北京交通大学

毕业设计（论文）

中文题目：OFDM 通信系统基带数据处理部分的 FPGA 实现

英文题目：Implementing Baseband Data Processing Section on FPGA of an OFDM-based Communication System

学　　院：电子信息工程学院

专　　业：通信工程

学生姓名：李　想

学　　号：02211070

指导教师：陶　成

2006 年 6 月 1 日

北京交通大学毕业设计（论文）成绩评议

题　目：OFDM 通信系统基带数据处理部分的 FPGA 实现

学　院：电子信息工程学院　　专业：通信工程

学生姓名：李 想　　学号：02211070

指导教师建议成绩：

评阅教师建议成绩：

答辩小组建议成绩：

答辩委员会意见：

最终成绩：

主管教学副院长或答辩委员会主席签字：

年　　月　　日

北京交通大学毕业设计（论文）任务书

题 目：OFDM 通信系统的设计与实现

适合专业：通信工程

指导教师（签名）：　　提交日期：2006 年 3 月 1 日

学院：电子信息工程学院　　专业：通信工程

学生姓名：李想　　学号：02211070

毕业设计（论文）基本内容和要求：

1、熟悉通信相关方面的知识，学习并掌握 OFDM 技术的原理。

2、熟悉 VHDL 语言，使用该语言进行数字电路（FPGA）设计。

3、设计并实现 OFDM 通信系统的调制、解调部分的数字电路。

4、采用实验板或自行设计电路进行调试，并采用相关仪器验证。

5、系统整体调试、优化，或就某一部分进行深入研究。

毕业设计（论文）重点研究的问题：

1、学习并掌握 OFDM 技术。

2、OFDM 调制、解调部分的工程设计与实现。

北京交通大学毕业设计（论文）任务书

毕业设计（论文）应完成的工作：

1、查阅相关的中英文文献资料。

2、完成调制、解调软件部分的编写和仿真。

3、实现 OFDM 硬件部分的设计，并进行调试与验证。

参考资料推荐：

Shinsuke Hara and Ramjee Prased, MULTICARRIER TECHNIQUES for 4G Mobile COMMUNICATIONS, MA: Artech House, 2003.

其他要说明的问题：

北京交通大学毕业设计（论文）开题报告

题 目：OFDM通信系统的设计与实现

学院：电子信息工程学院　　专业：通信工程

学生姓名：李　想　　学号：02211070

文献综述：

一、毕设题目背景

1、国内外的研究现状

OFDM 作为一种多载波的调制或者复用技术，近几年来得到了国际上的广泛关注。它采用正交技术以充分利用频谱的思想，是科学且符合通信技术发展方向的。随着电子技术和集成电路的发展，OFDM 技术已经开始崭露头角，成为了一些宽带数据通信的标准，比如 DAB 和 HDTV，以及比较知名的无线局域网（Wireless LAN i.e. IEEE802.11）等。从 20 世纪 60 年代 OFDM 技术的提出至今，在基本的理论上已经很成熟了，并且被预言为 3G 通信之后的主流复用技术[1]，具有很高的研究价值。

国内外关于 OFDM 的研究方向大致可分为这样几个方面：一个是在理论上的研究，以期能够进一步发挥 OFDM 的价值，提高系统性能，并指导工程实践，比如动态选择子信道技术，用于降低噪声的干扰；另一个方面是研究 OFDM 与其它技术的结合，比如与扩频调制技术或者 CDMA 技术结合等；还有就是纯粹意义上的 OFDM 如何实现，在硬件上如何做到最优化。

但是对于许多国内高等院校的大学本科学生来说，OFDM 还是一项高深的技术，即使是通信专业的学生也很难在四年中接触到它。在研究生阶段才有 OFDM 的课程，并且基本上停留在理论上，几乎没有硬件方面的设计与实现。但是在台湾省一些大学的 OFDM 课程上，已经把实践提高与理论同等重要的高度上了，在讲述理论课的同时即要求完成硬件部分[2]。我认为这对于学习工科的学生来说是很重要的。

2、毕设题目的研究意义与价值

幸运的，由于参加学院 SBH 的项目，我在大三年级接触到了 OFDM。但是因为种种原因，最终没能完成原定的目标。但是通过这段时间了努力，使我对 OFDM 有了一定的了解，并产生了很强的兴趣。我希望能通过毕设的形式，完成这样一个对自己来说很有挑战的题目。

相比国际研究领域的前沿，自己的能力肯定达不到那样的高度，也做不出那种

水平的东西。所以，我毕设的最终目标是完成一个普通的示意性的实现 OFDM 功能的通信系统。但是我认为，选这个题目还是有意义和价值的。

首先，对于通信专业的学生来说，能够完成一个 OFDM 的通信系统，过程很重要。在这个过程中，能够更好了理解 OFDM 技术，并最终实现一个完整的系统；第二，这是一项重要的考验，检验自己四年来学习的知识和能力；第三，OFDM 项目的实现目前来说在本科生范围内还是相对比较超前的，我在这个过程中积累的一些经验也许会对以后的研究甚至教学工作有所帮助。第四，在这个系统中，如果一些具体的数字模块（比如 FFT 等）的设计能够完成的话，对其它的一些项目中也有帮助，因为可以参考或直接拿去使用。

二、毕设题目主要内容

1、系统框架

因为 OFDM 的基本理论已经很成熟了，所以通常的 OFDM 系统框架图已经是确定的了，基本大同小异。系统框架图是指导我进行设计的主要依据。如下图所示：

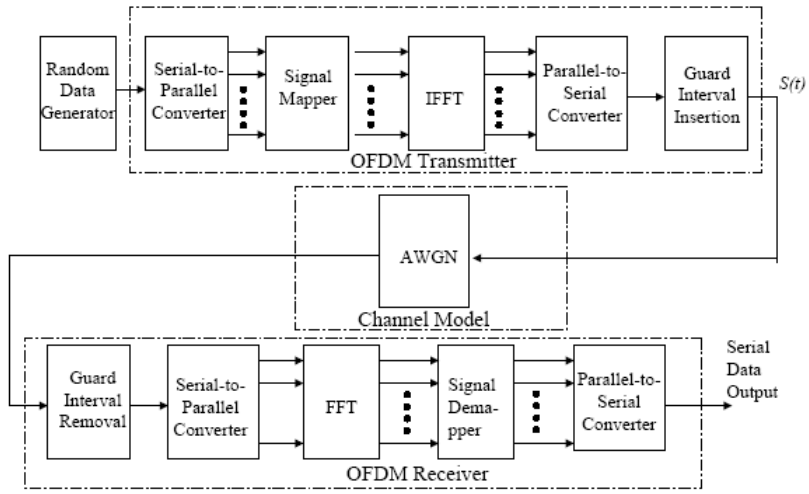


图1　OFDM 通信系统框图

2、系统实现的关键技术方案

下面分别介绍通信系统中的主要内容和一些关键技术的设计实现方案。

2.1、FFT 方案

设计一个 FFT 的硬件有多种方案，包括采用 DSP 芯片等，但是使用 FPGA 实现是其中速度最快的[3]。FFT 的 FPGA 实现现在已经并非一件很困难的事情了，可以很容易找到关于 FFT 的 VHDL 代码。另外也可以使用一些公司的 IP 核。ALTERA 公司就有关于 FFT 的 IP。

所以我设想 FFT 的方案大致有两种：首先是使用 ALTERA 的 IP，这种方法相

对简单，但是需要学习如何使用；另外是自己写一个 FFT，相对稍微复杂一点，需要对 VHDL 和器件的特点有所掌握。不过 FFT 可以写简单一点，比如我想可以写一个 8 点的 FFT，应用在一个示意性的系统中。

2.2、同步

同步是 OFDM 中的最重要的组成部分之一。由于 OFDM 对于频率的偏移十分敏感，因此同步的实现就非常重要。相对于 FFT 的实现来说，对于同步如何实现，我觉得自己还没有太多感性的认识，只能想到在一帧中采用加入同步数字序列的方法。同步的实现将会是我下个阶段的研究重点。

2.3、传输方案

当数字的调制和接收部分完成后，采用什么样的传输方案就显得很重要。其实选择传输方案还需要考虑信道的参数和特点等等。但是由于自己的水平有限，我计划采用相对简单的传输方案，有以下几种。

第一种方法是采用导线连接直接进行数字传输。这是最简单的办法，如果在 QuartusII 的时域仿真通过的话，那基本上就可以肯定这种方案一定会成功。虽然这样子就无法观查到频谱的正交性，使得精力全部投入在数字模块的调制和解调方面，最终完成的是一个 OFDM 的 Modem，但是，这不失为一个最保险的简单方案。

另一种方法是采用无线的数字收发模块。现在市场上有一些这类的模块。但是其实意义不大。因为它只不过是把第一种的数字传输从有线变成无线，还是与 OFDM 关系不大。因为基本上不准备考虑这种方法。

第三种方法是标准的 D/A 方法，在输出端加上 D/A 器件输出模拟波形，这样可以观察到正交的 OFDM 频谱。但是这种方法相对更复杂，即使是采用有线传输，在接收端也需要再进行 A/D 转换。

2.4、PCB 板

最终实现系统的硬件部分需要设计一个 PCB 板。对我来说这是一个难点，以前我没有设计 PCB 的经验。考虑的 PCB 板的制作周期长，花费较高，因此希望在这方面能够得到老师的指导。

2.5、系统接口

如果系统完成的顺利的话，可以考虑设计通信系统的接口，使整个系统更具应用性。比如设计一个单片机做控制发送字符串，或者采用串口与电脑相连并制作一个 Windows 程序收发内容等[4]。

三、毕设现有基础

1、现有基础及寒假进展

在之前的 SBH 项目中和寒假休息的时候，我也为毕业设计做了一些相关的准备。

在理论方面，主要参考了一本关于 OFDM 技术的 MULTICARRIER TECHNIQUES for 4G Mobile COMMUNICATIONS 的书，以及其它一些材料，使我进一步加深了对于理论方面的了解。现在虽然自己无法独立推导理论，但是达到了可以看懂并理解参考资料的程度。

在寒假中我对 FFT 的实现进行了尝试，主要是学习 ALTERA 公司的 IP Core。现在把手册等材料看完，并已经基本学会使用这个 Core。但是在进行仿真的时候，IFFT+FFT 的数据总还是不对，所以如果应用到实际中也还需要再学习。

在具体的实践方面，由于参加了去年的大学生电子设计竞赛，积累了一些硬件的经验。另外对于 ALTERA 器件的应用也有一些了解。

2、所需的研究条件和仪器

在数字器件方面决定选用 ALTERA 的 FPGA 器件，现在一些 Cyclone 的器件价格不算太高，而且能在中发买到，这是一个很方便的条件。

在设计 FPGA 所应用的 Quartus II 软件方面，ALTERA 网站提供免费的网络版，虽然有些功能不能用，但是对于现有的设计已经足够了。我现在使用 Quartus II 5.1sp1 网络版[5]。

在 IP Core 方面，ALTERA 也提供了更大的便利。正在 ALTERA 的一些 IP Core 加入了新的特性，可以免 License 进行下载。虽然下载后有使用时间上的限制（一小时），但是对于实验室研究已经足够了[5]。因此我考虑在可能的情况下更多采用 ALTERA 的 IP。

MATLAB 软件在系统模拟、分析和仿真的时候有很大的用处。现在我还没有可用的正版 MATLAB，但是如果毕设完成顺利，时间还比较充裕的话，可能会需要它做一些理论上的研究。

OFDM 的波形发出后需要用频谱仪来观察，因此希望能够使用实验室的频谱仪。

制作硬件需要一些研究资金购买器件，我可以自己负担。但是希望在这方面能够得到老师的指导，减少不必要的浪费。

在毕设的过程中，我希望能够得到研究生师兄和老师更多的指导，通过自己的努力和师兄们积累的经验，使我能够更快达到他们原来的水平，并能在此基础上有所提高和创新。

四、毕设题目的目标

将毕设题目的具体内容细化，可以把 OFDM 的题目分成一些子题目。又可以分为基础部分与研究部分两大块。基础部分是指最基本的 OFDM 调制解调部分，这是成功完成毕设的底线，至少应该实现的部分。如果顺利的话，还可以利用时间做一些深入的学习与创新，比如对 OFDM 的一些重要的问题（如同步和峰均比等）做些专门的研究与设计[6]。毕竟 OFDM 技术在毕设阶段是无止境的。

