摘 要

数字信号处理（DSP）技术已成为人们日益关注并得到迅速发展的前沿技术。然而，DSP 技术实现主要载体之一的 DSP 处理器的性能从体系结构到指令系统等诸方面虽具有灵活的可编程性，但其性能的提高往往是以相对较慢的计算速度为代价的，不能满足一些对数据处理速度要求很高的应用与需求，如实时视频处理，因此又出现了一种可以应用于 DSP 技术的新型计算方式，即可重构计算。可重构计算技术既具有软件编程的灵活性，又有 ASIC 方式实现 DSP 技术的高效性。

本文在对 DSP 技术、可重构计算技术及可重构硬件（FPGA）逻辑结构和可重构 DSP 系统结构进行剖析研究的基础上，首先提出了一个可重构 DSP 计算系统的基本结构；随后，从对相关的 DSP 算法及功能的研究出发，找出了两类与 DSP 实现相比更适合于 FPGA 实现的算法：一类是分时复用的算法，一类是基于卷积运算的 DSP 算法；在对 DSP 算法(FIR/IIR/DFT/DCT/DHT)的研究过程中，引入 CORDIC 理论，找到这几种算法实现中的共同的部分，设计出一个统一的可编程模块；然后利用这一模块设计出了一个可重构的并行处理系统结构；最后，利用 Altera 公司的 FPGA 芯片及综合仿真工具对所设计的系统结构的功能进行综合仿真，并将仿真结果与 C 语言软件编程及 MATLAB 仿真方式的结果做比较。比较结果表明，该设计对 FIR 滤波和 DFT 变换的结果与软件编程及 MATLAB 仿真的结果是一致的，证明本文中的可重构并行处理系统结构的设计是成功的，即在这一结构中，不需要改变系统的整体结构，只要为不同的系统功能配置不同的系统参数以及模块间互联网络结构，即可实现不同的 DSP 算法。这一可重构并行处理系统可以作为数据计算量大、对速度要求高的应用系统中的 DSP 计算引擎或者主机的协处理器来使用。

关键字：可重构计算；数字信号处理；CORDIC；FPGA；

Abstract

Digital Signal Processing (DSP) technique develops quickly and has been one of the advanced techniques. While although the performance of DSP processor has been improved in every aspect from its system architecture to its instruction architecture and have good flexibility, its flexibility is improved in cost of low computing speed, that is to say, current DSP processor can't meet with some application, such as real-time video processing. So a new computing method is needed, i.e. reconfigurable computing technique. It has not only the flexibility just like software, but also the high efficiency like ASIC.

In this thesis, on the basis of the analysis of DSP technique, reconfigurable computing technique and its hardware (FPGA) logic architecture, and kinds of reconfigurable DSP system architecture, two kinds of algorithms fit for implementation on FPGA are found firstly by the research of the related DSP algorithm and function. One is the algorithm whose different part can be implemented in different time. The other one is the DSP algorithm based on convolution. By the study of the DSP algorithms (FIR/IIR/DFT/DHT/DCT), the common module of these algorithms can be found using CORDIC theory. Also a reconfigurable DSP computing engine system architecture is designed, which can implement all those DSP algorithms according to different parameters and the different network without changing the whole system architecture. In the following, Synthesis and simulation are passed using FPGA from Altera. What's more, the result is compared with the result from C program and the simulation of MATLAB. It proves that the design in this thesis is good for it can implement different DSP algorithm in the same reconfigurable system and fits to be a coprocessor in the audio/video communication application.

Keyword: Reconfigurable Computing, Digital Signal Processing, CORDIC, FPGA

第一章　绪论

1.1 课题背景

在现代数字信号处理、数字图像处理及视频应用中，高速度和大规模的计算能力是十分必要的。为了实现这种应用，通常有两种方法：

第一种方法是利用硬布线(hardwired)技术。即要完成硬件中的运算，要么采用专用集成电路(Application Specific Integrated Circuit，ASIC)，要么采用由一组独立的元器件构成的板级(board-level)解决方案。

专用集成电路的设计专门用于实现某一特定运算，因此，它们在执行这一特定运算时相当迅速高效。然而，这种电路制造出来后不可再改变。一旦电路的某一部分需要修改，就必须将整个芯片进行重新设计和制造，尤其是要在大量已开发出来的系统中更换 ASIC 时，代价是非常昂贵的。另外，板级电路也存在一定程度的不灵活性，在应用中出现某些变化的情况下，也常常需要对电路板重新进行设计或者更换元器件。

第二种方法是利用软件可编程的微处理器，这是一种相当灵活的方案[1]。处理器通过执行一个指令集来完成运算。通过改变软件指令，就能在不改变硬件结构的情况下改变系统的功能。然而，这种灵活性是以性能的下降为代价的。处理器必须从内存读出每条指令并解释之，然后才能执行该指令。对于每一个独立的运算，这种处理的代价很高。因此，在速度方面，这种方法的性能要比专用集成电路的性能低的多。此外，在处理器制造时已经确定了程序所使用的指令集，如果需要指令集以外的指令则必须更新处理器的指令集。

由此可见，这两种方法有着各自的优缺点。若要取二者之长，则必须找到一种新型的计算方式，既保留软件为主导解决方案的内在可编程性和低成本，同时还要达到原有硬件解决方案的处理速度和低功耗，这种新方式就是可重构计算。

可重构计算技术是指:数字系统制造完成以后，其硬件结构可以根据需要重新配置的技术[2]。也就是说当机器制造完成后，机器运行时可在芯片上重写程序，从芯片外部读进程序，或可将功能规格自由加以变更，以求缩短研发期间或削减芯片数。它是一种结合了现有微处理器和 DSP 的“时间计算”方式及硬件 ASIC、FPGA 解决方案的“空间计算”方式的新型计算方式[3]，是一种可以解决速度、功耗、可编程性之

间矛盾的新方法。

随着可重构计算技术的发展，已有不少关于 DSP 算法在 FPGA 上实现的研究，比如 FFT 或者 FIR 滤波算法基于 FPGA 实现的研究，这些研究基本上是对某一个 DSP 算法的 FPGA 实现进行研究，而将多个 DSP 算法映射于同一个可重构计算系统中的研究尚未出现。

本文在对DSP体系结构、可重构DSP系统结构及可重构计算的研究基础上，提出了一个可重构的DSP计算引擎基本结构，并从相关DSP算法和功能的研究出发，找出了一类DSP算法特征，这类DSP算法用可重构计算方式实现比DSP处理器实现效果更佳。在此基础上，又设计了一个可以实现多种DSP算法的可重构DSP计算引擎系统结构，并对其功能进行了综合仿真。由于该系统能够在只改变系统参数、不该变系统整体结构的情况下完成多种低层的DSP计算，因此可作为数据计算量较大的音频/视频通信前端数据处理系统中的协处理器。

1.2 研究的内容及意义

可重构计算作为一种崭新的计算形式，其作用不止于简单地集成若干中小规模集成电路和仅仅作为 ASIC 器件廉价的代用品。可重构器件以及可重构计算系统的结构是一种与可编程通用处理器和专用硬件电路并列的结构形式，另外还有在此基础上产生的可重构 DSP 系统，都是值得我们深入研究的体系结构形式。

本文在对 DSP 体系结构、可重构计算技术进行深入的分析研究的基础上，主要在以下几个方面做了较深入的研究：

1) 结合了可编程性及 ASIC 特征的几种可重构 DSP 系统结构研究及性能优劣比较，并给出本文设计的可重构 DSP 计算引擎的基本结构框图。

2) 从相关 DSP 算法的分析出发，研究可重构计算技术的适应范围，给出这类算法的特征描述。

3) 引入 CORDIC 理论，研究如何将多个适合于可重构计算技术实现的 DSP 算法映射于同一个系统结构中。

4) 在以上研究的基础之上，设计一个可重构 DSP 计算引擎的系统结构，并基于 FPGA 对其进行功能验证。

基于本文的研究基础上的可重构 DSP 计算引擎系统结构，可按应用需求的不同将其与应用系统中主处理器进行不同程度的耦合，以作为系统的协处理器或可重构

处理单元来实现其高速、灵活的可重构性价值。

1.3 论文结构安排

本文是作者在从事 DSP 及可重构计算研究工作的基础上总结出来的，主要是在对 DSP 体系结构、可重构 DSP 系统结构的研究比较基础上，提出可重构 DSP 计算引擎基本结构框图，并从相关算法研究出发，找出不同算法的共同部分，设计出一个统一的可编程计算模块，并在此模块基础上设计出了一个可用作音频/视频及图像处理应用中的协处理单元的可重构 DSP 计算引擎系统结构，并以 FIR 和 DFT 为例对目前所完成可重构结构的设计功能进行验证。

本文的内容安排如下：

论文第一章，为绪论，对论文背景做简单介绍，说明本文的研究内容及意义。

论文第二章，对当前的 DSP 技术及可重构计算技术作介绍，研究 DSP 体系结构及可重构 DSP 系统结构，并提出本文设计的可重构 DSP 计算引擎的基本结构框图。

论文第三章，从 DSP 算法研究出发，找出可重构计算适用范围，提出移位加方式实现 FIR/IIR 滤波，并在此基础上，提出 DFT/DHT/DCT 的统一结构设计。

论文第四章，设计基于 FPGA 实现 FIR/IIR/DFT/DHT/DCT 算法的统一可编程模块。

论文第五章，利用第四章的统一可编程模块，设计一个基于 FPGA 实现 FIR/IIR/DFT/DHT/DCT 算法的系统结构，对该设计进行综合、仿真，并分析仿真结果。

论文第六章，作者对所做工作作了总结，给出相应结论，并对未来工作做了展望。

第二章　可重构 DSP 系统结构

DSP 处理器作为数字信号处理的主要载体之一，20 世纪 60 年代以来迅速得到广泛应用。作为微处理器大家庭中的一员，从低成本、低功率 DSP，到各种高端 DSP，DSP 的发展大约经历了起步、成熟、快速发展三个发展阶段，从容量、速度、功能各个方面都有了很大的发展及进步。然而，DSP 处理器依然不能完全达到某些应用中对于速度、灵活性等性能的要求，因此越来越多的 DSP 系统倾向于与其他逻辑器件，如 CPLD/FPGA 等结合来进行信号处理。

2.1 DSP 处理器体系结构

DSP 处理器体系结构的革新在很大程度上受到应用需求的影响，其指令集的设计是面向存储器和数字信号处理算法来进行性能优化的。当前，为了提供通信和多媒体处理、图像及图形处理、语音识别和语音融合、人工智能等方面的应用对高速计算的需求，DSP 通常采用 VLIW 和 SIMD（单指令多数据）等结构形式来加强处理器对数据的处理能力。

2.1.1　VLIW 结构

在 VLIW 处理器的硬件上，各功能单元共用大型寄存器堆，由功能单元同时执行的各种操作是由 VLIW 的长指令来同步，它把长指令中不同字段的操作码分送给不同的功能单元。相对于传统型 DSP 处理器，VLIW 处理器使用简单的指令集，一条指令只完成一个操作。这个处理器将简单指令并行地发射出去，并同时执行，有这样的多条指令构成一个超长指令字。由于使用了简单指令集后，简化了译码和执行操作。所以，相对于传统 DSP 处理器，可以采用更高的时钟频率。

VLIW 技术极大的提高 DSP 处理器的性能，但它也有缺点。由于它的指令字长增加了，所以较大增加了程序存储器的占用空间，使得 DSP 处理器的成本和系统开销随之增加。同时为了支持多个并行指令的执行，这种结构的 DSP 处理器要求必须有充足的指令译码器、总线寄存器和存储器带宽。

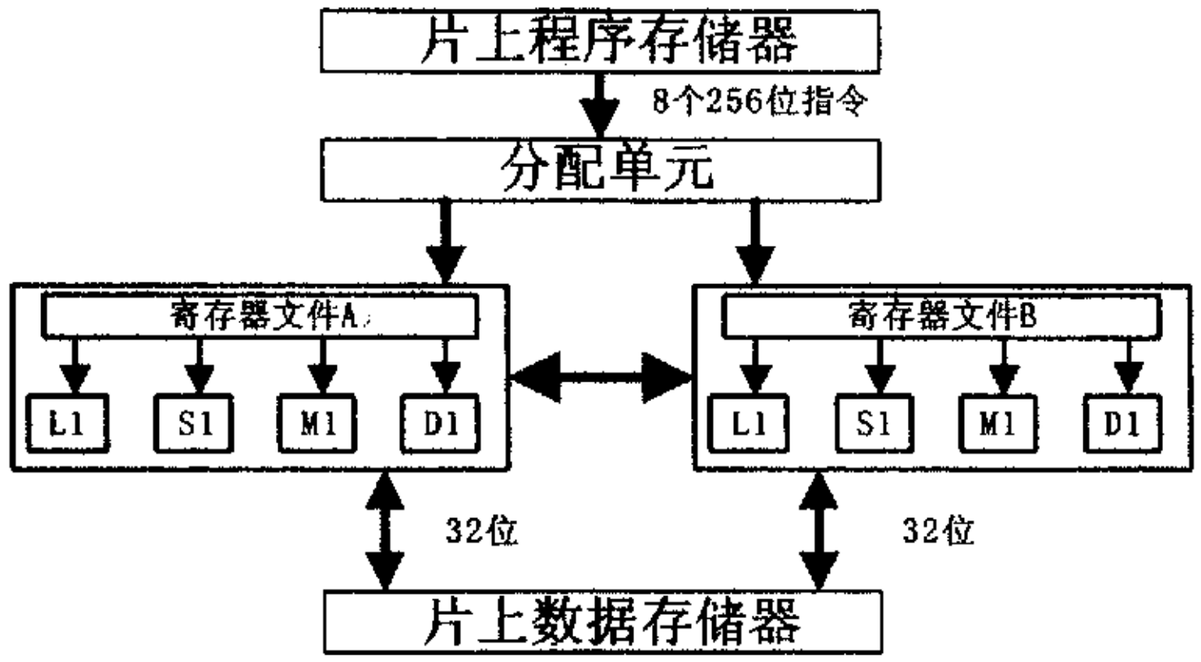


图 2.1 VLIW DSP处理器典型结构

2.1.2 SIMD 结构

SIMD（单指令多数据）处理器把输入的长数据分解为多个较短的数据，然后由单指令并行地操作，如图 2.2 所示。即处理器可以同时对多个数据进行进行操作，从而提高一些数学算法的计算性能。例如，一个 SIMD 乘指令可以在一个时钟周期内对不同的操作数据执行两个或两个以上的乘法操作。SIMD 使总线、数据通道等资源充分利用，可以极大提高某些向量计算的计算性能，在通信、多媒体信号处理和数字信号处理中有广泛的应用[4]。但是，这种结构只有处理并行算法时才是高效的。对于串行算法（算法中的结果作为下一个操作的输入），SIMD 处理器通常不使用。

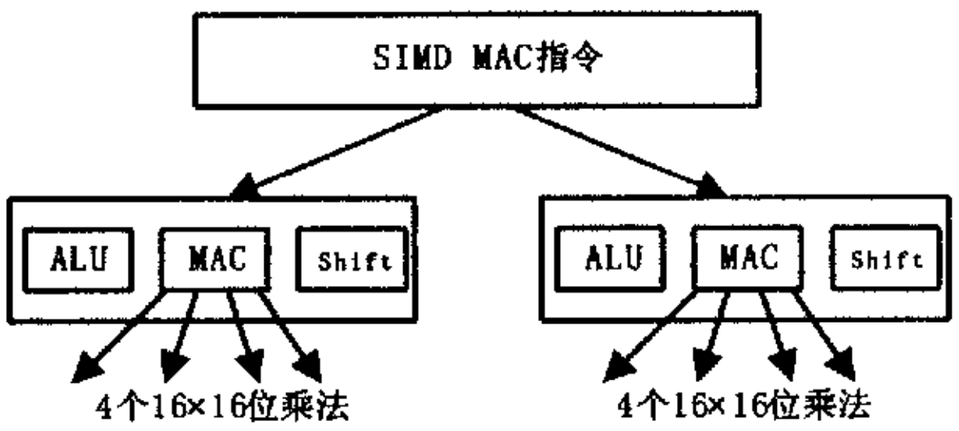


图 2.2 SIMD DSP处理器的典型结构

2.1.3 增加了指令和数据 Cache 的 DSP 处理器结构

在原有的 DSP 处理器结构基础上，为了追求更高的处理速度，DSP 处理器中开始采用 Cache 存储器。Cache 的使用是为了提高处理器在运算时取指令和取数据的速度，当需要的指令或数据在 Cache 中时，处理单元可及时地取得指令或数据。但是，如果需要的指令或数据不在 Cache 中，处理单元需要等待，直到 Cache 中装入了所需数据为止。因此，程序执行的时间与 Cache 有关。在有 Cache 的 DSP 处理器中，

程序执行时间是不完全确定的，与程序执行过程中处理的数据有关。考虑到 Cache 应用会带来处理性能上的改善，设计者还是将 Cache 引入到 DSP 处理器结构当中。由此带来的 DSP 应用的问题，如实时性和执行时间不定等，有望在今后的应用中得以解决。处理器开发者希望能给用户提供指令级的仿真工具，用于更精确估计软件的执行时间。但是，目前还只能靠程序员人工地在这方面做一些有限的工作[5]。

在 DSP 处理器中采用 Cache 存储器，还将会引起 DSP 处理器本身的成本和功耗的巨大增加，这些方面也限制了带有 Cache 的 DSP 处理器的应用范围。

然而，尽管今天的 DSP 处理器的处理速度很快，对许多 DSP 应用来说很有用，但仍有一些应用要求其性能再进一步提升。比如在实时视频处理中，对于系统性能的要求就极高，因此几乎所有只具备简单功能的通用 DSP 都不具备这么高的实时处理能力。然而，现代数字信号处理应用往往需要进行一些计算量大、速度要求高的运算，比如在数据通信和图像处理这样的应用中，需要强大的计算能力和实时处理能力。基于 DSP 的解决方案通常需要在单板上嵌入许多 DSP，以得到必需的处理能力，这无疑将增加程序资源开销和数据存储器资源开销。因此，为了实现高性能、低成本的目标，必须找到一种新型的计算方法，可重构计算方式恰恰弥补了这一空白[6]。

2.2 可重构计算

2.2.1 可重构计算的定义

可重构计算是一门新兴的技术，虽然早在 1960 年，可重构计算的思想就已经被提出，但是由于当时器件技术的限制，这种思想并没有得到广泛的关注和深入的研究。直到 80 年代末，才出现了第一代可重构计算系统——DECPeRLE[7]，掀起了可重构计算研究的热潮。

目前，对于可重构计算并没有一个统一的标准定义，不同的研究者由于研究问题角度的不同，对可重构计算有着不同的理解，因而也提出了不同的定义。这里我们选择几个有代表性的定义来描述可重构计算的特征。

（1） 可重构计算技术是指，用在系统内作为硬件处理单元的可编程逻辑器件，根据需要对其内部结构、功能、连线重新配置，使固定的硬件实现多种多样的可编程解法。

（2） 可重构计算机是一种通过将计算单元进行制造后（Post-Fabricate）空域连

接（Spatial Connection）以实现计算的器件。该定义中的两个要点是：一、制造后的功能定义；二、利用空域连接进行计算。所谓制造后，是指器件功能的定义是在 IC 器件出厂之后由用户完成的，而 ASIC 器件的功能是在芯片生产厂制造过程中完成的，在这个意义上可重计算系统与通用处理器十分相似。功能定义时间的延后意味着可重构计算系统可以由用户根据特定的应用背景实现多种不同的功能。空域连接指的是用逻辑门和连线来实现计算，与此相对的另一种完成计算的方式是用算术逻辑运算单元（ALU）和在 ALU 上实现的指令序列，后者被称为时域连接方式。

（3） 可重构计算系统就是通用（General Purpose）定制（Custom）硬件[8]。

这个定义突出通用和定制这两个特点。所谓通用就是可以用同一个硬件平台完成多种计算功能；而定制就是将硬件按照一个特定应用的结构、行为特点进行设计与优化。通用性是可编程处理器的优势，而通过定制获得高性能是 ASIC 器件的特长。这个定义说明了可重构计算可以兼具可编程处理器的通用性和 ASIC 器件高性能的优点。

从以上对于可重构计算特征的描述中可以看出，可重构计算的发展趋向就是要填补软硬件之间的空白，既要达到比软件方法更高的性能，同时具有比硬件更高水平的灵活性。包括现场可编程门阵列（FPGA）在内的可重构器件，都含有一个由计算单元构成的阵列，计算单元的功能是由多个可编程二进制的配置位决定的。有时也称为逻辑模块，这些块可以通过一系列的可配置布线（Routing）资源连接起来。用这种方法，通过对逻辑模块中电路的逻辑进行计算，并且利用可配置布线将各逻辑模块连接起来以生成所需电路，这样，就可以将定制数字电路映射到可重构硬件上。

