郑州轻工业学院

课程设计

题目：128 点 FFT 变换的 FPGA 实现

姓　　名：钟 广

院　　系：电气信息工程学院

专业班级：电子信息工程 11-02

学　　号：541101030256

指导教师：胡智宏

成　　绩：

时间：2014 年 6 月 16 日至 2014 年 6 月 28 日

摘 要

快速傅立叶变换(FFT)作为时域和频域转换的基本运算，是数字谱分析的必要前提。传统的 FFT 使用软件或 DSP 实现，高速处理时实时性较难满足。FPGA 是直接由硬件实现的，其内部结构规则简单，通常可以容纳很多相同的运算单元，因此 FPGA 在作指定运算时，速度会远远高于通用的 DSP 芯片。FFT 运算结构相对比较简单和固定，适于用 FPGA 进行硬件实现，并且能兼顾速度及灵活性。本文介绍了一种通用的可以在 FPGA 上实现 128 点 FFT 变换的方法。设计复数乘法器为核心设计了 FFT 算法中的基-2 蝶形运算单元，溢出控制单元和地址与逻辑控制模块等其它模块,并以这些模块和 FPGA 内部的双口 RAM 为基础组成了基-2FFT 算法模块。

关键词：FPGA、FFT

目　　录

1 绪论

1.1 研究背景

在现代通信中，提高频谱利用率一直是人们关注的焦点之一。近几年来，随着通信业务需求的迅速增长，寻找频谱利用率更高的数字调制方式已成为数字通信系统设计、研究的主要目标之一。为了实现这个目标，通信系统正采用比以前更加复杂的调制信号。OFDM(正交频分复用)是一种高效的多载波调制技术，其最大的特点是传输速率高，具有很强的抗码间干扰和信道选择性衰落能力，具有非常高的频谱利用率。

1.1.1 无线通信的发展和现状

现代无线通信技术的发展始于本世纪 20 年代，经历了早期专用移动通信系统的发展，公用移动通信业务的发展，到 1978 年底，美国贝尔试验室研制成功先进移动电话系统(AMPS)，建成了蜂窝状移动通信网，大大提高了系统容量。随后投入商用，服务区域在美国逐渐扩大。其它工业化国家也相继开发出蜂窝式公用移动通信网，这种模拟通信系统被称为第一代移动通信系统。

到 80 年代中期，欧洲首先推出了泛欧数字移动通信网(GSM)的体系。随后，美国和日本也制定了各自的数字移动通信体制。以 GSM 为代表的数字移动通信系统被称为第二代移动通信系统。

1.1.2 OFDM 通信技术的发展

正交频分复用(OFDM)是一种多载波数字调制技术，它由多载波调制（MCM）发展而来。随着 DSP 芯片技术的发展，大规模集成电路让 FFT 技术的实现不再是难以逾越的障碍，傅立叶变换/反变换、高速 modem 采用的 64/128/256QAM 技术、栅格编码技术、软判决技术、信道自适应技术、插入保护时段、减少均衡计算量等成熟的技术逐步引入到移动通信领域中来，人们开始集中越来越多的精力开发 OFDM 技术在移动通信领域的应用。

1.1.3 可编程器件的发展

可编程逻辑器件的两种主要类型是现场可编程门阵列（FPGA）和复杂可编程逻辑器件 （CPLD)。在这两类可编程逻辑器件中， FPGA 提供了最高的逻辑密度、最丰富的特性和最高的性能。 自 1985 年 Xilinx 公司推出第一片现场可编程逻辑器件 （FPGA） 至今， FPGA 已经历了十几年的发展历史。在这十几年的发展过程中，以 FPGA 为代表的数字系统现场集成技术取得了惊人的发展。

2 OFDM 的基本原理

2.1 OFDM 的基本原理

正交频分复用(OFDM)技术与已经普遍应用的频分复用技术十分相似。与普通的频分复用基本原理相同，OFDM 把高速的数据流通过串并变换分配到速率相对较低的若干个频率子信道中进行传输，不同的是，OFDM 技术更好地利用了控制方法，使频谱利用率有所提高。

2.1.1 OFDM 的产生和发展

OFDM 的思想早在 20 世纪 60 年代就已经提出，由于使用模拟滤波器实现起来的系统复杂度较高，所以一直没有发展起来。在 20 世纪 70 年代，S.B.Weinstein 提出用离散傅里叶变换(DFT)实现多载波调制，为 OFDM 的实用化奠定了理论基础；在 80 年代，L.J.Cimini 首先分析了 OFDM 在移动通信应用中存在的问题和解决方法，从此以后，OFDM 在移动通信中的应用得到了迅猛的发展。

OFDM 系统收发机的典型框图如图 2.1 所示。发送端将被传输的数字信号转换成载波幅度和相位的映射，并进行离散傅里叶反变换(IDFT)将数据的频谱表达式变换到时域上，IFFT 变换与 IDFT 变换的作用相同，只是有更高的计算效率，所以适用于所有的应用系统。其中，上半部分对应于发射机链路，下半部分对应于接收机链路。由于 FFT 操作类似于 IFFT,因此发射机和接收机可以使用同一硬件设备。