1、毕设基本目标

毕设的基本目标是完成 OFDM 题目最基本的目标，也是相对较简单的。如果能够顺利完成了话还可以继续做一些扩展。

1.1、OFDM 调制解调部分

包括所有的调制解调的数字部分：FFT、编码、映射、同步，以及反变换的部分。主要采用 VHDL 及 Quartus II 软件。采用软件仿真。

1.2、A/D、D/A 设计

选择合适型号的 A/D、D/A 器件，设计模拟电路。

1.3、硬件制作调试

完成 PCB 设计，实现硬件电路，并检验结果。

1.4、毕设报告与演示部分

2、毕设扩展部分

2.1、传输接口设计制作

传输接口部分是为了使 OFDM 通信系统更有实际作用，能够传输一些字符之类等等，也能使操作更方便。

2.2、收发不同时钟处理

OFDM 一个很重要的课题就是同步的问题，如何把同步做得更好并能应付时钟偏差是一个值得研究的问题。

2.3、峰均比处理

OFDM 另一个课题就是由于正交性导致的峰均比的问题，也值得深入研究，可以通过编码的方案改进。

2.4、MATLAB 建模与理论分析

如果有时间可以用 MATLAB 做一些 OFDM 的仿真，更好的与理论相结合，并能分析已有的系统。

2.5、动态选择分配子信道

如果能够实现动态分配子信道，可以更好地抵抗噪声的干扰。通过检测出误码率高的子信道，然后将数据传输安排到其它信道上去。

主要参考文献：

[1] Shinsuke Hara and Ramjee Prased, MULTICARRIER TECHNIQUES for 4G Mobile COMMUNICATIONS, MA: Artech House, 2003.
[2] 中国台湾国立中正大学电机工程学系，正交分频多重进接技术研究所教学课程课件，http://www.ee.ccu.edu.tw/~wl/ofdm/OFDMopendata.htm
[3] 潘松、黄继业、王国栋编著，现代 DSP 技术，西安电子科技大学出版社，2003 年
[4] 龚建伟、熊光明编著，Visual C++/Turbo C 串口通信编程实践，电子工业出版社，2004 年
[5] www.altera.com
[6] 王文博、郑侃编著，宽带无线通信 OFDM 技术，人民邮电出版社，2003 年

研究方案与计划：

根据目前的进度和目标，我设计了三种方案。当然，很可能计划得还不充分，需要根据下一阶段的进度进行再调整。

1、方案一：（正常的计划）

二月到三月底：OFDM 硬件电路的软件完成并通过仿真。其中 FFT 一周时间，编码一周，同步两周，剩余时间给串并变换及其它部分。

四月：完成 A/D、D/A 以及硬件电路。其中 A/D、D/A 选型测试一周，PCB 设计一周，硬件电路的调试与完成两周。

五月：做一些专题扩展。选一个毕设扩展部分的子题目去做。

六月：完成毕设报告、演示与答辩。

2、方案二：（比较顺利的计划）

二月到三月上旬：OFDM 硬件电路的软件完成并通过仿真。

到三月底：完成 A/D、D/A 以及硬件电路。

四、五月：更多地做一些专题扩展。

六月：完成毕设报告、演示与答辩。

3、方案三：（完成毕设最低的计划）

二月到三月底：OFDM 硬件电路的软件完成并通过仿真。

四月和五月：完成 A/D、D/A 以及硬件电路。

六月：完成毕设报告、演示与答辩。

北京交通大学毕业设计（论文）开题报告

毕业设计（论文）进度安排：（正常进展情况下的时间安排）

序号	毕业设计（论文）各阶段内容	时间安排	备注
1	OFDM 硬件电路的软件完成并通过仿真。其中 FFT 一周时间，编码一周，同步两周，剩余时间给串并变换及其它部分。	二月到三月底	
2	完成 A/D、D/A 以及硬件电路。其中 A/D、D/A 选型测试一周，PCB 设计一周，硬件电路的调试与完成两周。	四月	
3	做一些专题扩展。选一个毕设扩展部分的子题目去做。	五月	
4	完成毕设报告、演示与答辩。	六月	

指导教师意见：

指导教师签名：_____　　审核日期：_____年____月____日

北京交通大学毕业设计（论文）指导教师评阅意见

题　目：OFDM通信系统基带数据处理部分的FPGA实现

学　院：电子信息工程学院　　专业：通信工程

学生姓名：李　想　　学号：02211070

毕业设计（论文）完成情况(包括设计图纸、说明书、实验报告、计算机软硬件、外文翻译及摘要、论文书写及规范化等)评价（50 分）：
毕业设计（论文）成果质量评价意见（30 分）：
学生工作态度和考勤情况评价（10 分）：
开题报告的评定成绩（10 分）：

总成绩：_____指导教师 (签名)：_____日期：_____年____月____日

北京交通大学毕业设计（论文）评阅教师评阅意见

题　目：OFDM通信系统基带数据处理部分的FPGA实现

学　院：电子信息工程学院　　专业：通信工程

学生姓名：李　想　　学号：02211070

毕业设计（论文）完成情况评价(包括设计图纸、说明书、实验报告、计算机软硬件、外文翻译及摘要、论文书写及规范化等)（50 分）：
毕业设计（论文）成果质量评价意见（40 分）：
开题报告评价意见（10 分）：

评定成绩：＿＿＿＿评阅人：＿＿＿＿　日期：＿＿年＿＿月＿＿日

北京交通大学毕业设计（论文）答辩小组评议意见

题　目：OFDM通信系统基带数据处理部分的FPGA实现

学　院：电子信息工程学院　　专业：通信工程

学生姓名：李　想　　学号：02211070

毕业设计（论文）完成情况和成果质量（工作量、任务难度、专业理论的运用、综合运用能力、资料的充足与可信情况、成果水平）评价意见（80 分）：
答辩表现评价意见（20 分）：

评定成绩：　　答辩组长：　　日期：　　年　　月　　日

中文摘要

正交频分复用（OFDM，Orthogonal Frequency Division Multiplexing）是当前一种非常热门的通信技术。它即可以被看作是一种调制技术，也可以被看作是一种复用技术。由于它具有抗多径衰落和频谱利用率高的特点，因此被广泛应用于高速数字通信领域，比如应用于 IEEE 802.11a 无线局域网（WLAN）的物理层等等。

我的毕业设计的核心任务是：采用 FPGA 来实现一个基于 OFDM 技术的通信系统中的基带数据处理部分，即调制解调器。其中发射部分的调制器包括：信道编码（Reed-Solomon 编码），交织，星座映射，FFT 和插入循环前缀等模块。我另外制作了相应的解调器，可以实现上述功能的逆变换。

另外，我还对 OFDM 技术，IEEE 802.11a 的标准文献，基于 Simulink 的 OFDM 模型和仿真，ALTERA 公司的技术和 IP Core 的使用等方面进行了研究。这些在文章中都有体现。

关键词：OFDM, FPGA, ALTERA

Abstract

Because of wireless environment where multipath maybe significant, Orthogonal Frequency Division Multiplexing (OFDM), a special form of multicarrier modulation (MCM), where a single data stream is transmitted over a number of lower rate subcarriers has recently received considerable attention for its robustness to multipath selective fading and high bandwidth efficiency. It can be seen as either a modulation technique or a multiplexing technique.

The main work of my graduate design is to implement baseband data processing section on FPGA of an OFDM-based communication system. It contains Reed-Solomon channel coding (FEC), interleaver, constellation, FFT and Prefix Cyclic parts.

In addition, I also pay much attention to other aspects during the design. That is, the study of OFDM, IEEE 802.11a Standard, a demo model of OFDM based on Simulink, devices and IP Megacore of ALTERA corp., which are detailed in my paper.

Key words: OFDM, FPGA, ALTERA

目　　录

前　言

四年的大学时光很快就进入到了最后一个学期，我将在这个学期完成自己的毕业设计。在我看来，毕设对任何一个毕业生来说都应该是非常重要的。因为毕设需要通过自己四年来学到的知识来解决一个实际的问题，这是对自己这四年来综合能力的考验。毕设的水平也就代表了毕业生的水平，只有出色完成了毕设的人才能算做一个合格的毕业生。

正是有这样的思想，我非常珍惜这个考验自己的机会，花费了近半年的时间，认真地做好毕设中的每一个工作。尽管最后看来，毕设的作品仍然显得很简单，但是可以很负责地说，我已经尽到了自己的最大努力。

非常感谢我的指导教师，陶成老师。在他组织的对本科生开展的科研项目中，我有机会在大三的时候接触到了 OFDM 这种前沿的通信技术。虽然因为当时我的时间和能力都有限，这个项目没有完成，但是它大大开阔了我的眼界。所以借着毕设的机会，我决定把这个题目做完。

应该说，OFDM 技术的实现还是非常有难度的，甚至对于一个研究生来说，也需要很长时间才能完成。但是对于学习通信专业的我来说，实现一个有难度的通信系统正是一个绝佳的考验自己的机会。所以我毫不犹豫地就选择了这个题目。

在完成这个题目的过程中，我遇到了许多的困难，也走了许多的弯路。但是现在回头再看看整个过程，正是在不断克服困难的同时，我也学到了许多新的知识，提高了自己的能力。这些克服困难的过程对于我来说是一种宝贵的财富。

我最初的目标只是研究 OFDM 技术，并计划在掌握 OFDM 的原理后在 FPGA 上实现一个示意性的 OFDM 通信系统。之所以是示意性的，是因为我计划用最简单的参数来实现一个系统，而不是设计一个具有实际应用

能力的系统。关于 OFDM 这种技术本身，我参考了不少的资料，也有一定的收获，对于 OFDM 技术的介绍，我写在了第一章中。

开始的时候，我其实自己也不太清楚自己的目标究竟是什么。只是单纯地想要实现一个“ OFDM 系统 ”。但是随着对 OFDM 技术理解的不断深入，我也对自己工作的具体目标有了一个明确地定位，那就是用 FPGA 实现基于 OFDM 技术的通信系统中基带部分的数据处理功能。所以我又对最初的毕设题目进行了修改，变成了现在的这个题目，以期能够更好的反映出我所做的内容。

在毕设的过程中，另一个给我触动很大的是我查阅到了有关 IEEE 802.11a 的标准。以前我也知道这个 WLAN 的标准是基于 OFDM 技术的，但是我从没想过可以从一开始就完全按照这个标准来设计一个系统，这样具有更大的实际价值。等到我再想按照 802.11a 来修改我的设计时，时间已经不够了。为此我觉得很遗憾，也许这是我经验不丰富的一个表现吧。所以我最终设计的 OFDM 系统还是只具有示意的性质，并不符合某项标准。但是我仍然认真学习了一下 802.11a 这个标准，关于这些内容我写在了第二章中。

我之所以是采用 FPGA 器件来实现 OFDM 技术，而不是 DSP 芯片或其它的器件，是因为 FPGA 的出色性能。对于 FPGA 的性能，我对 ALTERA 公司的器件和技术比较熟悉，所以在第三章中，我介绍了 FPGA 的优势和 ALTERA 公司的技术概况。

在第四章中，我详细地介绍了 ALTERA FFT MegaCore 的使用。因为我采用了基于 IP 设计数字系统的理念，所以 OFDM 的核心部分，FFT 模块，我采用了这个 IP 来实现。在此，我必须要对 ALTERA 的工程师 Dylan 表示感谢，他为我解答了许多的问题，使我弄清了所有对这个 IP 的问题。非常感谢，Dylan。

在论文的最后一章，我具体地介绍自己设计的系统的技术细节。我并没有拘泥于介绍代码本身，而是重点介绍了每个模块的功能。

我摘出了一些 VHDL 代码放在了附录中。另外放在附录中的还有毕设要求翻译的科技文献的原文和译文。

我很重视网络的作用，所以我毕设的进展一直都放在了我的个人主页上：http://www.olivercamel.com。这个网页也使我认识了许多志同道合的朋友，我们一起交流 OFDM 技术，获益匪浅。西安电子科技大学的郑峰就是我这些朋友中的一位。在毕设完全整理完成后，我还准备继续将全部的内容放到网页上，可以在网上下载到所有代码和说明，这些链接我至少保证到 2007 年初有效。如果有机会，我很希望能和朋友们在网上交流，互相帮助，共同进步。

还应该值得感谢的是我的两位研究生师兄，杨晓涛和黄伟。他们也在做和我类似的题目。他们耐心地给我解答了许多问题，在和他们交流的过程中，我对 OFDM 技术有了更深入的了解。在此向他们表示感谢。

