

代号 10701 学号 0922121557

分类号 TP39 密级 公开

西安电子科技大学

硕士学位论文



题(中、英文)目 基于FPGA的1024点流水线结构FFT

算法的研究与实现

Research and Implementation of 1024-point Pipeline FFT

Algorithm Based on FPGA

作者姓名 赵国亮 指导教师姓名、职称 周端 教授

学科门类 工学 学科、专业 计算机应用技术

提交论文日期 二〇一一年六月

西安电子科技大学
学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名： 赵周亮

日期 2011.6.3

西安电子科技大学
关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名为西安电子科技大学。

(保密的论文在解密后遵守此规定)

本人签名： 赵周亮

日期 2011.6.3

导师签名： 周扬

日期 2011.6.3

摘要

随着数字电子技术的发展,数字信号处理技术已广泛应用于通信、图像处理、雷达和多媒体等领域。作为数字信号处理的核心技术之一,快速傅里叶变换(FFT)使离散傅里叶变换(DFT)的运算时间缩短了几个数量级,使得数字信号处理的实现和应用变得更加容易。因此对 FFT 算法及其实现方法的研究具有很强的理论和现实意义。

本文针对 FFT 算法的结构和特点,设计出一种优化的基于现场可编程门阵列(FPGA)的 1024 点 16bits 复数定点 FFT 实现方案。在研究了基-2 频域抽取 FFT 算法的理论和运算规律的基础上,结合蝶形运算流程图讨论了实现 FFT 算法的几种不同的硬件结构,综合系统性能和资源消耗选取了流水线结构实现算法。

采用自顶向下的分层设计方法完成了 FFT 算法的整体结构设计,将算法分为多个功能模块。在基于 XILINX Virtex-5 系列的 FPGA 硬件验证平台上,采用 Verilog HDL 硬件描述语言实现了算法的各个功能模块,并通过 ISE 集成开发环境进行了综合验证,得到了相应的 RTL 级硬件电路。

经过 ISE 软件综合后,算法的时钟频率达到了 250MHz 以上,完成一次完整的 FFT 运算需要 2073 个时钟周期,能够满足实时信号处理的速度要求。为了验证算法的精确度,以 MATLAB 软件的 fft 函数为基础设计了算法验证程序。验证结果表明本文算法达到了一定的精确度,能够满足一般信号处理的精确度要求。论文工作对基于 FPGA 的算法硬件实现的研究具有一定的参考价值。

关键词: 现场可编程门阵列 快速傅里叶变换 蝶形运算 流水线结构

Abstract

With the development of digital electronic technology, digital signal processing technology has been widely applied in various fields, like communications, image processing, radar and multimedia. As the core technology of digital signal processing, Fast Fourier Transform(FFT) reduces the computation time of Discrete Fourier Transform(DFT) by several orders of magnitude, which makes digital signal processing much easier to implement and apply. So it is of great theoretical and practical significance to research FFT algorithm and its implement.

Based on the structure and the operational characteristics of FFT, this paper proposes a complete implementation method of 1024-point, 16-bit fixed-point complex FFT on the basis of field programmable gate array(FPGA). First, it discusses the theoretical foundation and operational rules of Radix-2 Decimation-In-Frequency FFT algorithm. Then it discusses several different hardware architectures with butterfly flow graph of FFT algorithm. At last, it ascertains the pipeline structure to implement the design considering system performance and resource consumption.

The structure of the algorithm is designed according to the hierarchical top-down design method, and it is divided into several functional modules. These modules are implemented in hard description language Verilog HDL based on the verification platform of XILINX Virtex-5 FPGA. All functional modules are synthesized on the platform of ISE integrated development environment and we get corresponding RTL hardware circuits.

After the synthesis by ISE, the maximum frequency of the algorithm reaches 250MHz. It costs 2073 clock cycles for one complete FFT processing, which satisfies the speed requirement of real-time signal processing. The code used to verify the accuracy of the algorithm is fulfilled based on the fft function provided by MATLAB. Experiment results show that the designed algorithm achieves a certain accuracy that satisfies the request of common signal processing. The work done in the paper is valuable for FPGA-based hardware implement of algorithms in the future research.

Keyword: Field Programmable Gate Array (FPGA) Fast Fourier Transform (FFT) butterfly Pipeline Structure

目录

第一章 绪论	1
1.1 课题的研究背景	1
1.2 相关技术的研究和发展现状	2
1.3 论文的研究内容	3
第二章 基于 FPGA 的 FFT 算法分析	5
2.1 DFT 的计算方法.....	5
2.2 基-2 频域抽取算法的基本原理	6
2.3 基-2 频域抽取算法的运算规律	8
2.4 FPGA 技术	9
2.4.1 FPGA 的基本原理	9
2.4.2 FPGA 的基本结构	10
第三章 FFT 算法方案设计	13
3.1 FPGA 设计方法	13
3.1.1 自顶向下设计理念	13
3.1.2 FPGA 设计开发流程	14
3.2 FFT 算法硬件实现结构	16
3.3 FFT 算法数据格式	18
3.4 流水线结构	19
第四章 FFT 算法的 FPGA 实现	21
4.1 软硬件开发平台	21
4.2 FFT 算法总体设计	23
4.3 功能模块设计	25
4.3.1 流水线结构设计	25
4.3.2 FIFO 的设计	26
4.3.3 旋转因子单元设计	27
4.3.4 复数乘法单元设计	27
4.4 功能模块协同工作流程	29
4.5 溢出控制机制	32

4.6 FFT 算法的设计特点	33
第五章 FFT 算法的测试与验证.....	35
5.1 FFT 算法性能分析	35
5.2 FFT 算法仿真验证	38
5.2.1 实信号仿真.....	40
5.2.2 复信号仿真.....	43
5.2.3 仿真结果分析.....	46
第六章 总结与展望	47
致谢.....	49
参考文献.....	51

第一章 绪论

本章主要介绍课题的研究背景,分析当前相关技术的研究发展现状,阐述硬件实现 FFT 算法的三种主要方式,并指出用 FPGA 实现 FFT 相对于其它两种方式的优越性,最后给出论文的主要工作及章节安排。

1.1 课题的研究背景

20 世纪 60 年代以来,随着数字技术、计算机技术和微电子集成电路技术的迅猛发展,复杂信号处理的实现成为可能,将信号处理的发展推向了高潮,数字信号处理技术现已深入到各个学科领域。

离散傅里叶变换(Discrete Fourier Transform, DFT)可以使数字信号处理中的频域采样按照数字运算的方法进行,在各种信号处理中起着核心作用。由于直接计算 DFT 的计算量与变换区间长度 N 的平方成正比,导致计算量偏大,因此直接使用 DFT 算法进行信号的实时处理是不切实际的。快速傅里叶变换(Fast Fourier Transform, FFT)通过将长序列的 DFT 分解为短序列的 DFT 进行计算,从而使运算量大大减少,把 DFT 的运算效率提高了 1~2 个数量级,为数字信号处理技术应用用于各种信号的实时处理创造了良好的条件,大大推动了数字信号处理技术的发展^[1]。因此对 FFT 及其实现方式方法的研究很有意义。

FFT 已广泛应用于无线通信、语音识别、雷达处理、图像处理和频谱分析等领域。在不同的应用场合,需要不同性能的 FFT 处理器。在很多应用领域都要求 FFT 处理器具有高速度、高精度、大容量和实时处理的性能^[2],如何更快速、更灵活地实现 FFT 变得越来越重要。

近几年,随着现场可编程门阵列(Field Programmable Gate Array, FPGA)技术的迅速发展,利用高并行度、高速的 FPGA 芯片来实现 FFT 已成为必然趋势。FPGA 具有在线可编程能力,硬件可重构特点,有丰富的输入/输出管脚,大容量的逻辑单元,内置嵌入式 RAM 资源,嵌入多个硬件乘法器,适用于算法固定、运算量大的前端数字信号处理。研发基于 FPGA 的 FFT 算法,充分利用 FPGA 芯片设计的并行性、灵活性、高速性,实现并行算法与硬件结构的优化配置,提高 FFT 处理速度,满足现代信号处理对高速度、高可靠性的要求,是数字信号处理的一个研究热点。因此本文将基于 FPGA 的 FFT 算法的设计与实现作为研究课题。

1.2 相关技术的研究和发展现状

傅里叶变换的理论在一百多年前就已经提出, FFT 快速算法自美国库里(J.W.Cooley)和图基(J.W.Tukey)提出到现在也已有四十多年的历史,其理论已经非常成熟。其实现方法主要有两个方向^[3]:针对 N 等于 2 的整数次幂的算法,代表算法如 Cooley-Turkey 算法、实因子算法和分裂基算法及任意组合因子算法等,算法思想是利用系数的周期性、对称性和可约性,将一维 DFT 转化为容易计算的二维或多维 DFT,使长序列的 DFT 分解成短序列的 DFT,是递归型算法,可以大大减少运算量; N 不等于 2 的整数次幂的算法,如素因子算法,Winograd 算法等,利用下标映射和数论以及近现代数学的知识,去掉级间的旋转因子,减少运算量。相较而言,第二种算法的运算量少,用的乘法器少,但是控制复杂,控制单元实现起来较难,而第一种算法结构简单,但运算过程复杂。在硬件实现过程中,判断一个算法的优劣不仅要考虑计算量,还应从算法的实现的复杂性方面加以考虑。因此算法规则、控制简单的 Cooley-Turkey 算法更适合在硬件中实现。

软件实现 FFT 运算的速度慢,不能满足实时高速的系统性能要求。随着通信技术的快速发展,很多领域对高速实时运算的要求不断提高,硬件实现 FFT 算法的方法也在不断地优化。目前高速 FFT 算法的硬件实现方案主要有三种:面向 FFT 的各类专用集成电路(Application Specific Integrated Circuit, ASIC)芯片、通用可编程数字信号处理器(Digital Signal Processor, DSP)和 FPGA。

ASIC 专用芯片是为某一特定的数字系统设计、生产的集成电路,使用比较方便,运算速度快,相对于程序而言不易被破解,可靠性好,非常适合对实时性和可靠性要求较高的信号处理系统,但它设计周期长、开发费用高、风险大、纯硬件结构不灵活,不能重新组态,可编程能力有限。在产品的发展过程中,功能无法修改或改进,任何的线路改版都需要重新设计并重新制造,不太适合处理算法和参数经常改变的场合。

DSP 具有低功耗、高集成度和高性价比等特点,其实现方法具有软件设计多用性的优点。通过软件编程几乎能够适用于各种需要 FFT 运算进行信号处理的场合,灵活方便。DSP 在各领域担负起越来越重要的任务,特别是在现代信息产业,通信系统功能向硬件定制的方向发展,而 DSP 是实现这一转变的不可缺少的核心技术。如实现 1024 点复数 FFT 运算, TI 公司的 DSP62X 系列达到 66us 量级处理速度, DSP64X 系列达到 36us 量级^[4]。但是由通用 DSP 处理器构成的 FFT 处理器采用循环编码算法,存在许多冗余运算,需要大量的跳转操作,处理速度较慢,难以满足现代数字信号处理高速、大规模、实时性的要求。在进行大点数 FFT 计算时,并行算法与 DSP 处理器的寻址能力不相适应,不能有效利用数据传输的带宽和运算能力,造成硬件资源的浪费^[5]。

FPGA 综合了软件编程的灵活性和专用芯片的快速性,既避免了软件方式所带来的速度方面的限制,又可以降低开发的成本和周期。随着其成本的不断降低,FPGA 正越来越多地替代 ASIC 和 DSP 用作前端数字信号处理的运算^[6]。基于 FPGA 实现 FFT 主要有以下几个优势:

(1) FPGA 有内置的高速乘法器和加法器,非常适合于乘法和累加等重复性的数字信号处理任务。

(2) FPGA 有大容量存储器。FFT 实时处理运算需要存储大量的数据,DSP 内部一般没有大容量的存储器,只能外接,电路会更复杂和不稳定,运算速度也会下降。FPGA 则不用外接存储器便可实现 FFT 实时处理运算,其电路更简单、速度更快、集成度和可靠性也大大提高。

(3) FPGA 硬件可编程,比 DSP 更加灵活。DSP 通常需要外部接口和控制芯片配合工作,FPGA 则不需要,这样硬件更简单和小型化。

(4) FPGA 高带宽。除了一些专用引脚,FPGA 上几乎所有的引脚均可供用户使用,因此基于 FPGA 的信号处理方案具有非常高性能的 I/O 带宽。大量的 I/O 引脚和存储器可让系统在设计中获得优越的并行处理性能。

目前以 FPGA 芯片生产厂商为主的公司基于 FPGA 设计 FFT 的综合研究方面处于领先地位,而且 FPGA 芯片生产厂商对于自己生产的芯片性能很了解,能根据各自芯片的结构进行布局和布线优化,设计出的 FFT 处理器可以充分发挥芯片的性能。Xilinx 公司推出的 DSP Core 中的 FFT IP 核计算 16 位 1024 点的定点数据,在 Virtex II 系列基-2 结构、246MHz 外频时钟下,速度达到 25.49us 量级^[7]。Altera 公司 2006 年 4 月推出的 FFT IP 核,全面支持 Altera 公司的最新器件,使用此 IP 核计算 16 位 1024 点数据,在 332MHz 系统时钟下仅需 6.3us^[8]。FFT IP 核通常专为可靠性要求高的应用场合设计,但使用不灵活,不利于对特殊需求进行修改。

国内外学者在用 FPGA 实现 FFT 处理器方面做了大量的工作,并取得了良好的成效。我国的 FPGA 技术起步相对较晚,但进入 21 世纪后,发展非常迅速。目前不少大学及研究所都使用 FPGA 芯片设计开发具有自主知识产权的 FFT 处理器。根据项目要求,对 FFT 的实现方法进行研究,通过选用不同的硬件结构,在综合考虑硬件特性和满足系统设计要求的前提下,实现算法与硬件的优化配置,提高 FFT 的处理速度,开发具有自主知识产权的 FFT 处理器不失为一种更好的选择。

1.3 论文的研究内容

本文主要针对 1024 点级联流水线结构基-2 FFT 算法进行了研究与实现,涉

及基-2 FFT 算法的分析, FPGA 实现 FFT 算法的方案设计, FFT 算法各单元模块的编程实现、系统仿真和测试。本论文共六章, 各章的主要内容如下:

第一章介绍课题的研究背景, 相关技术的研究发展状况, 包括 FFT 算法和 FFT 处理器的研究现状, 阐述 FPGA 实现 FFT 算法的优越性, 并给出本论文的研究内容。

第二章首先介绍 DFT 和 FFT 算法, 着重分析本课题中采用的基-2 频域抽取 FFT 算法的基本原理, 并总结得出算法的一些运算规律, 这些规律对我们实现算法有很大的帮助。然后介绍 FPGA 的基本原理, 并以 Xilinx FPGA 为例讲述其基本结构。

第三章首先讲述 FPGA 的设计方法, 包括 FPGA 开发中基本的设计理念和基于 FPGA 的系统开发的基本流程。然后对 FFT 算法实现的四种基本结构作了深入的分析, 并选取级联流水结构作为本文算法的基本结构, 介绍算法的三种数据表示形式, 并选定定点实现方案来实现算法。最后对流水线结构进行介绍。

第四章首先介绍本文算法开发中使用的软硬件开发平台和开发语言。然后按照 FPGA 开发“自顶向下”模块化分层设计的思想, 将 FFT 算法分成几个功能模块, 对每个功能模块进行介绍, 简述各个功能模块协同工作的流程, 并介绍设计中为了防止数据溢出采取的措施。

第五章为 FFT 算法的测试与验证。用 Xilinx ISE 自带的工具对算法进行实现, 分析算法的资源占用情况和速度运行情况, 并用几组典型的实信号和复信号对算法进行仿真验证。仿真结果和 MATLAB 自带的标准 fft 函数的计算结果进行对比分析, 验证设计的正确性。

第六章对整个论文进行总结和讨论, 也分析设计的 FFT 算法的不足和需要改进的地方, 提出进一步研究的方向。

第二章 基于FPGA的FFT算法分析

本章首先介绍 DFT 的定义及直接计算 DFT 带来的计算量问题, 然后详细介绍本课题中采用的基-2 频域抽取 FFT 算法的基本原理, 并总结得出算法的一些运算规律, 这些规律对我们实现算法有很大的帮助。FFT 算法的实现是基于 FPGA 的结构特点的, 最后介绍 FPGA 的基本原理并以 Xilinx FPGA 为例讲述其基本结构。

2.1 DFT的计算方法

快速傅里叶变换是本论文设计的基础, 要研究基于 FPGA 的高性能 FFT 算法的实现, 首先要了解快速傅里叶变换算法的基本原理。FFT 算法并不是新的理论算法, 它只是 DFT 的一种快速算法, 根据 DFT 的奇、偶、虚、实等特性对其进行改进, 因此它是以 DFT 为基础的。

根据定义, 对于长度为 N 的序列 $x(n)$, 其 DFT 可表示为

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad k = 0, 1, \dots, N-1 \quad \text{式(2-1)}$$

其反变换 (IDFT) 为

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)W_N^{-nk}, \quad n = 0, 1, \dots, N-1 \quad \text{式(2-2)}$$

二者的差别只在于 W_N 的指数符号不同, 以及差一个常数因子 $1/N$ 。

$x(n)$ 和 W_N^{nk} 一般都是复数, 对某一个 k 值, 直接按式(2-1)计算 $X(k)$ 值需要 N 次复数乘法、 $(N-1)$ 次复数加法。因此对所有 N 个 k 值共需要 N^2 次复数乘法和 $N(N-1)$ 次复数加法运算。复数运算实际上是由实数运算完成的, 式(2-1)可写成

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{nk} = \sum_{n=0}^{N-1} \{\text{Re}[x(n)] + j\text{Im}[x(n)]\} \{\text{Re}[W_N^{nk}] + j\text{Im}[W_N^{nk}]\} \\ &= \sum_{n=0}^{N-1} \{\text{Re}[x(n)]\text{Re}[W_N^{nk}] - \text{Im}[x(n)]\text{Im}[W_N^{nk}] + j(\text{Re}[x(n)]\text{Im}[W_N^{nk}] + \text{Im}[x(n)]\text{Re}[W_N^{nk}])\} \end{aligned}$$

由上式可知, 一次复数乘法需要四次实数乘法和二次实数加法; 一次复数加法需要二次实数加法。因而每运算一个 $X(k)$ 需 $4N$ 次实数乘法及 $2N + 2(N-1) = 2(2N-1)$ 次实数加法。所以整个 DFT 运算总共需要 $4N^2$ 次实数乘法和 $N \times 2(2N-1) = 2N(2N-1)$ 次实数加法^[9]。

直接计算 DFT, 乘法次数和加法次数都是和 N^2 成正比的, 当 N 较大时, 由

于实时信号处理对算法的计算速度有十分苛刻的要求,为减少运算量,提高运算速度,就必须改进算法。DFT 运算的旋转因子 W_N^{nk} 具有以下特性:

$$(1) \text{ 共轭对称性: } (W_N^{kn})^* = W_N^{-kn} = W_N^{k(N-n)}$$

$$(2) \text{ 周期性: } W_N^{kn} = W_N^{k(N+n)} = W_N^{n(k+N)}$$

$$(3) \text{ 可约性: } W_N^{kn} = W_{mN}^{kn}, W_N^{kn} = W_{N/m}^{kn/m}$$

利用这些特性, DFT 运算中的有些项可以合并, 可以将长序列的 DFT 分解为短序列的 DFT, FFT 算法的思想就是基于此, 把长序列分解成几个短序列, 这些短序列的 DFT 可重新组合成原序列的 DFT, 而总的运算次数比直接 DFT 运算少得多, 从而达到提高运算速度的目的。通过第一章 FFT 算法的研究现状综述可知, Cooley-Turkey 算法是最适合用硬件实现的。这种算法多种多样: 按数据组合方式可分为时域抽取法 FFT (Decimation-In-Time FFT, 简称 DIT-FFT) 和频域抽取法 FFT (Decimation-In-Frequency FFT, 简称 DIF-FFT); 按数据抽取的不同又可以分为基-2、基-4、基-8 以及任意因子 ($2^n, n$ 为大于 1 的整数)。随着基数的增大, 对于点数相同的输入序列, 需要分解的级数越小, 完成一定点数的 DFT 所需要的时间也就越少。基-r 算法只能实现序列的点数是 r 的整数次幂的 FFT 运算, 因此基数越大, 所实现的点数越受限制, 基 2、基 4 算法较为常用。本课题中使用的是基-2 频域抽取 FFT 算法。下面分析其基本原理和算法规律, 以便于 FPGA 实现。

