H(z) = \frac{-a_1 + z^{-D_1}}{1 - a_1 z^{-D_1}} \cdot \frac{-a_2 + z^{-D_2}}{1 - a_2 z^{-D_2}} \cdot \frac{-a_3 + z^{-D_3}}{1 - a_3 z^{-D3}} \cdot \frac{-a_4 + z^{-D_4}}{1 - a_4 z^{-D_4}} \tag{4.7}$$

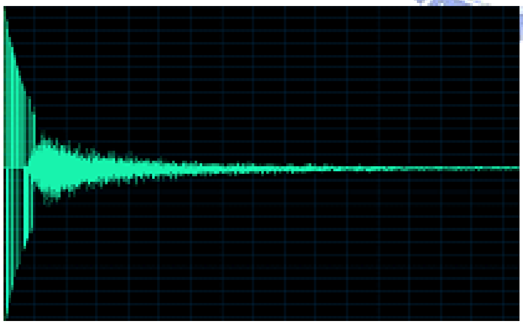
The algorithm is run on a single 80MHz (about 80MIPS) where each instruction cycle is 12.5ns. The original and processed sound is stored in the external RAM. For each filter, 2500 memory locations are used as a spatial buffer. The parameters of four comb filters and four all-pass filters are listed in Table 4-5 and Table 4-6, respectively. Simulated waveforms are shown in Fig. 4-3.

Table 4-5. Allpass filter coefficients.

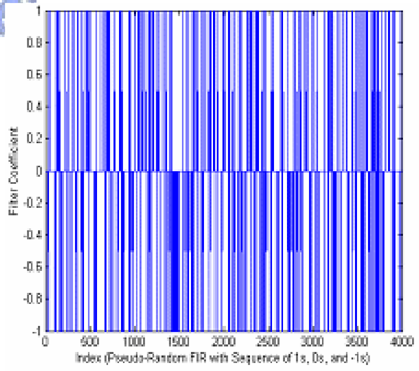
| Parameter / Filter | $D_i$ | $a_i$ |
|---|---|---|
| Allpass-1 | 22 | 0.45 |
| Allpass-2 | 36 | 0.45 |
| Allpass-3 | 23 | 0.45 |
| Allpass-4 | 33 | 0.45 |

Table 4-6. Comb filter coefficients.

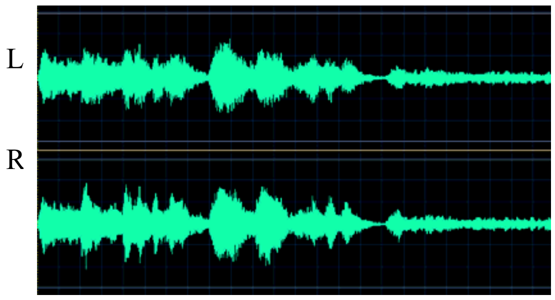
| Filter / Parameter | Comb-1 | Comb-2 | Comb-3 | Comb-4 |
|---|---|---|---|---|
| $a$ | 0.25 | 0.27 | 0.28 | 0.29 |
| g | 0.7 | 0.680 | 0.674 | 0.654 |
| $m$ | 37 | 40 | 41 | 43 |



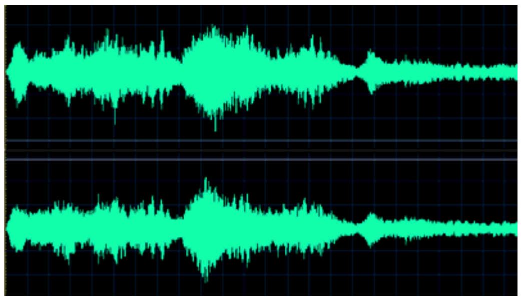
(a)



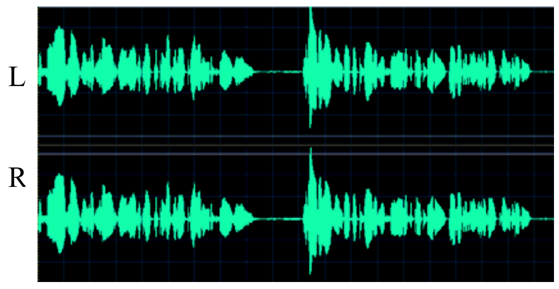
(b)



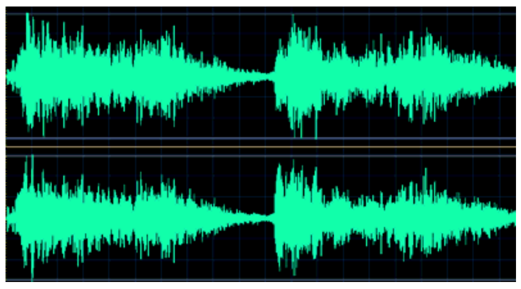
(c)



(d)



(e)



(f)

Fig. 4-3. The original and resulting waveforms after the reverberation algorithm: (a) is a simulated impulse response with early reflection in FIR; (b) is FIR coefficients using a pseudo random method; (c) and (e) are original audio music and female speech with 44.1 kHz sampling rate and 16-bit data format; (d) and (f) are the signals after processing (c) and (e).

### 4.2.2 Implementation of Application-Specific Reverberator

The multi-tap FIR filter constructed as two-stage pipeline architecture for audio reverberation applications is designed in HDL and C simulation. It consists of pipeline registers, two circular buffers, 16-bit carry look-ahead adders, shifters, and the fast state machine controller. Due to pseudo-random coefficients (existence of many zero values) based on (2.4), the executing time and computational consumption of FIR is reduced. Fig. 4-4 shows the results of desire and HDL FIR over 1,000 FIR orders. These two results are quite similar, but exist on 2% inaccuracy at the location of the 800th samples. This is because of the effect truncation errors.

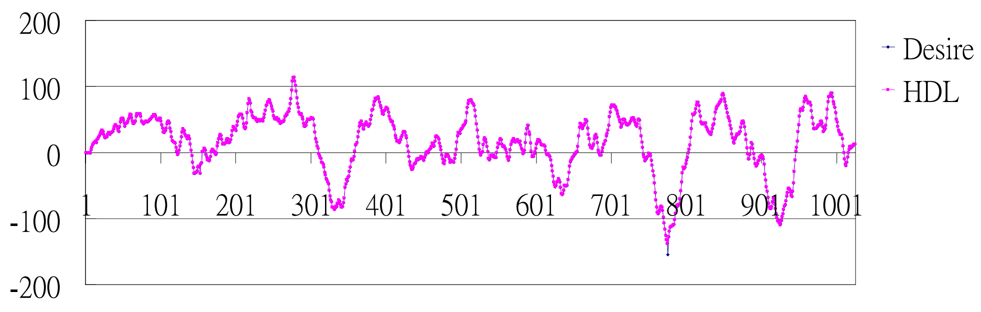


Fig. 4-4. Fully FSM control flows for two-stage architecture.

A given music as input sources via the I$^2$S interface is fed into the spatial circular buffer. After the FIR processing, the results are shown in Fig. 4-5. The circular buffer is set to 2,500 blocks. The test is to process single channel, 20,282 samples of input, which is about 0.5 ms of samples with 44,100 Hz sampling rate and 16-bit data width. The desire result shown in Fig. 4-5(a) is similar to the result of HDL simulation shown in Fig. 4-5(b).

Table 4-7 shows the comparison of different FIR schemes for implementation of early reflection. The number of adders, multipliers, and shifters and delay latency is estimated and compared. The different FIR style includes in terms of Direct Form (DF), Distributed Arithmetic (DA) [38], Canonic Sign Digit (CSD) [39], Digital Signal Processor (DSP) [40], and our proposed method. The delay latency is defined as the output of the first data. As can be seen in Fig. 4-5, the proposed method can greatly save multiplication power. Most of MAC instruction in DSP needs two or higher clock cycles to accomplish operations. Although DA and CSD do not need any multiplier, their delay latency is more than 1 stage due to bit and table operations. For the proposed two-stage FIR design, the number of adders and shifters is reduced to be 1/2 orders for each stage, and it is suitable for audio reverberation.

Table 4-7. Comparison of different FIR schemes for early reflection implementation.

| Schemes / Items | DF (TDF) | DA [38] | CSD [39] | DSP [40] | Proposed |
|---|---|---|---|---|---|
| Adder | Order | Order/2 | 2*Order | Order | Order/2 |
| Multiplier | Order | None | None | Order | None |
| Shift | None | Order/2 | Order/6 | None | Order/2 |
| Delay latency | None | 16 | 32 | None | 1 |

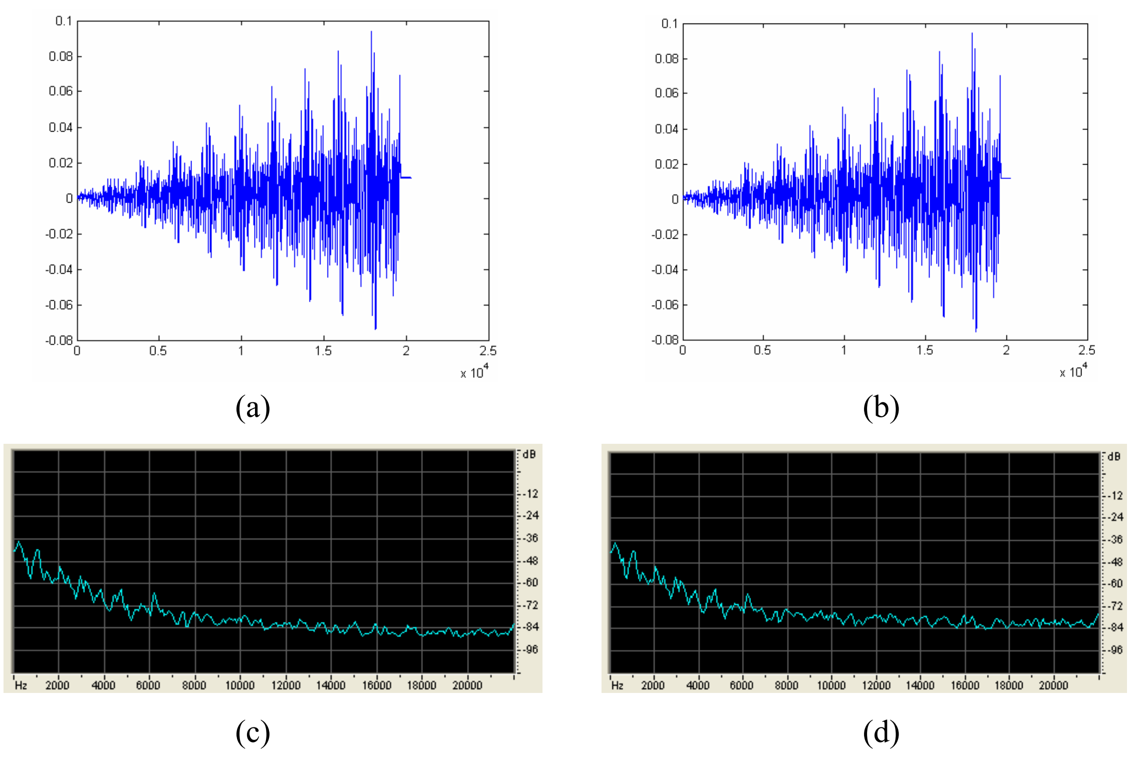


Fig. 4-5. Sound with 2,0282 digital samples after FIR processing: (a) Desire results and (b) design results. (c) and (d) are the results of frequency domain analysis with Hamming window for (a) and (b).

For multi-tap filter implementation, parallel architecture and random coefficients are not only computation reduction, but also can save multiplication power. At the same time, the circular buffer can effectively be used as a spatial size. Thus, the proposed two-stage architecture can be effectively used in FIR filter hardware implementation for the audio reverberator system. In the future, the adaptive pseudo-random FIR coefficient generator can be implemented by hardware according to the feature parameters of non-zero filter taps, sampling rate, and time variance.

## 4.3 Performance Analysis of LASP24

Complexity is measured using million instructions per second (MIPS), random access memory (RAM) and read only memory (ROM) measurements. MIPS are measured using the execution time and instruction counts. Linker memory maps are obtained with required sizes. As Table 4-8 shows MELP complexity exceeds LPC and CELP in both processor and memory requirements. Additionally, the total performing cycles is listed for MELP, CELP (TI DSP [84]), and reverberation algorithms.

