学校代号：10532

学　　号：S08073014

密　　级：不保密

湖南大学硕士学位论文

低功耗专用指令集处理器设计与优化

学位申请人姓名：　李新泽

导师姓名及职称：　胡锦　教授

培　养　单　位：　物理与微电子科学学院

专　业　名　称：　微电子学与固体电子学

论文提交日期：　2011 年 4 月 10 日

论文答辩日期：　2011 年 4 月 29 日

答辩委员会主席：　曾云　教授

Design and optimization of Low-power application specific instruction set processor

by

LI Xinze

B.E. (Hunan University) 2008

A thesis submitted in partial satisfaction of the

Requirements for the degree of

Master of Science

in

Microelectronics and Solid Electronics

in the

Graduate School

of

Hunan University

Supervisor

Professor HU Jin

April, 2011

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：李新泽　　日期：2011 年 5 月 16 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑。

（请在以上相应方框内打“√”）

作者签名：李新泽　　日期：2011 年 5 月 16 日

导师签名：胡绯　　日期：2011 年 5 月 16 日

摘 要

专用指令集处理器作为嵌入式设计中一种新型的处理器类型，受到越来越多的设计者的关注。它是一种为特定功能而设计，并能够根据应用的需要来对处理器速度，功耗，面积等方面进行衡量，得到最优化设计的一种处理器。在嵌入式设计领域，专用指令集处理器有非常好的应用前景。

本文主要介绍专用指令集处理器设计方法，包括内部结构设计和指令集设计。根据助听器算法中的多通道分离，宽动态压缩等算法，设计了加速模块，并且更新了指令集，优化了硬件结构，降低了处理器功耗。

本文所做的主要研究和设计包括：(1) 本文首先介绍专用指令集处理器的内部结构；（2）介绍该处理器中的指令集，包括在该处理器中该指令集是如何正常工作的；（3）按照专用指令设计方法，设计了指数运算模块，对数运算模块等加速模块；（4）重点介绍了针对助听器算法中较为常用的 FFT 运算中的蝶形运算模块；（5）对存储器电路进行了深入的分析，从内部结构到电路的整体特性进行了细致的介绍；（6）针对本处理器设计了循环缓存结构，优化了程序存储器结构，减少了整个处理器的功耗。

本文对专用指令集处理器电路进行了深入的分析和设计，并且给出了 FPGA 验证，物理硬件设计，后仿结果，功能验证以及芯片的功耗评估结果。程序存储器采用改进的循环缓存结构后，处理器功耗降低 20%，实现低功耗存储器设计。芯片的裸片面积为 1.33X1.23mm^2。在实际的芯片测试中，工作频率为 20MHz,芯片的总功耗为162μW/MHz，符合极低功耗的设计要求。

关键词：低功耗；助听器；存储器；循环缓存

Abstract

ASIP (Application Specific Instruction Processor) is a new type processor in embedded applications. It is a kind of special processor designed for specific applications. By making trade off between speed, cost, power consumption and flexibility, the designers customize ASIP to meet the demand of many design goals.The design of ASIP has great research value in embedded application.

This paper mainly introduces the ASIP design approach, which includes the design of structure and the design of instructions.By using the hearing aids algorithm of the WOLA and the WDRC, we design the speed of the hearing aids module , update the instruction, optimize the hardware structure and reduce the power consumption.

The major research and design work in this paper is shown as follows. 1) Based on the main structure of the processor, this paper make an analysis of the architecture of the hardware, and compare the differences of processors. 2) introduce the instruction of the processor, including the way that how the processor works with the instruction. 3) Introduce the exponent module and the logarithm module in the ASIP processor. 4) emphases the design of the butterfly module 5) Based on the analysis of the SRAM circuit, this paper proposes the whole introduction of the entire SRAM circuit. 6) based on the ASIP processor ,this paper proposed the loop buffering structure, which optimize the whole program memory, reduce the power consumption of the whole chip .

This thesis illustrates the analysis and expatiation for whole hearing aids processor. he FPGA verification ,ASIC design, post-layout results and chip test results are also shown.By using the loop buffering,the power consumption of ASIP can be reduced by 20%, fulfilling the requirement of the low power memory. The die occupies 1.33X1.23 mm^2, Chip test exhibits that the total power dissipation of functional module is only 162 μ W/MHz at clock frequency of 20MHz, feed to the demand of the low power.

Key Words: low pwer; hearing aids; SRAM; loop buffering

目 录

第1章 绪 论

专用指令集处理器（Application Specific Instructions Processor ASIP）是一种应用于特定应用的处理器，由于其具有专用性强，设计灵活，能够满足众多的低功耗，高性能的设计要求，被应用于大量的嵌入式设计中。因此，专用指令集处理器在嵌入式应用领域有非常好的应用前景。

本章首先介绍ASIP的应用背景，然后介绍ASIP设计中的相关研究方法，最后介绍应用于ASIP的低功耗设计的相关概念和知识点。最后给出本文的结构和论点。

1.1 选题背景

随着电子技术的发展，嵌入式设备越来越受到关注，嵌入式应用到了社会的各个方面，其中包括工控设备，信息家电，智能仪表，汽车电子，医疗仪器等各个领域。嵌入式系统是一种以应用为中心，以计算机技术为基础，并且软硬件可以裁减，对功能，可靠性，成本，体积，功耗有严格要求的专用计算机系统。

嵌入式系统设计目前较为常用的一方面是采用固定的芯片，例如ARM公司的ARM处理器。通常这种设计方法对整个嵌入式系统的体积要求不是很高，功耗要求不高，而像网络设备，医疗仪器等设备，对整个电路系统的要求较高。因此这种设备需要设计专用的处理器芯片来适应高性能，低功耗的设计要求，所以目前较为常用也较为的流行的设计方式是利用专用指令集处理器设计（ASIP）方法。

ASIP系统是一种介于通用处理器和专用集成电路之间的一种处理器形式，这种处理器兼备了通用处理器的通用性，也兼备了专用集成电路（ASIC）的高效性。

目前，ASIP作为一种全新的设计结构，受到了嵌入式设计工程人员越来越多的关注，它作为控制内核或者数据处理内核，能够成为通用处理器或者ASIC的替代，在现实中，ASIP被越来越多的应用于各个领域例如差错检验，代码压缩，图像处理。经过已有的调研显示，利用ASIP设计相关的嵌入式系统，一方面可以有效的提高性能，另外一反面可以缩短整个嵌入式系统的设计开发时间，相比较于ASIC设计。

图1.1中显示的在设计耗时，性价比和计算特性方面进行的对比，从图中可以看到，ASIP可以在DSP和ASIC之间进行随意的浮动，并且可以根据设计的需要进行自我的调控。当设计偏向ASIC时，表明在设计中，对性能要求较高，而当偏向DSP时，就表明需要提高整个设计中的设计灵活性。基于以上特点，嵌入式设计人员或者IC设计人员对ASIP越来越重视。因此，在本论文中，对ASIP的讨论就显

得有很大的研究价值。

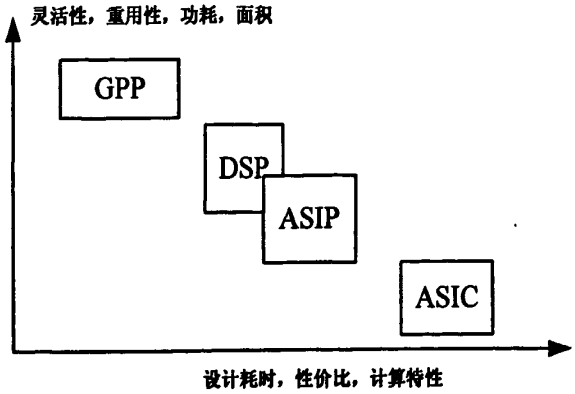


图 1.1 ASIP 与其他芯片的属性比较

1.2 ASIP 背景概述

早在 20 世纪八十年代国内外学术机构就开展了指令集自动设计问题的研究。这些早期的研究更多地集中在设计出复杂指令从而在机器语言级更好地支持高级语言以改善两者的界面。.该种运用指令集自动设计方法是面向 CISC 一类的通用处理器，不适合于 ASIP 这样的处理器设计。

在九十年代，开始了对扩展指令集的研究。当时主要研究可重构处理器。目前，由于嵌入式发展迅速，因此，对硬件设计越来越复杂，设计要求的针对性也越来越强，所以利用 ASIP 开源的高级硬件描述语言，找出在程序中最为频繁执行的代码段，然后分析这些代码的特点，以确定执行代码的操作，然后进行模块的修改，指令的修改。这种方法越来越受到广泛采用。在 90 年代，这项技术被广泛的应用于视频处理器研究领域。近年来，ASIP 处理器在网络与通信领域中被广泛应用，并且已经推出了诸多的产品。比如，Tensilica 公司的 Xtensa[1][2],ARC 公司的 ARCtangent[3][4],Improv 公司的 Jazz[5],3DSP 公司的 SP-5flex[6],Siemens 公式的 Carmel DSP,CoWare 公式的 LISATek.这些处理器都是在可配置的处理器核基础上，能针对特定应用进行优化，同时提供完备的开发工具的支持。

目前 ASIP 设计主要朝向设计更加高速，设计效率更加高效的方向发展，并且主要还在提升 ASIP 处理器的性能上面。因此对 ASIP 的研究有着重要的意义。

1.3 ASIP 设计概述

由于 ASIP 是一个将硬件，软件和算法协同运算的一种电路形式，因此这种电路的设计需要同时考虑软件和硬件的协同问题。在具体设计上面，主要是考虑 ASIP 的设计流程和设计步骤。一个完整的设计流程和步骤可以有效的完成整个设计，而且非常高效[7]。所以设计合适的流程设计至关重要。

ASIP 设计的具体步骤如下：

1. 功能定义：在设计开始时，首先必须明确系统需求，包括需要完成的整体性能，面积，速度，功耗，存储器容量，功耗问题。
2. 算法开发：对于专用的算法运算，需要相关的算法作为支撑。一般这种算法主要是与硬件电路没有直接联系。一般是以 C 语言的模式出现的。
3. 选择系统结构：需要根据算法，性能来选择合适的 ASIP 系统结构。
4. 软件和硬件之间的划分。一般来说，对于 ASIP 设计需要考虑哪些是需要硬件实现，哪些是需要软件实现，需要有合理的划分，然后进行合理的设计。
5. 软硬件的具体实现过程，这个过程需要首先编写相关的汇编代码实现软件的操作，另外需要使用硬件描述语言来实现处理器的基本架构，还需要协同分析整体的性能是否达到标准。
6. 接口设计，即处理器的外围电路的接口设计，例如 SPI 接口，i2c 接口等。
7. 集成测试；
8. 底层设计：电路版图的布局布线设计

图 1.2 是 ASIP 开发具体流程图。

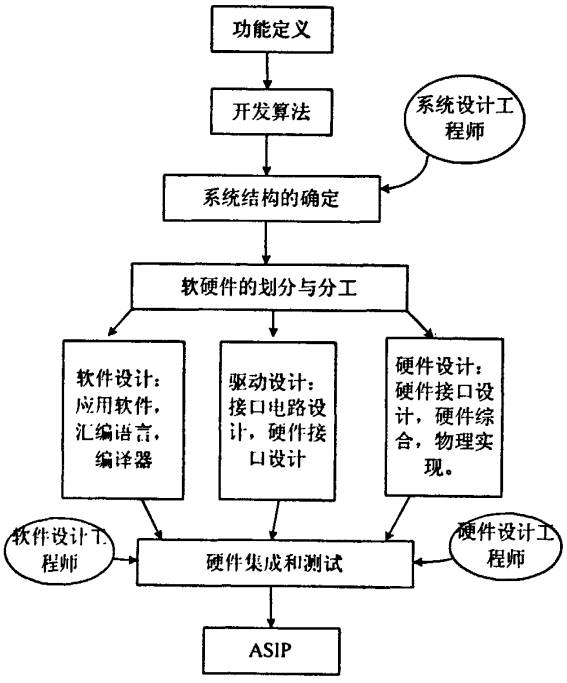


图 1.2 ASIP 开发流程

从图 1.2 中看出，对于 ASIP 设计，需要按照开发的具体流程了进行设计，由于 ASIP 设计是基于专用指令集结构设计，在设计中，功能定义作为第一步，就

确定了 ASIP 设计的硬件复杂度和指令集的复杂度，一方面需要衡量整体功能的特性如何，另外一反面需要兼顾整体芯片的功耗问题。对于 ASIP 处理器中，硬件设计和软件设计同样重要，硬件设计需要专用指令的专用模块，而软件设计需要根据专用的硬件来合理设计软件的运行。编译器的开发在 ASIP 设计非常重要，但是对于特定的算法，是不能够做到尽善尽美的，所以软件工程师需要按照手动汇编的设计方式对整个嵌入式系统进行更加合理的设计规划。

ASIP设计中，对功耗的要求较高，一般在嵌入式系统中，经常为了设计低功耗模块而进行硬件加速等一系列的设计，因此在ASIP设计中，会使用大量的低功耗设计的相关方法。

1.4 低功耗相关技术

在集成电路设计中，功耗受到了越来越多的重视。嵌入式系统由于其便携式特点，要求其具有耗电低，待机时间长的特点。

处理器电路的功耗可以从物理层面上面进行分析，也可以从系统角度进行分析，在这里可以归结为由于电压，由于漏电的压差，形成电流，通过电压和电流的乘积得到功耗。这是对功耗最为根本的一个定义。在 CMOS 集成电路中，功耗大致可以分成两部分，一部分是静态功耗[8]，主要包括有亚阈值漏电，栅漏电和反偏结漏电三大部分；另外一部分是动态功耗：包括短路电流和负载充放电所引起的电流。这部分的主要损失主要有：跳变损失，短路损失，漏电损失。

1.4.1 静态功耗

静态功耗[8]来源比较复杂，主要包括有亚阈值漏电，栅漏电和反偏结漏电三大部分，在这三种静态功耗中，亚阈值电流占整个漏电功耗的大部分，而当工艺进一步提高，栅氧厚度降到 2nm 以下时，栅漏电将可能成为 CMOS 工艺中漏电功耗的主导因素。反相器电路图如图 1.3 所示。

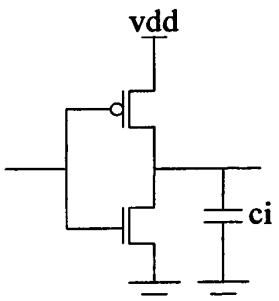


图 1.3 反相器 CMOS 电路图

在图 1.3 中的反相器不工作时，不会产生动态功耗，在这种情况下仅仅有静态功耗，而当输入突然变化，输出进行翻转时就会产生动态功耗，当然，在产生动态功耗的同时也会伴随着静态功耗的产生。

随着工艺尺度的减少，各种功耗会随着产生，例如在深亚微米的情况下，会有泄漏电流的产生。随着特征尺寸的减小，泄漏电流功耗变得不可忽视，减小泄漏电流功耗是目前的研究热点之一。泄漏电流主要是由于随着工艺尺寸的减少，然后会有从栅极到基极的电流产生，另外还有漏极扩散电流。这些电流构成了泄漏电流。

亚阈值电流[12]是由于少数载流子扩散形成了电流，类似一个横向晶体管。图1.4中所示就是亚阈值电流形成的原因。从图1.4中可以看出，电流主要是从源极流向栅极的电流，还有从源极流向漏极的电流。

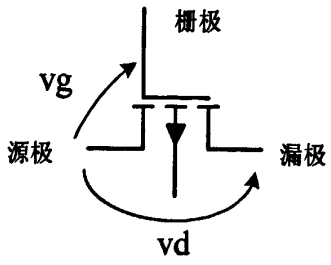


图 1.4　亚阈值电流的产生模型图

$$I_{sub} = I_s e^{[q(V_{GS} - V_T - V_{offset})/nkT](1-e^{(-qV_{DS}/kT)})} \tag{1.1}$$

从式（1.1）可以看出，亚阈值电流和*Vth*有关，当*Vth*增大的情况，会减少亚阈值电流，但是会影响晶体管的开关速度。在进行低功耗设计，在降低亚阈值电流的设计中，需要针对速度和功耗进行综合衡量。相关文献[10][11][12]中给出了亚阈值电流的估算方法。

1.4.2 动态功耗

动态功耗包含有多种类型的功耗，其中包括跳变损失和短路功耗。

跳变损失主要有电路的充放电所引起，这种动态功耗的损失构成了功耗的最为主要的部分，在门电路进行跳变的过程中，会有对负载电容进行充放电的过程，因此会有充放电电流，这部分电流构成了动态功耗的一部分。

在一个输入波形工作阶段，在充电过程和放电过程所需要的能量是和电容值有关的，公式中可知：$E = CloadVdd^2$，*Vdd*是供电电压。

从CMOS电路的设计初衷来讲，理想CMOS电路本不应该存在电源和地之间的直流通路，但在电路实际运行中，由于晶体管受输入信号的变化时间的影响，在输出信号变化的过程中存在PMOS管和NMOS管同时打开瞬间，这就导致了电源和地之间顺势短路电流的存在，这部分由电源地短路而引起的功耗就称为短路功耗。

研究表明短路功耗的大小依赖于信号输入变化时间与输出变化时间的快慢关

系，若输入变化时间比输出变化时间快，则短路功耗小，反之则大[错误！未找到引用源.]。

一般来说，开关频率较低时，瞬态功耗是重要组成部分。

从图 1.5 中的反向器翻转可以看出，当输入的信号从高电平翻转成为低电平

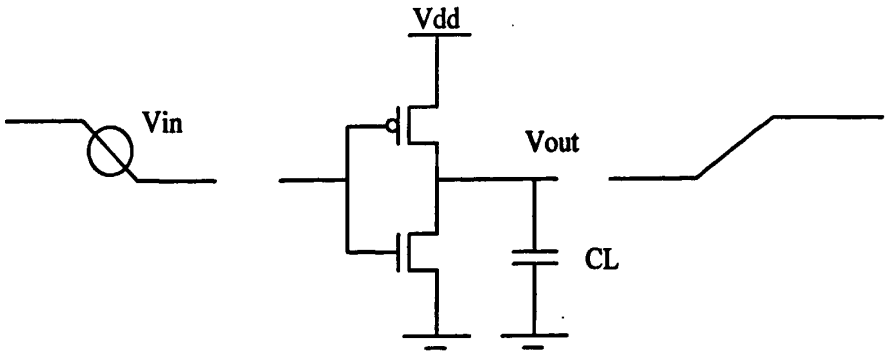


图 1.5　反相器电路的输入和输出关系

时，输出由低电平向高电平翻转。动态功耗产生在翻转的一瞬间。

图 1.6 中波形，当输出有翻转时，会有动态电流产生，这部分电流形成一个的尖脉冲，这样的脉冲电流构成了主要的动态功耗。

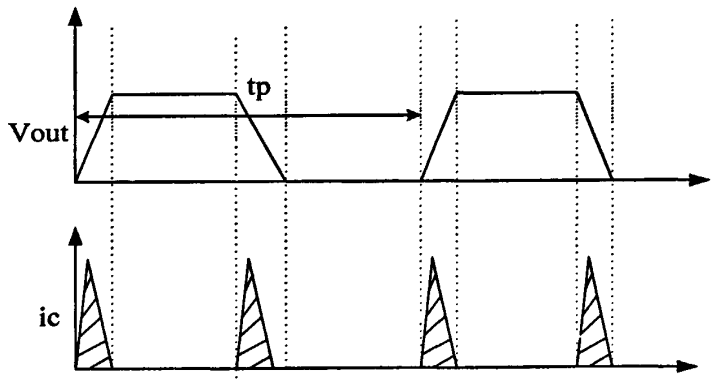


图 1.6　反相器中电压和电流关系图

通过上面的讨论可以得到，在进行低功耗设计过程中，需要通过以下三点来实现降低动态功耗的目的：1.降低电源电压；2.降低开关活动性；3.减少实际电容。

降低电源有电源门控技术[13]，这种技术需要后端版图和工艺的相关支持。降低开关活动性实际上就是减少不必要的门的活动，这种设计比较灵活，可以采用类似的许多的方法实现优化；而减少实际电容则是减少晶体管的数量，间接的减少电容的电容值。

综上所述，在 CMOS 设计中，低功耗设计需要从静态功耗和动态功耗两个方面来入手，其中可以使用的方法有采用高阈值工艺来实现降低阈值电压[14]的目的；使用门控电源[13]，减小电源值；使用相关的算法和相关的设计优化，减少门电路的面积和开关活动。

1.5 处理器低功耗设计优化

处理器的低功耗设计是需要综合考虑的一个问题。整体而言，对处理器的低功耗设计需要从不同的层次和角度来进行实现。在对处理器进行设计时，一般是按照层次化设计的思想进行的。一般会采用从工艺选择，器件设计，模块设计，电路结构设计，算法设计等多个角度进行。

虽然处理器的低功耗设计方法有多种多样，但是归根结底理论设计还是依据在 1.4 节论述的理论，根据产生功耗的根本的原因来进行低功耗的设计。对于电路功耗的评估在学术界也有相当多的研究[15][16]。通过一系列的功耗评估显示，在处理器市场上，为了追求高性能的运算，处理器中往往会添加大量的片上存储器电路，随着微电子技术的提高，嵌入式片上存储器系统会越来越多。嵌入式的存储器所占的比重会越来越大，面积开销也会越来越大，而功耗也会越来越大，研究显示为了提高性能，通常 SOC 中内嵌了大量存储器。嵌入式存储器的容量在整个系统芯片中所占的面积越来越大，其面积高达整个 SOC 芯片面积的 50%~60%[17]。存储器功耗约占整个 SOC 芯片功耗的 25%~40%[18]。因此对低功耗处理器的设计，一大部分关注点就需要关注在对片上存储器的优化上。

在处理器设计中，存储器电路作为一种非常重要的电路形式，完成了数据的传输和中转作用。在处理器中，按照哈佛结构设计，存储器分为程序存储器核数据存储器。由面积设计而言，存储器电路占据了整个处理器面积的 50%~60%的面积开销[19]。

存储器电路可以分成以下几部分，分别为存储单元阵列模块，灵敏放大模块，译码器模块，输入和输出模块，这些模块构成了整个存储器的整体设计[20]。常规的低功耗设计中，存储器模块主要会按照内部电路的特点进行分析，一般而言，存储器电路的存储单元结构式以六管单元为主[21]。灵敏放大器单元有多种电路形式，一般而言，电路为了实现低功耗设计，电路中经常使用锁存型灵敏放大器电路[22]；充电电路[23]和存储单元电路本身就是一体的电路形式，充电电路一方面能够对信号的放大驱动，另一方面能够提高对信号的读取；译码电路[41]是一种进行地址选择的电路形式，该种电路以纯组合逻辑电路为主。

通过查阅相关的文献，可以知道针对存储器有多种降低功耗的方法，一种方式是可以采用优化存储器电路内部结构电路的设计方式，这种设计方式较为复杂，通常为了实现非常快速的设计，采用高阈值工艺[24]设计实现。在设计角度考虑，电路本身有多种优化方式，一种方式是采用优化六管单元的设计方式，这种设计较为复杂，并且不容易实现，另外一种设计方式是采用分割子线和位线的设计方式[25][26][27]。除了对电路的内部结构进行修改，还可以采用利用降低电源电压的设计方式实现[28]，该种方法在后文中会提到，有自己的缺点，导致设计中的问题

产生。

在本文中，为了提高对存储器的优化低功耗设计，采用了从算法设计角度设计实现的一种低功耗优化。其中有大量的设计方法是关于分块设计优化方法[29][30]，还有相关采用 cache 设计方法的[31][32]。本设计中下文通过对比，采用了一种结合两种方法的设计方法，采用循环缓存结构设计方法。该种设计方法在其他处理器中多有提到[33][34][35]，但是电路设计方法多有不同，这种设计方法多是与实际的电路设计相关的。

通过上面的对比分析，可以看到，在存储器设计中有多种设计方法，这些设计方法各有各的优点和缺点，需要针对不同的优点和缺点进行分别的分析和对比，该部分的分析和设计会在后续的章节中提到。

1.6 整体设计思路及实现手段

本文是一篇基于低功耗专用指令集处理器设计的论文，本文主要设计了 ASIP 处理器中的硬件结构，指令集，专用设计模块，和针对该硬件结构所进行的相关低功耗设计。

本文所设计的处理器应用于助听器中，由于助听器体积小，可靠性高，并且要求功耗低，因此在进行极低功耗助听器设计中，就需要将 ASIP 的硬件结构按照助听器的相关算法进行结构改进，一方面设计专用的运算加速模块，另一方面，开辟专用的相关寄存器和相关外围接口电路。而在软件部分，由于助听器需要的算法较为单一，所以针对算法和硬件结构，进行了专门的汇编设计。

由于该助听器的低功耗设计要求，本项目对该助听器进行了低功耗的结构改进，针对处理器中占芯片面积最大的存储器系统进行优化处理，降低了整体的功耗。

综上所述，本文主要研究工作分为两个方面，第一，围绕着高性能和低功耗处理器设计思想，根据语音处理算法对处理器电路进行了加速电路设计，并且设计特有的专用指令模块电路；第二，针对处理器中占据功耗最大的存储器电路进行功耗分析，设计出切合本项目的低功耗设计方案，设计出相关的硬件电路。通过以上两个方向，完成对 ASIP 高性能和低功耗的设计。

最后利用数字后端实现方式完成了芯片的物理实现，并且给出了后端的版图，功耗结果。本芯片采用的是 SMIC.13 工艺实现。

1.7 论文的主要结构

本文分为以下章节，章节的主要内容如下：

第一章，绪论，首先阐述了低功耗助听器的研究背景和国内外发展现状，然后

说明了在现阶段针对低功耗助听器处理器的研究具有非常重要的意义，接着简单说明了本文针对该处理器设计采用了一些低功耗设计思路，最后是本文的结构。

第二章，介绍 ASIP 处理器硬件结构，内部硬件电路模块，指令集和根据指令级的流水线结构。

第三章，先介绍助听器相关算法，按照助听器语音算法设计汇编指令代码。根据 ASIP 设计特点，按照助听器相关算法的要求，设计相关加速电路和优化电路模块，并且设计相关的专用指令。

第四章，分析 SRAM 内部结构，提出相关的降低功耗设计方法，针对程序存储器，提出循环缓存结构，并给出功耗优化的验证。

第五章，主要介绍了本助听器处理器的相关 FPGA 实现，功能仿真和功耗验证结果，最后给出了相关版图结果和功耗测试数据。

对本文的工作总结，并对下一步的工作进行展望。

第 2 章 ASIP 处理器结构设计

本章主要介绍低功耗ASIP的硬件结构。一个处理器结构的好坏，决定这个处理器是否能够正常工作，工作特性如何，与之相关的功耗也会与结构相关。

2.1 专用指令集处理器结构

ASIP(application specific instruction-set processor)作为一种专用指令集处理器[36][37]，主要是应用于专业的例如语音处理，图像处理等算法。由于是有专门的应用范围，因此在进行设计时需要专门定制指令集。专用的指令集是需要和固定的硬件相互协调设计的，例如当进行指数运算时，如果采用通用处理器中的乘累加模块，会耗费许多的资源（包括大量的功耗，时间，需要很多时钟周期才能完成），而如果采用固定的指数运算模块，则既可以满足时序要求，减少时钟周期，又可以减少功耗损失。图 2.1 中显示了 ASIP 的相关内部基本结构。