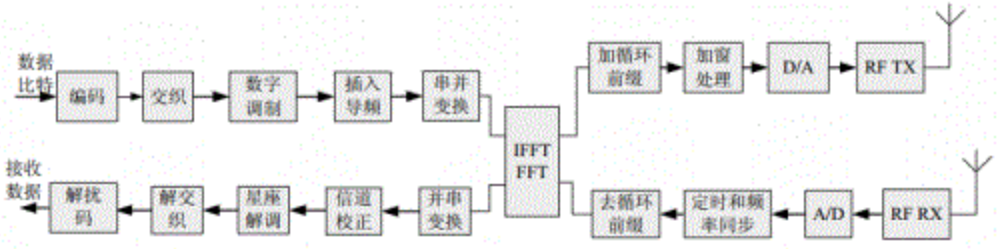

图 2-1 OFDM 收发件框图

3 FFT 算法原理

3.1 FFT 的主要算法

3.1.1 基-2FFT 算法

长度为 N 的有限长序列 x(n)的 DFT 的表达式为

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{kn},k=0,1,\cdots,N-1 \tag{3-1}$$

一个 N 点 DFT 可以看做是由几个较短的 DFT 组成的。基于这一思想，可以将 N 点 DFT 分解为几个较短的 DFT，这样一来乘法次数将大大减少，能够非常明显地降低 DFT 的运算量。此外，旋转因子W^mN具有明显的周期性和对称性。其周期性表现为：

$$W_N^{m+lN}=e^{-j\frac{2\pi}{N}(m+lN)}=e^{-j\frac{2\pi}{N}m}=W_N^m \tag{3-2}$$

其对称性表现为

$$W_N^{-m}=W_N^{N-m} \qquad [W_N^{N-m}]^*=W_N^m \tag{3-3}$$

不断的把长序列的 DFT 分解成几个短序列的 DFT，并且利用W^mN的周期性和对称性来减少 DFT 的运算次数，这就是 FFT 算法的基本思想。比较常用的 FFT 算法有基-2 FFT 和基-4FFT 两种。基-2 FFT 中的基 2 指的是 N=2^M，即有限长序列的长度 N 要到等于 2 的整数次幂)。下面就以 8 点的 FFT 为例详细分析基-2 FFT 算法。

3.1.2 基-2 FFT 算法基本原理

基-2 FFT 算法基本上分为时域抽取法 FFT(DIT-FFT)和频域抽取法 FFT(DIF-FFT)两大类。由于这两种算法的基本原理是相同的，所以下面主要介绍 DIT-FFT 算法。本课题采用的就是 DIT-FFT 这一算法。

设序列 x(n)的长度为 N，并且有以下的条件成立

$$N=2^M,\text{M 为自然数} \tag{3-4}$$

$x_1(r)$和 $x_2(r)$是 x(n)按 n 的奇偶性分解成的两个 N/2 点的子序列，如下式所示

$$x_1(r)=x(2r),\qquad r=0,1,\cdots\frac{N}{2}-1 \tag{3-5}$$

$$x_2(r)=x(2r+1),\qquad r=0,1,\cdots\frac{N}{2}-1 \tag{3-6}$$

那么 x(n)的 DFT 为

$$X(k)=\sum_{n-}x(n)W_N^{kn}+\sum_{n-}x(n)W_N^{kn}$$

$$=\sum_{r-0}^{N/2-1}x(2r)W_N^{2kr}+\sum_{r-0}^{N/2-1}x(2r+1)W_N^{k(2r+1)}$$

$$=\sum_{r-0}^{N/2-1}x_1(r)+W_N^k\sum_{r-0}^{N/2-1}x_2(r)W_N^{2kr} \tag{3-7}$$

由于

$$W_N^{2kr}=e^{-j\frac{2\pi}{N}2kr}=e^{\frac{-j\frac{2\pi}{N}kr}{2}}=W_{N/2}^{2kr} \tag{3-8}$$

所以

$$X(k)=\sum_{r-0}^{N/2-1}x_1(r)W_{N/2}^{kr}+W_N^k\sum_{r-0}^{N/2-1}x_2(r)W_{N/2}^{kr}=X_1(k)+\mathrm{W}_N^kX_2(k) \tag{3-9}$$

k=0, 1, …, N-1

其中 X1(k)和 X2(k)分别为 x1(r)和 x2(r)的 N/2 点 DFT，即

$$X_1(k)=\sum_{r-0}^{N/2-1}x_1(r)W_{N/2}^{kr}=DFT[x_1(r)] \tag{3-10}$$

$$X_2(k)=\sum_{r-0}^{N/2-1}x_2(r)W_{N/2}^{kr}=DFT[x_2(r)] \tag{3-11}$$

又由于 X1(k)和 X2(k)都是以 N/2 为周期，且

$$W_N^{k+\frac{N}{2}} = -W_N^k \tag{3-12}$$

所以 X(k)又可以表示为如下所示的表达式

$$X(k) = X_1(k) + W_N^k X_2(k) \qquad k = 0,1\cdots\frac{N}{2}-1 \tag{3-13}$$

$$X(k+\frac{N}{2}) = X_1(k) - W_N^k X_2(k) \qquad k = 0,1\cdots\frac{N}{2}-1 \tag{3-14}$$

这样一个 N 点的 DFT 就被拆分成为了两个 N/2 点的 DFT。式(3-7)和式(3-8)说明了原 N 点的 DFT 和这两个 N/2 点的 DFT 之间的关系。