在此总结一下我所做的主要工作：

1、查阅了大量中英文资料，并学习 OFDM 技术的原理；

2、学习 IEEE 802.11a 的标准以及在 Simulink 中的仿真模型；

3、学习 ALTERA 公司的技术，主要是 IP 的使用；

4、编写了几千行的 VHDL 代码来设计一个 OFDM 数据处理系统；

5、在硬件实验板上仿真并通过；

很快就要迎来答辩的日子了，希望能够顺利地通过，为毕设画上一个圆满的句号。

李想

2006 年 6 月

第一章　OFDM 技术介绍

第一章的目的是对 OFDM 技术进行简单的介绍，使读者能够对 OFDM 有一个大概的了解。本章从通信技术的发展说起，阐释了为什么会提出 OFDM 技术，并对它的各个组成部分的原理作了说明。在说明的过程中，作者也加入了一些个人的理解。

1.1 通信技术的发展

进入新的千年，我们进入了一个信息化的时代：可获取信息的途径变得更多更快捷，即使你不想，也每天都在被各种各样的信息包围着。

值得一提的是，近五十年来电子技术和集成电路的发展，对信息化起了极大的促进作用。它使得我们在信息技术、数字处理、有线和无线通信等方面的设想有了实现的可能。新的技术应用于实际的时候，最终总还是要体现在硬件上。即使在光纤通信已经有了很大发展的今天，在进行交换的时候依然需要电子器件。

回到通信的主题，信息化时代的到来离不开通信技术的飞速发展。在通信方面，人们其实一直都有一个理想，即可以在任何时间、任何地点以无限快的速度获得他所需要的信息。我们在向着这个目标不断地前进，也许总有一天理想会变成现实。

具体来看，这个理想包含了三个方面的要求：首先，需要能够先建立一个连接；第二，在这个连接的基础上以尽可能高的速率传输数据；最后，是要实现数据的控制和管理，比如安全和检索等等方面。

从现有的技术水平上来看，第一个方面的要求，即能够建立连接，已经可以说基本上实现了。比如：电视、电话基本覆盖了人们居住的地方；

互联网也已经普及了，我们可以通过网络服务提供商（ISP）连入主干网络，再连接到想去的地方；甚至连无线的电话通信网络，以独特的基站和移动终端的蜂窝网络方式，也覆盖了不少的区域。上述的这些网络，基本上已经满足了我们和任何人取得联系，或者从任何地方获取信息的需求。

再来看第三个方面，数据控制管理的实现。技术人员们已经在这个方面做出了不少的努力，提供了许多容易使用的服务，使人们能够更加方便地获取想要的信息。尽管在这个方面也仍然有很大的发展空间，但是服务的提供首先依赖于数据传输速度的提高。因此，尽管人们也有获得更好服务的需求，但是相比于数据传输速率的提高来说还不是那么的迫切。

所以从目前来看，在第二个方面，即在现有技术基础上尽可能的提高数据传输速率，就变得相对突出起来。由于所有信息都可以采用“0”和“1”的数字方式表示，因此各种通信方式都在向着提供更高的数据传输速率的方向发展。在电视方面，尽管我们已经能够收到清晰的信号了，但是数字电视的发展势头仍然很猛。如果能收到更多的数据量，就可以看到更清晰的画面。而且，数字电视的实现，也提高了交互性，视频点播也许会变得普及。在互联网方面，大城市已经开始推广 ADSL 的接入方式，这比原来拨号上网 56Kbps 的速率快了几十倍，但是仍然不能满足人们的要求，我们还期望着可以光纤到户。在无线通信方面也是这样，从第二代移动通信向第三代甚至第四代移动通信的发展过程，其实就是一个传输速率不断提高的过程。

总的来说，我们在通信的这三个方面的需求是相辅相成的，但是从目前来看，决定通信发展水平的主要因素还是通信数据的传输速率的高低。它也是评价一种通信技术的重要标准。

1.2 OFDM 技术的提出是必然的

1.2.1 无线通信的挑战

相对于有线通信来说，无线通信能够提供更为灵活的移动性能，也更加接近人们在通信方面的理想。所以近十年来发展得异常迅速。

但是无线通信也面临着更大的挑战。由于通信环境的影响，无线通信的信道往往是不能预测的，而且在通信的过程中不得不面对多径衰落，莱斯和瑞利干扰，阴影效应，时间弥散和延迟以及多普勒效应等等。很多关于无线通信的书籍中都有对这方面的描述，比如[1,2]等等。下面重点说一说多径效应的影响，对其它的方面就不再提及了。

1.2.2 多径效应的影响

在无线通信中，由于从发射端到接收端的路线并不是唯一的，这样会导致多径效应。所以，在接收端看来，它收到的信号是一系列的由一个信号沿不同路线反射或折射而来的信号的叠加。如果先不考虑幅度上的衰落，多径造成的影响是，会收到一系列具有随机相位偏差的信号[3]。下面给出了一个示意图。

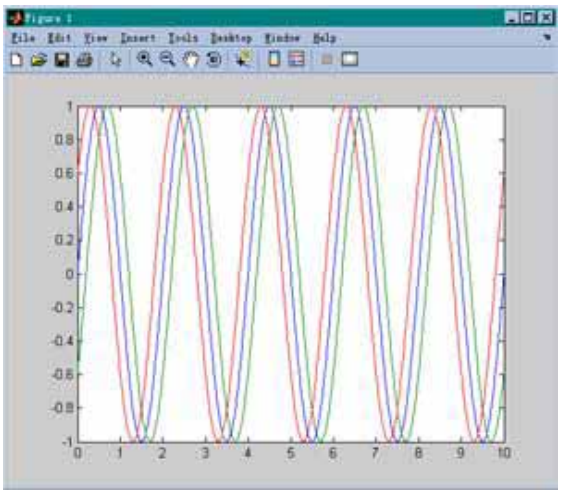


图 1.1　原信号和沿两条路线反射的多径效应示意图

在传输速率很低的时候，多径效应并不影响接收端的判决。但是如果想要提高传输速率，载波频率就会变得很高。这时候，由于多径效应的产生的延迟就会造成符号间的干扰（ISI），因为我们无法分辨一个信号是新信号还是上一个信号的延迟。

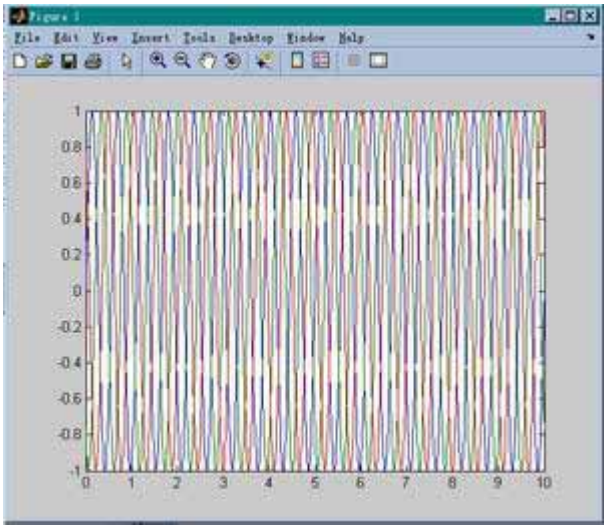


图 1.2　多径效应对高速率传输的影响

由于多径是无法避免的，所以提高传输速率就和多径效应的影响构成了一对矛盾。解决这个矛盾的办法是采用多载波技术。

1.2.3 多载波技术

多载波技术是指采用多个载波，将原来的数据分成多个序列，然后调制不同的载波，最后把多个子载波叠加起来发射出去。在接收端，再恢复出多个载波，然后将每个子载波的数据解调出来。

设想如果采用 100 个载波，那么在总的传输速率不变的情况下，每个子载波的传输速率就变成了原来的 1/100 了。这样就相对降低了速率，可以有效的对抗多径效应的影响。

OFDM 技术就是一种多载波技术。

关于多载波技术的原理方面，我参考了《现代通信系统》这本书[4]。

1.2.4 提高频谱利用率

在无线通信中，频率资源是一种非常稀有的资源，可使用的频带范围非常有限。为了能够充分利用这些资源，各种复用技术被广泛的采用，频率复用（FDM）就是其中的一种。

如图所示，为了能够充分地利用频率资源，可在原有的频率范围内划分多个信道。这与多载波的思想并不冲突，因为我们可以在每个信道上传输一个载波。

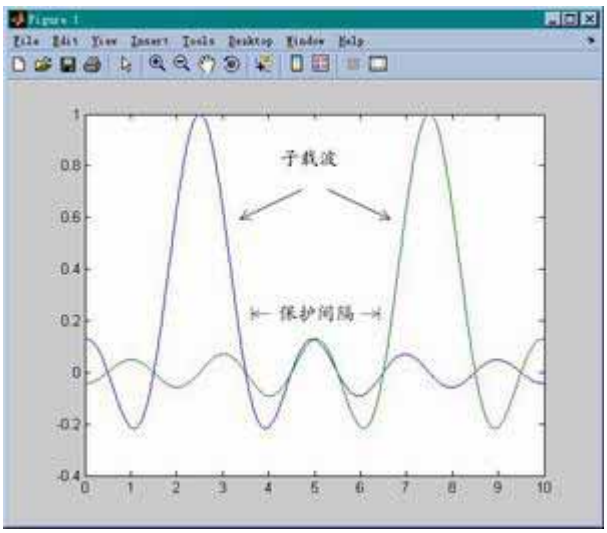


图1.3　划分多个信道

为了尽可能的充分利用频谱，信道当然划分得越多越好，但是依据传统的通信思想，需要在每个信道之间留出一定频率作为保护间隔。

但是如果采用 OFDM 技术，即使用正交的频率作为载波，频谱变成了如下图中所示的样子。在每个信道采样的时刻，其它的信道在这个频率上恰好都为零，这样可以省去用于保护间隔的频带的浪费，更大限度的利用频谱资源。

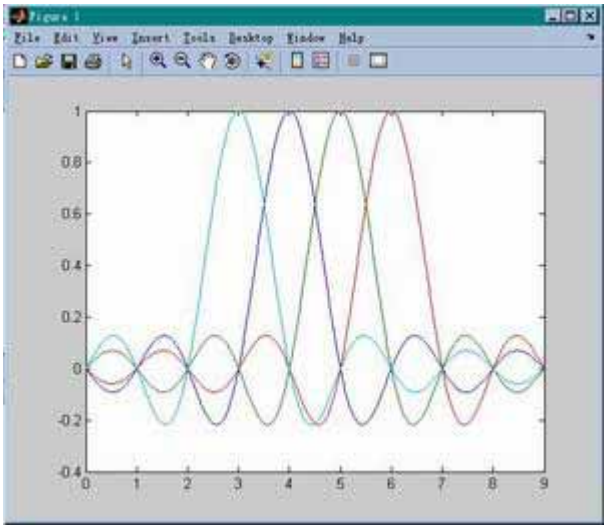


图 1.4　正交频分复用频谱

因此可以说 OFDM 也是 FDM 的一种。只不过比起 FDM，OFDM 更加充分地利用了频率资源。

1.2.5 OFDM 技术的定义

通过前面的介绍，OFDM 的定义已经很明显了。正交频分复用（OFDM, Orthogonal Frequency Division Multiplexing）是一种多载波技术，它将高速率的数据通过串并变换分散到多个子信道上去，使用正交的载波进行调制。OFDM 既可以被认为是一种调制技术，也可以被看作是一种复用技术。

另外，应该看到，OFDM 技术的提出，是有一定的必然性的。从多载波技术的思想被采用，以及必须需要采用频率复用技术尽可能充分地利用频率资源，这些思想直接导致了 OFDM 技术的产生。

至少从这种角度来看，OFDM 在理论上是合理的，理论发展必然产生的结果。一种合理的技术被广泛的应用和推广也是一种可以理解的趋势。也许这就是为什么近十几年来，OFDM 技术变得越来越热门了。

1.3 OFDM 的发展与应用

1.3.1 FFT 促进了 OFDM 的发展

其实 OFDM 思想的本身并不复杂，所以从 OFDM 的提出至今已经大约过了四、五十年。但是由于技术上的限制，主要是无法准确的使用多个载波调制及解调，长期以来 OFDM 一直停留在理论上。

电子技术的发展，特别是快速傅立叶变换（FFT）可以采用高速的 DSP 芯片或者 FPGA 来实现，使得 OFDM 的实现成为了可能。因为从理论上的研究发现，OFDM 的调制过程恰好与离散傅立叶变换（DFT）的公式一致，而 FFT 就是 DFT 的快速算法。

每一本介绍 OFDM 的书中都会乐此不疲地列举一堆公式，详细地说明为什么使用 FFT 可以达到 OFDM 调制的作用，比如[5]，在这里就不再细述了。

1.3.2 OFDM 的应用

从 1990 年至今，OFDM 被大量应用于宽带数据通信。包括：数字广播（DAB，DVB-T）、有线网络（各种 DSL）、高清晰电视（HDTV）、以及无线网络（HIPERLAN）等等。IEEE 也在 802.11 和 802.16 系列标准中使用了 OFDM 作为物理层标准。在下一章的内容中我会对 802.11a 的标准作介绍。