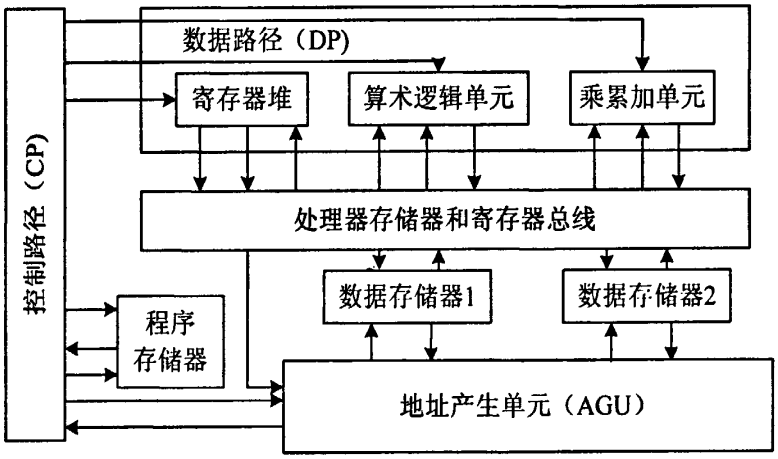


图 2.1 专用指令集处理器整体结构图

ASIP 基本内部结构与常规的 RISC 结构相似，都是以控制路径，数据路径，存储器寄存器相关总线，和相关的寄存器堆和其他的逻辑运算单元构成。在本项目中，由于进行复杂的数据运算处理，因此在设计整体结构时，加入了 MAC（乘累加单元）单元，可以高效便捷的实现卷积运算。

在处理器设计中，不论处理器的属性和结构怎样，它都遵循一个固定的模式，即在处理器设计中将处理器分成多个处理模块，这些处理模块可以分为以下部分：

数据通路：数据通路包括各种加法器，乘法器，数据总线，数据寄存器等结构，主要的作用是完成整个处理器的数据运算过程。在 DSP 处理器中，这部分电路模块占据了非常重要的作用。

控制电路：控制电路涵盖了程序计数模块（program counter PC），指令译码模

块(instruction decoder ID)模块，程序执行模块（EX），等一系列对 DSP 核进行控制的模块，这些模块的主要作用是完成对 DSP 处理器的过程控制，在具体的处理器设计中，能够体现系统性能的设计主要是在该模块电路中。

存储器：对于 DSP 处理器来说，包含了众多的数据处理和指令执行，因此，对于最为基本的 DSP 处理器结构，需要一块程序存储器模块（program memory,PM），用于存放程序二进制指令；另外需要一块数据存储器（data memory DM）,用于存放中间计算数据，例如在快速傅里叶运算中，蝶形运算中的旋转因子，或者在查表运算中的数据值。

由于是设计 ASIP 系统，因此在进行优化设计的同时，就需要额外添加模块来完成数据的加速。某些模块例如除法模块，对数运算模块等。这些模块的完成可以一方面加速硬件运行的速度，另一方面可以减少处理器核数据处理的负担；另外还包含一些硬件电路接口模块,这些电路接口可以完成对和外界数据的交换。

以上部分构成了 DSP 处理器最为基本的组成部分，为处理器实现复杂的功能打下了基础。以上的介绍部分仅仅是处理器电路中的硬件部分，如果想让整个处理器正常的工作，就需要通过软件系统将硬件系统运行起来。

2.2 处理器内部模块电路设计

本节主要针对处理器进行各个模块进行设计。

2.2.1 程序计数器单元

程序计数器单元(Program Counter PC):该部分是整个处理器的核心部分之一，在处理器正常工作时，需要指令来控制整个处理器进行工作，而这些指令是通过片外的 FLASH 模块加载到处理器上的片上 SRAM 中的，因为读片上的 SRAM 读取的方式不同，就会出现不同的运行操作。而程序计数器（PC）模块就是一个顺序执行整个程序计数的单元模块。在正常工作的情况下，程序指令计数器可以进行自加一的计数控制，也可以进行相关的跳转操作或者循环操作。这一系列的控制操作都是可以通过 PC 模块的控制来完成的。而在 PC 模块中，PC 状态机是非常重要的一个单元模块。

PC 状态机可以控制写入和读出路径，能够实现对数据流的控制。PC_FSM 控制模块对程序存储器中的指令可以分成两类指令：一种指令是产生正常指针自加类的指令；另外一种是跳转类的指令。例如 jump ,call ,ret ,repeat 指令。

图 2.2 是 PC 模块的具体工作模式状态。对于 PC_FSM 模块而言，PC_FSM 的程序指针控制有如下的运行方向：当没有跳转类指令时，则程序指针 PC=PC+1；重新启动整个状态机时，则需要将 PC=0 进行设置；当遇到跳转指令时，则可以将程序指针自动加上跳转后的地址量，PC=TARGET ADDRESS。

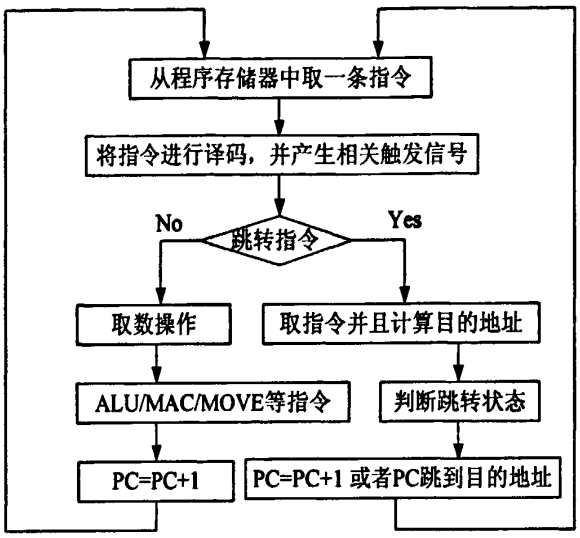


图 2.2 PC_FSM 状态机控制状态图

其中程序状态机控制如图 2.3 所示。当接收到不同的命令时，会有不同的转换结果产生。

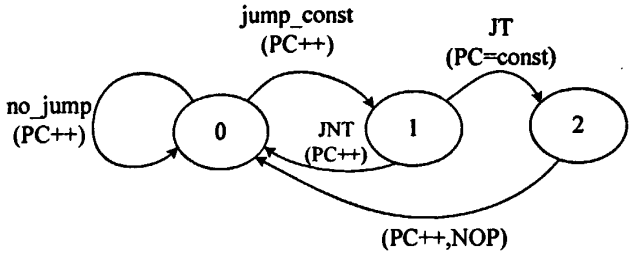


图 2.3 跳转控制流程图

图 2.3 中的跳转模式可以简单的归结为三类：一种方式是通过程序计数器产生，在一般程序运行时，程序计数器实现了程序的自累加一的过程，地址信号加一，自动读取下一位程序存储器中的程序；第二种方式是从程序累加状态机中产生，该种方式可以实现对程序的跳转执行；第三种方式是执行循环控制，该种方式是采用循环控制模块和程序指针控制模块协同工作，这样可以实现循环指令的执行操作。

2.2.2 指令译码器单元

指令译码单元(ID,instruction decoder):该部分电路主要是对输出指令进行译码操作。由于读取出来的指令包含了非常多的信息，ID 模块会产生大量的模块触发信号，指挥各级模块更好的完成工作。图 2.4 是 ID 级译码控制模块。

具体操作流程如下，对从程序存储器读取到的指令进行译码后，产生许多的控制信号，这些控制信号可以分成如下几类：

1. 提取立即数信号：在程序指令中，会包含立即数的部分，对相关的立即数的

操作指令，可以采用直接提取立即数的模式将相关的立即数存储到相关的普通寄存器中。

2. 产生控制信号：控制信号包括对立即数的控制操作，包括对普通寄存器或者数据存储器的相关控制操作使能。

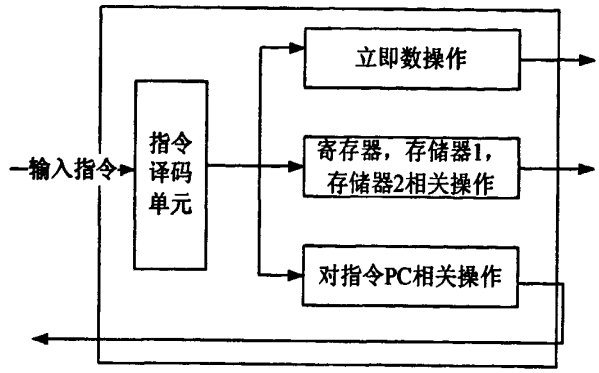


图 2.4 ASIP ID级译码控制模块

3. 输出指令的写回信号，该信号会对 PC 模块产生影响，影响程序的正常走向。正是 ID 一级对指令的正常解析，促成了处理器对整个程序存储器的指令的正常解析，处理器的正常运行，ID 一级非常关键。

2.2.3 算术逻辑单元

算术逻辑单元(ALU)：该部分电路主要是进行相关的数据通路的运算，分别进行的是加法运算，逻辑运算等。

ALU 单元是主要计算从 RF 或者其他寄存器中输出的操作数，该部分运算会消耗一个时钟周期，主要的运算包括数据运算，例如加法运算，逻辑运算（包括或运算，与运算，异或运算等），移位运算（左移运算，右移运算，循环左移运算，循环右移运算等），还有多路选择模块。ALU 结构图如图 2.5 所示：

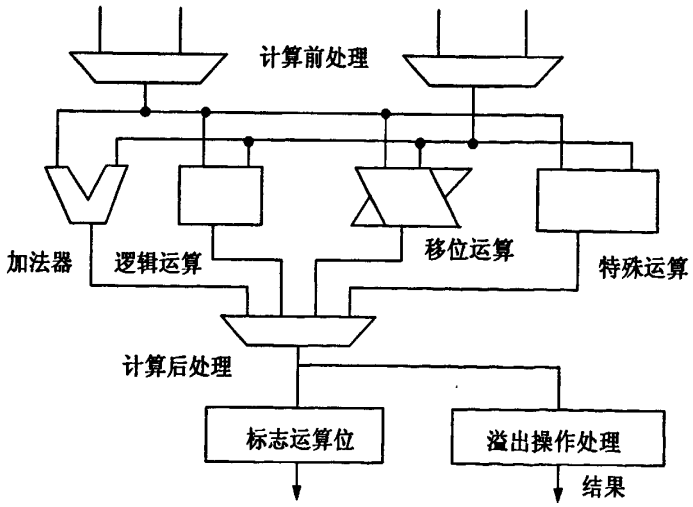


图 2.5 ALU 加法器内部模块结构图

在专用指令集处理器中，ALU 模块作为基本的运算模块，还负担了程序地址

的运算的任务。在某些跳转指令或者循环指令中，ASIP 处理器利用 ALU 来计算跳转地址的目标地址，结束地址和循环次数。

2.2.4 乘累加（MAC）单元

乘累加模块(MAC)：乘累加模块主要进行乘法运算，由于 DSP 运算中有大量的乘法操作，因此在电路实现过程中，使用专用的乘法模块，可以有效的提升处理器的处理能力。MAC 模块是在 DSP 模块中最为重要的模块单元，在该模块中包含了乘法器模块，累加器模块，累加寄存器模块，多路选择器模块，函数功能模块（包括旋转，缩放，饱和功能模块）。其中 MAC 支持的运算有卷积运算，数据转换运算（FFT 运算，DCT 运算等），双精度的浮点运算等。

在进行 MAC 设计中，最为重要的一个单元模块就是对乘法器的设计，在二进制乘法中，有符号数和无符号数的得到的结果是不一样的，因此在进行 MAC 设计中对乘法器的设计需要利用多路选择器对有符号数和无符号数进行分别的设计实现。图 2.6 是乘法器模块图。

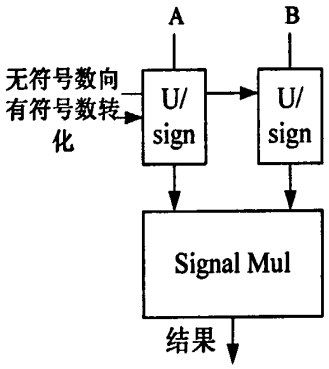


图 2.6 乘法器模块图

另外该 MAC 模块还可以实现 16 位的定点型或者浮点型的运算，32 位的长整形的加减运算或者求整运算。图 2.7 是 MAC 内部模块结构图[38]。

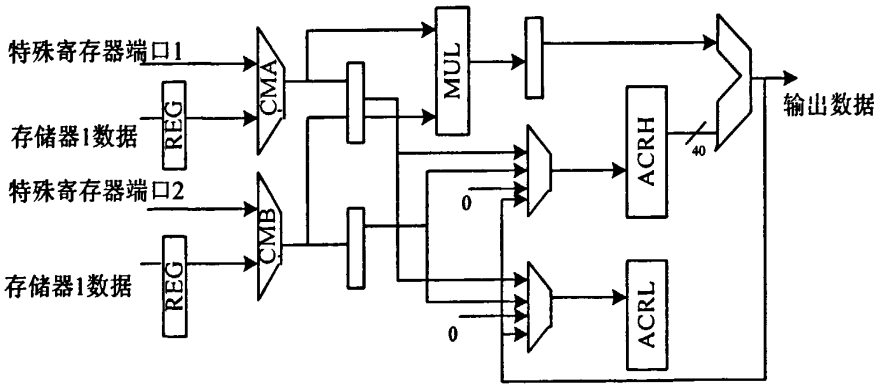


图 2.7 MAC 内部模块结构图

2.2.5 地址产生单元（AGU）

AGU（address generation unit），被称为地址产生模块。该模块属于处理器存储器模块中的一部分，该模块主要的作用是用来产生地址信号的一个模块。

图 2.8 中是整个处理器电路的模块图，其中 AGU 模块就是在图中下方的功能模块。该部分属于存储器系统，其中存储器系统包括以下结构：物理存储器，地址产生单元，存储器电路总线。

AGU 模块的主要功能是根据上一级的寄存器，存储器或者立即数的值来产生存储器的读取或者写入地址。输入到 AGU 的信号包括有经过 ID 一级得到的译码的地址信号，从数据寄存器的输出的数据，或者从地址寄存器的输出的地址信息。输出的数据是存储器的地址。电路结构的模型如图 2.8 所示：

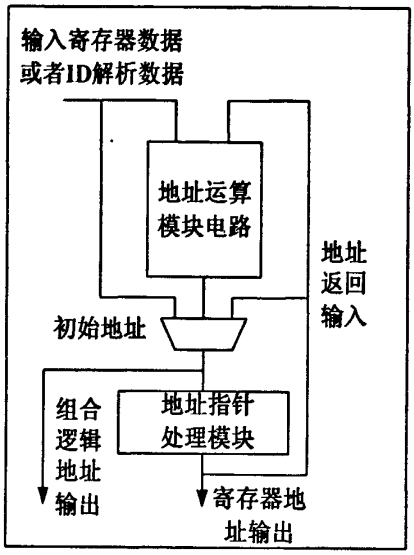


图 2.8 AGU 模块电路结构图

在 AGU 模块中采用了 AGU mode 模式选项，这个选项可以实现对多种地址寻址模式的选择，AGU 的寻址模式包括直接寻址，寄存器直接寻址，寄存器寻址，自增寻址，自减寻址，偏移量寻址，后模块自增寻址。

内部逻辑功能模块设计得到的结构图如下：从图 2.9 内部逻辑结构可以看出，在进行 AGU 设计时，需要将不同的通路都输入到 AGU 模块中，然后得到信号的输入实现信号的自增或者自减，当信号由地址控制信号选择控制时，则进行相应的操作。

在电路图 2.8 中，AGU 模块主要是进行各种译码电路的选择操作，地址模块是从该电路中生成。多种模式的操作，构成了电路的整体结构，使得电路的灵活性更高。

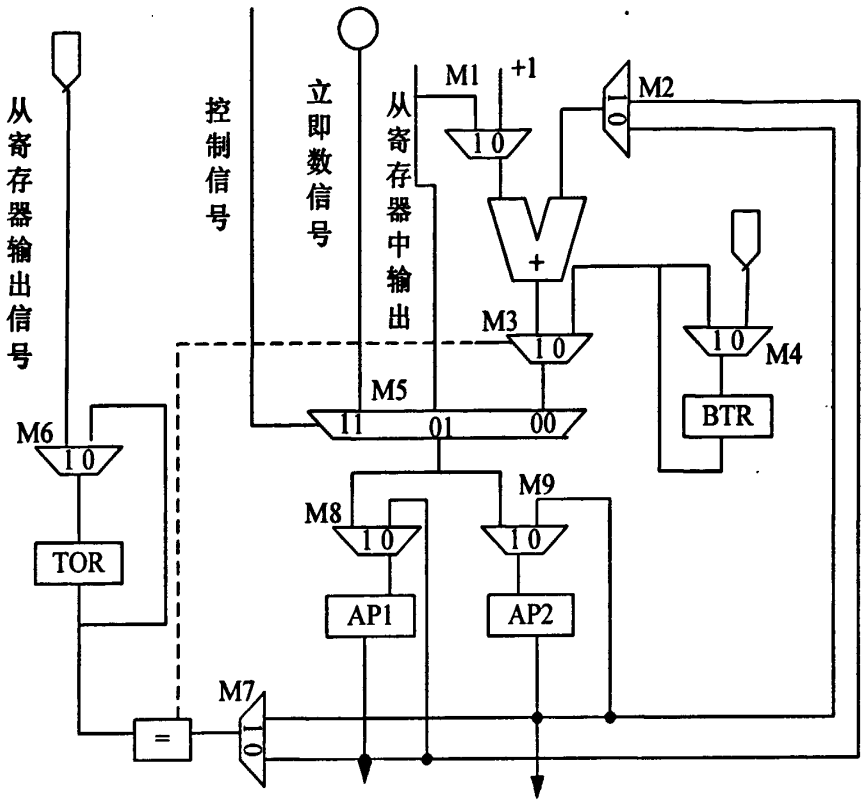


图 2.9 AGU 内部模块逻辑

2.2.6 寄存器堆模块

寄存器单元(REG):许多的操作例如乘法运算或者加法运算，逻辑运算等都是针对寄存器单元实现的，因此在 ASIP 中包含了大量的寄存器单元电路。寄存器中包含两种类型的寄存器，一种寄存器是普通寄存器，又称为通用寄存器，主要是用来存放中间数据的；另外一种寄存器是特殊寄存器，这种寄存器主要是为完成整个处理器的数据控制，模式选择等工作。

2.2.7 存储器单元

存储器模块（MEM）：存储器模块电路主要包括程序存储器和数据存储器，程序存储器主要是存储了大量的指令，用于从存储器中读取指令，然后用于译码操作。数据存储器主要用于实现对存储数据的操作。

图 2.1 中显示的是处理器中以上部分的模块结构图，从图上面可以看到 PC 输出的计数控制信号，通过 PM 输出出相关指令信号，指令信号传入到 ID 中，通过译码操作解析指令，然后分别从寄存器单元中读取数据或者从数据存储器中读取数据。利用 ALU 单元或者 MAC 单元进行数据的运算操作。

在 ASIP 处理器中的数据位宽为 16 位，在数据存储器 DM0 和 DM1 中分别都是 16 位的数据位宽，REG 寄存器也是 16 位的，通用寄存器有 32 个，分别执行了不同的运算操作。

2.2.8 处理器相关外围设备

中断模块：在处理器设计中，加入了相关的中断操作，这些中断操作主要是进行执行外部中断到来时，停止内部处理器的工作，转向另外的处理部分的作用。在 ASIP 的中断操作中，有多个中断源。这些中断源分别是外部六个中断，一个掉电中断，一个看门狗中断，这些中断端口构成了整个处理器中的中断系统。

该中断系统对应的中断使能寄存器（interrupt en）寄存器，中断标志寄存器（interrup flag）寄存器，中断优先级寄存器（interrupt priority）寄存器，这三个寄存器完成了对中断设置的控制。

在中断使能寄存器的控制中，包括中断使能总体控制位，中断使能分别控制位。当某位置 1，表示该中断使能有效，可以响应中断，当某位为 0 时，表示中断使能无效，则不能响应中断。

在中断标志寄存器中，如果某位的中断响应，则表示某位处于中断状态下。因此在执行中断处理时，可以根据该位的显示结果来实现中断的控制。

中断优先级寄存器，在中断优先级寄存器中，当某位置为 1 后，表示中断响应优先级较高，当某位置为 0 时，则表示中断响应优先级较低。

定时器模块：定时器电路主要是产生定时信号，为串口产生固定的波特率信号，定时计数器的时钟源可以是处理器的内部时钟，也可以是处理器的外部时钟，当是外部时钟时，通过对外部时钟沿的计数，实现对需要的控制。

在定时器的模块端口处，有特殊寄存器的地址信号，数据信号和输出信号。因此通过特殊寄存器可以有效的写入到定时器的寄存器中。在定时寄存器，有一个专门用来计数的寄存器，分为高八位和低八位，通过选择不同的工作模式，实现对计数器不同波特率的选择。

在处理器电路模块中，有许多的模块是在该处理器中单独设计的，数据前推模块(fwdmux)主要是将下一条流水线数据用的数据量需要上一条指令的结果，当在 ALU 级已经执行完成时，按照常规操作需要在写回级才能输出，但是有了数据前推模块后，可以将数据直接前推到执行的寄存器或者存储器处，这样可以节省时钟周期，避免许多不需要的等待。并且采用数据前推后，在处理器中的另外一个功能模块，旁路预取模块的问题就可以解决。

在处理器中，会有相关的分支冲突的问题。在该处理器中，对分支冲突的解决采用延迟槽的方法，在跳转指令旁边添加 NOP 指令，实现避免指令的冲突。

2.3 处理器指令集

指令系统是进行软硬件划分的依据。指令集的设计应该实现微处理器需要实现的所有功能，一个优化的指令集可以用一条或者几条复合指令实现用户程序中

的任一操作。为了提高指令执行的效率，该处理器结构采用了 RISC 简单格式的基本指令模式。

计算机指令格式有固定长度和可变长度两种，对应的有两种不同类型的计算机：精简指令集计算机（RISC）[39]和复杂指令集计算机（CISC）。RISC 处理器一般采用固定长度的指令格式，定长指令格式的好处在于指令译码和流水线执行比较容易。RISC 处理器的指令系统的共同特点是指令种类少而精，寻址方式简单，指令格式固定。

2.3.1 处理器指令分类

处理器可以分为以下四类指令：

1. 数据传输类指令
2. 运算类指令
3. 程序控制类指令
4. 加速类指令

正如前文所述，该内核部分包括有数据通路，控制通路，地址寻址通路三个部分。数据通路部分包括有通用的寄存器组，ALU 模块，MAC 模块；地址通路部分包括四类电路模型，AG0-AG3，AG1-AG3 累加器寄存器，可变寻址模块，位反模块；控制通路包括 PC 状态机，循环控制状态机，和重复指令。

2.3.2 数据传输类指令设计

数据传输类指令：数据传输类指令主要数据的传输，能够在通用寄存器，特殊寄存器，累加器，IO 端口部分,数据存储器部分进行传输。move 指令只能在寄存器和累加器之间进行数据的传输，load-store 指令能够在存储器寻址，数据端口操作等方面进行工作。in 指令用于将 IO 口的数据写入到寄存器中，out 指令用于将寄存器的数据传出到 IO 口上。

2.3.3 数据运算类指令设计

16 位的短运算操作指令主要关注的是 16 位的运算操作。许多的指令包含了对标志位的操作。短运算的操作指令包含了指令有：加法类指令（其中包括有带符号位，运算进位，溢出操作等，加法指令仅仅是只能进行寄存器和立即数的操作），减法类指令（操作方法和加法类相同），比较类指令（主要是比较寄存器和立即数）；最大，最小，求整类指令（求最大值，最小值，求整数）。

16 位的短逻辑操作，16 位的短逻辑操作主要是按照逻辑操作进行设计，并且在 ALU 和 MAC 模块中得到的结果进行操作，得到的标识对逻辑操作进行控制。其中操作指令包括与操作指令，或操作指令，异或指令都是在寄存器和立即数之间进行的，有标志位改变或者没有改变之分。

16 位短移位指令包括有 16 位的移位操作和循环移位操作。所有的指令都是可以实现条件操作，所有的指令都可以以寄存器中的数据或者立即数为移位的次数，移位操作的指令包括算术左移和右移，逻辑左移和右移，循环左移和循环右移，带进位的循环左移和带进位的循环右移。

2.3.4 32 位长运算类指令设计

32 位的长运算类指令，该长运算类指令能够利用 ALU 和 MAC 的标志寄存器的标志位进行状态控制，进行一系列的例如加法，减法，对存储器内部 32 位数据进行加操作，对存储器内部 32 位数据进行减操作。逻辑操作，比较操作，逻辑操作，乘法操作，乘累加操作和求卷积操作，这些操作主要就是进行寄存器和存储器的一系列操作模式。数字信号处理运算中，卷积运算非常重要，在该处理器中也包含卷积运算指令。

2.3.5 过程控制类指令设计

过程控制类指令，主要是进行指令的跳转，其中包括有跳转指令，子程序调用指令，返回指令，空操作指令。这些指令主要是执行程序的代码段的跳转操作，其中这些指令都对应了相关的寄存器操作和相关的标志位的操作。

综上所示，所有的指令代码结构都是以 opcode 为主要结构模型，然后配合源操作数和目的操作数，作为指令操作的目的所在。另外添加额外的一些指令也是按照这种结构模式设计的，例如在下面设计的 FFT 中，会有一个 butterfly 模块，根据该模块设计，专门定制了一条指令，通过设计专用的结构和指令集形式，实现了整个性能的最优化。

2.4 处理器流水线

一条指令在运行时会有牵扯到许多不同的模块操作，一条标准的运算指令，一般包含的操作包括以下部分：

FE(取指令阶段):该阶段主要是实现了程序计数器（PC）工作，产生地址指针信号，将地址指针信号写入到程序存储器中，然后对程序存储器进行读写，输出指令。在该部分中有一系列的跳转操作或者循环操作在里面，为处理器的灵活运行提供了方便。

ID（指令译码）：在该阶段主要是进行输出的程序指令进行译码操作，产生多个不同的使能操作，实现在以后进行的操作中，进行不同的控制。

OF(对寄存器操作)：在该阶段实现了对不同的指令控制下，针对不同的寄存器进行操作的目的，在该部分电路，可以对寄存器进行操作，也可以对立即数进行操作。针对不同的寄存器和数据通路进行不同的操作控制。

ALU（运算逻辑单元操作）：在该阶段主要是针对寄存器中的数据进行相关的数据运算操作等，该部分实现了最为基本的运算功能，不仅包括数据的运算，还包括了程序指针的跳转运算。

WB（写回模块）：通过在 ID 级的控制指令，将输出的运算完成的数据写回到相关的寄存器中，实现从哪里取数，从哪里存入的过程。

在指令执行中，处理器中的各个部分是相互衔接处理的。前文已经对处理器中算术运算模块(ALU)，程序计数器(PC),寄存器（RF）,写回模块（WB）等做了结构上面的介绍，在整体工作时会将以上各个部分进行衔接介绍。图 2.10 所示，是流水线前三级的电路形式。

