

班 级 1801016  
学 号 18010100356

西安电子科技大学

# 集成电路导论



题 目 基于FPGA的

16点FFT实现

学 院 通信工程学院

专 业 通信工程

学生姓名 于彤洲

## 摘 要

数字电子技术和信息技术日新月异，在多媒体应用技术、计算机应用、图像与语音信号处理，通信服务等众多领域中，都已经广泛地使用到基于数字信号处理的理论与技术。快速傅里叶变换(FFT)在很大程度上推动了数字信号处理技术的发展，因为它的运算时间和直接计算离散傅里叶变换(DFT)相比大大减少，更有利于实现数字信号处理的应用，并为数字信号处理技术在实时处理各种信号的应用中提供了优势。因此，本文对 FFT 算法和实现方法的研究对其进一步发展起到了非常重要的理论指导和实践意义。

本文的研究内容立足于FPGA实现DIT-FFT算法，这种算法不仅计算流程图比较简单，而且具有结构良好的特点，更重要的是这中算法在理论上是最快的DFT算法，只需要  $N/2 \log_2 N$  次复数乘法运算。本文首先对DIT-FFT算法进行简单的理论说明及其推到，并分模块具体分析了每个模块应该实现哪些功能。最后，用Verilog硬件描述语言对各个共功能模块进行功能仿真，仿真实现16点复数的FFT。用测试信号对本设计进行测试，并分析计算结果和Matlab计算结果的误差，结果表明本文的设计正确。

**关键词：**FFT变换 DIT-FFT算法 Verilog HDL 蝶形算法

# 目 录

<b>第一章 引言 . . . . .</b>	<b>1</b>
<b>第二章 FFT原理概述 . . . . .</b>	<b>1</b>
2.1 DIT-FFT基本原理 . . . . .	2
2.2 运算规律和编程思想 . . . . .	3
2.2.1 原位计算 . . . . .	4
2.2.2 旋转因子的变化规律 . . . . .	4
2.2.3 蝶形运算规律 . . . . .	4
2.2.4 编程思想及程序框图 . . . . .	5
2.2.5 序列的倒序 . . . . .	5
2.3 定点数与有限字长效应 . . . . .	5
<b>第三章 Verilog设计思路及说明 . . . . .</b>	<b>6</b>
3.1 设计框图 . . . . .	6
3.2 存储单元设计 . . . . .	6
3.2.1 缓冲单元(RAM) . . . . .	6
3.2.2 旋转因子存储单元(ROM) . . . . .	7
3.2.3 地址发生存储单元(ROM) . . . . .	7
3.3 蝶形运算单元 . . . . .	7
3.4 控制单元 . . . . .	7
<b>第四章 波形仿真及其分析 . . . . .</b>	<b>7</b>
4.1 蝶形运算单元 . . . . .	8
4.2 FFT所有单元 . . . . .	8
4.2.1 Verilog仿真 . . . . .	8
4.2.2 与matlab仿真对比 . . . . .	9
<b>第五章 总结 . . . . .</b>	<b>10</b>
<b>参考文献 . . . . .</b>	<b>10</b>

## 第一章 引言

随着数字技术的快速发展，数字信号处理已深入到各个学科领域。在数字信号处理中，许多算法如相关、滤波、谱估计、卷积等都可通过转化为离散傅立叶变换(DFT)实现，从而为离散信号分析从理论上提供了变换工具。但 DFT计算量大，实现困难。快速傅立叶(FFT)<sup>[1]</sup>的提出，大大减少了计算量，从根本上改变了傅立叶变换的地位，成为数字信号处理中的核心技术之一，广泛应用于雷达、观测、跟踪、高速图像处理、保密无线通信和数字通信等领域。

目前，硬件实现FFT算法的方案主要有：通用数字信号处理器(DSP)、FFT专用器件和现场可编程门阵列(FPGA)。DSP具有纯软件实现的灵活性，适用于流程复杂的算法，如通信系统中信道的编译码、QAM映射等算法。DSP完成FFT运算需占用大量DSP的运算时间，使整个系统的数据吞吐率降低，同时也无法发挥DSP软件实现的灵活性。采用FFT专用器件，速度虽能够达到要求。但其外围电路复杂，可扩展性差，成本昂贵。随着FPGA发展，其资源丰富，易于组织流水和并行结构，将FFT实时性要求与FPGA器件设计的灵活性相结合，实现并行算法与硬件结构的优化配置，不仅可以提高处理速度，并且具有灵活性高。开发费用低、开发周期短、升级简单的特点。本文进行了DIT-FFT的Verilog实现，并以16位长数据，16点FFT为例，用modelsim仿真，软件上通过综合和仿真。