2.2 基-2 频域抽取算法的基本原理

设序列 $x(n)$ 的长度为 $N = 2^M$, M 为自然数。如果不满足这个条件, 可以人为地加上若干个零值点, 使之达到这一要求。如果在时域 $x(n)$ 分解成前后两组, 那么在频域就会使 $X(k)$ 形成奇偶抽取分组, 称为频率抽取 FFT 算法^[10]。将 $x(n)$ 前后对半分, 得到两个子序列, 则式(2-1)可以表示为:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/2-1} x(n)W_N^{kn} + \sum_{n=N/2}^{N-1} x(n)W_N^{kn} \\ &= \sum_{n=0}^{N/2-1} x(n)W_N^{kn} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{k(n+N/2)} \\ &= \sum_{n=0}^{N/2-1} [x(n) + W_N^{kN/2} x(n+N/2)]W_N^{kn} \end{aligned}$$

$$\text{式中 } W_N^{kN/2} = e^{-j\frac{2\pi N}{N}k} = e^{-jk\pi} = (-1)^k$$

按 k 的奇偶把 $X(k)$ 分解成偶数组与奇数组, 当 k 取偶数 ($k=2r, r=0, 1, \dots$),

$N/2-1$) 时,

$$X(2r) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)]W_{N/2}^{nr} \quad \text{式(2-3)}$$

当 k 取奇数 ($k=2r+1$, $r=0, 1, \dots, N/2-1$) 时,

$$X(2r+1) = \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)]W_{N/2}^{nr} \quad \text{式(2-4)}$$

令

$$x_1(n) = x(n) + x(n + N/2) \quad \text{式(2-5)}$$

$$x_2(n) = [x(n) - x(n + N/2)]W_N^n, 0 \leq n \leq N/2 \quad \text{式(2-6)}$$

将 $x_1(n)$ 和 $x_2(n)$ 分别代入(2-3)和(2-4)式, 可得

$$\begin{cases} X_1(k) = X(2r) = \sum_{n=0}^{N/2-1} x_1(n)W_{N/2}^{nr} \\ X_2(k) = X(2r+1) = \sum_{n=0}^{N/2-1} x_2(n)W_{N/2}^{nr} \end{cases} \quad \text{式(2-7)}$$

式(2-7)表明, $X(k)$ 按 k 值的奇偶分为两组, 偶数组是 $x_1(n)$ 的 $N/2$ 点 DFT, 奇数组则是 $x_2(n)$ 的 $N/2$ 点 DFT。蝶形运算流程图如图 2.1 所示, 图中左边两支路为输入, 中间的小圆点表示前续运算中的加、减运算, 左上支路为相加输出, 左下支路为相减输出。右边两支路是输出, 如果在某一支路上信号需要进行后续相乘运算, 则在该支路上标以箭头, 将相乘的系数标在箭头边。

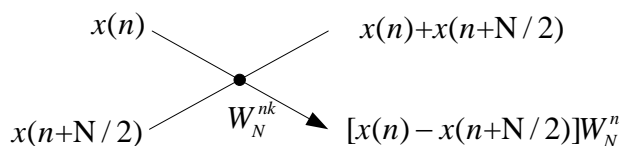


图 2.1 基-2 DIF-FFT 蝶形运算流程图符号

由图 2.1 可知, 完成一个蝶形运算, 需要一次复数乘法和两次复数加法。经过一次分解以后, 计算 N 点 DFT 共需要计算两个 $N/2$ 点 DFT 和 $N/2$ 个蝶形运算。计算一个 $N/2$ 点 DFT 需要 $(N/2)^2$ 次复数乘法和 $N/2(N/2-1)$ 次复数加法, 因此计算 N 点 DFT 共需要 $2(N/2)^2 + N/2 = N(N+1)/2 \approx N^2/2 (N \gg 1)$ 次复数加法和 $N(N/2-1) + N = N^2/2$ 次复数加法运算。因此经过一次分解就使运算量减少了近一半。

由于 $N = 2^M$, $N/2$ 仍然是偶数, 继续将 $N/2$ 点 DFT 分成偶数组和奇数组, 以此类推, 经过 $M-1$ 次分解, N 点的 DFT 最后分解为 $N/2$ 个两点 DFT。两点 DFT 就是一个基本蝶形运算流程图, 由此实现了 N 点 DFT 的频域抽取基-2 FFT 算法。为了便于理解, 一个完整的 8 点基-2 DIF-FFT 运算流程图如图 2.2 所示。由图 2.2

可知, 此运算采用的结构是顺序输入, 逆序输出。

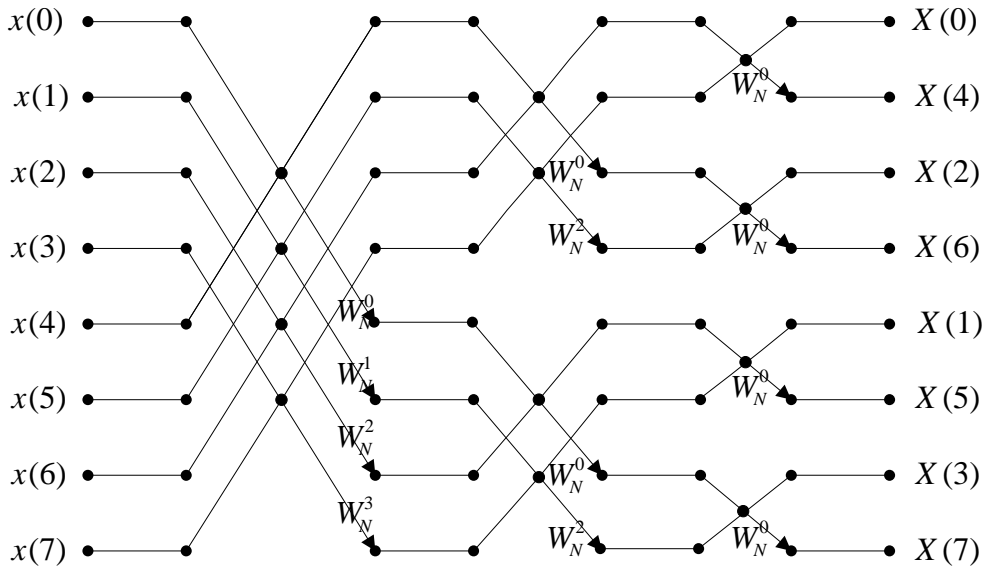


图 2.2 8 点 DIF-FFT 运算流图

由 DIF-FFT 算法的分解过程和图 2.2 可见, $N = 2^M$ 时, 其运算流图有 M 级蝶形, 每一级都有 $N/2$ 个蝶形运算。因此, 每一级蝶形运算都需要 $N/2$ 次复数乘和 N 次复数加。所以 M 级运算总共需要的复数乘次数为 $\frac{N}{2} \log_2 N$, 复数加次数为 $N \log_2 N$ 。由 2.1 节介绍可知, 直接计算 N 点的 DFT 需要 N^2 次复数乘和 $N(N-1)$ 次复数加, 因此 FFT 大大地提高了 DFT 的运算效率。

2.3 基-2 频域抽取算法的运算规律

FFT 算法不仅可以大大提高 DFT 算法的效率, 在结构上还有许多规律, 为我们实现算法带来方便。

(1) 原位运算 (同址运算)

由图 2.2 可以看出, 在同一级中, 每个蝶形的两个输入数据只对计算本蝶形有用, 并且每个蝶形的输入、输出数据节点又在同一条水平线上, 因此计算完一个蝶形后, 结果可以立即存入和它相应的输入数据所占用的存储单元。如果将每级蝶形运算全部计算完以后再开始下一级的蝶形运算, 则可将每级的运算结果仍然存在原有的同一组存储器里面。原位计算可以节省大量的硬件资源, 降低成本。

(2) 旋转因子的变化规律

由上节介绍可知, N 点 DIF-FFT 运算流图中, 每级都有 $N/2$ 个蝶形。每个蝶形都要乘以旋转因子 W_N^p 。由于各级的旋转因子和循环方式都有所不同, 在设计 FFT 算法时, 应该先找出旋转因子 W_N^p 与运算级数的关系。观察图 2.2 可以发现, 第一级有四种类型的蝶形, 旋转因子为 W_N^0 , W_N^1 , W_N^2 , W_N^3 ; 第二级有两种类型

的蝶形, 旋转因子为 W_N^0 , W_N^1 ; 第三级有一种类型的蝶形, 旋转因子为 W_N^0 。推广到 $N=2^M$ 的一般情况, 用 L 表示从左到右的运算级数 ($L=1, 2, \dots, M$), 第一级有 2^{M-1} 个旋转因子, 且为 $W_N^0, W_N^1, \dots, W_N^{N/2-1}$ 。以后每级旋转因子的个数是前级第的 $1/2$, 最后一级只有一个旋转因子 W_N^0 , 第 L 级共有 2^{M-L} 个不同的旋转因子。

(3) 位倒序的规律

由前面的流程图中可以看到, 顺序输入的数据经过蝶形运算后输出的结果不是自然的顺序排列, 好像杂乱无序, 其实是有规律的, 称之为原位倒位序。要使输出结果按频率自然顺序排列, 必须经过一次整序, 这在硬件中可以通过变址很方便的实现: 将频率序号用二进制数表示, 则将其二进制位反转过来就得到了对应的二进制倒序值。如频率序号 20 用二进制表示为(10100), 它的二进制倒序值为(00101), 即 5, 则频率序号 20 在自然顺序 5 的位置上。

(4) 蝶形运算两节点的“距离”

进入蝶形运算单元的两个数据, 它们的序号距离按照所处级数的不同而规则的变化。以图 2.2 所示的 8 点基-2 算法为例, 其第一级(第一列)每个蝶形的两节点间的“距离”为 4, 第二级每个蝶形的两节点间的“距离”为 2, 第三级每个蝶形的两节点间的“距离”为 1, 由此可以推出, 对于基-2 DIF-FFT 运算, 输入为自然序列, 输出为倒位序, 其第 L 级运算, 每个蝶形运算的两节点间的“距离”为 2^{M-L} 。

2.4 FPGA 技术

FPGA 是作为 ASIC 领域中的一种半定制电路出现的, 既解决了定制电路的不足, 又克服了原有可编程器件门电路数有限的缺点。FPGA 由许多独立的可编程逻辑模块组成, 用户可通过编程将这些模块连接起来实现不同的设计, 且可以反复擦写。在修改和升级时, 不用额外地改变 PCB 电路板, 只需在计算机上修改和更新程序, 使硬件设计转变为软件开发, 缩短了系统的设计周期, 提高了实现的灵活性并降低了成本。

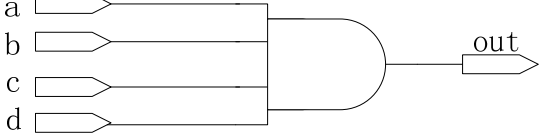
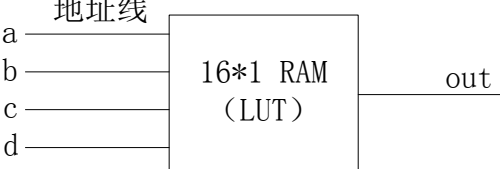
2.4.1 FPGA 的基本原理

由于 FPGA 需要被反复烧写, 它实现组合逻辑的基本结构不可能像 ASIC 那样通过固定的与非门完成, 而只能采用一种易于反复配置的结构, 目前主流 FPGA 都采用的是基于 SRAM 工艺的查找表(Look-Up-Table, LUT)。

由数字电路的基本知识可知, 对于一个 n 输入的逻辑运算, 不管是与或非运算还是异或运算, 最多只可能存在 2^n 种结果, 如果事先将相应的结果存放于一个存储单元, 就相当于实现了与非门电路的功能。FPGA 正是基于此, 通过烧写文件来配置查找表的内容, 从而在相同的电路情况下可以实现不同的逻辑功能。

LUT 本质上就是一个 RAM，对于 4 输入 LUT，可看成一个有 4 位地址线、大小为 16×1 的 RAM。用户通过原理图或 HDL 语言描述一个逻辑电路后，FPGA 开发软件（如 Xilinx 公司的 ISE、EDK 等）会自动计算逻辑电路的所有可能结果，并把真值表（结果）写入 RAM 中。这样，每当有信号输入需要进行逻辑运算时，不必再用门去搭电路，只要把输入作为一个地址进行查表，找出对应地址所存储的内容然后输出即可。表 2.1 为 4 输入与门的例子，从表中可以看到，LUT 具有和逻辑电路相同的功能。a,b,c,d 由 FPGA 芯片的管脚输入后进入可编程连线，然后作为地址线连到 LUT，LUT 中已经实现写好了所有可能的逻辑结果，通过地址查找到相应的数据然后输出，这样组合逻辑就实现了。

表 2.1 4 输入与门实现方式比较

实际逻辑电路		LUT 的实现方式	
			
a,b,c,d 输入	逻辑输出	地址	RAM 中存储的内容
0000	0	0000	0
0001	0	0001	0
.....	0	0
1111	1	1111	1

用户可以将编程数据存放在 FPGA 的片外存储器中，当系统加电时，FPGA 芯片将存储器中的编程数据读入片内 RAM 中，完成对 RAM 的配置，FPGA 进入工作状态。掉电后，FPGA 片内 RAM 的配置数据消失，逻辑关系解除，FPGA 恢复成白片，因此 FPGA 可以反复编程使用。对于同一片 FPGA，不同的编程数据，可以产生不同的电路功能，用户还可以根据不同的配置模式，采用不同的编程方式，因此 FPGA 的使用非常灵活。

2.4.2 FPGA 的基本结构

本论文中采用的硬件开发环境是 Xilinx 公司 Virtex-5 系列芯片，下面以 Xilinx 公司的 FPGA 芯片为例讲述 FPGA 的基本结构。

不同系列的 FPGA 由于其应用场合不同，内部结构会有所不同。但 FPGA 芯片一般都包含 7 个部分：可编程输入/输出单元、基本可编程逻辑单元、完整的时钟管理、嵌入式 RAM、丰富的布线资源、内嵌的底层功能单元和内嵌专用硬件模块。

(1) 可编程输入/输出单元(IOB)

芯片与外界电路的接口,完成不同电气特性下对输入/输出信号的驱动与匹配要求。为了便于管理和适应多种电气标准,IOB 被划分为若干个组(bank),每个 bank 的接口标准由其接口电压 VCCO 决定,每组都能够独立地支持不同的 I/O 标准。通过软件的灵活配置,可适配不同的电气标准与 I/O 物理特性,可以调整驱动电流的大小,改变上、下拉电阻。

(2) 可配置逻辑块(CLB)

FPGA 内的基本逻辑单元,由多个(一般为 4 个或 2 个)相同的 Slice 和附加逻辑构成。CLB 模块不仅可以实现组合和时序逻辑,还可配置为分布式 RAM 和 ROM。Slice 是基本的逻辑单元,一个 Slice 由两个 4/6 输入的查找表函数、进位逻辑、存储逻辑和函数复用器组成。

(3) 数字时钟管理模块(DCM)

Xilinx 公司的 FPGA 提供数字时钟管理和相位环路锁定。相位环路锁定能够提供精确的时钟综合,且能够降低抖动,并实现过滤功能。

(4) 嵌入式块 RAM(BRAM)

块 RAM 可配置为单端口 RAM、双端口 RAM、内容地址存储器(CAM)以及 FIFO 等常用存储结构。单片块 RAM 的位宽为 18bit、深度为 1024bit,即容量为 18Kbit,用户可以根据需要改变位宽和深度,但需要满足两个条件:修改后的容量不能大于 18Kbit;位宽最大不能超过 36bit。

(5) 丰富的布线资源

布线资源连通 FPGA 内部的所有单元。根据工艺、长度、宽度和分布位置的不同而划分为 4 个类别:第一类全局布线资源,用于芯片内部全局时钟和全局复位/置位的布线;第二类长线资源,完成芯片 bank 间的高速信号和第二全局时钟信号的布线;第三类短线资源,完成基本逻辑单元之间的逻辑互联和布线;第四类分布式的布线资源,用于专有时钟、复位等控制信号线。

布局布线器可自动根据输入逻辑网标的拓扑结构和约束条件选择布线资源连通各个模块单元,设计者不需要直接选择布线资源。

(6) 底层内嵌功能单元

主要指 DLL(Delay Locked Loop)、PLL(Phase Lock Loop)、DSP 和 CPU 等软处理核(Embedded Processor)。由于集成了丰富的内嵌功能单元,使得单片 FPGA 成为系统级的设计工具,具备了软、硬件联合设计的能力。DLL 和 PLL 可以完成时钟高精度、低抖动的倍频和分频,以及占空比调整和移相等功能。

(7) 内嵌专用硬核

内嵌专用硬核是相对底层嵌入的软核而言,等效于 ASIC 电路,主要目的是为了提高 FPGA 的性能。如集成专用乘法器提高 FPGA 的乘法速度;集成串并收发器(SERDES)来适应通信总线与接口标准,可以达到数十吉比特/秒的收发速度。

第三章 FFT算法方案设计

本章首先讲述 FPGA 的设计方法，包括 FPGA 开发中基本的设计理念和基于 FPGA 的系统开发的基本流程。然后以基-2 算法为例对 FFT 算法实现的四种基本结构作深入的分析，并选定级联的流水线结构作为本文算法的基本结构来加快 FFT 算法的处理速度。介绍三种不同的数据表示形式并选择定点运算方案实现算法，最后对流水线结构进行介绍。

3.1 FPGA设计方法

FPGA 开发过程中涉及到的软硬件知识非常多，在此只能介绍一些最基本的概念。在 FPGA 的开发过程中如果能养成良好的开发习惯，往往能够起到事半功倍的效果。

3.1.1 自顶向下设计理念

FPGA 技术发展非常迅猛，如今很多设计已经庞大到往往不是一个设计人员就能独立完成，通常需要若干设计人员组成团队来共同完成。在开始一个新的设计时，在详细地进行设计需求分析后，总设计师会根据需求将开发任务分解为若干个可操作的模块，并对其接口和资源进行评估，编制出相应的行为或结构模型，再将其分配给下一层的设计师。团队中的每一位设计人员负责其中的一个或多个模块的设计，当所有的子模块都设计完成后，再将它们联合成整个系统。

自顶向下的设计流程从系统级开始，划分为若干个二级单元，再把各个二级单元划分为下一层次的基本单元，逐级划分，直到能够使用基本模块或 IP 核实现为止，如图 3.1 所示。负责某一个模块的设计人员将模块设计完成并经测试正确无误后，就可以将它交付给上一级的设计人员。上一级的设计人员不需要了解该模块的具体实现原理，只需知道它实现的功能就可以用其开发自己的设计了。