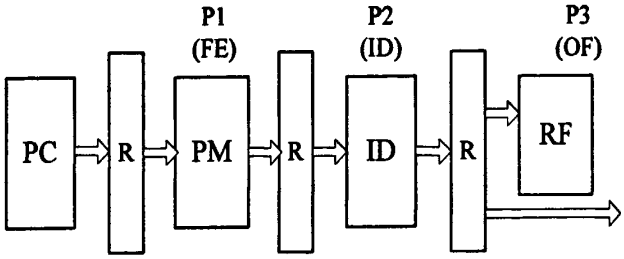


图 2.10 指令流水级前三级

常规的处理器设计中，主要有以上图 2.10 中的模块，在前面三级主要是进行 PC 指针计数，PC 指针从 PM 中读取数据，输出指令。然后输出的指令从 ID 级进行译码操作，给出相关操作的触发信号或者操作数据。

前面三级的流水线结构是较为常规的结构模型，而具体的操作的不同是和后级的运算的方式有关的。

算术类运算的指令，例如加法指令或者其他的简单指令，都是较为简单的运算，在计算过程中仅仅通过一个 ALU 模块就可以实现。图 2.11 所示。

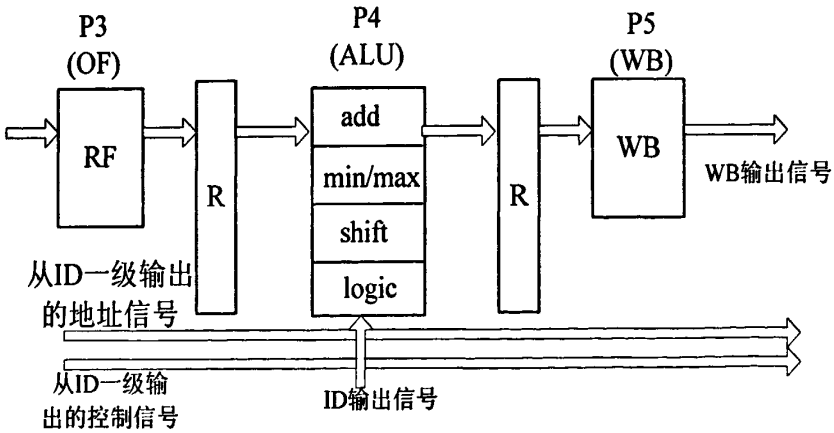


图 2.11 算术运算类指令结构模式图

在图 2.11 中，这个结构中需要有 5 级流水线的操作，需要 5 个时钟周期，顺接前文中 P1,P2,P3 级。从 P3，P4，P5 级可以看到，整个处理器中算术运算单元模块中包括另外的两级，分别是 P4 级的 ALU 单元，WB 单元即写回单元，在整

个单元中，从 ID 一级输出的信号会直接通过写回模块向外输出。而另外的由 P2 级输出的控制信号，也可以直接作用于 P4 级，对 ALU 进行控制，这样从 P3 级寄存器输出的数据和直接从 P2 级的 ID 级输出的立即数模块可以在 P4 一级进行逻辑运算。在 P4 级逻辑运算包括加法运算（add）,求最大值和最小值（min/max），移位操作（shift）,逻辑运算（logic）等。

在本 ASIP 结构中较为常用的结构是乘累加运算单元，完成复杂的乘累加运算。从上面的电路结构图 2.12 中可以看到，在利用乘累加单元时，需要额外的六级流水线运算，在其中 P4 级单独作为了乘法运算一级，这级运算主要是进行两个数据量的相乘运算，当进行到 P5 级时，这级主要是进行类似于加法累加的过程，其中模块 ACR 是累加器，即在进行乘累加运算的时候是必须在累加器中运算的。其中由前级 ID 级会输出多路的信号，这些信号会连接在多路选择器中，用于选择数据的通路。经过前一级的输出可以将由 ID 输出的数据信号累加到加法器中。输出的结果连通到 P6 的 WB 级。这样构成了整个乘累加的运算过程。

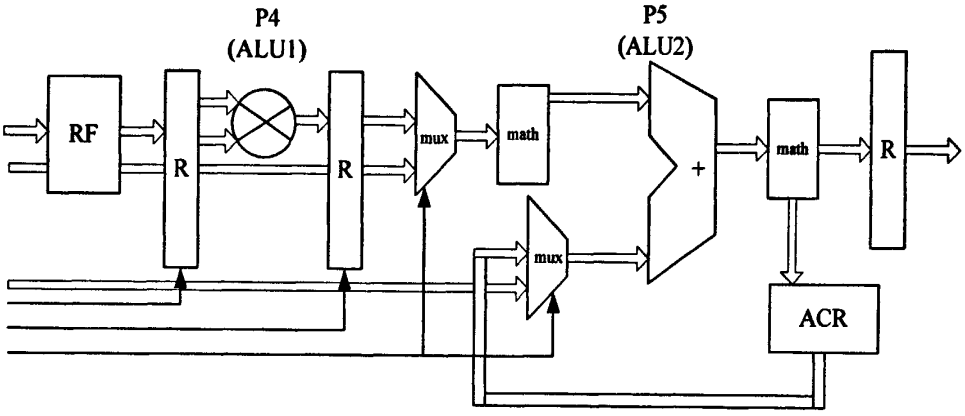


图 2.12　乘累加运算结构模式图

卷积类指令，在专用于助听器的处理器中，是一类比较常用的指令，这类指令主要用于设计进行数字信号处理的运算。卷积类运算时间周期长，并且由于涉及较多的数据计算，因此在计算过程中，需要直接对存储器进行读取处理，所以卷积类的运算结构就如图 2.13 所示。

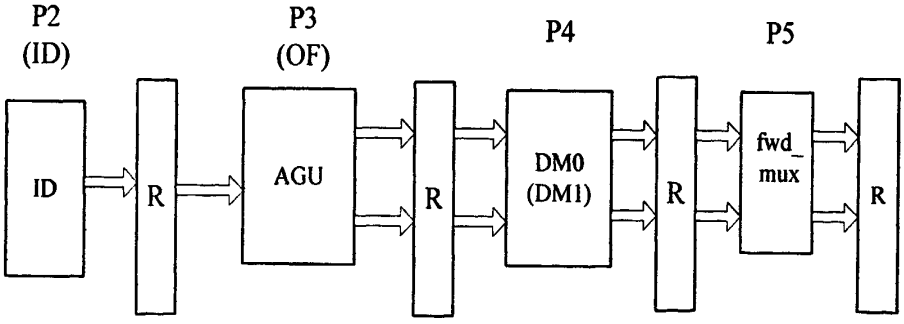


图 2.13　卷积运算流水线结构

上图 2.13 中，在进行计算时，添加了额外的三个计算模块，一个是 AGU 模块，一个是 DM 模块，另外一个是 fwd_mux 模块，AGU 模块产生查找地址，向外输出地址，而 DM0 为数据存储器，专门用来进行数据信号的存储，一般在进行卷积运算中都是需要进行数据系数的计算的。而 fwd_mux 模块是进行前插运算，表示计算得到的结果不用完全写回，就可以在下一级使用。后面的 p6 级和 p7 级分别是在乘累加运算中的 ALU1 和 ALU2，最后写出的结果从 WB 中进行输出。

数据传输类指令，数据传输类指令主要是进行数据从存储器中读取到寄存器中或者将数据从寄存器中写入到存储器中。标准的指令有 store 和 load 指令。图 2.14 所示是 load 指令的流水线结构。

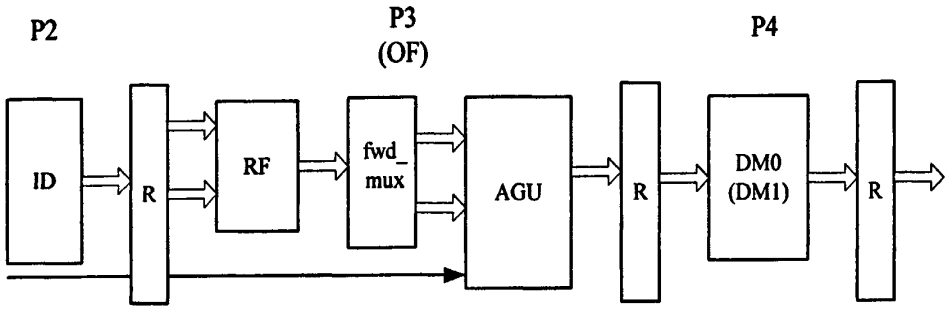


图 2.14　load 指令流水线结构

图 2.14 中是 load 指令，这个指令需要有五级运算，这五级运算主要是 ID 级，OF 级，P4（运算）级，WB 级。首先通过 ID 级输出指令的译码操作，从 RF 中读取到数据，然后通过 AGU 产生地址单元，写入或者读出 DM0 模块。最后写回一个信号到前级中。

程序跳转类指令运行结构，JUMP 指令的执行中与标准的五级流水执行是一样的，这种执行过程主要是在 PC 一级，会有条件判断出现，在进行程序运转时，如果后级检测到是 JUMP 指令时，则该指令就需要按照后级写回的数据进行条件判断。

PC 模块中包含有多个输入信号，这些信号构成了对 JUMP 指令的控制。图 2.15 中就是 PC 模块中相关控制端口，多个使能信号可以有效的实现多个指令结构的跳转过程，其中加一和减一模块实际上就是整个设计中对指针的自加处理和自减处理。

在循环控制指令 repeat 中，设置好初始的地址段和控制字，由后级的相关的控制端口进行输出的控制，可以实现循环指令控制操作。循环的次数是存储在后级的 RF 中，而指令的条数有 ID 级进行解析后，直接传输回 PC 前级的 FSM 模块处，通过 FSM 联合设计实现对 repeat 指令的控制调整。

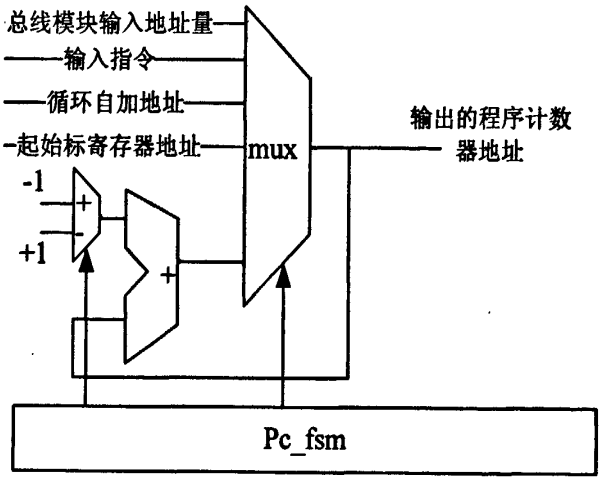


图 2.15 PC 模块内部结构图

2.5 本章小结

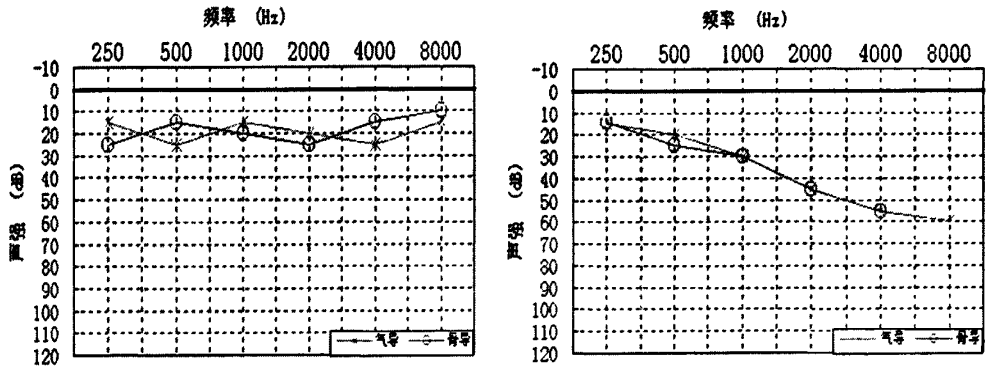
本章主要介绍 ASIP 处理器内部结构。在常规的 ASIP 处理器中，结构模式和 RISC 结构模式相似，本 ASIP 处理器仍是以控制路径，数据路径，寄存器堆，存储器系统，算术逻辑单元模块为基本组成模块单元。在处理器正常工作时，按照不同的指令操作会有不同的流水线级数参与运算。程序控制类指令为 5 级流水线，算术类指令为 6 级流水线，而例如乘累加类运算则有 7 级流水线的操作运算。本章主要介绍 ASIP 处理器的内部结构，在后续章节，将介绍 ASIP 的指令集结构和相关的助听器算法在 ASIP 处理器中的应用。

第 3 章 专用指令集处理器算法及应用

本章主要介绍ASIP的功能应用及相关特点指令模块的设计和指令设计。专用指令集处理器通常会使用在语音及图像处理当中，所以在本论文中，以助听器处理器设计为例进行专用指令设计的介绍。首先简单介绍助听器算法，然后介绍该助听器算法和处理器的软件实现，最后根据功能设计加速功能模块和专用模块。

3.1 语音助听器算法设计

助听器算法是一种语音处理算法。语音算法一般是以滤波器组为设计基础。由于听障人士的听力阈会在高频域部分衰减，如图 3.1 所示，所以需要将语音信号按照频率进行多通道的划分。随后的语音处理按照不同频率域进行处理器。常用的助听器语音处理算法包括：多通道分离（WOLA），宽动态压缩(WDRC)，噪声消除，反馈消除等。



（a）正常听力曲线　　（b）老年性耳聋听力曲线

图 3.1 正常听力曲线和耳聋听力曲线对比图

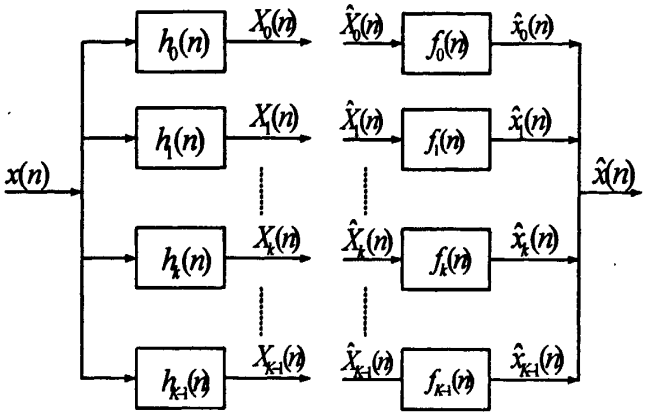
3.1.1 多通道分离算法

实现声音信号的多频率通道分离，可以采用直接 FFT 法，滤波器组的方法，加权叠接相加（WOLA,weighted overlap-add）算法等，其中 WOLA 算法以其性能好，开销小等优点在助听器中被广泛应用。

通过比较几种实现多通道分离的算法，可以看出，利用 FFT 算法会出现较多的频谱泄露，在后续处理中会带来混叠问题；滤波器组由于计算量大，计算量和频率数呈正比关系增加。所以针对滤波器组进行了结构改进，简化了滤波器组的设计。实际上多通道分离算法就是滤波器组算法的简化。

多通道分离算法可以分为两部分处理，分析部分和综合部分，如图 3.2 所示。

一般分析部分会有如下功能：下变频，分析滤波器低通滤波，下采样。而综合部分则是有上采样，合成滤波器低通滤波，上变频。



（a）分析滤波器组　　（b）综合滤波器组

图 3.2　WOLA 结构图

图 3.3 是多通道分离算法的具体实现。

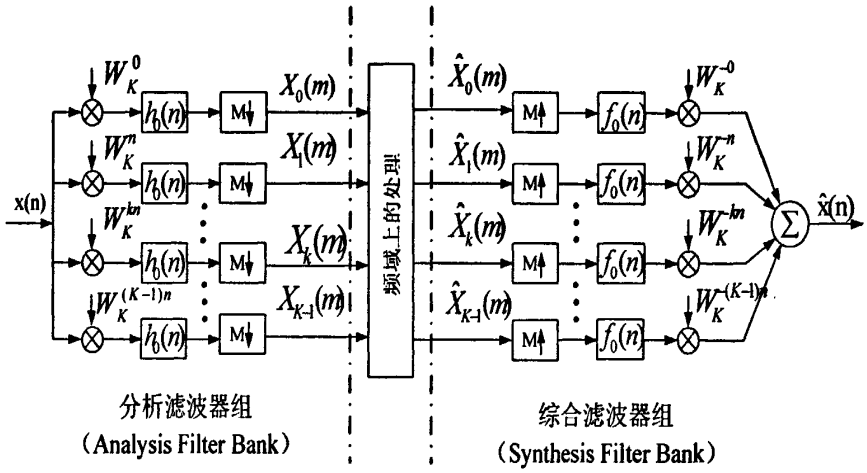


图 3.3　基于多通道的滤波器组

在图 3.3 中，可以看到在设计中在分析过程中，可以利用调制的方式，将各个频段的信号移至低频段，然后通过低通滤波器滤波处理，经过下采样，减小采样率，降低复杂度。以上为分析滤波器。

而在综合部分将处理完成的信号进行上采样，提高采样率，然后通过频率搬移，移到各自的频段范围内，将数字信号相加，即得到了最后处理完成的信号的输出，以上的过程，被称为综合滤波器组。

经过综合以上的分析运算，利用 WOLA 算法能够较为高效的实现整个多通道滤波器的实现。WOLA 结构最初主要是应用于短时傅里叶变换中，将短时傅里叶变换的设计方法应用于信道化的数据接收当中。

在分析阶段，采用输入数据叠乘-衔接的方式，和分析窗进行相乘，得到了短

时内的傅里叶变化，然后进行下采样的数据抽取过程，减少采样率，减少处理的复杂度，然后通过 DFT 运算运算实现。在采用 DFT 运算中，可以利用 FFT 运算的方式实现。然后乘以一个 W_K^{kmM} 量，实现从固定时间域观察角度转向滑动时间域观察角度，实现最后的多通道数据分离。

$$\hat{x}(r+mM)|_{m=m_0} = f_0(r)\frac{1}{K}\sum_{k=0}^{K-1}\hat{X}_k(m)W_K^{-kn}W_K^{-kmM}|_{m=m_0} + (m \neq m_0 \text{项}) \qquad (3.1)$$

而在综合部分，首先，将这些短时变换乘以线性增加的相位因子 W_K^{kmM}，从固定时间框转换为滑动时间框，随后进行傅里叶反变换，产生 K 个抽样的序列，$\hat{x}(m)$，接下来将这个序列周期性地延拓，并用综合窗进行窗处理，为了提高采样率，采用上采样的方式，在每段数据中补足 M 个零。以上分析和综合的算法部分反映的就是加权的叠接相加。

整体说来，WOLA 算法将滤波的卷积简化为乘以窗（滤波器），叠接相加减少了傅里叶变换的点数，可以说大大减少了运算量，为了硬件实现整个 WOLA 算法提供了方便。

3.1.2 宽动态压缩算法

正常人的听觉是在整个频段范围内感受到的声音是没有衰减的，而听障人士，由于其低频部分声音没有衰减，而在高频部分声音信号的感知能力下降，因此在整个助听器设计中需要针对听障人士不同频率范围内的声音进行处理。上文中的多通道频率算法就是将不同频率范围内数据进行分离，而在综合分析部分和综合部分就需要对声音信号进行处理。

由于人耳存在听阈，痛阈，最适阈三个阈值，正常人听阈与痛阈有一定范围，耳听障人士则不同，听障人士的听阈和痛阈很窄。传统的助听器设备，在方法声音信号时，会造成超出痛阈，造成声音非常的刺耳，所以利用听力补偿方法，将声音在不同频率阈内进行压缩，及所谓的宽动态压缩，增强在不同频率下声音信号的压缩比，将信号设计为人耳最适宜的声压级。

在助听器系统中的听力补偿的算法主要是基于宽动态压缩的算法来实现的，该算法是目前最为主要的一种助听器算法。传统的单通道听力补偿算法对信号在整个频率范围内进行统一的压缩放大；而基于多通道分离的听力补偿算法将信号分成多个频带，在不同的频带上进行独立的压缩放大。相比较而言，后者能够更好的拟合患者的听力曲线，满足不同频率的听力需求。

宽动态压缩算法[40]（Wide Dynamic-Range Compression，WDRC）是实现多通道分离然后进行多个通道进行压缩的算法。

宽动态压缩如图 3.4 所示：

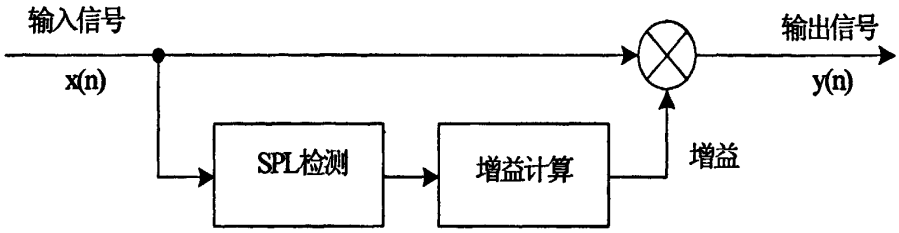


图 3.4 宽动态压缩结构图

而在本文中的助听器实现的算法中，采用的是利用查找表的方法实现了WDRC 的实现，在设计中，当进入的信号经过信号叠接-相加（WOLA）算法处理后，通过多通道的分离，分成多个频率通道内的信号，经过能量计算，然后根据能量计算的结果通过 SRAM 查找表，实现对不同频率范围内的信号强度的增益控制，然后根据增益递归平滑模块，将因为增益值较大的波动的地方进行平滑处理，减少声音的失真，这样实现 WDRC 的整体处理。多通道分离算法的实现模块图如图 3.5 所示。

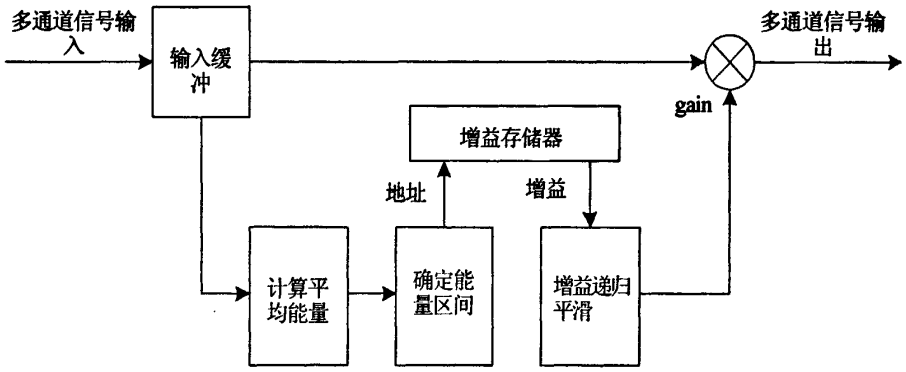


图 3.5 宽动态压缩实现结构图

需要指出的是，在进行宽动态压缩算法中，会涉及到利用平滑的能量直接计算增益值，会涉及指数，对数，除法等运算，始终周期开销太大。所以在下文中，会针对这种特定的运算进行专用模块设计。

3.1.3 噪声消除算法

噪声消除算法中最为常用的一种算法是功率谱想减算法。出发点是利用语音信号的短时平稳性，从带噪声语音的短时功率谱估计值中减去噪声短时功率谱，从而得到较为纯净的语音的频谱，达到噪声消除的目的。

3.2 关键算法指令

在助听器算法中，主要包括三类算法，分别是多通道分离算法，宽动态压缩算法，噪声消除算法。

多通道分离算法是在主函数中实现的，由于宽动态压缩和噪声消除算法是基于多通道分离实现的，所以在设计中以多通道分离算法作为主函数，将宽动态压缩算法和噪声消除算法作为函数进行分别调用。

3.2.1 主函数设计

主函数中主要进行了调用函数的初始化设计，由于多通道分离算法中，有相关的插值运算，窗运算等，因此有相关的卷积指令具体设计结构如下图 3.6 所示。

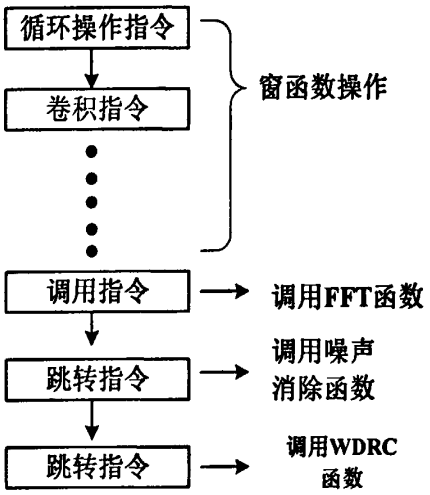


图 3.6 多通道分离算法指令

本运算中多采用调用模块，在采用关键算法设计时，采用循环指令循环操作，减少了程序存储器的长度。主函数中有 100 条指令，其中多通道分离占据了 80%的指令比例。在设置中以设置函数（set）函数和数据传输(move)类指令为主，使用 repeat 指令，分别完成了数据输入控制，数据输出控制，增益控制(用于 WDRC)，窗函数设置等一系列的重复操作，由于算法中牵扯了 32 通道的数据分离，因此，数据的重读次数以 32 次，16 次为主。

3.2.2 WDRC 指令设计

在 WDRC 中，由于需要对虚部数据和实部数据分别计算操作，因此在该函数模块中，多以双数据操作为主，例如双读取操作.双写入操作，乘法操作，等一系列的乘法操作为主。其中，运算涉及了加法操作，乘法操作，减法操作，求整操作。

3.2.3 消噪运算指令设计

噪声消除算法运算较为复杂，并且没有太多的循环操作。该算法中，涉及到了众多的指数运算，对数运算，开方运算。所以包含了很多的指数等指令。在噪声消除算法中，大多以功率谱计算很多，乘法运算指令和开方运算指令较多。

3.3 助听器算法的指令程序

整个处理器在处理助听器的指令时是按照整个存储空间进行划分的，

程序存储器指令结构划分是按照如下方式，前面 100 条指令是跳转指令和相关跳转执行指令，这种指令设计形式和 51 处理器的设计方式相似，在这部分中可以添加指令，将需要执行操作的指令跳转到这个代码段中，在处理器初始化操作中 SPI 的初始化操作就是在这部分代码段中执行。图 3.7 中是中断指令段。

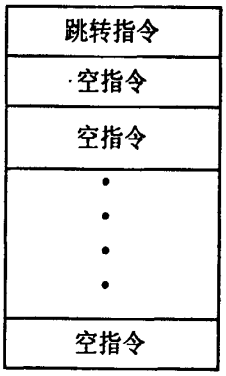


图 3.7 程序存储器中断指令段形式

然后进行 initial 代码处理阶段，实际上这部分也可看做是中断操作的一部分，这部分操作是在执行 main 操作时首先跳转到的部分。图 3.8 是相关初始化配置指令模块示意图。

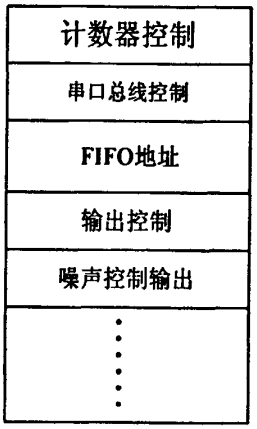


图 3.8 初始配置指令模块

这些处理主要是进行串口的初始化，对于数据接收部分，主要是进行端口部

分的 FIFO 口的地址处理，外部端口的设计处理，还有一些特殊处理单元的模块化处理设计，例如在外部的 WDRC 模块设计中的地址初始化设计，另外还有噪声估计部分的初始化设计。