采用蝶形运算符号的这种图示方法，可以用图 3-1 来表示前面所讲到的运算。在图 3.2 中，$N=2^3=8$，式(3-13)给出了 X(0)～X(3)的计算方法，而式(2-14)给出了 X(4)～X(7)的计算方法。

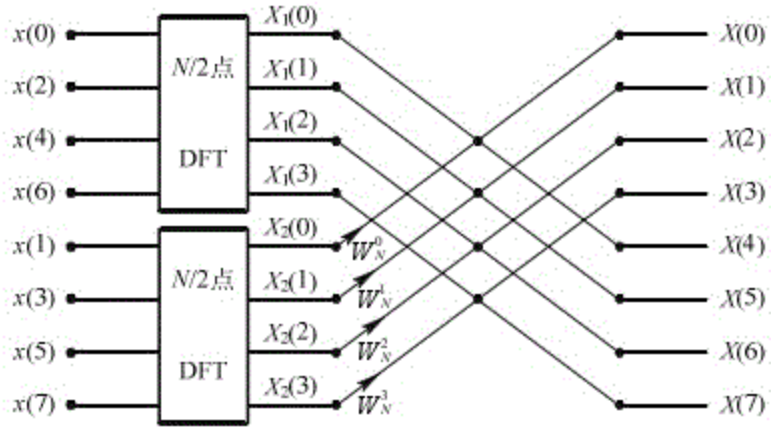

图 3-1 N 点 DFT 的一次时域抽取分解图(N=8)

由图 3-1 可以看出，经过一次分解后，计算一个 N 点 DFT 共需要计算两个 N/2 点 DFT 和 N/2 个蝶形运算。由前面的说明可以知道，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。那么按图 3-1 计算 N 点 DFT 共需要 $2(N/2)^2$+N/2=N(N+1)/2≈$N^2/2$(N>>1)次复数乘法和 N(N/2-1)+2N/2=$N^2/2$ 次复数加法运算。通过对比可以看出，只进行过这样的一次分解就使得运算量减少了近一半，充分说明了这样分解对减少 DFT 的运算量是十分有效的。由于这里 $N=2^M$，N/2 仍然是偶数，为了使得计算量能够得到进一步的减少，可以仿效前面的做法对 N/2 点 DFT 再做进一步分解。

与第一次分解相同，x3(1)和 x4(1)为 x1(r)按奇偶分解成的两个长为 N/4 的子序列，即

$$\left.\begin{aligned} x_3(l) &= x_2(2l) \\ x_4(l) &= x_1(2l+1) \end{aligned}\right\}, l = 0,1,\cdots,\frac{N}{4}-1 \tag{3-15}$$

那么，X1(k)又可表示为

$$X_1(k)=\sum_{i=0}^{N/4-1}x_1(2l)W_{N/2}^{2kl}+\sum_{i=0}^{N/4-1}x_1(2l+1)W_{N/2}^{k(2l+1)}$$

$$=\sum_{i=0}^{N/4-1}x_3(l)W_{N/4}^{kl}+W_{N/2}^{k}\sum_{i=0}^{N/4-1}x_4(l)W_{N/4}^{kl}$$

$$=x_3(k)+W_{N/2}^{k}X_4(k),k=0,1,\cdots N/2-1 \tag{3-16}$$

其中

$$x_3(k)=\sum_{i=0}^{N/4-1}x_3(l)W_{N/4}^{kl}=DFT[x_3(l)] \tag{3-17}$$

$$x_4(k)=\sum_{i=0}^{N/4-1}x_4(l)W_{N/4}^{kl}=DFT[x_4(l)] \tag{3-17}$$

同理，由 $X_3(k)$ 和 $X_4(k)$ 的周期性和 Wm^{N2} 的对称性 $W_{N/2}^{k+N/4}=-W_{N/2}^{k}$ 最后得到：

$$\left.\begin{array}{l}X_1(k)=X_3(k)+W_{N/2}^{k}X_4(k)\\X_1(k+N/4)=X_3(k)-W_{N/2}^{k}X_4(k)\end{array}\right\},k=0,1,\cdots,N/4-1 \tag{3-19}$$

同理可得

$$\left.\begin{array}{l}X_2(k)=X_5(k)+W_{N/2}^{k}X_6(k)\\X_2(k+N/4)=X_5(k)-W_{N/2}^{k}X_6(k)\end{array}\right\},k=0,1,\cdots,N/4-1 \tag{3-20}$$

其中有

$$X_5(k)=\sum_{i=0}^{N/4-1}x_5(l)W_{N/4}^{kl}=DFT[x_5(l)] \tag{3-21}$$

$$X_6(k)=\sum_{i=0}^{N/4-1}x_6(l)W_{N/4}^{kl}=DFT[x_6(l)] \tag{3-22}$$

$$\left.\begin{array}{l}x_5(l)=x_2(2l)\\x_6(l)=x_2(2l+1)\end{array}\right\},l=0,1\cdots N/4-1 \tag{3-23}$$

这样，如图 3-2 所示，经过第二次的分解，一个 N/2 点的 DFT 就被拆分成为了两个 N/4 点的 DFT 了。式(3-10)和式(3-11)说明了原 N/2 点的 DFT 和这两个 N/4 点的 DFT 之间的关系。依次类推，经过 M-1 次分解，最后将 N 点 DFT 分解成 N/2 个 2 点 DFT。将前面两次分解的过程综合起来，就得到了一个完整的 8 点 DIT-FFT 运算流图。