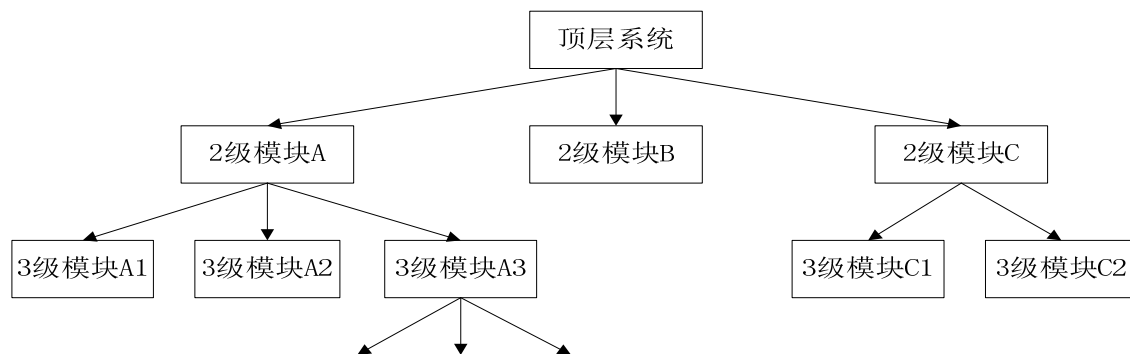


图 3.1 自顶向下的 FPGA 设计开发流程

自定向下的设计是一种层次化和模块化的设计。利用这种设计方式实现的系统更加易于维护和升级，是一种现代化的设计方法^[11]。

3.1.2 FPGA 设计开发流程

FPGA 的设计流程就是利用 EDA 开发软件和编程工具对 FPGA 芯片进行开发的过程，其开发流程一般如图 3.2 所示，包括电路功能设计、设计输入、功能仿真、综合、综合后仿真、实现与布局布线、时序仿真与验证、板级仿真与验证以及芯片编程与调试等主要步骤^[12]。

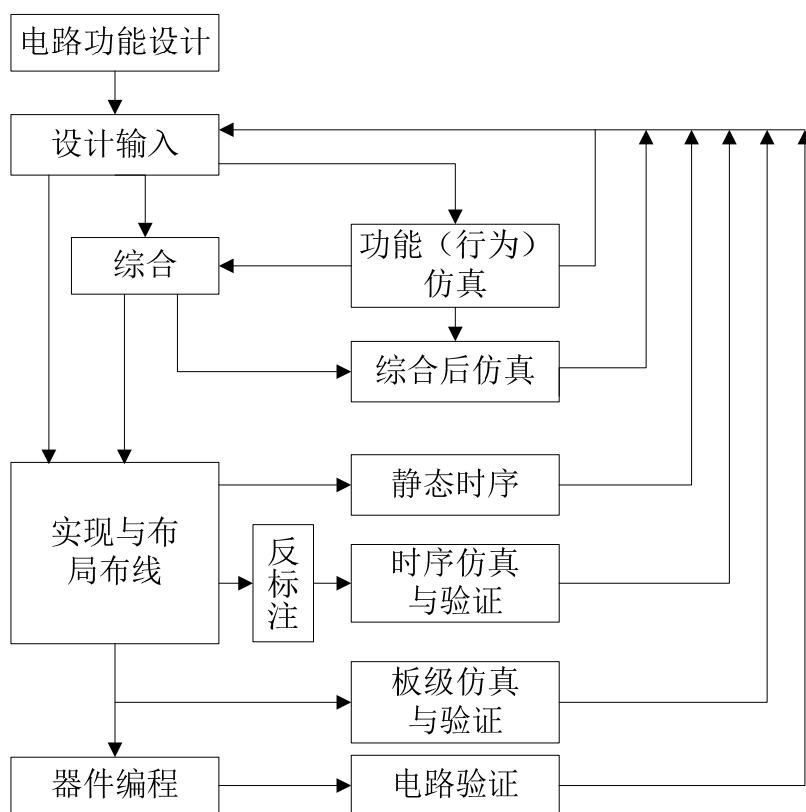


图 3.2 FPGA 开发的一般流程

(1) 电路功能设计

在开始对系统进行设计之前，必须先对整个设计有一个非常全面的认识。根据任务要求，如系统的指标和复杂度，对速度、资源和成本等进行权衡，选择合理的设计方案和合适的器件类型。FPGA 的设计是基于模块化的、自顶向下的设计，如图 3.1 所示，因此对模块的划分很重要。把系统分成若干个基本模块，每一个模块的输入输出，所要实现的功能和各个模块间的接口等都要考虑清楚，提高设计的效率。

(2) 设计输入

设计输入是将所设计的系统或电路以开发软件要求的某种形式表示出来，并输入给 EDA 工具的过程。现代数字电子系统设计中，主要采用硬件描述语言(HDL)输入法，原理图方式一般只作为辅助方式。目前常用的 HDL 语言是已经成为 IEEE

标准的 Verilog HDL 和 VHDL，其共同的突出特点是语言与芯片工艺无关，便于模块的划分和移植，具有很强的逻辑描述和仿真功能，输入效率很高，但是由于是文本设计方式，整个设计不是很直观。原理图设计方式是将需要的设计模块从集成开发环境的相应库中调出来，然后画出原理图。其优点是比较方便和直观，便于观察信号和调整电路，缺点是效率低，不便于设计的移植。通常在大型的设计中会将这两种设计方式配合起来使用，先用 HDL 实现各个模块，然后在原理图中将各个模块连接起来。

(3) 功能仿真

通常也称为前仿真。在进行综合之前对用户所设计的电路进行逻辑功能验证，看结果是否与设计意图一致。仿真没有延迟信息，因此仿真速度较快，便于发现设计前期中所存在的问题。可以通过两种方式进行功能仿真，即画波形图方式和硬件描述语言输入方式。画波形图是用波形编辑器来建立波形文件作为输入的激励，硬件描述语言输入方式是用硬件描述语言编写测试向量作为输入的激励。

(4) 综合

将较高级抽象层次的描述转化成较低层次的描述，即将设计输入编译成由与门、或门、非门、RAM、触发器等基本逻辑单元组成的逻辑连接网表。

(5) 综合后仿真

检查综合结果是否和原设计一致。仿真时把综合生成的标准延时文件反标注到综合仿真模型中，可估计门延时带来的影响。

(6) 实现与布局布线

实现是将综合生成的逻辑网表配置到具体的 FPGA 芯片上，布局布线是其中最重要的过程。布局将逻辑网表中的硬件原语和底层单元合理地配置到芯片内部的固有硬件结构上，并在速度最优和面积最优之间做出选择。布线根据布局的拓扑结构，利用 FPGA 芯片内部的各种连线资源，合理、正确地连接各个元件。布线结束后，软件工具会自动生成报告，提供有关设计中各部分资源的使用情况。

(6) 时序仿真与验证

时序仿真又称后仿真或延时仿真。将布局布线的延时信息反标注到设计网表中以检测有无时序违规（即不满足时序约束条件或器件固有的时序规则，如建立时间、保持时间等）现象。不同器件内部的延时不一样，不同的布局布线方案也给延时造成不同的影响，通过对系统和各个模块进行时序仿真，分析其时序关系，估计系统性能，以及检查和消除竞争冒险是非常有必要的。

(7) 板级仿真与验证

板级仿真主要应用在高速电路设计中，对高速系统的信号完整性、电磁干扰等特征进行分析，一般都以第三方工具进行仿真和验证。

(8) 芯片编程与调试

芯片编程是指产生使用的数据文件，然后将编程数据下载到 FPGA 芯片中。目前，主流的 FPGA 芯片生产商都提供了内嵌的在线逻辑分析仪以供调试。

在 FPGA 设计的各个环节都有不同公司提供的不同 EDA 工具。集成开发环境一般由 FPGA 厂商提供。为提高设计效率，优化设计结果，很多厂家提供了各种专业软件，来配合 FPGA 厂家提供的工具进行更高效的设计。

3.2 FFT 算法硬件实现结构

由第二章介绍的 FFT 算法可知，蝶形单元是算法的基本运算模块。根据蝶形运算单元的不同，算法可以用四种基本结构实现：递归顺序型、级联流水型、并行迭代型和阵列型。选择合适的系统结构，是提高 FFT 算法性能的关键。下面以基-2 FFT 算法为例，分别讨论这四种实现形式^[13]。

1. 递归顺序型

图 3.3 给出了递归顺序型 FFT 算法结构框图。这种形式的 FFT 只有一个蝶形运算单元，蝶形运算按递归的方式，根据蝶形图从左向右、从上向下先计算第一级的每个蝶形，然后计算第二级、第三级，逐级地循环运算，直至第 $N/2 \log_2 N$ 个蝶形，完成 N 点 FFT 的全部运算。若执行一次蝶形运算的时间为 T ，则完成 N 点 FFT 计算，所需的时间为 $N/2 \log_2 N T$ 。在实际应用中，输入缓冲单元和输出缓冲单元可以是同一个存储单元，可以采用 2.3 节介绍的原位运算规律，完成 N 点 FFT 运算最少只需要 N 个存储单元来缓存输入数据和中间计算结果。如果输入数据是连续的，那么一次 N 点 FFT 运算必须在下一组 N 点输入数据输入结束之前完成，这往往需要数倍于输入数据时钟的内部运算时钟。这种结构的优点是只有一个蝶形运算单元，所占的硬件资源少，结构简单，稳定性能好，缺点是运算速度缓慢，且时序控制较为复杂。

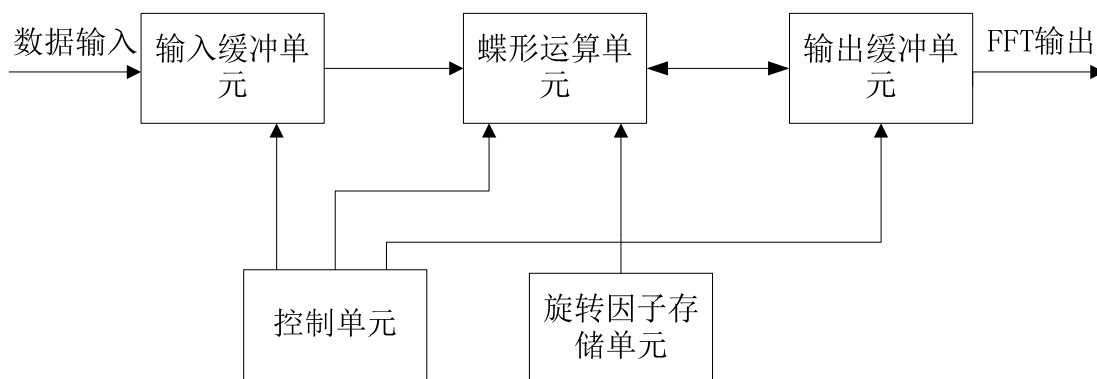


图 3.3 递归顺序型 FFT 结构框图

2. 级联流水型

一些高速数据的实时处理系统，需要在相对较短的时间内快速的完成 FFT 操

作, 级联的流水线型结构比较适合这种高速数据处理系统。流水线处理把一个重复的过程分解为若干个子过程, 每个子过程可以与其它子过程并行进行。本结构中每一级的 $N/2$ 个蝶形结都有一个独立的蝶形运算单元和存储单元, 负责本级数据的处理, 每一级的处理都相对独立, 各蝶形运算单元之间按流水方式工作, 数据流量比递归顺序型增加了 $\log_2 N$ 倍, 其结构框图如图 3.4 所示。级联流水处理的特点是:

- (1) 用 $\log_2 N$ 个运算单元并行运算;
- (2) 每个运算单元顺序执行 $N/2$ 次蝶形运算;
- (3) 每级数据运算所需的时间为 $N/2T$;

(4) 每个运算单元必须含有延时用的缓冲存储。由图 2.2 显然可见, 仅当第一个运算单元算完三对数据以后, 第二个运算单元才能开始运算。同样仅当第二个运算单元完成两对数据以后, 第三个运算单元才能开始运算。完成全部 FFT 运算所需时间还应考虑上述等待时间。

级联流水线结构的 FFT 相对于递归顺序处理结构的 FFT 增加了硬件资源的消耗, 但运算速度上提高了 $\log_2 N$ 倍, 且易于扩展。由于级联的级数决定了变换的点数, 因此通过删减级联的模块就能方便地实现不同变换点数的 FFT。

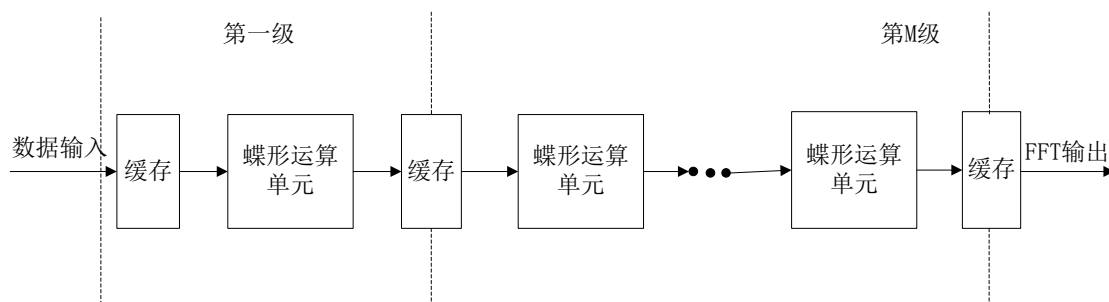


图 3.4 级联流水型 FFT 结构图

3. 并行迭代型

对每一级的 $N/2$ 个蝶形, 用 $N/2$ 个运算单元并行运算。如在图 2.2 中, 采用四个运算单元, 首先同时算第一级的四个蝶形, 然后依次计算第二级的四个蝶形和第三级的四个蝶形。每级中的蝶形运算是并行的, 但是一级到一级的迭代是顺序的, 完成 N 点 FFT 所需的时间为 $\log_2 N T$ 。运算速度很快, 但相应的硬件开销也很大, 特别是对存储器的带宽有很高要求, 并且控制单元复杂。

4. 阵列型

对 FFT 算法流图中的每个运算节点都有一个蝶形运算单元相对应, 即共有 $N/2 \log_2 N$ 个运算单元, 构成阵列, 并行实现全部蝶形运算。执行运算的时间是 T , 但由于各级蝶形运算的先后次序关系, 完成 N 点 FFT 运算的总时间仍为 $T \log_2 N$ 。采用流水线工作方式, 数据吞吐率增大了 $\log_2 N$ 倍。

从实现复杂度和资源使用量上来看, 递归顺序型最小, 级联流水型次之, 并

行迭代和阵列处理则资源占用量最大，成本最昂贵。从处理速度上看，递归顺序型最慢，级联流水型次之，并行迭代和阵列处理速度最快。蝶形运算单元的数目与 FFT 运算所需的时间成反比，即处理速度的提高是以资源使用量和成本的增加为代价的。因此在具体的设计中，必须综合考虑和适当采用上述几种形式的结构，根据实际需求来确定具体的实现方案，在运算速度、硬件开销和实现的难易程度之间进行折中。在满足系统要求的前提下，尽量少占用硬件资源，以取得“速度/成本”的最佳值。并行迭代和阵列处理实现的 FFT 算法的处理速度最快，但占用资源太多，只有在点数较少速度要求非常快的系统中才使用，当 N 点数较大时，这两种结构在 FPGA 中很难实现；递归顺序结构占用的资源少，但处理速度较慢，且时序控制复杂；级联流水结构速度比递归顺序快，且控制简单，可以通过删减级联的模块方便地实现不同变换点数的 FFT，不足之处是每级都需要各自的存储单元，增加了延时用的缓冲存储器的使用量。在本文的设计中采用级联流水型结构设计来实现 1024 点 16 位复数的 FFT 算法，处理方式速度较快，消耗的资源适中，适合采用较大规模集成电路来实现。

3.3 FFT 算法数据格式

在数字信号处理系统中所采用的二进制算法最基本的有定点制、块浮点制和浮点制^[14]，在运算过程中它们对数据位数的取位和表示形式不同，实现时对于系统资源的要求也不同，而且有着不同的适用范围。

定点数指的是在二进制中小数点的位置是固定的数，小数点的位置按照约定一般在数的最高有效位的左边或最低有效位的右边，即化为纯小数或纯整数。用定点制实现 FFT 是指所有计算过程中都是定点运算，其优点是实现结构简单，需要的存储空间小，运算速度快，但因受到字长的限制，数据的动态范围有限，舍入误差会降低最终处理结果的精度。如果在 FFT 各级运算中的截尾规则不适当，很容易出现溢出，导致输出结果错误。

浮点数据是由一纯小数和一因子组成，输入要转成纯小数和因子的浮点表示形式。浮点制运算中，无论是相乘还是相加，尾数的尾数都可能超过寄存器的长度，都要做截尾的量化处理。这种表示形式的优点是表示数的动态范围很大，可避免溢出，不需要比例因子，因而能在很大动态范围内达到很高的量化信噪比。但其运算电路复杂，需要的存储空间大，实现困难，系统造价较高，并且速度慢，在加法和乘法运算过程中还会产生舍入误差。

块浮点算法是介于浮点算法和定点算法之间的一种算法。在这种表示法中，一组数据只有一个共同的指数，这个指数是这组数中绝对值最大的数的指数。这样在对数据执行加法和乘法运算时，无需进行额外的指数操作，仅对尾数进行加

减和乘法运算即可。在运算过程中，需要在每级运算结束后进行本级运算溢出最大位数判断，以对数据块的指数进行修改，实现动态范围扩展。但由于这一组数中各个数据的大小不同，因此绝对值最大的数具有规格化的尾数，而其它数不可能刚好全部规格化^[15]。

定点运算实现简单，只要对原始数据和中间结果进行适当的处理，就可以保证数据不会溢出，并且用 16 位定点数（实部、虚部分别采用 16 位表示）处理就可以达到足够的精度，因此在本设计中，选择定点运算方案完成系统的设计。

3.4 流水线结构

流水线的设计思想是 FPGA 开发中常用的设计思想，是提高系统的工作频率的一种常用设计手段。FFT 算法的流水线结构设计源于上世纪 70 年代。其基本思想是利用流水线技术，将复杂的数字逻辑电路分级实现，这样就可以使每一级的电路结构简化，从而减少输入到输出间的电路延时，在较小的时钟周期内就能完成本级电路的功能。在下一个时钟周期到来的时候，将前一级的结果锁存为该级电路的输入，这样逐级锁存，由最后一级完成最终计算结果的输出。在这个过程中，数据就好像流过了一个数据管道，流水线技术也就由此得名。

如果某个设计的处理流程分为若干步骤，而且整个数据处理是“单流向”的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入则可以考虑采用流水线设计方法提高系统的工作频率^[16]。流水线操作的最大特点是数据流在各个步骤的处理从时间上看是连续的，如果将每一个操作步骤简化，假设为通过一个 D 触发器（就是用寄存器打一个节拍），那么流水线操作就类似一个移位寄存器组，数据流依次经过 D 触发器，完成每个步骤的操作。流水线设计时序示意图如图 3.5 所示。