随后的部分是进入主操作程序，这部分是进行语音处理设计的主要部分。包括了许多 FIFO 操作，例如在进行 WOLA 取数时，从外部 A/D 输入的数据进入到寄存器中，需要 FIFO 地址通过专用的寄存器逐个的将数据读入。关于调用 FFT 子程序和相关的消噪程序的设计都是在该部分进行编写。图 3.9 是主程序控制段结构图。

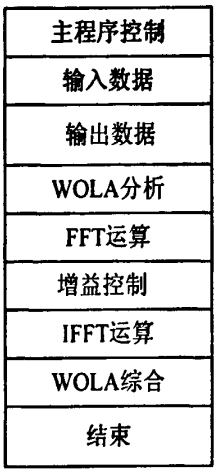


图 3.9　主程序指令模块分布结构图

图 3.9 中的执行顺序正好是进行助听器算法中最为重要的算法设计，按照原理分析可以看做在进行算法操作中首先将数据的输入地址和输出地址设置好，然后进行 WOLA 分析的操作设计，接着进行 FFT 操作，将声音信号设计成为了多通道频域信号后，采用宽动态压缩的设计方法，分别对不同的频域量乘以一个增益，然后进行 IFFT 操作，然后是 WOLA 的综合部分，最后是 main 程序的结束。

下一部分就是从 main 主程序中调用的子程序了，例如消噪模块程序，WDRC 程序和 FFT 程序，这部分是在每一个代码段中执行的，每一个操作代码段和另外的操作代码段之间是有一定的执行空间差别的。

图 3.10 中就是各个指令操作中的代码段部分，各个不同的操作指令是按照每一段的形式执行操作的，并且在执行的过程中都会包含返回（ret）指令，对这些指令的操作是采用调用（call）指令进行调取实现的。

从图 3.10 的代码中可以看到，整个设计中相关的所有的计算都是在处理器中处理完成的。优化设计空间还是比较大的。例如在进行具体的 FFT 操作中，对于蝶形运算的操作就是一个比较繁琐的问题，需要频繁的调用寄存器模块和相关的存储器，这样需要非常多的时钟周期，乘法运算占据大量的运算资源，因此在设计中需要额外的对这个频繁使用模块进行单独的模块化设计，这就是 butterfly 模块单独设计的初衷。

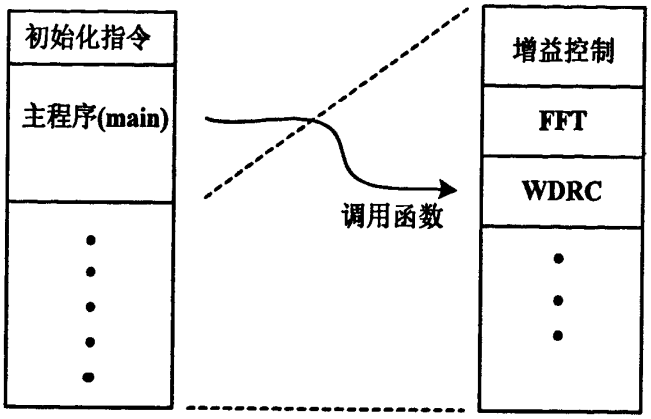


图 3.10 嵌套指令分布结构图

3.4 专用指令设计及其模块设计方法

在专用指令集处理器中，由于处理器的硬件结构不是很复杂，ASIP 结构是以硬件描述语言进行描述的，所以在设计中，可以添加额外的模块优化设计。在 RISC 结构模块中，基本的模块单元是和上面章节描述相一致的，以控制路径，数据运算路径，数据存储为主要结构，在 DSP 运算中，常使用的卷积运算单元，乘累加运算单元，都是在原有的硬件结构基础上添加的，所以由于 ASIP 设计面对的是硬件描述语言进行，所以为硬件加速设计和模块设计提供了方便。

ASIP 的专用模块设计基于固定的算法，有很强在针对性。例如在语音处理中，将众多算法分解，包含了大量的复数运算，对数运算，指数运算。这些运算在使用精简 ASIP 实现时，会耗费大量的指令集和功耗。因此为了提高系统的性能，对特定大量的运算进行模块设计是相当必要的。另外有些运算例如对数运算，这类运算利用加法器，乘法器运算会有很大的误差，在硬件设计中，会经常使用查找表等方法实现该种运算。

在 ASIP 处理器中，为了添加额外的指令，首先需要在 ID 一级中添加相关的对照的解析指令。在硬件描述语言中，ID 一级多用 case 语句来完成，添加相关的搜索项就可以有相关指令的解析。如图 3.11 所示。

图 3.11 中可以看到，通过添加搜索的项，在输出给出相关的控制项，在后面为了配合相关流水线级数的逻辑设计和满足时序的要求，添加相关的控制逻辑信号，然后就是专用的指令集设计模块了，在专用指令模块中，设计方式和普通的 ASIC 设计方式相似，模块的触发采用单线信号来完成，具体的细节出下面章节描述。

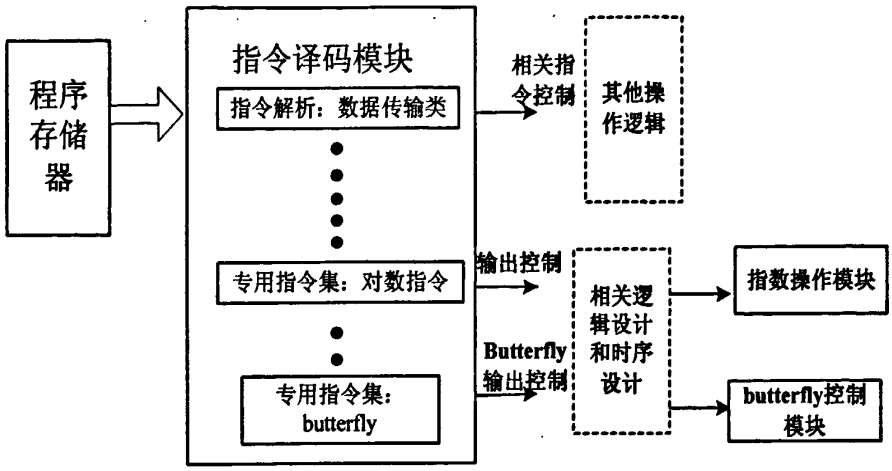


图 3.11 指令译码模块控制操作

3.5 对数，开方等运算类专用指令模块设计

在助听器处理器设计中，助听器算法包含了多通道分离（WOLA），宽动态压缩（WDRC）运算，噪声消除算法。

对噪声消除模块进行运算类型的分析，可以得到如表 3.1 中的结论。

表 3.1 消噪运算中运算所占比例

操作	运行周期	消噪百分比 100%	AIDS100%
消躁模块不包含 log Recipe sqrt 运算	4060	55.1	23.3
Log 运算	1216	15.3	7.0
Recipe 运算	1200	15.1	6.9
Sqrt 运算	1472	18.5	8.5

表 3.1 中显示得到，在消噪运算中，指数运算，对数运算，开放运算占据了非常大的比例。为了简化指令操作的复杂度，并且减少时序开销和功耗开销，因此对这些模块进行加速指令模块的设计。

在 DSP 设计中，为了实现提高数据的处理运算速度，减小功耗，在处理器核中，添加了相关的对数运算（log），开方运算(sqrt),倒数运算(recip)模块。如果按照常规的利用 ALU 或者 MAC 等模块设计方法来进行计算的话，这部分设计耗费大量的时钟周期，而如果采用专门定制的模块进行设计的话，可以在时钟周期和功耗上面都有所提高，通过验证可以知道，利用上面提到的设计模块，复杂的对数运算或者开放运算仅仅需要两个时钟周期。

以求对数模运算 $\log_2^x$ 为例。在进行求对数运算中，采用查找表的方式实现了

对对数运算的快速处理，这种方法避免了繁琐的化简和近似运算，并且很好的解决了进行对数运算时钟周期长的问题。不过缺点是，在设计中由于引入了查找表或者存储器电路模块，这些运算单元无法进行模块复用，所以该模块会在使用效率上不及通用模块，并且占用额外的芯片面积。但是整体而言，由于消噪运算中大量的指数或者对数运算存在，平衡考虑，采用该加速和改进模块是有利处的。

在对对数 $\log_2^x$ 的运算中，将对数 $\log_2^x$ 进行了处理，将其分解成为了两部分：$\log_2^{10}$ 和 $\log_{10}^x$。由于在设计中，数据是采用的定点运算，数据小于 1，因此数据会首先进行定点化处理，为了减少电路规模，在设计中会采用按照高位寻址的设计方式，通过高位确定权值，然后再截取高位后四位的地址，在下一级的译码模块中输入地址寻址，在预先设计好的查找表中寻到合适的结果，将权值和结果相加，得到最后 $\log_2^x$ 的值。

在对数运算电路中包含有两个模块，如图 3.12 所示，两个模块将整个对数的运算分解为了两部分，一部分是进行整体数据的截取，另外一部分是进行数据查找表设计。如果直接采用查找表的设计方法，需要内部较大的查找表的面积开销，如果采用分级处理，一方面可以减少内部模块面积开销，另外整体的精度也是可以保证的。以下是求对数模块电路图：

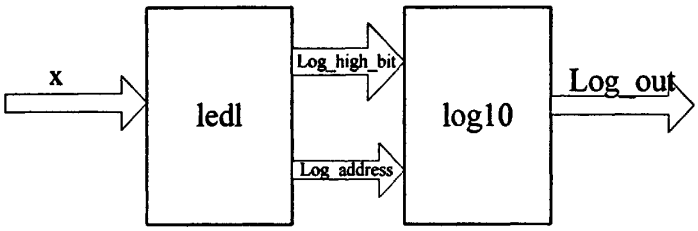


图 3.12　对数运算模块

上面的 x 信号是输入的信号，然后在 ledl 中进行判断，来判断哪一位不为 0 是最高位，然后再对信号数据进行截取，在截取中采用的方法是仅仅截取最高位不为 0 的 6 位，实现较短的处理，以便于节省功耗。

运算过程：将输入信号的数据写入到 ledl 模块中，然后将数据判断最高位的数据，然后判断输出的数据一个输出权位，另一个输出高六位的数据输出，这样实现了对输入数据的分解过程，然后将数据写入到 log10 模块中，通过查表的方法得到除最高位后的尾数的数据，进行数据的叠位相加。为了实现低功耗的设计，将最高位的数据省去，因为最高位通过 ledl 的截取，必定是可以节省 log10 模块中的查表大小。

在求开方，求倒数模块中同样也是采用的这种方法实现，所以这种设计方法非常的有效，减少了芯片的设计复杂度，在精度上，也比采用 MAC 和 ALU 模块精确。

对数运算有两种运算指令，opcode 分为两种。这两种 opcode 主要是进行了

两种运算，一种运算是进行累加器的对数运算，另外一种是进行直接对寄存器的对数运算。如图 3.13 所示。

op code	寄存器	累加器	空白填充

图 3.13 对数运算的指令格式

图 3.13 中所示，在进行专用的结构设计时，通过查询不同的 opcode 模式，得到在 opcode 为确定值以外的值时，这段代码是没有被解析，在处理器结构的 ID 一级添加了一个译码单元，将结果进行了译码处理。图 3.13 中格式是代码段，然后是寄存器地址，然后是对应的累加器地址。

至于开方，求余等运算的指令操作格式和上面的格式相似，在这里就不一一说明，具体的格式代码如表 3.2 所示。

表 3.2 特殊运算指令格式

运算类别	指令说明
对数 1	log10 r19,acr2
对数 2	log10 r13,r10:r9
开方 1	sqrt r13,acr1
开方 2	sqrt r13,acr0
求倒 1	recipe r13,r0:r13
求倒 2	recipe r13,acr1

为了实现对模块操作，特别设计了相关的求对数，开方，求倒的指令。在计算中，各个模块分别对普通寄存器和立即数寄存器进行设计的。

3.6 基于 ASIP 设计方式的 Butterfly 运算设计

表 3.3 中是相关的助听器运算指令所占对比情况。

表 3.3 助听器算法中各个运算所占比例

操作	运行周期	100%
数据读入（IN）	32	0.2
数据输出（OUT）	64	0.4
多通道分离分析	576	3.3
噪声消除	7948	47.5（热点程序）
宽动态压缩	1397	8.1
多通道分离综合	1344	7.7
(I) FFT	6042	34.8（热点程序）
合计	17403	100

除了上文中提到的加速运算类指令外，还有额外的大数量的数据运算，例如在助听器算法中，占据较大比例的噪声消除，WOLA 运算等。

表 3.3 中显示的是各个算法中在整个处理器运算中所占的比例，从上表中可以看到除了上文中提到的噪声消除外，FFT 占据了很大的比例。所以优化 FFT 运算成为了 ASIP 设计另外一个设计专用模块的方向。专用蝶形运算的相关操作如图 3.14 所示。

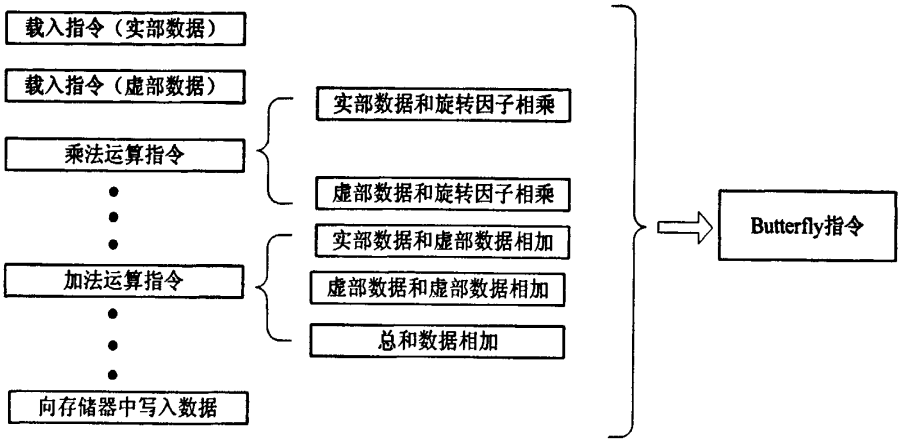


图 3.14　蝶形运算相关操作

通过对比指令运算可以看到图 3.14 所示，在 FFT 运算中，对蝶形运算的操作需要多个指令相互配合，在图 3.14 中可以看到，要完成一个蝶形运算，需要有载入指令，乘法指令，加法指令，输出指令等，共需要有 16 条指令。因此在这个运行中，最少需要 16 个时钟周期，而如果将 butterfly 指令设计成为专用指令，则仅仅需要 4 个时钟周期，FFT 处理的时间，并且减少了整体运行中处理器所耗费的功耗。

蝶形运算是整个 FFT 运算中，较为核心的一类运算形式。运算公式如下式（3.2）所示：

$$\left.\begin{aligned} x_{m+1}(p) &= x_m(p) + W_N^r x_m(q) \\ x_{m+1}(q) &= x_m(p) - W_N^r x_m(q) \end{aligned}\right\} \tag{3.2}$$

上式中 x_m 和 x_{m+1} 分别是进行蝶形运算中的输入数据和输出数据，p 和 q 分别代表的是不同的两个输入模块数据，而 W_N^r 旋转因子。上式中包含了输入数据的实部运算和虚部运算，完整的将数据展开可以看到，整个数据的运算按照如下形式：

$$\left.\begin{aligned} real(x_{m+1}(p)) &= real(x_m(p)) + real(x_m(q))\cdot real(W_N^r) + imag(x_m(q))\cdot imag(W_N^r) \\ imag(x_{m+1}(p)) &= imag(x_m(p)) + imag(x_m(q))\cdot real(W_N^r) - real(x_m(q))\cdot imag(W_N^r) \\ real(x_{m+1}(q)) &= real(x_m(p)) - real(x_m(q))\cdot real(W_N^r) - imag(x_m(q))\cdot imag(W_N^r) \\ imag(x_{m+1}(q)) &= imag(x_m(p)) - imag(x_m(q))\cdot real(W_N^r) + real(x_m(q))\cdot imag(W_N^r) \end{aligned}\right\} \tag{3.3}$$

式（3.3）中的“real”表示实部部分，而“imag”表示虚部部分。从式（3.3）中可以看到，需要进行四次乘法运算，另外进行四次三个项的加法运算，单纯依

靠乘累加运算，会需要大量的时钟周期，并且会有大量的流水线功耗。因此本着提高性能和减少功耗的目的，将上面蝶形运算设计成为了固定的运算核模块，利用 ID 输出的指令对该模块触发实现。

Butterfly 模块的硬件设计：在 Butterfly 模块中有 7 个端口，这些端口主要包括一个控制端口，四个数据的输入和输出端口，另外两个端口是数据的输出端口。在设计中将设计控制端口 ctrl_i 作为一个设计的控制调度，通过将控制字逐位的移位实现过程控制的实现。如图 3.15 所示。在控制中有 ctrl_i 中有以下时钟节拍：ctrl_i，ctrl_p4， ctrl_p5，ctrl_p6 ，ctrl_p7，ctrl_p8 每过一个时钟，向下走一个时钟节拍。上面的节拍设计是这样的，ctrl_i 和 ctrl_p4 为数据的加载，ctrl_p6 ctrl_p7 为数据的存储。

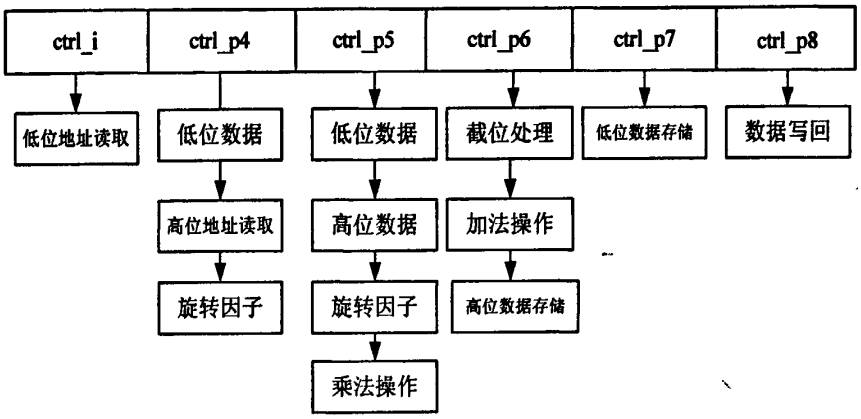


图 3.15 指令节拍运算模式进程图

旋转因子的加载是在 ctrl_i，而数据的加载是在 ctrl_p5。由于在 ctrl_p4 部分会将地址输出，然后在下一个时钟节拍将数据得到，在 ctrl_p5 会读取到数据。在 ctrl_p4 部分将旋转因子通过另外一级寄存器寄存到目的寄存器中。P5 时间数据加载完成后，就进行乘法运算处理。组合逻辑电路设计。截位处理和加法处理同时是在 P6 时间段完成。这段时间主要是先进行乘法运算后的截断处理，然后再进行加法操作。

具体的操作步骤流程如上图 3.15 所示，在过程流程控制中，简单的顺序执行采用移位操作的步骤，通过多个寄存器的中转控制，实现按顺序执行的操作。

图 3.16 是整个 butterfly 设计模块，从模块设计中可以看到输入信号由旋转因子和数据输入信号，而输出中有写入到数据存储器的数据，这样通过每个模块的协同操作，可以实现整个 butterfly 模式的工作。

在专用的 butterfly 模块中，模块中的设计需要按照专用指令的设计实现，由于处理器模块与专用指令集模块不同，因此在设计中不可能按照专用指令结构方法设计，如上面文中提到的，需要增加额外的时序同步电路模块，来完善整个电路结构形式。

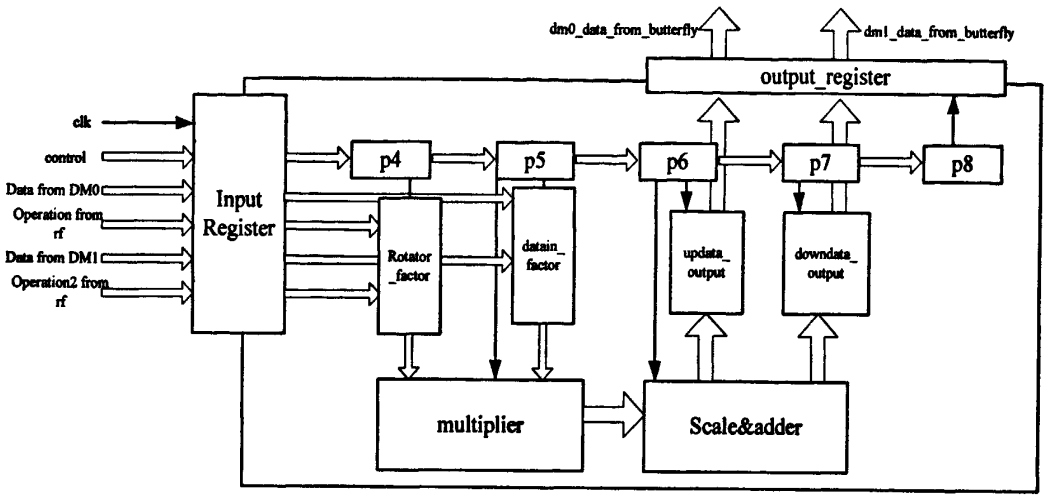


图 3.16 butterfly 模块设计结构示意图

整体 butterfly 模块和整个 DSP 设计中结构模式图如图 3.17 所示：

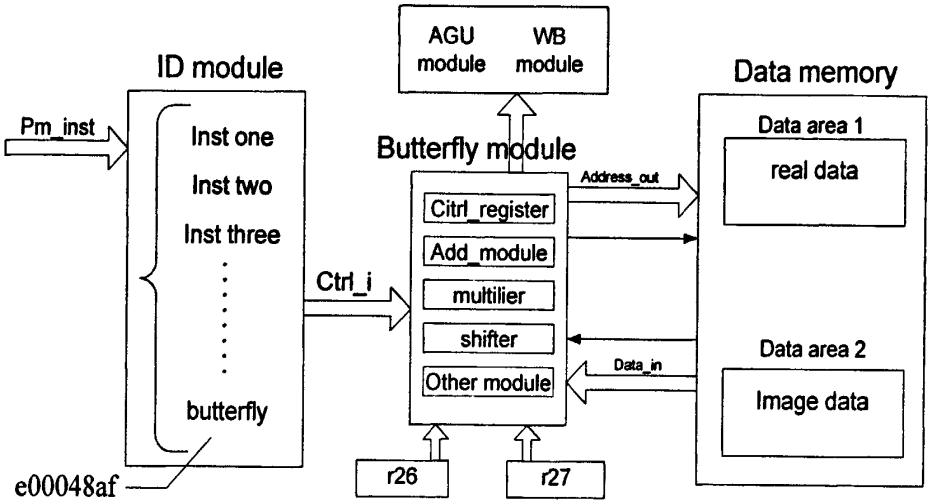


图 3.17 butterfly 模块和处理器模块连接关系图

从上面的图中可以看到，添加的 butterfly 模块作为 ASIP 设计的模块，添加的模块放置在了 ID 模块后面。在上图中，指令的译码是按照 ID code 进行译码操作的。在译码选项中，添加了 butterfly 的译码操作，加入了某字符串为 butterfly 指令的译码指令项。然后 butterfly 模块和 ID 模块的接口是通过 ctrl_i 模块连接在一起的，在 ctrl_i 中包含了很多 butterfly 信息，包括流水技术控制和相关数据读取，但是由于 butterfly 结构模式较为固定，因此在设计中，将 ctrl_i 设计成为了一个固定的数据段。Butterfly 模块和其他数据的交换是通过与 data_memory 完成的，这个结构是使用了 data_memory 中的两个模块作为输入和输出数据流，下端的数据的输入控制采用了 r26 和 r27 作为数据的转换控制。

在进行 FFT 运算中，需要操作指令给予 4 个时钟周期的空操作，即在运行完 butterfly 模块后，添加四个 nop 指令。让整个处理器的其他模块等待该模块，在

各级的流水线部分不工作，没有数据流。

Butterfly 模块的默认操作是在 r26 和 r27 模块中的，并且蝶形运算的实部存在 r27 中，虚部存在 r26 中。在进行地址译码过程中会有将高位运算和低位运算进行分类，蝶形运算的缩放比例位的数据。

Butterfly 指令的指令格式如图 3.18 所示：

op code	填充	缩放比例	有符号/无符号

图 3.18 butterfly 代码的格式

图 3.18 中就是 butterfly 模块指令的形式。包括数据的定点化设计需要，数据有符号数和无符号数的设计方式实现。

图 3.19，3.20 和 3.21 是 butterfly 指令的波形示意图。

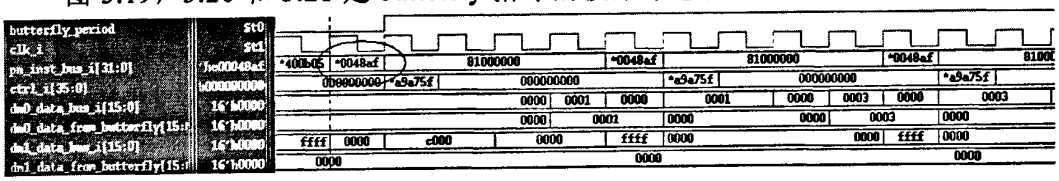


图 3.19 butterfly 指令波形示意图

图上黑圈部分是关于 pm_inst_bus_i 信号，这个信号主要是进行 butterfly 模块的启动，信号的输出的主要是指令。这个指令预先存储在程序存储器中。当检测到该指令后，butterfly 模块开始正常工作，butterfly_period 信号正式打开，被置为高电平。从上面图中可以看出，每次的 butterfly 操作总是持续 5 个时钟周期的。

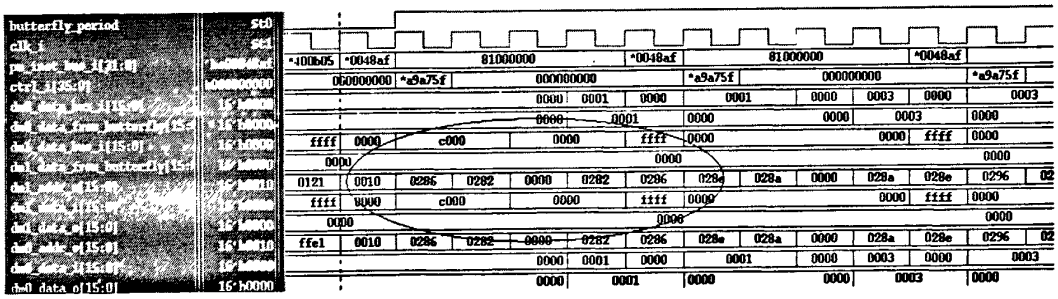


图 3.20 butterfly 指令波形示意图

图 3.20 中黑圈部分是从数据存储器中读取数据得到的数据波形图，该 butterfly 模块分别连接了各个不同存储器电路模块，地址生成模块电路。上面的四个 81000000 信号表示的在运行完 butterfly 指令后等待的空闲信号。