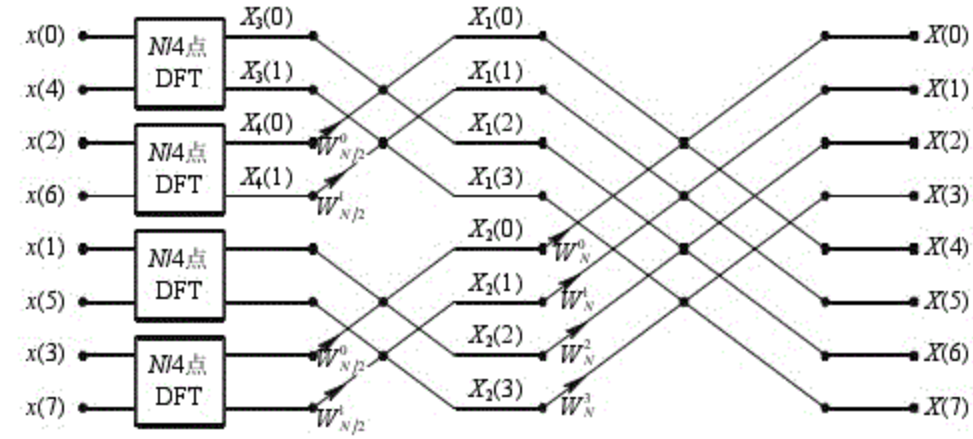

图 3-2 N 点 DFT 的第二次时域抽取分解图（N=8）

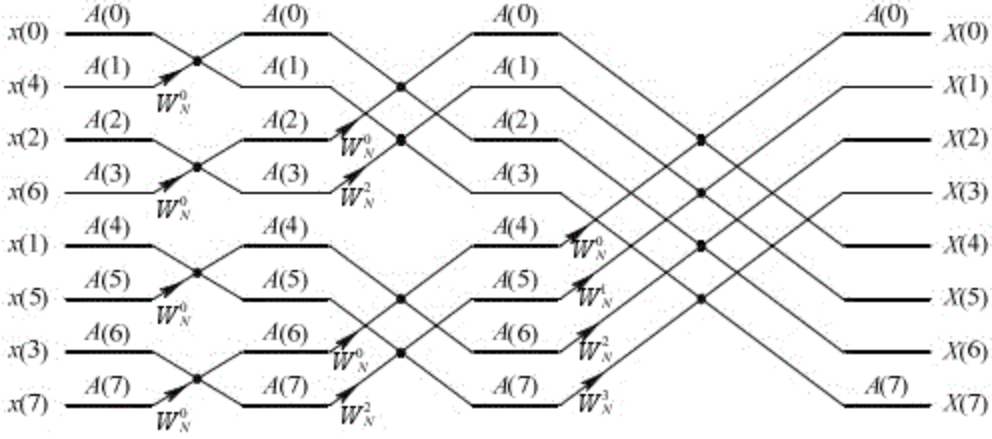

图 3-3 N 点 DIT-FFT 运算流图（N=8）

4 FFT 硬件实现

4.1 设计准备

4.1.1 VHDL 语言简介

VHDL 的英文全名是 Very-High-Speed Integrated Circuit Hardware Description Language,诞生于 1982 年。 1987 年底， VHDL 被 IEEE 和美国国防部确认为标准硬件描述语言。1993 年，IEEE 对 VHDL 进行了修订，从更高的抽象层次和系统描述能力上扩展 VHDL 的内容，公布了新版本的 VHDL，即 IEEE 标准的 1076-1993 版本，(简称 93 版)。现在，VHDL 和 Verilog 作为 IEEE 的工业标准硬件描述语言，又得到众多 EDA 公司的支持，在电子工程领域，已成为事实上的通用硬件描述语言。

4.1.2 可编程器件简介

FPGA 是一类高集成度的可编程逻辑器件，起源于美国的 Xilinx 公司，该公司于 1985 年推出了世界上第一块 FPGA 芯片。从最初的 1200 个可用门，90 年代时几十万个可用门，发展到目前数百万门至上千万门的单片 FPGA 芯片，Xilinx、Altera 等世界顶级厂商已经将 FPGA 器件的集成度提高到一个新的水平。FPGA 结合了微电子技术、电路技术、EDA 技术，使设计者可以集中精力进行所需逻辑功能的设计，缩短设计周期，提高设计质量。

4.2 单蝶形设计方案

FFT 处理系统的总体结构如图 4-1 所示：FFT 处理器被分成以下几个主要功能模块进行设计：蝶形运算单元、数据地址产生单元、控制单元等。此外还需要一些存储单元，如 RAM、ROM。各模块的功能概述如下：