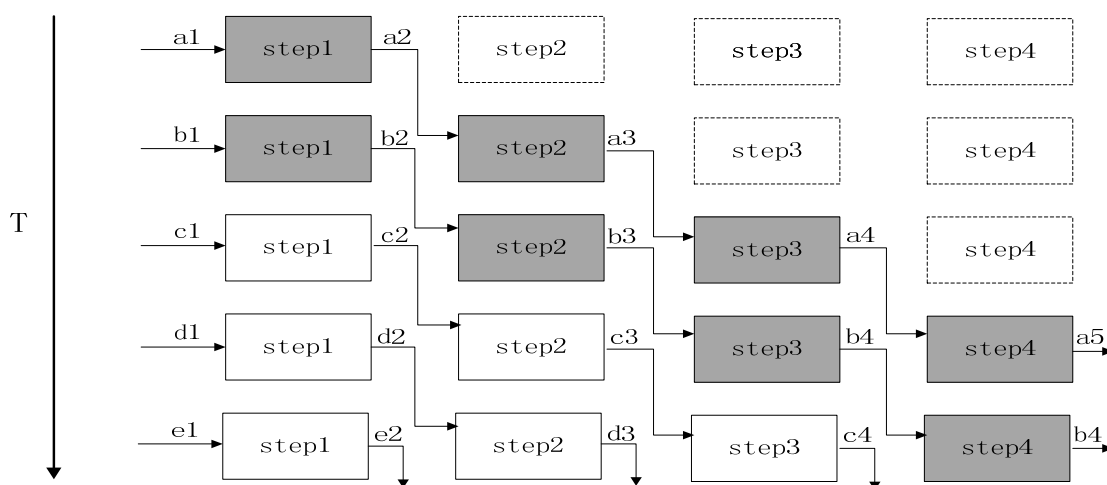


图 3.5 流水线设计时序示意图

流水线设计的关键点在于整个设计的时序安排要合理。每个操作步骤要有合理的划分,如果前级操作时间恰好等于后级的操作时间,设计最为简单,前级的输出直接连到后级的输入即可。如果前级的操作时间大于后级的操作时间,则需要对前级的输出数据进行适当的缓存,才能汇入后级的输入端。如果前级的操作时间小于后级的操作时间,则必须通过复制逻辑,将数据流分流,或者在前级对数据采用存储、后处理的方式,否则会造成后级数据溢出。

基-2 N 点 FFT 共有 $L(L=\log_2 N)$ 级蝶形运算,把每一级蝶形运算看作是流水线的一级,则本文中 1024 点基-2 FFT 算法共有十级流水。因为每级中参与蝶形运算的两个节点间有“距离”,因此每级必须包含延时用的缓冲存储器,以准备参与蝶形运算的数据。

由于流水线的处理方式相当于对处理模块进行了复制,因此通过这种方式的设计的工作频率较高,在 FPGA 的设计中使用相当普遍。

第四章 FFT算法的FPGA实现

在分析了基-2 DIT-FFT 算法基本原理、运算规律和实现方案的基础上,选择级联流水结构加快 FFT 算法的处理速度,通过插入衰减因子和符号位与最高有效位的进位情况进行溢出控制来提高 FFT 的运算精度,利用 FPGA 设计一种 1024 点 16 位复数 FFT 定点运算处理器。本章首先介绍算法开发的软硬件平台,然后根据 FPGA 的设计要求,设计 FFT 算法需要经历从整体到子系统的设计过程,即采取“自上而下”的分层设计方法并结合模块化思想,将整个设计分成多个功能模块,对各个功能模块进行介绍,并简述各模块协同工作的流程。

4.1 软硬件开发平台

(一) Xilinx Virtex-5 系列芯片

本课题中使用的硬件开发平台是 Xilinx 公司的 FPGA 开发板 ML507,它上面的 FPGA 芯片是 XC5VFX70TFFG1136。Virtex 系列是 Xilinx 的高端产品,本文中采用的 Virtex-5 FPGA 是世界上首款 65nm FPGA 产品。

Virex-5 内部包含多达 20 万的逻辑单元,等效系统门数超过 1000 万,核电压为 1V,工作频率高达 550MHz,基于创新的 ExpressFabric 架构,实现了真正的 6 输入 LUT,与以往产品相比,性能提升 30%,大约相当于两个速度等级,动态功耗降低 35%,面积缩小 45%。

Virex-5 使用 1.0V 三栅极氧化层工艺,提升了性能,降低了功耗,有利于提高系统性能(功耗越小,温度越低;半导体延迟越小,工作频率越高),降低系统设计的成本,提高可靠性。内置有用于构建更大型阵列的 FIFO 逻辑和 ECC 的增强型 36Kbit Block RAM,工作频率可以高达 550MHz,带有低功耗电路,可以关闭未使用的存储器;I/O 引脚多达 1200 个,单端 I/O 的最高速度为 800Mbit/s,差分 I/O 速度高达 1.2Gbit/s,可以实现高带宽的存储器/网络接口;低功耗收发器 24 个,可以实现 100Mbps-3.75Gbps 高速串行接口;内嵌的 DSP48E(25bit*18bit)模块(主要包含硬件乘法和加法单元,能在一个时钟内完成乘法和加法运算)的最高工作频率是 550MHz;利用内置式 PCIe 端点和以太网 MAC 模块提高面积效率;更加灵活的时钟管理管道(Clock Management Tile)结合了用于进行精确时钟相位控制与抖动滤除的新型 PLL 和用于各种时钟综合的数字时钟管理器(DCM);增强型配置电路,支持商用 flash 存储器,简化了系统重配置,降低了成本。

(二) ISE 集成开发环境

使用 Xilinx 的 FPGA 进行系统设计时, ISE 是必备的设计工具。ISE 是一个集成开发环境, 集成了大量实用工具, 包括 HDL 编辑器 HDL Editor、IP 核生成器 CORE Generator System、约束编辑器 Constraints Editor、静态时序分析工具 Static Timing Analyzer、布局规划工具 FloorPlanner、FPGA 编辑工具 FPGA Editor 和功耗分析工具 Xpower 等^[17], 这些工具都具有简便易用的内置式向导, 使得 I/O 分配、功耗分析、时序驱动设计收敛、HDL 仿真等关键步骤变得容易而直观, 帮助设计人员完成设计任务, 提高工作效率。

ISE 可以完成 FPGA 开发的全部流程, 包括设计输入、综合、仿真、布局布线、生成 BIT 文件、配置以及在线调试等, 从功能上讲, 其工作流程无须借助任何第三方 EDA 软件。

(1) 设计输入: ISE 提供的设计输入工具包括用于 HDL 代码输入和查看报告的 ISE 文本编辑器(ISE Test Editor), 用于原理图编辑的工具 ECS(Engineering Capture System), 用于生成 IP Core 的 Core Generator, 用于状态机设计的 StateCAD 以及用于约束文件编辑的 Constraint Editor 等。

(2) 综合: ISE 的综合工具不但包含了 Xilinx 自身提供的综合工具 XST, 同时还可以内嵌 Mentor Graphics 公司的 LeonardoSpectrum 和 Synplicity 公司的 Synplify, 实现无缝链接。

(3) 仿真: ISE 本身自带了一个具有图形化波形编辑功能的仿真工具 HDL Bench, 同时又提供了使用 Model Tech 公司的 Modelsim 进行仿真的接口。

(4) 实现: 此功能包括了翻译、映射、布局布线等, 有具备时序分析、管脚指定以及增量设计等高级功能。

(5) 下载: 下载功能包括了 BitGen, 用于将布局布线后的设计文件转换为位流文件; 还包括了 ImPACT, 功能是进行设备配置和通信, 控制将程序烧写到 FPGA 芯片中去。

(三) 硬件描述语言 Verilog HDL

随着电路设计规模的增加, 使用原理图输入法进行设计变得越来越困难, 并且用原理图输入法的设计的通用性和移植性也很差。因此在设计中都采用基于硬件描述语言的设计方式。本课题就是用 Verilog HDL 实现了整个设计。

Verilog HDL 是在用途最广泛的 C 语言的基础上发展起来的一种硬件描述语言, 用于数字电子系统设计, 用形式化的方法来描述数字电路, 允许设计者进行各种级别的逻辑设计, 也可以进行仿真验证、时序分析、逻辑综合操作, 是目前应用最广泛的硬件描述语言之一。设计者用 Verilog HDL 可以从上层到下层(从抽象到具体)逐层描述自己的设计思想, 用分层次的模块表示极为复杂的数字系统, 然后利用 EDA 工具逐层进行仿真验证, 再把其中需要变成具体物理电路的模块经由自动综合工具转换成门级电路网表^[18], 最后用 FPGA 的自动布局布线工

具把网表转换成具体的电路布线结构。

Verilog HDL 在 1983 年由 GDA(GateWay Design Automation)公司的 Phil Moorby 首创,之后 Phil Moorby 设计出了 Verilog-XL 仿真器和用于快速门级仿真的 XL 算法,使得 Verilog 语言得到迅速发展。基于 Verilog 的优越性,IEEE 于 1995 年制定了 Verilog HDL 标准,即 Verilog HDL1364-1995;2001 年发布了 Verilog HDL1364-2001 标准;2005 年 SystemVerilog HDL1800-2005 标准的公布,使得 Verilog HDL 语言在综合、仿真验证和模块的重用等性能方面都有大幅度的提高^[19]。

用硬件描述语言设计电路的流程与介绍的 FPGA 设计流程基本上是一致的,但是二者所强调的重点不同:基于 FPGA 的设计流程主要是从整体上,从可编程逻辑器件的范畴来说明的,而这里的侧重点在于硬件描述语言上。用硬件描述语言设计电路的主要过程可分为图 4.1 所示的几个步骤^[20]。

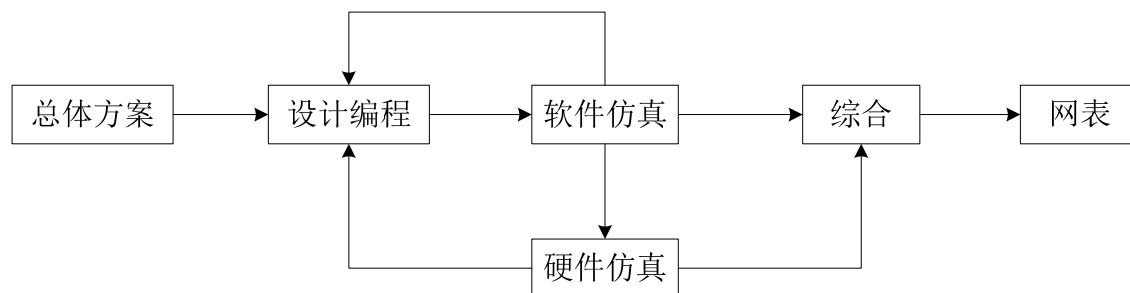


图 4.1 用硬件描述语言设计电路的主要步骤

使用 Verilog HDL 进行设计时,可以很容易地把完成的设计移植到不同厂家的不同芯片中,并在不同规模的应用时可以容易地做修改。这不仅因为用 Verilog HDL 所完成的设计的信号位数很容易改变,以来适应不同规模的应用,而且还因为采用 Verilog HDL 综合器生成的数字逻辑是一种标准的电子设计互换格式(EDIF)文件,独立于所采用的实现工艺。而相关工艺的参数描述可以包括在语言提供的属性中,利用不同厂家的布局布线工具,在不同工艺的芯片上实现。Verilog HDL 语言的最大优点是工艺无关性,因此设计者在功能设计、逻辑验证阶段时,可以不必过多考虑门级及工艺实现的具体细节,只需要利用系统设计时对芯片的要求,施加不同的约束条件,即可设计出实际电路。

4.2 FFT算法总体设计

本论文主要研究的是 1024 点的按频域抽取的基-2 FFT 算法的 FPGA 实现,为了提高算法的运行速度采用了级联流水线结构。多个蝶形单元流水线处理存在多种形式,主要有多路时延交换结构(Multi-path delay commutator, MDC)、单路时延反馈结构(single-path delay feedback, SDF)和单路时延交换结构(single-path

delay commutator, SDC)^[21]。MDC 结构可提供较高的吞吐率,但存储单元利用率低,存储单元多,硬件规模较大;SDF 结构的吞吐率相对低些,但提高了存储单元利用率,存储单元较少,硬件规模小。在流水线结构中,考虑到 FFT 算法中 75% 的功耗用于存储单元的数据读写和乘法器操作上,因此通过提高存储单元和乘法器的利用率来减少硬件单元既可以减少硬件消耗又可以减少功耗。因此本文采用单路延时反馈结构实现 FFT 算法的流水处理。SDF 结构的主要特点是 FFT 输入数据及中间运算结果均采用一路反馈单元进行存储。基于单延迟反馈的流水线不仅利用了 FFT 算法的并行性,而且很好的利用了输入数据的串行性。数据由输入端串行进入流水线,进行适当时间的缓冲后开始计算。

由第二章介绍的基-2 FFT 算法的基本原理可知,用基-2 FFT 算法基于流水线结构实现 1024 点复数的 FFT 运算,总共需要十级流水。本文设计的算法的整体架构如图 4.2 所示。由架构图可知,算法每级的基本结构几乎相同,每级包含的基本模块有进行数据缓存并为蝶形运算准备运算数据的 FIFO 单元,旋转因子产生单元,蝶形运算单元等。由基-2 DIF-FFT 基本蝶形单元运算的公式可知,蝶形运算要分两步,首先是对两个输入数据进行加减,产生一个加项和一个减项,然后减项再与相应的旋转因子相乘,本文实现时也采用两步进行,蝶形运算单元分为加减运算单元和乘法运算单元。前一级计算有计算结果输出时自动启动下一级运算,以此来实现数据的流水处理。

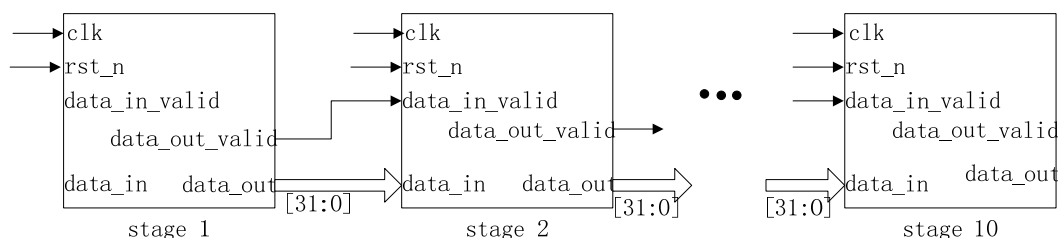


图 4.2 算法的整体架构图

设计的 FFT 算法的顶层信号如表 4.1 所示。`data_in` 是输入的要进行 FFT 变换的数据,本文设计的算法处理的数据是 16 位定点复数,输入的数据的 32 位分别为复数的实部与虚部,高 16 位是实部,低 16 位是虚部,分别用补码表示,最高位为符号位。算法运算中的中间数据和输出数据也采用相同的表示方法表示。`data_out` 是经过 FFT 运算后的输出数据。由于采用的是频域抽取法,输入数据经过各级模块处理后,输出就变为倒位序,因此在对输出结果进行分析前,应该首先根据前面介绍的倒位序规律把数据转换为正位序。`data_out_valid` 表示算法的数据输出端口已有经过 FFT 计算过的数据,高电平有效。`data_out_flag` 表示 1024 个复数的 FFT 已经计算完毕,高电平有效。

表 4.1 FFT 算法顶层信号列表

信号	属性	位宽	描述
clk	输入	1	系统时钟信号
rst_n	输入	1	系统复位信号
data_in_valid	输入	1	输入数据有效信号
data_in	输入	32	输入数据
data_out_valid	输出	1	输出数据有效信号
data_out_flag	输出	1	数据转换完毕
data_out	输出	32	输出数据

4.3 功能模块设计

4.3.1 流水线结构设计

算法的十级流水线结构如图 4.3 所示, 数据流正序从第一级输入, 经过十级蝶形单元, 在每两级间还需要乘以旋转因子, 最后在第十级倒序输出。由第二章介绍的基-2 DIF-FFT 算法的运算规律可知, 第十级蝶形运算只有一种类型的蝶形, 且要乘以的旋转因子只有一个值, 且为 W_N^0 , 复数相乘后结果不变, 因此在设计时最后一级蝶形加减运算后并没有与对应的旋转因子相乘。

在算法的各级模块中有输入使能端口和输出使能端口。当前一级有数据输出时, 把输出使能端口置为“1”, 无数据输出时置为“0”; 当输入使能端口为“1”时, 运算单元开始工作, 反之当输入使能端口为“0”时, 运算单元停止操作。这样把前一级的使能输出端口和后一级的使能输入端口相连, 就可以保证流水线间各级时序的相互配合。

一个基-2 运算出来的两个数据, 其中一个直接输出到下一级, 另外一个存入 FIFO 中, 当下一组数据进入 FIFO 时, 先前存入的数据从 FIFO 中输出, 这样就保证了数据输出的连贯性。流水线结构前一级的运算结果可以直接用于下一级运算而无需等到本级运算全部完成, 因此可以提高运算速度。

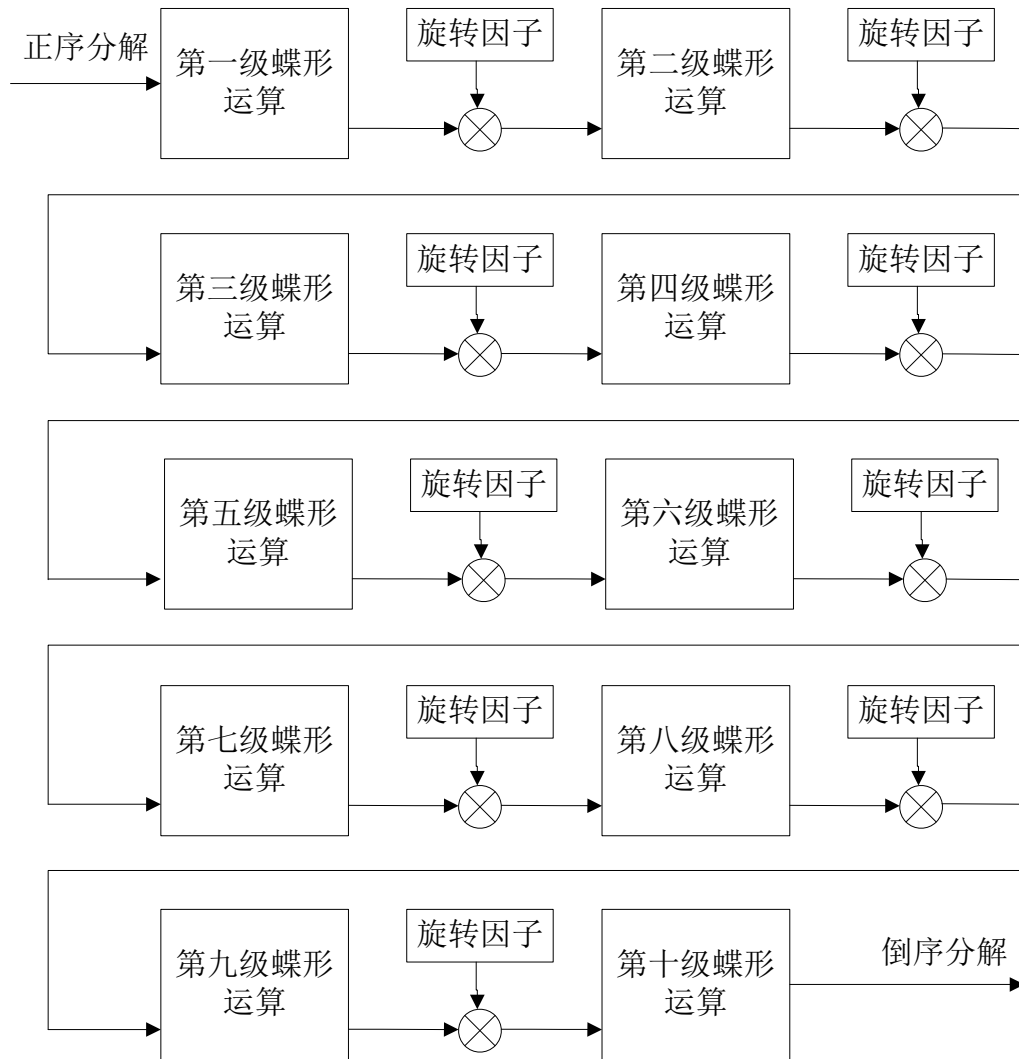


图 4.3 FFT 算法流水线整体架构

4.3.2 FIFO的设计

在本设计中,硬件实现采用单延迟反馈(SDF)。R2SDF 结构由蝶形单元和 FIFO 组成,蝶形单元完成蝶形运算;FIFO 进行数据的缓存,并为蝶形运算准备运算数据。FIFO 是一种先进先出的数据缓存器,只能顺序写入数据、顺序读出数据,其数据地址由内部读写指针自动加 1 完成。由第二章介绍的基-2 DIF-FFT 算法的蝶形运算两节点的“距离”的变化规律可知,参加第一级蝶形运算的两节点的“距离”为 512,由于参加运算的数据是串行输入,因此需要把前面的 512 个数据先存储起来,当第 513 个数据到来时,取出第一个数据,使这两个数据进入蝶形运算单元进行处理。FIFO 正好满足这种先进先出的特性,第一级的反馈长度为 512,因此设置第一级的 FIFO 大小为 512,由于参加蝶形运算的两节点的“距离”逐级递减,且后一级是前一级的 1/2,因此以后每级 FIFO 的大小为前一级 FIFO 的 1/2。