2.2.2 可重构计算硬件结构

目前大多数可重构计算器件由逻辑单元阵列和互连结构两个主要部分组成。以下将分别介绍这两个主要组成部分的内部结构。

2.2.2.1 可重构硬件的逻辑单元阵列

可重构硬件的逻辑单元[9]的设计也因不同系统而各异。其内部结构可以像 3 输入查找表（LUT）一样简单，也可以像 4 位的 ALU 一样复杂。逻辑模块的粒度（Granularity）通常就是指这种逻辑单元结构的大小。

FPGA 中逻辑单元体系结构的设计是一个复杂性与通用性权衡的过程。可以用粒度(Grain)来描述 FPGA 中逻辑单元的复杂程度。从理论上说，逻辑单元由细到粗可以是晶体管级、与非门级、MUX 级、LUT 级、PLA 级直到 CPU 级，可以大致分为粗粒

度(Coarse Grain)与细粒度(Thin Grain)两大类。XC4000 系列、FLEX8000 系列 AT6000 系列属于细粒度 FPGA，它们包含数量较多、内部结构较为简单的逻辑单元。目前，大多数商用 FPGA 的逻辑单元主要由 4-LUT(四输入查找表)和触发器组成。图 2.3 为具有细粒度逻辑模块的 Xilinx6200 系列的一个功能单元图[10]。

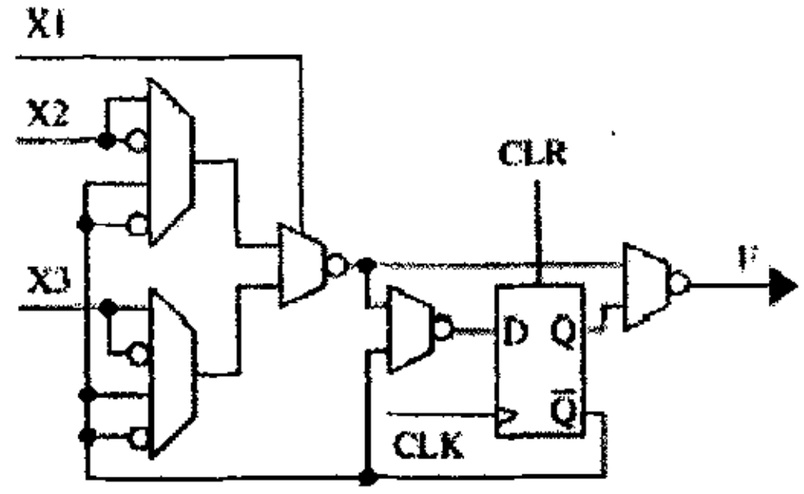


图 2.3 Xilinx 6200的功能单元

细粒度模块有助于位级处理，而粗粒度块则在标准数据路径应用方面有更好的优化。为了有效地支持各种类型的运算，有些系统结构在同一可重构阵列中采用了不同大小或类型的模块。由此可以得到另一种 FPGA 逻辑单元的分类方法，即同构型 FPGA 和异构型 FPGA。FPGA 中所有逻辑单元为同一类型的称为同构 FPGA；包含多种类型逻辑单元的 FPGA 称为异构 FPGA。同构的 FPGA 有很好的通用性。能被现有的综合工具很好的支持，但它在实现某些具有特定要求的应用时效率较低；异构 FPGA 在牺牲通用性的前提下，在某些应用中能够获得更高的速度和更强的功能，目前大多数商用的 FPGA 属于同构型 FPGA。

2.2.2.2 可重构硬件的互连结构

可重构架构中的互联资源用来将器件的可编程逻辑单元相互连接起来。这些资源通常是可配置的，且信号的路径是在编译或运行时间确定的，而不是在制造时。这种逻辑单元间的灵活互联，允许多种类型的电路结构映射到可重构硬件上，且每种都可以有其自己的互联需求。

可重构硬件中逻辑模块之间的这种布线同样非常重要，布线的好坏对于可重构硬件的整个面积的大小起着关键作用。然而，当 FPGA 中的逻辑模块的比例很高时，自动布线工具往往很难完成各块之间的必要连接。因此，好的布线结构对于确保一个设计能够成功的在可重构硬件上完成布局与布线是非常重要的。

在 FPGA 逻辑模块的体系结构方面已有很多的实验，同样，在互联结构方面也已经做过大量的研究。随着逻辑模块基本上标准化为基于 LUT 的结构，布线资源结构也基本上形成了岛式结构（island-style）为主(如图 2.4)，布线通道是一些不同

长度的导线，环绕着逻辑块。当然，在这种布线构架类型中，仍然存在着差异，如系统中导线与逻辑之间的比率、每根导线的长度、以及这些导线应该采用分段还是分层的方式连接等等。

大多 FPGA 构架把它们的布线结构组织成一个相对规则的片状布线资源，允许在块中沿着行与列进行快速而高效的通信。如图 2.4 所示即为 FPGA 的典型结构。

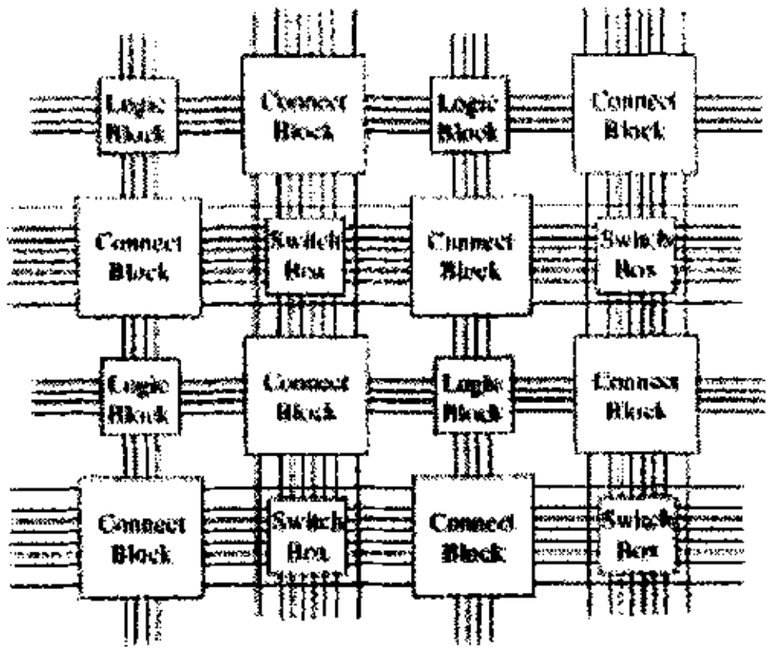


图 2.4 一种岛状的FPGA布线架构

2.2.3 可重构计算机制

如前所述，大多数可重构计算器件由逻辑单元阵列和互连结构两个主要部分组成，而这两部分均由配置开关(Switch)来控制，用户可以通过对这些配置开关的配置来完成指定的功能。

可重构计算器件实现配置开关的机制称为可编程技术(Programming Technology)。常见的可编程技术可以分为三类：一次性可编程技术、多次可编程技术以及无限次可编程技术，基于反熔丝（Anti-fuse）技术的可编程技术属于一次性可编程（OTP：One Time Programming）技术；基于 EPROM，EEPROM 和 Flash Memory 技术的可编程技术属于多次可编程技术；而基于 SRAM 技术的属于无限次可编程技术。可重构计算要求器件具备无限多次可编程的能力，所以一般采用基于 SRAM 的可编程技术。

当前的 FPGA 及可重构器件大部分是可编程的静态随机读写存储器，这意味着 SRAM 位（bits）被连接到 FPGA 的配置点上，因此通过对 SRAM 位编程就可以完成对 FPGA 的配置。于是，就可以像使用标准的静态 RAM 一样对这些芯片进行简单地编程和再编程。

2.3 基于可重构计算的 DSP 系统结构

为了弥补 DSP 处理器在进行数字信号处理中的不足，并充分利用可重构器件的优势，以达到各种 DSP 系统的要求，用可重构器件来实现部分或全部 DSP 系统功能很自然的被提了出来。

2.3.1 可重构计算系统

可重构计算系统也常被称作可重构的定制计算系统（CCM：Custom Computing Machine）。可重构计算是八十年代末、九十年代初开始兴起的一个研究领域；其基本特征是系统中有一个或多个可重构处理器（最典型的可重构处理器就是已进行管脚分配的、大容量的可重构 FPGA 器件），可重构处理器之间或可重构处理器与 ISA 结构处理器之间通过某种灵活的方式互相连接起来可以构成一个完整的计算系统。

可重构计算系统可以分为两大类：一是可以动态重构的计算系统，也即在线局部可重构的计算系统，其特点是可重构处理器中一部分硬件进行运算及信号处理的同时能够对另一部分硬件进行重构；其余的可重构计算系统则属于静态可重构系统。

在可重构计算系统中，全部或大量的关键算法由可重构硬件来实现，其方法为：根据具体算法得出符合可重构硬件实现的结构，并由结构设计出相应的电路，通过 CAD 系统将电路描述转化为以可重构硬件进行功能重构的配置代码。其实质是对可重构硬件中的基本功能单元通过丰富的互连资源进行结构重组，这相当于先在可重构处理器中定制出基本的可重构计算单元（如信号与图像处理中的乘法－累加器），再确定硬件系统设计策略（如全局并行局部串行、局部并行全局串行等），并在这些计算单元的基础上实现更复杂的规则阵列结构，从而完成复杂的计算任务。

可重构计算系统具有许多与可编程的 ISA 结构计算系统所不同的行为特征，其设计手段和方法也有着较大差异。由于可重构系统主要通过可重构硬件来实现各类关键算法，没有 ISA 结构通用系统中的指令开销问题，其计算性能往往高出通用指令集系统一个数量级以上；同时，可重构计算系统中可重构处理器功能也可以被动态改变，这对于需要自适应能力的系统是很有意义的。

通常，可重构硬件是与传统的微处理器耦合在一起来工作的，因为可编程逻辑通常不能有效地实现某些类型的运算，如可变长循环和分支控制。为了使可重构计算系统能够高效地完成应用需求，可以将那些不容易映射到可重构逻辑上的程序，交给主机处理器处理；而将那些能够由硬件实现的密集型计算映射到可重构逻辑上。

2.3.2 可重构 DSP 系统结构

对可重构 DSP 系统的划分已有多种标准，如可重构阵列中 FPGA 的连接方式、可重构块的规模、运行在可重构系统中的应用程序的粒度等等。目前已有的 DSP 可重构系统中分别包含了可重构逻辑资源和固定逻辑资源[11]。固定逻辑资源既可能是一个宿主机也可能只是一个微控制器，与 FPGA 集成在同一个板子上。则按照它们之间耦合程度可以将现有可重构 DSP 系统分为四类：

- 宿主机与可重构逻辑松耦合
- 处理器与可重构逻辑松耦合
- 处理器与可重构逻辑紧耦合
- 处理器、存储器与可重构逻辑紧耦合

2.3.2.1 宿主机与可重构逻辑松耦合

属于这类系统的可重构逻辑部分一般包含多个 FPGA，而固定部分则是一个宿主计算机，二者之间通过接口总线连接起来。可重构逻辑类似于宿主机协处理器，接收宿主机发出的指令和数据，并将结果通过 I/O 接口(如 ISA. PCI、并口等)传回宿主机，如图 2.5 所示：

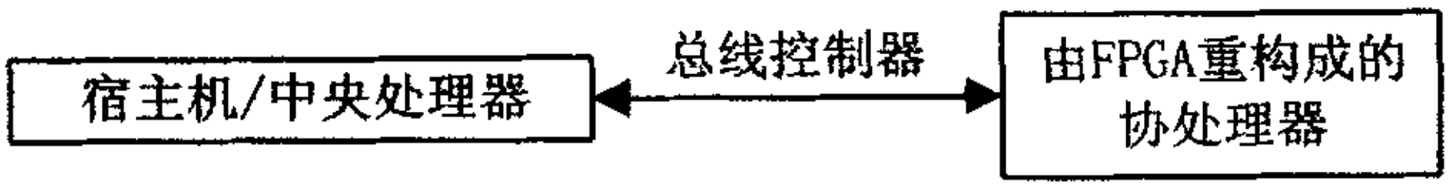


图 2.5 宿主机与可重构逻辑松耦合

这类系统的典型优点是设计简单。由于所有部件非常容易得到，所以可以快速地制造，方便地编程，还可以根据需要灵活地选用不同的宿主机和 CPU。但其缺点也是明显的，因为所有的数据交换都需要通过外部的接口总线来完成，这部分的性能成了系统的瓶颈，限制了系统的性能加速比。

2.3.2.2 处理器与可重构逻辑松耦合

属于这个分类系统的 FPGA 阵列直接与中央处理器通过独立的数据和控制总线进行连接，对 CPU 来说，相当与增加了一个多功能协处理器。与上一节结构相比，该结构最大改进在于取消了同宿主机的外部通用接口，从而大大提高了 FPGA 阵列同 CPU 的数据交换速度。这类系统的结构如图 2.6 所示：

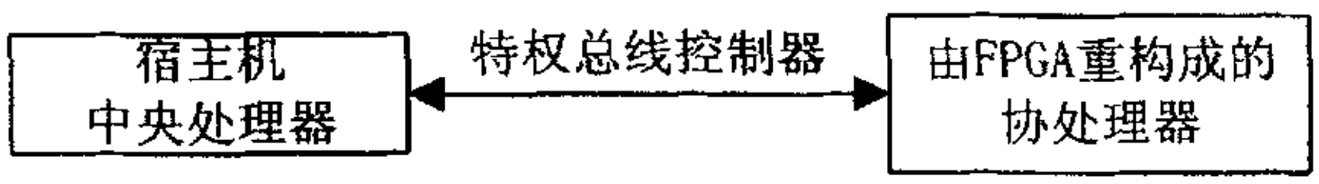


图 2.6 处理器与可重构逻辑松耦合

在这类系统中执行某个运算时，任务被分解到硬件部分(FPGA)和软件部分(中央处理器)，可以用硬件结构快速实现的运算被映射到 FPGA 构成的特定电路上，而简单的运算处理可控制功能则仍由 CPU 来完成。任务划分主要由用户来决定，很大程度上应该由编译系统来完成或由手工完成。

可重构逻辑在系统中的组织形式非常灵活，既可以配置成一个复杂的功能单元用来执行某些耗时的操作也可以配置成多个简单的单元来一次处理多个数据。对于 CPU 来说前者相当于指令集中增加了一条复杂指令，后者则相当于增加了单指令多数据(SIMD)的支持能力。

这类系统同样也有前一节系统的易用性特点。图 2.6 中的中央处理器可以根据需要选择不同的产品，只要该 CPU 支持类协处理器的工作方式即可。所以这个分类系统最适合于需要人为执行复杂指令的应用场合。

2.3.2.3 处理器与可重构逻辑紧耦合

上一节介绍的结构在性能上受处理器——协处理器接口限制。为此研究人员将这个接口设计为紧耦合系统来解决这个问题，这形成了第 3 类可重构计算结构，如图 2.7 所示。

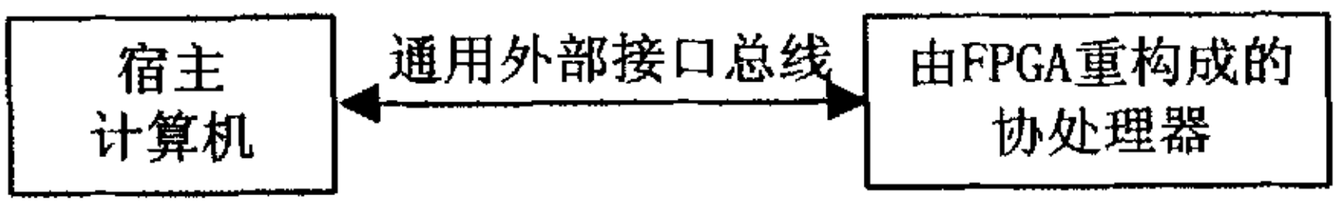


图 2.7 处理器与可重构逻辑紧耦合

通常，属于这个分类系统中的 CPU 不是通常商业中可以直接应用的成品，而是以核的形式提供设计模块，需要与可重构逻辑资源在芯片上集成起来。CPU 核可以与通用处理器结合，也可以是完全自行开发设计的专有电路。最终的系统一般是一个或多个专用的集成电路芯片。

由于这个类型系统将 CPU 核与可重构逻辑资源集成在同一个芯片之中，CPU 的一些参数，如地址和数据总线宽度，可以根据需要进行必要调整来实现二者之间更快的数据交换。前面两个系统中 CPU 管脚的位置对系统速度的影响在这个系统中也变得很小，可以实现更快的速度。

这类系统的缺点是设计比较困难。前面两类松耦合结构的系统均可以直接选用现成芯片，而这类系统需要设计者按照全定制的方式设计专用集成电路，难度较大。目前处理器与可重构逻辑紧耦合的结构是可重构计算器件部分的研究热点，除了研究人员以外，业界也有大公司注意到了这个趋势。目前传统意义上的可编程器件公司和逻辑电路公司均对此展开研究，有的公司已经推出系列产品。一类是如 XILINX 等 FPGA 公司，他们在自己的产品中计划增加内嵌式的微处理器核;另一类是如 LSI 等逻辑电路供应商，他们则在自己的处理器产品上提供了可编程的逻辑单元以满足电路的灵活性的需要。这两类产品均属于处理器与可重构逻辑紧耦合的分类。

2.3.2.4 处理器、存储器与可重构逻辑紧耦合

在 CPU 和 FPGA 可重构逻辑之间实现高速通讯后，存储子系统成为系统的瓶颈。在现代计算环境中，CPU 和存储器之间速度差别越来越大，而处理器与可重构逻辑紧耦合结构中的 CPU 仍然受到与存储器交换数据带宽以及速度问题。在现代先进微处理器设计中，增加片上高速缓存可以明显提高 CPU 性能。当 CPU、存储器和可重构逻辑集成在一起后，三者之间分别以非常高的带宽传递数据。所以第 4 类结构中将 CPU、存储器和可编程逻辑集成在同一个芯片上，称为单芯片可重构计算系统。其结构如图 2.8 所示:

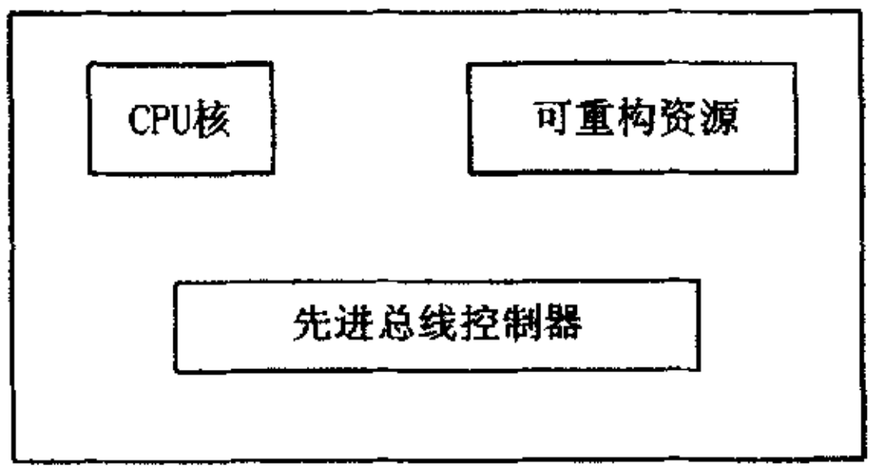


图 2.8 处理器、存储器与可重构逻辑紧耦合

这个类型可重构系统依然保持了上一节所述系统的 CPU 和 FPGA 紧耦合结构，所以也保持了它所带来的高速灵活接口的优点。同时，片上存储系统和 CPU 与 FPGA 接口能力则进一步提高的系统的性能。运算数据可以直接与 CPU 和 FPGA 高速交换，而在其它几种结构，系统中数据往往通过 CPU 的寄存器后才送往可编程单元。受到目前工艺水平的限制，嵌入式存储器的容量还不能做得足够大，所以这里的存储器需要引入多级存储的概念，即片上存储器的作用类似于现代高性能处理器中的片上 Cache，在芯片外部还有大容量存储器模块，片上存储器则可以设计成各种宽度和深度来满足系统的需要。

每种类型都具有不同的优缺点。可重构硬件集成度越高，其通信开销就越低，

在应用中的使用频率就越高。然而，在没有主机处理器干预的时间段里可重构硬件不能高效地运行，而且可用的可重构逻辑的数量通常非常有限。越松的耦合类型，就会越多的考虑程序执行的并行性，但是它的通信代价也就越高。在需要大量通信的应用中，这将降低或抵消通过这种类型的可重构硬件获取的加速效益。

2.3.3 可重构 DSP 计算引擎系统结构

由于成本、系统功耗和面市时间等原因，许多通讯、视频和图像系统已无法简单地用现有的单片 DSP 处理器来实现。实时视频处理对系统性能的要求极高，因此几乎所有只具最简单功能的通用 DSP 都不具备这项功能。而且，基于 DSP 的解决方案通常需要在单板上嵌入许多 DSP，以得到必需的处理能力，这无疑将增加程序资源开销和数据存储器资源开销。然而，可编程逻辑器件允许设计人员利用并行处理技术实现视频信号处理算法，并且只需单个器件就能实现期望的性能。现场可编程门阵列(FPGA)尤其适合于乘法和累加(MAC)等重复性的 DSP 任务。