## 第二章 FFT原理概述

FFT算法就是不断地把长序列的DFT分解成几个短序列的DFT，并利用旋转因子  $W_N^{kn}$  的周期性和对称性来减少DFT的运算次数<sup>[2]</sup>。最简单最常用的是基2FFT，其算法分为两类：DIT-FFT和DIF-FFT，本文采用的是第一种。

## 2.1 DIT-FFT基本原理

设序列  $x(n)$  的长度为  $N$ , 且满足  $N = 2^M$ ,  $M$  为自然数。按  $n$  的奇偶把  $x(n)$  分解为两个  $N/2$  的子序列。

$$\begin{aligned} x_1(r) &= x(2r) & r = 0, 1, \dots, \frac{N}{2} - 1 \\ x_2(r) &= x(2r + 1) & r = 0, 1, \dots, \frac{N}{2} - 1 \end{aligned}$$

则  $x(n)$  的DFT为

$$\begin{aligned} X(k) &= \sum_{n \text{ is even}} x(n) W_N^{kn} + \sum_{n \text{ is odd}} x(n) W_N^{kn} \\ &= \sum_{r=0}^{N/2-1} x(2r) W_N^{2kr} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{k(2r+1)} \\ &= \sum_{r=0}^{N/2-1} x_1(r) W_N^{2kr} + W_N^k \sum_{r=0}^{N/2-1} x_2(r) W_N^{2kr} \\ &= \sum_{r=0}^{N/2-1} x_1(r) W_{N/2}^{kr} + W_N^k \sum_{r=0}^{N/2-1} x_2(r) W_{N/2}^{kr} \quad (W_N^{2kr} = W_{N/2}^{kr}) \\ &= X_1(k) + W_N^k X_2(k) \quad k = 0, 1, 2, \dots, N-1 \end{aligned} \tag{2-1}$$

其中  $X_1(k)$  和  $X_2(k)$  分别为  $x_1(r)$  和  $x_2(r)$  的  $N/2$  点DFT, 即:

$$X_1(k) = \sum_{r=0}^{N/2-1} x_1(r) W_{N/2}^{kr} = DFT[x_1(r)]_{N/2} \tag{2-2}$$

$$X_2(k) = \sum_{r=0}^{N/2-1} x_2(r) W_{N/2}^{kr} = DFT[x_2(r)]_{N/2} \tag{2-3}$$

由于  $X_1(k)$  和  $X_2(k)$  均以  $N/2$  为周期, 并且  $W_N^{k+N/2} = -W_N^k$ , 因此  $X(k)$  又可以表示为:

$$\begin{aligned} X(k) &= X_1(k) + W_N^k X_2(k) \quad k = 0, 1, \dots, \frac{N}{2} - 1 \\ X\left(k + \frac{N}{2}\right) &= X_1(k) - W_N^k X_2(k) \quad k = 0, 1, \dots, \frac{N}{2} - 1 \end{aligned} \tag{2-4}$$

这样, 就将  $N$  点DFT分解为两个  $N/2$  点DFT和2-4 式的运算, 2-4式的运算可以由图2.1的流图符号表示, 称为蝶形符号。

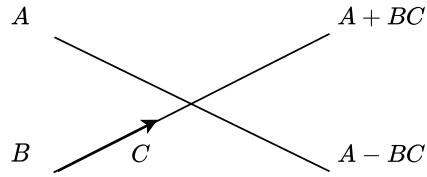


图 2.1 蝶形运算符号

由图 2.1 可以看出，完成一个蝶形运算，需要一个复数乘法和两次复数加法运算。

$N = 2^M, N/2$  仍然是偶数，故可以对  $N/2$  点 DFT 再进行进一步分解。与第一次分解相同，将  $x_1(r)$  按照奇偶分解为  $\frac{N}{4}$  点的子序列。

.....