Now LASP24 can perform the two practice applications in real time. We analyze the performance between them. For the MELP coder, the program performs 1,338,280 cycles in 60 MHz. The frame size is 22.5ms (180 samples) with a sampling frequency of 8000 Hz. Hence, the latency is about 21 ms (1,338,280×16.67 ns) for the encoder. As the result for the decoder, the latency is about 9.1ms. Due to many filters used in the reverberation algorithm, the required execution time is larger. The program performs 1,574,430 cycles at 80 MHz. The frame size is 22.7 us (stereo channels) with a sampling frequency of 44,100 Hz. The latency is about 19.67 us. Anyway, LASP24 can operate max frequency at 100 MHz. By the above analysis, it is able to satisfy all conditions with operating frequency 80MHz.

Table 4-8. Complexity comparison between LASP24 and memory with optimization codes.

| Items / DSP Algorithm | MIPS | RAM | ROM | Total Cycles |
|---|---|---|---|---|
| | | Unit: byte | | |
| MELP Decoder | 40 | 96K | 10K | 546449 |
| MELP Encoder | 60 | 96K | 26K | 1338280 |
| CELP Decoder (TI 320C3X) | 30 | 14.8K | 128K | 364299 |
| Reverberation | 80 | 96K | 30K | 1574430 |

# CHAPTER 5

# THE INTEGRATED PLATFORM FOR MULTIMEDIA PROCESSING

## 5.1 Introduction

Today, the VLSI growing gap between the silicon gate capacity and the engineering productivity has lead to the advance of System-on-Chip (SoC) designs and the need for new forms of design reuse and methodologies [50]. With the rapid progress of semiconductors, SoC is very popular recently. Reuse is done at the chip level called Virtual Component (VC) or intellectual property (IP), which represents functions of specification domains like DSP or multimedia modules. In order to connect each IP on SoC, the standardized bus is indispensable [55].

Several bus protocols enjoying a certain degree of popularity are currently used in SoC design. IBM's CoreConnect [51] is supported by a vast set of tools that allow the automatic generation of many parts of the system. The Wishbone specification [52] offers a set of guidelines for a basic, simple bus structure. This protocol has been selected by OpenCores organization [53] as the standard based on [52] to follow for the development of the free IP library. Advanced RISC Machines Inc. (ARM) developed the very popular AMBA AHB/APB protocol [54] and it has been used in many products. This protocol is also adopted in this thesis to develop the SoC platform and audio IPs.

A single proposed SoC platform [80] which combines microprocessor, memory, and other functional modules such as GPIO (General-Purpose Input/Output), $I^2S$ (Inter-IC

Sound), and communication (UART) into a single IC is popular recently. To verify these functions [73] of the proposed platform for audio processing, the FPGA rapid prototype approach is used. FPGA were primarily used for prototyping and lower volume applications in years and custom ASICs were used for high volume, cost-sensitive designs. Today, state-of-the-art wafer fabrication finds that FPGAs are an excellent mechanism for testing new wafer technology because of their reprogrammable and high volume natures. Hence, more and more designers use platform FPGA technology [56] to develop and verify their SoC design quickly. In this thesis, we offer a way to develop low cost SoC products within the FPGA environment for fast design and verification, and the SoC integration platform for digital audio applications as reverberation and speech processing has also been presented in Chapter 2 for demonstration.

## 5.2 SoC Platform

The proposed SoC platform is shown in Fig. 5-1 for speech/audio processing. The primitive prototype is constructed in two platforms: FPGA Integration Platform (FIP) using Altera Cyclone Edition and DSP Verification Platform (DSPVP) using the Analog Devices DSP KIT [59] evaluation system for Blackfin embedded media processors [76].

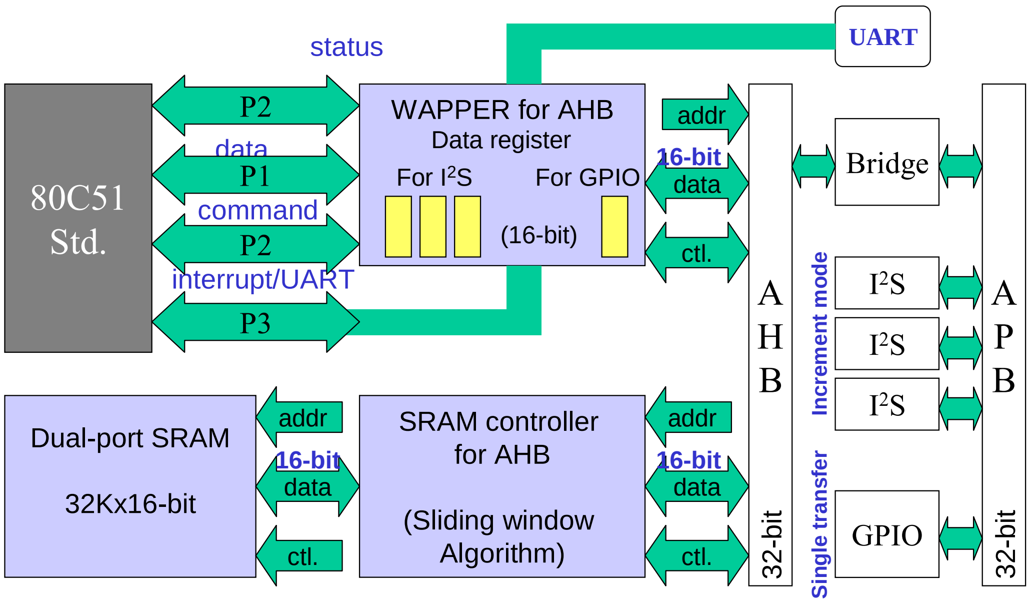
FIP has a full implementation of AMBA AHB and APB on-chip buses. A flexible configuration scheme makes it simple to add new IP cores. Also, all provided peripheral units implement the AMBA AHB/APB interface making it easy to add more of them, or reuse them on other components using AMBA. In the SoC platform, the 8051 microcontroller is used as a main resource dispenser. Due to timing request, the 8051 microcontroller can not directly connect to AHB. It is necessary to be packed by a wrapper. Likewise, the wrapper is required if a DSP processor is added to AHB. In the case, the DSP wrapper is not shown due to the use of the DSP evaluation system. An on-chip memory called dual-port synchronous static RAM (SSRAM) is embedded into the system. The SSRAM, which store multi-channel audio streaming, is defined as a share buffer between FPGA and DSP. Four peripheral devices as GPIO (General-Purpose I/O), I$^2$S (Inter-IC Sound), Interrupt Controller, and UART (Universal Asynchronous Receiver Transmitter) are hanged on APB. These modules are listed in Table 5-1 and explained in detail in the next section.

Table 5-1. Module design of the SoC platform.

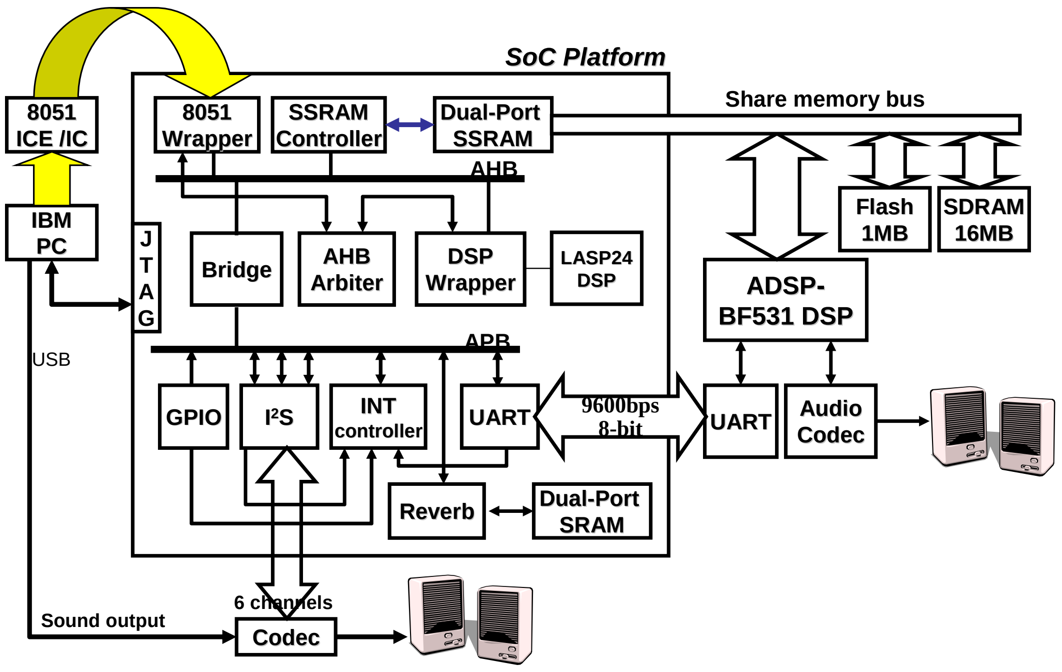
| Module Design/BUS | Technology Description |
|---|---|
| Dual-port SSRAM Controller (AHB) | Dual port synchronous static RAM: offer general-purpose memory accessing interface. |
| 8051 wrapper (AHB) | Offer the conversion of 8051 signals into AHB. It is an interface and buffer devices. In the platform, it is a main control center, i.e., Master. |
| UART (APB) | A device, usually an integrated circuit chip, which performs the parallel-to-serial conversion of digital data to be transmitted and the serial-to-parallel conversion of digital data that has been transmitted. |
| GPIO (APB) | It can be individually configured through software as either an input or output, and provide additional control and monitoring when the microcontroller or chipset has insufficient I/O ports, or in systems where serial communication and control from a remote location is advantageous. In this platform, GPIO provides a little as 4 ports and up to 24 ports. |
| I2S (APB) | It provides digital sound processing interface, which is serial communication to connect digital sound devices. In the peripheral bus, if processing multi-channel sound, we can put one or more I$^2$S groups. |

| Module Design/BUS | Technology Description |
|---|---|
| Interrupt Controller (APB) | The interrupt allows edge or level triggers to handle interrupt routines with the programmable method. It provides 8-bit or 16-bit data width for fast and general interrupt as input. |
| APB Bridge (between AHB and APB) | Handle signal transformation between high-speed bus and low-speed bus. The bridge can maintain devices between two buses at the same time. |
| AHB Decoder (AHB) | Offer interconnection and mechanism that uses the bus between master and slave devices in AHB, i.e., bus arbiter. |
| DSP (LASP24) | 24-bit floating-point digital signal processor described in Chapter 3. |

DSPVP performs given 3-D audio algorithms such as reverberation in real-time, but these two audio algorithms are not shown in this thesis. Audio streaming via SSRAM is fed into DSP. Processed audio streaming is exported to stereo speakers via the audio codec. In the developing process, DSPVP is an auxiliary platform to cooperate with the verification of the designed SoC system.



(a)



(b)

Fig. 5-1. Multimedia SoC platform: (a) SoC architecture and (b) the proposed prototype system.

## 5.3 Intellectual Property Design

### 5.3.1 Microprocessor

The microprocessor is compatible with the MCS-51 family, originally designed in the 1980's by Intel. The processor has gained great popularity since its introduction and is estimated it is used in a large percentage of all embedded system products. It features are 8-bit CPU, on-chip memory which has separated Data and Program (read-only) memory, two 16-bit timer/counters and four 8-bit I/O ports including two interrupts. There are 64K bytes of off-chip program memory and up to 4K bytes of on-chip program memory. Remaining part of the program memory is external and can be reached with a specific signal EA. The some features of the 8051 IP core referred to Opencores [53] are described as follows.

- 8-bit CPU optimized for control applications
- Extensive Boolean processing (single-bit logic) capabilities
- 64K program and data memory address space
- 32 bidirectional and individually addressable I/O lines
- 6-source/5-vector interrupt structure with two priority levels
- Up to 4K bytes of on-chip program memory
- Two 16-bit timer/counters

The instruction set of the 8051 core is already said optimized for 8-bit control applications. This optimization shows in a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set is also good for systems that require a lot of Boolean processing because it has an extensive support for one-bit variables as a separate data type (that makes direct bit manipulation a lot easier). The total of addressing modes is five kinds, which include direct, indirect, register, register-specific, immediate, and index addressing.