R2SDF 的蝶形单元存在两种模式,第一种模式的蝶形单元不作运算,只是将数据传输到先进先出存储器(FIFO)中然后输出,起到的仅是延迟作用。第二种模式是将从 FIFO 中输出的数据连同输入数据一起送入到蝶形单元做运算,做相加

运算的结果将被送入到下一级，做相减运算的结果将被送至 FIFO 中。两种模式交替使用直到最后输出结果。

4.3.3 旋转因子单元设计

在设计基于 SDF 的流水线结构中，每级数据在经过加减运算后都需要与对应的旋转因子相乘，因此设计了旋转因子存储单元。旋转因子单元介于每两级蝶形单元之间，它负责产生每个蝶形输出所需要的旋转因子。旋转因子 $W_N^r = e^{-j\frac{2\pi}{N}r} = \cos(\frac{2\pi}{N}r) - j\sin(\frac{2\pi}{N}r)$ ，求正弦和余弦函数值的计算量是很大的，在每级运算中直接产生将严重影响运算速度。本文采用的方法是根据 2.3 节讲的每级蝶形运算的旋转因子的变化规律，用 MATLAB 预先计算出旋转因子的值，作为旋转因子表，在程序执行中直接查找表得到所需的旋转因子，不用再计算，可节省计算旋转因子占用的大量时间，大大提高运算速度。

由旋转因子的公式可知，旋转因子的实部虚部的取值均在 $[-1,1]$ 之间，不方便定点整数运算，因此需要将计算出来的旋转因子定点化，定点化的过程是将其扩大，转换成整数。由于本文对旋转因子的实部和虚部分别采用 16 位数据存储，因此对每个旋转因子的实部和虚部分别乘以 2^{15} ，把旋转因子转换到有符号的 16 位二进制表示的十进制范围，然后转换成相应的补码。程序中对旋转因子采用 32 位数据存储，高 16 位为旋转因子的实部，低 16 位为旋转因子的虚部。

综上所述，通过直接计算旋转因子的值，并将其扩大，以便参与蝶形结中乘法的定点运算，从下节复数乘法单元的设计中可以看到，随后又将乘法器的结果缩小了 2^{15} ，这样就可以保证计算结果的正确性。

对于 1024 点 FFT 来说，采用基-2 DIT-FFT 算法，每一级需要 512 次蝶形运算，采用频域抽取，第一级共有 512 个不同的旋转因子，以后旋转因子的个数逐级递减，分别是前一级的 $1/2$ ，最后一级只有一个旋转因子。按照这样分级的方式存储旋转因子，会存储一些重复的值，但可以大大简化旋转因子地址发生及控制单元的设计。

4.3.4 复数乘法单元设计

FFT 算法中的数据在经过加减运算后要与旋转因子相乘，而两者都是复数，所以需要有一个复数乘法器。

两个复数相乘的计算过程为：假设两个复数分别为 $A=a+jb$ ， $B=c+jd$ ，那么

$$Y=A*B=(a+jb)*(c+jd)=(ac-bd)+j(ad+bc)=y_r+jy_i \quad \text{式(4-1)}$$

因此做一次复数乘法需要 4 次实数乘法和 2 次实数加法。其基本架构如下图 4.4 所示。运算时先进行四次实数乘法，然后再相加减，相加减的最终结果有可能产生溢出，因此需要对最终结果进行溢出判断。

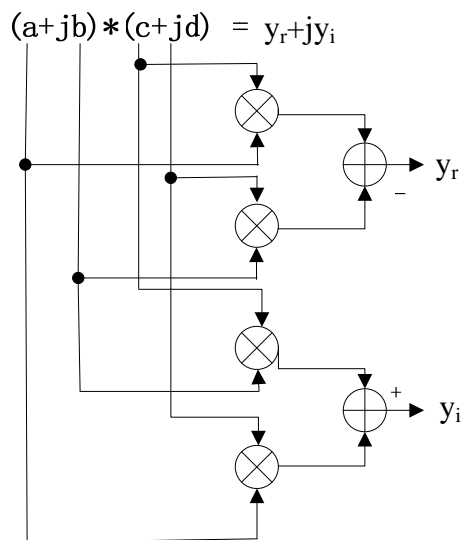


图 4.4.4 乘 2 加复数乘法器原理框图

补码定点加减运算有三种方法来判断是否产生溢出，分别是用一位符号位判断、用两位符号位判断和利用数据编码的最高位（符号位）和次高位（数值部分的最高位）的进位状况判断。使用一位符号位进行溢出判断时，需要将计算结果的符号位与原操作数的符号位进行比较，因此需要保持原操作数，而利用数据编码的最高位和次高位的进位状况判断的方法仍然是使用一位符号位，但无需原操作数，节省了判断的时间，只需计算结果比操作数多一位即可，判断方法为用符号位产生的进位与最高有效位产生的进位异或操作，若异或结果为 1，则产生溢出；异或结果为 0，则无溢出。本文中就是采用这种方法进行溢出判断。当发生正溢出时，计算结果取 16 位能表示的最大正补码数 0x7fff，当发生负溢出时，计算结果取 16 位能表示的最小负补码数 0x8000。

本文设计的 FFT 算法在处理的过程中，每一级的输入和输出都采用了同样的位宽，因此需要对数据和旋转因子相乘的结果进行截尾。程序中输入数据和旋转因子都是 16 位，相乘后字长会加倍，即相乘的结果是 32 位，需要截去低 16 位数据，以满足下一级输入对数据宽度的要求。

乘法器在 2 个时钟周期内完成一次二进制补码乘法运算，加上为了使相乘数与相应的旋转因子对其而延时的一个时钟周期，输入乘法器的两个数据经过 3 个时钟周期后输出相乘的结果。乘法器的 RTL 原理图如图 4.5 所示。

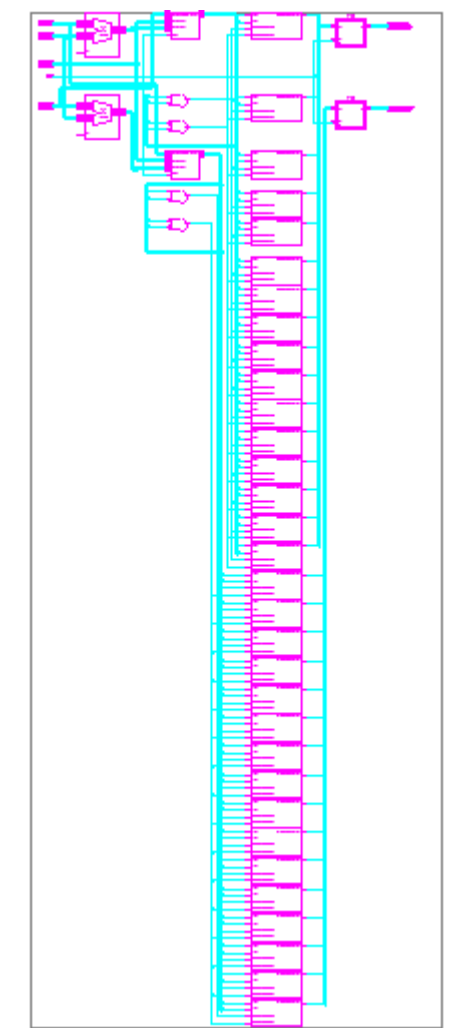


图 4.5 复数乘法单元的 RTL 原理图

4.4 功能模块协同工作流程

由前面介绍的算法的整体架构和流水线结构可知，本文设计的十级流水处理结构中，每级几乎都包含相同的处理模块，其算法处理过程也相似。下面以第一级处理结构为例，详细分析一下蝶形运算单元的工作过程，第一级蝶形运算的 RTL 原理图如图 4.6 所示。各处理模块连接的框图如图 4.7 所示，其中流动的数据均是 32 位，高 16 位为复数的实部，低 16 位为复数的虚部。

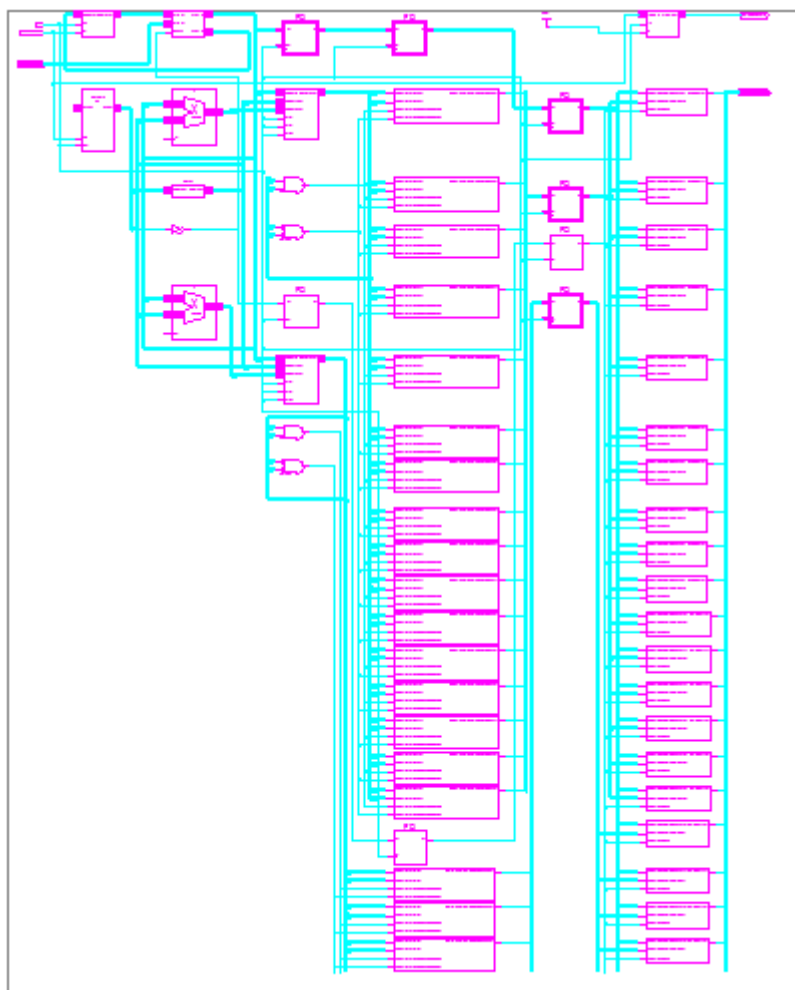


图 4.6 第一级蝶形运算的 RTL 原理图

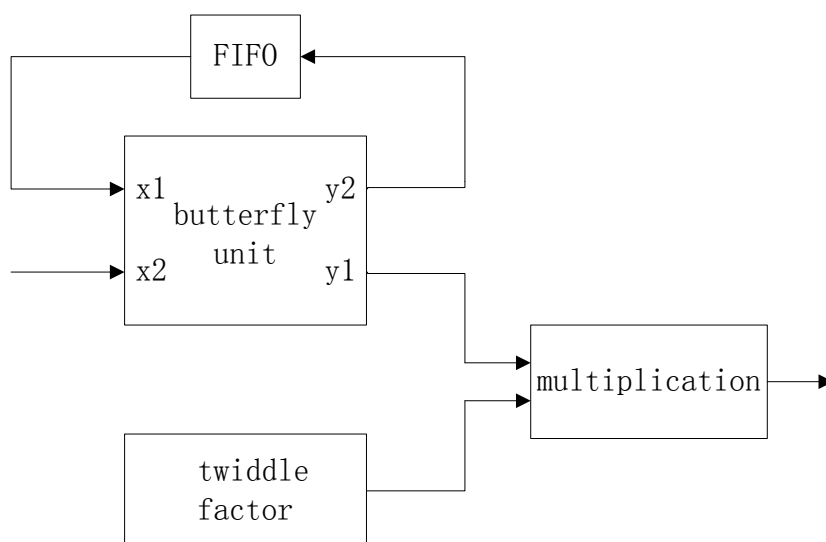


图 4.7 各处理模块的连接框图

框图中 butterfly unit 有两种功能，一是基-2 蝶形加减运算单元，二是 FIFO 缓冲的数据通道。输入为两个参与蝶形运算的节点，x2 来自上一级蝶形的运算结果（不是第一级运算时）或原始的数据输入（第一级运算时），输出为两个数据

的加减项或是 x_1 、 x_2 。由 2.3 介绍的级联流水型结构的特点可知, 每个运算单元必须含有延时用的缓冲存储, 以等待两个参与运算的数据。第一级蝶形运算两节点间的“距离”为 512, 因此需要将 x_2 输入的前 512 个数据放到 FIFO 中, 即第一级的 FIFO 的大小为 512, 当第 513 个数据从 x_2 中输入时, 第一个数据刚好从 FIFO 中输出, 参与蝶形加减运算。当用作基-2 蝶形加减运算单元时, 由基-2 频域抽取的数据流图可知, 前 $N/2$ 个相加项首先参与下一级蝶形运算, 因此 y_1 为相加项输出, 经过乘法器单元直接输出到下一级运算, y_2 为相减项输出, 连接到 FIFO 暂时缓存, 等待 $N/2$ 个时钟周期后再经过 y_1 输出。因此 FIFO 中始终有数据, 或是参与蝶形加减运算数据, 或是加减运算的相减项输出, 利用率为 100%。由以上分析可知, 在前 $N/2$ 个时钟周期 butterfly unit 作为数据通道, 后 $N/2$ 个时钟周期作为加减运算单元。butterfly unit 的 RTL 原理图如图 4.8 所示。

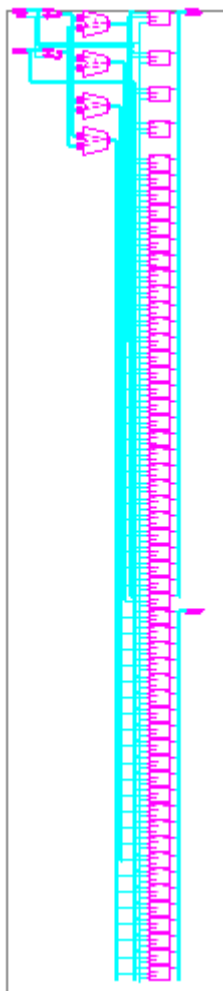


图 4.8 butterfly unit 的 RTL 原理图

twiddle factor 为旋转因子产生单元。旋转因子的值预先计算好存储起来, 根据旋转因子的读取地址输出相应的旋转因子值。

multiplication 为乘法器单元, 输入为旋转因子和蝶形加减运算单元输出的加减项, 输出为两输入相乘的结果, 结果直接输出到下一级。基-2 蝶形加减运算单

元输出的加减项中,相加项并不与旋转因子相乘,只有相减项才与旋转因子相乘。当乘法器有加减项输入时,由前面分析的 butterfly unit 输出特点可知,前 $N/2$ 个时钟周期数据是相加项,后 $N/2$ 个时钟周期数据是相减项,因此乘法器的利用率是 50%。

基-2 频域抽取 FFT 算法的空间规则性很好,算法的同步控制相对简单,一个简单的 $\log_2 N$ 位的二进制计数器就可以实现以下两个目的:一是处理整个算法单元的同步控制器,完成流水线计算流程控制;二是读取在每一级中的旋转因子地址计数。算法中通过 counter 计数器来实现。

由以上分析可知, butterfly unit 和 multiplication 都以 $N/2$ 个时钟周期为界实现不用的功能,程序中通过设置一个开关变量 bypass 来实现。

以上简略叙述了设计的算法的第一级数据的处理过程,以后九级具有类似的结构,只是参与运算的两节点间的“距离”不同,因而进行数据缓存的 FIFO 的大小不同,旋转因子的个数和产生规则也要相应改变,进行算法控制的变量做相应的调整即可。

4.5 溢出控制机制

本设计中采用定点运算方案,由于有限字长效应,如果对各中间运算结果的截尾规则使用不当,很容易出现溢出,必须要有溢出控制机制。因此在实现 FFT 算法的过程中必须采取一些措施来保证数据的精度。下面分析基-2 DIF-FFT 算法,提出设计中采用的定点溢出控制。

由 2.2 节介绍的基-2 DIF-FFT 算法的基本原理可知,对于一个蝶形运算,其第 m 级与第 $m+1$ 级的运算关系如下:

$$\begin{cases} X_{m+1}(k) = X_m(k) + X_m(j) \\ X_{m+1}(j) = (X_m(k) - X_m(j))W_N^r \end{cases} \quad \text{式(4-1)}$$

式中 $X_m(k)$, $X_m(j)$ 为第 m 级蝶形输入, $X_{m+1}(k)$, $X_{m+1}(j)$ 为第 $m+1$ 级蝶形输入,均为复数,且 $|W_N^r| \leq 1$, 因此由式(4-1)可以得到

$$\begin{cases} |X_{m+1}(k)| \leq |X_m(k)| + |X_m(j)| \\ |X_{m+1}(j)| \leq |X_m(k)| + |X_m(j)| \end{cases} \quad \text{式(4-2)}$$

$$\text{因此 } |X_{m+1}(k)| \leq 2 \max \{|X_m(k)|, |X_m(j)|\}$$

$$|X_{m+1}(j)| \leq 2 \max \{|X_m(k)|, |X_m(j)|\}$$

$$\max \{|X_{m+1}(k)|, |X_{m+1}(j)|\} \leq 2 \max \{|X_m(k)|, |X_m(j)|\} \quad \text{式(4-3)}$$

由式(4-1)还可以得到

$$\begin{cases} X_{m+1}(k) = X_m(k) + X_m(j) \\ X_{m+1}(j)W_N^{-r} = X_m(k) - X_m(j) \end{cases}$$

$$\text{所以} \begin{cases} X_m(k) = \frac{1}{2}(X_{m+1}(k) + X_{m+1}(j)W_N^{-r}) \\ X_m(j) = \frac{1}{2}(X_{m+1}(k) - X_{m+1}(j)W_N^{-r}) \end{cases} \quad \text{式(4-4)}$$

$$\text{可以推出} \begin{cases} X_m(k) \leq \frac{1}{2}(|X_{m+1}(k)| + |X_{m+1}(j)|) \leq \max\{|X_{m+1}(k)|, |X_{m+1}(j)|\} \\ X_m(j) \leq \frac{1}{2}(|X_{m+1}(k)| + |X_{m+1}(j)|) \leq \max\{|X_{m+1}(k)|, |X_{m+1}(j)|\} \end{cases} \quad \text{式(4-5)}$$

$$\text{即} \max\{|X_m(k)|, |X_m(j)|\} \leq \max\{|X_{m+1}(k)|, |X_{m+1}(j)|\} \quad \text{式(4-6)}$$