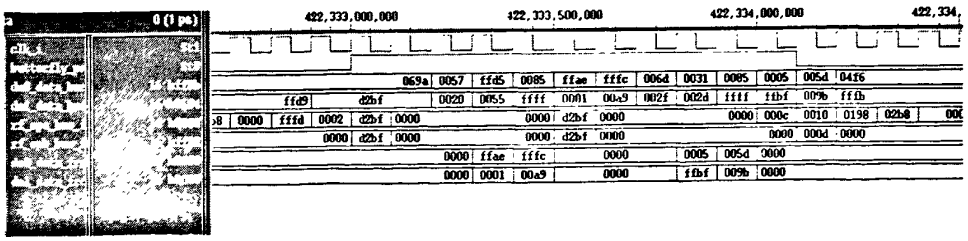


图 3.21 处理器运行时 butterfly 模块输出示意图

上图 3.21 中就是 butterfly 模块中在整个处理器中运行的设计波形图，从图 3.21 中可以看到在整个设计中，当 butterfly_period 为高时，这个模块数据从寄存器中输入，然后相关的旋转因子从模块输入，得到的数据会从 buffer 中输出，输出到相关的数据存储器中。一共运行了两次，产生了两个 butterfly 运算的输出。

通过仿真验证对比 3.22 和 3.23 所示。

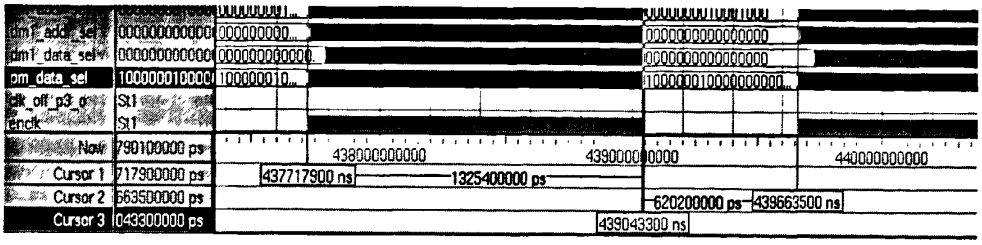


图 3.22 语音处理器周期波形图 1

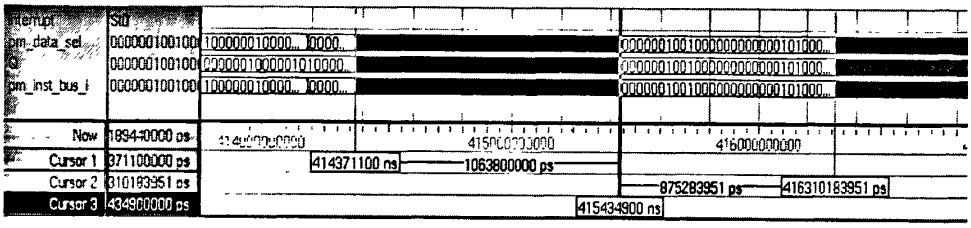


图 3.23 语音处理器周期波形图 2

图 3.22 是没有采用蝶形运算模块的处理器，而图 3.23 是采用蝶形运算模块的处理器电路。在设计中采用专用加速单元后，处理器的整体工作周期不变，而在完成特定数据处理的时间加快，其中图 3.22 中数据处理时间占总时间的 68.1%，而在图 3.23 中处理时间占总时间的 54.8%。可以看出数据处理时间大大减少。在外部 D/A 采样率和时钟频率一定的情况下，通过采用加速模块，可以提高处理速度，减少功耗。

通过功耗仿真，在未使用 butterfly 指令模块时，动态功耗为 828.8063 μW，在使用了 butterfly 指令后，动态功耗为 722.2014 μW，降低功耗 12.8%，功耗降低效果明显。

综上所述，在原有的处理器结构中，为了提升数据处理的速度和减少数据运算时功耗，需要设计专用的指令集和专用的指令运行模块，上文中介绍的 butterfly 模块是在助听器设计中众多专用指令模块中的一个。正是在添加了各种各样的优化模块，才使得在芯片的面积和芯片的效率方面有了很好的平衡。通过添加专用定制模块，在特定设计领域，减少灵活度，提升系统性能，减少系统功耗，这种优化处理器的设计思想正是 ASIP 设计的优点所在。

3.7 小结

本章介绍了相关助听器的基本算法：多通道分离算法和宽动态压缩算法，其

中算法中包含了大量的指数运算，对数运算，复数运算操作，另外 FFT 运算在整个助听器算法中占据了很大的比例，最后本章介绍了助听器算法转化为汇编指令，如何在 ASIP 处理器中实现的。在分析了助听器设计中频繁设计的相关算法和操作的基础上，采用 ASIP 所特有的专用指令设计方法，对处理器进行了设计改进。由于助听器设计中，需要进行噪声消除的设计，因此在指令设计中，将其频繁使用的对数运算，开方运算等设计成了专用模块，利用该模块完成了普通乘法器和加法器难以完成的运算。而后对蝶形运算模块进行了介绍，由于助听器算法中设计到了大量的 FFT 运算，而蝶形运算又是 FFT 运算的核心运算，所以为了加快运算速度，降低数据处理中间所耗费的功耗，决定利用专用指令模块的设计方法实现。通过设计 butterfly 模块，实现了从原有 16 个指令周期完成的运算缩减到了 4 个指令周期，大大节省了时钟周期，减少了整个处理器的运算功耗耗费。

第 4 章 程序存储器低功耗优化设计

在处理器设计过程中，各个部分，模块之间占据的比例都是不一样的。主要的逻辑控制部分，占据的面积极小，而在主要的数据存储部分，占据的面积却非常大。在目前的低功耗处理器设计中，存储器的低功耗设计已经成为了研究热点，并且趋向于层次化设计的特点。

4.1 存储器设计分析

存储器在处理器中有着非常重要的作用，一方面实现数据的传输，交换，另外一方面进行整个处理器的控制。由于存储器采用了特殊的电路结构，因此在进行低功耗设计中，存储器需要有特殊的优化方法。一般而言，存储器的大小和处理器的规模有关，如果运行的程序操作很长，很多，则整个程序存储器的面积就较大。一般而言，处理器中的存储器占据了整个处理器中 50%~60%的面积开销。当存储器在频繁的读取过程中功耗会非常大，而且存储器的静态功耗也是一个非常大的功耗开销。所以合理的使用存储器模块，合理的对存储器进行优化设计，是处理器中存储器低功耗设计中极为重要的一步。

4.1.1 存储器面积结构

在处理器设计中，存储器一般至少为三块，一块程序存储器 PM（program memory），两块数据存储器 DM（data memory）。其中程序存储器 PM 主要是存储程序代码段，并且进行程序运行控制使用，在本次设计的处理器中，程序存储器的位宽定为了 32 位。另外是数据存储器，数据存储器主要是进行数据的计算和交换作用，由于中间数据的运算是在 24 位的，因此数据存储器的数据宽度定在了 24 位。

图 2.1 中可以看到 PM 属于路径控制，DM1(数据存储器 1)和 DM2（数据存储器 2）和数据路径有关。当数据进行正常的工作时，每一个时钟周期对会对程序存储器 PM 和数据存储器 DM 进行数据的读取和写入。

通过对整个处理器进行综合可以看到，在整个处理器设计中各个部分的面积列表如下所示，从上面的列表 4.1 中可以看出，占据整个处理器最为重要的部分是存储器系统模块，占据了近 60%的面积开销。其次是专门设计的加速模块 butterfly 模块和输入和输出的缓冲模块。所以针对存储器在面积上合功耗上面占据重要的地位，采取合理的措施来进行低功耗设计具有非常重要的意义。

表 4.1 处理器各部分功耗比重表

模块结构	总面积值	百分比
总 ASIP 核	697826.9375	100%
初始载入模块	8222.1836	1.17%
ALU 模块	11723.9385	1.68%
MAC 模块	44131.9648	6.32%
AGU 模块	17968.6895	2.6%
Butterfly 模块	69141.6250	9.9%
数据前插模块	835.1204	0.1%
指令译码模块	17817.4648	2.6%
循环计数模块	5614.9927	0.8%
程序计数器状态机	1113.4944	0.2%
程序计数器模块	2074.2227	0.3%
寄存器堆模块	35820.1797	5.1%
输入输出缓冲模块	57059.1328	8.2%
存储器系统	415705.4062	59.6%

4.1.2 存储器内部电路结构

存储器电路作为一个固定的 IP 核，在进行设计和调用过程中可以被看成是一个整体。对存储器电路内部进行修改，就需要对电路内的结构进行分析，修改电路结构，或者针对工艺条件进行改进。

存储器的结构图如上图 4.1 所示。

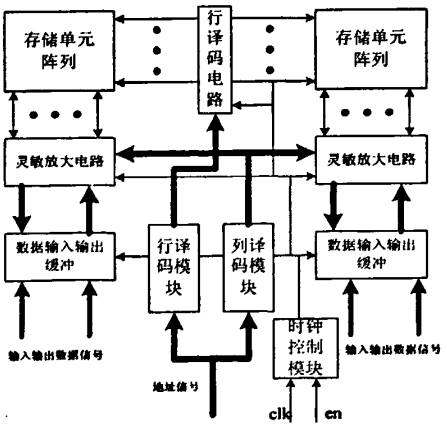


图 4.1 存储器电路内部结构图

在该图中可以看出，存储器电路的基本结构是有存储单元阵列模块(sram_cell)，灵敏放大模块（sensor amplifer）,译码器模块（decoder），输入和输

出模块（In-out-buffer）。这些模块构成了整个存储器的模块设计。各个模块的功能不同，但是在整个存储器使用过程中，是被看做一体的。

1）六管单元电路

目前对存储器电路的描述语言建模是采用寄存器堆的形式来实现的，但是这种设计方法是不能代表存储器电路的内部结构。寄存器堆结构主要是以 flip-flop 结构为主，即以寄存器为主。而在存储器电路中，存储单元的结构主要以六管单元为主[21]。图 4.2 是六管单元的结构图：

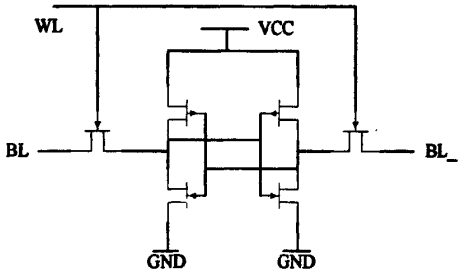


图 4.2 六管单元结构图

图 4.2 中可以看出在标准的六管单元为存储单元的四个晶体管，两个 PMOS 管和两个 NMOS 管。然后是两个读取晶体管，分别连接了位线，另外是读写晶体管的栅极连接了字线。

六管单元进行存储数据的原理是通过两个反相器的互锁相应构成了整个存储的机理，通过输入信号到达阈值电压时，得到输出的翻转。如图 4.3 所示：

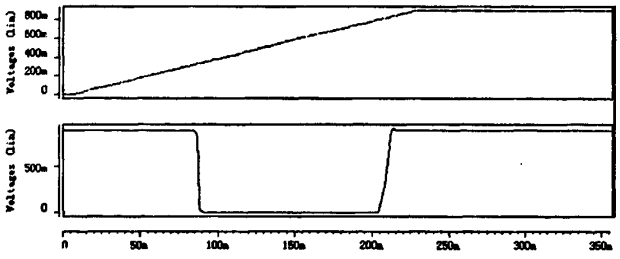


图 4.3 六管单元波形翻转图

core1 和 core2 构成两个互补的电位节点，然后存储该位的信号值。该位的信号值可以通过读取信号进行读取出来。

2）灵敏放大器电路

图 4.4 中的灵敏放大器就是在存储器电路中经常使用的锁存型灵敏放大器。该灵敏放大器电路主要是采用了两个反相器进行互锁，当一个较小的微弱的信号到来时，可以在非常快的时间内达到放大的目的，迅速将电平信号抬高到所需要的电平值。利用快速提高电路中电平信号的反转速度，来完成对中间电路状态的转化，这种设计需要在电路版图和电路结构上面进行精细的分析和设计。

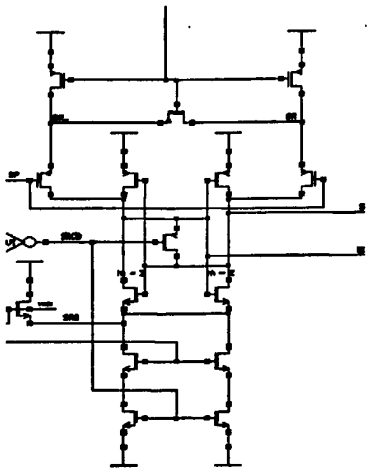


图 4.4 灵敏放大器电路

在该电路中上面的三个晶体管是预充电电路，对存储单元有非常好的灵敏放大作用的四个晶体管是中间四个构成锁存功能的四个晶体管，另外在四个晶体管中中间有一个晶体管，该管是平衡管，由时钟信号 GTP 来控制。下面的四个晶体管是用来对灵敏放大器进行控制的单元。图 4.5 是灵敏放大电路结构形式。

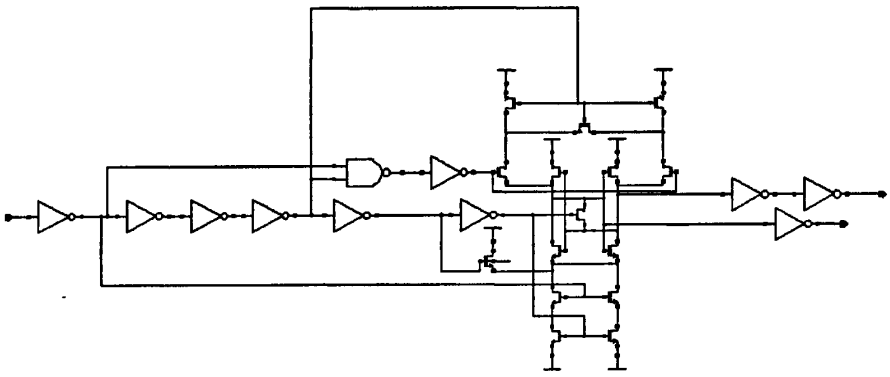


图 4.5 灵敏放大器电路整体电路图

如图 4.5 中所示就是全部的灵敏放大器电路设计，在电路前面有一串反相器链，这部分电路构成了一个反相器的延时单元，在某一个静态时刻，在整体的锁存型灵敏放大电路中是没有静态电流通过的，但是当出现 GTP 电平变化时，则会出现一个电平变化的延时，当这个延时传送到下一级时，会有一个非常小的脉冲时间内将灵敏放大器工作，这个就是整个电路工作的原理，可以看到这个设计中非常重要的一点就是利用了灵敏放大器延时的特性，利用传输门的延迟来实现整个灵敏放大器电路的低功耗设计。在每一个时钟到来的时刻，在信号进行翻转时，会有 1ns 的时间，整个灵敏放大器是导通的。

3）充电单元电路

在灵敏放大器电路和存储单元部分有一种电路模块，在整体设计存储器电路中，该充电和放电模块起到了非常好的连接灵敏放大电路和存储单元电路的设计部分，因此在电路的版图设计中，这部分电路就是位线网络和字线网络。在整体

考虑存储器功耗的问题上，这部分电路占据很大一部分功耗，所以首先对该部分电路进行分析。

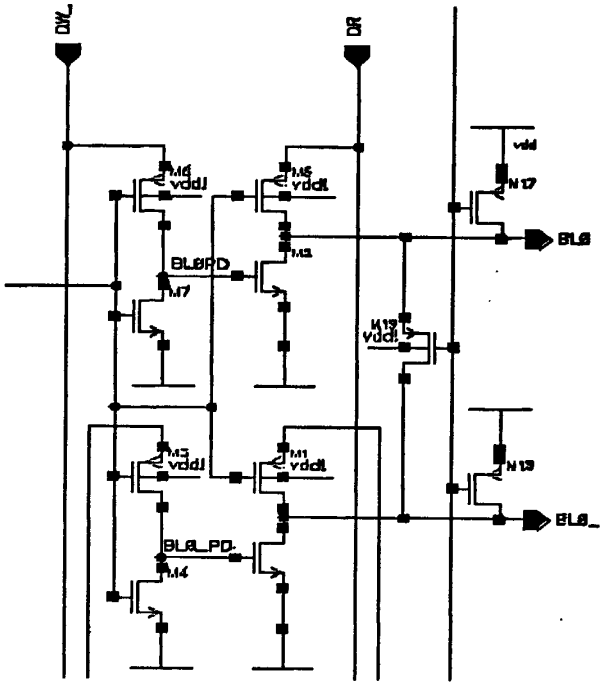


图 4.6　充电单元电路电路图

图 4.6 所示就是充电电路，利用一系列的上拉晶体管和下拉晶体管，形成了一系列的充电电路和放电电路，当时钟窄信号到来的时候，该电路可以根据脉冲信号快速的实现电路的充电过程。

4）译码电路

在存储器电路中，译码电路是非常重要的电路结构，该电路可以实现对整个存储器地址的译码寻找，由于电路采用的是纯组合逻辑电路，因此该电路的动态功耗非常的低。并且随着电路规模的增大，这部分电路的功耗所占比重会比较小。

5）时钟控制电路

这个电路在以往的存储器电路设计中不是重要电路，但是在低功耗设计的存储器电路中，这部分电路有非常重要的意义。在存储器电路中，电路功耗的高低由时钟电路控制，在进行设计时，如果能够将时钟信号的脉冲变小，则电路的功耗就会非常的低。设计方法是采用利用电路反馈效应实现整个电路的信号自关断，将一个宽脉冲的时钟信号调整成为一个窄脉冲信号，脉宽仅仅是 1ns 的时长。这种设计方法有效的减少功耗值。

以上各个部分就是存储器电路的各个单元。在存储器正常工作时，当控制使能开启时，则数据线上的信号存储在存储单元区域内，当控制使能信号关闭时，则信号输出地址信号所指的存储单元区域。当写使能信号关闭时，则会进行读操作，这样在输出端会得到地址所指的存储单元的输出值。

图 4.7 和图 4.8 是存储器读取时序图和写入时序图[43]

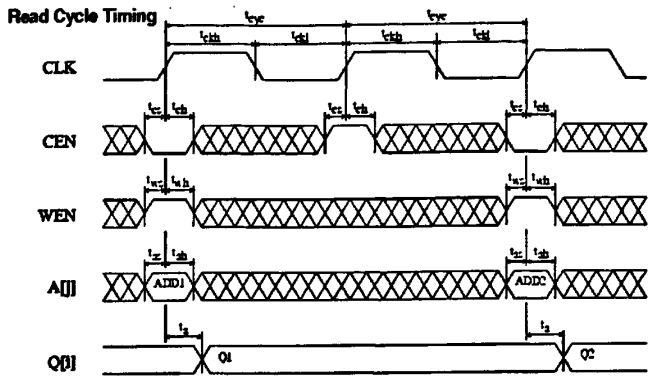


图 4.7 存储器电路读取时序图

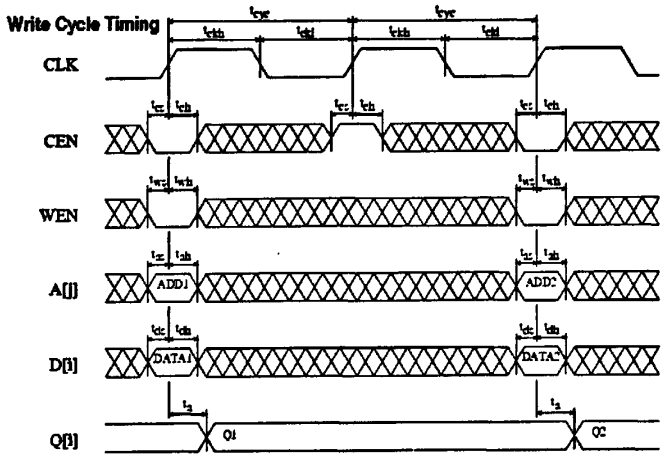


图 4.8 存储器电路写出时序图

4.1.3 存储器电路功耗分析

通过利用 HSPICE 软件对存储器电路进行精确的功耗分析，对 Atisan[42]生成的 CDL 网表编写激励文件，经过仿真得到图 4.9 和 4.10 柱状图。

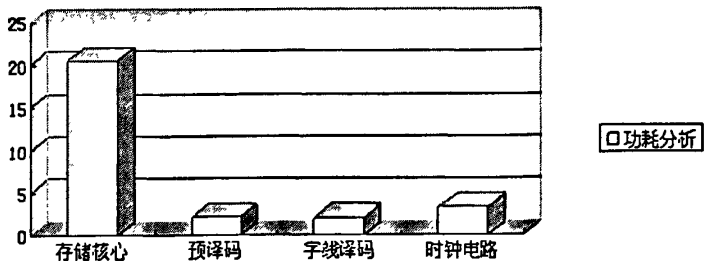


图 4.9 存储器电路功耗情况对比

在存储核心部分包括了灵敏放大电路和位线和字线充电电路。则内部功耗的功耗对比为：

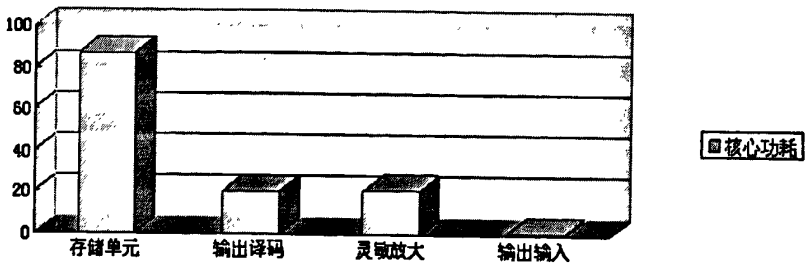


图 4.10 存储器电路功耗情况对比 2

对比图 4.9 和 4.10 可知，在存储单元中占据了非常大的比重，但是这部分比重主要是在静态功耗部分，由于存储单元的占据的面积巨大。虽然单次读取信号，只有一个六管单元工作，动态功耗非常小，但是整体看来这个模块的静态功耗很大。另外，输出译码部分就是位线和字线的控制电路，该部分电路有大量的充放电过程，所以该部分电路的功耗值大小和灵敏放大电路不相上下。

下面的柱状图 4.11 是关于写操作的功耗对比：

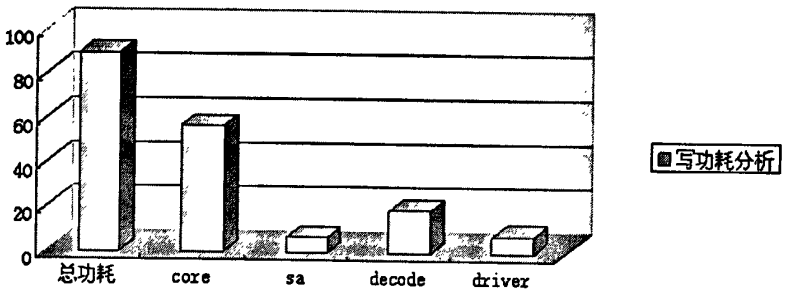


图 4.11 存储器电路各部分功耗对比图

这里看出，写操作中核部分的功耗最高，而译码部分就是包括了字线和位线的操作部分占据了非常大的功耗，灵敏放大和驱动电路的功耗占据的比重就非常小。

因此根据以上进行的功耗分析对比，得到的结论是在进行存储器低功耗设计时，降低核的功耗是非常重要的，另外减少位线和字线上面的功耗也是一个值得讨论的论点。

4.2 存储器设计方法分析

存储器低功耗设计的有许多设计思路：

1. 从电路设计角度来进行低功耗设计，修改存储器电路内部的结构，对存储器内部的电路进行优化设计。
2. 降低电源，从器件工艺角度对电路进行优化设计。
3. 从存储器使用角度对存储器设计进行优化，主要是根据存储器的器件特性对

其读取的过程优化设计。

通过对比分析得到的结论是，从存储器使用角度进行优化，可以起到最优的存储器优化效果，第二优化的是从降低电源角度进行优化设计，第三的就是对存储器内部电路进行优化改造。下面对三种方法分别进行论述。

1）从电路设计的角度进行优化设计：

从电路优化的设计的角度来看，存在诸多的设计方法。通过以上的存储器的功耗进行分析，可以得到的结论，存储器电路的功耗主要集中在存储器的存储单元，灵敏放大器和字线和位线网络三个部分。

由于存储单元电路主要是由六管单元构成，所以通过结构改进是不能够有效的降低动态功耗的，因此利用改变工艺角度来实现，由于存储单元的六管单元的面积非常的大，因此静态功耗占据非常大的比重，因此减少存储单元的静态功耗需要减少亚阈值电流。一般在器件或者电路设计中，降低亚阈值电流需要用的方法是采用高阈值工艺来实现。

通过利用 HSPICE 进行仿真，得到的一个仿真数据，下表是仿真得到的结果。对双端口 SRAM 进行写操作，得到的结果是高阈值功耗为 4.6822e-05W，而普通阈值的结果是 6.3521e-05W，可以看到利用高阈值的设计方法降低功耗有 27%。但是随之而来出现的问题是速度变慢，这是一个非常重要的问题，降低速度后，预示着电路的读写速度变慢，影响整体的性能。

通过查阅文献可知，有诸多的关于阈值设计的方法，利用将整体的存储器设计成为部分阈值为高阈值，部分阈值为低阈值，这样的设计方法平衡了速度和功耗的关系。例如，将整个存储单元设计成为高阈值工艺，而在译码器，灵敏放大器，充电放电电路等都设计成为普通阈值电路，这样可以有效的提高读取速度，并且降低整个存储单元的静态功耗。另外，如果认为读写速度还是不能达到要求，可以将六管单元部分的晶体管设计成为部分为高阈值，部分为低阈值，当部分为高阈值时可以减少亚阈值电流值，当部分为普通阈值时，可以有效的减少电路的读写时间。

利用高阈值设计的优点：采用工艺设计，可以实现，并且实现方法不会非常复杂，并且有现成的 IP 核的库使用，使用便利。缺点：电路的整体性能不是特别好，当要求速度优先时，就会出现读取和写入的速度不能达标的问题。

综合考虑，在使用该设计时，由于本处理器工作的频率不高，所以存储器即使采用了高阈值的工艺角，也不会影响整体的存储器的性能，所以在整体设计中，采用了高阈值设计方法。

图 4.12 中就是使用高阈值晶体管的示意图，从图中可以看出在双端口的存储器中，作为存储单元的 4 个晶体管已经设计成为了高阈值的晶体管，而作为数据写入和数据读出的晶体管，则用低阈值的晶体管来代替，这样做的好处是减少了