这些都是已经采用 OFDM 技术的范畴，还有更多的领域正在或将要采用 OFDM 技术。根据《Muticarrier techniques for 4G Mobile Communications》一书的预测，在第四代移动通信中很有可能采用 OFDM 技术[1]。如果这能成为现实的话，我估计 OFDM 技术的应用会达到顶峰。

但是不论是否真是这样，OFDM 技术正被应用于越来越多的领域是毋庸置疑的。正如[6]所言，OFDM 这种“古老”的技术焕发的新的活力，有了新的市场。

1.4 OFDM 的结构和各部分原理

1.4.1 OFDM 的结构框图

根据 OFDM 的原理，可以画出大致的结构框图。基本上，各种介绍 OFDM 的书籍中都会有类似的结构图。如下图所示。

图 1.5　OFDM 发射端结构框图

接收端的框图与发射端的类似，只是进行的过程相反而已。

经过编码的数据会依次进行星座映射，FFT 变换，插入循环前缀后再采用无线数字通信的方式发射出去。其中 OFDM 调制的部分包括星座映射，FFT 变换，插入循环前缀这三个步骤。下面依次进行介绍。

1.4.2 星座映射

星座映射是指将输入的串行数据，先做一次调制，再经由 FFT 分布到各个子信道上去。调制的方式可以有许多种，包括 BPSK、QPSK、QAM 等。下图示意了采用 QPSK 调制的星座图。

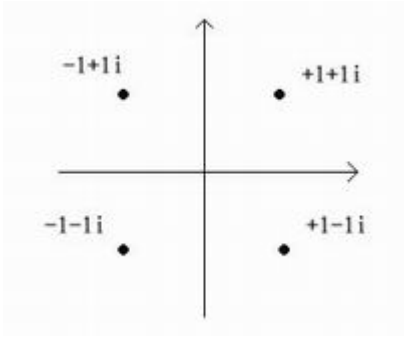


图 1.6　星座映射的过程

OFDM 中的星座映射，其实只是一个数值代换的过程。比如输入为“00”，输出就是“-1+1i”。它为原来单一的串行数据引入了虚部，使其变成了复数。这样一方面可以进行复数的 FFT 变换，另外，进行星座映射后，为原来的数据引入了冗余度。因为从原来的一串数，现在变成了由实部和虚部组成的两串数。引入冗余度的意义在于以牺牲效率的方式降低误码率。通过牺牲效率来换取可靠性在通信上是一种非常经典的思想。

1.4.3 串并变换和 FFT

在星座映射之后，下面进行的是串并变换，将串行数变为并行，主要是为了便于做傅立叶变换。串并变换之后进行傅立叶变换，在发射端是反变换（IFFT），在接收端是下变换（FFT）。最后再通过并串变换变为串行数据。

其实串并变换和并串变换都是为了 FFT 服务的。如果把它们三个看作一个整体的话，那么相当于输入和输出都是串行的数据。假设是 64 点 FFT 的话，那么一次输入 64 个串行数据，再输出 64 个串行数据。

这样做是为什么呢？分析 FFT 的意义，虽然它的输入和输出都是 64 个数，但是对于输入的 64 个数来说，它们互相之间是没有关系的。而输出就不同了，经过了 FFT 变换，输出的 64 个数相互之间有了一定的关联。

在理论上说，就是用输入的数据来调制相互正交的子载波。从直观上来看，64 个数之间产生了互相间的关联，如果有一个数据在传输中发生错误的话，就会影响其它的数据。这就是采用 FFT 所起到的作用，也是 OFDM 技术的精髓所在。

1.4.4 插入循环前缀

OFDM 调制中还有一个必不可少的步骤是插入循环前缀。尽管 OFDM 通过串并变换已经将数据分散到了 n 个子载波，速率已经降低到了 n 分之一，但是为了最大限度地消除符号间的干扰（ISI），还需要在每个 OFDM 符号之间插入保护前缀，这样做可以更好地对抗多径效率产生的时间延迟的影响。

有意思的是，与 FDM 中的使用频率保护间隔类似，对于 OFDM 这样的频率使用率高的系统来说，需要在时域上插入保护间隔。如果对时域和频域相互关系理解较为深刻的话，也许可以找出其中的内在联系。

插入循环前缀本身非常简单，就是把每个 OFDM 符号的最后一部分提到符号前，使整个符号加长即可。如下图所示。

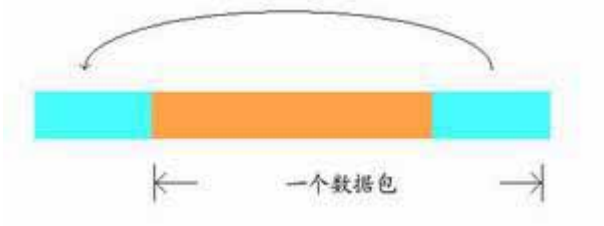


图 1.7　插入循环前缀

1.4.5 对于 OFDM 调制过程的理解

通过上面对于 OFDM 调制过程三个步骤原理的描述，已经作了一个初步的介绍。下面再回到 OFDM 发射端的图，写一写我自己对于 OFDM 调

制过程的理解。

如果把 OFDM 技术发射端的结构图分成两部分：一部分是 OFDM 数字调制部分；另一部分是无线发射部分。前一部分是数字处理的部分，后一部分是发射模拟波形信号的部分。如图所示。

图 1.8　OFDM 发射端组成图

在数字通信中，除了 D/A 变换和无线发射信号以后，在空间中传播的是模拟信号，在发射机的系统中，也就是上图所示的 OFDM 调制部分，始终都是在传输数字的信号。调制的过程，其实就是在做一个数字处理的工作。输入一串数据，经过数值上的代换后变成另一串数据输出。整个调制的过程可以看作一个函数：y=f(x)。x 是输入的串行数据，f 代表调制的过程，y 代表输出的数据。所以如果不考虑那些复杂的理论，那么在 OFDM 的物理层上的所有工作都是按照一定步骤不断地做函数变换，设计 OFDM 物理层硬件的过程也就是实现 OFDM 函数变换的过程。

具体来看，星座映射是将比特流在数值上变换为以星座表示的规范的数值，FFT 是将一串数变成另一串相互间有关联的数，而循环前缀的插入进一步引入了冗余度，使数据扩展得更长。

从这个角度上来说，OFDM 技术也可以看成是一种编码技术。它将一般数值的比特流进行 OFDM 编码后传输。和未经过 OFDM 编码的数据相比，假定以相同的速率传输，以 OFDM 编码的数据在传输的过程中具有频带利用率高，可以对抗多径效应等等的优点，而且误码率也更小。

这就是我在设计 OFDM 基带数据处理部分的过程中所领悟到的。

1.5 小结

第一章简要的介绍了通信的发展及人们目前对于通信的主要需求，即尽可能地提高传输速率。OFDM 作为一种通信技术，因为它能提供更高的传输速率，近几年来被广泛地采用。OFDM 的提出和应用有着一定的必然因素。第一章还介绍了 OFDM 的组成结构，和各个组成部分的原理。

OFDM 也存在一些技术上的问题，比如：非常容易受到频率偏差的影响；具有很高的峰均功率比等等，但是任何一种技术都不会是完美无缺的，不断克服现有技术的缺点的过程，也就是科学技术前进的过程。关于 OFDM 的这些问题也许有天总会被解决。

参考文献

[1] Shinsuke Hara and Ramjee Prased, *MULTICARRIER TECHNIQUES for 4G Mobile COMMUNICATIONS*, MA: Artech House, 2003.

[2] Gordon L. Stüber, *Principles of Mobile Communication*, Second Edition, 裴昌幸, 聂敏, 岳安军译, 电子工业出版社, 2004.

[3] Web ProForum Tutorials, *OFDM for Mobile Data Communications*, The International Engineering Consortium, http://www.iec.org.

[4] John G. Proakis, Masoud Salehi, Gerhard Bauch, *Contemporary Communication Systems Using MATLAB and Simulink*, Second Edition, 刘树棠译, 电子工业出版社, 2005.

[5] 王文博、郑侃编著，《宽带无线通信 OFDM 技术》，人民邮电出版社，2003 年 11 月

[6] Steven J. Vaughan-Nichols, *OFDM: Old Technology for New Markets*, Tutorial of Wi-Fi Planet, http://www.wi-fiplanet.com, November 14, 2002.

第二章　802.11a 标准介绍

在 IEEE 协会为无线局域网（WLAN）制定的标准 802.11 系列的 802.11a 标准中，采用了 OFDM 技术。本章就是对 802.11a 标准的部分内容进行介绍。希望通过介绍，能够展现出 OFDM 技术在实际中的是如何应用的。本章还对 Mathworks 公司的基于 Simulink 的 802.11a 的 Demo 仿真模型作了介绍。

2.1 802.11a 标准介绍

2.1.1 WLAN 和 802.11

无线网络是无线通信中的一个重要的应用，根据网络范围的大小又可以划分为局域网、城域网和广域网。IEEE 为无线网络专门制定了相关的标准，802.11 和 802.16 就属于这方面的标准。其中 802.11 针对范围更小的无线局域网。

无线局域网（WLAN）对在一个小的范围内（比如办公室内）联入 Internet 给予了极大的方便，只要你处于支持 WLAN 的区域，再外加一个无线网卡，就可以轻松地接入网络。特别是对笔记本电脑来说，这种方便更为明显，可以省去再连接网线的困扰，而且移动性能被大大加强了。可以说，正是笔记本电脑上网的问题促进了 WLAN 的发展，并使得 WLAN 变成了一个热门的技术。

802.11 标准包括 802.11a、802.11b、802.11g 等等一系列标准，各自采用不同的物理层技术，其中 802.11a 即采用了 OFDM 技术。

802.11 标准的制定开始于 1997 年，被设计成为一个支持 1M 至

2Mbps 速率的系统。但是这个速率还是不能满足人们的要求。1999 年 802.11a 标准通过，它应用于 5GHz 的频段，并且最高支持 54Mbps 的速率。其它这个速率也还不是很高，但是它毕竟把 WLAN 速率的最高界限提高到了 54Mbps。

2.1.2 IEEE 802.11 标准的获取

IEEE 的所以标准都可以从它的网站上获取，http://www.ieee.org[1]。但是这些标准是需要付费的。幸运的是，IEEE 目前提供了一个 Get IEEE 802®的项目[1]，所以我们可以免费地下载到已经出版了六个月以上的 802 标准的 pdf 文档。应该感谢这个项目的那些主办人（sponsor），因为 Get IEEE 802®无疑为 802 标准的推广和 WLAN 技术的普及做出了巨大的贡献。

通过 Get IEEE 802®，我们可以下载到 802.11a 的物理层标准，这是一个名称为 802.11a-1999.pdf[2]的文件，另外如果想要对 802.11 有一个全面的了解，还应该看一下 802.11 这个最初制定的标准[3]。

2.2 802.11a 部分内容介绍

802.11 和 802.11a 包括这个 WLAN 标准的所有技术细节，加起来足有好几百页。如果把这些内容都描述一遍的话将会是一个非常巨大的工程。所以，在下面的文章中，我只对 802.11a 的部分内容进行介绍。如果想要对其有个全面的了解的话，我建议应该花点时间把标准的原件好好研究一下。

这下面的介绍中，我引用了一些 802.11 标准中的原图，在此特别声明。在所引用图的文字说明部分我都加入了参考文献的标志。

2.2.1 802.11a 结构

802.11a 的物理层结构包括三个部分，一个是 PMD 子层（Physical medium dependent），另一个是 PLCP 子层（Physical layer convergence procedure），还有一个是 PLME 实体（PHY layer Management Entity)。整体结构如下图所示。

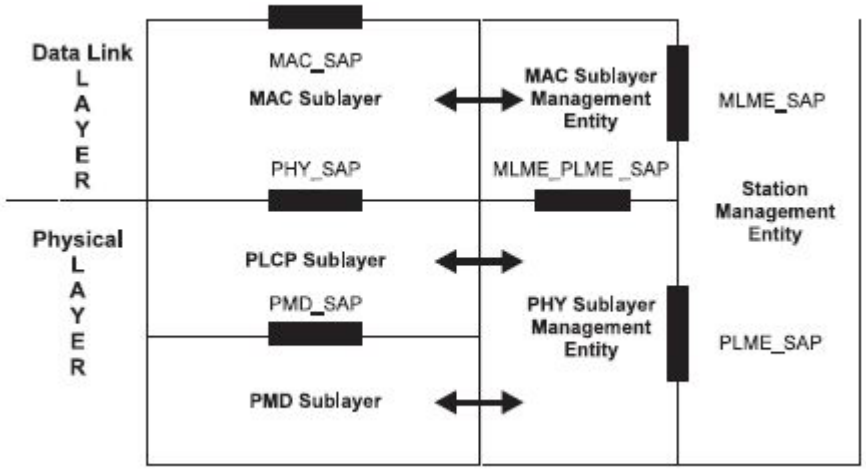


图 2.1　802.11 标准在 OSI 七层模型中的位置[3]