从音频/视频及图像处理应用的角度出发，本文将设计一个可用于此类应用的前端数据计算的可重构 DSP 计算引擎的系统结构，其基本结构框图如图 2.9。虚线框部分即为该系统的主要部分，即可重构 DSP 计算单元的抽象结构。

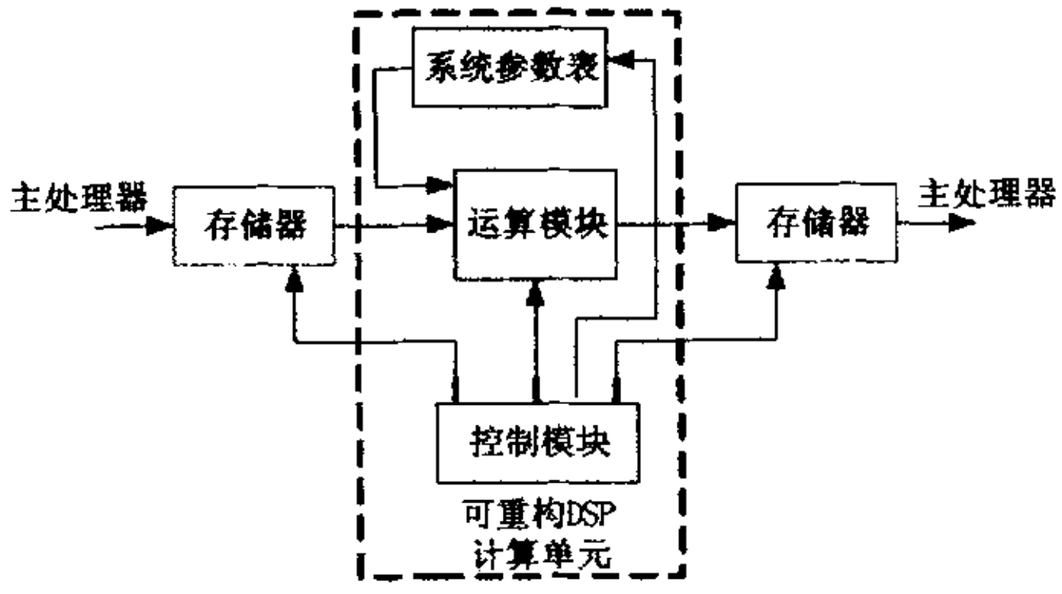


图 2.9 可重构DSP计算引擎基本框图

从上图我们可以看出该计算引擎所需的系统参数是由主处理器计算后传到计算引擎中的，经由计算引擎计算出的结果值再传回到主处理器。其中整个计算单元主要由可编程计算模块阵列和以互联网络为主体的控制模块组成，其功能的设计实现是基于 FPGA 的。

文中的设计将从 DSP 相关功能及算法的研究出发，设计出一个可以实现多种 DSP 运算的并行处理系统结构，作为音频/视频及图像处理应用中的前端数据处理的计算引擎，充分利用可编程功能的并行及流水线机制，保证系统只需要改变系统参数即

可进行不同的 DSP 计算，且达到 ASIC 设计的计算速度。

2.4 本章小结

本章首先给出了 DSP 处理器的几种体系结构，然后给出可重构计算定义，在此基础上，以可重构硬件中的典型代表 FPGA 为例，从多个角度分析了可重构计算硬件结构。根据硬件中的基本计算模块的大小和复杂性，大体上可分细粒度、粗粒度和中等粒度 FPGA，大量可重构系统所使用的逻辑模块粒度是我们所分类为中等粒度的系统。给出了几种可重构 DSP 系统结构，并对各种结构的优劣性进行分析研究。最后，给出本文将要设计的可重构 DSP 计算引擎系统结构的抽象结构框图。

本章对 DSP 体系结构、可重构计算及其硬件的剖析，以及可重构 DSP 系统的分析为后面 DSP 技术的可重构研究做好了充分的准备。

第三章　基于可重构计算的 FIR/IIR/DT 功能

从以上对 FPGA 器件结构分析可知，可重构计算对于 DSP 算法的实现是有一定范围的，比如自身结构比较复杂的 DSP 算法更适合于 DSP 处理器实现。本章将分析一下究竟什么样的算法与 DSP 实现相比更适合于 FPGA 实现，即更适合用可重构计算的方式来实现的 DSP 算法具有的特征。

首先，可重构计算之所以有较高的性能，是通过并行处理和重构获得的。而重构特性和通用处理器的编程性的差别是显而易见的，所以从本质上来说，可重构计算应该属于 VLSI 并行计算机系统。从指令流与数据流的多倍性的角度，FLYNN 在 1996 年将计算机分为以下四类[12]：

- ✧ 单指令流单数据流(SISD)
- ✧ 单指令流多数据流(SIMD)
- ✧ 多指令流单数据流(MISD)
- ✧ 多指令流多数据流(MIMD)

VLSI 并行结构大多是属于单指令流多数据流(SIMD)的结构，这种结构对多个重复的 PE 由单一指令部件，按照同一指令流的要求同时向它们分配各自需要的不同数据。这种结构改变了过去计算机结构上以 CPU 为中心或以主存为中心的典型冯.诺依曼结构，采用以互连网络为中心的结构。

但 VLSI 并行计算机大都是属于专用计算机范畴，它们面向具体问题，对算法有较强的依赖性。互连结构是固定的，而可重构计算系统也是面向具体问题，对算法有一定的依赖性，但是它的互连结构是可变的，这正是可重构计算系统与专用计算机的最大区别。一般来说，专用集成电路完成的算法也能用 FPGA 器件实现。专用集成电路是针对具体算法把电路进行了最优化设计的，因而往往性能也是最高的。而 FPGA 器件是由单元阵列和互连结构两部分组成的，且这两部分都要通过配置开关来完成各种功能，因而必然降低了芯片的有用面积；同时对电路也无法进行最优化。但正是由于有了这些可以进行逻辑重构的单元阵列和互联结构，才使得 FPGA 具有极大的灵活性，能够实现更多的功能。

下面将对几类 DSP 算法进行研究，找出在 FPGA 上实现较 DSP 实现效果更佳的 DSP 算法，并为这些算法寻求一种于同一系统中实现的有效方法。

3.1 可重构计算在 DSP 算法中的适应范围

信号处理算法有很多，但大多数信号处理系统的设计都具有许多共同的特征，原因很明显，这些设计中所用到的基本算法正是基本的 DSP 算法。DSP 算法大体上可以归结为以下几类[13]：

- ✧ 卷积
- ✧ 信号滤波
- ✧ 信号变换

当然，这种分类并没有非常严格的标准，比如线性时不变系统中对信号滤波所用到的基本运算即为卷积运算。因此，虽然大体上将 DSP 算法分成了这样几类，而事实上类与类之间还是有着比较多的联系，并非一种严格意义上的分类。

3.1.1 卷积及滤波

3.1.1.1 卷积

卷积[14]是数字信号处理常用的基本运算。从数学上讲，一维卷积问题可定义如下：

给定权系数$\{w_1,w_2,\ldots,w_k\}$、输入序列$\{x_1,x_2,\ldots x_n\}$、计算序列$\{y_1,y_2,\cdots y_{n-k+1}\}$，其计算公式如下：

$$y_i=\sum_{j=1}^{k}w_jx_{i+j-1}=w_1x_i+w_2x_{i+1}+w_kx_{i+k-1} \qquad 3\text{-}(1)$$

当卷积用于数字滤波时，$\{x_i\}$表示将被处理的输入信号序列，而$\{w_i\}$表示数字滤波处理的冲激响应函数。根据$\{w_i\}$的不同，卷积处理可以达到不同的滤波效果，如图像增强、信号平滑等。一般地，输出序列$\{y_i\}$表示已被处理的信号。

从计算结构上看，它可以看成特殊形式的矩阵相乘的运算，写成矩阵形式如式3-(2)：

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ y_{n-k+1} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdot & x_k \\ x_2 & x_3 & \cdot & x_{k+1} \\ \cdot & \cdot & \cdot & \cdot \\ x_{n-k+1} & x_{n-k+2} & \cdot & x_n \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ w_k \end{bmatrix} \qquad 3\text{-}(2)$$

3.1.1.2 传统滤波

数字滤波[15]是数字信号处理中经常使用的一种计算，它在图像处理中也有很重要的作用。下面的数学表达式 3-(3)描述了数字滤波的过程

$$y(i) = \sum_{k=0}^{k} b(k)x(i-k) - \sum_{m=1}^{m} a(m)y(i-m) \qquad 3\text{-}(3)$$

很显然该过程需要大量的加法（Σ ）、减法和乘法$(a(m)y(i-m))$运算。$b(k)$和$a(m)$分别是存储器中的两个滤波器系数表，它们分别与过去的输入$x(i-k)$和输出$y(i-m)$样值表相乘。每当一个新的样值到来，将其放入输入样值表的上端，并删掉样值表中最老的样值。这样，对任意滤波器的设计，两个样值表总能保持固定长度。这个过程表明，实现这一算法的硬件应该能够高效访问和管理其存储器的能力，同时能够处理数据端口的输入和输出数据流。

最为普通的数字滤波器就是线性时间不变量（linear time-invariant,LTI）滤波器。LTI 数字滤波器通常分成有限冲激响应（finite impulse response,即 FIR）和无限冲激响应（infinite impulse response,即 IIR）两大类。顾名思义，FIR 滤波器由有限个采样值组成，将上述卷积的数量降低到在每个采样时刻为有限个。而 IIR 滤波器需要执行无限多次卷积。

这里以传统的有限冲激响应滤波器(FIR)为例来分析数字滤波器实现中的卷积过程。有限冲激响应(FIR)滤波可定义如下：

给定权系数$\{w_1, w_2, \ldots, w_k\}$，输入序列$\{x_1, x_2, \ldots, x_n\}$，计算序列$\{y_1, y_2, \cdots y_{n-k+1}\}$，使得：

$$y_i = \sum_{j=1}^{k} w_j x_{i+j-1} = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1} \qquad 3\text{-}(4)$$

3.1.2 FPGA 实现卷积及滤波算法的优势

对照 3-(4)式和 3-(1)式，可以看出，二者的基本运算完全一样。对以上公式进行分析可以发现，相对于输入序列而言，权系数是不变的，即每个 x 的值对应一个固定的权系数 w。若用通用 DSP 处理器来实现以上算法，则权系数必然被当作变量来处理，因而芯片的利用率不高，导致处理器的性能下降。而这种算法正好适于用可重构计算来实现，因为在可重构器件中实现时可以将这些权系数作为常数来使用，可利用分布式算术等方法来实现，以充分发挥可重构器件的优势。

IIR 滤波器与 FIR 滤波器的根本区别在于前者需要执行无限次卷积，而后者的卷积次数是有限的，但他们所用到的基本运算是相同的，即卷积运算；同样，对于 QMF 滤波器组，它的实现是建立在 FIR 和 IIR 滤波的基础之上的，因此其中所用基本运算仍然是卷积运算。故由以上分析可知，这些常见的基于卷积运算实现的滤波算法均适于用可重构计算技术来实现。而且其中最常见的 FIR 和 IIR 滤波器已经被以不同的方式在 FPGA 中实现[11]，取得了较 DSP 处理器实现更好的效果。然而，这种传统的滤波方式无论是用 DSP 还是 FPGA 实现时都是充分利用乘法器和累加器，用乘累加(MAC)方式实现算法，需要占用较大的硬件面积，并且提高算法的实现效率主要靠硬件性能的改善。而且，用这种 MAC 方式将上述的各种滤波算法统一于同一种结构中实现，并取得 DSP 功能的重构却是困难的。因此，要做到这一点，还需要寻求其他的滤波实现方式。

3.2 移位加方式实现滤波算法

上一节中分析得出了更适合于 FPGA 实现的 DSP 算法的特征，且已有实例证明基于 FPGA 实现某些 DSP 算法要比基于 DSP 处理器实现的效果更佳些。然而，这种传统的 MAC 方式实现的效率对硬件性能的依赖较大，且难于将这一类算法统一于一个可重构系统中来实现。基于这些方面的考虑，本小节将使用一种更合适的方式来完成 DSP 算法的 FPGA 实现。

在数字信号处理中，格型(Lattice)网络[15]起着重要的作用。它除了对有限寄存器长度效应敏感度低外，在功率谱估计、语音处理、自适应滤波、线性预测和逆滤波等方面已得到广泛应用。这里主要来分析一下全零点(FIR)格型滤波和全极点(IIR)格型滤波。

3.2.1　FIR 格型滤波

给定一个 M 阶 FIR 滤波器的系统函数 $H(z)$ 可写成如下形式：

$$H(z)=B(z)=\sum_{i=0}^{M}b_i z^{-i}=1+\sum_{i=1}^{M}b_M^{(i)}z^{-1} \qquad 3-(5)$$

其中 $b_M^{(i)}$ 表示 M 阶 FIR 滤波的第 i 个系数，并假设首项系数 $b_0=1$。$H(z)$ 对应的格型结构如图 3.1 所示：

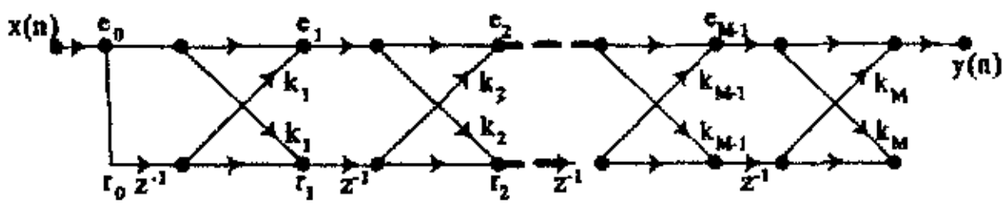


图 3.1 FIR 格型滤波结构

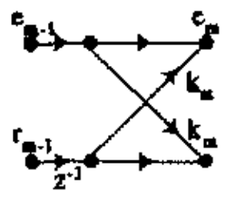


图 3.2 FIR 格弄滤波单元结构

该结构图可看成由 M 个如图 3.2 所示的格型网络单元级联而成。每个格形单元有两个输入端和两个输出端，输入信号 $x(n)$ 同时送到第一级格形单元的两个输入端，而在输出端仅取最后一级格形单元上面的一个输出端作为整个格型滤波器的输出信号 $y(n)$。

下面推导由 $H(z)=B(z)$ 的系数 $\{b_i\}$ 求出格型结构网络系数 $\{k_i\}$ 的递推公式。图 3.2 所示基本格型单元的输入、输出关系如下式：

$$e_m(n)=e_{m-1}(n)+r_{m-1}(n-1)k_m \qquad 3-(6)$$

$$r_m(n)=e_{m-1}(n)\cdot k_m+r_{m-1}(n-1) \qquad 3-(7)$$

且有

$$e_0(n)=r_0(n)=x(n) \qquad 3-(8)$$

$$y(n)=e_M(n) \qquad 3-(9)$$

其中，$e_m(n)$、$r_m(n)$分别为第m个基本单元的上、下端的输出序列，$e_{m-1}(n)$、$r_{m-1}(n)$分别为该单元上、下端的输入序列。

设$B_m(z)$、$J_m(z)$分别表示由输入端$x(n)$至第m个基本单元上、下输出端$e_m(n)$、$r_m(n)$对应的系统函数，即

$$B_m(z)=E_m(z)/E_0(z)=1+\sum_{i=1}^{m}b_m^{(i)}z^{-i},m=1,2,\ldots,M \qquad 3-(10)$$

$$J_m(z)=R_m(z)/R_0(z),m=1,2,\ldots,M \qquad 3-(11)$$

当$m=M$时，$B_m(z)=B(z)$。对3-(6)和3-(7)式两边进行Z变换得

$$E_m(z)=E_{m-1}(z)+k_mz^{-1}R_{m-1}(z) \qquad 3-(12)$$

$$R_m(z)=k_mE_{m-1}(z)+z^{-1}R_{m-1}(z) \qquad 3-(13)$$

对3-(11)和3-(12)式分别除以$E_0(z)$和$R_0(z)$，再由3-(8)和3-(9)式得

$$\begin{bmatrix}B_m(z)\\J_m(z)\end{bmatrix}=\begin{bmatrix}1 & k_mz^{-1}\\k_m & z^{-1}\end{bmatrix}\times\begin{bmatrix}B_{m-1}(z)\\J_{m-1}(z)\end{bmatrix} \qquad 3-(14)$$

$$\begin{bmatrix}B_{m-1}(z)\\J_{m-1}(z)\end{bmatrix}=\frac{\begin{bmatrix}1 & -k_m\\-k_mz & z\end{bmatrix}\begin{bmatrix}B_m(z)\\J_m(z)\end{bmatrix}}{1-k_m^2} \qquad 3-(15)$$

上面两式给出了格型结构中由低阶到高阶（或由高阶到低阶）系统函数的递推关系，但这种关系中同时包含$B(z)$和$J(z)$。实际中只给出$B_m(z)$，所以应找出$B_m(z)$与$B_{m-1}(z)$之间的递推关系。

由3-(11)和3-(12)式有$B_0(z)=J_0(z)=1$，所以$B_1(z)=B_0(z)+k_1z^{-1}J_0(z)=1+k_1z^{-1}$，$J_1(z)=k_1B_0(z)+z^{-1}J_0(z)=k_1+z^{-1}$，即$J_1(z)=z^{-1}B_1(z^{-1})$。

令$m=2,3,\ldots,M$，可推出

$$J_m(z)=z^{-m}B_m(z^{-1}) \qquad 3-(16)$$

将上式分别代入 3-(13)和 3-(15)式得

$$B_m(z) = B_{m-1}(z) + k_m z^{-m} B_{m-1}(z^{-1}) \qquad 3\text{-}(17)$$

$$B_{m-1}(z) = \frac{B_m(z) - k_m z^{-m} B_m(z^{-1})}{1 - k_m^2} \qquad 3\text{-}(18)$$

下面导出 k_m 与滤波器系数 b_m^m 之递推关系。将 3-(10)代入 3-(17)及 3-(18)式，利用待定系数法可得到如下两组递推关系：

$$\begin{cases} b_m^{(m)} = k_m \\ b_m^{(i)} = b_{m-1}^{(i)} + k_m b_{m-1}^{(m-i)} \end{cases} \qquad 3\text{-}(19)$$

$$k_m = b_m^{(m)}$$

$$b_{m-1}^{(i)} = \frac{b_m^{(i)} - k_m b_m^{(m-i)}}{1 - k_m^2} \qquad 3\text{-}(20)$$

上面两式中，$i = 1,2,\ldots,(m-1); m = 1,2,\ldots,M$ 。

这里 k_m，$m = 1,2,\ldots,M$ 称为反射系数或者 PARCOR 系数。计算出所有 k_m 的值后，FIR 滤波器的格形结构即可如下描述：

$$\begin{bmatrix} e_{m-1} \\ r_{m-1} \end{bmatrix} = \begin{bmatrix} 1 & -k_m \\ -k_m & 1 \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} \qquad 3\text{-}(21)$$

3.2.2　IIR 格型滤波

设一个全极点系统函数由下式给定：

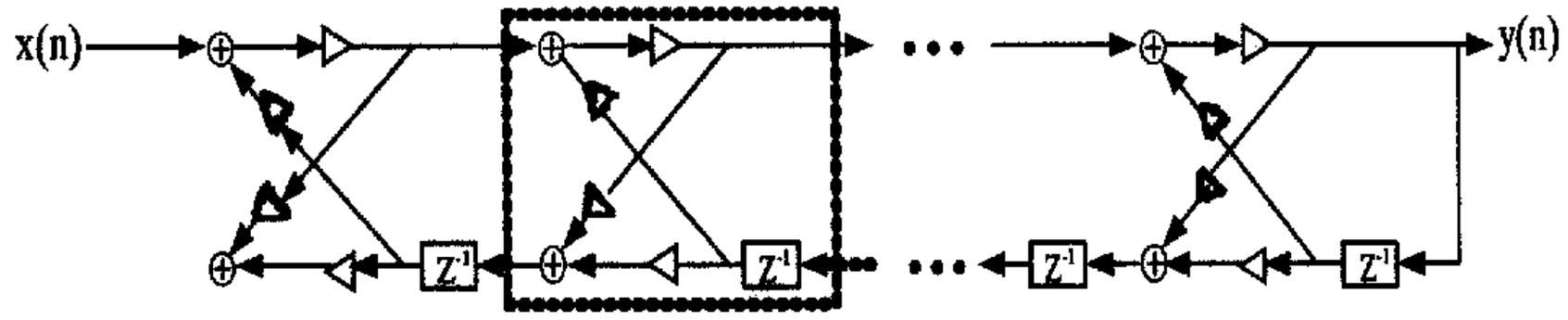
$$H(z) = \frac{1}{1 + \sum_{i=1}^{M} a_M^{(i)} z^{-i}} = \frac{1}{A(z)} \qquad 3\text{-}(22)$$