The 8051 core contains four I/O ports. All four ports in the 8051 core are bidirectional. Each port has SFR (Special Function Registers P0 through P3) which works like a latch, an output driver and an input buffer. Both the output driver and the input buffer of Port 0, and the output driver of Ports 2 are used for accessing the external memory. It works like this: Port 0 outputs the low byte of the external memory address (which is time-multiplexed with the byte being written or read) and Port 2 outputs the high-byte of the external memory address (this is only needed when the address is 16 bits wide). If the address in question is 8 bits wide the Port 2 pins are not needed in this application. The Port 3 pins are multifunctional. Their alternate functions are listed in Table 5-2. The alternate functions are activated with the 1 written in the corresponding bit latch in the port SFR.

Table 5-2. Microprocessor's alternate functions.

| P3 Port Pin | Alternate Function |
|---|---|
| PIN 2 | $\overline{\text{INT0}}$ (external interrupt) |
| PIN 3 | $\overline{\text{INT1}}$ (external interrupt) |
| PIN 4 | T0 (timer/counter 0 external input) |
| PIN 5 | T1 (timer/counter 1 external input) |
| PIN 6 | $\overline{\text{WR}}$ (external data memory write strobe) |
| PIN 7 | $\overline{\text{RD}}$ (external data memory read strobe) |

The new value arrives at the latch during the last phase (Phase 2), of the final cycle of the instruction that changes the value in a port latch. Because the port latches are sampled by their output buffers only during Phase 1 of any clock period (during Phase 2 the output buffer holds the value it saw during the previous Phase 1), the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at the beginning of the following machine cycle.

## 5.3.2 Inter-IC Sound Interface

The $I^2S$ is used only to handle audio serial data. To minimize the number of pins required and to keep wiring simple, a 3-line serial bus consisting of a line for two time-multiplexed data channels, a word select line (WS), and a clock line (SCK) is used.

Serial data (SD) is transmitted in two's complement with the MSB first. A simple configuration and the basic interface timing are illustrated as Fig. 5-2. The MSB is transmitted first because the transmitter and receiver may have different word lengths. The WS indicates the channel being transmitted: when WS=0, SD belongs to the left channel; conversely, SD belongs to the right channel. The WS line changes one clock period before the MSB is transmitted. This allows the slave transmitter to derive synchronous timing of the serial data that will be set up for transmission. Serial data sent by the transmitter may be synchronized with either the trailing (high-to-low) or the leading (low-to-high) edge of the clock signal. However, the serial data must be latched into the receiver on the leading edge of the serial clock signal, and so there are some restrictions when transmitting data that is synchronized with the leading edge.

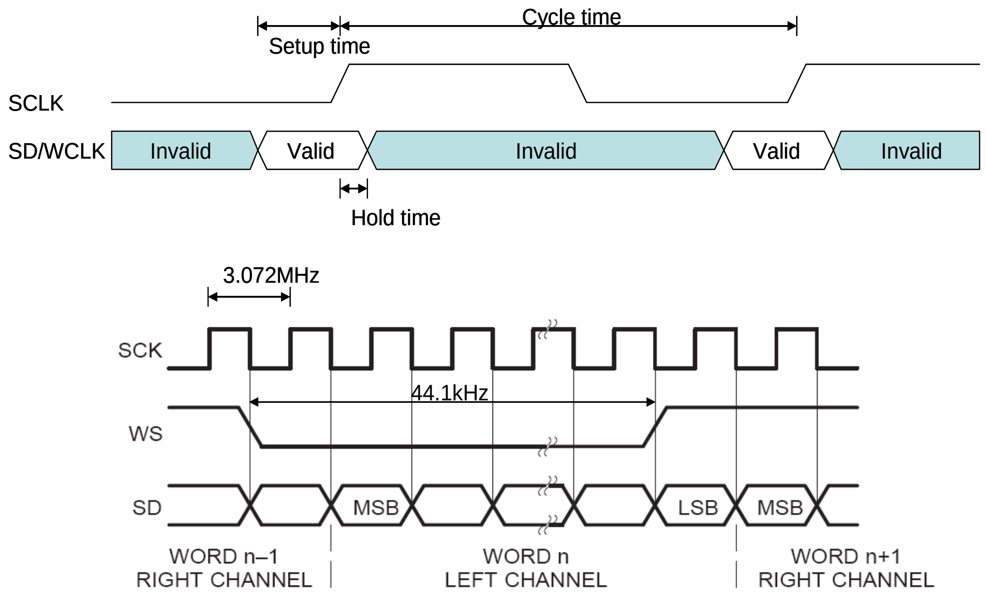
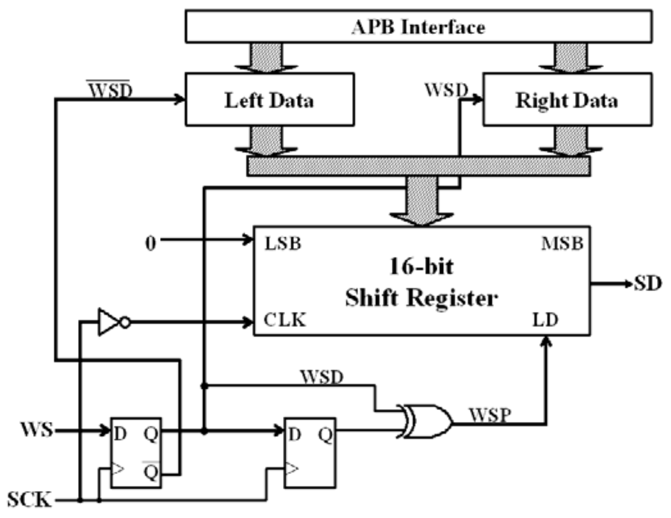
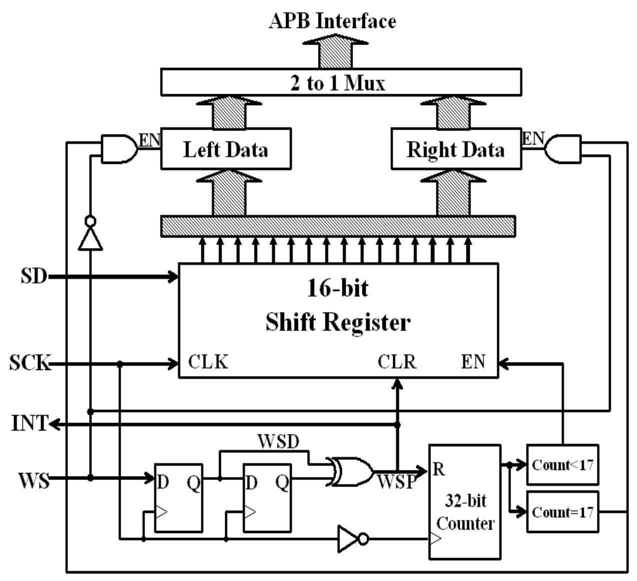


Fig. 5-2. The basic interface timing of $I^2S$.

The hardware configuration of $I^2S$ transmitter and receiver are shown in Fig. 5-3. At each WS-level change, a pulse WSP is derived for synchronously parallel-loading the shift register. For the transmitter, the output of one of the data latches is then enabled depending on the WS signal. Since the serial data input is zero, all the bits after the LSB will also be zero. For the receiver, following the first WS-level change, WSP will reset the counter on the falling edge of SCK. As the counter increases by one every clock pulse, subsequent data bits are latched into the 16-bit shift register.



(a) Transmitter



(b) Receiver

Fig. 5-3. The block of audio $I^2S$ configuration ($SCK=64\times f_s$ and $WS=f_s=48kHz$).

In the $I^2S$ format, any device can act as the system master by providing the necessary clock signals. A slave will usually derive its internal clock signal from an external clock input. This means, taking into account the propagation delays between master clocks and the data and/or word-select signals, that the total delay is simply the sum of the delay between the external (master) clock and the slave's internal clock; and the delay between the internal clock and the data and/or word-select signals. For data and word-select inputs, the external to internal clock delay is of no consequence because it only lengthens the effective set-up time (see Fig. 5-2). The major part of the time margin is to accommodate the difference between the propagation delay of the transmitter, and the time required to set up the receiver. All timing requirements are specified relative to the clock period or to the minimum allowed clock period of a device. This means that higher data rates can be used in the future. Fig. 5-4 shows the operation of audio data transmission from the I$^2$S interface to the internal system high-speed bus (indicated by arrowheads). Then audio data is stored into the share memory.

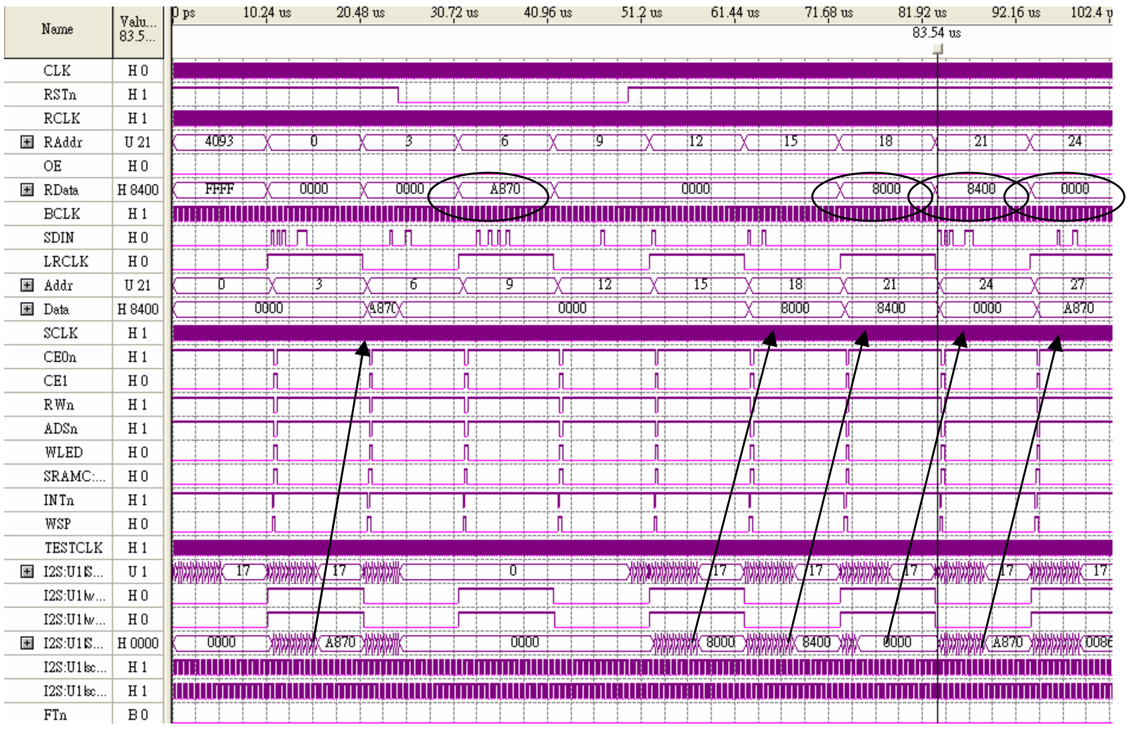


Fig. 5-4. FPGA simulation of I$^2$S transmission.

## 5.3.3 Serial Communication Design

UART (Universal Asynchronous Receiver/Transmitter) is designed to make an interface between a RS-232 line and an AMBA bus. It works fine connected to the standard serial port of any device for data exchange with custom electronic. It was built in the perspective to be very small, but efficient. It has to fit in a small FPGA. It is not suited to interface a modem since there is no control handshaking (CTS/RTS). It integrates two separate clocks, one for AMBA bus and the other for bitstream generation. This has the advantage to let the user bring own desired frequency for the baud rate. The baud rate, however, is defined as 9,600 bps in the case.

The core implements the AMBA SoC bus interface for communication with the platform. It has an 8-bit data bus, even parity, and 1 stop bit for compatibility reason. The core requires one interrupt. It requires 2 pins in the chip (serial in RX and serial out TX). The block diagram of the core is shown in Fig. 5-5. The line control register assigns one of operations between the transmitter and receiver. If the received operation is active, serial data (RX) is fed into the receiver shift register. When the action is finished, the receiver logic will send an interrupt signal to the microprocessor. Conversely, if the transmitted operation is active, 8-bit data is fed into TX from the transmitter shift register.