1)数据地址产生单元：产生双口 RAM 和存放旋转因子的 ROM 的地址信息，包括读地址和写地址。

2)蝶形运算单元：采用基 2 时域抽取算法，一次计算两个数据。

3)数据选择单元：对蝶形运算结果的 16Bit 数据进行选择，取其中 9Bit。

4)双口 RAM1:存储输入数据和中间处理数据。

5)双口 RAM2:存储最终结果。

6) ROM:存储旋转因子数据。

7)时序控制单元:实现对各模块的时序控制。

8) PLL:提供系统所需时钟信号。

输入数据通过选择器倒序存储到双口 RAM1 中，倒叙的地址预先存储在 ROM 里，在数据输入时读出并和数据一同送到双口 RAM1，输入数据完成后开始进入各级蝶形运算，由地址产生单元产生蝶形运算所需数据和旋转因子的地址分别从 RAM 和 ROM 中读出数据后，一起送到蝶形运算单元进行蝶形运算，蝶形运算的结果通过数据选择单元，将选出的数据回存到取数据的地址中，完成一次蝶形运算。在 FFT 最后一级蝶形运算中，得到的是最终结果，不必再截取 9 位回存，直接根据前面所作的截位处理进行适当的移位保留结果就行，结果存到双口 AM2 中，蝶形运算完成以后，再将双口 RAM2 中的运算结果顺序输出。

设计总体结构框图如图 4-1：

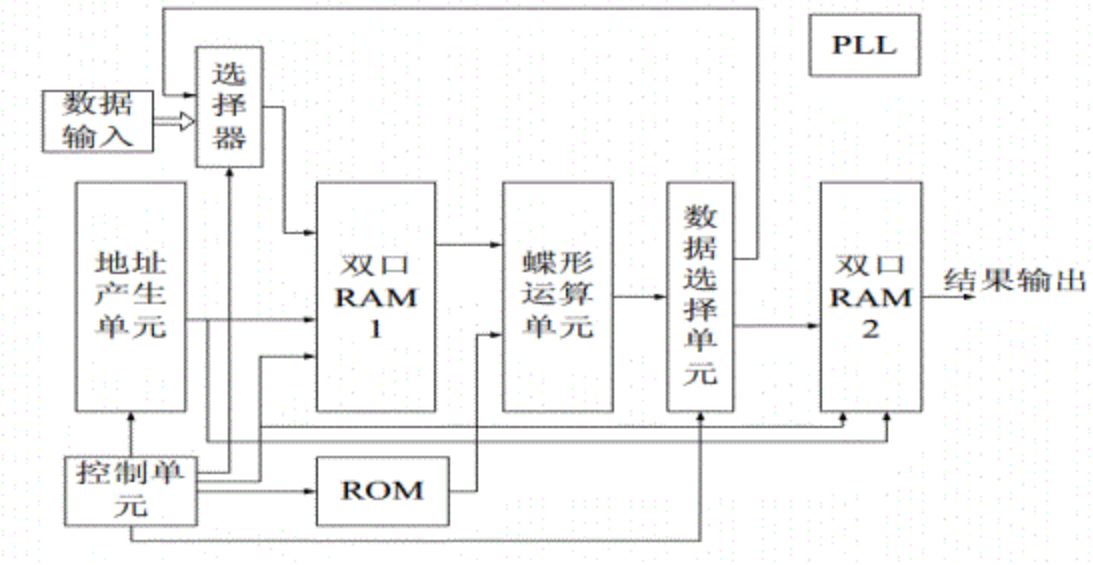

图 4-1 总体结构

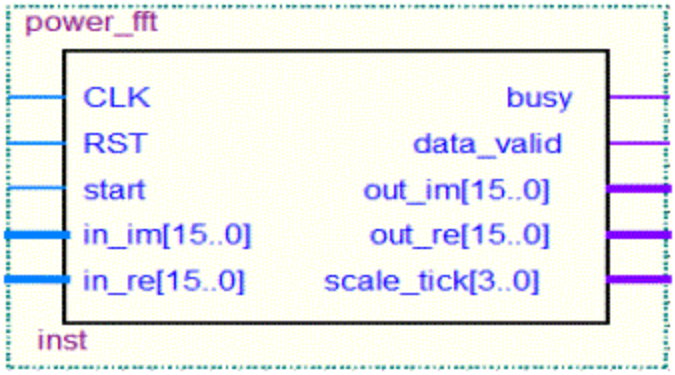

图 4-2 系统实体图

如图 4-2 所示：

CLK：工作时钟输入引脚。

RST：复位引脚，低电平有效，恢复初始状态。

Start： 启动信号，高电平有效，至少维持 2 个时钟。第 2 个时钟开始的同时第一个有效输入数据必须出现在输入端。

In_im[15..0]：复数的虚部，位宽 16 位，但有效位不得超过 14 位。Start 拉高的同时，必须出现第一个有效的虚部数据。

In_re[15..0]：复数的实部，位宽 16 位，但有效位不得超过 14 位。Start 拉高的同时，必须出现第一个有效的实部数据。

Busy：在 Start 生效之后，FFT 运算器将其拉高。表明开始一帧数据的输入，运算与输出。此时不能将 Start 再次拉高，只有 Busy 恢复低电平时，才能将 Start 拉高进行下一帧。