经过  $M$  轮分解，最后得到  $N$  点 DFT 分解成  $N$  个 DFT 和  $M$  级蝶形算符，而 1 点 DFT 就是时间序列本身。图中输入序列不是顺序序列，但后面可以看到，其排列是有规律的。

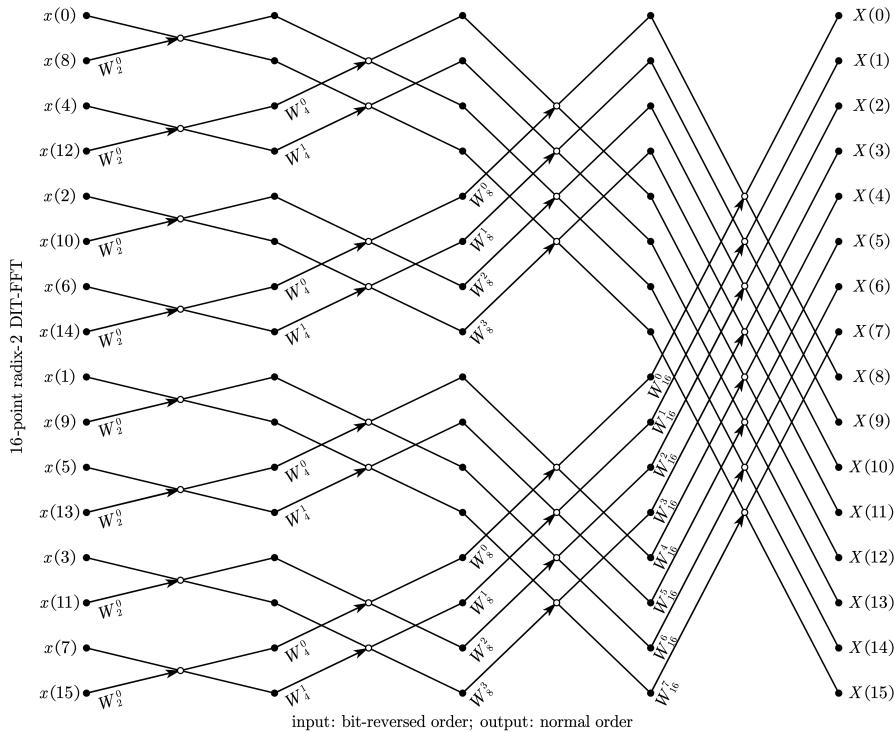


图 2.2 16点DIT-FFT的运算流图

## 2.2 运算规律和编程思想

为了方便设计出DIF-FFT的硬件电路，下面介绍DIF-FFT的运算规律和编程思想。

### 2.2.1 原位计算

由图2.2,可以看出。 $N = 2^M$  点的FFT共进行  $M$  级运算, 每级由  $N/2$  个蝶形组成。计算完一个蝶形后, 所得的输出数据可以立即存入原输入数据所占用的存储单元(数组元素)。这样, 经过  $M$  级运算后, 原来存放输入序列的  $N$  个存储单元便依次存放  $X(k)$  的  $N$  个值。因为  $M$  级, 存储单元的值要每一级进行更新, 所以需设计  $N$  个**RAM**单元。这种利用同一存储单元存储蝶形计算输入、输出数据的方法称为原位计算。原位计算可以节省大量内存, 从而使设备成本降低。

### 2.2.2 旋转因子的变化规律

每个蝶形都要乘一个因子  $W_N^p$ , 称为旋转因子,  $p$  为旋转因子的指数。(在Verilog中, 可以用一个**ROM**来存储旋转因子的数值)但各级的旋转因子和循环方式都有所不同。应获得旋转因子  $W_N^p$  与运算级数的关系。用  $L$  表示从左到右的运算级数  $L = 1, 2, \dots, M$ . 观察图2.2可以发现, 第  $L$  级共有  $2^{L-1}$  个不同的旋转因子。第  $L$  级的旋转因子为

$$W_N^p = W_{2^L}^J = W_{N \cdot 2^{L-M}}^J = W_N^{J \cdot 2^{M-L}} \quad J = 0, 1, 2, \dots, 2^{L-1} - 1 \quad (2-5)$$