PMD 子层是每个终端和另一个终端通过媒介相连的部分，比如无线设备中的发射电路部分。在 WLAN 中，它负责将 OFDM 调制好的数据发射出去，或者将收到的数据检测出来上交给 PLCP 子层。

PLCP 子层是 OFDM 调制的部分，它将数据进行 OFDM 调制或解调后，生成固定的 PPDU（PLCP protocol Data Unit）帧格式，并上交给 MAC 层进行处理。

PLME 不能算作一个子层，而是一个模块，它对上述的两个子层进行管理。

2.2.2 PLCP 子层的 PPDU 帧格式

在 802.11a 的文档中，对 PLCP 的 PPDU 帧格式进行了详细的说明。在物理层和数据链路层之间的数据就是以这种固定的 PPDU 帧格式进行交换。下图是 PPDU 帧格式的说明图。

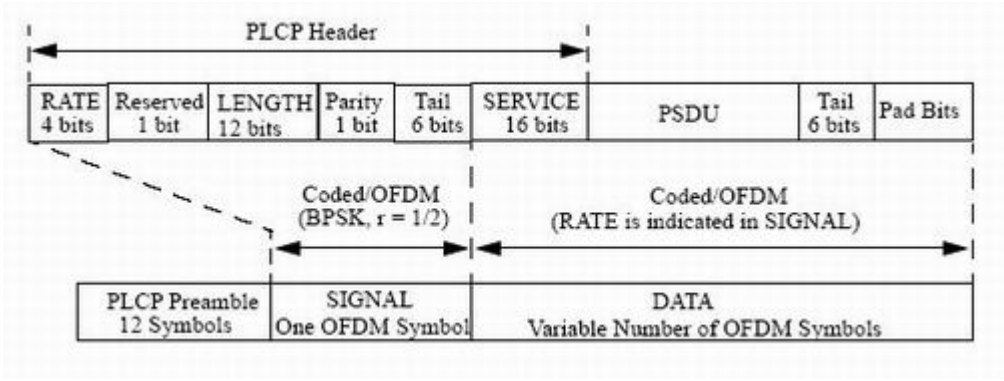


图 2.2　PPDU 帧格式[2]

图中的 PSDU（Physical sublayer Service Data Unit）是 PMD 子层向上提交的最基本的数据单元，通过加入 PLCP Header、Tail 和 Pad Bits，和 12 个符号的 PLCP Preamble 就形成了 PPDU 帧。

关于形成 PPDU 的过程，在 802.11a 标准的文档中给出了非常详细的说明，共有从 a 到 n 的 14 个步骤。请参见标准的 17.3.2.1 小节[2]。

2.2.3 Preamble 的作用

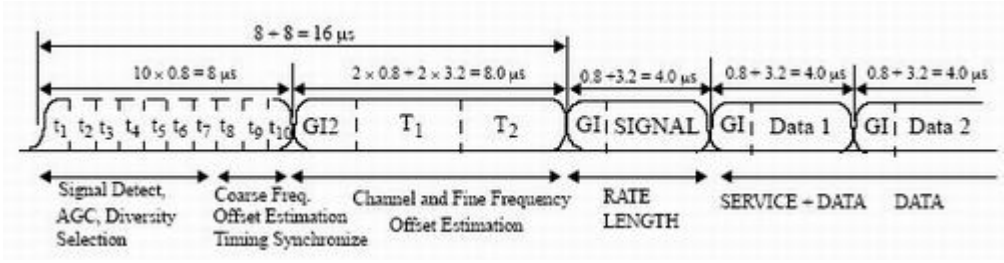


图 2.3　PPDU 的训练序列结构[2]

在 PPDU 中，使用了 12 个符号的 Preamble，这叫做训练序列。分

别是十个短序列和两个长序列。结构如图 2.3 所示。

训练序列在接收端进行帧检测、帧定位、时钟同步、频率偏差估计和信道估计和均衡中起着非常重要的作用。所谓的通信的“同步”主要是靠这十二个训练符号实现的。各部分的具体作用在上图中已经做了说明。

2.2.4 附录 G，一个 OFDM 物理层数据编码的实例

上面重点地介绍了 802.11a 的几个部分的功能。至于具体的数据如何进行调制的过程，还是请参见原标准文档。在此就不一一花费时间描述了。

值得的一提的是，在 802.11a 文档中包含了几个附录(annex)，其中的 Annex G 是一个 OFDM 调制过程的实例。这个实例假定一个确定的输入，给出了每一步变换后的结果，值得一看。看过之后，对于 OFDM 整个调制过程的理解应该会更具体，更直观。

另外，Annex G 其实也直接说明了：OFDM 调制的过程就是对输入数据作一个数值变换的过程。这就是我在 1.4.5 中阐述的对于 OFDM 调制的一种理解。

2.3 基于 Simulink 的 802.11a 的 Demo 仿真模型

2.3.1 Demo 模型和获取

当研究某种技术的时候，我们总希望在制作硬件之前可以进行一下仿真，既是对硬件实际性能的测试，也能够预先发现一些问题并且进行改进。

提到仿真的时候，我们总会想起 MATLAB 和 Simulink，这个功能强大的软件工具。能不能用 Simulink 对 OFDM 进行仿真呢？当然可以，而且非常简单。因为在 MATLAB7.0 版本之后，在 Commuication Blockset

中包含了许多基于 OFDM 技术的 Demo，包括 DVB、ADSL 和 HIPERLAN/2，特别是包含了一个 802.11a WLAN 的物理层仿真模型。如下图所示。

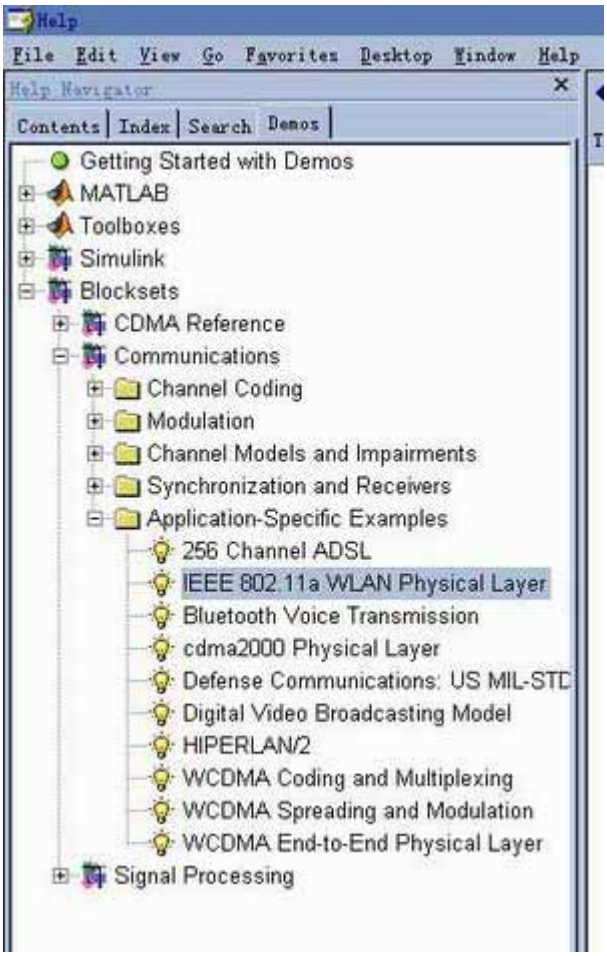


图 2.4　选择仿真模型

如果使用 MATLAB 更早的版本的话，在 Mathworks 网站的社区论坛中也可以找到这个仿真的相关文件[4]。直接搜索“802.11a”即可。

2.3.2 模型介绍和实时仿真

除了 MAC/PHY 的接口、PLCP header、Data Scrambler、短训练序列和 OFDM 符号定时窗口[5]，基本上，这个基于 Simulink 的 802.11a

物理层模型包含了所有 OFDM 的模块和功能。下图所示的就是这个模型。

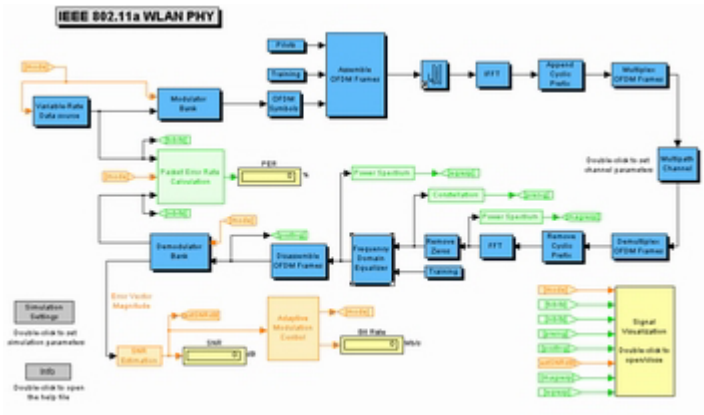


图 2.5　802.11a 物理层 Simulink Demo

双击一个模块可以设置参数，或者看到模块的内部结构。虽然模型的作者给出的结论只是功能上的仿真，并没有就某个模块进行详细介绍，但是如果稍有一点 Simulink 的知识的话，就可以分析出 OFDM 的工作原理。比如下图所示的是“ 频域均衡 ”模块的内容结构，可以分析出具体实现均衡的过程。

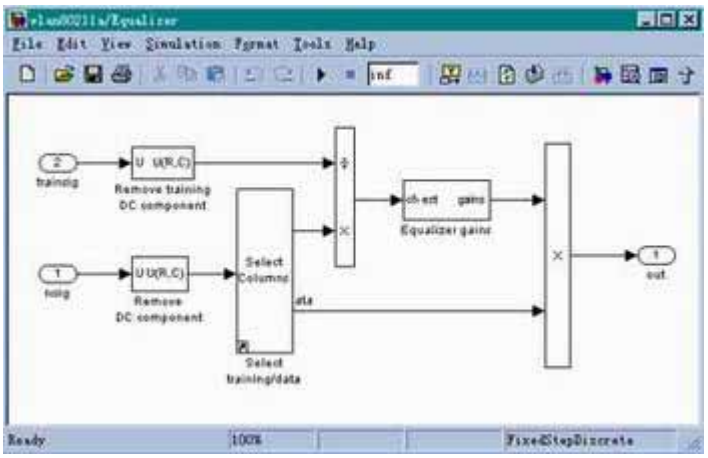


图 2.6　频域均衡模块内部结构图

启动 Simulink 进行实时仿真，在观察窗口中会给出实时的数据结果，包括发送的数据，接收的星座和误码率等等，如下图所示。

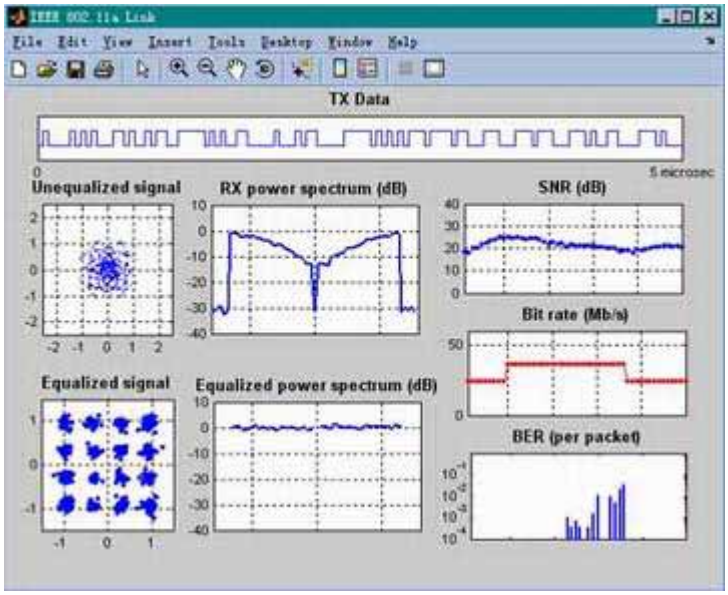


图2.7　实时仿真图

2.3.3 仿真模型的亮点

这个仿真模型有个最大的特点，就是可以在发送端自动的改变前身纠错编码方式、编码速率、以及星座映射的方式等，以期达到最低的误码率。模型假定信道的特性是实时变化的，即信噪比会有很大的起伏。这样，如果采用较高的速率发送，比如 54Mbps 和 64QAM 的映射方式，而信噪比又很低的话，那么在接收端的误码率就会非常的高，这个时候，接收端会把这一信息反馈回发送端，而发送端会相应进行调整，降低发射速率，并改变映射方式，比如采用 QPSK。这种自动调节的方式一方面降低了误码率，另一方面也最大限度地提高了传输效率。对实际中基于 802.11a 的系统也具有非常好的指导意义。