式(4-6)表明, FFT 运算中从前一级到后一级, 节点变量的最大模值是逐级非减的, 只要后一级不发生溢出, 前一级计算一定不会溢出。由式(4-3)可知, 只要前一级结果不大于 1/2, 后一级结果必定小于 1, 因此后一级就不会产生溢出。

由式(4-3)可知, 蝶形运算单元输出的最大模值小于等于输入最大模值的两倍, N 点基-2 FFT 共有 $L = \log_2 N$ 级蝶形运算, 因此 FFT 最后输出的最大值小于等于输入最大值的 $2^L = N$ 倍, 即 $\max\{X(k)\} \leq 2^L \max\{x(n)\} \leq N \max|x(n)|$ 。

设输入信号 $x(n)$ 的模值小于 1, 这可以通过对数据进行归一化实现, 只要将 $x(n)$ 衰减 N 倍, 或者将 $x(n)$ 乘以比例因子 N^{-1} , 就可以防止 FFT 运算过程中溢出的发生。但是这种防止溢出的办法, 会使输入数据的幅度被限制得过小, 运算的精度不高。

由式(4-3)与式(4-6)还可以得到一种更好的防止数据溢出的措施, 即对每一级蝶形运算的输入分别乘以衰减因子 1/2, 就可以保证蝶形运算不发生溢出。对于 $L = \log_2 N$ 级蝶形, 相当于乘以衰减因子 N^{-1} , 与前一种方法的不同之处在于把衰减因子 N^{-1} 分散到了各级运算之中, 提高了运算的精度。

因此在本设计中在数据进入基-2 蝶形单元之前, 将数据右移一位, 相当于乘以衰减因子 1/2, 这样就可以减少截断误差的引入并防止数据溢出。

4.6 FFT算法的设计特点

综合前面的介绍, 本文设计的 FFT 算法具有以下特点:

- (1) 基于基-2 DIF-FFT 实现 1024 点 16bits 复数 FFT 处理器, 运算数据顺序输入, 运算结果倒位序输出。
- (2) 输入数据和中间运算过程均采用带符号的补码表示, 方便算法内部加减和乘法运算的处理。

(3) 根据各级蝶形运算的旋转因子的变化规律, 预先计算出旋转因子的值, 并进行定点化处理, 使之方便定点整数运算, 作为旋转因子表存储, 然后按照每级运算的需求直接读取, 从而减少运算过程, 提高运算速度。

(4) 采用多个蝶形单元级联流水处理的结构, 实现了数据的连续输入和输出, 大幅增加了数据的吞吐量, 提高了算法的运行速度。

(5) 由 SDF 结构实现算法的流水处理, 使用的存储单元少, 存储单元利用率高, 并很好的利用了输入数据的串行性。

(6) 在对定点制的动态范围进行研究后, 通过插入衰减因子和符号位与最高有效位的进位情况进行溢出控制, 提高运算的精度。

第五章 FFT算法的测试与验证

采用自上而下的分层设计方法并结合模块化思想,完成了 FFT 算法的整体设计和 RTL 代码级设计后,选用 Xilinx 公司的 Virtex-5 系列 FPGA 芯片实现整个 FFT 算法,并从资源占用、速度运行方面对算法进行性能分析,最后基于 ISE 对算法进行仿真测试,并和 MATLAB 自带的标准 fft 函数的计算结果进行对比,测试结果表明设计的算法具有较高的精度。

5.1 FFT算法性能分析

上一章对 FFT 算法的各功能模块分析并用 Verilog HDL 语言设计实现后,选用 Xilinx 的“Virtex-5”器件,“XC5VFX70T”系列芯片,封装为“FF1136”,速度等级为“-1”对算法进行实现。按照 FPGA 开发的方法和步骤,首先进行语法检查(Check Syntax),无误后用 Xilinx ISE 自带的综合工具“Synthesize - XST”进行编译、分析、综合,生成综合报告文件(.syr)。经过综合后设计的 FFT 算法的 RTL 级结构如图 5.1 所示。

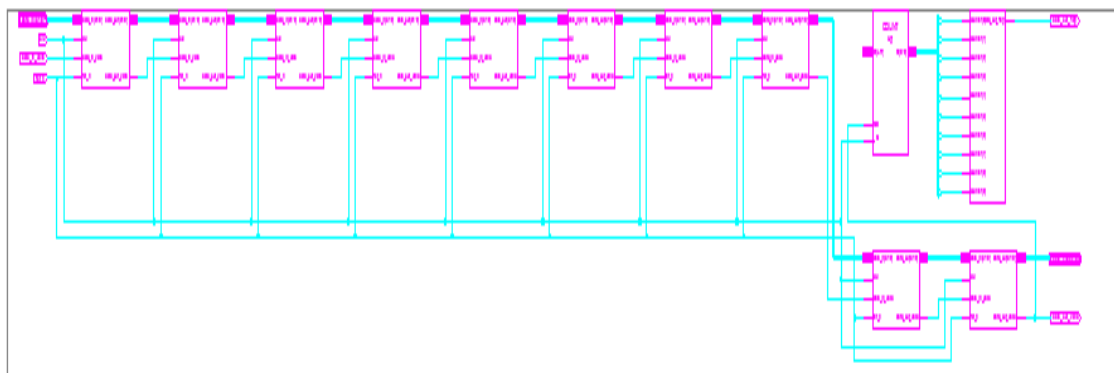


图 5.1 FFT 算法的 RTL 原理图

对 FFT 算法完成逻辑综合后,利用 ISE 自带的实现工具(Implement Design)对算法进行实现,包括翻译(Translate)、映射(Map)、布局布线(Place & Route),生成相应的报告和仿真模型。其具体步骤为首先将综合生成的网表文件(NGC)和用户约束文件(UCF)相结合,通过翻译(Translate)生成一个 NGD(Native Generic Database)文件,它以 Xilinx 基本单元的形式描述了逻辑设计。然后通过映射(Map)在选定器件上将设计适配成为可用的资源,此过程将 NGD 定义的逻辑映射到 FPGA 中,输出设计是 NCD(Native Circuit Description)文件,它在本质上表述了

Xilinx FPGA 映射到组件中的逻辑，另外还生成 PCF 文件。最后通过布局与布线 (Place & Route) 将一个映射过的 NCD 文件设计布局布线后生成一个 NCD 文件作为比特流生成的输入。布局布线后相关的资源占用情况如图 5.2 所示

1024 Project Status (05/07/2011 - 16:08:02)				
Project File:	1024.isc	Current State:	Placed and Routed	
Module Name:	FFT_R2SDF	• Errors:	No Errors	
Target Device:	xc5vfx70t-1ff1136	• Warnings:	10 Warnings	
Product Version:	ISE 10.1.03 - Foundation	• Routing Results:	All Signals Completely Routed	
Design Goal:	Balanced	• Timing Constraints:	All Constraints Met	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)	
Device Utilization Summary				[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	1,539	44,800	3%	
Number used as Flip Flops	1,539			
Number of Slice LUTs	4,008	44,800	8%	
Number used as logic	2,579	44,800	5%	
Number using O6 output only	2,546			
Number using O5 output only	29			
Number using O5 and O6	4			
Number used as Memory	1,425	13,120	10%	
Number used as Shift Register	1,425			
Number using O6 output only	1,420			
Number using O5 output only	5			
Number used as exclusive route-thru	4			
Number of route-thrus	33	89,600	1%	
Number using O6 output only	33			
Slice Logic Distribution				
Number of occupied Slices	1,379	11,200	12%	
Number of LUT Flip Flop pairs used	4,057			
Number with an unused Flip Flop	2,518	4,057	62%	
Number of fully used LUT-FF pairs	1,490	4,057	36%	
Number of unique control sets	32			
Number of slice register sites lost to control set restrictions	52	44,800	1%	
IO Utilization				
Number of bonded IOBs				
Number of bonded	69	640	10%	
Specific Feature Utilization				
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of DSP48Es	36	128	28%	

图 5.2 布局布线后 FFT 算法的资源消耗信息

由上述资源消耗报告可知，完成布局布线后，十级流水线的 FFT 算法总共使用 Virtex-5 芯片系列 xc5vfx70t-ff1136 器件的 Slice 寄存器(Slice Registers)基本单元 1539 个，约占芯片 Slice 寄存器总量的 3%；总共使用 Slice 查找表(Slice LUTs)为 4008 个，使用率为 8%；使用的乘法器单元 DSP48Es 约占乘法器总资源的 28%；总共使用 69 个 I/O 引脚，使用率为 10%。可见，本设计节省了片内 Slices 的使用数量以及乘法器资源，满足了对 FPGA 资源合理利用的系统要求。

ISE 综合后的设计总结(Design Summary)中的时序报告(Timing Report)如图

5.3 所示。

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal	Clock buffer (FF name)	Load
clk	BUFGP	3000

Asynchronous Control Signals Information:

Control Signal	Buffer (FF name)	Load
rst_n_inv(s9_inst/rst_n_inv1_INV_0:0)	NONE(counter_8)	65

Timing Summary:

Speed Grade: -1

Minimum period: 3.942ns (Maximum Frequency: 253.678MHz)
Minimum input arrival time before clock: 4.026ns
Maximum output required time after clock: 6.171ns
Maximum combinational path delay: No path found

图 5.3 FFT 算法的时序报告

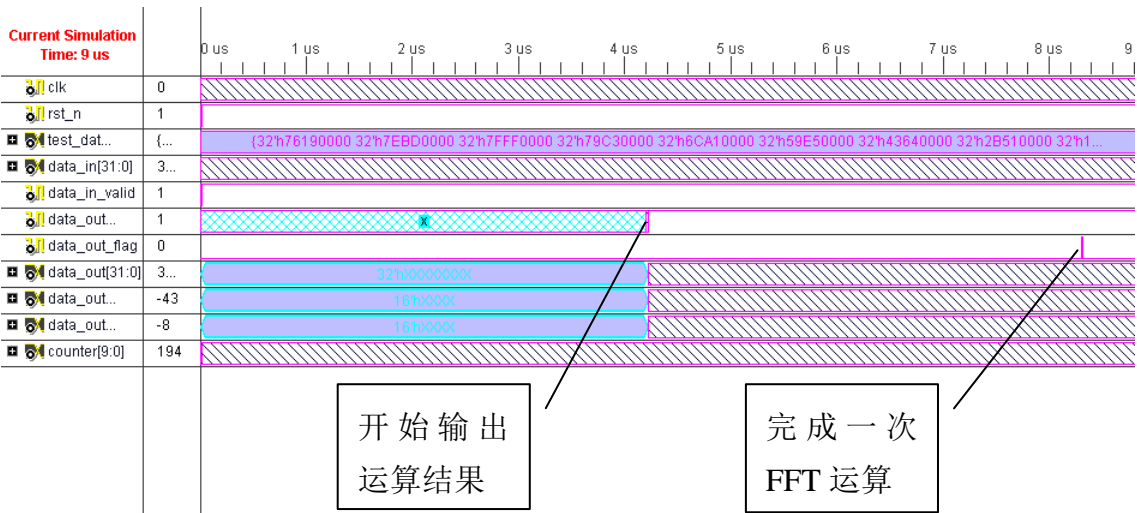


图 5.4 FFT 算法的仿真

由图 5.4 所示的仿真数据图可知，本算法从输入第一个数据到输出第一个计算结果所需要的时钟周期的个数为 1050，因为本文中的算法采用的是流水线设计，从输出第一个计算结果后可以连续输出结果，因此从输入第一个数据到输出第 1024 个计算结果(即完成一次 FFT 计算)所需的时钟周期的个数为 1050 + 1023

= 2073。由 ISE 综合出的时序报告可知, 算法的最高时钟频率为 253.678MHz, 其最小的时钟周期为 3.942ns, 当时钟周期为 4ns 时, 完成一次 FFT 运算所需的时间为 8292ns, 完全能满足实时信号处理的要求。

TI 公司的 TMS320C67, 实现 1024 点 FFT 处理系统 (主频 167MHz) 需要 120us^[22]; AD 公司的 ADSP21160 (主频 100MHz) 需要 90us^[23]; 上海交通大学芯片与系统研究中心设计的 4096 点 FFT 模块 (主频 120MHz) 需要 240.57us^[24]。相比之下, 本文设计的基于 FPGA 的 FFT 算法充分利用了其硬件的并行性和流水线结构的优点, 速度上具有很大的优势。

5.2 FFT 算法仿真验证

代码编写完成后, 需要搭建一个测试平台(Testbench)来验证所设计的模块是否满足要求。仿真验证与设计流程密切相关, 一般都是通过仿真、时序分析、下载调试等来检验设计的正确性。在 FPGA 的开发流程中, 仿真主要包括功能仿真和时序仿真两个部分。功能仿真是为了检验设计的功能是否正确, 通过功能仿真, 可以及时发现设计中的错误, 加快设计的进度, 提高设计的可靠性。时序仿真是为了保证设计满足时序要求, 使数据能被正确地采样。准确高效的仿真验证可以最大限度地避免设计失误所造成的风险。为提高数字电路设计的仿真验证效率, 可采取的主要措施有提高仿真验证工具的速度和精度, 改进仿真验证的设计方法。

Testbench 是最基本的仿真验证手段^[25], 其主要作用是: 例化待测试的设计; 通过测试向量进行仿真; 输出仿真结果。Testbench 通过读取激励数据, 添加到待测逻辑单元输出一个响应信号, 从输出的仿真结果就可知道设计电路的正确与否。Testbench 是一个具有独立结构的 Verilog 模块, 它包括测试激励的生成和监视、待测试系统和响应显示器三个部分, 其结构如图 5.5 所示。

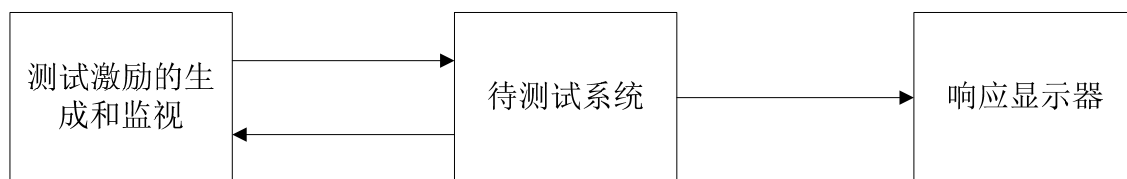


图 5.5 Testbench 的基本结构

ISE 提供了两种 Testbench 的建立方法, 一种是使用 HDL Benchner 的图形化波形编辑功能编写, 另一种是利用 HDL 语言, 相对于前者使用简单、功能强大。本文中使用第二种测试方法。使用 HDL 语言测试时, ISE 提供了辅助设计测试激励的功能, 根据设计源文件的模块名称与输入输出端口生成测试激励文件模板, 使用模板书写测试激励, 省去了许多格式化部分, 如文件头、测试激励模块名、测试激励端口信号、待测试模块的调用和部分信号的初始化等部分。

本文对整个算法所测试的向量数据庞大, 为了验证系统设计的正确性, 采用了 MATLAB 来辅助仿真测试。MATLAB 作为一种强大的数学和信号处理工具, 非常适合用作系统算法的验证。MATLAB 是 Math Works 公司于 1982 年推出的一种功能强大、效率高、交互性好的数值计算和可视化计算高级语言, 它将数值分析、矩阵运算、信号处理和图形显示有机地融合为一体, 形成了一个极其方便、用户界面友好的操作环境, 被广泛地用于研究和解决各种具体工程问题^[26]。MATLAB 有着功能强大、丰富的函数工具箱, 这是整个 MATLAB 语言得以如此快速发展的重要因素之一。其中信号处理工具箱(signal processing toolbox)主要针对数字信号处理问题而设计, 它以 MATLAB 的数值运算和图形可视化为基础, 由一组描述信号处理算法的 M 文件和 GUI 工具组成, 可以完成数字或模拟系统中常规的信号处理任务。其中, 两个重要的信号处理函数 fft 和 filter 是 MATLAB 的内建函数^[27], 非常适合作为本文设计算法的验证。

具体的测试方法为: MATLAB 产生仿真测试数据, 对数据进行预处理, 使之符合设计的 FFT 算法对输入数据的要求, 并用 MATLAB 文件操作命令将测试数据输出到一个数据文件, 在 FFT 算法的仿真文件中用 Verilog HDL 语言的文件操作系统任务读入测试数据文件, 作为仿真的激励信号, 并把仿真结果输出到一个数据文件, 最后使用 MATLAB 对产生的仿真输出文件作频域分析, 其过程如图 5.6 所示。

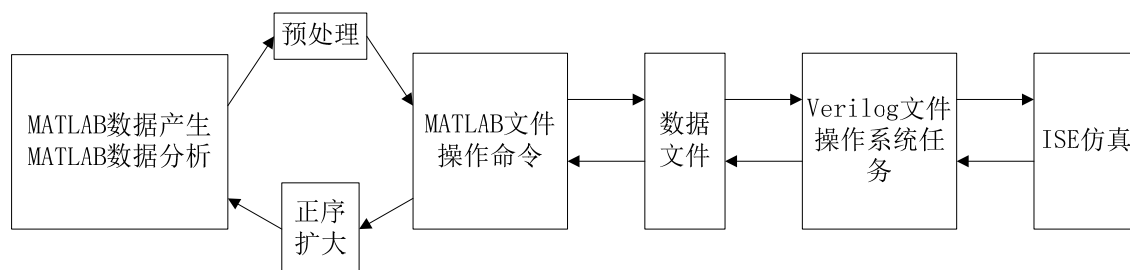


图 5.6 数据仿真测试过程

MATLAB 中定义的标准 fft 函数为测试验证提供很大方便, 调用方式为 $Y=\text{fft}(X)$ 或 $Y=\text{fft}(X,N)$, X 是要进行 fft 的数据, N 是数据长度, 这里设置 N 为 1024。把相同的测试数据代入 MATLAB 定义的标准 fft 函数中, 计算结果与设计的 FFT 算法的仿真结果进行对比, 分析误差, 验证设计的正确性。

MATLAB 产生仿真测试数据的预处理过程主要有首先对采样得到的 1024 个数据归一化处理, 使其的数据范围在 $[-1,1]$, 然后进行定点化, 设计的算法接收的处理数据是 16 位复数, 实部和虚部分别用 16 位表示, 因此对每个数据分别乘以 2^{15} , 把数据转换到有符号的 16 位二进制表示的十进制范围, 然后转换成相应的补码表示。转换成补码的过程中为了避免最大的正数 32768 与最小的负数 -32768 产生错误, 把 32768 用 32767 表示。然后把实部与虚部分别用十六进制表

示,组合成 32 位数据,实部在前,虚部在后。转换成补码便于在 FPGA 中算法的运算,MATLAB 中 fft 函数的输入数据只定点化到 16 位二进制表示的十进制范围,不用转化为补码。

设计中采用的算法是基于频域抽取,数据正序输入,FFT 结果逆序输出,而 MATLAB 中定义的标准 fft 函数是数据正序输入,结果正序输出,因此在使用 MATLAB 读取算法的仿真数据后要首先调用 MATLAB 中的 digitrevorder 函数对结果进行正序,才能与 MATLAB 中定义的标准 fft 函数的计算结果做对比分析。

为了防止蝶形运算过程中计算结果溢出,设计算法时在每级蝶形运算中插入了一个衰减因子 0.5。本文的算法共有 10 级蝶形运算,因此设计的 FFT 算法输出的计算结果比正常值缩小了 2^{10} 倍。因此在使用 MATLAB 读取算法的仿真数据后需要先将仿真数据扩大 $2^{10}=1024$ 倍,才能和 MATLAB 的 fft 函数计算出的理论值相比较。