与 3-(6)式比较可知，$H(z) = \frac{1}{A_M(z)}$ 是 FIR 系统 $B_M(z) = A_M(z)$ 的逆系统。所以按照系统求逆准则得到 $H(z) = \frac{1}{A_M(z)}$ 的格型结构如图 3.4 中的 a 所示。

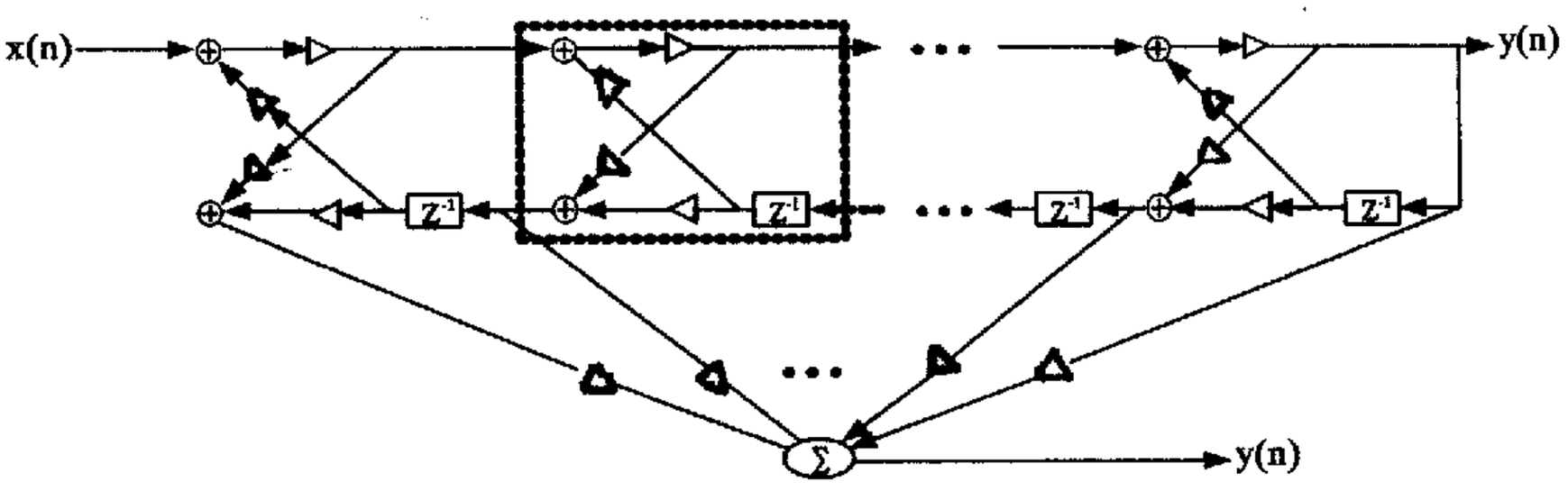
一般的 IIR 滤波的系统函数：

$$H(z)=\frac{\sum_{r=0}^{M} b_r z^{-r}}{1+\sum_{k=1}^{N} a_k z^{-k}} \qquad 3-(23)$$

其格形结构如图 3.4 中的 b 所示。



a 全极点IIR格形结构



b 一般的IIR格形结构

图 3.3 IIR格形结构

系统求逆步骤如下：

- 将输入至输出的无延时通路全部反向，并将该通路的常数值支路增益变成原常数值的倒数(此处为 1)；
- 将指向这条新通路各节点的其它支路增益乘以-1；
- 将输入与输出交换位置。

由此可见，尽管 IIR 滤波器中的基本格形结构单元和 FIR 格形结构单元很相似，但 IIR 中的数据流向和 FIR 是相反的。因此，要将二者整合于一个统一的系统中还是存在一些困难，需要寻求别的思路。下一节中将通过同样具有卷积基本运算的一类算法---离散变换（DT）来探讨这类适于可重构计算技术实现的 DSP 算法如何统一于同一可重构系统中。

3.3　DFT/DHT/DCT 离散变换及其统一结构设计

DSP 算法中常见的变换有 DFT、DHT、DCT 等，首先来看一下几个离散变换的定义。

3.3.1　DFT 变换

离散傅里叶变换（DFT）是一种把有限个时域数据变换为频域数据的方法。其反变换称为离散傅里叶反变换，即 IDFT。DFT 定义如下：

当具有 N 个序列值的离散时间信号 $X_n(n=0,1,\ldots,N-1)$ 给定时，下式变换

$$X_k=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x_n\exp(-j\frac{2\pi nk}{N}),k=0,1\ldots,N-1 \qquad 3-(24)$$

就称为 x_n 的离散傅叶变换。它的反变换式为

$$x_n=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}X_k\exp(j\frac{2\pi nk}{N}),n=0,1,\ldots,N-1 \qquad 3-(25)$$

则上两式为一对离散傅里叶变换对。

3.3.2　DHT 变换

对实序列 $x(n)$，其 N 点 DFT 具有简单的共轭对称性

$$X(N-k)=X^*(k),k=0,1,\ldots,N-1 \qquad 3-(26)$$

所以，只要计算出 $X(k)$ 的前 $N/2$ 个值，则后 $N/2$ 个值可用 3-(27) 式求得，$X(k)$ 的 $N/2$ 个复数数据正好对应 N 个实数数据。由此可见，一个实序列的 N 点 DFT 完全可由 N 个实数数据确定。离散哈特莱变换就是直接对实序列进行实数域变换，记为 DHT。

离散哈特莱变换定义如下[16]：

设 $x(n),n=0,1,\ldots,N-1$，为一实序列，其 DHT 定义为

$$X_H(k)=DHT[x(n)]=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)cas\left(\frac{2\pi}{N}kn\right),k=0,1,\ldots,N-1 \qquad 3-(27)$$

式中，$cas(\alpha)=\cos\alpha+\sin\alpha$。其逆变换（IDHT）为

$$x(n)=IDHT\left[X_H(k)\right]=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}X_H(k)cas\left(\frac{2\pi}{N}kn\right),n=0,1,\ldots,N-1 \qquad 3-(28)$$

DHT 中将 $x(n)$ 看成实数数据，而不像 DFT 中将 $x(n)$ 看成虚部为零的复数数据进行运算。所以，相对一般的 DFT，DHT 可节省一半的存贮空间，运算效率提高近一倍，且 DHT 与 DFT 之间存在简单的关系，容易实现两种谱之间的相互转换。

为了便于比较，将 DFT 表示重写如下：

$$X(k)=DFT\left[x(n)\right]=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)e^{-j\frac{2\pi}{n}kn}=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)\left[\cos\left(\frac{2\pi}{N}kn\right)-j\sin\left(\frac{2\pi}{N}kn\right)\right] \qquad 3-(29)$$

$$x(n)=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}X(k)e^{j\frac{2\pi}{N}kn}=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}X(k)\left[\cos\left(\frac{2\pi}{N}kn\right)+j\sin\left(\frac{2\pi}{N}kn\right)\right] \qquad 3-(30)$$

容易看出，DHT 的核函数 $cas\left(\frac{2\pi}{N}kn\right)=\cos\left(\frac{2\pi}{N}kn\right)+\sin\left(\frac{2\pi}{N}kn\right)$ 是 DFT 的核函数 $e^{j\frac{2\pi}{N}kn}=\cos\left(\frac{2\pi}{N}kn\right)+j\sin\left(\frac{2\pi}{N}kn\right)$ 的实部与虚部之和。

将 $X_H(k)$ 分解为奇对称分量 $X_{H_0}(k)$ 与偶对称分量 $X_{H_e}(k)$ 之和

$$X_H(k)=X_{H_e}(k)+X_{H_0}(k) \qquad 3-(31)$$

其中

$$X_{H_e}(k)=\frac{1}{2}\left[X_H(k)+X_H(N-k)\right] \qquad 3-(32)$$

$$X_{H_0}(k)=\frac{1}{2}\left[X_H(k)-X_H(N-k)\right] \qquad 3-(33)$$

由 DHT 定义有

$$X_{H_e}(k)=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)\cos\left(\frac{2\pi}{N}kn\right) \qquad 3-(34)$$

$$X_{H_0}(k)=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)\sin\left(\frac{2\pi}{N}kn\right) \qquad 3-(35)$$

所以，$x(n)$ 的 DFT 可表示为

$$X(k)=X_{H_e}(k)-jX_{H_0}(k) \qquad 3-(36)$$

同理，$x(n)$ 的 DHT 可表示为

$$X_H(k)=\mathrm{Re}[X(k)]-\mathrm{Im}[X(k)] \qquad 3-(37)$$

因此，已知 $x(n)$ 的 DHT，则 DFT 可用下式求得：

$$X(k)=\frac{1}{2}[X_H(k)+X_H(N-k)]-\frac{1}{2}j[X_H(k)-X_H(N-k)] \qquad 3-(38)$$

如果不考虑因子 1/2，只要增加 $2N$ 次实数加法运算就能由 $x(n)$ 的 DHT 谱求出 $x(n)$ 的 DFT 谱。

3.3.3　DCT 变换

DCT[16]在压缩中用得极多，因为转换后的数据很容易就可以找到其中的冗余并除去。因此，很多像 JPEG 和 MPEG 的标准压缩技术大量使用了 DCT 和它的反变换 IDCT。实质上，DCT 是由离散傅里叶变换，即 DFT 的实部组成的，也就是说只用到了 DFT 中的余弦部分。DCT 也可以用优化的算法来实现，叫快速 DCT。提高 DCT 运算速度的技术和快速傅里叶变换提高 DFT 速度的技术非常相似。DCT 的基本定义如下：

$$X_{cosine}(m)=\sum_{n=0}^{N-1}x(n)\cos(2\pi nm/N),m=0,1,\ldots,N-1 \qquad 3-(39)$$

DFT 的定义中，离散时间和频率分别用 n 和 m 表示，DCT 序列的长度定义为 N。在这种情况下，只有当输入信号为实数时，输出才会是实数。

3.3.4　统一的离散变换设计

分析以上各种变换的定义式可以发现，上述变换可以统一为一个表达式 3-(40)，只要给表达式中的参数赋以不同的值，表达式即可表示相应的变换或者成为相应变

换的定义式。

$$\begin{cases} X_C(k) = \alpha \sum_{n=0}^{L-1} \cos\left[(2n+1)\omega_k + \theta_k\right] x(n) \\ X_S(k) = \alpha \sum_{n=0}^{L-1} \sin\left[(2n+1)\omega_k + \theta_k\right] x(n) \end{cases} \qquad 3\text{-}(40)$$

其中 $k = 0,1,\ldots,N-1$。

a) 对于 DFT：

$$L = N, \alpha = \frac{1}{\sqrt{N}}, \omega_k = -\frac{k\pi}{n}, \theta_k = -\omega_k$$

则 DFT 可以表示为：

$$R_e = \left\{X_{DFT}(k\partial)\right\} = X_C(k), I_m\left\{X_{DFT}(k)\right\} = X_S(k)$$

b) 对于 DHT：

$$L = N, \alpha = \frac{1}{\sqrt{N}}, \omega_k = -\frac{k\pi}{N}, \theta_k = -\omega_k$$

则 DHT 可以表示为：

$$X_{DHT}(k) = X_C(k) + X_S(k) \qquad 3\text{-}(41)$$

c) 对于 DCT：

$L = N, \alpha = c_k, \omega_k = \frac{k\pi}{2N}, \theta_k = 0$，其中 c_k 是 DCT 变换中的尺度因子。

则 DCT 可以表示为：

$$X_{DCT}(k) = X_C(k)$$

3.4 本章小结

本章主要通过对 DSP 算法的研究找出了一种适于可重构计算技术实现的 DSP 算法特征，并针对将此类算法统一于同一可重构系统中这一问题进行讨论。虽然已经有多个利用 FPGA 实现这类算法中的某一个算法的例子，但这些例子的实现都是采用传统的 MAC 方式，然而利用这种传统方式来解决这一问题显然存在很大的困难，必

须引入更有效的方法。

从对 DT 算法的分析及统一的 DT 结构的设计情况来看，有必将三角函数引入到这一问题中进行讨论。因此，在下一章的 FIR/IIR/DT 统一可编程模块设计中引入了专于三角函数计算的 CORDIC 理论。

第四章 FIR/IIR/DT 功能的统一可编程模块设计

根据上一章对适于可重构计算技术实现的 DSP 算法及相关问题的讨论，在本章的统一可编程模块设计中引入了 CORDIC(Coordination Rotation Digital Computer)理论，最终设计出一个可用于可重构计算系统的统一可编程模块。

4.1 CORDIC 理论

CORDIC 算法[17](The Coordinate Rotational Computer)是 Volfer 等人于 1959 年在美国航空控制系统的设计中提出来的，它是一种用于计算一些常用的基本运算和算术操作的循环迭代算法，其基本思想是用一系列与运算基数相关的角度的不断偏摆从而逼近所需旋转的角度。从广义上讲它是一个数值性计算逼近的方法，由于这些固定的角度与计算基数有关，运算只有移位和加减。可用该算法来计算的函数包括乘、除、平方根、正弦、余弦、反正切、向量旋转(即复数乘法)以及指数运算等。1971 年，J. S. Walther 提出了统一的 CORDIC 算法形式，把圆周旋转、双曲旋转和直线旋转统一到同一个 CORDIC 迭代方程里，为同一硬件实现多功能提供了前提。

在传统的硬件算法设计中，乘、除等基本数学函数运算是一种既耗时又占用面积大的运算，甚至有时是难以实现的，CORDIC 算法正是为解决这种问题而产生的。它从算法本身入手，将其分解成为一些简单的且在硬件中容易实现的基本算法，如加法、移位等，因此使得这些算法在硬件上可以得到较好的实现。又由于该算法是一种规则化的算法，它满足了硬件对算法的模块化、规则化的要求，因此 CORDIC 算法可以充分发挥硬件的优势，利用硬件的资源，从而实现硬件与算法相结合的一种优化方案。正是由于上述原因以及前面所提到的各个算法的特点，我们才将 CORDIC 算法引入到统一的系统结构设计当中来。

4.1.1 CORDIC 基本理论概要

下面简单介绍一下 CORDIC 算法理论。

设输入为 x_{in}、y_{in}、z_{in}，参数为 m，则 CORDIC 算法如式 4-(1)所示：

$$\begin{aligned} x_{i+1} &= x_i + m\delta_i \alpha_i y_i \\ y_{i+1} &= y_i + m\delta_i \varepsilon_i x_i \\ z_{i+1} &= z_i + \delta_i \theta_i \end{aligned} \qquad 4\text{-}(1)$$

式 4-(1)中　　当 $m=-1$ 时，$\theta_i = \tanh^{-1}(\alpha_i)$；

当 $m=0$ 时，$\theta_i = \delta_i$；

当 $m=1$ 时，$\theta_i = \tan^{-1}(\alpha_i)$。

$x_1 = x_{in}$；　$y_1 = y_{in}$；　$z_1 = z_{in}$

其中 i 表示循环的次数，其范围是 $1...N$；

δ_i 可取 1 或－1，其值由 x_i、y_i、z_i 的符号决定；

α_i 取值为 2^{-i}；

m 为参数，可取 0、－1、＋1。当 m 取 0 时，最后得到的函数称为线性函数；当 m 取 1 时，得到的函数为平方函数；当 m 取－1 时，得到的函数为立方函数。这 3 种函数的具体形式在文献[6]中已经给出。

4.1.2 CORDIC 的旋转和矢量运算

4.1.2.1 旋转运算

这里以在三角范围里的旋转操作为例，说明 CORDIC 的基本思想。

设 $M(x_0,y_0)$ 是平面直角坐标中的一点，向量 $\overrightarrow{OM}$ 与 X 轴的夹角 Φ，$\overrightarrow{OM}=R_0$，求 $\overrightarrow{OM}$ 旋转 θ 角后，$M(x,y)$ 的坐标。

把 θ 角分成若干个旋转角之和，设每次旋转角度为 θ_i。适当选取 θ_i，使 $\left|\theta - \sum_{i=1}^{n}\theta_i\right| \to 0$。

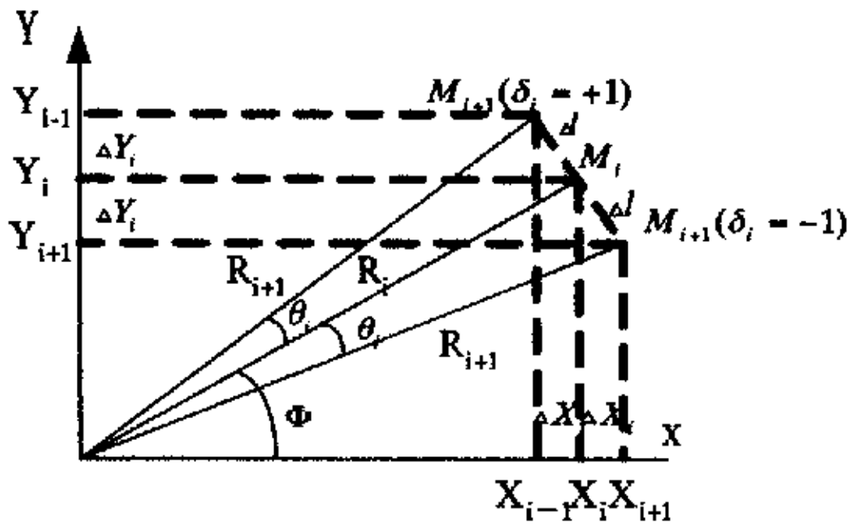


图 4.1 第 i 旋转步骤

图 4.1 表示了第 i 次旋转步骤的坐标图，为简化运算，假设 $M_{i+1}M_i \perp OM_i$ 对每次旋转有

$$\begin{aligned} y_{i+1} &= R_{i+1} \cdot \sin(\Phi_i + \theta_i) \\ &= y_i \pm \Delta y_i \end{aligned} \qquad 4-(2)$$

$$\begin{aligned} x_{i+1} &= R_{i+1} \cdot \cos(\Phi_i \pm \theta_i) \\ &= x_i \mp \Delta x_i \end{aligned} \qquad 4-(3)$$

其中 $R_{i+1} = \sqrt{R_i^2 + (\Delta l_i)^2}$， $\Delta l_i = R_i \tan\theta_i$。则有

$$y_{i+1} = \sqrt{1 + \tan^2\theta_i} R_i \sin(\Phi \pm \theta_i) \qquad 4-(4)$$

$$x_{i+1} = \sqrt{1 + \tan^2\theta_i} R_i \cos(\Phi \pm \theta_i) \qquad 4-(5)$$

如何选取旋转的步长和方向？观察下列角度序列：

$$\begin{aligned} &\theta_1 = 90^\circ = \tan^{-1}\infty \\ &\theta_2 = 45^\circ = \tan^{-1}2^0 \\ &\theta_3 = 26.5^\circ = \tan^{-1}2^{-1} \\ &\qquad \cdots\cdots \\ &\theta_i = \tan^{-1}2^{-(i-2)} \end{aligned} \qquad 4-(6)$$

第一次旋转 $\theta_1 = 90^\circ$，第二次旋转 $\theta_2 = 45^\circ \cdots\cdots$，第 i 次转角约为前一次转角的 1/2，

当 $\theta - \sum_{i=1}^{n}\theta_i < 0$ 时，第 $i+1$ 次旋转取负值，使 $\left|\sum_{i=1}^{n}\theta_i\right| = \theta$。由 4-(6) 式得：

$$\begin{aligned}\Delta l_i &= 2^{-(i-2)} R_i \\ \Delta y_i &= \frac{2^{-(i-2)}}{\sqrt{1+2^{-2(i-2)}}} R_i \\ &= 2^{-(i-2)} x_i\end{aligned} \qquad 4-(7)$$

将 4-(7)式带入 4-(2)和 4-(3)式得

$$y_{i+1} = y_i \pm 2^{-(i-2)} x_i \qquad 4-(8)$$

$$x_{i+1} = x_i + 2^{-(i-2)} y_i \qquad 4-(9)$$

角度 $$Z_{i+1} = Z_i - \theta_i \qquad 4-(10)$$

设 δ_i 表示旋转方向，逆时针为正，则 $\delta_{i+1} = \begin{cases} +1 & Z_i \geq 0 \\ -1 & Z_i \leq 0 \end{cases}$，那么 4-(8)—4-(10)式可以完整地表达成

$$y_{i+1} = y_i + \delta_i 2^{-i} x_i \qquad 4-(11)$$

$$x_{i+1} = x_i - \delta_i 2^{-i} y_i \qquad 4-(12)$$

$$z_{i+1} = z - \delta_i \tan^{-1} 2^{-(i-2)} \qquad 4-(13)$$

x 、 y 、 z 均为二进制数表示， x_{i+1} 、 y_{i+1} 的求解，便简化成移位和加法运算了。

然而，以上推导中尚存在两个问题：

第一、 $\sum_{i=1}^{n} \theta_i$ 是否收敛？即 $\theta - \sum_{i=1}^{n} \theta_i \to 0$？答案是肯定的。Volder 的文章中做过证明，见文献[18]