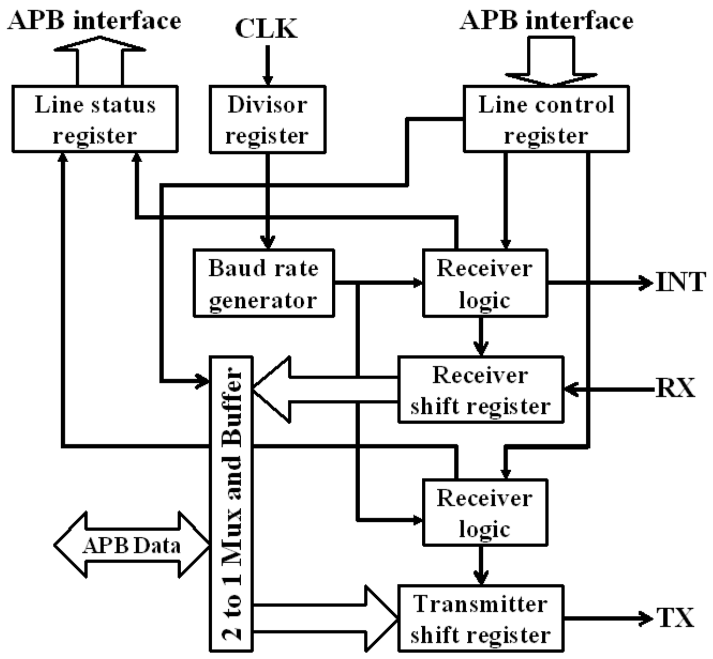
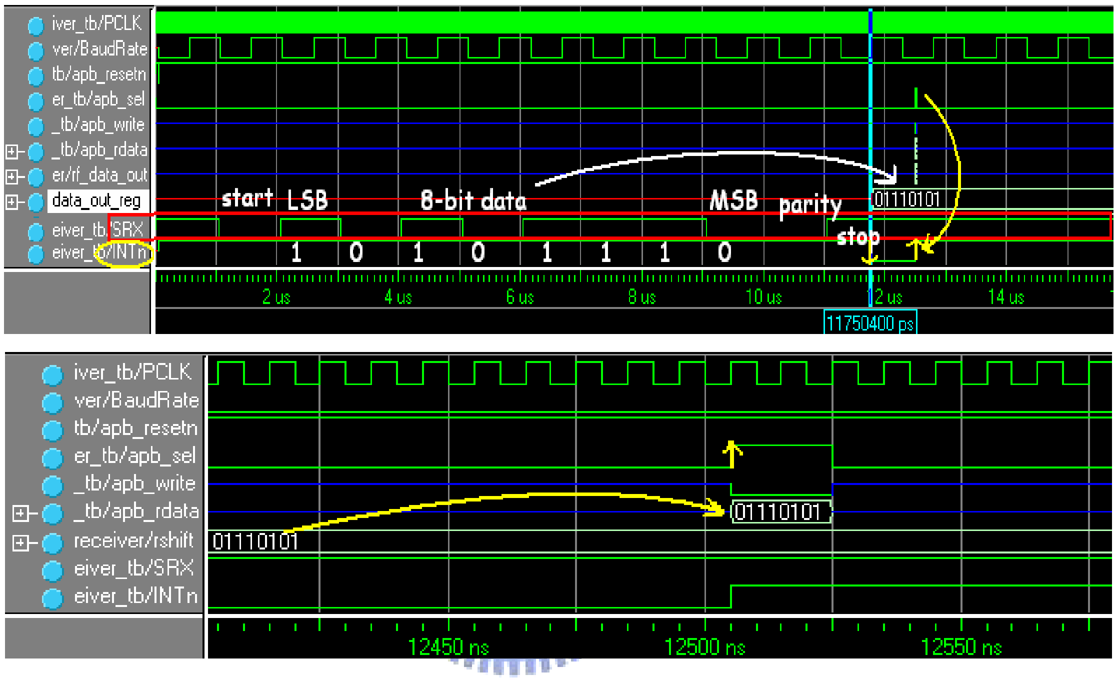
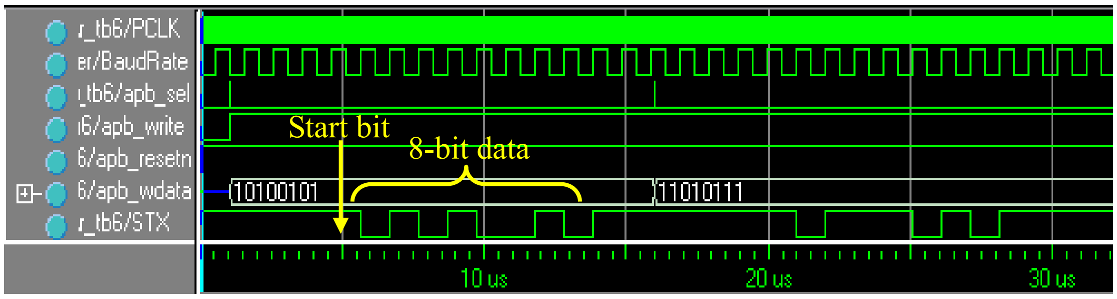


Fig. 5-5. The block diagram of UART (Baud rate at 9,600 b/s).

The UART simulation is shown in Fig. 5-6. For the receiver (Fig. 5-6(a)), after SRX has received 8-bit data, data is then stored into the data_out_reg register. This monmentm, an interrupt is triggered by INTn in order to info the master device. Hence, the master device can obtain data from the peripherial bus. Note that when the bus is selected (apb_sel=1), the interrupt signal INTn has to be disabled. The received data is stored into apb_rdata. For the transmittor (Fig. 5-6(b)), it is very easy. STX first sends a start bit (=0), then sends 8-bit data from LSB to MSB in order. Follow the parity and stop bits.



(a)



(b)

Fig. 5-6. FPGA simulation of UART (a) receiver and (b) transmitter.

### 5.3.4 Wrapper and Interrupt Design

Since 8051 I/O signals can not directly meet AHB timing constraints, it is necessary for the 8051 wrapper design. The interface is prepared for 8-bit accesses. In each read or write access to the AMBA AHB will not require wait state cycles. Thus, all AMBA AHB single transfer modes are supported in the 8051 wrapper design. The simulation of single transfer mode is shown in Fig. 5-7.

Fig. 5-8 shows the complete state machine for the 8051 wrapper design. The structure of the state machine is divided into three main parts: I$^2$S data processing, user-programmable inputs via GPIO registers, and UART communication. When an interrupt occurs, 8051 can handle corresponding procedures via the wrapper. Note that the wrapper spends most of the term in I$^2$S data communication. In other words, each time the wrapper must access three-channel data when the sampling rate starts.

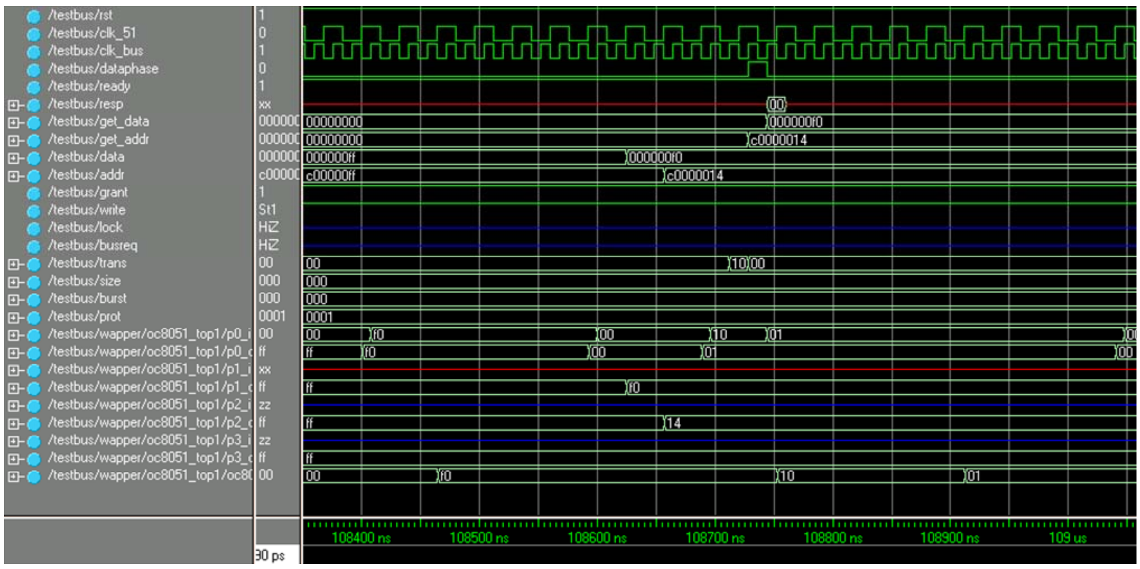


Fig. 5-7. FPGA simulation of data transfer for 8051 wrapper.

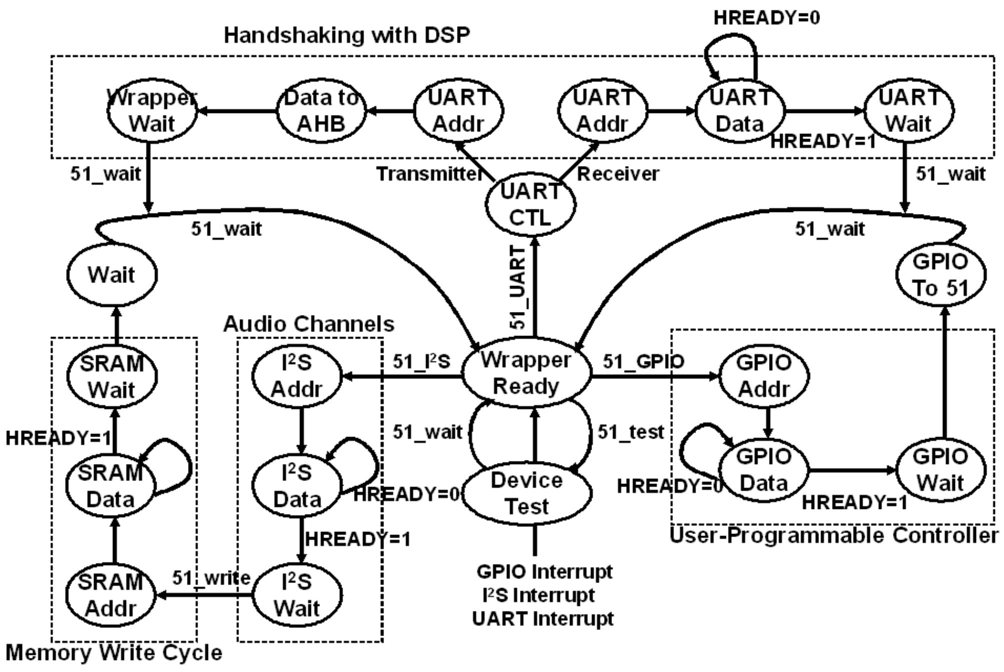


Fig. 5-8. Complete stat machine for 8051 wrapper.

Due to 8051 accepted only two interrupts, in order to handle more interrupts for the system, the interrupt controller can decide the priority of all interrupts. Of course, GPIO has the highest priority. The following is UART and I$^2$S, respectively. The interrupt controller supports up to 16 interrupts: 3 interrupts from the internal APB devices and other reserves.

### 5.3.5 Specialized Hardware for System Verification

To realize the benefits of emulation, virtually all of the circuit and testbench for the design must run on the emulator. This means that the testbench should be synthesizable. One approach would be to make the testbench synthesizable from the beginning, and to use the same testbench for both RTL verification and emulation. The bus functional models (BFM) used in the SoC platform are common method of creating testbenches. Typically they are written in the register-transistor level (RTL), a testbench automation tool, or in C/C++, and use some form of command language to create sequences of transactions on the system bus. The intent of the methods is to model only the bus transactions of an agent on the bus. They do not model any of the functionality of an agent on the bus; each read and write transaction is specified by the test developer explicitly. Because of their simplicity, these bus models place little demand on simulator performance; simulation speeds are mostly determined by the macro itself.

Many testbemchs require multiple BFMs, as in the SoC platform above. In this case, it is best to use a single command file to coordinate the actions of the various models as AHB and APB. The models must be written so that they can share a common command file. Many commercial BFMs offer this capability. If we take our canonical design, the following approach seems reasonable. In Fig. 5-9, the software for the processor is compiled and loaded into memory in the emulator. This allows the processor and peripherals to perform at full emulation speed. The stimulus for the data transformation block is also loaded into memory on the emulator. We can store a bit stream that represents audio data through the $I^2S$ interface. A simple state machine transfers data from the stimulus memory to the I/O interface. Similarly, the serial data from the output of the SoC platform is sent to a response capture memory in the emulator. Another simple state machine handles the handshake for the data transfer.