以下分别采用几个典型的实信号和复信号进行仿真测试。

5.2.1 实信号仿真

(1) 三角波信号仿真

使用 MATLAB 产生一个三角波信号:

$$x(n) = \begin{cases} n/512 - 0.5 & 0 \leq n \leq 511 \\ 0.5 - (n - 512)/512 & 512 \leq n \leq 1023 \end{cases} \quad \text{式(5-1)}$$

把得到的测试数据按照前面介绍的方法进行预处理,完成仿真后的输出结果和 MATLAB 分析的结果如图 5.7 所示。由图可知 MATLAB 自带的标准 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线十分吻合。图 5.8 是算法的仿真结果和 MATLAB 的标准 fft 计算结果之间的误差。从图 5.7 可知信号功率主要集中在 $k=1$ 和 $k=1023$ 两个频率点。从图 5.8 的仿真结果误差可知在这两个频率点的仿真误差分别是 1445.6 和 421.69。通过计算,这两个频率点的信号量化噪声比(SQNR)分别达到 78.45dB 和 90.17dB。

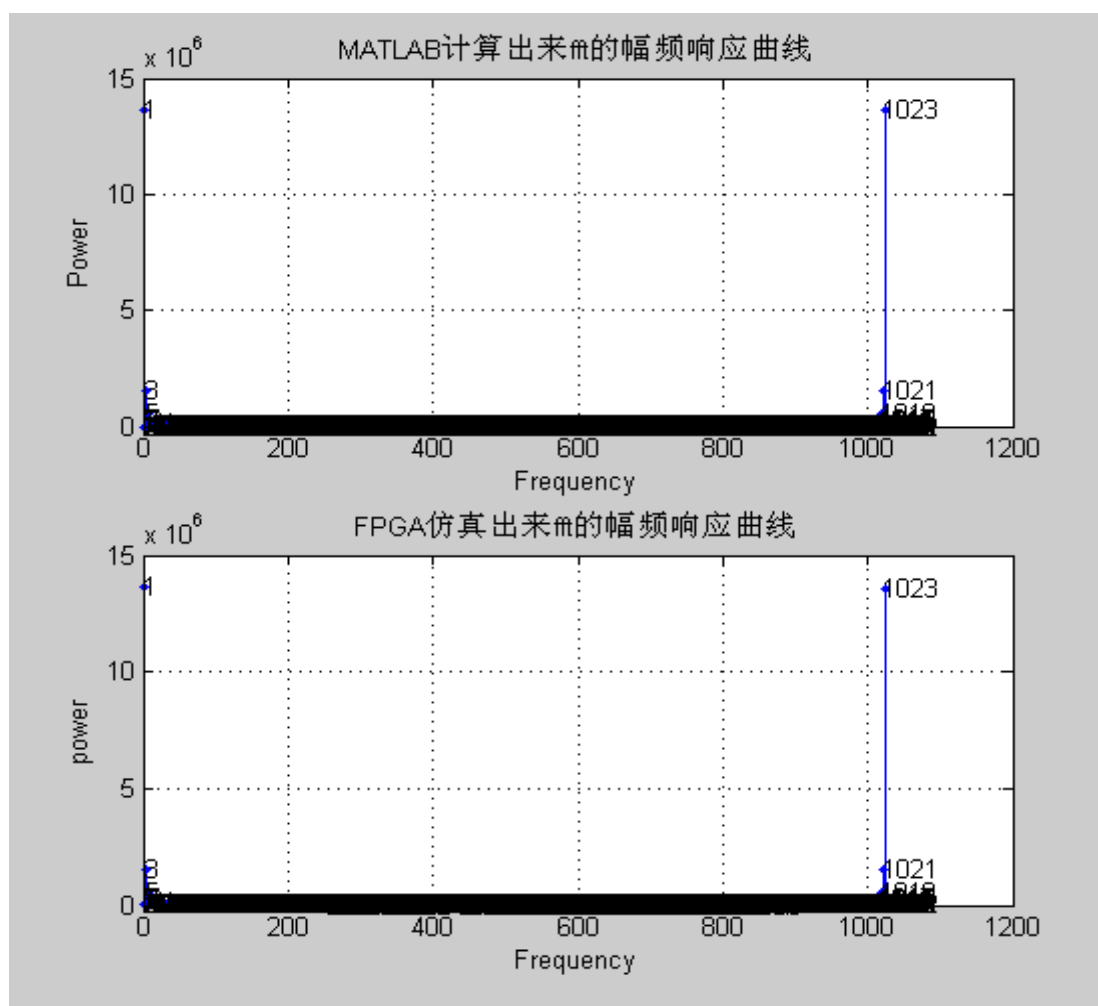


图 5.7 三角波信号频谱对比

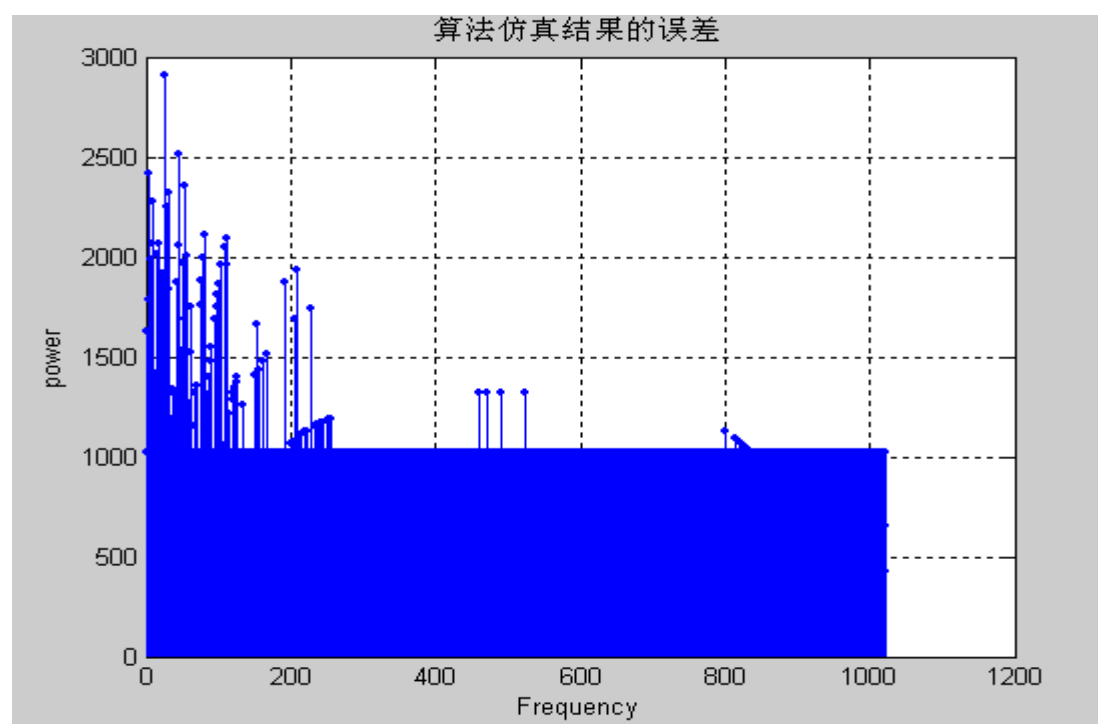


图 5.8 三角波信号算法仿真的误差

(2) 正弦信号仿真

使用 MATLAB 产生一个幅值是正弦规律变化的信号:

$$x(t)=2+3 \times \sin (2 \times \pi \times 50 \times t) \quad \text{式(5-2)}$$

采样得到测试用的离散时间信号:

$$x(n)=2+3 * \sin (2 * \pi * 50 / 500 * n) \quad n=0: 1023 \quad \text{式(5-3)}$$

由式(5-3)可知,对正弦信号的采样频率为 500Hz。产生的正弦信号带有直流分量,因此同时对直流信号进行了仿真。把测试数据按照前面介绍的方法进行预处理,完成仿真后的输出结果和 MATLAB 分析的结果如图 5.9 所示。由图可知 MATLAB 自带的标准 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线基本上是一致的。图 5.10 是算法的仿真结果和 MATLAB 的标准 fft 计算结果之间的误差。从图 5.9 可知信号功率主要集中在 $k=0$ 、 $k=102$ 和 $k=922$ 三个频率点,其中 $k=0$ 是直流分量的频率值。从图 5.10 的仿真结果误差可知在这三个频率点的仿真误差分别是 2036、1808.1 和 1781.2。通过计算,这三个频率点的信号量化噪声比分别达到 75.59dB、72.76dB 和 72.89dB。

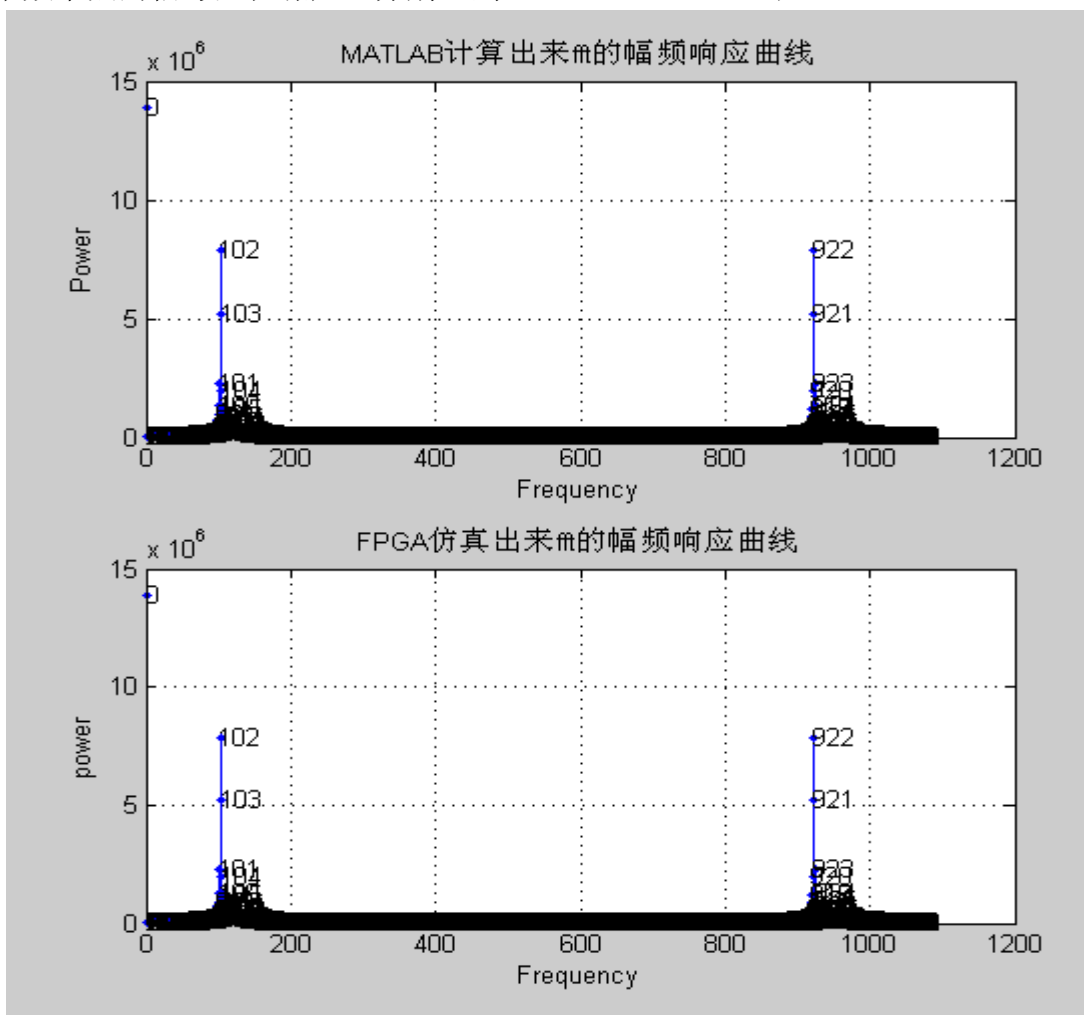


图 5.9 正弦信号频谱对比

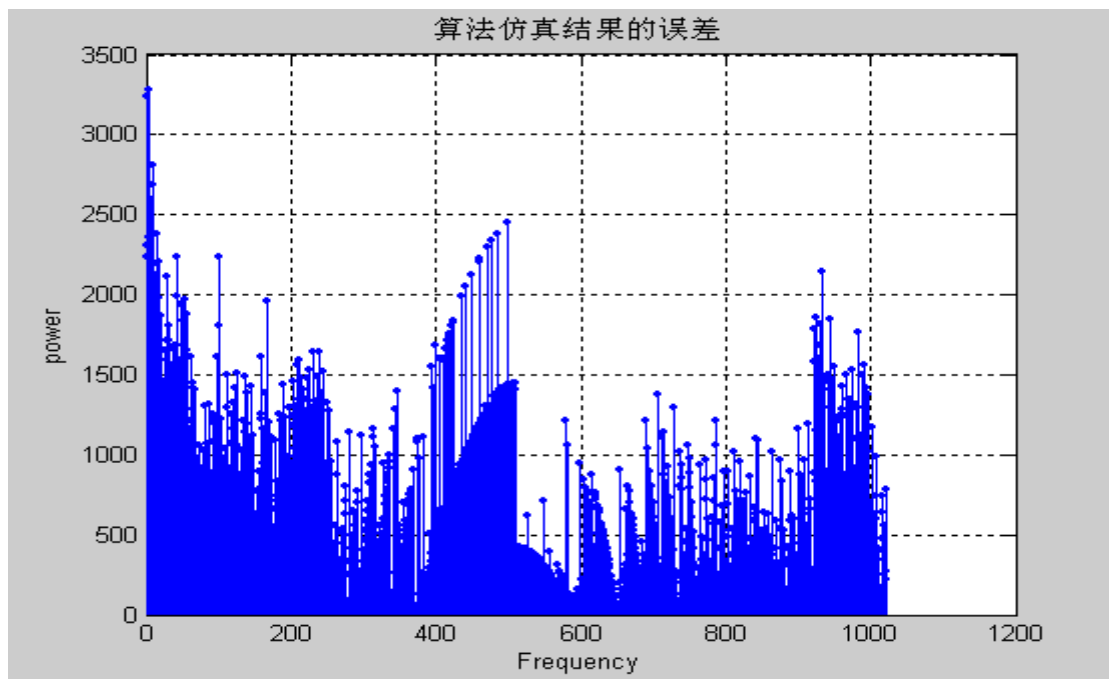


图 5.10 正弦信号算法仿真的误差

5.2.2 复信号仿真

设计的算法能完成复数 FFT 运算, 下面采用复三角波信号和复正弦信号分别对算法进行仿真测试。

(1) 复三角波信号仿真

使用 MATLAB 产生一个幅值是三角波的复信号:

$$x(n) = \begin{cases} (n/512 - 0.5) \times e^{j\pi/3} & 0 \leq n \leq 511 \\ (0.5 - (n - 512)/512) \times e^{j\pi/3} & 512 \leq n \leq 1023 \end{cases} \quad \text{式(5-4)}$$

把得到的测试数据按照前面介绍的方法进行预处理, 完成仿真后的输出结果和 MATLAB 分析的结果如图 5.11 所示。由图可知 MATLAB 自带的标准 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线基本上是一致的。图 5.12 是算法的仿真结果和 MATLAB 的标准 fft 计算结果之间的误差。从图 5.11 可知信号功率主要集中在 $k=1$ 和 $k=1023$ 两个频率点。从图 5.12 的仿真结果误差可知在这两个频率点的仿真误差分别是 1445.6 和 421.69。通过计算, 这两个频率点的信号量化噪声比分别达到 79.47dB 和 90.17dB。

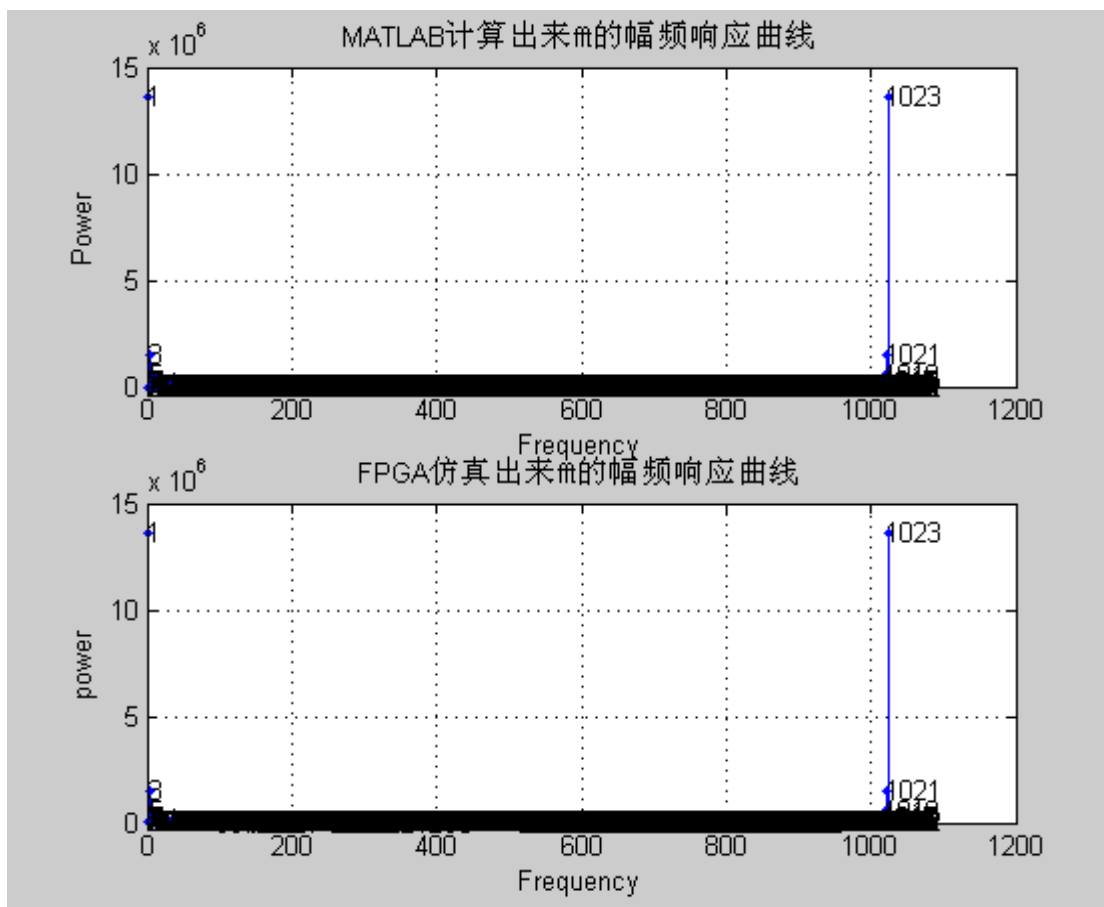


图 5.11 复三角波信号频谱对比

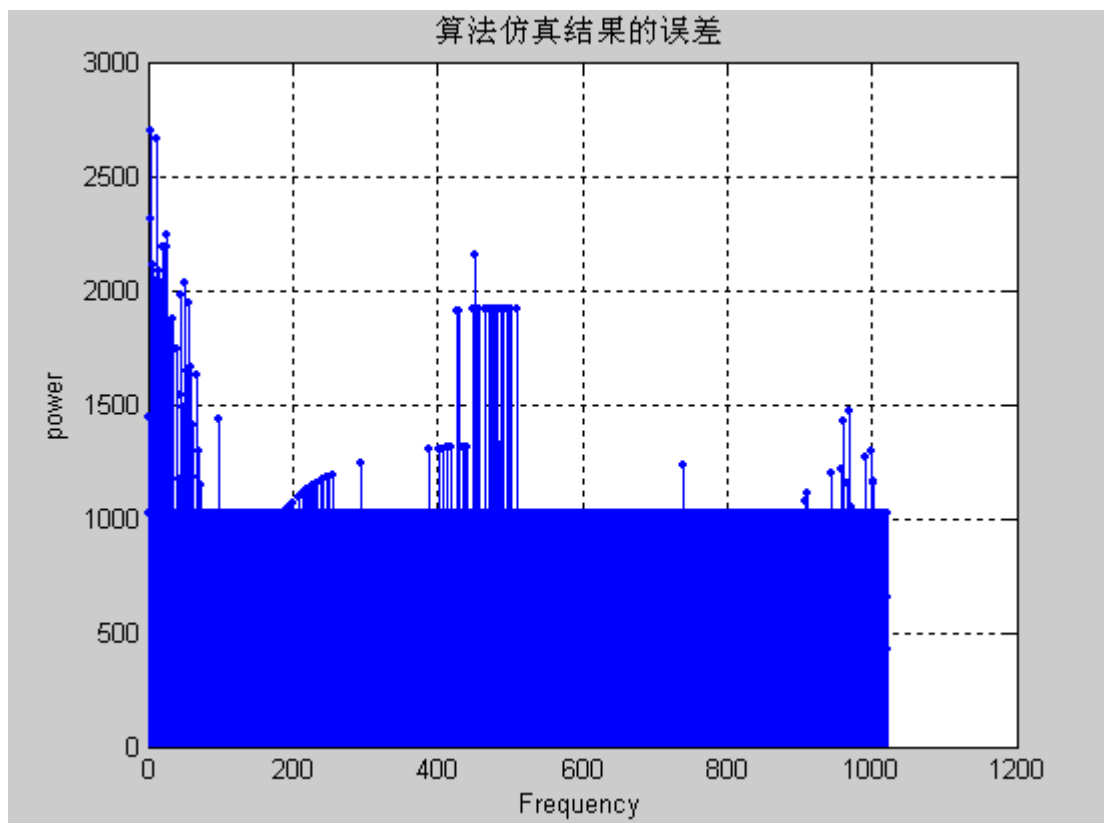


图 5.12 复三角波信号算法仿真的误差

(2) 复正弦信号仿真

使用 MATLAB 产生一个幅值是正弦规律变化的复信号:

$$x(t)=(3+4\times\sin(2\times\pi\times50\times t)+5\times\sin(2\times\pi\times100\times t))\times e^{j\pi/3} \quad \text{式(5-5)}$$