这样, 就可以通过2-5式来确定第  $L$  级的旋转因子。

### 2.2.3 蝶形运算规律

设序列  $x(n)$  经时域抽选(倒序)后, 按照图2.2存入数组(内存)  $A$  中。如果蝶形运算的两个输入数据相互距离  $B$  个点, 应用原位计算, 有

$$A_L(J) \Leftarrow A_{L-1}(J) + A_{L-1}(J+B)W_N^p$$

$$A_L(J+B) \Leftarrow A_{L-1}(J) - A_{L-1}(J+B)W_N^p$$

式中  $p = J \times 2^{M-L} \quad J = 0, 1, \dots, 2^{L-1} - 1; L = 1, 2, \dots, M$

#### 2.2.4 编程思想及程序框图

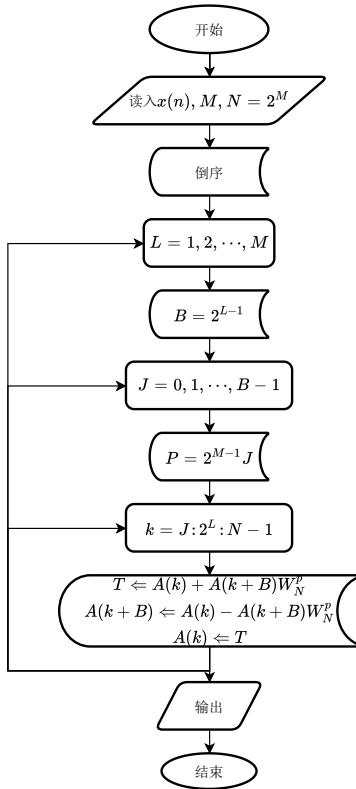


图 2.3 DIT-FFT运算程序图

#### 2.2.5 序列的倒序

DIT-FFT算法运算流图的输出 $X(k)$ 为自然顺序，但为了适应原为计算，其输入序列不是按照 $x(n)$ 的自然顺序排列，这种经过 $M$ 次偶奇抽选后的排序称为序列 $x(n)$ 的倒序(0-N-1十进制数转为二进制数，然后翻转二进制数，译出十进制数)。因此，在运算 $M$ 级蝶形之前应先对序列 $x(n)$ 进行倒序。

### 2.3 定点数与有限字长效应

定点数指的是在二进制数中小数点的位置是固定的数，在本设计中采用定点数，目的是浮点运算在Verilog中耗时较慢。浮点数虽然扩大了数的表示范围，但这是以降低精度为代价。浮点运算要比定点运算复杂，定点运算时，当运算结果超出数的表示范围，就会发生溢出。

当FPGA实现乘法运算时，例如计算两个 $N$ 位宽的二进制数的乘积，乘积的结果一半都会用 $2N$ 位宽的二进制数表示，此时都会将结果进行适当的舍位处理，以较少最终的数据宽度。进行舍位就会引入误差，这种误差术语运算量化误差，

也称为运算噪声。

## 第三章 Verilog设计思路及说明

### 3.1 设计框图

图3.1给出了FFT算法的结构框图。这种形式的FFT只有一个蝶形运算单元，蝶形运算按照递归的方式，根据蝶形图从左往右，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，逐级地进行运算，直至第 $\frac{N}{2} \log_2 N$ 个蝶形，完成 $N$ 点FFT的全部计算。

在实际设计时，可以将输入缓冲单元和输出缓冲单元做成同一个存储单元。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<sup>[3]</sup>。

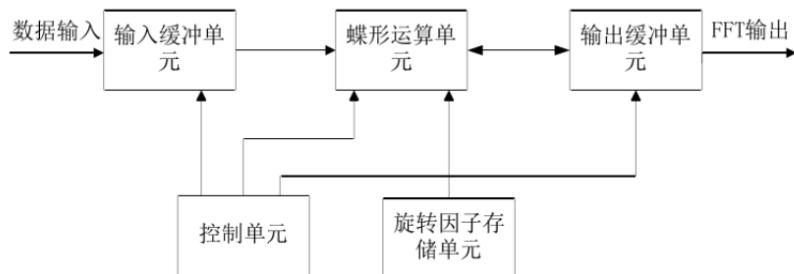


图 3.1 Verilog设计结构框图

以下所有的单元设计对应的Verilog程序见<https://github.com/DestinyEnvoy/Fast-Fourier-Transform/tree/master/unit/>