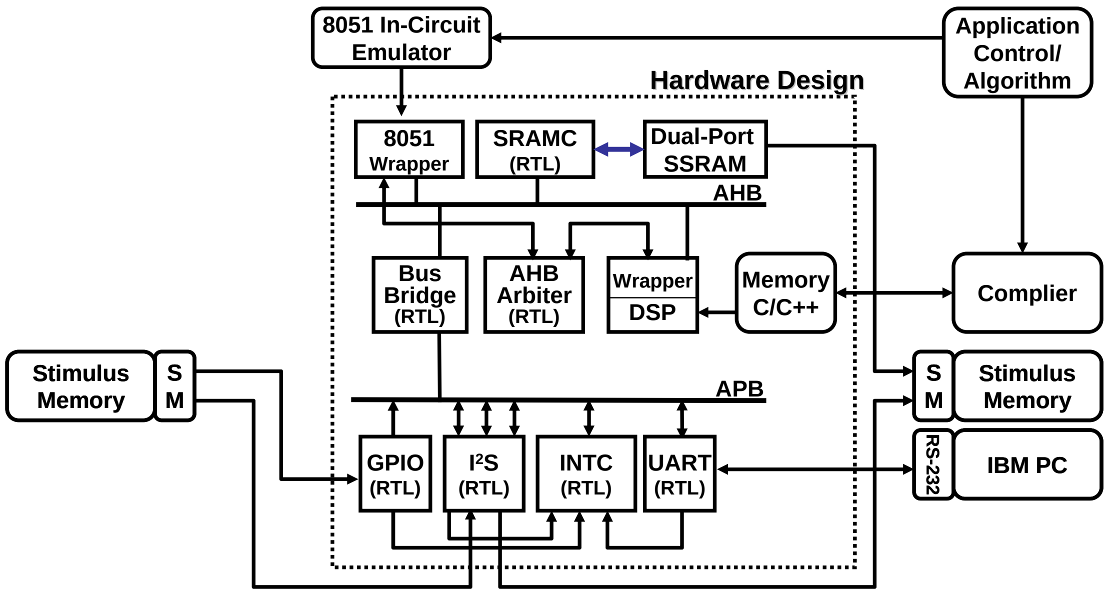


Fig. 5-9. Emulation testbench.

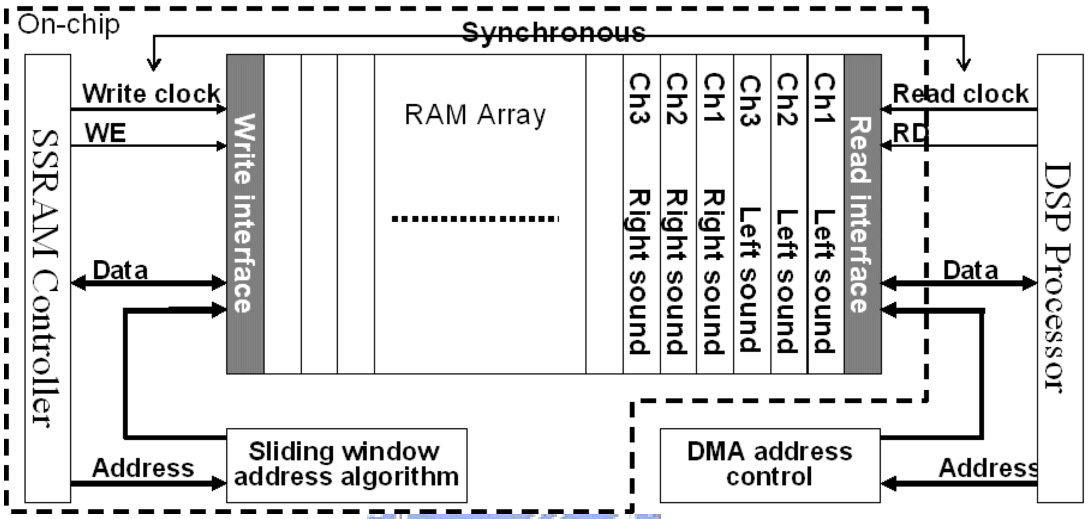
## 5.4 System Prototype

The proposed SoC platform design for audio applications has been synthesized on the Altera FPGA. In particular, the EP1C20F400 Cyclone FPGA has been employed to carry out all the synthesis and place-and-route processes. The synthesis results are shown in Table 5-3. By using Altera QuartusII, it is mapped, placed, and routed in less than a minute. The basic system occupies 1 block RAM about 65536 bits, 837 logic cells, 470 registers, 452 LUTs (look-up table), and 100 available pins. The SoC platform consumes just 5% of the logic resources, one of the smallest devices in that product family. LASP24 and reverberator consume 618 registers and 3,796 LEs. These two IPs can be performed on 100MHz frequency.

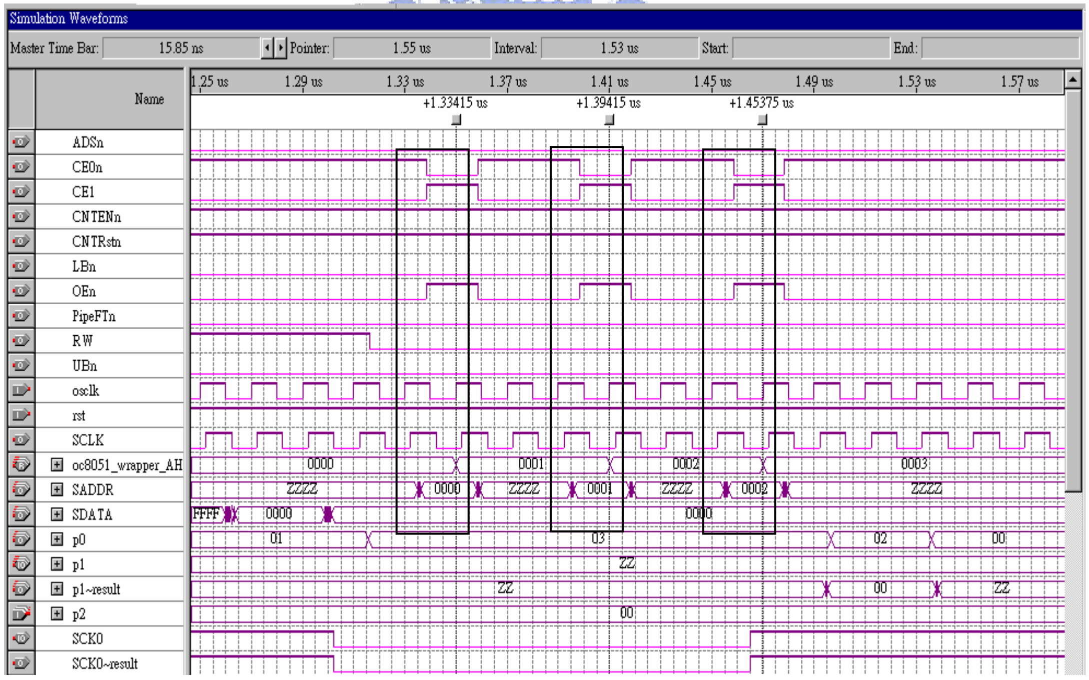
Table 5-3. The FPGA synthesis results of audio SoC design.

| Results / Components | FSM (states) | Registers | Sizes (LEs) | Operating frequency |
|---|---|---|---|---|
| 8051 wrapper (AHB) | 40 | 125 | 322 | 40 MHz above |
| $I^2S$ (APB) ×3 | - | 231 | 264 | 3.072 MHz |
| GPIO (APB) | 3 | 29 | 97 | 20 MHz |
| UART (APB) | 10 | 53 | 92 | Fixed baud rate: 9.6k bps |
| SSRAM controller (AHB) | 4 | 15 | 29 | 40MHz for read and write operations |
| Interrupt controller (APB) | - | 8 | 17 | 20MHz |
| Clock generator for $I^2S$ design (Independent module) | - | 9 | 16 | Input clock: 18.432 MHz<br>Bit clock: 3.072 MHz<br>Sampling rate: 48 KHz |
| LASP24 | 17 | 1,835 | 22,747 | Up to 100MHz |
| Reverberator (APB) | 25 | 287 | 7,049 | Up to 100MHz |

The operating frequency of the AMBA system and 8051 is at 40 MHz and 12 MHz, respectively. The 8051 transducer (1251 cycles) allows fetching the wrapper data and providing these data to each component in AMBA. In other words, it executes the 8051 simple handshake enable. The dual-port memory contains 4096 with 16-bit width 6-channel audio streaming data, and the data format is shown in Fig. 5-10(a). The sync signal is used as an interrupt to trigger DSP. It is important for the synchronous problem of audio data. When the microprocessor receives audio data via an interrupt, data is then written into the dual-port SRAM shown in Fig. 5-10(b) by using Fig. 5-10(a) format.



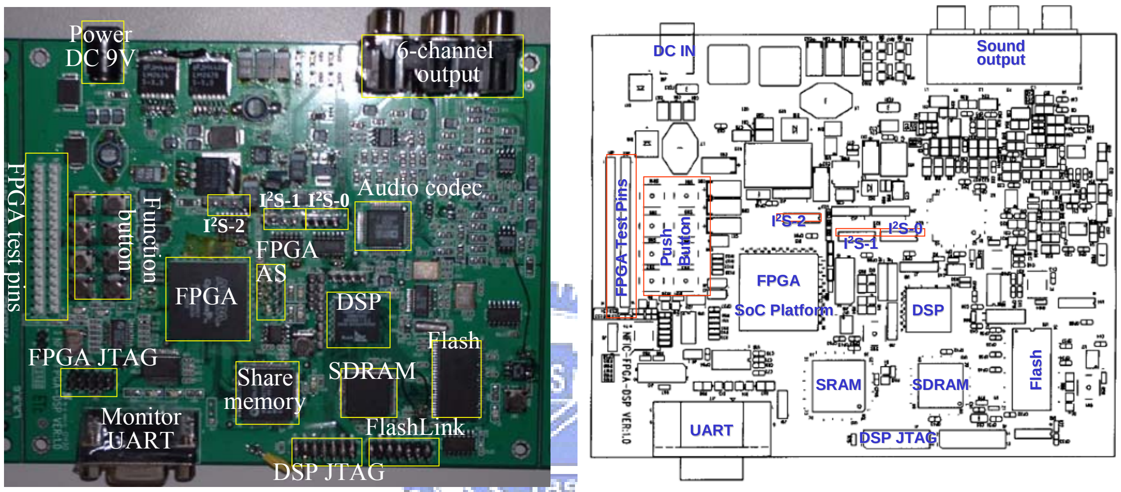
(a)



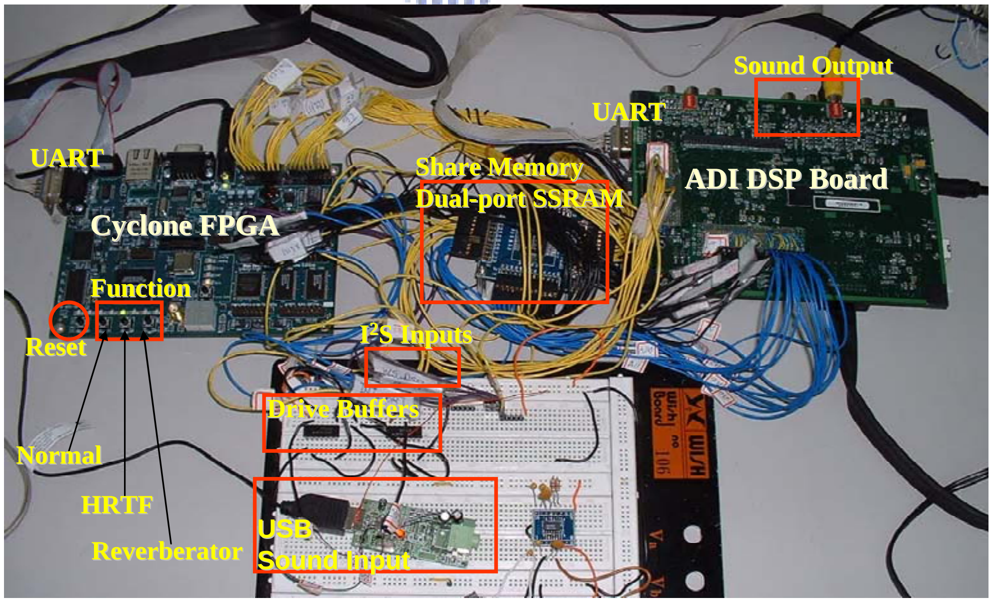
(b)

Fig. 5-10. Share memory: (a) The data format and control of audio streaming and (b) simulation results of processing three-channel data in the dual-port SRAM.