Data_valid：结果输出数据有效标志，高电平有效。

Out_im[15..0]：在 Data_valid 拉高的同时，输出位宽为 16 位的虚部数据。

Out_re[15..0]：在 Data_valid 拉高的同时，输出位宽为 16 位的实部数据。

Scale_tick[3..0]：在 Data_valid 拉高的同时，输出数据结果被衰减的次数，Scale_tick *4 为实际衰减的倍数。

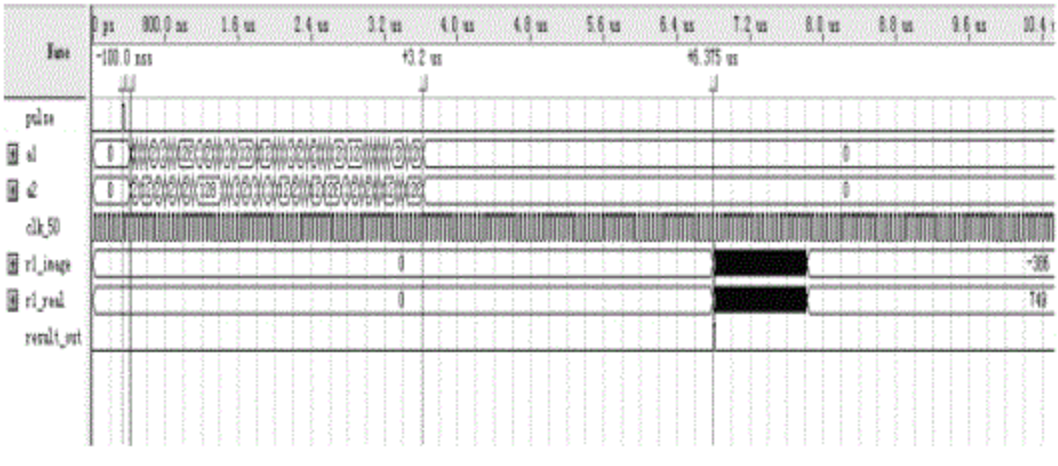

图 4-3 仿真波形

5 总结

FPGA 是 20 世纪 80 年代中期出现的一类新型用户可编程器件。近几年来，这项技术得到了迅速的发展。与一般的可编程逻辑器件不同， FPGA 的集成度高、逻辑实现能力强、设计灵活性更好。现在，FPGA 的容量已经跨过了百万门级，不仅可以解决电子系统小型化、低功耗、高可靠性等问题，而且其开发周期短、芯片价格不断降低，这些因素促使 FPGA 越来越多地取代了 ASIC 甚至 DSP 的市场，成为解决系统级设计的重要选择方案之一。

目前， FPGA 正处于革命性的数字信号处理技术的前沿。过去，前端的可编程数字信号处理算法，例如 FFT、 FIR 和 IIR 滤波器都是用 ASIC 或者 DSP 处理芯片构建的，但现在大多被 FPGA 替代。利用 FPGA 实现数字信号处理，可以达到 DSP 芯片无法达到的速度，而且在处理大规模数据方面，有极大的优势;当然它也有不足之处，比如不能采用浮点的计算方式，对一些比较复杂的算法，很难用硬件语言描述，但不论怎样，利用 FPGA 进行数字信号处理是一种趋势。

本文正是基于 FPGA 实现 FFT 这一在数字信号处理中有着广泛应用的算法。经过一年的努力，本次设计任务已圆满完成。在这次毕业设计中，我所完成的工作主要有以下几项:首先，细致地分析了 FFT 算法的算法结构以及其中相关计算数据的流动特点，并在此基础之上进行整个 FFT 处理系统的电路构思。从系统级上完成了各个子电路功能模块的划分。其次，深入地学习了大规模可编程集成电路 FPGA 的知识，弄清楚了 FPGA 的物理特点及其包含的硬件资源的数量和特点。确定了在进行电路设计时将要使用到的硬件资源类型。再次，对支持系统级行为描述的 VHDL 语言的设计方法和设计思想进行了深入的学习和理解。

第四，认真地学习了 FPGA 的开发软件的功能特点和具体的操作使用方法。研究了如何利用软件来快速、有效地实现硬件电路设计的方法。最后，使用 quartus 软件完成了整个 FFT 处理器的电路设计、实现、仿真、验证，电路功

能正确无误。在达到了一定的数据精度上，满足了系统的设计要求。但也存在一些问题：

1)对后仿真中的一些约束条件不是十分熟悉，基本依靠计算机自动布局布线，这样可能会导致一些错误。

2)测试激励采用波形输入的手动方法，没有采用硬件语言编写测试文件的方法，这样不但费时费力，而且容易出错。

参考文献

[1] 于效宇.基于 FPGA 的 FFT 处理器的实现[D].[哈尔滨理工大学硕士学位论文],2005:28-30.

[2] 植强.一种基于 FPGA 的 FFT 阵列处理器[J].电子对抗技术,2002,17(6):36-39.

[3] 刘国栋,陈伯孝,陈多芳.FFT 处理器的 FPGA 设计[J].航空计算技术,2004,31(3):101-104.

[4] 王诚,吾继华,范丽珍,等.Alter FPGA/CPLD 设计(基础篇)[C].北京:人民邮电社.2005:96-108.

[5] 陈丽安,张培铭.定点 DSP 块定点算法及其实现技术[N].福州大学学报(自然科学版),2004,32（6）：689-693.

[6] Zhang Qi,Yi Qingming.A New Base-6 FFT Algorithm.Semiconductor Photonics and Technology,2002,9(1):23-25.

[7] Currie S M. Implementation of a Single Chip,Pipelined,Complex,One-Dimensional Fast Fourier Transform in 0.25um Bulk CMOS. Application-Specific Systems,Architectures and Processors,2002.Proceeding.The IEEE International Conferenceon,17-19 July 2002:335-343.