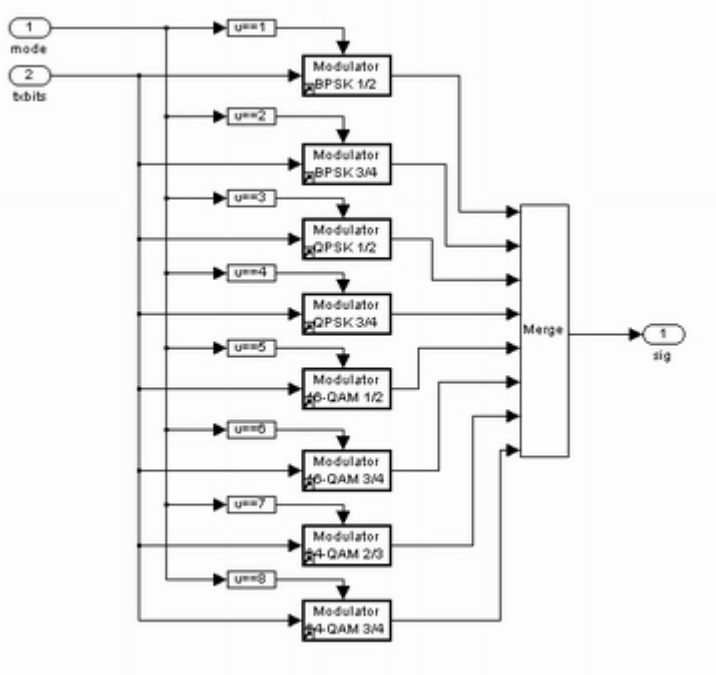


图 2.8　选择编码和星座映射方式

2.4 小结

OFDM 的一个重要应用就是基于 IEEE 802.11a 标准 WLAN。本章从两个方面对其进行了介绍，一是介绍标准本身，另外是介绍了在 Simulink 下面的仿真模型。如果认真阅读了标准并研究过仿真模型的话，我想就应该会对 OFDM 技术本身及其应用有一个非常充分的掌握了。

但是遗憾的是，我在开始毕业设计题目的时候，只专注于 OFDM 技术，并没有想到要去参考相关的标准。等到毕业设计进展到一半的时候，我才在查找资料的过程中想到应该去了解 OFDM 在实际应用中是如何实现的，比如 802.11 对 OFDM 的技术参数是如何要求的。如果能够在一开始就以研究标准为起点的话，那么我在设计硬件的时候就可以依照标准的规定来进行设计了。因此，我所做的 OFDM 的硬件方面的设计只能说是一个示意

性的系统。如果以后有时间的话，我想也许我会在此基础上做一些进一步的改动，使我做的设计能够符合 802.11a 或者更新、更高性能的标准。还有，由于时间有限，我对 802.11a 标准和 Simulink 模型的研究也都是很浅显的，只是在论文中进行了一下轮廓上的描述，难免会有出错的地方，如果可能，我希望读者能够通过邮件向我指出文中的错误。如果希望对 802.11a 的标准进行深入研究的话，我还是建议去阅读标准原件和参考 Simulink 模型。

至此，在 OFDM 技术理论上的介绍全部结束了。在下一章，内容将会转到关于硬件设计的方面。

参考文献

[1] http://www.ieee.org

[2] IEEE Std. 802.11a 1999 Edition (R2003), “Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-speed Physical Layer Extension in the 5-GHz Band”, IEEE, 2003.

[3] IEEE Std. 802.11 1999 Edition (R2003), “Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, IEEE, 2003.

[4] http://www.mathworks.com

[5] Demonstration Models, *IEEE 802.11a WLAN Physical Layer*, Communication Blockset, The Mathsworks, Inc.

第三章　FPGA 和 ALTERA

在介绍完 OFDM 的理论之后，本章开始介绍硬件方面的相关内容。要采用 FPGA 芯片来实现 OFDM 技术，首先需要对 FPGA 技术有所了解。本章介绍了 FPGA 技术的优势所在，这也正是我采用 FPGA 来实现 OFDM 的原因。

ALTERA 公司和 Xilinx 公司是 FPGA 行业的两大巨头，因为我在毕业设计中采用的是 ALTERA 公司的器件和技术，自然而然的，在本章也同时对 ALTERA 公司的技术和理念进行了介绍。

3.1 FPGA 技术的优势

3.1.1 可编程技术

在上个世纪，设计大规模数字电路的需求促使了可编程技术和可编程器件（PLD）的出现，而 PLD 又对数字电路的发展起到了巨大的促进作用。现在，一提到数字电路，人们往往把它和 PLD 器件联系起来。

可编程技术是指，利用计算机语言对硬件结构和功能进行设计，再把编译后的文件下载到指定的 PLD 芯片中，即可实现这些功能。VHDL 和 Verilog 就是两种现在广泛采用的硬件描述语言（HDL）。

所以，自从可编程技术的诞生之日起，与其它技术相比，采用 HDL 进行 PLD 的设计就显得非常灵活和方便。我们可以用不同的程序实现不同的功能，也可以随时对程序进行改动并立即在硬件上看到结果。

CPLD 与 FPGA 是现在常常提到的两种 PLD 芯片，它们的内部结构有所差别，但是在 PLD 的本质上是一致。相对于 CPLD，通常 FPGA 的内部

资源更加丰富，性能更强，可以实现更多的电路功能。我设计的 OFDM 数据处理部分就是在 ALTERA 的 FPGA 上实现的。所以在下文中有时也直接用 FPGA 替代 PLD 这个词的涵义。

3.1.2 FPGA 的技术特点

FPGA 之所以能够通过语言描述来实现不同的硬件功能，是由其内部的逻辑单元（LE）实现的。一个 LE 能够实现一些基本的功能，而许许多多个 LE 组合在一起就可以实现更大规模的电路和更强大的功能。不同厂家生产的 FPGA 的 LE 结构不尽相同，具体的内容可以去参考厂家提供的技术手册。

工艺上的发展导致可以生产出的尺寸越来越小的电路。现在的前沿制造技术已经能够计划几十个纳米（nm）级别的电路了。这意味着在芯片大小不变的情况下，内部的 LE 的数量将会越来越多。ALTERA 公司最新推出的 Stratix GX 系列器件中的 LE 数目已经达到了最多 4 万多个[1]。而且随着技术的发展，这个数目还会变得更多。

这种由 LE 实现电路功能的方式是 FPGA 最大的技术特点。当使用 HDL 设计完成，并对每一个 LE 进行配置之后，FPGA 系统是一个完全的硬件电路，而不含有指令控制的部分。它使得 FPGA 系统的时钟速率非常的快。以 Stratix GX 系列器件为例，内部时钟的速度超过了 300MHz，而且更可支持高达 6Gbps 的收发速率[1]。

3.1.3 FPGA 相比于 DSP 芯片的优势

在上一小节中中所述的 FPGA 的技术特点，使得 FPGA 的速率非常快，这是它相对于 DSP 芯片的主要优势所在。

因为 DSP 是基于指令系统的，所以需要依次地执行每一条指令，这样

最后才能得出结果，而 FPGA 是并行的系统，相当于一次性执行所有的指令，这样看起来，整体的速度性能会比 DSP 芯片强很多。

以 OFDM 系统中的 FFT 部分为例，下面给出了一个 8 点 FFT 的流程图。

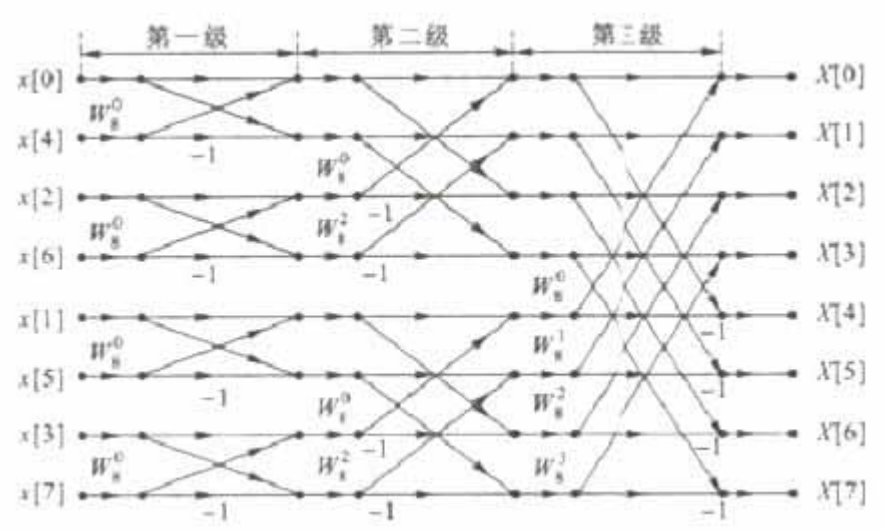


图 3.1　基 2 时间抽取 8 点 FFT 运算流程图[2]

如果要采用 DSP 芯片来完成 FFT 功能，在第一级的四个蝶形模块需要从上到下依次地进行运算处理，而如果用 FPGA 的话，第一级的四个蝶形模块是同时处理的。这样就要比 DSP 的速度快了四倍。对于更多点 FFT 运算，比如 1024 点来说的话，速度快的优势就更为可观了。

上述的思想来源于《现代 DSP 技术》[3]这本书。它使我更深刻地领会了 FPGA 并行计算的特点和 FPGA 在数字信号处理领域的优势。

但是采用指令系统依然有它的好处所在，就是软件方面的智能性会强一些。但是，ALTERA 嵌入式处理器 NIOS 的推出使得在 FPGA 内部也可以实现一个处理器，并采用指令系统来操作。所以 DSP 的优势就荡然无存了，我相信总有一天在数字信号处理领域，FPGA 会完全替代 DSP 芯片。

NIOS 是 32 位的软核处理器，可以应用于 ALTERA 的 FPGA 中。它

具有非常高的性能，而且成为 SOPC（即整个数字系统全部在 PLD 上实现）的重要的组成部分。最近 ALTERA 又推出了 C 语言到 HDL 的编译器，可以把特定 C 函数变成硬件，进一步提高了程序运行速度。

关于这些内容的详细信息请参考 ALTERA 公司网站[4]。

3.1.4 FPGA 相比于 ASIC 技术的优势

ASIC 是指专门定制的集成电路。由于是为了某种功能而专门制作的芯片，所以性能比较高，另外由于一次性大批量生产，所以单片的成本很低。但是 ASIC 的缺点也很突出，首先是设计周期长，所以投入市场的时间也会很长，而且随之带来的是设计风险很大。因为如果一次性设计不成功，那么投入就都泡汤了。

相比于 ASIC，FPGA 的优势在于风险小，设计灵活，完成一个设计的周期也要短得多。

自从 ALTERA 推出了 HardCopy 技术，使 FPGA 的这种优势更为明显。HardCopy 可看作是 FPGA 和 ASIC 间的一座桥梁。它把已经设计好的 FPGA 电路进行优化，之后作为 ASIC 的原型来设计生产 ASIC。这样，ASIC 的设计人员就可以先在 FPGA 上进行设计，再转换为 ASIC。这不仅加快了设计周期，而且大大降低了风险。不仅消除了原来 ASIC 的缺点，而且还融入了 FPGA 的优点。HardCopy 的提出，使得 FPGA 变得更具竞争力。

3.1.5 对 FPGA 发展的预测

FPGA 的优势使它具有非常广阔的发展空间，而且还有着越来越好的市场前景。如果能找到一种将 FPGA 单片成本再降低一些办法，如前文所述，我认为 FPGA 将很有可能取代 DSP 芯片在数字信号处理领域的应用；

如果 NIOS 软核 CPU 的性能能够再有所提升，甚至对于基于 ARM 结构的处理器也将是一个挑战。

从 FPGA 器件的发展趋势来看，FPGA 的性能还会变得更强，主要体现在三个方面：第一，由于工艺的发展，FPGA 中会集成更多的 LE；第二，在频率方面，FPGA 还有上升的空间，可以支持更高的时钟速率；第三，在集成性方面，FPGA 已经集成包括 PLL、RAM 等等的资源，可以期待它还能做得更出色。