As a result, we can be able to test quickly the prototyping system (Fig. 5-1) in the development kit environment. The SoC system can perform with accuracy and control DSP operations of optimized audio algorithms as well as high-quality sound in real-time. The development environment and the final demo board for audio processing are shown in Fig. 5-11.



(a)



**(b)**

Fig. 5-11. (a) The final Demo board and (b) the initial development environment.

# CHAPTER 6

# CONCLUSIONS AND FUTURE WORKS

This thesis has proposed a new low-cost application-specific processor (LASP24) for sound processing in multimedia. High performance is achieved by vector and matrix operations that are not usually supported by general-purpose DSPs. The LASP24 has precise 24-bit floating-point arithmetic units. This processor makes the speech coding algorithms ready to run in real-time operations. In addition to the LPC calculation and pitch estimation built-in to the LASP24 core, the other algorithms such as a codebook search can also be implemented in the designed processor. Furthermore, a technique of gated clocks on power optimization of sequential circuits was involved in this design to reduce power dissipation. Based on these features, LASP24 can share huge calculations in real-time speech coding. It can also reduce power consumption: 25.75% at 100MHz, 12.75 at 80MHz, 31.1% at 40 MHz, 27.6% at 33 MHz, and 62.75% at 25 MHz. The performance of LASP24 was about 4.75 times higher than TMS320C30 and about 3.2 times higher than TMS320C31. Several experimental tests have been done, and the performance comparisons to a series of TMS320C3x processors are also presented in the thesis. As these testing results, we can find that LASP24 for sound processing has a very satisfactory performance, and it has also verified all the designed functions.

Finally, a SoC platform design for digital sound processing by using FPGA development environment is presented. Through the 8051 embedded microcontroller, we can easily program two audio processing and completely control all actions of the audio system. The platform has been verified and performed in the Altera Cyclone FPGA, and it can control the DSP processor to execute speech coding such as MELP and audio enhancement such as Reverberation as well. The proposed SoC integration platform offers outstanding performance and flexibility at very low cost for a wide range of multi-channel audio applications. In the future, the DSP processor for audio processing will be developed such that it can be integrated into the proposed SoC platform design to perform a complete audio SoC system. Under FPGA verification and testing, on average the whole performance obtains 80MIPS and 90mW power consumption. Due to a cross-platform implemented method, it can be applies into an embedded and portable multimedia system and can also be integrated to a single silicon chip.

The modern day computing technology ought to be one supporting interactive and intelligent processing [74] that transforms and transfers information ubiquitously and in real-time speed. The future computing must provide both economic bandwidth utilization and efficient information extraction. More importantly, the industry must be prepared for the inevitable trend that (1) *computing* (2) *control* and (3) *users* will be separated by long distances. As a result, the users can anticipate the near-future convergence of computing and communication. A truly integrated media system must connect with individual users and content addressable multimedia databases. This new trend bring about a great technological challenge as

- Future multimedia technologies will need to handle information with an increasing level of intelligence, i.e., automatic extraction, recognition, interpretation, and interactions of multimodal signals, and the ability to seamlessly handle different representations. This will lead to what can be called intelligence multimedia processing technology, and integrated into the SoC platform.

- We envision a major impact by integrating adaptive neural processing into the state-of-the-arts multimedia technologies. The main power of neural networks hinges upon their adaptive learning capability [57], which enables machines to be taught to interpret possible variations of the same object or pattern, e.g., scale, orientation, and perspective.

- The system integration will also be a challenging task as it involves complex tradeoff in integrating subsystems into a functional SoC system. For example, we need to estimate the necessary storage space for application codes (such as adaptive on-the-fly incremental training). The objective is to have the total system implemented under the specified power, size, weight, and cost constraints.

In the future, General-purpose workstations and PCs are already equipped with powerful programmable microprocessors; these processors, however, have not been able to perform image and video processing tasks efficiently as the special algorithm characteristics are not exploited. Therefore, a special class of programmable multimedia processors [64] has evolved that incorporate architectural enhancements to increase their multimedia processing capabilities. These enhancements include as follows.

- Subword parallelism: A number of lower-precision data items are processed in parallel on the same ALU (split-ALU). This enables to exploit data parallelism in highly regular low-level algorithms involving identical operations executed on large data volumes.

- Very long instruction word (VLIW) [82]: Several operations are specified within a single instruction word for concurrent execution on multiple function units. Instruction level parallelism available in image and video algorithms can thus be exploited. Code scheduling has to be performed statically by the compiler.

- Coprocessor architecture: By incorporating one or more separate modules adapted to specific tasks, highly regular program parts with high processing requirements can be executed on dedicated hardware, while more irregular but less computationally intensive control tasks can be performed on a programmable processor core.

- Memory system design: Due to the high data volumes encountered particularly in video processing, the memory system has a significant impact on overall performance. Stream caches have been proposed that employ prefetching techniques to access shortly needed data in advance. Data structures of static nature, such as filter coefficients or look-up tables, can be placed into on-chip SRAMs where they are always accessible within shortest times. For instruction memory design, conventional cache strategies may prove useful for speeding up instruction access, provided the cache is large enough and mutual code replacement can effectively be prevented.

The architectural enhancements of current multimedia processors alone, targeting almost exclusively audio enhanced algorithms, will not be sufficient for the emerging multimedia applications. With increasing sophistication of multimedia algorithms and less predictable program flow, new concepts are required.

# BIBLIOGRAPHY

[1] S. Barua, "An interactive multimedia system on computer architecture, organization, and design," *IEEE Transactions on Education*, vol. 44, pp. 41-46, Feb. 2001.

[2] A. Werf, F. Briils, R. Kleihorst, and E. Waterlander, "McIC: A Single-Chip MPEG2 Video Encoder for Storage," *In Proceedings of the 1997 IEEE International Solid-State Circuits Conference* (*ISSCC 1997*), pp. 254-255, February 6-8, 1997.

[3] B. G. Haskell et.al., "Image and Video Coding-Emerging Standards and Beyond," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 8, no. 7, pp. 814-837, Nov. 1998.

[4] Y. Okada, T. Nakamoto, H. Gunji, and M. Hase, "An 80mm$^2$ MPEG2 Audio/Video Decode LSI," *In ISSCC,* pp. 264-265, February 6-8, 1997.

[5] M. Toyokura, M. Saishi, S. Kurohmaru, and K. Yamauchi, "A Video DSP with a Macroblock-Level-Pipeline and a SIMD Type Vector-Pipeline Architecture for MPEG2 CODEC," In *ISSCC,* pp. 74-75, February 16-18, 1994.

[6] G. Pachanek, C. Glossner, W. Lawless, and D. McCabe, "A machine organization and architecture for highly parallel, scalable, single chip DSPs." *In Proceedings of the 1995 D S F Technical Program,* pp. 42-50, May 15-18, 1995.

[7] G. Pachanek, M. Stojancic, S. Vassiliadis, and C. Glossner, "M.F.A.S.T.: A Single Chip Highly Parallel Image Processing Architecture," *in International Conference on Image Processing,* pp. 69-72, Oct. 1995.

[8] D. R. Begault, 3-D Sound for Virtual Reality and Multimedia, Academic Press, Inc., 1994.

[9] E. Torick, "Highlights in the history of multichannel sound," *J. Audio Eng. Soc.*, vol.46, Jan. 1998.

[10] C. Kyriakakis, "Fundamental and technological imitations of immersive audio systems," *Proceedings of the IEEE*, vol. 86, pp. 941-951, May 1998.

[11] M. R. Schroeder, "Natural sounding artificial reverberation," *J. Audio Eng. Soc.*, vol. 10, pp. 219, 1962.

[12] A. J. Moorer, "About this reverberation business," *Comput. Music J.*, vol. 3, pp. 13-28, 1979.

[13] P. Rubak, L.G. Johansen, "Artificial reverberation based on a pseudo-random impulse response," *the AES 106th Convention*, Munich, Germany, Paper No. 4900(G6), May 8-11, 1999.

[14] D. Carugo, "Development of a surround encoding algorithm," *Proceedings of the COST G-6 Conference on Digital Audio Effects* (DAFX-00), Verona, Italy, December 7-9, 2000.

[15] U. Zoler, Digital Audio Signal Processing, John Wiley & Sons, 1997, Ch6.

[16] M. J. D. Powell, Radial Basis Functions for Multivariate Interpolation, Edited by J. C. Mason, M. G. Cox, pp. 143-166, 1987.

[17] C. P. Brown and R. O. Duda, "A structural model for binaural sound synthesis," *IEEE Trans. on Speech and Audio Processing*, vol. 6, pp. 476-488, Sep. 1998.

[18] P. S. Ray, "Characteristic properties of some integer Toeplitz matrices," *in Proceedings of Information, Decision and Control* (*IDC 99*), pp. 177-179, Feb. 1999.

[19] F. L. Wightman and D. J. Kistler, "Headphone simulation of free field listening- I: Stimulus synthesis," *J. Acoust. Soc. Am.*, vol. 85, pp. 858-867, 1989.

[20] F. L. Wightman and D. J. Kistler, "Headphone simulation of free field listening- II: Verification," *J. Acoust. Soc. Am.*, vol. 85, pp. 868-878, 1989.

[21] E. A. G. Shaw, "Transformation of sound pressure level from the free field to the eardrum in the horizontal plane," *J. Acoust. Soc. Am.*, vol. 56, pp. 1848-1861, 1975.

[22] K. J. Arnold, On Spherical Probability Distributions, Ph.D. Thesis, Massachusetts Institute of Technology, 1941.

[23] N. I. Fisher, T. Lewis, and B. J. J. Embleton, Statistical Analysis of Spherical Data, Cambridge University Press, Cambridge, 1987.

[24] K. C. Chen, Multi-Band Room Effect Emulator for 5.1 Channel Sound, Master thesis, Department of Electrical and Control Engineering, National Chiao Tung University, June 2001.

[25] R. L. Jenison, R. A. Reale, J. E. Hind, and J. F. Brugge, "Modeling of auditory spatial receptive fields with spherical approximation functions," *American Physiological Society*, pp. 2645-2656, 1998.

[26] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. of IEEE*, vol. 78, pp. 1481-1496, Sep. 1990.

[27] A. Kulkarni et al., "On the minimum-phase approximation of head-related transfer functions," *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*, 1995.

[28] S. Chen, S. A. Billings, and W. Luo, "Orthognal least-squares methods and their application to non-linear system identification," *Int. J. Control*, pp. 1873-1896, 1989.

[29] S. Chen, P. M. Grant, and C. F. N. Cowan, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. on Neural Networks*, vol. 2, pp. 302-309, 1991.

[30] B. Gold and N. Morgan, *Speech and Audio Signal Processing,* John Wiley & Sons Inc., 1999.

[31] G. Kane and J. Heinrich, *MIPS RISC architecture*, Englewood Cliffs, N.J., Prentice Hall Publishing Company, 1992.

[32] ANSI/IEEE-754-1985, IEEE standard for binary floating-point arithmetic, SIGPLAN Notices, 1985.

[33] J. D. Markel, and A. H. Gray, Linear Prediction of Speech, New York: Springer Verlag, 1976.

[34] C. G. Lee and M. G. Stoodley, "Simple vector microprocessors for multimedia applications," *in Proc. 31th Annual ACM/IEEE International Symposium*, pp. 25-36, 1998.

[35] D. Spaderna, P. Green, K. Tam, T. Datta, and M. Kumar, "An integrated floating point vector processor for DSP and scientific computing," *in Proc. IEEE International Conference Computer Design: VLSI in Computers and Processors*, pp. 8-13, 1989.

[36] L. Benini, P. Siegel, and G. De Micheli, "Saving power by synthesizing gated clocks for sequential circuits," *IEEE Design & Test of Computers*, vol. 11, 1994, pp. 32-41.

[37] Oh Jaewon and M. Pedram, "Gated clock routing for low-power microprocessor design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, Issue 6, pp. 715-722, June 2001.

[38] M. Mehendale, A. Sinha, S. D. Sherlekar, “Low power realization of FIR filters implemented using distributed arithmetic,” *in Proc. of the Asia and South Pacific on Design Automation Conference*, pp. 151–156, 1998.