读取时间和写入时间。提高电路的读取速度。

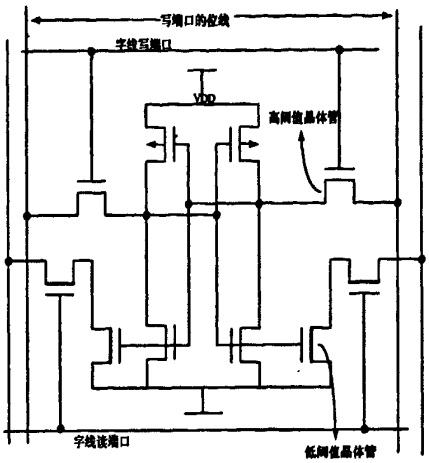


图 4.12 高阈值晶体管的六管单元模块

在电路设计方面，由于字线和位线上面占用的功耗比重较大，因此在进行存储器低功耗设计时，需要额外的减少充电和放电的比例的大小。由于一块灵敏放大器对应一部分字线单元和位线单元，因此在进行设计时，经常使用分割字线和位线的设计方法，减少字线和位线的充电和放电的比例。这种设计方法经常使用，在存储器模块比较大时，优势较为明显，但是在存储器块不是非常大时，效果就不甚明显了。由于采用的是固定的 IP 核设计，因此如果采用分割字线和位线，虽然能够起到减少功耗的目的，但是整体工作量巨大，效果也不是非常的明显。因此综合衡量，对字线和位线进行低功耗设计还是不行的。图 4.13 是分割字线和分割位线电路结构图。

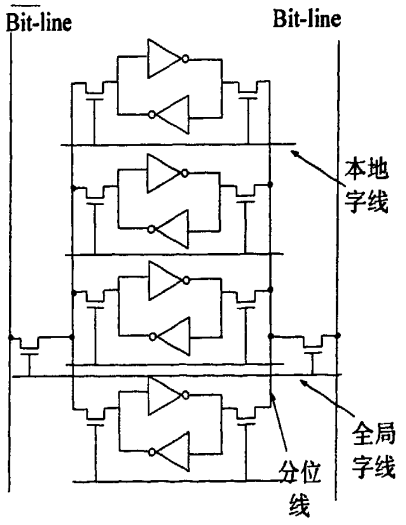


图 4.13 分割字线和位线的结构示意图

从图 4.13 中可以看到位线单元和字线单元是单独进行布线的，因此在电路中将全局字线的信号作为一级控制信号，来实现对下级字线的信号控制，这样单次充电的字线或者位线的数量不大。这样可以节约非常多的充电电流和漏电电流。

对存储器灵敏放大器的设计，主要包括改进灵敏放大器的速度，减小灵敏放

大器的反应时间，目前应用于存储器的灵敏放大器主要是锁存型的灵敏放大器，主要特定是该种灵敏放大器的反应速度很快，能够快速实现信号的放大，信号的反应速度较快。灵敏放大器电路形式如图 4.4 所示。

以上三种设计方法就是存储器电路的低功耗电路改进设计，其中针对的主要是设计中的功耗占存储器比例较大的部分。三种方法中，较为容易实现的方法是全部电路采用高阈值设计方法。由于在 Artisan 存储器电路中灵敏放大器已经采用了多种低功耗的设计方法，因此，在进行电路设计中，对灵敏放大器的优化设计空间已经不大，而在对字线和位线进行分割处理设计中，由于工作量巨大，所以也是较难实现的。所以综合考虑，仅仅采用统一全部使用高阈值的设计方法。

2）降低电源电压的设计方法：

根据动态功耗的公式可以得到如下公式：

$$p=\alpha\cdot C\cdot\Delta V\cdot V_{dd}\cdot f \tag{4.1}$$

当电源电压越低则动态功耗越低，因此在进行低功耗设计时，会考虑到采用降低电源电压的设计方法。由于使用的是固定 IP 库，因此在进行低电压设计时需要按照功耗，面积和速度进行一个权衡。

当进行正常工作是，电压越低，会使得工作速度变慢，以反相器为例，会在 0 到 1 之间的电压的中间态处于很长时间。因此这种设计需要额外的验证，如果出于很长时间，动态功耗相反会很大。

总体来一般工艺会规定标准的工艺参数，即例如 SMIC.13 的工艺，标准工作电压为 1.8v。这样在左右上下电压浮动 0.12v 时，工作情况还是满足工艺角的变化的，即按照仿真得到的结果是在 1.62v 到 1.92v 之间时，能够满足正常的工作，但是当电平信号小于 1.2 时，则功耗结果就会有非常大的差别，并且在速度和性能上面也有差别。

从仿真上面看可以知道，在利用 HSPICE 进行试验得到的结果是：

标准工艺水平下，在 1.8v 电压下，得到的功耗为 3.1517e-05。

标准工艺水平下，在 1.0v 电压下，得到的功耗为 1.8063e-05。

在高阈值工艺水平下，在 1.8v 电压下，得到的功耗为 2.2339e-05。

在高阈值工艺水平下，在 1.0v 电压下，得到的功耗为 2.0688e-05。

所以从上面得到的结果是虽然在高阈值工艺下，亚阈值电流会小，但是在 1.8v 和 1.0v 电压之间时，整个功耗会比在标准工艺下的功耗更大，因为，在高阈值电路中，整个电路的在非稳定状态时的电流和电压会非常大，会额外的多出功耗值。所以从这里看，如果采用标准工艺，在 1.0v 电压下，可以实现整个电路功耗的降低，而在高阈值工艺下，在 1.0v 电压下，可以实现整个静态功耗最低。

图 4.14 和图 4.15 中是低电压下波形延迟对比图。

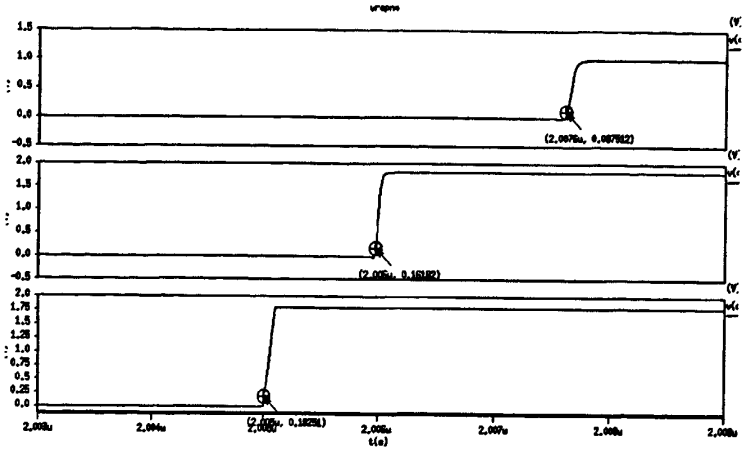


图 4.14　1.0V 电压下信号延迟对比图 1

图 4.14 中是低电压情况下的波形输出，而第二个分栏是在标准电压的波形，第三个是输入信号波形。

从图 4.14 中可以看出在使用了低电压后，会出现传输延迟时间增大。这个可以看出输入信号翻转后，得到输出信号变化延迟了 1ns。

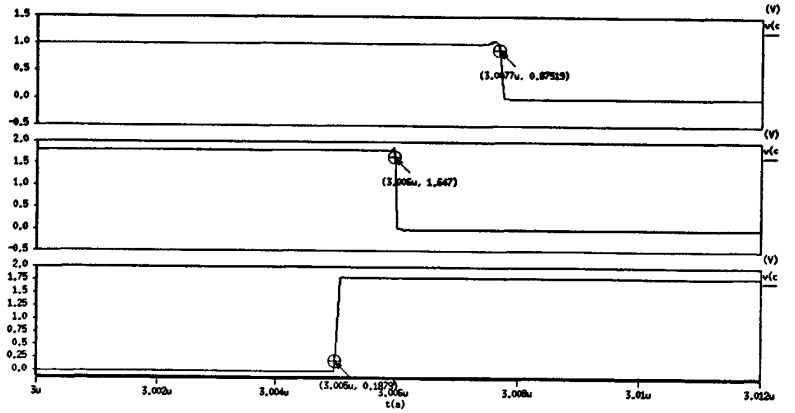


图 4.15　1.0V 电压下信号延迟对比图 2

图 4.15 中是低电压情况下的波形输出，而第二个分栏是在标准电压的波形，第三个是输入信号波形。

从图 4.15 中可以看出，在当地址信号输入后，数据的信号也会得到翻转，数据的延迟也是 1ns。因此整体看来加低电压后，会将延迟增大，但是在整个设计中不会影响整个处理器工作的时序。

综上所述，利用低电压的特点可以实现整个电路的低功耗，但是输入和输出的电平信号会变小，然而延时时间不是很大。所以在整个设计中，该种低功耗的设计思路和方法可以方便的应用于整个处理器设计中，整个设计不涉及电路设计，不会更改电路结构和 IP 核。

3）存储器使用低功耗角度设计

存储器在使用时，可以利用读取控制设计方法，来完成对存储器的优化控制，从而降低功耗。因为不需要改变存储器 IP 核的库类型和内部结构，电路不需要大

量的人员去设计，并且从使用方法入手，会非常的方便，稳定性高。

在根据存储器的特性进行设计时，需要额外的了解存储器的宏观特性。对于存储器的特性需要从多个方面进行综合考虑，一方面是存储器的容量，容量越大，存储空间越大，使用效率就会越高。第二是功耗，对于真正考虑功耗的设计，需要不同类型存储器的性能比较。第三，面积问题，虽然有些情况下面积问题不是主要问题，但是在进行设计时，面积还是限制了整个的布局布线。

对于 artisan 存储器的综合考虑分析，在 artisan 的可写入存储器中，有两类存储器，一类存储器是 register file(RF)，另外一类存储器是 SRAM 存储器。这两种存储器的结构特点和内部特性是稍有差别的。

从内部的结构看，RF 存储器和 SRAM 存储器是相同结构，在存储单元部分，是标准的六管单元，而在灵敏放大器部分，则是锁存型的灵敏放大器，至于充电电路，则由于 RF 存储器规模较小，所以这部分电路较为简单，这就是预示着 RF 电路比 SRAM 电路速度更快，效率更高。所以在进行读写操作时，RF 存储器在小容量设计中是优先选择的，而 SRAM 则是适用于大容量的存储器，对于容量超过 256 的存储器设计，就需要选择大容量的 SRAM 进行。

因为进行的设计是关于 Artisan 的存储器进行的。所以首先对存储器进行设计的分析对比。在存储器中，位数相差一半和容量相差一半是有差别的。位数相差一半之间的功耗对比如表 4.2 中所示：

表 4.2 不同位宽存储器性能对比

型号类型	面积 μm^2	功耗（mA）
1024x32	139619	5.24e-2
1024x16	75481	3.02e-2

从上面表 4.2 中可以看出，当位数有一半之差时，面积会相差一半，而且功耗也会相差一半。仅仅从相同位数的 SRAM，不同容量的结果进行对比得到：

表 4.3 相同位宽，不同容量存储器性能对比

型号类型	面积 μm^2	功耗（mA）
512X32	94488.3	5.09e-2
1024X32	139619.0	5.24e-2
2048X32	229265.3	5.55e-2
4096X32	409067.8	6.16e-2
8192X32	772961.8	7.38e-2

由上面的列表 4.3 中可以清晰的看到变化趋势，在增大存储器容量一倍时，整个存储器的面积会增大一倍，相比之下，增大存储器位宽一倍时，也会增大存储器面积一倍。而在功耗方面，当存储器容量增大一倍时，其功耗不会增大很多，增大的相当有效，仅仅只有 4%左右。所以得到的结论是，在使用存储器时，利

用增大容量，而不额外增大功耗的设计方法是非常有效的。在存储器使用中，会涉及到非常多的使用存储器的不同大小类型，这样对于小于等于256容量的存储器采用RF存储器，而在大于等于512容量的存储器采用RA大小的存储器。

4.3 存储器循环缓存设计

4.3.1 存储器设计讨论

在存储器使用的优化设计中，经常使用两种优化方法，一种方法是对存储器进行分块的优化设计，将大存储量的存储器进行优化的分块处理，另外是利用缓存cache的设计思想，将电路中的存储容量较小的模块存放较为常用的数据。利用上面这种设计方法来实现对存储器电路的优化设计。

1）分块设计优化方法：

对存储器电路，如果分块处理，会将电路中的部分存储器电路关闭，通过开启某些部分模块，关闭某些部分模块，降低整体的读写功耗。这种类型的设计方式是比较常用的，并且是比较简便的。模块电路示意图如图4.16所示：

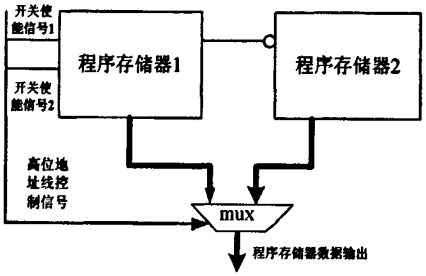


图4.16 存储器分块电路模块图

从上面的图4.16中看出，在进行分块设计存储器电路的时候，需要高位的选择控制线，另外还需要额外的一个多路选择器。这种设计方式比较方便，并且简单。但是在优化空间和面积开销方面不是十分理想。由于将存储器分块，额外了添加了许多的译码器电路和充电回路和放电回路电路，有大量的面积开销，虽然功耗会减少很多。

通过编写该硬件电路，并且采用SMIC 0.13工艺进行验证，实现将两个存储器拼接成为一个存储器，然后得到与一个存储器相同功能的模块。将一块512*32的存储器分成了4块128*32的存储器，在整个设计中，存储器功耗降低了50%左右，但是在面积方面却增大了整个处理器芯片面积的19%。增加了较大的面积。综上所述，得到的结论是在存储器使用过程中，如果所使用的面积较小，利用分块的方法还是非常有效的，但是如果使用的存储器面积较大，并且在设计中数据的读取频繁的大范围跳动，则不适宜采用分块的设计方法。

2）采用Cache结构设计方法

由于在进行设计中，分块设计思想添加了太多的面积，所以这部分设计需要进行优化，一般说来，可以采用 Cache 结构设计方法来对结构进行优化。

Cache 结构的设计思想是在设计中添加入一个小的存储器，这个存储器用来存放当前要进行读取的数据。由于当进行读取时，添加的小存储器不论在静态功耗方面还是在面积方面都会优于原先的分块的设计方法。

在传统的 Cache 结构中，Cache 设计总是动态变换的，即 cache 结构有着优良的自动更新能力，这种设计方法比较难做，尤其是对于低功耗设计而言，这种设计方法会显著的增加额外的功耗，所以这种 cache 结构不能原班的抄去。具体的方法由具体的设计方法来对待，下面进行的是针对低功耗 ASIP 处理器设计的一种低功耗 cache 缓存模型。

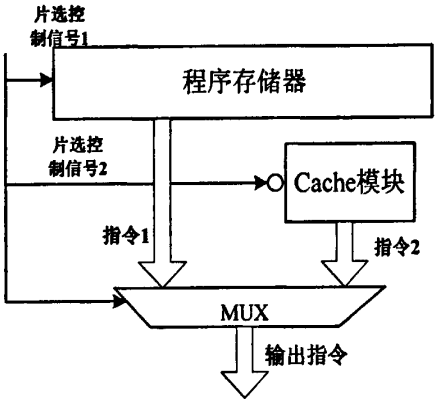


图 4.17　程序存储器 cache 结构示意图

从图 5.17 中可以看出，这个 cache 结构有两大部分组成，一部分是 mux 多路选择器，另外一部分是 cache 结构，这部分结构是用来进行短代码存储使用的。为了简化操作步骤，会将程序存储器中的一段代码存储在 Cache 中，这段代码的特点是具有非常好的重复性，这段代码会重复的使用，这就是这段代码的特点。固定存入后，就可以非常方便的对代码进行读取，当地址信号命中 Cache 内部的地址范围时，可以自动关闭 program memory 的工作，然后打开 cache，这种设计方法较为简单。具体的工作流程图如下：

1）Program Counter 进行自加处理，PC 值写入到 PROGRAM MEMORY 和 CACHE 中。

2)电路控制进行选择判断，当地址位于 CACHE 区域内部时，关闭 PROGRAM MEMORY，打开 CACHE 结构。

3)进行数据读取，当地址超出范围后，CACHE 结构关闭，然后打开 PROGRAM MEMORY。

以上两种结构都各有优点和缺点：

1）分块结构功耗优化空间很大，分的块数越多，则功耗优化越大，但是面临的问题是面积越大，这就是所说的利用面积换取功耗。CACHE 结构的面积优化

程度不如分块结构。

2）CACHE 结构的使用是与具体的算法有关的，如果算法规律性不强或者程序没有频繁执行的操作，那么 cache 结构的优化空间就非常有限，效果不如分块结构好，但是面积会比分块结构小很多。

综上所述，如果要从根本上优化整个存储器系统，如果要达到既要优化功耗，又不增添过大的面积，需要从控制的角度入手，完成整个的系统的优化。以下就是本文提出的循环缓存结构的介绍。

4.3.2 循环缓存设计

循环缓存结构主要是将前面讨论到的分块结构和 CACHE 结构进行了相结合的设计。由于分块结构的缺点是增加面积大，而 cache 结构的缺点是代码覆盖率低，因此将两者进行结合，设计出一个合理的功耗优化策略。

由于 Cache 对代码覆盖率较低，因此提高执行代码的覆盖率成为了研究的一个重点，在这里设想如果在小的存储器中的代码是可以自动更新的，那么在正常工作的时候，小 Cache 的使用率就会大大的提高，在面积分块上面，这块的面积大小也会比分块设计面积小很多。这就是我们设计的一个重点思路。

循环缓存电路的研究已经有很多，其中大都是基于两种设计思路，一种设计思路是从电路设计角度入手，通过修改整体电路结构，来实现增个循环缓存的优化[44][45]。这种设计方法的优点是较为简单，控制方面较为简单，但是缺点是代码涵盖率相对较低；另外一种设计思路是从软件设计的角度入手，这种设计思想这样的，通过执行操作，得到最后频繁执行的代码的代码段，然后对其进行编译器的修改，插入标签标识，在随后的正常工作中，进行代码的执行操作[46][47]。

以上的两种设计方法相比较，有着自己各种的优点和缺点。从电路设计角度进行定制设计，这种设计方法较为方便，并且功耗优化程度相对较高，而在从软件设计角度入手来看时，这部分设计主要是覆盖率较高，如果涉及专用的优化设计，会出现大材小用的现象。所以在综合了以上的比较后，得到的结论是，在本设计中，采用电路专用设计的方法来进行优化设计。

为了完成专用设计结构优化，采用了对指令集中的循环指令作为标示指令的做法，循环指令是一个在本 ASIP 处理器中非常重要的指令，该指令可以实现对整个循环操作进行控制，这部分控制主要是标示出需要执行的代码段，然后根据循环指令的信息，执行执行的遍数。其中在循环指令中涵盖了循环代码段的运行遍数，执行的起始地址，结束地址等。图 4.18 中是循环指令格式。

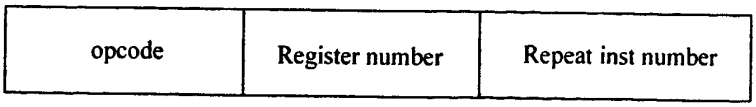


图 4.18 循环指令指令格式

图 4.18 中是该循环指令的执行代码格式，其中 opcode 是循环指令的执行码，这部分执行码主要是进行代码的识别操作，判断代码是什么代码。Register number 项主要记录的是在执行操作中哪一个寄存器中记录的执行次数。由于该寄存器是通用寄存器，在处理器代码执行中，会有等到执行到 EX 级，才识别出执行的次数，然后再返回到 PC_FSM 中，进行执行的具体操作。最后一项是 repeat inst number,即代码执行中执行的条数，有多少条指令执行循环操作。

因此在这种设计中，需要考虑合理的设计方法。对于非常关键的执行此数的信息，由于隐藏在寄存器中，所以在执行时，会有较多的延迟，并且数据信息不实非常明确，对内部数据连接到缓存结构中非常不方便。由跳转指令的结构图中可以看出。

当执行完 ID 一级时，后在下一级对寄存器中的数据进行解析处理，然后得到最后的执行结果。这样需要额外的三个时钟周期才能读取到 loop_control 模块，控制较为复杂，难以实现，并且为了读取内部寄存器中的值，然后加入数据通路，会额外的增加功耗，这种设计方法非常不便利。真正方便的设计方法是能够简单实现，并且不需要额外修改处理器中内部的电路结构的设计方法。本次循环缓存结构中，放弃了对内部寄存器的读取，不进行次数的记录操作，改用记录循环指令中断点的设计方法。图 4.19 中是循环缓存操作流程。

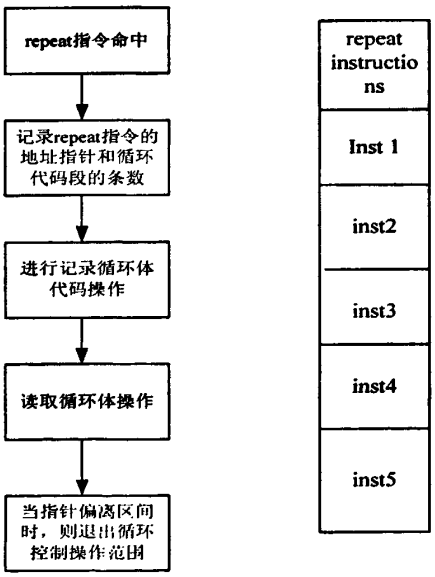


图 4.19　循环缓存操作流程

循环缓存设计方法中，采用如下的设计方法进行设计：

1）对电路中的起点端点进行记录

2）记录完成起点端点后，然后解析代码中的代码执行的条数。

3）通过起点端点和代码执行条数得到结束端点。

4）保证工作在起点端点和结束端点之间时就可以实现整个循环结构的控制。

利用这种设计方法的优点：

1）能够保证不对处理器电路内部电路进行过多的干扰。

2）保证缓存电路的反应速度加快。

3）对于像中断或者跳转这样的操作，由于采用的是 PC 指针控制，所以可以方便的完成跳出，不会出现 IF 语句的判断跳转问题。

采用指针作为运行操作的重点。在利用指针操作时，由于循环指令是放在代码段的前面，因此在执行操作时，可以执行完一遍循环指令后，即刻完成存储，然后关闭大存储器，打开小存储器，进行低功耗的读取操作。

4.3.3 循环缓存的结构

循环缓存结构降低功耗的原理主要是利用不同大小的存储器的读取功耗不同，通过关闭大程序存储器读取小程序存储器的方式来实现程序存储器的低功耗设计。例如在 0.13μm 工艺条件下，基于 ARM Artisan 的存储器自动生成工具得到的存储器 IP 核中，在典型工作情况下（1.2V、25℃），容量为 64x32 的存储器单次读取的功耗为 9.23 μA，容量为 1024x32 的存储器单次读取的功耗为 44.5 μA。因此可以看出在设计中利用小存储器代替大存储器进行读取可以有效的降低整个系统的功耗。图 4.20 是循环缓存结构示意图。

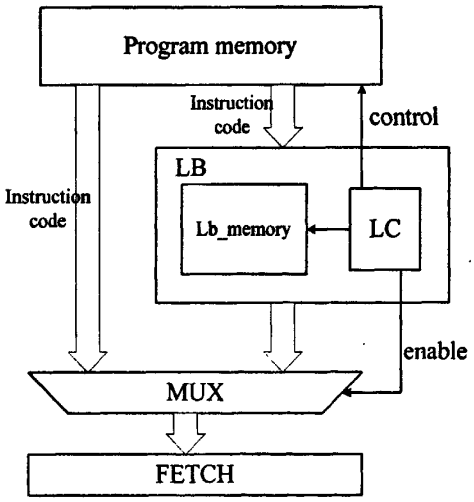


图 4.20 循环缓存结构示意图

图 4.20 中所示是循环缓存与主程序存储器结构。LB 代表整个循环缓存电路模块，包括循环存储器（Lb_memory）和循环缓存控制模块（LC）。程序存储器输出的指令代码进入 LB 模块内部，通过 LC 控制，识别出 repeat 循环指令并将随后的循环体代码存储在 lb_memory 中。系统在第一次进入循环操作时，对 program memory 输出的指令代码进行读取，将输出的指令分别输入到 MUX 中和 LB 模块中，当对循环体进行第二次操作时，LC 会关断 program memory,指令会从 LB 中直接输出，并且 LC 控制 MUX，选择从 LB 中输出的指令。当地址信号超出了循环体的地址范围时，LC 模块会关闭 lb_memory,开启 program_momery。MUX 模

块会选择 program_memory 输出的指令代码并将其传输到 FETCH 模块。

4.3.4 对于循环缓存结构的改进型设计

传统的循环缓存结构都是基于一段代码的寄存存储。但是在设计中，会出现对多个代码进行迭代式的循环操作，这样就出现了频繁的对相同代码的写操作。这样会造成额外的功耗浪费，如何减少这种功耗浪费，本文中设计出了采用多段式结构的循环缓存设计。

普通的循环缓存结构一般仅仅是存储一段循环代码，一般一段代码只有 8 到 31 条指令，在对存储器分析后得知，在 RF 存储器中，32X32 的存储器和 64X32 的存储器的功耗情况几乎相同。所以在进行优化设计中，为了以后设计的方便，采用利用大容量的存储器作为缓存存储器。

当进行单次操作时，由于每次仅仅存储一段代码，然后就不存储，这样的设计方法本身对存储器就是一种浪费，所以提高存储器利用率，减少对存储器的写操作，就变成了越来越至关重要了。图 4.21 是多段式循环缓存结构示意图。

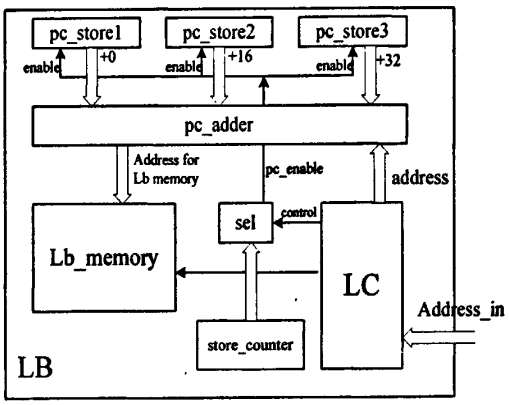


图 4.21　多段式循环缓存结构示意图

多段式循环缓存结构实际上是将缓存存储器分成多段，在每一段中分别存储一个循环体，配合 LB 模块进行控制操作，由于采用多段存储，避免了临近相同循环的重复写操作。利用该方法可以提高缓存存储器的利用率，减少对缓存存储器的写操作，降低整体程序存储器系统的功耗。

通过对比深度为 32，64，128 的缓存存储器可知，深度为 32 的缓存存储器中容量过小，命中率过低，从功耗的角度衡量，深度为 64 的存储器单次读取的功耗相对于深度为 32 的存储器增加了 6%，而深度为 128 的存储器单次读取的功耗比深度为 32 的读取功耗增加了 20%，因此综合考虑，选取深度为 64 的程序存储器作为缓存存储器。

通过分析本设计的相关算法的指令，可以发现，循环操作中最大的循环体为 31 条指令，因此在设计缓存存储器时，就需要考虑如何更加合理的划分存储器中各段的空间大小。由于包含多于 16 条指令的循环体所占比例很小，大部分循环体

的指令数都在 7 条到 14 条之间，因此如果在存储器中开辟的缓存空间均为 32 时，依然会造成存储空间的浪费和读取存储器命中率过低。因此，综合考虑，本文采用将缓存存储器按照 16：16：32 的空间分布划分为三个存储段的结构，这种设计在提高存储器利用率和命中率上比较合理。