### 3.2 存储单元设计

#### 3.2.1 缓冲单元(RAM)

**mem\_32x16.v**文件实现16个32bit的内存单元。每一个单元前16bit存储复数的实数部分，后16bit存储复数的叙述部分。通过内容2.2.1可以知道，每个蝶形运算的输入、输出数据均要经过RAM的读写操作，因此RAM的频繁读写操作速度对FFT的处理速度影响较大。为了加快速度，我们设计可以同时读取/写入两个单元数据的RAM，加速蝶形运算。

### 3.2.2 旋转因子存储单元(ROM)

**w\_lut.v**文件实现8个旋转因子的存储，即  $W_{16}^0, W_{16}^1, \dots, W_{16}^7$ 。每个旋转因子用32bit存储，高16bit存储实数部分，低16bit存储虚数部分，每部分存储都用2.14格式的定位数存储。即第  $k$  个旋转因子实数，虚数部分分别存储  $2^{14} \cos \frac{2k\pi}{16}$ ,  $-2^{14} \sin \frac{2k\pi}{16}$  整数部分的二进制码。

### 3.2.3 地址发生存储单元(ROM)

**read\_addr\_lut.v**文件实现了第  $L$  级第  $k$  个蝶形单元，需要结合的三个数据  $X(k)$ ,  $X(k+B)$ ,  $W_N^p$  的在以上**RAM**, **ROM**对应的地址信息。如第0级第0个蝶形结合**RAM**钟第0第8个数据，第0个蝶形存储单元，这三个地址对应的数参与蝶形运算。(参照图2.2) 该地址发生存储单元是控制单元的一部分。

## 3.3 蝶形运算单元

**bufferfly.v**文件实现了复数的蝶形运算，运算原理参考图2.1。蝶形运算两个端口输出也是32bit, 高实，低虚数，第一二三四级蝶形输出分别是9.7,10.6,11.5,12.4格式的定位数。同时对该蝶形运算做了饱和溢出与舍入处理：高于最大正数32'h7fff0000计做16'h7fff, 小于最大负数32'h80000000，计为16'h8000，否则取乘法结果的第28到13位作为数据输出。

## 3.4 控制单元

**fft\_ctrl\_sm.v**文件实现每一个蝶形运算前后所有的控制信息。如某个蝶形运算接收后，下一个蝶形序号的变化或是否进入下一级FFT，以及此时**RAM**的读写控制信号。很容易知道，该单元包含了地址发生单元(**ROM**)。

**fft\_top.v**文件实现所有单元信号的综合。是最顶层的模块。

# 第四章 波形仿真及其分析

以下两个**test bench** 文件对应Verilog源码见 <https://github.com/DestinyEnvoy/Fast-Fourier-Transform/tree/master/sim>

## 4.1 蝶形运算单元

**butterfly\_tb.v**文件实现了蝶形运算单元的仿真。蝶形运算单元是FFT设计中最重要的一部分，因此我们对蝶形运算单元进行分析。

我们设计一个test bench,使得输入一个8.8格式的数据输入  $A = 2.5 + 5.25j$ ,  $B = 6 + 1.5j$ ,  $C = 1 + 0.375j$ , 因此输出  $Y = A - BC = -2.9375 + 1.5j$ ,  $X = A + BC = 7.9375 + 9j$ ,  $X, Y$ 写成 9.7格式的定位数  $2^7Y = -376 + 192j$ ,  $2^7X = 1016 + 1152j$ , 详细计算如下表4.1

表 4.1 蝶形运算单元理论值

输入	A	B	C
浮点数	$2.5 + 5.25j$	$B = 6 + 1.5j$	$1 + 0.375j$
定位数(8.8 /2.14格式)	$640 + 1344j$	$1536 + 384j$	$16384 + 6114j$
输出	$X = A + BC$	$Y = A - BC$	
浮点数	$7.9375 + 9j$	$-2.9375 + 1.5j$	
定位数(9.7格式)	$1016 + 1152j$	$-376 + 192j$	