第二、4-(11)—4-(13)式的推导是在 $M_{i+1}M_i \perp OM_i$ 前题下进行的，求增量 Δy_i（或 Δx_i）才转化成对 x_i（或 y_i）的（$i-2$）次移位。由 4-(4)和 4-(5)式可知，每次叠代 x_{i+1} 、 y_{i+1} 分别比准确值增大 $\sqrt{1+2^{-2(i-2)}}$ 倍。 n 次旋转后， x_{n+1} 、 y_{n+1} 比准确值增大 $\sqrt{1+2^{0}} \cdot \sqrt{1+2^{-2}} \cdots \sqrt{1+2^{-2(n-2)}}$ 倍。故需要对 x_{i+1} 、 y_{i+1} 进行修正，乘以 $\left[1+2^{-2(i-2)}\right]^{-\frac{1}{2}}$ 因子。这

样，就给 VLSI 设计带来了困难。因此，要找到一组数列，其第 i 项既能近似 $\left[1+2^{-2(i-2)}\right]^{-\frac{1}{2}}$，又便于 VLSI 实现。如果用 $2^{-1}+2^{-2}+\cdots+2^{-i}$ 代替 $\left[1+2^{-2(i-2)}\right]^{-\frac{1}{2}}$，结果是满意的，即有

$$2^{-1}+2^{-2}+\cdots+2^{-i}=1-2^{-i}$$

$$\begin{aligned} y_{i+3,2} &= (1-2^{-i})y_{i+1,1} \\ &= y_{i+1,1}-2^{-i}y_{i+1,1} \end{aligned} \qquad 4-(14)$$

$$\begin{aligned} x_{i+1,2} &= (1-2^{-i})x_{i+1,1} \\ &= x_{i+1,1}-2^{-i}x_{i+1,1} \end{aligned} \qquad 4-(15)$$

校正计算也转化为二进制数的移位与加法运算了。为了保证校正误差收敛，校正步选择 $i=2,3,4,7,8,10,12,14,16,19,20,21\cdots\cdots$。

4.1.2.2 矢量运算

CORDIC 另一种模式为矢量运算[17, 18]。

$M(x,y)$ 是直角坐标 XOY 是的一点，求向量 $\overrightarrow{OM}$ 的模和幅角 θ。

与旋转相类似。通过移位和加法运算，使 y 向零靠拢，相应修正 x、z。y 逐渐趋向于零，n 次操作后，z_0 即向量的幅角，x_0 即该向量的模。矢量运算与旋转的差异为：

旋转 $\quad \delta_i=\begin{cases}+1 & z_i\geq 0\\ -1 & z_i<0\end{cases}$

矢量 $\quad \delta_i=\begin{cases}+1 & y_i\geq 0\\ -1 & y_i<0\end{cases}$

由 CORDIC 基本算法扩展，可以方便地在线性、三角、双曲等函数范围内实现旋转与矢量操作。

4.2 基于 CORDIC 算法的 FIR/IIR 格型滤波单元结构设计

为使上述 FIR 格形滤波基于 CORDIC 算法方式实现，可将 3-(21)式表示为：

当$|k_m|<1$时

$$\begin{bmatrix} e_{m-1} \\ r_{m-1} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{1-k_m^2}} & \frac{-k_m}{\sqrt{1-k_m^2}} \\ \frac{-k_m}{\sqrt{1-k_m^2}} & \frac{1}{\sqrt{1-k_m^2}} \end{bmatrix} \times \begin{bmatrix} \sqrt{1-k_m^2} & 0 \\ 0 & \sqrt{1-k_m^2} \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} = \begin{bmatrix} \cosh\theta_m & \sinh\theta_m \\ \sinh\theta_m & \cosh\theta_m \end{bmatrix} \times \begin{bmatrix} \sqrt{1-k_m^2} & 0 \\ 0 & \sqrt{1-k_m^2} \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} \quad 4-(16)$$

其中 $\theta_m = \tanh^{-1}(-k_m)$

当$|k_m|>1$时

$$\begin{bmatrix} e_{m-1} \\ r_{m-1} \end{bmatrix} = \begin{bmatrix} \frac{|k_m|}{\sqrt{k_m^2-1}} & \frac{-sign(k_m)}{\sqrt{k_m^2-1}} \\ \frac{-sign(k_m)}{\sqrt{k_m^2-1}} & \frac{1}{\sqrt{k_m^2-1}} \end{bmatrix} \times \begin{bmatrix} -sign(k_m)\sqrt{k_m^2-1} & 0 \\ 0 & -sign(k_m)\sqrt{k_m^2-1} \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} = \begin{bmatrix} \cosh\theta_m & \sinh\theta_m \\ \sinh\theta_m & \cosh\theta_m \end{bmatrix} \times \begin{bmatrix} -sign(k_m)\sqrt{k_m^2-1} & 0 \\ 0 & -sign(k_m)\sqrt{k_m^2-1} \end{bmatrix} \times \begin{bmatrix} e_m \\ r_m \end{bmatrix} \quad 4-(17)$$

其中 $\theta_m = \tanh^{-1}(-1/k_m)$。

为方便各种算法及功能的格形模块统一到同一基本单元中，这里规定两输入端符号为 x_{in} 和 x_{in}'，输出端符号为 x_{out} 和 x_{out}'，反射系数 k_i。因此，$|k_i|<1$ 和 $|k_i|>1$ 时 FIR 格形滤波基本单元分别如图 4.2 和图 4.3 所示。

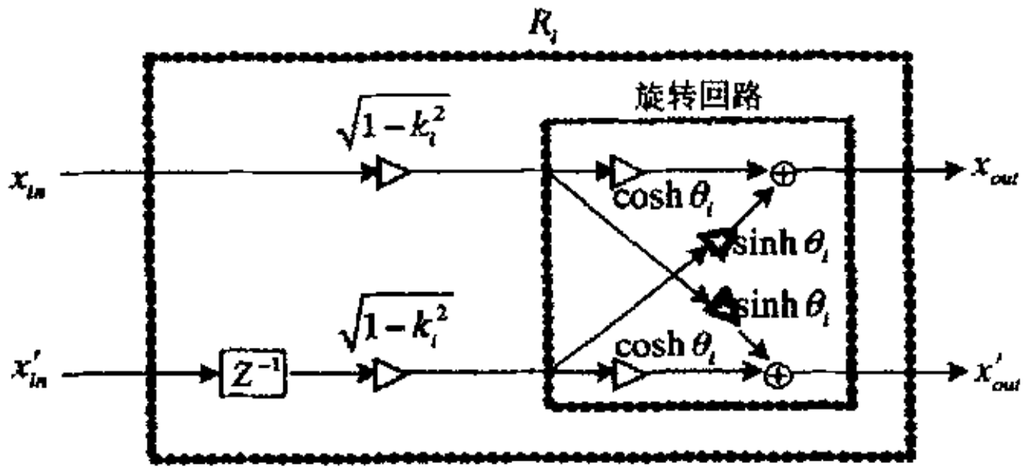


图 4.2 $|k_m|<1$ 时格形滤波单元结构

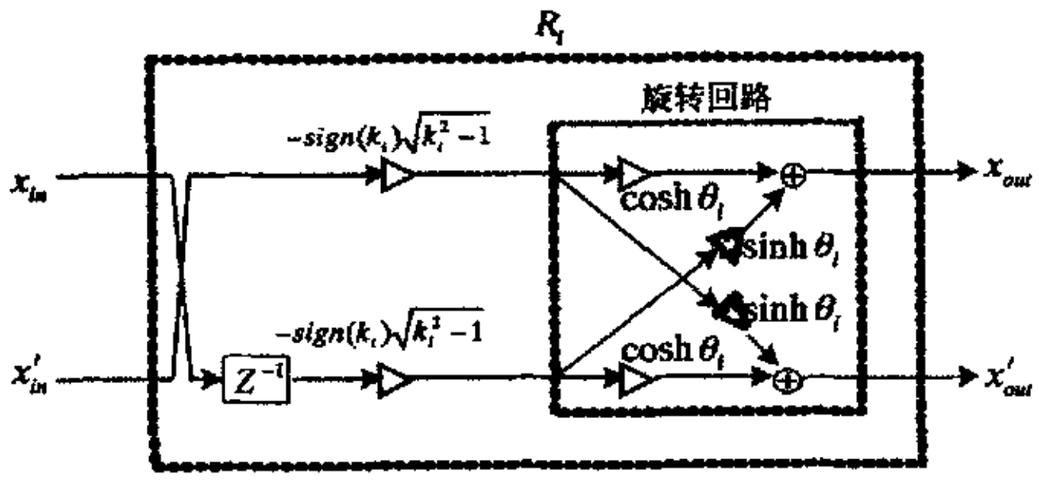


图 4.3 $|k_m|>1$ 时格形滤波单元结构

则整个格形 FIR 滤波结构如图 4.4 所示

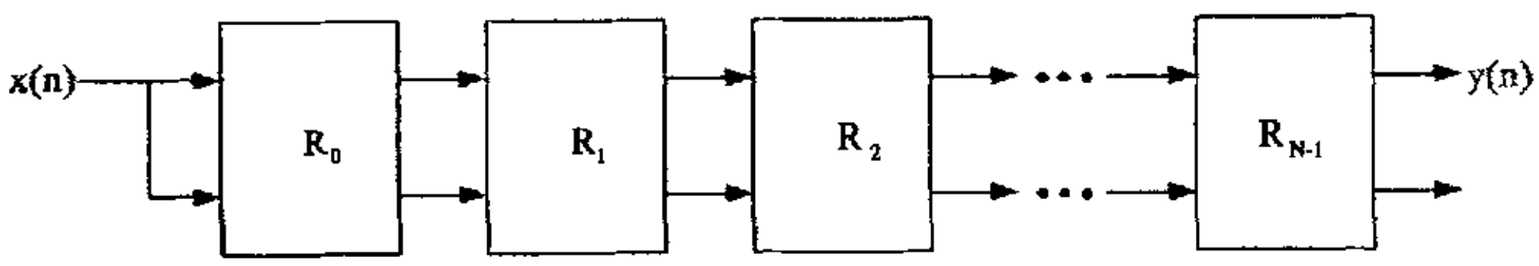


图 4.4 FIR格形滤波结构

同样，对于基于 CORDIC 的 IIR 格形滤波单元如图 4.5 所示：

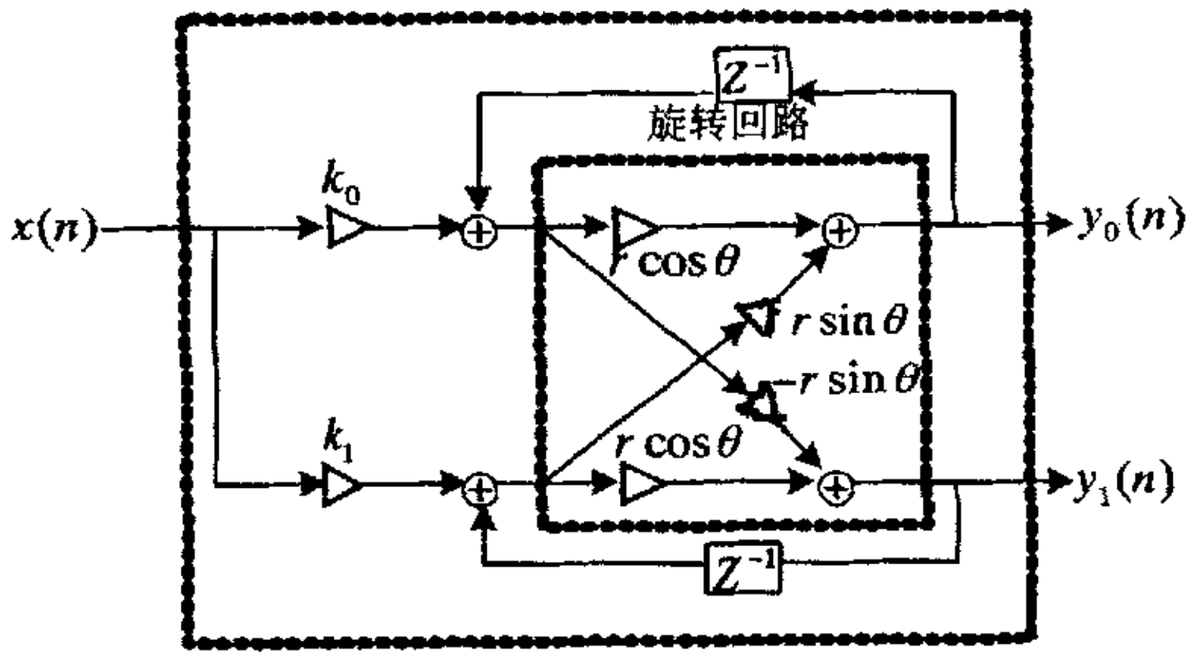


图 4.5 二阶IIR格形滤波单元

其输出函数可表示为

$$\tilde{H}_0(z)=\frac{Y_0(z)}{X(z)}=\frac{r(k_0\cos\theta+k_1\sin\theta)-r^2k_0z^{-1}}{1-2r\cos\theta z^{-1}+r^2z^{-2}} \qquad 4\text{-}(18)$$

$$\tilde{H}_1(z)=\frac{Y_1(1)}{X(z)}=\frac{r(k_1\cos\theta-k_0\sin\theta)-r^2k_1z^{-1}}{1-2r\cos\theta z^{-1}+r^2z^{-2}} \qquad 4\text{-}(19)$$

给定偶数阶实系数 IIR(ARMA)滤波器，可首先改写为级联形式

$$H(z)=\prod_{i=0}^{N/2-1}H_i(z) \qquad 4\text{-}(20)$$

且每个子滤波器 $H_i(z)$ 形式如下

$$\begin{aligned}H_i(z)&=\frac{c_i+d_iz^{-1}+c_iz^{-2}}{1+a_iz^{-1}+b_iz^{-2}}\\&=c_i+z^{-1}\frac{d_i'+c_i'z^{-1}}{1+a_iz^{-1}+b_iz^{-2}}\\&=c_i+z^{-1}A_i(z)\end{aligned} \qquad 4\text{-}(21)$$

这里的 $d_i'=d_i-a_ic_i$，$c_i'=c_i-b_ic_i$。将式 4-(21)与 4-(18)和 4-(19)比较，我们可以看出，只要设置好式 4-(18)或 4-(19)中的参数 k_0、k_1、r 和 θ 即可得到 $A_i(z)$ 的值 。参数转换如下：

FOR $i=0,1,\ldots,N/2-1$,

1) 找出 $H_i(z)$ 的极点

$$p_{0,i} = \frac{-a_i + \sqrt{a_i^2 - 4b_i}}{2}, p_{1,i} = \frac{-a_i - \sqrt{a_i^2 - 4b_i}}{2} \qquad 4\text{-}(22)$$

2) 计算r_i和θ_i的值

a) 当$a_i - 4b_i < 0$时， $r_i = mag(p_{0,i})$， $\theta_i = \arg(p_{0,i})$ 4-(23)

b) 当$a_i - 4b_i > 0$时，可通过下式计算r_i和θ_i的值

$$\begin{aligned} 2r_i \cos\theta_i &= p_{0,i} + p_{1,i} \\ r_i^2 &= p_{0,i} \cdot p_{1,i} \end{aligned} \qquad 4\text{-}(24)$$

则有：

$$r_i = \sqrt{p_{0,i} \cdot p_{1,i}}, \quad \theta_i = \cos^{-1} \frac{p_{0,i} + p_{1,i}}{2\sqrt{p_{0,i} \cdot p_{1,i}}} \qquad 4\text{-}(25)$$

3) 计算k_0和k_1

由等式

$$\begin{aligned} r_i(k_0 \cos\theta_i + k_1 \sin\theta_i) &= d_i' \\ -r_i^2 k_0 &= c_i' \end{aligned} \qquad 4\text{-}(26)$$

可计算得

$$k_0 = -c_i' / r_i^2, \quad k_1 = (d_i' / r_i - k_0 \cos\theta_i) / \sin\theta_i \qquad 4\text{-}(27)$$

End.

现在我们可以基于式 4-(20)和 4-(21)，利用图 4.5 所描述的结构来实现$H(z)$。每一阶段都完成一个$H_i(z)$滤波，其中$i = 0,1,\ldots,N/2-1$，这里$A_i(z)$是由图 4.5 中的二阶 IIR 模块实现的。注意尺度因子c_i也是由 IIR 模块通过设置$k_0 = c_i$， $k_1 = 0$， $r = 1$，$\theta = 0$并断开反馈数据路径来实现的。我们用一个虚线框标出了这一实现。这样，实现c_i和$A_i(z)$的模块有相同的响应时间。因此，图 4.5 中加法器的两个输入可以同步。