图 4.21 所示是本设计提出的多段结构的循环缓存电路结构图。由于缓存存储器分成了三个存储段，在 LB 模块中，添加了三个寄存器分别为 pc_store1,pc_store2,pc_store3。这三个寄存器分别记录了最近三次存储的循环体对应的 repeat 指令的地址。当下一次 LB 遇到 repeat 指令时，LB 会将 repeat 的地址和三个寄存器中的地址进行对比，如果和其中一个寄存器的地址相同，表示在缓存存储器中已经记录了该循环的循环体代码，可以直接关断主存储器，并从 LB_memory 读取指令。由于循环体存储到缓存存储器中的位置与三个寄存器存在一定的映射关系，因此在读取某段已缓存的循环体代码时，缓存存储器的读取地址需要加上一个地址偏移量。在本设计中当 pc_store1 命中，则地址偏移量为 0，当 pc_store2 命中，则地址偏移量为 16，如果 pc_store3 命中，则地址偏移量为 32。

在遇到循环指令操作时，如果已存指令没有命中，则需要对新的循环缓存进行存储。若新循环体包含的指令多于 16 条，直接将 repeat 指令地址存储在 pc_store3 寄存器中，并以 32 为起始地址在缓存中存储该循环体。如果循环体小于 16 条指令，则根据计数器 store_counter 输出的存储序号，来判断应该写入的存储器段，并进行存储，同时将相应的地址写入相应的地址寄存器中。

图 4.22 中是循环结构状态机转换图。

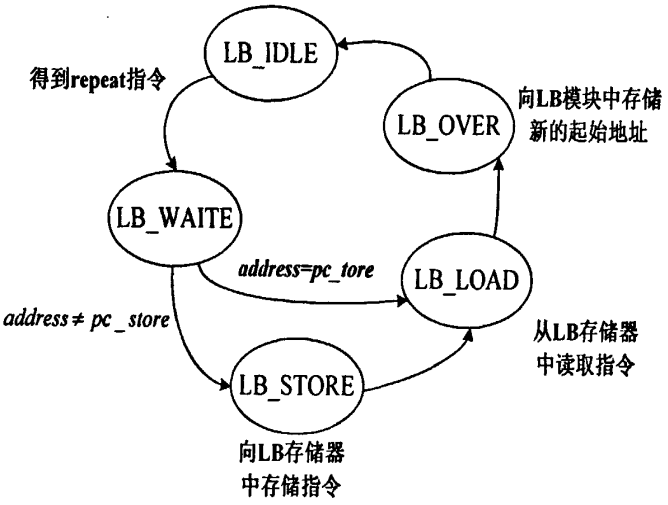


图 4.22　多段式循环缓存状态转换图

图 4.22 所示是多段式循环缓存控制模块的状态转换流程。在状态机设计中，加入了 LB_WAIT 状态，目的是实现对地址信号的映射和对已存循环体的命中判断。当输入的 repeat 地址和 pc_store 相同时，直接进入 LB_LOAD 状态，进行读取操作。当 repeat 地址和 pc_store 中的地址不相同时，表示没有命中，进入

LB_STORE 状态，进行循环体指令的存储，然后再进入 LB_LOAD 状态，进行指令读取。当完成读取操作后，在 LB_OVER 状态中，对 pc_store 存储的地址进行更新。

通过仿真得到图 4.23 和 4.24 仿真波形结果：

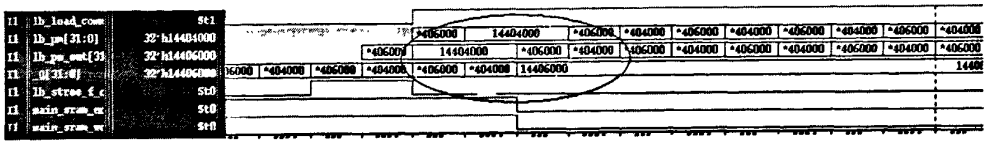


图 4.23 循环缓存仿真结果示意图

从图 4.23 中可以看到，整个设计中首先是 Q 的输出数据，然后经过一个循环体的操作后，得到的操作波形入上图所示，整个设计可以看到当 main_sram_wen 信号置为高信号时，主存储器正常，而缓存存储器会关闭。当 main_sram_wen 置为低电平时，则缓存存储器工作，主存储器不工作。

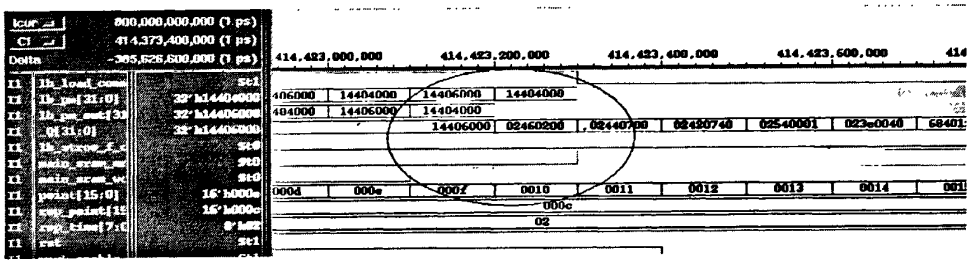


图 4.24 循环缓存仿真结果示意图 2

当从循环缓存中退出来时，所示的波形如图 4.24 所示，当跳出循环后，整个指令存储器会输出另一个主存储器的指令，而 LB 会关断，从使能控制信号上可以看到，当 main_sram_wen 为高后，缓存存储器不工作，主存存储器正常工作。

在循环缓存结构设计中，对于存储器设计，需要最大限度的对其进行优化，因此这种设计需要对电路进行重点设计，然后进行综合考虑，最后得到优化。

首先需要对已经进行的程序进行分析，在对循环指令的工作情况进行分析可以得到，小于 8 条指令的循环体操作有 8 个，超过 8 个指令的循环体有 5 个。所以从分析看，可以使用将循环体进行更进一步分离的方法进行设计。

由于 RF 存储器的容量有一个下限限制，并且利用完成的存储器模块，容量越小，附加的功耗耗费就会很大，所以需要另外利用别的设计方法进行优化设计。在本项目中，我采用了利用寄存器堆来搭建一个小的存储器模块，利用这种设计，可以实现缓存存储器双存储结构的结构原理。

双存储的结构模型如图 4.25 所示，在整个设计中 FSM 作为控制模块，进行数据的控制，而由外面来的地址 PC 信号，会进行分析判断，当地址段小于 8 时，会在寄存器堆存储器中进行读取，而当地址段大于 8 时，则会形成偏移量，进行 RF 缓存存储器的寻址。

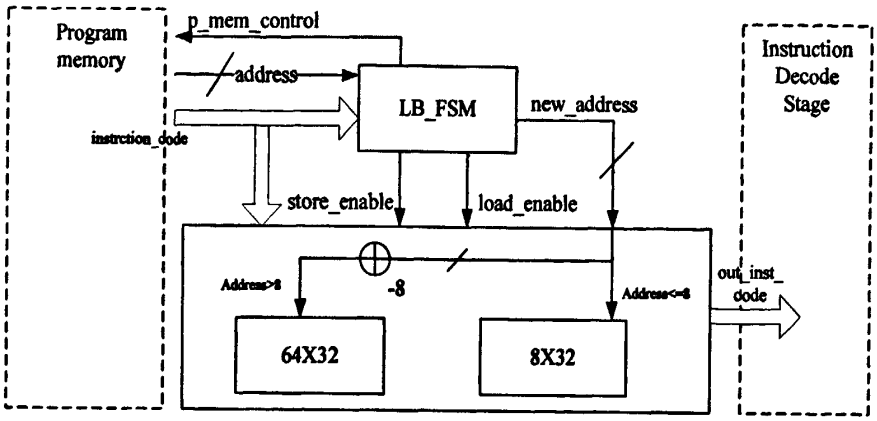


图 4.25 双存储器循环缓存电路结构

在电路设计上面，本设计为了减少功耗，在存储器模块的使用上面采用了时钟门控[48][49][50]的设计方法。时钟门控的设计主要是利用单元库中的时钟门控单元，这个单元主要是用来利用使能信号，将门控关断，屏蔽时钟信号。由于在存储器电路还是逻辑电路中，时钟信号的输入后造成门电路的翻转，所以电路中需要将时钟门控电路进行屏蔽。

时钟门控电路图如图 4.26 所示：

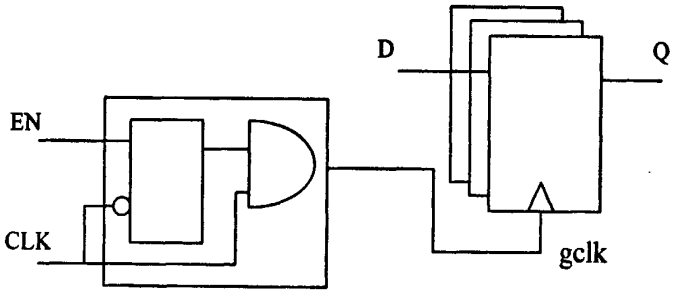


图 4.26 时钟门控电路结构图

从图 4.26 中可以看出在时钟门控内部，是有一个触发器模块和一个与门。这种设计主要是为了防止由于在添加使能 EN 后，出现的抖动问题。防止时钟毛刺的产生。

在 SMIC.13 的库中，有相关的时钟门控信号，这个信号的库文件名为：TLATNCAX2TH。使用方法是在需要使用的库文件中，对该文件进行例化，连接需要连接的控制信号和时钟信号。

仿真得到的逻辑波形图如图 4.27 所示：

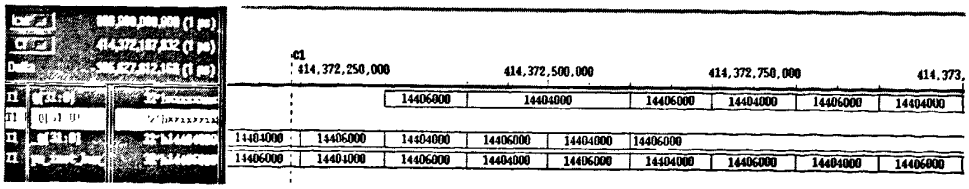


图 4.27 分模块存储器波形图 1

从图 4.27 中可以看到在整个处理器工作时，采用添加两层寄存器缓存的结构时，可以实现整个对大缓存存储器的关闭处理，然后仅仅从小存储器中读取数据，上面白色的 Q 是缓存存储器的数据输出，可以看到没有数据的输出。

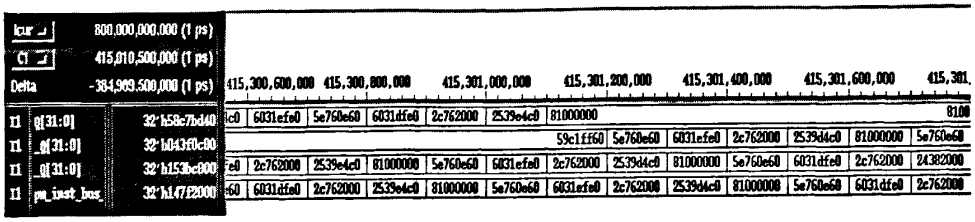


图 4.28　分模块存储器模型图 2

从图 4.28 中可以看到当数据量大于 8 时，可以将数据首先存入到小的寄存器缓存中，然后进行数据对大的缓存数据中进行写入，当数据在读取时，大于了地址 8 时，数据就会从大缓存存储器中读取，从上面的数据波形中可以看到数据能够正常的实现转换。利用这种方法可以实现对整体缓存的功耗的进一步降低，降低整体的功耗。

4.4 小结

本章从处理器低功耗设计的角度出发，分析得到在处理器中，存储器占有的比重非常的大，占据了处理器整个功耗部分 40%。因此在实际的设计中，为了减少整个处理器的功耗，就需要对存储器进行低功耗设计。本章首先对存储器进行了分析介绍，通过对比存储器位宽特性和存储容量特性进行了简短的分析，得到了对存储器进行适度的分块可以有效的减少存储器的使用功耗。本章随后介绍了多种基于处理器的低功耗设计技术，包括分块设计，cache 结构，缓存结构设计。通过对比，得到的结论是当使用循环缓存结构时，可以有效地减少处理器的功耗。随后对循环缓存结构进行了优化的设计，设计出了多段式的结构，通过功能验证验证得到整体功耗优化非常好，能够起到较优的低功耗设计。

第 5 章 ASIP 设计实现和相关功耗结果

以上章节分别对助听器处理器进行了结构设计，指令集设计。然后针对专用程序存储器模块进行了低功耗设计。本章对该助听器处理器进行 FPGA 验证设计，然后根据 ASIC 设计流程对该处理器电路进行电路综合和版图设计，并且给出最后的电路功能验证和性能分析。

5.1 功能仿真和 FPGA 验证

5.1.1 前端功能仿真

用硬件描述语言 Verilog HDL 完成对电路的 RTL 级描述以后，首先要进行功能仿真，验证每一个模块级整体电路系统的功能是否满足要求。前端的仿真工具是 Modelsim，通过编写相关的测试激励，和处理器执行汇编代码进行前端的仿真。

5.1.2 FPGA 验证

FPGA 是 IC 设计中一个重要的环节，它具有设计周期短，布局布线较为简单，研发费用低等特点。

在将设计的 RTL 代码完成后，首先先将 RTL 映射为 FPGA 的逻辑电路，然后下载到 FPGA 中，进一步检验电路功能设计的正确性。FPGA 是可编程逻辑器件，可以重复多次擦写，易于改进设计中存在的问题，提高电路的可靠性，降低流片的风险。本设计的 FPGA 验证是在 Altera 公司的 Stratix II EP2S180F1020C4 的 FPGA 开发板上进行的，它有着丰富的外围模块电路和丰富的 RAM 资源，适合用于对处理器电路进行验证。图 5.1 是 FPGA 验证原理图。

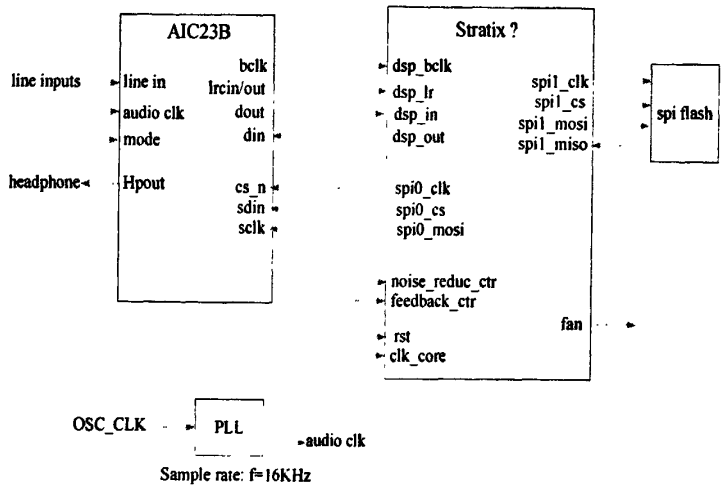


图 5.1 FPGA 验证功能仿真

平台主要包括 Altera 的 Straix II EP2S180F1020C3 开发板，TI 的 TLV320AIC23 系列 CODEC 音频解码器（图 5.1 中的 AIC23B）和 Winbond 的 W25X10AV Flash 存储器（图 5.1 中的 spi flash）。其中，FPGA 开发板为主体，实现数字助听器 DSP 芯片的音频处理功能，配套的仿真综合软件采用的是 Altera 公司的 Quartus II 8.0。TLV320AIC23 芯片内置 A/D 和 D/A，支持 MICIN 和 LINEIN 两种输入方式以及耳机输出和 LINEOUT 两种输出方式，为验证平台的 I/O 设备；验证中使用的是开发板自带的 CODEC 芯片和相应接口。Flash 存储器为外置存储器件，用来保存处理器中程序存储器所需要执行的代码和所需要配置 CODEC 芯片的寄存器参数，上电之后写入 FPGA 系统。平台工作过程如下：DSP 系统的 RTL 代码在 Quartus 中综合完成后，通过 USB Blaster 下载到 FPGA 中；用通用串行 EEPROM 编程器将系统所需的参数写入片外 Flash，上电后由 Flash 通过 SPI 模式对 FPGA 进行参数配置；FPGA 通过 SPI 模式对 CODEC 芯片的寄存器参数配置完成后，系统进入正常工作状态，外界模拟音频信号经过 CODEC 中的 A/D 转换成 16 位数字信号，通过 DSP 模式以比特流的形式送入 FPGA，经过 FPGA 的数字信号处理后通过 DSP 模式以比特流的形式送入 CODEC 芯片，由芯片内部的 D/A 转换成模拟音频信号输出。FPGA 照片验证平台如图 7.2 所示：

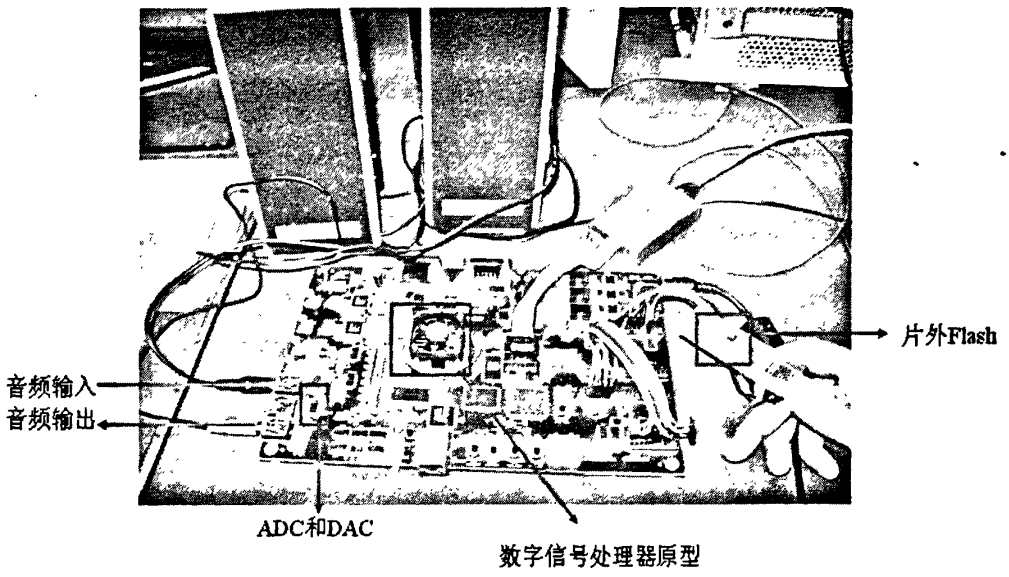


图 5.2 FPGA 验证平台照片示意图

利用图 5.2 中立体声音响，对比输入信号和输出信号声音信号的异同，验证整个处理器设计工作正确性，实验验证表明，该处理器功能正确，能够对声音信号完成多频率段的放大。

5.2 ASIP 设计及其相关的流程

专用指令集处理器主要采用 ASIC 设计流程来进行设计实现，采用 SMIC.13 的工艺。利用全定制实现整体处理器，具有运算速度快，可靠性高，体积小，保密性好等优点。标准的全定制的设计流程包括设计规范，算法设计，架构设计，RTL 设计，功能仿真，FPGA 验证，逻辑综合，门级验证，版图设计，版图后仿，物理验证。在前面已经完成了 RTL 设计，功能仿真和 FPGA 验证，下面需要进行的就是利用 SYNOPSYS 的软件进行综合和相关的物理实现。

5.2.1 综合设计综述

所谓综合，就是将硬件描述语言所描述的 RTL 电路网表转换为门电路网表结构，需要使用芯片工艺厂提供的 foundry 或者自己设计的基本电路单元实现 RTL 级电路的功能，这个过程称为综合。综合的具体步骤包括有转译，优化，映射。

综合现在比较流行的软件是 SYNOPSYS 的 DESIGN COMPILER。具体综合的过程包括以下设定：设定综合库，读入设计，设定工作环境，设定约束条件，选择约束策略，设计优化，分析并解决出现的问题。该软件是在 LINUX 下完成工作的，在 linux 工作环境下，可以编写脚本文件，然后 DC 通过脚本文件，快速的完成对电路的综合过程。

5.2.2 综合准备阶段

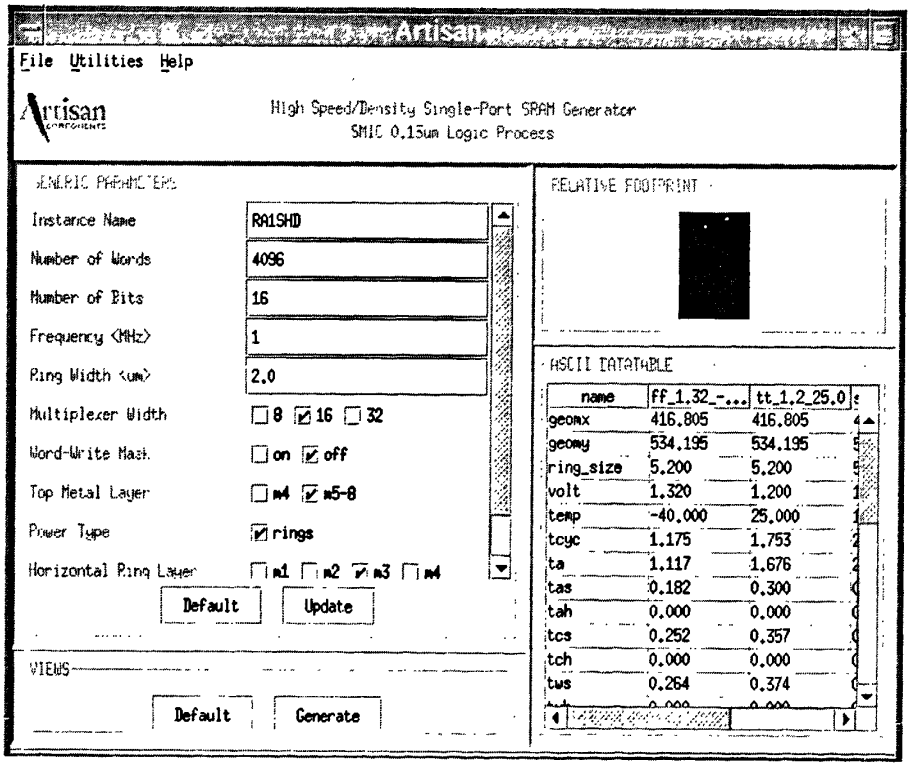


图 5.3 Artisan 存储器生成界面

在进行设计中，由于本项目使用到了许多固定的 IP 核，这些 IP 核包括专用的库和专用的存储器。这些核是采用购买的方式得到的，最为常用的核是存储器的核，这种核是 ARM 公司的 Artisan 存储器。Artisan 存储器主要是进行固定 IP 存储器的生成工作，图 5.3 中显示 Artisan 存储器生成界面图示。

在 Artisan 存储器生成工具中，可以生成不同容量大小的存储器，数据的位宽和存储容量的大小都可以设置，该工具能够生成不同类型的文件，方便 synopsys 工具或者其他 EDA 工具进行设计。本项目中所使用的存储器 IP 核就是基于 Artisan 公司的产品。

在综合设计中，需要这些存储器内核的 synopsys 文件，即后缀为.lib 的文件。包括各种拐角的文件类型。

5.2.3 综合脚本的编写

综合脚本是一个非常高效的设计方式，通过编写批处理文件，可以高效的完成整个设计的诸多操作。在 SYNOPSYS 的核心工具，design compiler 中，经常采用的就是使用批处理文件，即脚本文件。

在设计脚本文件中，首先需要定义综合的设计文件名，在随后的设计中统一采用该文件名作为报表。然后读入 RTL 网表，设定综合的参数，这部分参数主要是包括建立时间，保持时间，输入阻抗，输出阻抗，最大的输出的容性负载，最大的输出阻抗。

然后建立时钟，时钟信号的建立主要是设定整个 ASIC 的工作频率。这个整个工作频率和后面的建立时间和保持时间有关。在本设计中，主要的工作频率为 20Mhz，即整个处理器的时钟为 20Mhz。

VCS 作为 SYNOPSYS 的仿真工具主要是进行数字前端和数字后端的仿真使用，其中仿真的界面和 modelsim 比较相似，但是使用和操作方法是和 modelsim 有一定的差别的。这个工具主要是在 LINUX 环境下运行，因此在这个环境下需要类似于批处理的文件设计。VCS 仿真设计需要的环境设置有以下几点：需要编辑相关的.f 文件，这个文件主要是文件的目录索引输入，另外需要编辑一个启动命令文件

5.2.4 功耗仿真

针对综合完成的网表，需要进行功耗仿真。在前期的综合中，也会出现一个功耗结果，这个结果计算芯片的面积得到的。因此，在初次使用 DC 进行综合时得到的结果是比较大的，不准确。在使用 DC 进行功耗仿真中，需要额外的对电路运行时，进行整体的一个评估。进行评估的方法是在验证文件中写入一个命令，将库中的翻转文件写入，通过运行整个电路仿真，得到另外的一个翻转文件，这个翻转文件可以整个 ASIC 中门工作的一个记录。

5.2.5 后端版图布局布线

电路进行综合以后得到门级网表，当确保时序收敛且门级动态仿真通过后，可以利用 SYNOPSYS 的相关版图设计工具进行版图设计，这类的版图设计主要有 ICC,Astro 等。版图设计工作主要包括布局规划，布局，时钟树综合，布线等。

5.2.6 电路后仿真

版图设计完成后，需要从版图中提取寄生参数信息，并将这些 RC 参数转换为标准延迟.sdf 格式反标回设计中进行后仿，以验证版图设计的功能和时序是否正确。

5.3 循环缓存功耗优化对比

通过前面几章的阐述和说明可以知道，在利用循环缓存结构后，处理器能够像原来的结构那样正常的工作，并且真正起到降低功耗的作用。如下表 7.1 所示：下表中是经过 DC 后得到的仿真出的功耗结果：

表 5.1 中给出了对利用传统循环缓存结构和三段缓存结构的面积对比和功耗对比，分析是基于 SMIC 的 0.13μm 工艺，工作电压为 1.2V，温度为 25℃，工作频率为 20MHz。

表 5.1 存储结构面积和功耗对比

存储结构	面积	功耗
原始存储结构	151144.6250 μm^2	0.2024mW
基本缓存结构	175347.1250 μm^2	0.1064mW
三段缓存结构	180612.4375 μm^2	0.0870mW

通过表 5.1 可以看出，利用缓存结构后，程序存储器部分的面积增大了大约 16%，而功耗降低了大约 50%。而利用了三段缓存结构后，在原有的缓存结构面积的基础上增大了大约 3%，而功耗降低了基本缓存结构的 20%左右。可以看出使用缓存结构可以有效的降低程序存储器的功耗，并且三段式的缓存结构可以有效的对基本缓存结构的功耗进行更进一步的优化。

将整个结构全部放置在处理器中，全部审核在不同的算法结构情况下，得到的功耗结果，如下图所示：

表 5.2 中表示了助听器语音处理各算法的运行周期比例关系，可以看到在以上操作中 WOLA 算法和 NOISE_REDUCE 算法占据了极大的比例。其中 WOLA_A 和 WOLA_B 算法中最主要的运算是 FFT 运算和 IFFT 运算，FFT 和 IFFT 运算的运行周期为 6042，占总的运算指令周期的 34.8%。