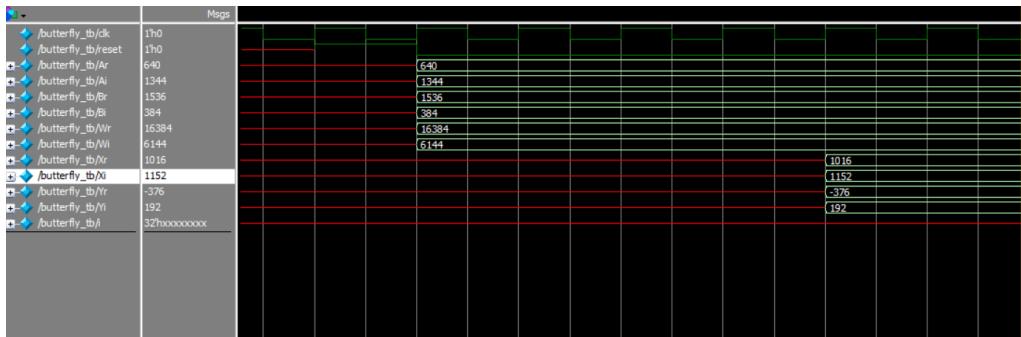


图 4.1 蝶形运算单元仿真

对比表4.1和图4.1,可以发现，两者结果相同，蝶形运算单元可以正常工作。

## 4.2 FFT所有单元

### 4.2.1 Verilog仿真

**fft\_top\_tb.v**文件实现了**fft**单元的仿真测试。我们分两路传输复数，第一路串行输入复数的实部，第二路同步串行输入复数的虚部。输入都是8.8格式的定位数。如下图：

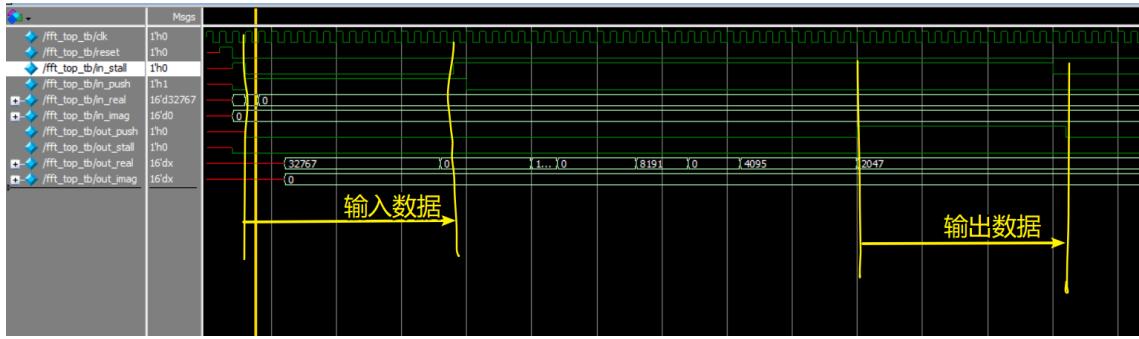


图 4.2 FFT 单元仿真

输入  $x(0) = (2^{15} - 1)/2^8, x(k) = 0, k = \text{其他}$  ( $x(0)$  为 Verilog 输入的最大值), 理论得出  $X(k) = x(0)/16$ , 其 12.4 格式的定位数为  $\lfloor 2^4 X(k) \rfloor = 2047$ 。图 4.2 与理论相符。

#### 4.2.2 与 matlab 仿真对比

测试四组数据(fft\_top\_tb.v中有四组数据的生成方式), 得到matlab的数据以及verilog FFT单元的FFT(忽略了小数的舍入误差), 如下图4.3