总之，FPGA 是一种非常有活力的技术，也有很大的潜在价值。

3.2 ALTERA 公司的理念

本小节将要针对 ALTERA 公司的一些经营理念作介绍，也许在内容上可能离 OFDM 的实现略有偏差，但是还是有一定的借鉴意义。

3.2.1 免费的工程师培训

在进行 FPGA 设计的时候，我总是不断地为自己只能局限于 VHDL 语言和 ALTERA 的器件而抱怨。因为在学校上课的时候，老师讲授的就是这些内容，而且由于要考试，所以必须要掌握好。虽然我也知道 Verilog 和 Xilinx 公司的存在，但是由于个人能力有限，实在无法去对它们进行更多的学习和研究了。就像每个人的母语一样，一旦掌握了，以后在应用中往往就会产生依赖。

说了这么多，并不是在批评学校的灵活性，而是应该看到教育对掌握一门技术的重要性。当你学会了一门技术，以后就不断地需要去应用它。对于 ALTERA 来说，当你掌握了如何使用它的器件和技术，你也就成为了 ALTERA 公司的一个长期客户了。

所以，现在 ALTERA 通过它的经销商在中国开设免费的工程师培训课

程，在网站上可以找到专门的信息。猛然间看上去觉得很划算，因为类似的技术培训基本上都需要高额的费用，其实最终的获益者还是 ALTERA。

除去开设培训班，ALTERA 还通过其它一些手段尽可能地通过教育的手段推广自己的技术。包括在线的讲解，开展大学计划，为学校建设实验室，赠送实验套件，举办竞赛等等。这些思想值得借鉴。

3.2.2 免费的设计软件

ALTERA 公司一直以来都很清楚地把自己定位在出售硬件芯片的公司，最终的盈利都要通过硬件的卖出来实现，所以在其它方面就尽量提供免费的服务。ALTERA 的软件就是一个很好的例子。

ALTERA 的 FPGA 设计软件以其方便易用而出名，不论是 Maxplus 还是 Quartus。而且这些软件都提供可免费获得和使用的网络版。只需要在网站上进行简单的注册，就可以获得一个 license 认证文件，进而完成从 HDL 编辑到为 FPGA 芯片编程的全部过程。虽然免费版软件的功能不像付费版那样的强大，但是已经够用了。

有一个真实的故事：以前，某芯片公司提供设计软件的时候需要花钱购买，但与此同时，ALTERA 却在提供免费的设计软件。所以大批的用户都转而使用 ALTERA 的软件，当然也就购买 ALTERA 的器件了。这个故事体现出成功的经营理念是多么重要。当然，这已经成为了历史，现在所有 FPGA 厂商都在提供免费的网络版设计软件了。

3.2.3 OpenCore plus 技术

在硬件设计的领域，采用 IP（Intellectual Property）进行设计的思想已经深入人心了，这在第五章的内容中还会详细介绍。ALTERA 不仅提供丰富的 IP，而且提供 OpenCore plus 技术，极大的方便了 IP 的测试

与应用。

OpenCore plus 技术的核心精神，就是使用户在购买之前能够完全地感受到这个 IP 的使用。如果选择了 ALTERA 的 IP，从网站上下载、安装、进行设计、RTL 仿真都是免费的。特别地，如果这个 IP 支持 OpenCore plus 的话，你可以把 IP 下载的 FPGA 中观看实际的运行效果。但是 OpenCore plus 生成的编程文件在 JTAG 线拔掉后的一小时就会失效。使得 IP 不能继续工作。我非常好奇 ALTERA 是通过什么手段来实现产生精确时效性的编程文件的。

不论怎样，这项技术为是否决定购买一个 IP 提供了极大的帮助。这个技术背后的思想，与上述 ALTERA 的经营理念有异曲同工之妙。

3.2.4 技术支持与服务

ALTERA 还提供非常出色的技术支持服务。通过 ALTERA 网站上的 MySupport 服务，你可以直接给 ALTERA 的工程师发出信息，叙述自己遇到的困难和问题。

ALTERA 的工程师会在极短的时间内做出响应。并且可以通过发贴的方式进行深入的交流。

我使用过几次 MySupport，每次都在非常短的时间内解决了我的问题。

3.2.5 总结

ALTERA 的经营理念既为用户提供了极大的方便，同时也为自己获取了不小的收益。其中的一些思想使得我们参考和借鉴。

3.3 ALTERA 公司的技术

这个小节再简要而全面地介绍一下 ALTERA 的技术。

在 FPGA 器件方面，ALTERA 目前推出的是两个系列，高性能的 Stratix 系统和低成本的 Cyclone，这是主流的 FPGA 产品。

在 CPLD 器件方面，已经全面升级到了 MAX II 系列。这种 CPLD 其实在内部结构上已经和 FPGA 没有差别了，只不过是多了一个 Flash，用来存储编程信息，来实现 FPGA 掉电不丢失的功能[5]。如果 MAX II 能够获得成功，有理由相信在不久的将来，所有的 FPGA 都可以实现掉电不丢失的功能，从而取消上电配置的环节。

在设计软件上，Quartus II 已经推出了 6.0 版本，进一步提高了性能。Quartus 虽然易于操作，但是对于代码仿真还是稍显麻烦，特别是工程较大的时候，每进行一次改动都需要先编译，这要花费很久的时间。现在 Quartus 软件还加入了 ModelSim 的 OEM 版，估计是想在这方面借助于 ModelSim 的强大功能。

但是 ALTERA 丝毫没有放弃 Quartus 在验证方面的努力。SignalTap 嵌入式逻辑分析仪可以让设计人员查看到内部的节点能力。还有随 Quartus II 6.0 版一起最新推出的 TimeQuest 时序分析仪，它能够对高性能的 FPGA 进行彻底的时序分析。

DSP Builder 是 ALTERA 专门为 DSP 应用而设计的软件。它是一个基于 Simulink 的 Blockset，也就是说它可以在 MATLAB 下应用。通过 Simulink 建模和仿真，最后再通过 DSP Builder 转化为 HDL 语言，加快了设计的流程。

此外，ALTERA 还提供各种类型的 IP 核，以方便 FPGA 的设计。

再加上前文介绍过的 HardCopy 技术，NIOS，SOPC 等，这些基本上涵盖 ALTERA 当前的所有技术范围。

3.4 小节

本章介绍了 FPGA 技术及其优点，这也正是我使用 FPGA 来实现 OFDM 技术的原因。因为我使用 ALTERA 公司的器件和技术，所以也对这些范围进行了全面而概括的介绍。在下一章，我将介绍 OFDM 硬件的核心，ALTERA 的 FFT MegaCore 的使用。

参考文献

[1] Data Sheet, “Stratix GX FPGA Family”, version 2.2, ALTERA corp., 2002.

[2] 陈后金主编，数字信号处理，高等教育出版社，2004 年

[3] 潘松、黄继业、王国栋编著，现代 DSP 技术，西安电子科技大学出版社，2003 年

[4] http://www.altera.com

[5] Handbook, “MAX II Device Handbook”, preliminary, ALTERA corp.

第四章　ALTERA FFT MegaCore 使用指南

在数字信号处理的领域，经常会用到离散傅立叶变换（DFT）。快速傅立叶变换（FFT）是 DFT 的快速算法，效率更高，具有广泛的应用。

在 FPGA 的设计中，基于 IP（Intellectual Property）的设计方法已经被大多数人所接受。在一个大的工程中，如果需要使用 FFT 功能，往往会使用一个 FFT 的 IP 核。ALTERA 公司的 FFT MegaCore 就是这样的一个 IP 核，本章着重介绍了它的使用方法。其中大部分内容来自 ALTERA 的 FFT MegaCore User Guide[1]，同时也加入了作者在使用过程中的一些经验和体会。看过本章之后，应该就能掌握如何使用 FFT MegaCore 了。

4.1 FFT MegaCore 介绍

ALTERA 公司推出的 IP 核，都会冠以“MegaCore”的商标，其实 FFT MegaCore 就是一个可以实现 FFT/IFFT 功能的 IP 核。

FFT MegaCore 的使用非常方便，可以大大加速产品的设计周期。它可以应用在 ALTERA 的主流 FPGA 系列芯片上，包括 Cyclone、Stratix 和 HardCopy。而且因为是 ALTERA 自己设计的 IP，所以专门为这些 FPGA 芯片做了优化，相对于个人编写的 FFT 电路来说，ALTERA 的 IP 具有更高的性能，占用更少的资源。毕竟最了解这些 FPGA 器件的还是 ALTERA 公司自己。FFT MegaCore 支持超过 300M 的时钟频率。

总的来说，FFT MegaCore 的优点在于使用方便，参数可配置，高性能，可加快设计周期。但是的它的价格还是比较贵的，一位在中国的代理商给我的报价是大约 8000 美元，看来还是只有大企业和实验室能用得起。

4.2 FFT MegaCore 应用流程

整个 FFT MegaCore 的应用流程非常简单。可以在 Quartus 软件中非常方便地进行配置并把 FFT 模块加入到自己的工程设计中。本小节对使用的流程进行了介绍。

4.2.1 下载和安装

FFT MegaCore 可以在 ALTERA 的网站上免费进行下载，属于 DSP 类 IP 核。最新的版本是 v2.2.0。安装的过程也很简单，和一般的软件没什么不同。

4.2.2 在工程中插入 FFT MegaCore

使用 FFT MegaCore 需要新建一个工程或者在一个已有的工程中。在本文中我们新建一个工程作为范例。

要在工程中应用 FFT MegaCore，需要先启动 IP Toolbench，一个 FFT MegaCore 的参数配置程序。先点击“ Tool ”菜单下的“ MegaWizard Plug-In Manager... ”。如下图所示。

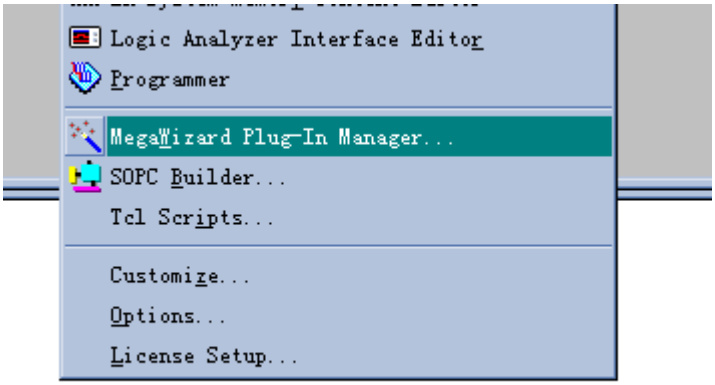


图 4.1　打开 MegaWizard Plug-In Manager

之后选择建立一个新的 Megafunction，就打开了 IP 核的选择页面。

在左侧选择 FFT 核，在右侧可以选择所应用的 FPGA 系列器件；输出文件的语言；为 IP 起一个名称。如下图所示。这本文的例子中，我们的设置为 Cyclone，VHDL 语言，起名 FFTcore。

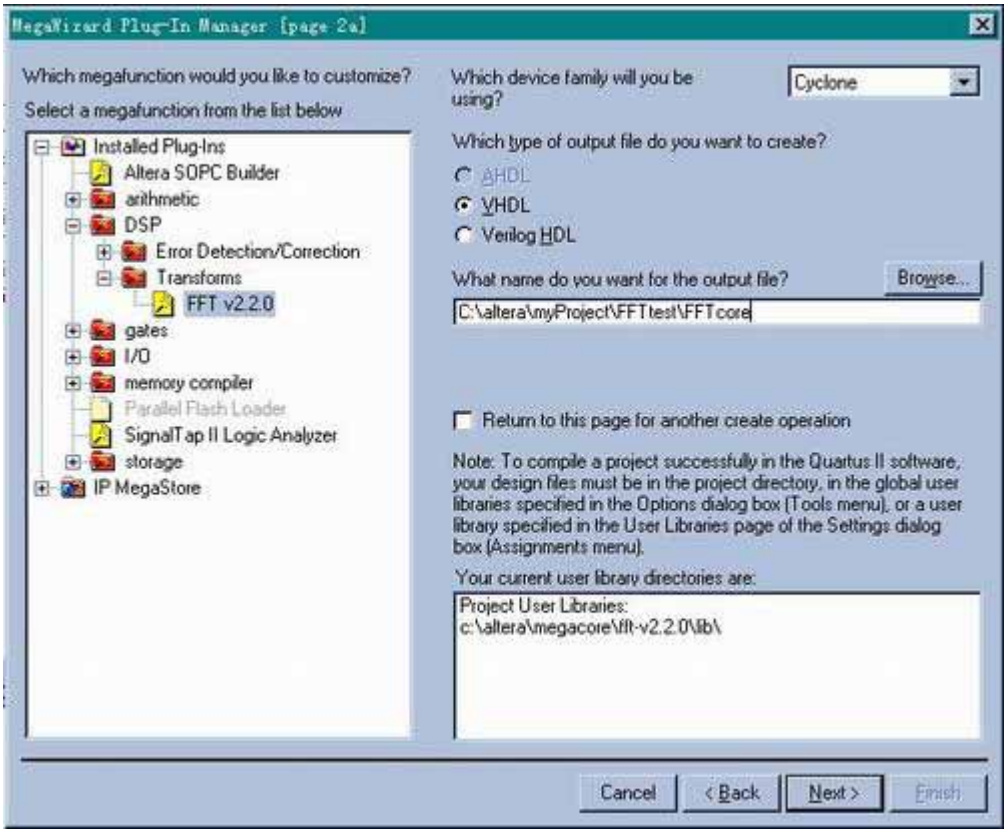


图 4.2　选择 FFT MegaCore。

点击确定，IP Toolbench 就启动了。

4.2.3 IP Toolbench 的使用

IP Toolbench 的使用简单明了，只有三个步骤。先是配置参数，之后是设置仿真，最后是生成。另外，还可以看到一些相关介绍信息。如图所示。

图4.3　IP Toolbench

4.2.4 配置参数

在配置参数的部分，我们只把“Transform Length”设定为64，意味着做64点FFT变换。其它的都不做改变。

每个参数的作用在下文中会作更具体说明。

4.2.5 生成FFT MegaCore

因为设置仿真的内容在本文中没有使用到，所以我们在配置参数之后不选择“Set up simulation”对仿真的部分再做设置，而是直接点击“Generation”生成IP。