[39] Y. Jang and S. Yang, “Low-power CSD linear phase FIR filter structure using vertical common sub-expression,” *Electronics Letters*, vol. 38, pp. 777-779, Jul. 2002.

[40] A. T. Erdogan, T. Arslan, “Low power FIR filter implementations based on coefficient ordering algorithm,” *in Proc. of the IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI’04)*, pp. 226-228, 2004.

[41] Federal Information Processing Standards Publication (Draft), Specifications for the Analog to Digital Conversion of Voice by 2,400 Bit/Second Mixed Excitation Linear Prediction, Jan 1998.

[42] M. R. Schroeder and B. F. Logan, “Colorless Artificial Reverberation,” *J. Audio Engineering Society*, vol. 9, 1961.

[43] M. Dolle, S. Jhand, W. Lehner, O. Müller, and M. Schlett, “A 32-b RISC/DSP Microprocessor with Reduced Complexity,” *IEEE J. Solid-State Circuits*, vol. 32, pp. 1056-1066, Jul. 1997.

[44] J. Nurmi, V. Eerola, E. Ofner, A. Gierlinger, J. Jernej, T. Karema, and T. Raita-aho, “A DSP core for speech coding applications,” *in Proc. ICASSP*, vol. 2, pp. 429-432, 1994.

[45] J. Eyre and J. Bier, “The evolution of DSP processor: From early architectures to the latest developments,” *IEEE Signal Processing Magazine*, March 2000.

[46] H. R. Jang, S. H. Kim, and Y. H. Chang, “A Digital Signal Processor for Low Power,” *in Proc. AP-ASIC*, pp. 42-45, 1999.

[47] E. A. Lee, “Programmable DSPs: A brief overview,” *IEEE Micro*, vol. 10, pp. 14-16, Oct. 1990.

[48] E. Zwicker and H. fastl, Psychoacoustics: Facts and Models, New York: Springer Verlag, 2nd Ed., 1998.

[49] 余建政等編著, MATLAB 6.X使用入門, 新文京開發圖書公司, 2003.

[50] M. Keating and P. Bricaud, Reuse Methodology Manual for System-on-a-Chip Designs, Kluwer Academic Publishers, 3rd Edition, 2002.

[51] IBM Inc., The CoreConnect Bus Architecture, website available at

http://www3.ibm.com/chips/products/coreconnect.

[52] Silicore Inc., WISHBONE System-on-Chip Interconnection Architecture for Reuse IP Cores, Oct. 2001, website available at http://www.silicore.net/wishbone.htm.

[53] Opencores Organization, website available at http://www.opencores.org.

[54] ARM Inc., AMBA Specification Rev. 2.0, ARM Document NO. ARM IHI0011A, website available at http://www.arm.com, May 1999.

[55] P. J. Aldworth, "System-on-a-chip bus architecture for embedded applications," *IEEE International Conference on Computer Design*, pp. 297-298, Oct. 10-13, 1999.

[56] E. Hwang, "Building a custom system-on-a-chip," *International Conference on Computer Sciences*, pp. 1-8, 2004.

[57] S. Haykin, *Neural Networks*: *A Comprehensive Foundation*, New York: Macmillan College Publishing Company, 1994.

[58] L. John and A. David, Computer architecture a quantitative approach, Morgan Kaufmann Publishers, Inc., Second Edition, 1996.

[59] Analog Device Inc., ADSP Blackfin DSP Hardware Reference, Nov. 2002.

[60] Inho Lee, et al., "A hardware-like high-level language based environment for 3D graphics architecture exploration," *Proceedings of the 2003 International Symposium on Circuit sand Systems* (*ISCAS2003*), vol. 2, pp. II-512-II-515, 25-28 May, 2003.

[61] G. Cote, B. Erol, M. Gallant, and F. Kossentini, "H.263+: video coding at low bit rates," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 849-866, Nov. 1998.

[62] T.R. Gardos, "H.263+: the new ITU-T Recommendation for video coding at low bit rates," *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3793 – 3796, 12-15 May, 1998.

[63] M. Hans, R.W. Schafer, "Lossless compression of digital audio," *IEEE Signal Processing Magazine*, vol. 18, pp. 21-32, July 2001.

[64] L.-H. Chen; O.T.-C. Chen, R.-L. Ma, "A high-efficiency reconfigurable digital signal processor for multimedia computing," *Proceedings of the 2003 International Symposium on Circuits and Systems* (*ISCAS2003*), vol. 2, pp. II768-II771, 25-28 May, 2003.

[65] P. Coussy, A. Baganne, E. Martin, "Platform-based design for digital signal processing systems: a case study of MPEG-2/JPEG2000 encoder," *Proceedings of*

*the 2002 International Symposium on Communications, Circuits and Systems and West Sino Expositions*, vol. 2, pp. 1361-1366, 29 June-1 July, 2002.

[66] M. Keating and P. Bricaud, Reuse Methodology Manual (RMM) for System-on-a-Chip Designs, Second Edition, Kluwer Academic Publishers, 1999.

[67] P. Nsame, "Building on innovation: The dynamics of SoC design methodologies and application-specific platforms", *SoC Conference,* 2001.

[68] P. Nsame, Y. Savaria, "Virtualising on-chip bus interfaces for improved embedded processor system performance," *IP Based Design 1999,* December 1999.

[69] J. Nurmi, V. Eerola, E. Ofner, A. Gierlinger, J. Jernej, T. Karema, and T. Raita-aho, "A DSP core for speech coding applications," in *Proc. ICASSP*, vol. 2, 1994, pp. 429-432.

[70] J. F. Wang, L. Y. Liu, C. H. Cheng, M. H. Sheu, Y. L. Jeang, and J.Y. Lee, "An ASIC design for linear predictive coding of speech signals," *Euro ASIC '92, Proceedings*, 1992, pp. 288-291.

[71] R. Gumzcj and M. Colnaric, "HW/SW components for real-time systems co-design," in *Proc. the 23rd International Conference on*, 2001, pp. 455-460.

[72] S. Furber, ARM System-on-Chip Architecture, Addison Wesley Publishing Company, Second Edition, 2000.

[73] Andrew Nightingale, Platform Validation Methodology, ARM technology report, Oct. 2002.

[74] S. Y. Kung and I. J. Lin, "Intelligent multimedia signal processing: technology application and challenge," *The IEEE 3rd Workshop on Multimedia Signal Processing*, pp. 629-640, Sept. 1999.

[75] H.-J. Stolberg, M. Berekovii, L. Friebe, **s**. Moch, M. B. Kulaczewski, **A.** Dehnhardt, and P. Pirsch, "HIBRID-SOC: a multi-core soc architecture for multimedia signal processing," *IEEE Workshop on Signal Processing Systems* (*SIPS 2003*), pp. 189-194, Aug. 2003.

[76] D. J. Katz and R. Gentile, Embedded Media Processing, Elsevier Inc., pp. 1-35, Chapter 1, 2005.

[77] S. Osborne, A.T. Erdogan, T. Arslan, and D. Robinson, "Bus encoding architecture for low-power implementation of an AMBA-based SoC platform," *IEE Proc.-Computer Digital Technology*, vol. 149, July 2002.

[78] M. S. McCorquodale, E. D. Marsman, R. M. Senger, F. H. Gebara, M. R. Guthaus, D. J. Burke, and R. B. Brown, "Microsystem and SoC design with UMIPS," Technology Report*, Center for Wireless Integrated Microsystems, University of Michigan*, 2003. http://www.eecs.umich.edu/~brown/Publications/VLSI-SoC-McC.pdf

[79] Y. L. Jeang, G. Y. Wu, and L. B. Chen, "A Microcontroller IP generator for SOC platform," *IEEE Asia-Pacific Conference on Advanced System Integrated Circuits* (*APASIC2004*), pp. 46-49, Aug. 2004.

[80] J. F. Chung and C. T. Lin, "A Low-Cost and Application-Driven Digital Signal Processor for Speech/Audio Processing," *IEEE Asia-Pacific Conference on Circuits and Systems* (*APCCAS 2004*), pp. 373-376, Dec. 2004.

[81] J. F. Chung, C. T. Lin, and D. J. Liu, "Multiband Room Effect Simulator for 5.1-Channel Sound System," *IEEE International Symposium on Circuits and Systems* (*ISCAS 2005*), pp. 2859-2862, May 2005.

[82] S. Agarwala, et. al, "A 600-MHz VLIW DSP," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, Nov. 2002.

[83] J. Edwards, "Technical discussion of parallel DSP on the TI TMS320C40," *IEE Colloquium on General-Purpose Signal-Processing Devices*, pp. 7/1-7/5, May 1993.

[84] G. Hui, W. Tian, W. Ni, and D. Wang, "Real-time implementation of 16 kb/s low-delay CELP speech coding algorithm on a TMS320C30," *IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, pp. 283-286, Oct, 1993.

[85] C. T. Lin and J. F. Chung, "Design of a Low-Cost and Application-Driven Speech/Audio Embedded Digital Signal Processor," *WSEAS Transactions on Circuits and Systems*, vol. 3, pp. 1427-1435, Aug. 2004.

[86] C. T. Lin, J. F. Chung, and D. J. Liu, "A Programmable DSP Core Design for Speech/Audio Codec SoC," *The 2nd International Conference on Circuits and Systems for Communications* (*ICCSC 2004*), Moscow, Russian, June 29-July 1, 2004.

[87] J. F. Chung and C. T. Lin, "A Low-Cost and Application-Driven Digital Signal Processor for Speech/Audio Processing," *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems* (*APCCAS 2004*), vol. 1, Tainan, Taiwan, Dec. 6-9, pp. 373-376, 2004.

# APPENDIX A

# LASP24 Instruction Set and Examples

Notice that the symbol { } represents an optional set. In the set, only choose a term as operand.

| | |
|---|---|
| MOV | Operand：<br>● R{0..7}, R{0..7}<br>● R{0..7}, RAM{0, 1}[#direct]<br>● RAM{0, 1}[#direct], R{0..7}<br>● R{0..7}, {EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}]}<br>● {FIL[{R_FIL，R_FIL+R, R_FIL+J, R_FIL-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}]}, R{0..7}<br>● RAM{0, 1}[{R, J, R+J, J-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]<br>● EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}]<br>● WIN[R], RAM{0, 1}[{R, J, R+J, J-R}]<br>● WIN[R], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]<br>● FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}], RAM{0, 1}[{R, J, R+J, J-R}]<br>● FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}] |
| | Description:<br>● The operation "MOV R1, R3" is to copy data from R1 to R3.<br>● The operation "MOV RAM0[10], R4" shows that data of RAM bank 0 with the address 10 is copied to R4.<br>● The operation "MOV FIL[R_FIL], RAM0[R]" is to copy data from Filter ROM with the address R_FIL to RAM0 with the address R.<br>● R, J, R_EXT, and R_FIL are all auxiliary registers. It is used as the address index. The registers, R and J, is use for RAM bank 0 及 RAM bank 1. Their data width is 10bits. The registers, R_EXT and R_FIL, are use for EXT RAM and FIL ROM, and their data width is 14bits。<br>● RAM{0, 1} indicates internal memory. The symbol EXT indicates external random access memory. The symbol FIL indicates external read-only memory. |

| | |
|---|---|
| ADD | Operand<br>● R{0..7}, R{0..7}, R{0..7}<br>● R{0..7}, {RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]}, R{0..7} |
| | Description:<br>● The operation “ADD R1, R2, R3” means R1+R2→R3.<br>● The operation “ADD R0, RAM0[R], R1” means R0+RAM0[R]→R1. |
| SUB | Operand：<br>● R{0..7}, R{0..7}, R{0..7}<br>● R{0..7}, {RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]}, R{0..7} |
| | Description:<br>● The operation “SUB R1, R2, R3” means R2-R1→R3.<br>● The operation “SUB R0, RAM0[R], R1” means RAM0[R]-R0→R1, where R is the internal memory address register. |
| ADDI | Operand：<br>● R6, R7, R7<br>● R7, #Value (immediate value) |
| | Description:<br>● The operation “ADDI　R6, R7, R7” means R6+R7→R7 (integer addition).<br>● The operation “ADDI　R7,+5” means R7+5→R7 (integer addition). |
| VMPY | Operand：<br>● R{0..7}, R{0..7}, R{0..7}<br>● R{0..7}, {RAM{0, 1}[#direct], EXT[#direct], FIL[#direct]}<br>● RAM{0, 1}[#direct], R{0..7}<br>● {RAM{0, 1}[{R, J, R+J, J-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]}, {RAM{0, 1}[{R, J, R+J, J-R}], WIN[R], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, {RAM{0, 1}[{R, J, R+J, J-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]<br>● R{0..7}, {EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, R{0..7} |
| | Description:<br>● The operation “MPY　R2, R3, R4” means R2×R3→R4.<br>● The operation “MPY　R1, RAM0[20]” means R1×RAM[20]→R1.<br>● The operation “MPY　RAM1[R], EXT[R_EXT+R], RAM0[J] means RAM1[R]×EXT[R_EXT+R]→RAM0[J]. |