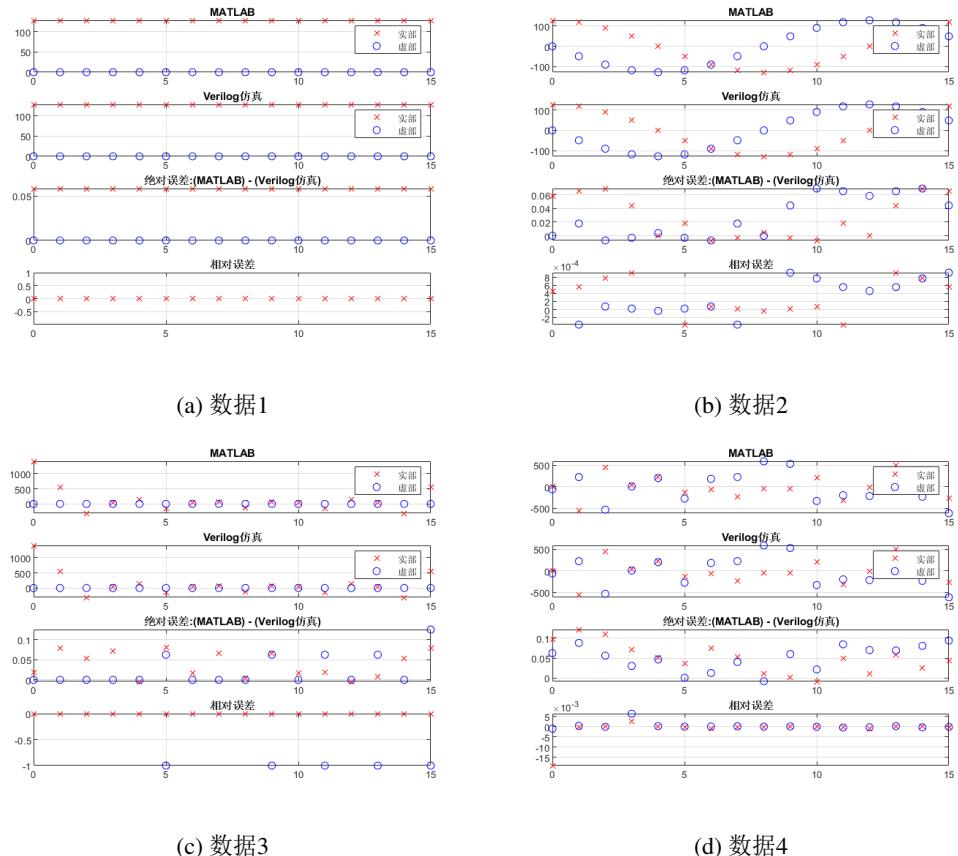


图 4.3 数据对比

从图4.3可以发现，数据1,2,4绝对误差在0.05以内，相对误差在0.005以内，这些都是在容许的范围内。数据3的绝对误差也在0.05以内，5, 9, 11, 13处的相对误差为1原因可能是该点处的理论值较小，造成了相对误差的扩大。

## 第五章 总结

本文分析了DIT-FFT的各个细节，从而用硬件描述语言实现了各个模块的编码。测试各个模块正确之后，进行整体的算法设计。

最后，对整个设计进行实现、仿真、验证，并将计算结果与matlab计算结果进行比较，观察FFT变换之后的仿真结果，通过数据分析得出本次设计的仿真结果与matlab计算结果在允许的误差范围内完全吻合，达到设计要求。本设计虽然取得了一些成果，但也存在一些不足：

1. 本实验虽然采用了DIT-FFT的思路，但因为整个模块共用一定蝶形运算单元，因此运算速度必然会降低很多，同时，本实现没有优化具体的设计，换而言之，程序有优化的空间，使得运算速度加快。
2. 本设计采用的是整数处理的方法，误差必然存在，对运算结果有一定的影响。
3. 数据的输入和运算输出均采用串行传输，对于 $2^M$ 较大时，会产生很严重的时延，因此本程序不能直接扩展为高点数的FFT

通过这次设计，我熟悉了FPGA先进期间和一些良好的设计思想和设计方法。同时也提高了自己分析和解决问题的能力。这些都成为我以后学习工作中的参考。

## 参考文献

- [1] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex fourier series[J/OL]. Mathematics of Computation, 1965, 19(90):297-301. <http://www.jstor.org/stable/2003354>.
- [2] 高西全, 丁玉美. 数字信号处理[M]. 第四版. 西安电子科技大学出版社, 2016.5: 121-130.
- [3] @ameyk1. [EB/OL]. (2016-5-27). <https://github.com/ameyk1/Fast-Fourier-Transform>.