[8] Bidet E.A Fast Single Chip Implementation of 8192 Complex Point FFT.Solid-StateCircuits, IEEE Journal of,Volume:30,Issue:3,March 1995:300-305.

[9] Bi B,Joney E.V.A Pipelined FFT Processor for Word Sequential Data.Acoustics,Speech,and Signal Processing,IEEE Transactions,Volume:37,Issue:12,Dec.1989:1982-1985.

[10] Gold B,Bially T.Parallelism in Fast Fourier Transform Hardware.Audio andElectoacoustics, IEEE Transactions on,Volume:21,Issue:1,Feb 1973:5-16.

[11] Lenart T,Owall V.A 2048 Complex Point FFT Processor Using A Novel Data Scaling Approach .Circuits and Systems,2003.ISCAS'03.Proceeding of the 2003 International Symposium on,Volume:4,25-28 May 2003:IV45-IV48.

[12] 唐江,刘桥.基于 FPGA 的基-4FFT 算法的硬件实现[N].重庆工学院报,2007,21(3):82-84.

[13] 潘明海,刘英哲,于维双.一种基于 FPGA 实现的 FFT 结构.微计算机信息 2005,21（9-2）:156-158

[14] Jessani R M,Pulrino M.Comparison of signal-and dual-pass multiply-add fused
floating-point units[J].IEEE Trans Compute,1998,47(9):927-937