为了完成一个 N 阶(N 是偶数) $H(z)$，我们总共需要 N 个 IIR 格形模块并行运行。最大数据吞吐速率受 IIR 模块内部的反馈回路的限制。

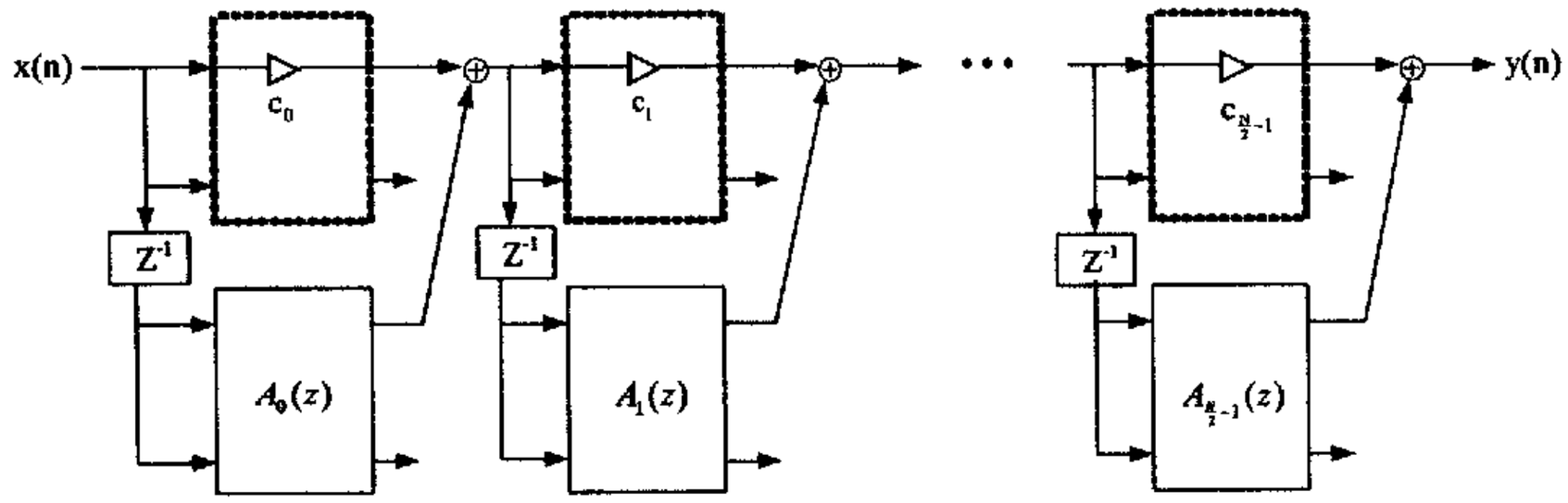


图 4.6基于二阶IIR格形单元的IIR（ARMA）结构

4.3 基于 CORDIC 算法的 DT 单元结构设计

对于 DT 变换，设计基本单元结构如图 4.7 所示

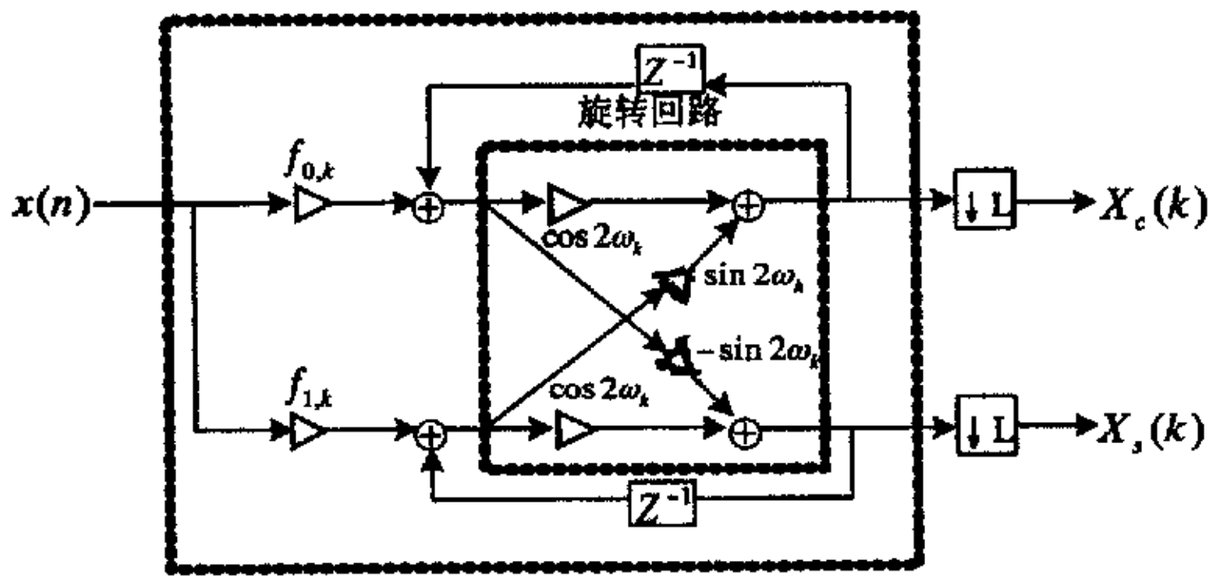


图 4.7 DT基本单元

根据 3.4 节对离散变换的统一结构描述，其尺度因子和旋转操作可给出如下：

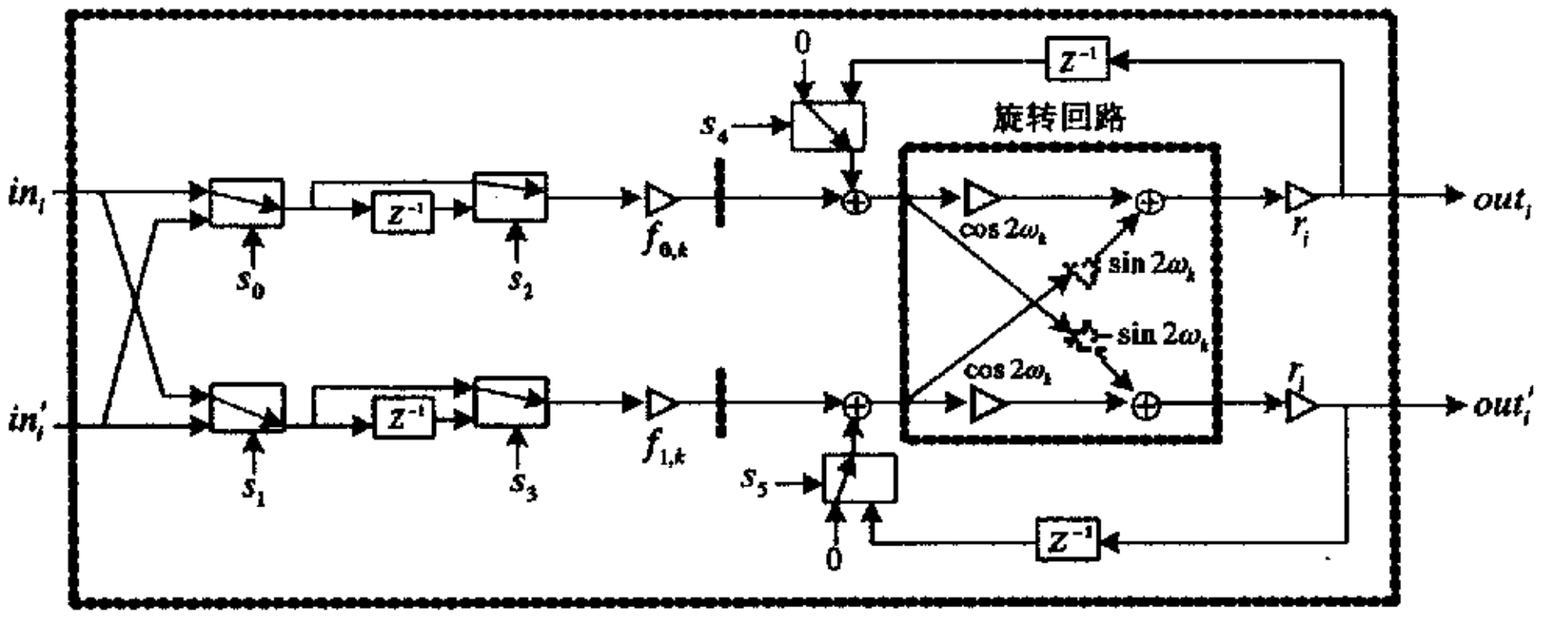
$$f_k = \begin{bmatrix} f_{0,k} \\ f_{1,k} \end{bmatrix} = \begin{bmatrix} \alpha\cos((2L+1)\omega_k + \theta_k \\ \alpha\sin((2L+1)\omega_k + \theta_k \end{bmatrix} \qquad 4-(28)$$

$$R_k = \begin{bmatrix} \cos(2\omega_k) & \sin(2\omega_k) \\ -\sin(2\omega_k) & \cos(2\omega_k) \end{bmatrix} \qquad 4-(29)$$

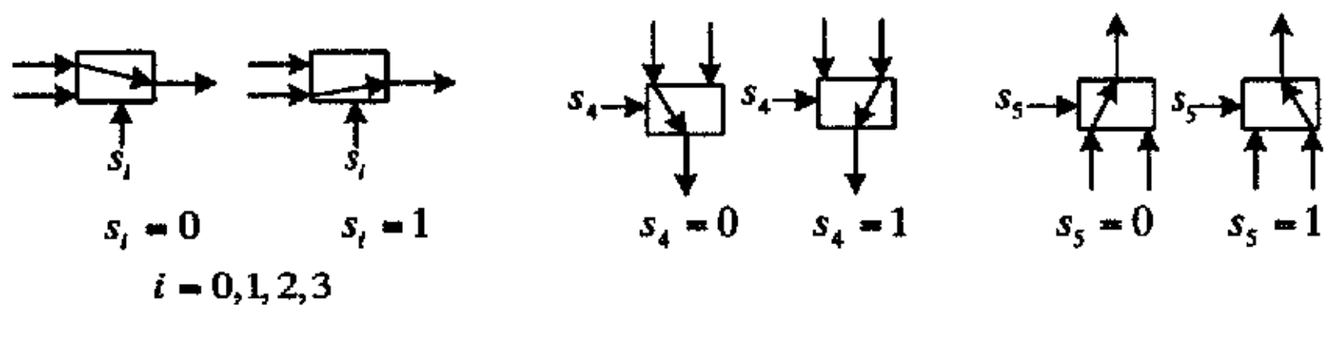
4.4 统一可编程计算模块设计

从图 4.2、图 4.3、图 4.5 和图 4.7 我们可以看出所有的基本单元结构中，除了在数据路径、参数(因子系数和旋转角度)不同外，其他都是统一的。因此，考虑用

六个开关来控制数据路径，用四个尺度因子及一个统一的旋转回路来确定不同的系数及旋转角度。则 FIR/IIR/DT 的统一可编程基本处理单元如图 4.8 所示。为了缩短该处理单元的关键路径，在尺度因子 $f_{0,i}$ 和 $f_{1,i}$ 后分别插入一个流水线（图 4.8 中的虚线部分）。通过设置开关及参数，统一可编程模块可用作 FIR/IIR/DT 中的基本处理单元（PE）。



(a)



(b)

图 4.8 a 统一的可编程单元结构
b 单元结构中用到的开关

六个开关$[S_0S_1S_2S_3S_4S_5]$控制了模块内的数据路径：开关对S_0和S_1从in_i或in_i'选择输入：当$S_0S_1=00$时，in_i成为 FIR 和 IIR 模块中需要的格形输入。$S_0S_1=10$，可以交换$|k_i|>1$时的 FIR 格形输入。开关S_2和S_3决定是否使用延迟单元。$S_2S_3=01$，则左下延迟单元被包括在数据路径中，FIR 格形中要用到它。设置$S_2S_3=11$，则图 4.5 中的延迟单元可被整合到模块$A_i(z)$中。因此，不需要在 IIR 滤波操作中直接实现延迟单元。最后一个开关对是S_4和S_5。他们控制着模块中的两个反馈路径：当$S_4S_5=11$时，IIR 和 DT 操作中需要的当前输入的延迟模块输出被加进来。设置$S_4S_5=11$将断开反馈路径，因

为只有当将 IIR 功能整合到这个统一模块设计中时，才需要旋转回路输出的两个因子。

除了数据路径，还需要设置标度乘数 $f_{0,i}$、$f_{1,i}$ 和 r_i 的值，以及旋转角度 θ_i。给定 DSP 功能描述，这些参数可以根据第三章中讨论得出的计算公式来确定。FIR /IIR/DT 操作的可编程模块的完整设置见图 4.1。

表 4.1　第 i 个可编程模块设置

	$S=[s_0s_1s_2s_3s_4s_5]$	$r_i=[f_{0,i},f_{1,i}]^T$	R_i
FIR ($\lvert k_i\rvert<1$)	$\begin{cases}000100 & (M_0)\\ 010100 & (M_i,i\neq 0)\end{cases}$	$\begin{bmatrix}\sqrt{1-k_i^2}\\ \sqrt{1-k_i^2}\end{bmatrix}$	$\begin{bmatrix}\cosh\theta_i & \sinh\theta_i\\ \sinh\theta_i & \cosh\theta_i\end{bmatrix}$
FIR ($\lvert k_i\rvert>1$)	$\begin{cases}000100 & (M_0)\\ 100100 & (M_i,i\neq 0)\end{cases}$	$\begin{bmatrix}1\\ 1\end{bmatrix}$	$\begin{bmatrix}\cosh\theta_i & \sinh\theta_i\\ \sinh\theta_i & \cosh\theta_i\end{bmatrix}$
IIR (c_i)	000000	$\begin{bmatrix}c_i\\ 0\end{bmatrix}$	$r_i=1,\theta_i=0$
IIR ($A_i(z)$)	001111	$\begin{bmatrix}k_0\\ k_1\end{bmatrix}$	$\begin{bmatrix}r_i\cos\theta_i & r_i\sin\theta_i\\ -r_i\sin\theta_i & r_i\cos\theta_i\end{bmatrix}$
DT	000011	$\begin{bmatrix}\alpha\cos((2L+1)\omega_k+\theta_k)\\ \alpha\sin((2L+1)\omega_k+\theta_k)\end{bmatrix}$	$\begin{bmatrix}\cos 2\omega_i & \sin 2\omega_i\\ -\sin 2\omega_i & \cos 2\omega_i\end{bmatrix}$

4.5　参数计算用例

下面将举例说明如何将一个给定系统中的参数转换为用于可编程模块的参数。这里限定 IIR（ARMA）滤波器中的分子和分母的幂次都是偶数的，这样便于完成所有必要的分解。并假设用来完成 FIR/IIR 功能的系统的模块数为 10。则 FIR/IIR 的最大幂次是 10，DT 的变换大小也被限于 10。对于 DT，我们使用一个 8 点 DCT 作为设计用例，因为这在变换编码应用中比较流行一些。

4.5.1　FIR 滤波

给定 FIR 传输函数如下：

$$\begin{aligned}H(z) = {} & 1-0.8843z^{-1}-0.1327z^{-2}-1.1219z^{-3} \\ & +0.5328z^{-4}-0.8882z^{-5} \\ & +0.1038z^{-6}-0.3786z^{-7} \\ & +0.2195z^{-8}-0.1094z^{-9}\end{aligned} \qquad 4-(30)$$

首先根据 4-(22)式和 4-(27)式来计算所有的 PARCOR 系数。

$k_0 = -0.4472, k_1 = -0.6917, k_2 = -0.5865,$
$k_3 = 4.1573, k_4 = 1.1595, k_5 = 0.2655,$
$k_6 = 0.2942, k_7 = -0.1243, k_8 = 0.1094.$

完整设置列于表 4.1 中。

4.5.2 IIR(ARMA)滤波

给定 IIR(ARMA)滤波器如下：

$$H(z) = \frac{1+\sum_{i=1}^{M} p_i z^{-i}}{1+\sum_{i=1}^{N} q_i z^{-i}} \qquad 4-(31)$$

其中 $M = 4$， $N = 10$ 且有

$p_1 = -107314, p_2 = 1.6788, p_3 = -0.7913,$
$p_4 = 0.2304, q_1 = 0.4036, q_2 = 1.3227,$
$q_3 = 0.2376, q_4 = 1.1558, q_5 = 0.0047, q_6 = 0.6950$
$q_7 = -0.0733, q_8 = 0.2735, q_9 = -0.0542,$
$q_{10} = 0.0788$

我们首先将其写成级联形式如下：

$$\begin{aligned}H(z) = {} & \left(1+z^{-1}\frac{-0.2122+0.2175z^{-1}}{1-0.9192z^{-1}+0.4225z^{-2}}\right) \\ & \times\left(1+z^{-1}\frac{0.1500-0.2025z^{-1}}{1-0.7500z^{-1}+0.5625z^{-2}}\right) \\ & \times\left(1+z^{-1}\frac{-0.8000-0.6400z^{-1}}{1+0.8000z^{-1}+0.6400z^{-2}}\right) \\ & \times\left(1+z^{-1}\frac{-1.2728-0.8100z^{-1}}{1+1.2728z^{-1}+0.8100z^{-2}}\right)\end{aligned} \qquad 4-(32)$$

接下来可依据式 4-(22)—4-(27)计算所有参数，详细设置见表 4.2。

4.5.3 块 DCT 变换

对于 8 点块 DCT 变换，可根据 3.4 节的推导来计算每个模块的 $f_{0,i}$， $f_{1,i}$ 和 θ_i。详

细设置见表 4.3。

表 4.1 FIR 滤波参数

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
S	000100	010100	010100	100100	100100	010100	010100	010100	010100
$f_{0,i}$	0.8944	0.7222	0.8100	4.0352	-0.5870	0.9641	0.9557	0.9922	0.9940
$f_{1,i}$	0.8944	0.7222	0.8100	4.0352	-0.5870	0.9641	0.9557	0.9922	0.9940
r_i	1	1	1	1	1	1	1	1	1
θ_i	0.4812	0.8512	0.6723	0.2450	-1.3027	-0.2720	-0.3032	0.1249	-0.1098

表4.2 IIR滤波参数

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
S	000000	001111	000000	001111	000000	001111	000000	001111	000000	001111
$f_{0,i}$	1	-0.5148	1	0.3600	1	1	1	1	1	1
$f_{1,i}$	0	0.0531	0	0.0231	0	-0.5774	0	-1	0	0
r_i	1	0.6500	1	0.7500	1	0.8000	1	0.9000	1	0.8000
θ_i	0	0.7854	0	1.0472	0	2.0944	0	2.3562	0	1.5708

表4.3 DCT滤波参数

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
S	000011	000011	000011	000011	000011	000011	000011	000011
$f_{0,i}$	0.3536	-0.4904	0.4619	-0.4157	0.3536	-0.2778	0.1913	-0.0975
$f_{1,i}$	0	-0.0975	0.1913	-0.2778	0.3536	-0.4157	0.4619	-0.4019
r_i	1	1	1	1	1	1	1	1
θ_i	0	0.3927	0.7854	1.1781	1.5708	1.9635	2.3562	2.7489

4.6 本章小结

本章引入了 CORDIC 算法，对 FIR/IIR/DT 算法进行了更深一步的讨论。在此基础上找出他们基于旋转方式实现时的共同部分，为这些算法的实现设计了一个统一的可编程模块。在这一可编程模块中，不同的算法具有不同的数据路径和系统参数，这些参数的计算是由主处理器完成并传到可重构 DSP 计算引擎系统中来的，数据路径的选择开关也是由模块外部的控制逻辑来控制的。因此在完成这一可编程模块的

设计之后，给出了几参数计算用例。

下一章将讨论如何利用本章的可编程模块来设计可重构 DSP 计算引擎的系统结构。

第五章　基于 FPGA 的 DSP 计算引擎系统结构设计

在该系统的设计中，采用了自顶向下（Top-Down）的设计方法。由于在现代电子设计中，电路的功能越来越复杂，规模越来越大，再要使用门级的设计方法，不但会导致设计周期的延长，而且，要想保证设计的正确性和可靠性必须要以大量的人力和物力投入为代价。因此在如今的电子设计中自顶向下的高层设计方法已经成为主流。这种设计方法的主要流程就是：先从电路的功能出发，使用 HDL（硬件描述语言）从较高层次对电路进行描述，然后在仿真工具上进行仿真，在理想状况下对电路进行最基本的检查。保证无误之后，把设计文件交给综合工具进行综合与优化，形成门级的网表（netlist）。在此基础上再对产生的带有延时信息的网表文件进行仿真，验证其功能。最后，把设计交给布局布线工具，生成真正的电路设计。整个过程都是在 EDA 工具的帮助下完成的。

5.1　系统结构设计

这节将在上一节可编程计算模块设计的基础上讨论具有 FIR/IIR/DT 功能的系统结构的设计。通过恰当的设置前一节中讨论的统一模块的参数，并利用可重构的互联网络对多个模块进行连接，即可以完全的流水线方式完成 FIR/ IIR/DT 中的功能。

在前一节中，分析并设计出了一个可用作 FIR/IIR/DT 中的基本 PE 的统一可编程模块。然而，对于 FIR/IIR/DT 中的每一个功能，这些基本 PE 都以不同的方式连接的，因此需要用一个可重构的互联网络来完成这种互联任务，以达到在同一系统结构中完成不同的 DSP 功能的目的。下面将分别看一下不同的功能实现中所使用的不同网络互联模式。

以下将用 N 来表示滤波器的阶数和离散变换中的块数。

5.1.1　FIR 滤波功能设计

在实现 FIR 滤波功能的设计中，各模块间的网络互联可按如下算法进行：

1. $in_0 = x(n)$.

2. For $m = 0,1,\ldots,(N-2)$

$in_{m+1} = out_{m,}$

$in'_{m+1} = out'_{m,}$

End

3. $y(n) = out_{N-1,}$

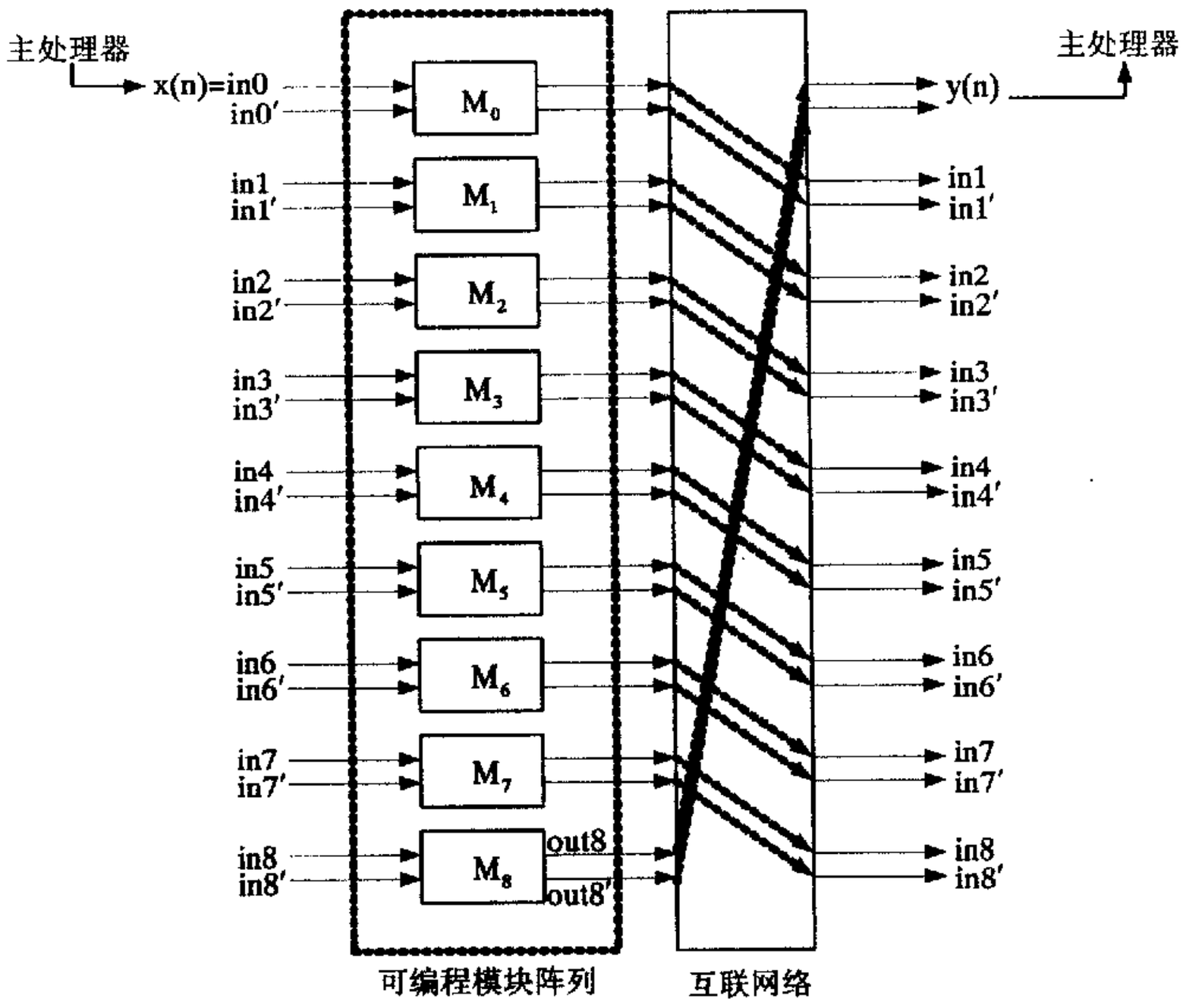


图 5.1 FIR滤波系统结构

图 5.1 所示为按以上算法配置互联网络结构的 FIR 滤波模式下的系统结构。此系统包括了两部分：一是 P 个独立的统一可编程模块组成的可编程模块阵列。另一个是根据数据路径连接可编程模块的可重构互联网络。