| | |
|---|---|
| MAC | Operand：<br>● R{0..7}, R{0..7}, R3<br>● R{0..7}, {RAM{0, 1}[#direct], EXT[#direct], FIL[#direct]}<br>● {RAM{0, 1}[#direct], EXT[#direct], FIL[#direct]}, R{0..7}<br>● {RAM{0, 1}[{R, J, R+J, J-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]}, {RAM{0, 1}[{R, J, R+J, J-R}], WIN[R], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, {RAM{0, 1}[{R, J, R+J, J-R}], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}]<br>● R{0..7}, {EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, R{0..7}<br>● WIN[R], EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}] |
| | Example:<br>● The operation “MAC R2, R1, R3” means R2×R1+R3→R3.<br>● The operation “MAC R1,RAM0[20]” means R1×RAM[20]+R3→R1.<br>● The operation “MPY RAM1[R], EXT[R_EXT+R], RAM0[J]” means RAM1[R]×EXT[R_EXT+R]→RAM0[J].<br>● The operation “MAC RAM0 [j], FIL [R_FIL+J], EXT[R_EXT+J]” means (RAM0[J]×FIL[R_FIL+J]→EXT[R_EXT+J])+ACC→R3. |
| SHF | Operand：<br>● R{0..7}, #Value |
| | Example:<br>● The operation “SHF R2, +5” means that R2 shifts right 5 bits.<br>● The operation “SHF R1, -3” means that R1shifts left 3 bits.<br>● Notice that the operation can shift right or left 24 bits at most. |
| AND | Operand：<br>● R{0..7}, R{0..7}, R{0..7}<br>● R{0..7}, {RAM{0, 1}[#direct], EXT[#direct], FIL[#direct]}<br>● R{0..7}, {EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, R{0..7} |
| | Example:<br>● The operation “AND R2, R3, R4” means R2 *Bit-wise-AND* R3→R4.<br>● The operation “AND R1, RAM0[20] means R1 *Bit-wise-AND* RAM[20]→R1.<br>● The operation “AND R0, EXT[R_EXT], R2” means R0 *Bit-wise-AND* EXT[R_EXT]→R2. |

| | |
|---|---|
| OR | Operand：<br>● R{0..7}, R{0..7}, R{0..7}<br>● R{0..7}, {RAM{0, 1}[#direct], EXT[#direct], FIL[#direct]}<br>● R{0..7}, {EXT[{R_EXT, R_EXT+R, R_EXT+J, R_EXT-R}], RAM{0, 1}[{R, J, R+J, J-R}], FIL[{R_FIL, R_FIL+R, R_FIL+J, R_FIL-R}]}, R{0..7} |
| | Description:<br>● The operation "OR　R2, R3, R4" means R2 *Bit-wise-OR* R3→R4<br>● The operation "OR　R1, RAM0[20]" means R1 *Bit-wise-OR* RAM[20]→R1<br>● The operation "OR　R0, EXT[R_EXT], R2" means R0 *Bit-wise-OR* EXT[R_EXT]→R2 |
| FIX | Operand：<br>● R{0..7}, R7 |
| | Description:<br>● The operation "FIX　R1, R7" means that the value of the float-point register R1 transfer the value of signed integer to R3.<br>● The range of FIX is a 14-bit sign number (from -8192 to 8191). |
| Float | Operand：<br>● R7,　R{0..7} |
| | Description:<br>● The operation "FLOAT　R7, R1" means that the 14-bit sign value of the integer register R7transfer the floating-point value to R1. |

# APPENDIX B

## Addressing Modes

(a) Register addressing mode

| | | | | | | |
|---|---|---|---|---|---|---|
| RC <= RA OP RB or RC <= RA | | | | | | |
| 23 ~ 19 | 18~16 | 15~13 | 12~10 | 9 ~ 7 | 6 | 5 ~ 0 |
| OPCODE | 000 | RC | RA | RB | I/R | Unused (NU) |

(b) Direct addressing mode

| | | | | | |
|---|---|---|---|---|---|
| RAM[Add] <= OP R or R <= RAM[Add] or R <= R OP RAM[Add] | | | | | |
| 23 ~ 19 | 18~16 | 15~14 | 13 | 12~10 | 9 ~ 0 |
| OPCODE | 001 | RAM | R/W | R | Address |

(c) Indirect addressing mode

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| RAM[Add] <= OP RA or RC <= RAM[Add] or RC <= RA OP RAM[Add] | | | | | | | |
| 23 ~ 19 | 18~16 | 15~14 | 13 | 12~10 | 9 ~ 7 | 6 ~ 5 | 4 ~ 0 |
| OPCODE | 111 | RAM | R/W | RA | RC | AR | Unused |

(d) Vector addressing mode

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| VC <= VA[AR_A] OP VB[AR_B] | | | | | | | | | |
| 23 ~ 19 | 18~16 | 15~14 | 13~12 | 11~10 | 9 ~ 8 | 7 ~ 6 | 5 ~ 4 | 3 ~ 2 | 1 ~ 0 |
| OPCODE | 011 | NU | FIL | EXT | RAM0 | RAM1 | VC | VA | VB |

| | **FIL** | **EXT** | **RAM0** | **RAM1** | **VC** | **VA** | **VB** |
|---|---|---|---|---|---|---|---|
| | **13~12** | **11~10** | **9 ~ 8** | **7 ~ 6** | **5 ~ 4** | **3 ~ 2** | **1 ~ 0** |
| **00** | **FIL** | **EXT** | **AR0** | **AR0** | **RAM0** | **RAM0** | **RAM0** |
| **01** | **FIL+AR0** | **EXT+AR0** | **AR1** | **AR1** | **RAM1** | **RAM1** | **RAM1** |
| **10** | **FIL+AR1** | **EXT+AR1** | **AR0+AR1** | **AR0+AR1** | **EXT** | **EXT** | **WIN** |
| **11** | **FIL-AR0** | **EXT-AR0** | **AR1-AR0** | **AR1-AR0** | **R3** | **—** | **FIL** |

(e) Immediate addressing mode

| | | | | |
|---|---|---|---|---|
| R <= R OP Literal or PC <= (Condition) Literal | | | | |
| 23 ~ 19 | 18 | 17~15 | 14~12 | 11 ~ 0 |
| OPCODE | R/C | Condition/Value | R | Literal |

# VITA

## 博士候選人簡歷

姓名： 鍾仁峯

性別： 男

生日： 民國 60 年 1 月 2 日

籍貫： 台灣省苗栗縣

論文題目：

中文：多媒體系統晶片平台設計與應用研究

英文：Design and Application of Multimedia System-on-Chip Platform

學歷：

1. 民國 86 年 6 月私立中華工學院資訊工程系學士畢業
2. 民國 88 年 6 月私立中華大學電機工程系碩士畢業
3. 民國 88 年 9 月國立交通大學電機與控制工程學系博士肄業

榮譽：

1. 參加 88 年度教育部大專院校矽智產 (SIP) 設計競賽，榮獲 Hard IP 優等獎、書面報告特色獎。
2. 參加 2004 年全國 SOC 設計消費性電子類競賽，榮獲特優獎。

經歷:

1. 民國 89 年間擔任電機與控制工程系 VLSI 課程助教。
2. 民國 90 年 9 月至 91 年 1 月擔任中華大學資訊工程系兼任講師。
3. 民國 92 年 10 月至 93 年 10 月參與經濟部科專計劃專案-模組化關鍵音訊 IP 技術開發。

# PUBLICATION LISTS

博士候選人著作目錄

姓名： 鍾仁峯 (Jen-Feng Chung)

期刊部分：

[1] C. T. Lin and J. F. Chung, "Design of a Low-Cost and Application-Driven Speech/Audio Embedded Digital Signal Processor," *WSEAS Transactions on Circuits and Systems*, vol. 3, pp. 1427-1435, Aug. 2004.

[2] C. T. Lin, C. L. Chang, and J. F. Chung, "New Horizon for CNN: with Fuzzy Paradigms for Multimedia," *IEEE Circuits and Systems Magazine*, vol. 5, Issue 2, pp. 20-35, 2005.

[3] C. T. Lin, C. M. Yeh, S. F. Liang, J. F. Chung, and N. Kumar, "Support-Vector-Based Fuzzy Neural Network for Pattern Classification," *IEEE Transactions on Fuzzy Systems*, vol. 14, pp. 1-12, Feb. 2006.

[4] C. T. Lin, C. M. Yeh, J. F. Chung, S. F. Liang, H. C. Pu, "Support-Vector-Based Fuzzy Neural Networks," *International Journal of Computational Intelligence Research* (*IJCIR*), vol. 1, pp. 138-150, Jan. 2006.

[5] C. T. Lin, J. F. Chung, D. J. Liu, H. W. Hein, "Incorporating Spectral Dynamics into LSF Vector Quantization with Error Shaping," accepted paper for publication, *International Mathematical Journal*, Oct. 2005.

[6] C. T. Lin, J. F. Chung, H. C. Pu, "Pedestrian Detection System," accepted paper for publication, *International of Journal Fuzzy Systems*, 2005.

會議論文部分：

[1] C. C. Chen, J. F. Chung, and C. T. Lin, "A Speech Coding Processor Design for MELP Algorithm," *The 11th VLSI Design/CAD Symposium*, KengTing, Taiwan, pp. 323-326, Aug. 2000.

[2] C. T. Lin, Shi-An Chen, Chao-Hui Huang, and J. F. Chung, "Cellular Neural Networks and PCA Neural Networks Based Rotation/Scale Invariant Texture Classification," *2004 IEEE International Joint Conference on Neural Networks*, vol. 1, pp. 153-158, 25-29 July 2004.

[3] S. A. Chen, J. F. Chung, S. F. Liang, and C. T. Lin, "Cellular Neural Network (CNN) Circuit Design for Modelling of Early-Stage Human Visual System," *IEEE International Workshop on BioMedical Circuits and Systems* (*BioCAS 2004*), Singapore, pp. 191-194, Dec. 1-3, 2004.

[4] C. T. Lin, J. F. Chung, and D. J. Liu, "A Programmable DSP Core Design for Speech/Audio Codec SoC," *The* $2^{nd}$ *International Conference on Circuits and Systems for Communications* (*ICCSC 2004*), Moscow, Russian, June 29-July 1, 2004.

[5] J. F. Chung and C. T. Lin, "A SoC Integration Platform for Digital Audio Applications Using FPGA Verification Environment," *The 15th VLSI Design/CAD Symposium*, KengTing, Taiwan, Aug. 10-13, 2004.

[6] J. F. Chung and C. T. Lin, "A Low-Cost and Application-Driven Digital Signal Processor for Speech/Audio Processing," *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems* (*APCCAS 2004*), vol. 1, Tainan, Taiwan, Dec. 6-9, pp. 373-376, 2004.

[7] Y. C. Cheng, J. F. Chung, C. T. Lin, and S. C. Hsu, "Local Motion Estimation Based on Cellular Neural Network Technique for Image Stabilization Processing," *The 9th IEEE International Workshop on Cellular Neural Networks and Their Applications* (*CNNA 2005*), Hsinchu, Taiwan, May 28-30, 2005.

[8] J. F. Chung, C. T. Lin, and D. J. Liu, "Multiband Room Effect Simulator for 5.1-Channel Sound System," *The 2005 IEEE International Symposium on Circuits and Systems* (*ISCAS 2005*)**,** Kobe, Japan, pp. 2859-2862, May 23-26, 2005.

[9] C. T. Lin, S. A. Chen, J. F. Chung, and Y. C. Cheng, "CNN-Based Local Motion Estimation Chip for Image Stabilization Processing," accepted paper for Lecture, *The 2006 IEEE International Symposium on Circuits and Systems* (*ISCAS 2006*), Island of Kos, Greece, May 21-24, 2006.