[15] 任淑艳,关丛荣,杨永刚,等.应用 VHDL 语言的 FFT 算法实现[N].哈尔滨理工大学学报 2005,8(6):24-26

附录

程序:

```
module fft_16(xn_r,xn_i,RST,CLK,START,OUT,Xk_r,Xk_i);

input  [15:0] xn_r,xn_i;          //输入的实部与虚部
input  RST,CLK,START;
                                  //FFT 启动信号与时钟信号和复位信号
output [15:0] Xk_r,Xk_i;          //FFT 输出实部与虚部
output OUT;                       //输出标志信号

reg    [15:0] Xk_r,Xk_i;
reg    OUT;
reg    OUT1,STRT1;      //级联 FFT4 的输出标志和启动信号
reg    [2:0] k,j,m,n,l,p;  //循环指针
reg    [4:0] i;

reg    [15:0] IN_r[15:0],IN_i[15:0];        //存储输入的实部与虚部
reg    [15:0] OUT_r[15:0],OUT_i[15:0];      //存储输出的实部与虚部
reg    [15:0] TRANIN_r,TRANIN_i;            //输入序列与 FFT4 输入之间
的转接
reg    [15:0] TRANOUT_r,TRANOUT_i;          //输出序列与 FFT4 输出之间
的转接
reg    [2:0]  state;

parameter   Idle=3'b000,                     //空闲
            Input=3'b001,                    //输入
            Compute0=3'b010,                 //计算引擎
            Compute1=3'b100,                 //第二级计算的控制
            Butfly=3'b101,                   //蝶形计算
            Output=3'b110;                   //输出
//定义三个移位函数
function[15:0] Shift03;//乘以 0.3827
input [15:0] xn;
begin
       Shift03={xn[15],xn[15:1]}- {xn[15], xn[15], xn[15],xn[15:3]}
              + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15],xn[15:7]}
              - {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15], xn[15],  xn[15], xn[15], xn[15], xn[15], xn[15],xn[15:13]};
end
endfunction
```

```
function[15:0] Shift07;//乘以 0.7071
input [15:0] xn;

begin
      Shift07={xn[15],xn[15:1]}+ {xn[15], xn[15], xn[15],xn[15:3]}
             + {xn[15], xn[15], xn[15], xn[15], xn[15:4]}
             + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15:6]}
             + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
             xn[15], xn[15],xn[15:8]}+{xn[15], xn[15], xn[15], xn[15],

             xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
             xn[15], xn[15], xn[15], xn[15],xn[15:14]};
end
endfunction

function[15:0] Shift09;//乘以 0.9239
input [15:0] xn;
begin
       Shift09=xn -{xn[15], xn[15], xn[15], xn[15],xn[15:4]}
              - {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15:6]} + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15], xn[15],xn[15:9]};
end
endfunction

//调用 FFT4 模块计算 4 组 4 点的 FFT
//FFT4
XFFT(.CLK(CLK),.START(STRT1),.xn_r(TRANIN_r),.xn_i(TRANIN_i),.OUT(OUT
1),.Xk_r(TRANOUT_r),.Xk_i(TRANOUT_i));

always @(posedge CLK or posedge RST)
       if(RST)
           begin
             state<=Idle;     //异步复位，高电平有效
             OUT<=0;          //输出无效
           end
       else
        begin
         case (state)

              //空闲状态
              Idle: begin
```

```
          OUT<=0;
          if (START) //只有在空闲状态下才能启动 FFT
             begin
               state<=Input;
               i<=0;
             end
          else
             begin
               state<=Idle;
             end
        end

 //16 位复数输入
 Input: begin
          if(i<16)
            begin
              IN_r[i]<=xn_r;
              IN_i[i]<=xn_i;
              i<=i+1;
            end
          else if(i==16) //输入完毕进入的计算
            begin
              state<=Compute0;
              STRT1<=1;//启动 FFT4
              j<=0;    //指针复位
              k<=0;
              m<=0;
              n<=0;
            end
        end

//第一二级计算
Compute0: begin
           if(j<4) //4 组串行输入计算
             begin
               STRT1<=0;//关断 FFT4 启动信号
               if(k<4)
                 begin
                   TRANIN_r<=IN_r[4*k+j];  //时选法输入
                   TRANIN_i<=IN_i[4*k+j];
                   k<=k+1;
                 end
               else if(k==4)//k=4 输入等待 FFT4 的输出
                 begin
```

```
                        if((OUT1==1)&&(m<4)) //可以输出
                          begin       //这两个信号同时消失
                         OUT_r[4*j+m]<=TRANOUT_r;//连续存储
                            OUT_i[4*j+m]<=TRANOUT_i;
                            m<=m+1;
                          end
                        else
                              begin
                            if(m==4) //输出完毕，并非没有开
始
                             begin
                              j<=j+1;   //进入下一组的计算
                               m<=0;  //清零，以备用
                              k<=0;
                          STRT1<=1;//为下一个计算作准备启动信号
                                 end
                               end//输出完毕
                      end
                   end
                else if(j==4)
                   begin
                     if(n==0)
                        begin
                         state<=Butfly;//第一级完进入蝶形计算
                          n<=n+1;
                        end
                     else if(n==3)
                            begin
                               state<=Output;  //第二级完毕
                               l<=0;
                               p<=0;
                            end
                   end
                end

          Butfly:  begin
                    if(n==1)
                      begin
                       IN_r[0]<=OUT_r[0];
                       IN_i[0]<=OUT_i[0];
                       IN_r[1]<=OUT_r[1];
                       IN_i[1]<=OUT_i[1];
                       IN_r[2]<=OUT_r[2];
                       IN_i[2]<=OUT_i[2];
```

```
                            IN_r[3]<=OUT_r[3];
                            IN_i[3]<=OUT_i[3];
                            IN_r[4]<=OUT_r[4];
                            IN_i[4]<=OUT_i[4];

IN_r[5]<=Shift09(OUT_r[5])+Shift03(OUT_i[5]);
                            IN_i[5]<=Shift09(OUT_i[5])-
Shift03(OUT_r[5]);

IN_r[6]<=Shift07(OUT_r[6])+Shift07(OUT_i[6]);
                            IN_i[6]<=Shift07(OUT_i[6])-
Shift07(OUT_r[6]);

IN_r[7]<=Shift03(OUT_r[7])+Shift09(OUT_i[7]);
                            IN_i[7]<=Shift03(OUT_i[7])-
Shift09(OUT_r[7]);
                            IN_r[8]<=OUT_r[8];
                            IN_i[8]<=OUT_i[8];

IN_r[9]<=Shift07(OUT_r[9])+Shift07(OUT_i[9]);
                            IN_i[9]<=Shift07(OUT_i[9])-
Shift07(OUT_r[9]);
                            IN_r[10]<=OUT_i[10];
                            IN_i[10]<=0-OUT_r[10];
                            IN_r[11]<=Shift07(OUT_i[11])-
Shift07(OUT_r[11]);
                            IN_i[11]<=0-Shift07(OUT_i[11])-
Shift07(OUT_r[11]);
                            IN_r[12]<=OUT_r[12];
                            IN_i[12]<=OUT_i[12];

IN_r[13]<=Shift03(OUT_r[13])+Shift09(OUT_i[13]);
                            IN_i[13]<=Shift03(OUT_i[13])-
Shift09(OUT_r[13]);
                            IN_r[14]<=Shift07(OUT_i[14])-
Shift07(OUT_r[14]);
                            IN_i[14]<=0-Shift07(OUT_i[14])-
Shift07(OUT_r[14]);
                            IN_r[15]<=0-Shift09(OUT_r[15])-
Shift03(OUT_i[15]);
                            IN_i[15]<=Shift03(OUT_r[15])-
Shift09(OUT_i[15]);
                             n<=n+1;
                           end
```

```
                    else
                      begin
                        if(n==2)
                           state<=Compute1; //进入第二级计算
                      end
                   end

            Compute1: begin
                        n<=3;
                        j<=0;
                        STRT1<=1;
                        state<=Compute0;
                      end

             Output: begin//倒序输出
                       if(l<4)
                          begin
                               OUT<=1;
                               Xk_r<=OUT_r[4*p+l];
                               Xk_i<=OUT_i[4*p+l];
                               p<=(p==3)?0:(p+l);
                               l<=(p==3)?(l+1):l;
                          end
                       else if(l==4) //输出完毕
                          begin
                               OUT<=0;
                               state<=Idle;
                          end
                     end

              default: begin
                          state<=Idle;
                       end
         endcase
        end
endmodule
```

课程设计成绩评定表

评定项目	内　　容	满分	评分	总分
学习态度	学习认真，态度端正，遵守纪律。	10		
答疑和设计情况	认真查阅资料，勤学好问，提出的问题有一定的深度，分析解决问题的能力较强。	40		
说明书质量	设计方案正确、表达清楚；设计思路、实验（论证）方法科学合理；达到课程设计任务书规定的要求；图、表、文字表达准确规范，上交及时。	40		
回答问题情况	回答问题准确，基本概念清楚，有理有据，有一定深度。	10		
总成绩	采用五级分制：优、良、中、及格、不及格			

指导教师评语：

签名：

年　月　日