5.1.2 IIR 滤波功能设计

在实现 IIR 滤波功能的设计中，各模块间的网络互联可按如下算法进行：

1. $in_0 = x(n), i = 0,1.$

2. For $m = 0,1,\ldots,(N-3)$

$in_{m+2} = out_{2\lfloor \frac{m}{2} \rfloor} + out_{2\lfloor \frac{m}{2} \rfloor + 1.}$

End

3. $y(n) = out_{N-1} + out_{N-2.}$

图 5.2 所示即为图 4.9 中的 IIR 格形模块按以上算法配置互联网络结构的 IIR 滤波模式下的系统结构。

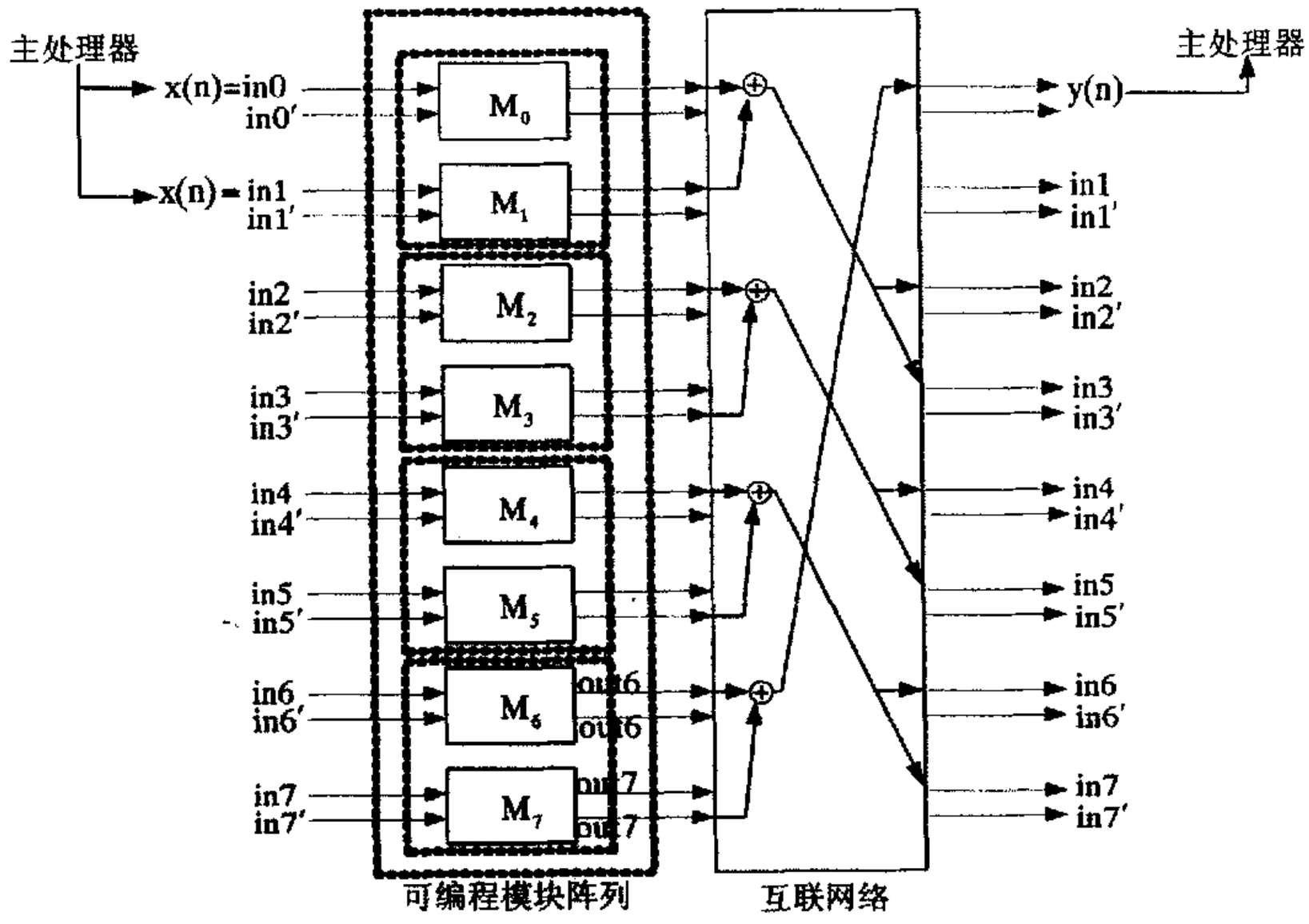


图 5.2 IIR滤波系统结构

5.1.3 DT 功能设计

各个离散变换的网络互联可分别按以下算法实现：

DFT：

1. $in_i = x(n)$,

   $i = 0,1,\ldots,N-1$

2. $X_{DFT}(i) = out_i + j * out_i'$,

   $i = 0,1,\ldots,N-1$.

DHT：

1. $in_i = x(n)$,

   $i = 0,1,\ldots,N-1$

2. $X_{DHT}(i) = out_i + out_i'$,

   $i = 0,1,\ldots,N-1$

DCT：

1. $in_i = x(n),$

$i = 0,1,\ldots,N-1$

$X_{DCT}(i) = out_i,$

$i = 0,1,\ldots,N-1$

从以上算法分析及系统结构设计图不难看出，基于本设计的完整的可重构 DSP 计算系统的工作流程如下：

在初始化模式下，主机处理器将根据功能类型(FIR/IIR/DT)和表 II 中所列的功能描述计算所有必需的参数 f_i、r_i、θ_i，并以查找表的形式存贮。系统通过查找表找到必要的参数以减少模式配置时间。接下来，主机处理器需要根据表 III 中所列的功能类型配置互联网络。一旦计算系统初始化完成，就进入了执行模式。在 FIR/IIR/QMF 应用中，主机处理器连续的将数据序列传递给计算系统。所有的 PE 都并行运行，且处理器可以以完全的流水线方式收集滤波输出。在块 DT 应用中，块输入数据被串行送入计算引擎。可编程模块阵列的每一个 PE 会同时更新 4-(41)中的 $X_c(k)$ 和 $X_s(k)$，其中 $k = 0,1,\ldots,N-1$。当最后一个数据进入可编程模块阵列之后，DT 系数的赋值也随之完成。接下来，互联网络将根据表 I 中所定义的连结功能连续的进行模块的输出。变换系数可以在网络输出过程中并行取得。同时，主机处理器将重置所有可编程模块的寄存器，使其为 0，这样可迅速产生下一个块变换。

5.2 仿真综合及结果分析

当系统的结构确定后，对于系统中各模块的设计首先是使用硬件描述语言完成系统的描述，然后使用系统仿真及调试工具进行语言级功能仿真与调试。可重构计算技术是以 FPGA 为基础的，本文的整个设计是基于 FPGA 的，最终下载到可重构实验板上。FPGA 的设计流程如图 5.3 所示[19]：

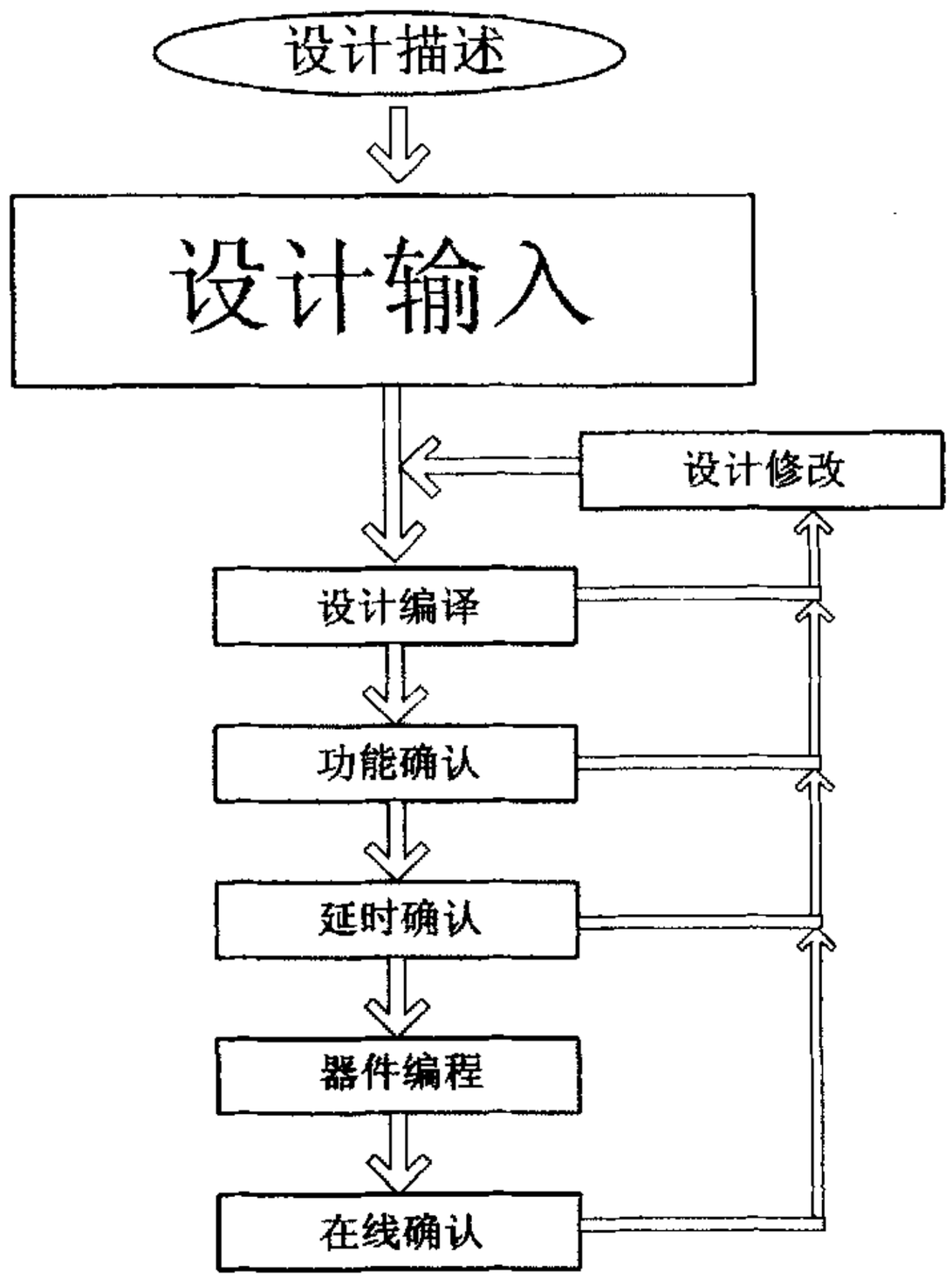


图 5.3 FPGA设计流程

下面将以本设计中的 FIR 滤波及 DCT 变换为例介绍对该设计的综合、仿真与测试。

5.2.1 系统描述与功能仿真

本论文的实例算法设计部分采用的是自顶向下的全正向设计方法，以 Verilog HDL[20]作为设计输入工具，在使用 Verilog HDL 语言对系统进行描述时，就必须考虑到对系统的功能仿真。这是因为 Verilog HDL 语言本身是一种可仿真的高级语言，象其它高级语言经过编译可以运行一样，Verilog HDL 经过高层仿真工具处理后也可以“运行”，只不过这种运行是把描述的对象---硬件电路“模拟”地表现出来，这种运行称之为仿真，通常对初步描述的电路所做的仿真称为功能仿真。功能仿真不带时延信息，只是对设计的系统的功能做初步测试。仿真前首先要利用波形编辑器或硬件描述语言建立测试向量，仿真结果以报告的形式或波形的方式输出，从中可以看到各节点的结果，如果有错误，则返回设计输入中修改逻辑设计，系统功能仿真的结果直接决定着时序仿真的正确性，所以功能仿真的结果一定要正确。

常用的方法是测试基准程序(Testbench)法[21]，专门设计的仿真程序，也称为测试程序。利用测试基准程序，可以被测模块的测试向量实现自动输入，并通过波形输出文件记录输出，这种方法便于将仿真结果记录归档和比较。

本设计使用 ModelSim[21]作为语言级仿真工具，测试用例的编写也是使用 Verilog HDL 语言，要完成一个系统的正确的 Verilog HDL 描述，就必须要进行系统的功能仿真与调试。在进行仿真调试时，除熟练使用高层次设计与仿真工具外，主要还是测试基准程序的编写，它是验证验证系统是否正确的基础，而且也是进行高层次仿真的先进的方法。本系统的 Testbench 功能主要是从文件中读入输入数据，也包括提供一些对象初始化和时钟信号以及复位信号。功能仿真不仅仅局限于整个系统，事实上，设计完成每一个模块都要设计对应的 TestBench 进行功能仿真，用以保证当整个系统完成时各个模块可以正常工作，在保证各模块功能正确的情况下，整个系统的仿真更倾向于逻辑结构正确性的验证。

输入数据是测试运算功能的重要途径，因此测试向量的设计也要涉及到各种运算情况。

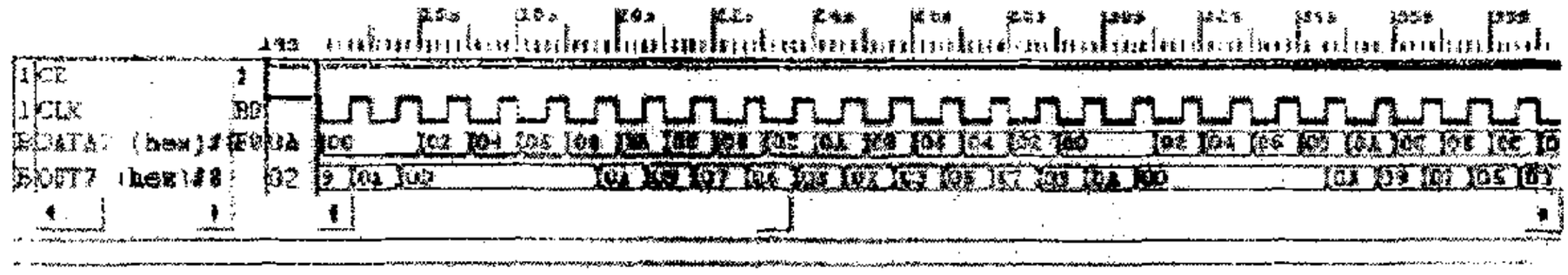
16 点 DFT 的仿真结果如下所示：

```
Input:
0X3FB0 0x0000 0xC080 0x0000 0x0000 0x0000 0x0000 0x0080 0X4000 0X0000
0xE500 0x0B00 0X3E60 0XD300 0x0000 0x0000 0xCF80 0x0000 0X0000 0X0004
0x0000 0X0000 0xC040 0X0000 0X0000 0x0000 0X0000 0x0015 0X8100 0X0001
0xBF80 0X0000 0X0000 0x0000 0X0000 0X0000 0X0000 0x0000 0X0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
0X0000 0x0000 0x0000 0x0000 0x0000 0x0000 0X0000 0X0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000
Output:
0X0000 0X0000 0X3E60 0XD300 0XC100 0X0000 0X4071 0X22D0 0X30B0 0X0020
0xE152 0x0B11 0X3E00 0x02B2 0X3EFF 0x0000 0x3F00 0X0000 0x3EFF 0X0000
0X3E00 0x02B2 0xE152 0x0B11 0X30B0 0X0020 0X4071 0X22D0 0xC100 0X0000
0X3E6D 0XD300 0X0000 0X0000 0XBF86 0xC360 0xAD00 0X1001 0xBF86 0xC360
0xEE00 0X12B0 0xBF86 0xC360 0xE090 0X2254 0xD020 0X58C1 0X0000 0X0000
0x6D20 0X58C1 0X6090 0X2254 0X3F86 0xC360 0X3ED0 0X12B0 0X3F86 0xC360
0X2D00 0X1001 0X3F86 0xC360
```

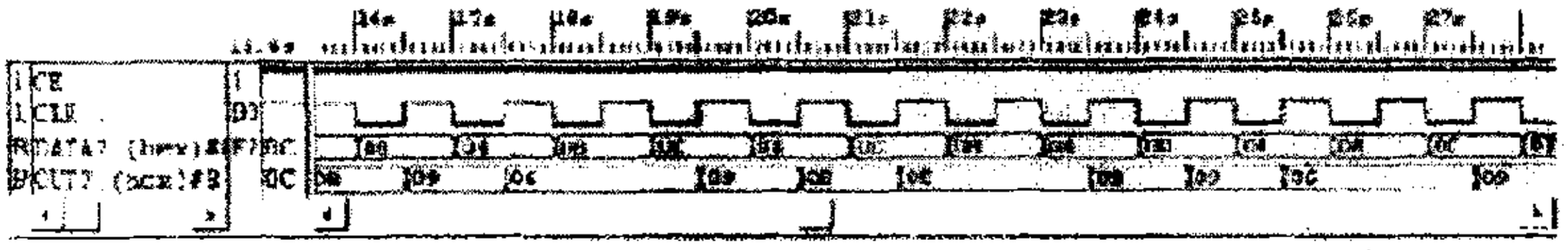
为了验证结果的正确性，在以硬件描述语言完成设计的同时，作者使用 C 语言进行了软件设计实现，尽管软硬件设计在时间上没有可比性，但可用来对结果行验证，最终得到的结果是一致的。表明本文对于该功能的设计是完全符合设计要求的。

以下是对本文所设计的 FIR 功能的 ModelSim 仿真结果，实例为 FIR 低通滤波，所希望的频率响应在 $0 \le |\omega| \le 0.25\pi$ 为 1，在 $0.25\pi \le |\omega| \le \pi$ 之间为 0，则有 8 抽头 8 点 FIR

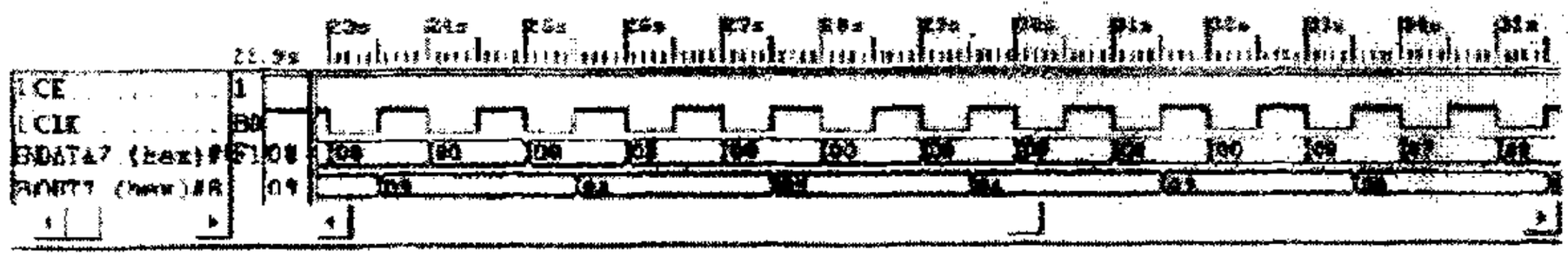
滤波仿真示图如下：



(a) 信号频率小于截止频率



(b) 信号频率等于截止频率



(c) 信号频率大于截止频率

图 5.4 FIR功能的ModelSim仿真结果

图 5.6 是将图 5.4 中的数据用曲线的形式表示，这样更直观些。图 5.5 是用 MATLAB 工具仿真的结果。

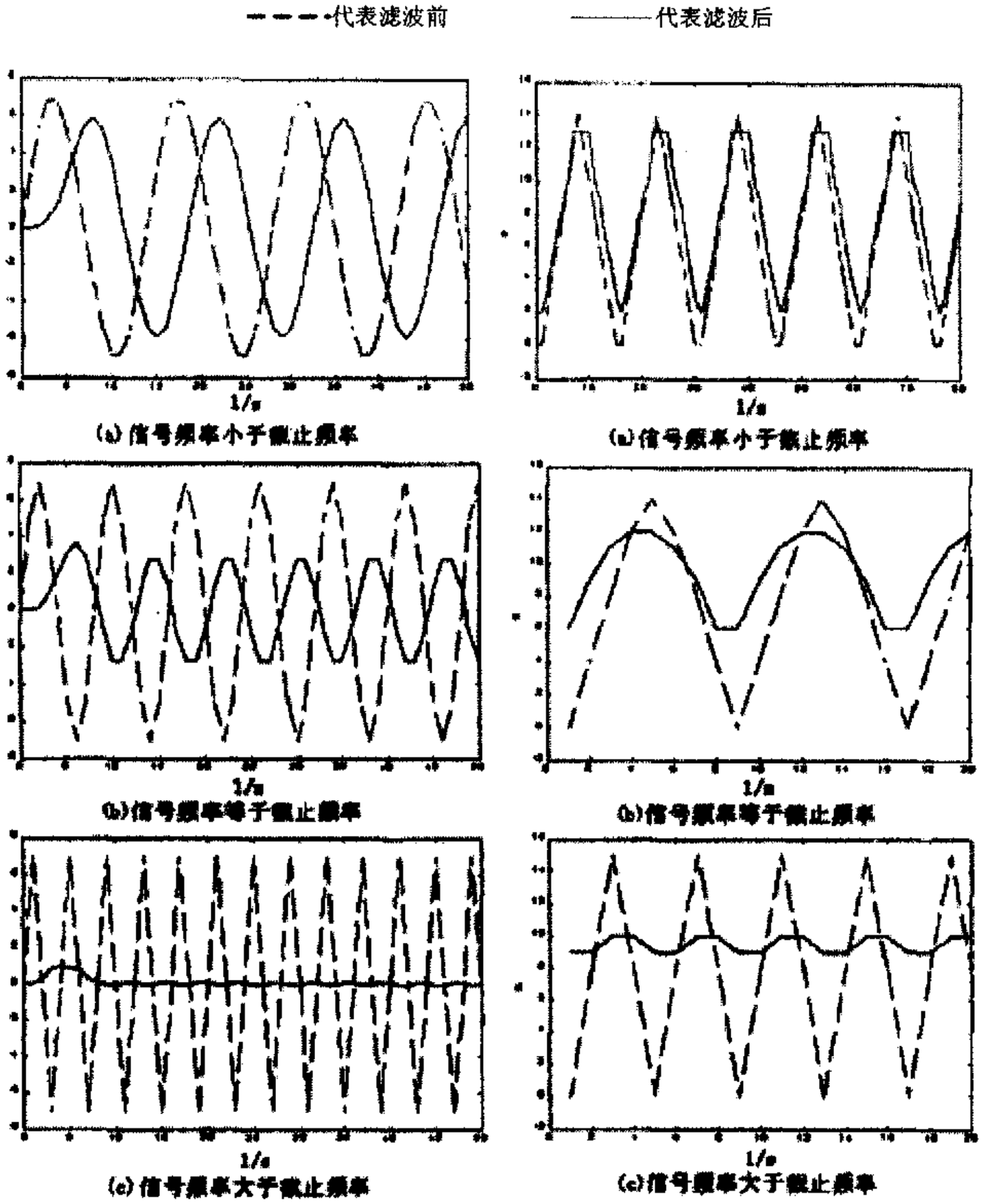


图 5.5 FIR的MATLAB仿真图　　5.6 FIR仿真曲线形式

5.2.2 仿真分析

(1)如图 5.4，从输入 00 开始，经滤波后产生输出信号，电路的延时只有 10 个时钟脉冲。因此系统的运行速度是非常快的。

(2)不管输入信号的频率是多少，系统的采样频率始终是一个时钟脉冲。如果系统的时钟脉冲是 50MHz，那采样频率也为 50MHz。

(3)当信号周期为 16s 时，系统的输出基本与输入相同，低频信号保留下来。当信号周期为 8s 时，幅值由 15 变成 6。当信号周期为 4s 时，系统的输出幅值是 1，基本上被滤掉。对比用 MATLAB 仿真工具仿真的结果，看出系统的滤波效果还是不错的。