采样得到测试用的离散时间信号:

$$x(n)=(3+4*\sin(2*\pi*50/500*n)+5*\sin(2*\pi*100/500*n))*e^{j\pi/3} \quad n=0:1023 \quad \text{式(5-6)}$$

产生的正弦复信号由直流复分量和两个复正弦信号叠加而成, 因此同时对复常数信号进行了仿真测试, 两个正弦信号的频率分别是 50Hz 和 100Hz, 对信号的采样频率为 500Hz。把测试数据按照前面介绍的方法进行预处理, 完成仿真后的输出结果和 MATLAB 分析的结果如图 5.13 所示。由图可知 MATLAB 自带的标准 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线基本上是一致的。图 5.14 是算法的仿真结果和 MATLAB 的标准 fft 计算结果之间的误差。从图 5.13 可知信号功率主要集中在 $k=0$ 、 $k=102$ 、 $k=205$ 、 $k=819$ 和 $k=922$ 五个频率点, 其中 $k=0$ 是复直流分量的频率值。从图 5.14 的仿真结果误差可知在这五个频率点的仿真误差分别是 2551、1234.8、318.17、509.4 和 885.76。通过计算, 这五个频率点的信号量化噪声比分别达到 71.87dB、72.22dB、87.75dB、83.66dB 和 75.11dB。

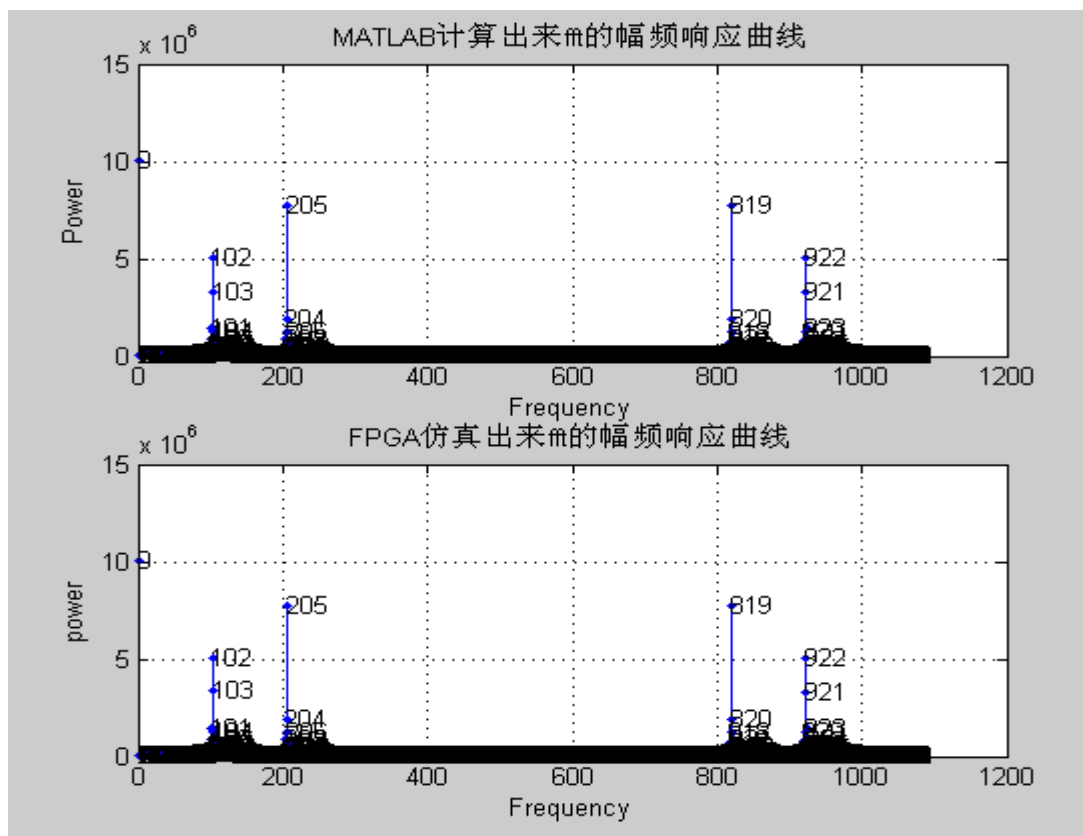


图 5.13 复正弦信号频谱对比

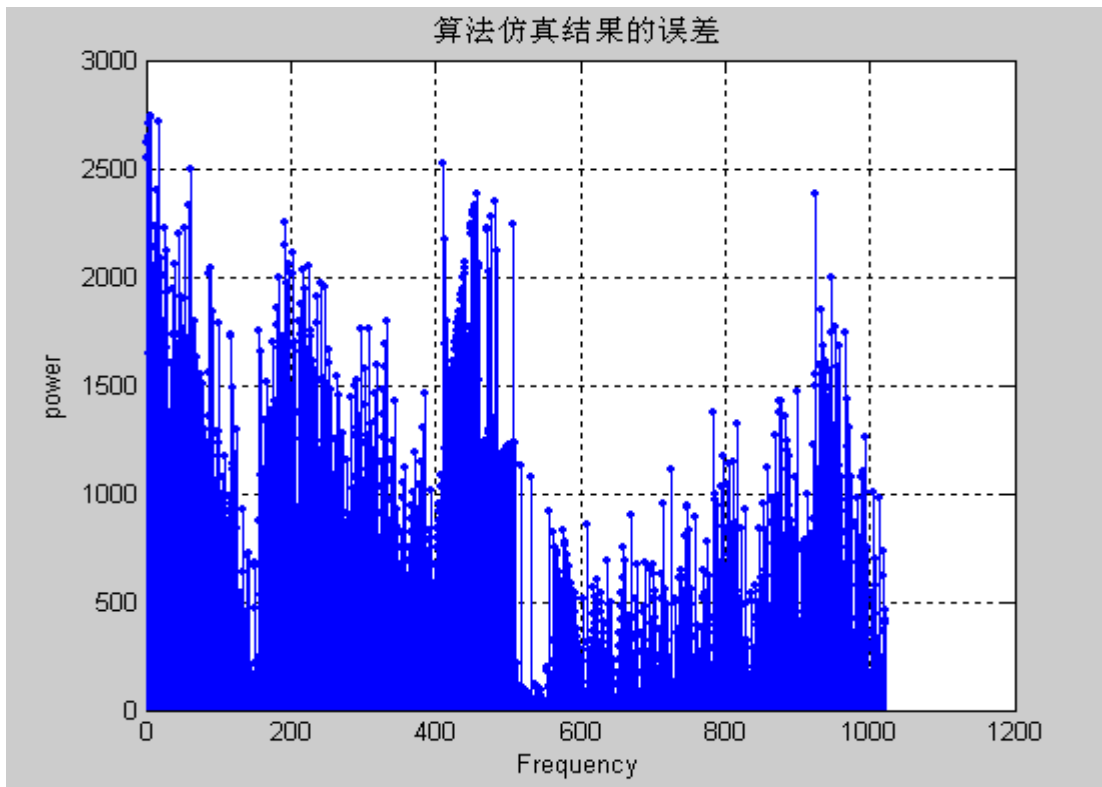


图 5.14 复正弦信号算法仿真误差

5.2.3 仿真结果分析

以上通过设计的 FFT 算法对几组典型的实信号和复信号进行了仿真, 仿真结果由 MATLAB 分析, 并和 MATLAB 的计算结果进行了比较, 得出了仿真结果的误差和信噪比。FPGA 实现 FFT 算法的过程中, 误差主要有 3 个来源: 第一种是输入数据的量化误差, 包括使用有限位数表示旋转因子和输入数据时的近似处理带来的舍入误差; 第二种是蝶形单元所造成的误差, 主要是源于数据做加减法时产生的误差, 并且数据在各级迭代的过程中误差会传递累计; 第三种是乘法器运算产生的误差^[28], 两个 B 位数相乘, 会得到一个 $2B$ 位的积, 如果把该积的低 B 位舍去, 就会产生误差, 当被截去的各位都是 1 时, 误差最大。对以上各例信号的仿真结果表明, 对于相同的数据输入, MATLAB 自带的标准 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线基本上是一致的, 设计的算法实现 FFT 运算的计算误差都在可接受的范围内, 频域信号主功率点的信号量化噪声比(SQNR)都在 70dB 以上, 达到了高性能 FFT 的运算要求。

本章从资源占用、速度运行和仿真结果三个方面对设计的 FFT 算法的性能进行了详细的分析。结果表明本设计满足对 FPGA 资源合理利用的系统需求, 且运行速度达到了高速运算的目的。通过几组典型的实信号和复信号对该算法进行了仿真测试, 并将其与 MATLAB 计算的理论值进行比较, 证明完全符合处理精度。因此, 本设计是切实可行的。

第六章 总结与展望

本文以基于 FPGA 的 FFT 算法的设计与实现作为选题,采用基-2 频域抽取的算法,在对 FFT 算法结构和相关计算数据流动特点仔细分析的基础上,采用自顶向下的设计方法,将算法分成多个功能模块,并用硬件描述语言 Verilog HDL 分别实现。以 FPGA 芯片 Virtex-5 XC5VFX70T 为约束条件,以 XILINX 公司提供的 ISE10.1 集成开发环境为软件平台,对算法的各个功能模块进行综合、仿真和实现,得到了相应的门和触发器构成的硬件电路。编写了测试平台对算法进行测试,计算结果与 MATLAB 自带的标准 fft 函数的计算结果相比,对于相同的数据输入, MATLAB 的 fft 函数计算出来的幅频响应曲线和本文设计的 FFT 算法的仿真结果的幅频响应曲线基本上是一致的,设计的算法实现 FFT 运算的计算误差都在可接受的范围内,频域信号主功率点的信号量化噪声比(SQNR)都在 70dB 以上,达到了高性能 FFT 的运算要求。

FFT 算法的蝶形运算采用流水线结构可实现数据的连续输入和输出,十级流水线大幅增加数据吞吐量,加快了 FFT 的处理速度;由 SDF 结构实现算法的流水处理,使用的存储单元少,存储单元利用率高,并很好的利用了输入数据的串行性;在对定点制的动态范围进行研究后,通过插入衰减因子和符号位与最高有效位的进位情况减少了截断误差的引入并防止了数据的溢出,提高了 FFT 的运算精度;输入数据和中间运算过程均采用带符号的补码表示,方便算法内部加减和乘法运算的处理;根据各级蝶形运算的旋转因子的变化规律,预先计算出旋转因子的值,并进行定点化处理,使之方便定点整数运算,作为旋转因子表存储,然后按照每级运算的需求直接读取,从而减少运算过程,提高运算速度。

ISE 软件综合后算法的最高时钟频率达到了 250MHz 以上,布局布线后的资源使用报告显示,整个设计占用 FPGA 的 Slice 数为 1379,约占芯片 Slice 总量的 12%,经过仿真算法完成一次完整的 1024 点 16bits 复数 FFT 运算需要 2073 个时钟周期。

本文研究的 FFT 算法取得了一定的成果,但还有许多不足之处。对 FFT 算法的设计与研究仅仅停留在算法的仿真阶段,由于时间原因,没有实现项目的实际应用,希望以后的工作中,可以把 FFT 算法的研究成果,应用到实际的项目中去;FFT 算法的实现采用的是定点制的表示方法,虽然仿真结果表明设计的正确性,但与块浮点制、浮点制相比,精度还有待提高;对最简单最常用的基-2 算法进行了分析和研究,基-4,基-8,分裂基等高基算法还有待进一步研究。在资源允许

的情况下,可以使用基-4 FFT 算法来实现设计。相对于基-2 FFT 算法,采用基-4 算法实现的设计能提高一倍左右的运算速度,但是所耗费的 FPGA 的片上资源也会提高近一倍。

随着 EDA 技术的发展,在数字信号处理的研究与设计中 FPGA 将占主导地位。FFT 算法已广泛应用于各个领域,是 OFDM 系统中的核心调制技术,对于第四代移动通信技术的开发和应用起着关键的作用,对其的研究具有很高的实际意义和社会价值。在后续在研究中,需要继续研究实现 FFT 算法的方法和结构,寻找实现 FFT 的最佳方式。

致谢

本设计及学位论文是在我的导师周端教授的亲切关怀和悉心指导下完成的。她严肃的科学态度，严谨的治学精神，精益求精的工作作风，平易近人的态度，深深地感染和激励着我。从论文的选题，毕业设计计划的制定以及设计的最终完成，周老师都始终给予我细心的指导和不懈的支持。整个设计过程中，周老师多次为我指点迷津，帮我开拓思路，在遇到困难时能给予我真诚的鼓励，找老师和同学帮我分析、解答，在此谨向周老师致以诚挚的谢意和崇高的敬意。

感谢我的学长张建贤博士，他丰富的知识，对专业的深刻理解让我在课题的研究过程中得到了很多帮助和指导，使我受益匪浅。感谢实验室里一起学习和工作的同学们，正是由于你们的帮助和支持，我才能克服一个一个的困难和疑惑，直至本文的顺利完成。

在我即将走上工作岗位之际，在此论文的完成过程中，周老师和实验室一起工作的同学给我提供了好多锻炼的机会，使我学到了很多将来会用到的知识，使我在顺利完成毕业设计的同时收获了很多快乐。在论文即将完成之际，我的心情无法平静，从开始进入课题到论文的顺利完成，有多少可敬的学长、同学、朋友给了我无言的帮助，在这里请接受我诚挚的谢意！最后我还要感谢培养我长大含辛茹苦的父母，谢谢你们！

最后要感谢各位评审老师在百忙中抽出时间对论文进行审稿和参加答辩会，并对各位参加答辩会的老师同学表示感谢！

参考文献

- [1] 丁玉美,高西全. 数字信号处理. 第二版. 西安:西安电子科技大学出版社, 2001. 97。
- [2] 韩泽耀,韩雁,郑为民. 一种高速实时定点 FFT 处理器的设计. 电路与系统学报. 2002, 7(1). 18。
- [3] 张欣. VLSI 数字信号处理——设计与实现. 北京: 科学出版社, 2003。
- [4] <http://nova.stanford.edu/bbaas/fftinfo.html>。
- [5] 植强. 一种基于 FPGA 的 FFT 阵列处理器. 电子对抗技术. 2002, 17(6). 36-39。
- [6] 杨兴,谢志远,戎丽. 基于 FPGA 的 FFT 处理器设计. 国外电子元器件. 2008, 5. 25-28。
- [7] Fast Fourier Transform v3.2 Product Specification. www.xilinx.com. August 2005。
- [8] FFT MegaCore Function User Guide. www.alter.com. April 2006。
- [9] 程佩青. 数字信号处理. 第三版. 北京: 清华大学出版社, 2007, 2。
- [10] 刘泉,厥大顺. 数字信号处理的原理及实现. 北京: 电子工业出版社, 2005。
- [11] 王城,吴继华,范丽珍等. Altera FPGA/CPLD 设计(基础篇). 北京: 人民邮电出版社, 2005。
- [12] 田耕,徐文波. Xilinx FPGA 开发实用教程. 北京: 清华大学出版社, 2008. 10。
- [13] 倪养华,王重玮. 数字信号处理——原理与实现. 上海: 上海交通大学出版社, 2003. 88-89。
- [14] 贾玉臣,吴嗣亮. 快速傅里叶变换的误差分析. 北京理工大学学报. 2005, 8. 84。
- [15] 范展,梁国龙,刘洋. 基于 FPGA 的新型浮点 FFT 处理器设计. 电子技术应用. 2008, 5. 23-26。
- [16] 朱冰莲,刘学刚. FPGA 实现流水线结构的 FFT 处理器. 重庆大学学报. 2004, 27(9) . 33-36。
- [17] 薛小刚,葛毅敏. Xilinx ISE 9.X FPGA/CPLD 设计指南. 北京: 人民邮电出版社, 2007. 32。
- [18] 连冰,宫丰奎,张力. 基于 FPGA 的快速傅里叶变换. 国外电子元器件。

- 2003, 12. 26。
- [19] 夏宇闻. Verilog 数字系统设计教程. 第二版. 北京: 北京航空航天大学出版社, 2008. 11。
- [20] 王旭东, 潘广桢. MATLAB 及其在 FPGA 中的应用. 北京: 国防工业出版社, 2006. 112。
- [21] 徐海波, 杜欢. OFDM 中 FFT 的高效实现. 机电工程技术. 2006, 35(9). 42。
- [22] 毛俊, 张学智. 快速傅里叶变换算法的比较. 西安工业学院学报. vol.22 No.2 June 2002。
- [23] 段玉波, 刘继新, 刘树庆. 基于 FPGA 的快速傅里叶变换(FFT)处理器的设计. 自动化技术与应用. 2006, 25(4)。
- [24] 鲁欣, 陈进, 付宇卓. 基于 FPGA 的 4096 点基-4 FFT 模块的实现. 系统工程与电子技术. 2004, 3, 26(3)。
- [25] Janick Bergeron. Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic Publishers, 2002。
- [26] 陈亚勇. MATLAB 信号处理详解. 北京: 人民邮电出版社, 2001。
- [27] 王宏. MATLAB 6.5 及其在信号处理中的应用. 北京: 清华大学出版社, 2004. 192。
- [28] 唐江, 刘桥. 基于 FPGA 的基-4 FFT 算法的硬件实现. 重庆工学院学报 (自然科学版). 2007, 21(3). 82。



西安电子科技大学

地址：西安市太白南路2号

邮编：710071

网址：www.xidian.edu.cn