表 5.2 助听器语音处理各算法运行周期比例

操作	运行周期	百分比
IN	32	0.2%
OUT	64	0.4%
WOLA_A	3597	20.7%
NOISE_REDUCE	7948	45.7%
WDRC	1397	8.1%
WOLA_S	4365	25.1%

在 WOLA 算法中包含了大量的 repeat 指令操作，循环体的指令有 16 条，根据 64 点 FFT 的 6 级运算，分别执行次数为 32x1 16x2 8x4 4x8 2x16 1x32，即循环指令中，执行 16 条指令，每次循环指令 32 次，并且由于 FFT 和 IFFT 运算是 6 级的，因此相邻的循环操作是相同的，执行 6 次。在 FFT 和 IFFT 操作中，循环指令操作有很大的重复性，因此在实际使用三段式进行功耗优化时，可以既降低主程序存储器的功耗，也可以降低程序存储器的功耗。

表 5.3 程序存储器在各算法下比较

算法选择	单程序存储器	基本循环暂存	三段式循环暂存
WOLA	0.2024mW	0.1064mW	0.0870mW
WOLA+WDRC	0.2303mW	0.1285mW	0.1091mW
WOLA+WDRC+NOISE_REDUCTION	0.3406mW	0.2088mW	0.1895mW

进过功耗评估，在表5.3中可以看到，当处理器仅仅进行WOLA运算时，三段缓存结构将节省程序存储器功耗大约60%。而在加入了WDRC运算后，三段缓存结构的功耗优化为53%。在整个助听器处理器工作中，利用三段的缓存结构可以降低45%的功耗。经过DC分析得到的面积统计结果，利用三段式的缓存结构，增加的面积占整个处理器的4%，而功耗减少了0.16mW，占总芯片功耗的20%，功耗优化特性很可观。

5.4 功能验证

该专用指令集处理器主要是进行语音信号的处理，针对听障人士的听力特点所设计的，正如前文中所述的，在设计中是以多通道分离算法为基础，然后以宽动态压缩为主要算法，配合相关的噪声消除和反馈消除等算法完成对整个助听器的设计工作。

本项目中，将预先录制好的声音信号转换为数字信号，通过外置的麦克风将声音信号读入，然后通过处理器处理，最后得到适合听障人士的语音信号。将输

出信号通过语音波形显示软件显示。图 5.3 中显示的是输入信号的波形图，图 5.4 中显示的输出信号波形图。

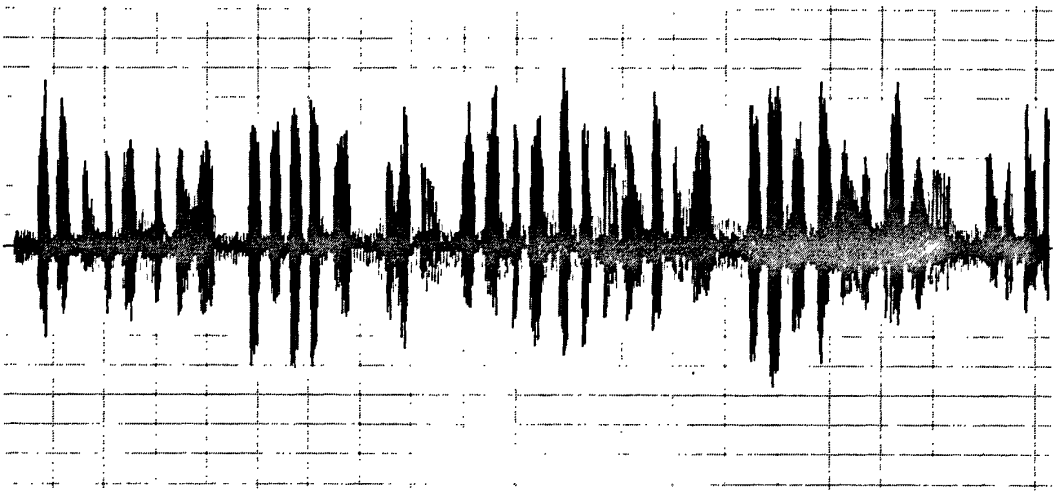


图 5.3　输入语音信号波形图

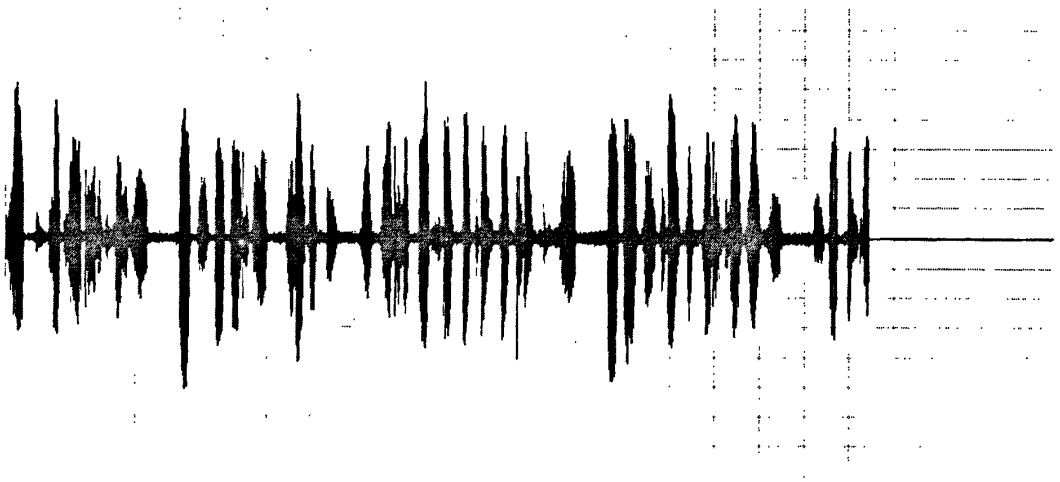


图 5.4　输出语音信号波形图

图 5.4 中显示的是输出信号的信号强度波形图。

由于是采用的宽动态压缩算法，增强在较高频率的信号的强度，因此在输出的波形文件中按照不同频率中信号强度不同可看出，图 5.5 中是输入信号的波形能量强度图，声音信号纵轴值越高，则表示频率越高，而图中越量，表示能量越大，从图 5.5 中看出，在高频率段，声音信号强度很弱，在图 5.6 中，则显示的是处理后的声音信号的强度值，可以看到在处理过后，在高频率段中，声音信号的强度明显增强了。

在利用助听器算法的 WOLA 算法和 WDRC 算法，对信号进行了分通道的加强，根据具体数据写入的差别，会对信号进行增强处理，因此在频率段可以看到这些信号明显的增强了，声音能量的强度增大了。

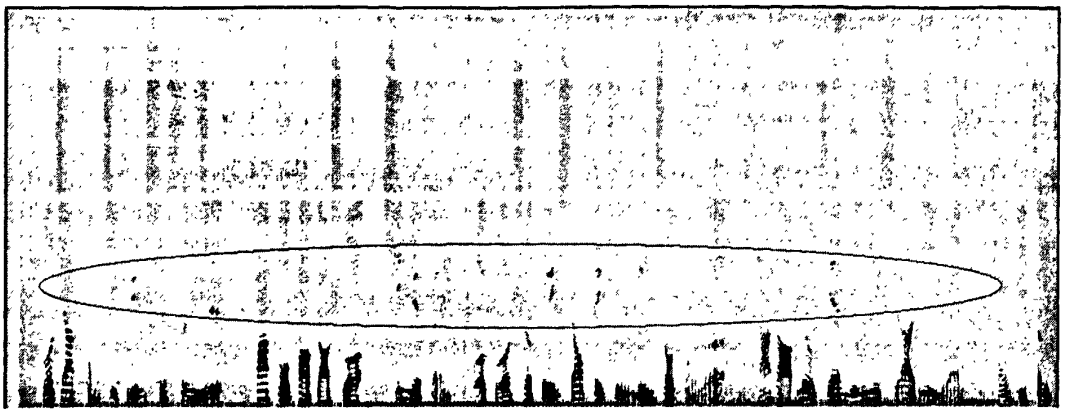


图 5.5 输入语音信号能量波形图

图 5.6 输出语音信号能量波形图

图中 5.5 和 5.6 中圈中对比得到的结果就是在相同的频率阈内，纵轴表示能量，而横轴表示时间，颜色越深表示能量越大。在中高频段，颜色变深，表示能量被放大，将这部分的声音信号放大，在功能上验证可知，整个处理器设计能够实现高性能放大语音信号。图 5.7 是整个专用指令集功耗报表。

```
36
37 Global Operating Voltage = 1.2
38 Power-specific unit information :
39     Voltage Units = 1V
40     Capacitance Units = 1.000000pf
41     Time Units = 1ns
42     Dynamic Power Units = 1mW    (derived from V,C,T units)
43     Leakage Power Units = 1pW
44
45
46   Cell Internal Power  = 614.7357 uW   (95%)
47   Net Switching Power  =  32.3280 uW    (5%)
48                          ---------
49 Total Dynamic Power    = 647.0637 uW  (100%)
50
51 Cell Leakage Power     =  89.2505 uW
52
```

图 5.7 专用指令集处理器功耗报表

整个处理器的数字部分芯片面积为 0.729mm^2，面积较小。在使用了加速单元

设计，专用模块设计，存储器低功耗设计后，使得整个处理器功耗大大降低，在时钟频率为 20Mhz 频率下，整个处理器的平均功耗为 736.314μW。功耗较低。

整个专用指令集处理器设计是按照面积尽可能小，助听设计功能尽可能好，功耗尽可能低的设计方法。由于本项目中数字助听器设计是属于一种移动仪器，因此，在设计中对功耗的要求非常高，从上面的功耗结果可以看出整个设计是符合低功耗设计要求的，为整个处理器功耗在微瓦级提供了可能。下面部分是整个芯片的版图和整体的功耗。图 5.8 是专用指令集处理器芯片面积报表。

```
16
17 Number of ports:                11
18 Number of nets:                437
19 Number of cells:               115
20 Number of references:           16
21
22 Combinational area:       192858.586803
23 Noncombinational area:    532455.292610
24 Net Interconnect area:      undefined  (No wire load specified)
25
26 Total cell area:          725313.879413
27 Total area:                 undefined
28
```

图 5.8 专用指令集芯片面积报表

5.5 ASIP 芯片版图和验证结果

图 5.9 中是助听器 ASIP 物理版图

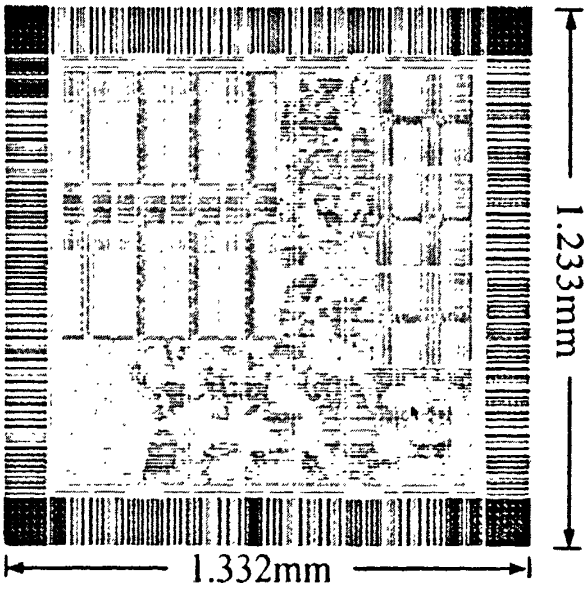


图 5.9 助听器 ASIP 物理版图

上图 5.9 是经过布局布线后得到的 ASIP 的芯片版图图片。该芯片采用的是

smic0.13μm 的工艺，芯片的整个面积是 1.233mmX1.332mm。通过后端仿真验证得到，该处理器电路最高频率可以工作在 50MHz。使用片上的 SRAM 有 8KB，验证得到的功耗结果为162$\mu W/MHz$，低于数字系统要求的500$\mu W/MHz$项目设计要求，满足低功耗处理器的设计目标

5.6 小结

本章对整个助听器处理器进行了 FPGA 验证和 ASIC 设计。文章首先针对FPGA的验证平台进行了介绍，介绍了整个FPGA验证平台中的三部分模块(FPGA芯片，FLASH 芯片和数字音频芯片)。文章随后介绍了该处理器的 ASIC 实现，包括综合部分，功耗分析，布局布线等流程。然后利用上面提到的功耗验证的流程对程序存储器中用到的循环缓存结构进行了验证，验证表明该结构能够有效的减少程序存储器和整体处理器的功耗，并且不影响整体处理器的功能和性能。然后对助听器功能进行了验证，通过波形分析对比，可以得到该助听器能够有效的在多通道分离的基础上对不同频率信号进行声音放大，增强高频率声音的强度。本章最后给出了整体处理器的芯片版图和整体处理器的工作频率和功耗，经过仿真验证表明，整体功耗为162μW/MHz，满足系统的系能要求。

结　论

全文总结

专用指令集处理器设计最为嵌入式系统设计的一个发展方向，越来越受到设计界和学术界的关注。本文中主要针对的是助听器系统所设计的处理电路，为了实现极低功耗和相关设计高效的特性，所以采用了 ASIP 设计方法。

本文首先从 ASIP 概念入手，介绍 ASIP 设计的主要研究方向，然后介绍了 ASIP 设计中针对低功耗设计方面的优势。通过介绍本项目中专用指令集处理器的内部结构和相关的指令集，对 ASIP 整个系统有了清晰的了解。然后以本项目的设计实例——助听器处理器为例。进行了相关的算法分析，并且将相关算法和硬件指令集相结合，设计出了汇编程序，实现整个助听器设计。

本文针对 ASIP 设计的特点，通过分析助听器算法中 WOLA 算法和消噪算法中各个运算所占据的比例，决定采用硬件加速模块设计方法，对消噪运算中指数运算，对数运算等运算进行加速运算设计，设计了专用模块和专用个指令。由于 WOLA 运算中，FFT 运算占据了很大比例，所以针对 FFT 运算中的核心运算——蝶形运算进行专用模块的设计。通过设计专用运算模块和专用指令，将整个蝶形运算由原来的 12 个时钟周期减少到了 4 个时钟，提高了工作效率，减少了功耗。

本文从硬件电路结构的角度对处理器进行再分析，为了降低处理器中程序存储器的功耗，从存储器电路的内部硬件结构，到外部功能都进行了详细的分析。为了实现低功耗改进，从降低电压，改变存储器电路内部结构，到划分电路模块设计，进行了诸多的尝试，最终选择使用循环缓存设计方式来优化电路结构。

本文设计的循环缓存结构基于本项目的专用指令集处理器设计，有非常好的针对性和非常高的功耗优化程度。由于是针对该处理器的指令集设计的电路硬件结构，符合专用指令集（ASIP）的设计思想，为最大限度优化程序存储器功耗，在设计中分别采用了一段式结构，多段式结构和复合结构，功耗优化程度很高。

本处理器采用 SMIC0.13μm 的工艺，采用 1.2v 的供电电压。通过编写硬件描述语言 verilog HDL 描述其硬件结构，在 modelsim 中进行前端仿真，然后在 Design Compiler 中进行综合，然后经过布局布线，完成版图设计后，通过后仿，验证功能和功耗特性。

经过仿真验证表明，在使用循环缓存电路结构后，程序存储器的功耗减少了 45%，降低整个处理器功耗的 20%。

该助听器处理器经过布局布线，得到芯片面积为 1.23mm x 1.32mm，整个处

理器功耗为 810μW（包括外围 A/D 部分电路），符合将整体功耗限制在μW 级别的要求。

本文研究的结果在专用指令集处理器中有非常好的使用价值和借鉴意义，创新之处在于：

1.基于专用指令集处理器设计，进行功耗优化。针对特定的算法实现，优化专用指令集结构。

2.设计基于专用指令集的循环缓存结构，通过不断改进结构模式，对程序存储器进行功耗优化。

改进及后续工作建议

本文通过采用 SIMC.13 工艺，设计出了数字助听器处理器，采用多种算法设计和结构设计，降低功耗。本文的循环缓存结构电路在优化程序存储器上面有非常好的效果，为以后设计低功耗处理器有很好的借鉴意义。

同时，由于是专用指令集处理器，在提升处理器性能上面还是有许多的不足，为了更好的适应以后专用指令处理器发展，需要提升一下几点：

第一：对于低功耗处理器，在低功耗技术使用上面还有很大的空间，例如电源管理技术，该技术没有使用到整个处理器的控制上面，如果能够能够使用电源门控技术，整个处理器的功耗优化程度会更高。

第二：对于专用指令集处理器而言，为了增强功能，需要添加额外的功能模块，例如在该处理器中，为了提高语音效果，需要额外的增加相关的算法，包括噪声消除，反馈消除算法等。加上这些算法的硬件模块后，可以一方面提升芯片的整体性能，另一方面减少芯片的功耗。

第三：针对处理器内部的电路而言，本论文只是改进了程序存储器设计，对于处理器中的数据存储器，设计相对复杂，没有固定的设计模式，因此对这种处理器的改进一方面需要从算法角度入手提升，另一方面需要采用更加先进的工艺技术。

参 考 文 献

[1] Tensilica, Xtensa. http://www.tensilica.com, 2001-4-5

[2] Gonzales R, Xtensa. A Configurable and Extensible Processor. Micro IEEE, 2000, 20(2): 60-64

[3] Parker Hannifin. ARC Core Ltd. ARC Programmers Reference Manual, Dec. 1999, 56-58

[4] Parker Hannifin. ARC Core Ltd. ARC tangent Processor, http://www.arccores.com, 2001-7-8

[5] Leibson S H. Jazz Joins VLIW Juggernaut – CMP and Java as an HDL Take System-on-Chip. Microprocessor Report, 2000, 10(5): 56-59

[6] Kievits P, Lambers E, Moerman C. DSP Technology for Telecom Baseband Processing, Signal Processing Applications and Technology(ICSPAT), Toronto, CA, 1998, 17(5): 35-39

[7] Kucukcakar K. An ASIP Design Methodology for Embedded Systems. Proceedings of the 7th International Workshop on Hardware/Software Codesign, 1999, 17-21

[8] Veendrick H J. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. Journal of Solid-state Circuits, 1984 No.4, Vol. SC-19: 468-473

[9] Lee D, Kwong W, Blaauw D. Analysis and minimization techniques for total leakage considering gate oxide leakage, DAC'03, Anaheim, California, USA, 2003, 175-180

[10] Shiue W. Leakage power estimation and minimization in VLSI circuits, ISCAS, 20010, 178-181

[11] Vivek De. Leakage-tolerant design techniques for high performance processor, proceedings of 2002 international symposium on physical design, san diego, California, USA, april 7-10, 2002, 28

[12] Koushik K D, Richard B B. Ultra Low-leakage power strategies for sub-1v VLSI: novel circuit styles and design methodologies for partially depleted silicon-on-insulator CMOS technology, processdings of the 16th international conference on VLSI design design (VLSI 03), New Delhi, India, 2003, 291-296

[13]陈中建. 低功耗 CMOS IC 设计. 第一版. 北京:北京大学微电子学院, 2001, 128-134

[14]Nebel W, Mermet J P. Lowpower design in deep submicron electronics, Norwell, MA, USA: 1997, 68

[15]Mudge T. Power: A first-class architectureal design constraint. IEEE Transaction on Computer, April 2001, Vol.34, No.4: 68-78

[16]Ghosh A, devadas S, Keutzer K. Estimation of average switching activity in combinational and sequential circuits, proceedings of the 29th ACM/IEEE conference on Design automation conference, june 1992, 253-259

[17]Rajsuman. Design and test large embedded memories: An overview. IEEE Design&Test of Computers, 2001, 18(3): 16-27

[18]Panda P R. Data and memory optimization techniques for Embedded Systems. ACM Transactions on Design Automation of Electronic Systems, 2001, 6(2) : 149-206

[19]Karandikar, Parhi A. Low power SRAM design using hierarchical divided bit-line approach. Computer Design: VLSI in Computers and Processors, Austin, TX, 1998, 82 – 88

[20]周润德. 数字集成电路——电路，系统与设计. 第二版. 北京: 电子工业出版社, 2005.7, 45-47

[21]周润德. 数字集成电路——电路，系统与设计. 第二版. 北京: 电子工业出版社, 2005.7, 460-463

[22]Jan Rabaey, Anantha Chandrakasan. Digital Integrated Circuits-A Design Prespective. 北京: 清华大学出版社, 2004, 529

[23]Chen C W. A Fast 32K*8 CMOS SRAM with Address Transition Detection. IEEE J. Solid-State Circuits, 1997, 22(4): 533-537

[24]Mutoh S, Douseki T. 1-V power supply high-speed digital circuit technology with multi-threshold voltage CMOS. IEEE J.solid-state circuits, Vol2: 34-37

[25]Chen C W. A Fast 32K*8 CMOS SRAM with Address Transition Detection. IEEE J.Solid-State Circuits, 1997, 22(4) : 533-537

[26]Karandikar A, Parhi K K. Low Power SRAM Design using Hierarchical Divided Bit-line Approach. Internal Conference on Computer Design. IEEE, 1998, 82-89

[27]Seevinck E. Current-Mode Techniques for High-Speed VLSI Circuits with Application to Current Sense Amplifier for CMOS SRAM. IEEE J.Solid State Circuits, 1991, vol.26, No.4: 525-536

[28]Neil W, David H. CMOS VLSI Design: A Circuit and System Perspective. Third edition. Boston: Addison-Wesley, 2005, 520-535

[29]Singh S, Azmi S. Architecture and design of a high performance SRAM for SOC design. IEEE Design Automation Conference, 2002, 447

[30]Angiolini F, Benini L, Caprara A. An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning. Transactions on Computer-Aided Design of Integrated Circuites and Systems, 2005, 24(11): 1660-1676

[31]Ali, Aboelaze K. Modified Hotspot Cache Architecture: A Low Energy Fast Cache for Embedded Processors. IC-SAMOS Embedded Computer Systems, 2006, 35

[32]Barnes P. A 500MHz 64b RISC CPU with 1.5Mb On-Chip Cache. IEEE International Solid State Circuits Conference,Digest of Technical Papers, 1999, 86-87

[33]Lea H L, Moyer, Arends B. Instruction fetch energy reduction using loop caches for embedded applications with small tight loops. IEEE. Low Power Electronics and Design, 1999, 267 – 269

[34]Bellas N, Hajj I, Polychronopoulos C, and etal. Energy and Performance Improvements in Microprocessor Design Using a loop Cache.Proceedings of the International Conference on Computer Design, 1999, 378-383

[35]Ann G R, Susan C, Frank V. Exploiting Fixed Programs in Embedded Systems:A Loop Cache Example,IEEE Cimputer Architecture Letters,2002,1(1):2-2

[36]Keutzer K, Malik S, Newton A R. From ASIC to ASIP: the next design discontinuity. Computer design, 2002, 14(11): 84-90

[37]Carro L, Pereira G A, Alba C, and etal. System Design Using ASIPs. Proceedings of the IEEE Symposium and Workshop on Engineering of Computer Based Systems, 1996, 80-85

[38]Dake Liu. Embedded DSP Processor Design. 2008: 46-48

[39]David A, Patterson. Reduced instruction set computers.Communications of the ACM, 1985, 28(1): 8~21

[40]Neeraj Magotra, Sanmati Kamath, Frank Livingston. Development and fixed-point implementation of multiband dynamic range compression (MDRC) algorithm. Conference Record of the 34th Asilomar Conference on Signals, Systems and Computers, 2000. Vol.1: 428-432

[41]Anis M, areibi S, Mahmoud M, and etal. Dynamic and Leakage Power Reduction in MTCMOS Circuits Using An Automated Efficient Gate Clustering Technique. The 39th DAC, New Orleans: 2002, 89-94

[42]Arm, Artisan, Physical IP overview, 2008, 4-5

[43]Stephen Theobald. the design of the physical of the SOC IP design. http://www.arm.com/products/physicalip/product_overview.html, 2008-5-6

[44]Jayapala, Barat M. Clustered loop buffer organization for low energy VLIW embedded processors. Computers , Volume: 54 , Issue: 6, 2005, 672 - 683

[45]Brackenbury. An instruction buffer for a low-power DSP. Advanced Research in Asynchronous Circuits and Systems, 2000. Eilat: 176 - 186

[46]Sias J W, Hunter H C. Enhancing loop buffering of media and telecommunications applications using low-overhead predication. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium. 2001: 262-273

[47]Dankanikote P, Jin Hwan Park, Yul Chu. Branch Prediction and Power Reduction Techniques in the Clustered Loop Buffer VLIW Architecture. Communications, Computers and Signal Processing, 2007. BC Victoria, 2007, 82-88

[48]Wu Q, Pedram M, Wu W. Clock-gating and its application to low power design of sequential circuits. In: IEEE Custom Integrated Circuits Conference; New York; 1997, 57-59

[49]Wu Q, Pedram M. Clock-Gating and Its Application to Low Power Design of Sequential Circuits. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 2000, 47(3): 415-420

[50]Li H, Bhunia S, Chen Y. DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design. IEEE Transactions on Very Large Scale Integration(VLSI) Systems, 2004, 12(3): 245-254

致 谢

本毕业论文是我研究生学习的一个总结，在本文完成之际，衷心感谢所有指导和帮助过我的老师，朋友和亲人。

本文是在胡锦，黑勇两位导师的悉心指导和鼓励下完成的。从论文的选题，具体的工作到撰写的过程，都凝聚着两位导师的心血和汗水。胡老师渊博的学识，精益求精的治学态度，谆谆善诱的教学方法，严谨的工作作风，崇高的事业追求和朴实的生活态度，将使我终身受益。黑老师严于律己，宽以待人的高尚品质和献身科学，忘我工作的精神是对我永远的鞭策。在此谨向两位老师表示衷心的感谢和崇高的敬意！

我的毕业设计是在中国科学院微电子研究所二室低功耗处理器和医疗电子项目组完成的。非常感谢乔树山老师在我在二室学习期间给予了我很大的帮助和指导。感谢陈黎明博士在我完成论文过程中给予的帮助，理解和支持。非常感谢项目组于增辉博士，袁甲博士，薛金勇博士对我的帮助，感谢你们在我学习和研究中给予我的耐心的指导和帮助。感谢微电子所朱勇旭，李春阳，赵慧冬，巨浩，张振东等师兄，师姐对我的帮助。感谢同学李海龙，戴建平，张心冲，朱婷，吕俊盛，叶茂，袁莉，于伽等对我的帮助。与你们在日常生活中的交流和学术中的讨论都让我受益匪浅。

在论文的研究工作进行中，也得到了物理与微电子科学学院曾云老师，颜永红老师，陈迪平老师，曾建平老师，张红南老师，杨红官老师，晏敏老师的教诲和帮助，由衷的感谢他们。

感谢师兄刘观承，刘清波，师姐黑花阁，杨亚光，师弟陶可欣，师妹翟媛以及其他师弟师妹对我的关心和帮助。

最后，我要感谢我的父母和亲人们，无论在什么情况下，他们都给我无微不至的关怀，支持和爱护，没有他们就没有我的一切。

最后向参加论文评审和答辩的专家表示由衷的感谢！

附录 A　攻读学位期间所发表的学术论文目录

[1] 胡锦，李新泽，黑勇. 基于低功耗 ASIP 的循环缓存的设计.微电子学与计算机，已接受，待发表，文章编号 20110509