5.2.3 系统综合

在完成系统结构设计及系统的语言描述后，接下来就是将语言的描述转化为实际电路的过程-----综合与实现。

为了控制优化输出和映射工艺，在进行系统综合时加入综合约束。它为优化和映射试图满足的工艺提供了目标，并且他们控制设计的结构实现方式。目前综合工具中可用的约束包括面积、定时、功率和可测试性约束，将来我们或许会看到对封装的约束和对布图的约束等，但是，目前最普通的约束是按面积和按时间方式的约束。

一般情况下，在进行系统综合时，首先要在面积约束下进行综合。这样，在综合时可以将一些冗余逻辑进行简化，从而减少系统的面积。在一般的综合工具中，在最顶层的面积约束条件可以覆盖整个设计。也就是说，只要在最顶层的设计上加上面积最小的约束条件，则在其下一层乃至最低层的所有单元中这一约束条件都有效。所以在进行面积优化时，只要在最顶层单元中加入相应的约束条件即可。通常情况下，综合工具中。允许设计者指定非 0 的面积，当综合工具满足面积约束条件时就停止优化。换言之，如果指定面积约束条件为 0，则综合工具将试图用各种可能的规则和算法尽可能地减少设计的面积。这种面积约束条件的设置在对面积要求非常苛刻时比较有效，但有可能会增加一些电路的延迟，使某些模块电路的速度达不到设计的要求。因此，在初次优化时按照面积约束进行优化，然后根据不同模块对速度的要求再进行优化。

时间约束优化的目的就是通过优化，使系统在一定的速度范围内能够正常工作。要达到这一目的，首先要找出系统中的关键路径。在 Synopsys 综合工具中，提供了查找最大路径的功能，但是，对于一个复杂系统的设计来讲，任何一个综合工具都很难准确地找出系统中真正的关键路径来。这一工作，必须在设计者的干预下，通过对系统的分析，找出系统中真正的关键路径，有的放矢的进行综合优化。

另外系统的综合是基于某一给定的综合工艺库来进行的。工艺库一般是由生产厂家或 FPGA 供应商提供的。它是综合的基础，系统的综合实际上是将系统按照综合的约束映射到工艺库上的一个过程，属于一种工艺约束。

本系统的综合主要是综合工具按照设计者给出的综合约束条件自动地产生出优化的门级网表。所以，设计者在综合阶段要做的工作就是如何加入合理的综合约束条件。

由于系统是要用 FPGA 来实现，在 FPGA 中由于结构的限制，面积的约束是非常

重要的。而另一方面时间约束优化是将一些非并行方式的结构用并行的方式来实现，这样有可能增加电路的面积。所以在进行时间约束优化时，要在速度与面积之间找出一个较为合理的均衡，这样才能使系统工作在最佳状态。

在本文的设计中，采用了行为级描述，然后使用了 Synplicity 公司的高层综合工具 Synplify 进行综合优化，FSM 编码为 OneHot 格式，器件为 APEX20ke。整体综合后产生综合报告和门级网表，这生文件的后缀为.vqm，综合结果是 16 点 DFT 的最大工作频率为 59.5MHz。

5.2.4 布局布线

Altera 公司的 QuartusII 可以对整个 FPGA 设计流程进行支持，包设计输入、综合、功能仿真、布局布线和时序仿真，在本设计中只使用 QuartusII 工具对设计进行布局布线。

QuartusII 支持多种输入方式，包括：原理图式图形设计输入、文本编辑、AHDL、VHDL。Verilog、内存编辑、Hex、Mif 以及第三方工具如.edif、.hdl、.vqm 等，或采用一些别的方法去优化和提高输入的灵活性：如混合设计格式或利用 LPM 和宏功能模块来加速设计输入。

这里使用 Synplify 输出的.vpm 文件及为第三方工具输入 QuartusII 进行布线，生成.sof 文件，并使用 QuartusII 下载到可重构试验调试板上。下载模式为串行下载方式（passive serial）。

16 点 DFT 结果：

最大工作频率 61.3MHz。

Total logic elements	**8390/51840**	**16%**
Total pins	**100/488**	**20%**
Total memory bits	**2560/442368**	**<1%**

从上面的报告可以看到，通过专用工具的布局布线优化，系统的最大工作频率比综合时又有所上升，达到 60MHz 以上。

5.2.5 在线验证

本设计使用的验证环境是为华科技的 EXA1 系列开发板用于功能验证。本开发板的功能及资源介绍如下：此板采用 ALTERA 最新的 SOC 芯片 EPXA1F484C。该芯片内部集成一颗 ARM922T CPU 和 APEX20KE FPGA（10 万到 100 万门），具体资源配置

情况如下：

板上资源：

1、EXPA1F484C芯片（ARM922T、APEX20KE）；
2、10/100以太网接口；
3、两个RS232；
4、8M flash；
5、32M SDRAM；
6、三路时钟资源；
7、JTAG接口；
8、Multi-ICE调试接口；
9、两个扩展插槽；
10、八路用户自定义DIP开关；
11、八个用户自定义LED；
12、四个用户自定义按钮。

EXPA1F484C片内资源：

1、ARM922T 32位RISC CPU（带MMU、8KB指令和8KB数据缓存）；
2、APEX20KE可编程逻辑。10万门到100万门，软件可在线配置内部逻辑；
3、内部采用AMBA的AHB总线；
4、片上可编程外设（看门狗、UART、中断控制器、ETM9调试模块）；
5、256KB内内单口SRAM;128KB片内双口SRAM；
6、外部总线接口（EBI）。支持四个外设（如flash、SDRAM等），每个外设最多可达32MB地址空间；
7、三个可编程PLL。

系统采用JTAG接口下载，测试向量以文件(data,dat)形式读入，结果存入文件result.dat。

分别对8点FIR和16点DFT进行验证，验证向量与功能仿真时使用相同数据，输出结果正确，与仿真结果相同。即在无需改变硬件电路结构的情况下，只要将预先设计的固件、不同功能的网络配置及系统参数下载到可编程器件中，即可实现不同的功能，达到可重构计算的目的。

5.3 本章小结

本章利用前一章中设计的统一的可编程计算模块将前章所分析的几个重要的DSP 算法映射到同一可重构系统中去，只需简单的加载不同 DSP 算法的参数和可重构互联网络，即可在同一系统中完成不同的 DSP 任务，如 FIR，IIR，DT 等。这一并行结构保持了 ASIC 设计的优势，但比 ASIC 设计灵活的多。而且，这一结构是规则的和模块化的，非常适合于 VLSI 实现。

值得注意的是，当我们将很多算法映射到一起时，不得不在某种程度上牺牲单个 DSP 算法的最优化性能。例如，尽管根据速度及复杂性，基于 IIR 格形结构设计不如它的直接形式的性能好，但它具有较好的灵活性，适合于基于 CORDIC 的设计。即单个算法的相对低效并非问题的关键，关键的是能够在同一架构中处理这么多不同的 DSP 算法。

另外，在进行功能仿真时，出现过各模块功能单独仿真都正确，但全系统仿真结果却错误的情况，作者采用逐步增加模块，渐近仿真的方法，最后定位错误编码，虽然更改起来很容易，但发现问题的时间却耗了很多，从中也体会到功能仿真的重要性，可以及早发现问题，尽量避免模块的错误向更大的模块系统传播，毕竟在越是复杂的系统中发现和纠正错误的代价越大。如果整个系统完成以后再发现错，改正起来代价就更大了，而且耗时、费力，错误也不容易定位，除此之外器件在最大/最小延迟情况下的工作是否正常也要被仿真，并调整输入的信号边沿。

在综合过程中要根据设计目的和要求合理的添加约束条件。而且布局布线的时候可以加入布局布线的编译约束，这些约束分为全局和局部约束。在全局约束中，可以由设计者选择布局布线的策略、系统映射方式、进位链接方法，以及系统最大最小工作频率等。在局部约束中，最主要的约束是系统引脚的定位。在进行引脚定位时，一方面要考虑到布局布线的均衡性，另一方面还要考虑到印刷电路版的布线的简单。

事实上，对于数字系统的开发而言，各个步骤都需要反复验证，发现错误要及时返回修改，有时的设计甚至需要全部从头来过，经过这种反复修改验证再修改再验证的过程才能逐步完善和优化系统，最终才能达到设计要求。

第六章 结束语

本文在对 DSP 技术、可重构计算技术及其硬件逻辑结构进行剖析的基础上，分析了几种可重构 DSP 系统结构及其优劣性，在此基础上给出本文所设计的可重构 DSP 计算引擎的基本结构框图。之后，从相关的 DSP 算法及功能的研究出发，首先找出了两类适合于 FPGA 实现的算法，即分时复用的算法和基于卷积运算的 DSP 算法。然而现有的基于卷积的 DSP 算法的实现往往是基于 MAC 方式的，于是，从改进并行性及降低占用面积角度考虑，引入 CORDIC 理论，提出以移位加方式实现这类 DSP 算法的方法。在对新的 DSP 算法实现方式的研究中，找出 FIR/IIR/DT 这几种算法具有的共同部分，在此基础上首先设计出一个统一的可编程计算模块，并对不同的算法配置不同的模块间互联网络结构，最终设计出一个可重构的 DSP 计算引擎系统结构，达到了在同一系统结构中实现不同的 DSP 算法的研究目的。

本文在完成系统结构设计的同时，以 FIR 滤波及 DFT 变换为例验证该设计的正确性，对系统的综合与实现工作了介绍。通过本课题的研究，对数字系统设计方面积累了一定的经验，并深刻体会到系统设计的重要性。结构设计的优劣，不仅影响系统性能的好坏，也是系统能否实现的关键。

本文中所设计的可重构DSP计算引擎系统结构可用做其他的对计算要求较高的主处理器的协处理器。然而，这一结构在计算速度方面的优势并不明显，因此在以后的工作中，可以考虑将其计算速度提升一些。比如，滤波计算中的多采样滤波将可能会是一个比较好的提高计算速度的方法。另外，可以将更多的 DSP 算法加入到系统中来实现。比如 QMF 滤波以及其他的变换算法。这些问题都有待于在后面的研究中来探讨。

参考文献

[1]牟澄宇《单芯片可重构计算系统体系结构研究》西北T业大学1999

[2]谷荻隆嗣编(日)《VLSI与数字信号处理》科学出版社2003

[3]李丽.辛勤.杨乐平《基于FPGA的可重构系统的应用》CJFD收录期刊微处理机

2001年03期

[4]Ronny Ronen, Avi Mendelson. Coming Challenges in Microarchitecture and Architecture[C]. In: Proceedings of the IEEE, 2001;89(3):325-340

[5]陈波，于泠.DoS攻击原理与对策的进一步研究[J].计算机工程与应用，2001；37(10)；30-33

[6]张崇，于晓琳，邓长军.FPGA在图像处理中的应用.集成电路与元器件，2004第03期

[7]RUSSELL TESSIER AND WAYNE BURLESON, Reconfigurable Computing for Digital Signal Processing: A Survey, Journal of VLSI Signal Processing 28, 7 - 27, 2001

[8]Michael Barr, A Reconfigurable Computing Primer, Multimedia Design, 1998

[9]KATHERINE COMPTON, Reconfigurable Computing: A Survey of Systems and Software, 1998

[10]Constant Coefficient Multipliers for the XC4000E, Ken Chapman, XAPP054 December 11.1996

[11]Uwe Meyer-Baese著，刘凌，胡永生译《数字信号处理的FPGA实现》，清华大学出版社，

[12]陈国良，陈峻，《VLSI计算理论与并行算法》，中国科学技术大学出版社，1991

[13]L.B.Jackson,《Digital Filters and Signal Processing》, 2nd ed. Norwell, MA:Kluwer, 1989

[14]司马苗，周源华，基于FPGA的二维DCT变换的实现，红外与激光工程，2003年8月

[15] 谷荻隆嗣编(日)《数字滤波器与信号处理》科学出版社，2003

[16] 谷荻隆嗣编(日)《数字信号处理基础理论》科学出版社，2003

[17]Oskar Mencer, Martin Morf, Parallel, Pipelined CORDICs for Reconfigurable Computing,

[18]J.E.Volder, The CORDIC trigonometric computing technique, IEEE Trans. Electron. Comput, Sept. 1959

[19]L.B.Jackson,《Digital Filters and Signal Processing》, 2nd ed. Norwell, MA:Kluwer, 1989

[20]夏于闻，Verilog数字系统设计教程，北京航空航天大学出版社

[21]王诚，薛小刚，钟信潮.《FPGA/CPLD设计工具使用详解》.2003年6月第1版

[22]K. Aono et al., “A video digital signal processor with a vector-pipeline architecture,” IEEE J. Solid-State Circuits, vol. 27, pp. 1886 - 1893, Dec. 1992.

[23] GENEL. HAVILAND , ALA. TVSZYNSKI, A CORDIC Arithmetic Processor Chip, IEEE Trans,

[24]丁玉美，高西全，《数字信号处理（第二版)》西安电子科技大学出版社，2004

[25] K. Herrmann et al., "Architecture and VLSI implementation of a RISC core for a monolithic video signal processor," in VLSI Signal Processing VII, J. Rabaey, P. M. Chau, and J. Eldon, Eds. Piscataway, NJ: IEEE Press, pp. 368 - 377, 1994.

[26] , "Time-recursive computation and real-time parallel architectures: A framework," IEEE Trans. Signal Processing, vol. 43, pp. 2762 - 2775, Nov. 1995.

[27]J. -H. Hsiao, L. -G. Chen, T. -D. Chiueh, and C. -T. Chen, "High throughput CORDIC-based systolic array design for the Discrete Cosine Transform," IEEE Trans. Circuits Syst. Video Technol., vol. 5, pp. 218 - 225, June 1995.

[28] J. Makhoul, "Linear prediction: A tutorial review," Proc. IEEE, vol. 63, pp. 561 - 580, Apr. 1975.

[29]K. N. Ngan and W. L. Chooi, "Very low bit rate video coding using 3D subband approach," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 309 - 316, June 1994.

[30], "Algorithm-based low-power transform coding architectures," in Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, Detroit, MI, May 1995, pp. 3267 - 3270.

[31] L. Tolhuizen, H. Hollmann, and A. Kalker. On the realizability of bi-orthogonal m-dimensional 2-band _lter banks. IEEE Trans. on Signal Proc., 43:640{648, 1995.

[32] T. S. Chihara, An Introduction to Orthogonal Polynomials. New York Gordon and Breach Science F' ub., 1978.

[33] C. T. Chiu, R. K. Kolagolta, K. J. R. Liu, and J. F. Jaja, "VLSI implementation of real-time parallel DCT/DST lattice structures for video communications," in VLSZ Signal Processing, V, Kung Yao ef al., Eds. New York IEEE Pr., 1992, pp. 101-110.

[34] C. T. Chiu and K. J. R. Liu, "Real-time parallel and fully pipelined twodimentional DCT lattice structures with application to HDTV systems," IEEE Trans. Circuits Syst. [Video Technol.], vol. 2, no. 1, pp. 25-31, Mar. 1992.

[35] J. A. Beraldin and W. Steenhart, "Overflow analysis of a fixed-point implementation of the Goertzel algorithm," IEEE Trans. Circuits Syst., vol. 36, pp. 322-324, 1989.

[36] J. A. Beraldin and W. Steenhart, "Time-recursive computation and real-time parallel architectures, with application on the modulated lapped transform," in Proc. SPIE, Znt. Symp. Opt. Appl. Sei. Eng., Advanced Signal Proc

[37] L. Tolhuizen, H. Hollmann, and A. Kalker. On the realizability of bi-orthogonal m-dimensional 2-band _lter banks. IEEE Trans. on Signal Proc., 43:640{648, 1995.

[38] M. Holschneider and U. Pinkall. Quadratic mirror _lters and loop groups. Technical report, TU-Berlin, 1993.

攻读学位期间的研究成果

[1] 张华, 于忠清. 《信息可视化分类及研究》. CIDE' 2004.

[2] 张华,李道煜,于忠清,董强. 《可重构硬件实现 K-means 聚类算法研究》 CIAC' 2005,已录用.

[3] 董强,李道煜,于忠清, 张华. 《Wishbone 总线的研究与实现》 CIAC' 2005,已录用.

致　谢

值此论文完成之际，我首先向我的导师于忠清教授致以衷心的感谢和由衷的敬意。三年来，于老师一直给予我严格的要求和精心的指导，本文的选题、研究工作和撰写都凝聚了于老师的许多心血。导师渊博的知识、严谨的治学态度使我受益匪浅；导师的睿智、宽容、豁达、谦逊和正直是我终生追求的目标。

我深深感谢德高望重的邵峰晶博士，她的严谨的科学态度和务实的工作作风使我受益匪浅。

感谢海尔青大软件有限公司及李道煜等诸位老师的顶力支持与帮助。

感谢郭振波、赵志刚等老师三年来给予的关心和帮助。

感谢周奕辛、董强、陆华奇等同学在工作学习中给于的支持与帮助。

感谢我的家人及朋友，他们始终在默默的支持我，我永远感谢他们。

另外，在作者硕士学习期间还得到了许多老师、同学、朋友在生活和学习上给予的帮助，在此一并真诚感谢在我成长道路上倾注心血、给予关心和帮助的所有老师、同学和朋友们。

学位论文独创性声明

本人声明，所呈交的学位论文系本人在导师指导下独立完成的研究成果。文中依法引用他人的成果，均已做出明确标注或得到许可。论文内容未包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。

本人如违反上述声明，愿意承担由此引发的一切责任和后果。

论文作者签名：

日期：　　年　　月　　日

学位论文知识产权权属声明

本人在导师指导下所完成的学位论文及相关的职务作品，知识产权归属学校。学校享有以任何方式发表、复制、公开阅览、借阅以及申请专利等权利。本人离校后发表或使用学位论文或与该论文直接相关的学术论文或成果时，署名单位仍然为青岛大学。

本学位论文属于：

保密 □，在　　年解密后适用于本声明。

不保密□。

（请在以上方框内打“√” ）

论文作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日