IP Toolbench 会自动生成所有的文件并加入到工程中，并在生成信息中显示所有的文件名和 IP 管脚名列表。

然后，我们就可以在工程中使用这个 IP 了。

4.2.6 在工程中应用 FFT MegaCore

在工程中应用 FFT MegaCore 有两种方法。

一种是新建或在已经的原理图文件(bdf)中插入 FFT 的符号。如果已经生成过 IP 的话，可以在“ Project ”下面找到。

另一种方法是针对用 HDL 语言设计。如果使用 VHDL 语言，在生成的文件中会有一个“ .cmp ”文件，其中的就是对 FFT 的 component 的声明，直接把它复制到调用 FFT 核的上层文件中即可。

在本例中我们采用原理图的设计方法。在整个工程中只采用一个 FFT 符号，并将其输入，输出脚连到 input 和 output。如下图。

图 4.4　采用 bdf 设计 FFT 核

4.2.7 编译和仿真

之后，就可以对整个工程进行编译了。这个过程可能会稍显得漫长：在我的 P4 1.4G 256M 内存的机器上需要 15 分钟左右才能完成一次全编译。

按照许多人的习惯，之后要进行的是编程一个波形文件进行仿真。如

下图所示。

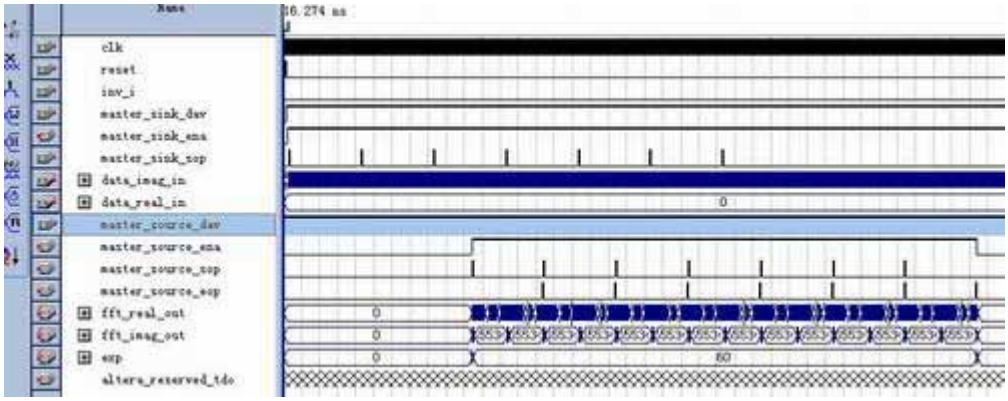


图 4.5　采用波形文件进行功能验证

其实也可以用其它的方法对 FFT MegaCore 的功能进行验证，比如在 MATLAB 中，在后面我会再做介绍。

4.2.8 OpenCore plus 特性

最后的一个步骤就是下载的芯片中了。通常，在购买 IP 之前都只能进行软件仿真，而不能下载的，也就是说，我们还是不能确定自己设计的工程在硬件中能否正常的工作。但是现在，即使不购买 IP，也可以下载了。

这要感谢 ALTERA 的 OpenCore plus 特性。这个特性可以生成具有时效性的编程文件。具体来说，是指在 JTAG 线拔掉后的一小时后，编程文件才会失效了。这种机制对验证一个 IP 能否正常工作提供了巨大的帮助，现在，即使不购买 IP 也可以看到它在硬件中的运行情况了。

对于实验室的研究人员来说这真是一个福音。

4.2.9 购买 license 认证

如果一切验证都通过了的话，就可以向 ALTERA 购买 IP 了。购买 IP 之后，ALTERA 会根据不同的机器提供 license 文件，也就是说只能应用

于一台计算机。在这台机器上可以无限次地下载 FFT MegaCore 了。

4.3 各项具体参数说明

本小节会对 FFT MegaCore 参数配置中的各项参数的含义进行说明，介绍不同参数的用途和区别。

4.3.1 介绍

在参数设计中共包括：Parameters、Architecture 和 Implementation Options 三个栏，分别对不同的方面的参数进行设置。

需要注意一点，在参数设计窗口的下半部分，是使用了多少资源还有性能的统计数据。这些数据会根据你修改的参数实时计算并反映出来。比如，在相同情况下，选择 1024 点的 FFT 就会比 64 点的 FFT 多使用 1000 多个 LE，占用的 RAM 资源也会更多，FFT 变换的计算周期也会更长。见下图。

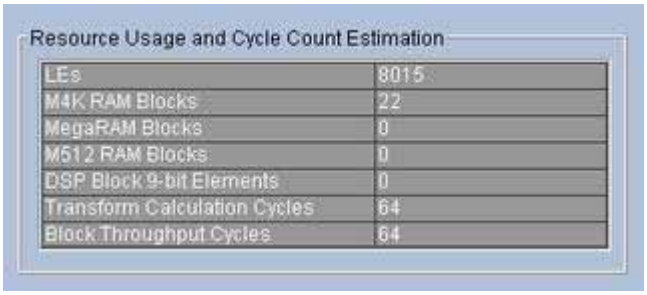


图 4.6　FFT 资源使用和性能统计数据

所谓的 FFT 参数配置，实质上无非就是我们根据自己项目需求的情况，在占用的 FPGA 资源和 FFT 运算效率之间做一个折衷。在资源允许的情况下设置参数以期达到最好的性能。所以反映资源使用情况和 FFT 变换性能数据的部分非常有帮助，每次更改一个参数后都可以看一下。

从下一小节开始，将依照三个栏的顺序介绍参数，有些显而易见的参

数就不做介绍了。

4.3.2 FFT 点数（Transform Length）

这个参数代表了 FFT 的点数，只能在固定的几个数中间进行选择，最低 64 点，最高可达到 16384。

需要提出来的是，ALTERA 的 FFT MegaCore 会根据 FFT 的点数自动地设置 FFT 过程中的基数。对于 2 的偶数次幂的点数，比如 64 点，会在 FFT 的每一级都设置为基 4；但是对于 2 的奇数次幂的点数，比如 128 点，就无法满足每一级都为基 4，所以前面的几级是基 4 的，而最后一级设置为基 2。

总的来说，ALTERA 会尽可能地使用基 4 的 FFT，因为这样效率更高。

4.3.3 数据位数和旋转因子

在 FFT 点数的下面是设置数据位数和旋转因子，通常将这两个数都设置为相同的长度。比如将数据位数设置为 16bits，意味着输入和输出都是 16bits 的数。

ALTERA 的 FFT MegaCore 采用的基本上相当于定点数运算。因为对于 FPGA 来说，如果在 FFT 中完全使用浮点数运算的话，将会是相当占用资源的。但是如果不采用浮点数的话，就意味着 FFT 的过程中会出现误差。因为在一次 FFT 的运算过程中，会有多次的乘加运算，这导致了某一个数会变得很大，超过原来的数据宽度可表示的范围。在 FFT MegaCore 中，为了正确表示这个最大的数，其它相对小的数就会被忽略了，所以误差也就产生了。也就是说，一组数，先经过 FFT 变换，再经过 IFFT 反变换，得出的结果和原先的数不同。

有的时候，我们并不要求完全没有误差，而是要求误差在一个允许的

范围以内。这时，数据位数和旋转因子的设置还具有一个非常重要的作用：当把宽度设置得更大的时候，误差就小。比如 16bits 的 FFT+IFFT 运算后产生的误差就要比 8bits 的 FFT+IFFT 的误差小得多。注意在设置参数的时候，ALTERA 使用的是“ Data Precision 和 Twiddle Precision ”，用了“ precision ”这个词，而不是“ width ”。

当然，数据位数和精度的加宽，意味着会消耗更多的硬件资源。

4.3.4 I/O Data Flow 设置

在 Architecture 栏内，主要可进行 I/O Data Flow 和 Engine Option 两部分的设置。主要是决定 FFT MegaCore 的内部结构。

I/O Data Flow 的设置共有三种选择：Streaming、Buffer Burst 和 Burst。

在不同的 FFT 系统中，有时候使用连续的数据流，数据不断地被送到 FFT 核，经过变换之后再连续的输出。但是 FFT 的运算是需要一定时间的，而在这个时间内新来的数据还是被源源不断地送到 FFT 核，所以为了能保证这些数据也能被处理，就需要一个足够大的 RAM。新来的数据先被暂存起来，等 FFT 核工作完后再送入。从整体上来看，FFT 的工作是不间断的，输入和输出都是连续不断的数据。

在有的系统中，数据是突发的，比如在网络中。有时候会一次来一个数据包，而有时候系统会没有数据，这时候 FFT 是空间的。所以在这种系统中，就不需要那么多的 RAM 资源了。

I/O Data Flow 主要就是设置这个选项。Streaming 针对连续的数据处理，Buffer Burst 和 Burst 都是针对突发数据的情况。Streaming 模型下使用的 RAM 最多，Buffer Burst 又比 Burst 的 RAM 多一些。

假定在突发模式下，突发一个数据包的机率非常高的话。那么相比于

Burst 模式，Buffer Burst 虽然多用了一些 RAM，但是效率更高。因为它有更多的 RAM 用来缓存数据，而对于 Burst，由于没有额外用来缓冲数据的 RAM，只能是收到一个包后就拒绝再收包，直到它把这些数据处理完为止。

4.3.5 FFT Engine Architecture

FFT 运算过程中的数据处理模块，叫做 FFT Engine。可以在参数中设置它的结构和数目。下面引用原手册中的两张图片，说明 Quad 和 Single 两种结构的区别。

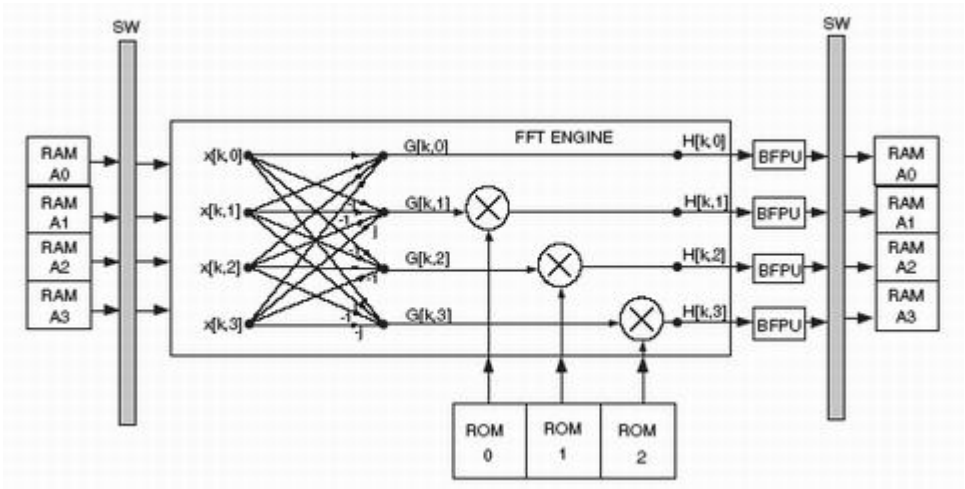


图4.7　Quad Output[1]

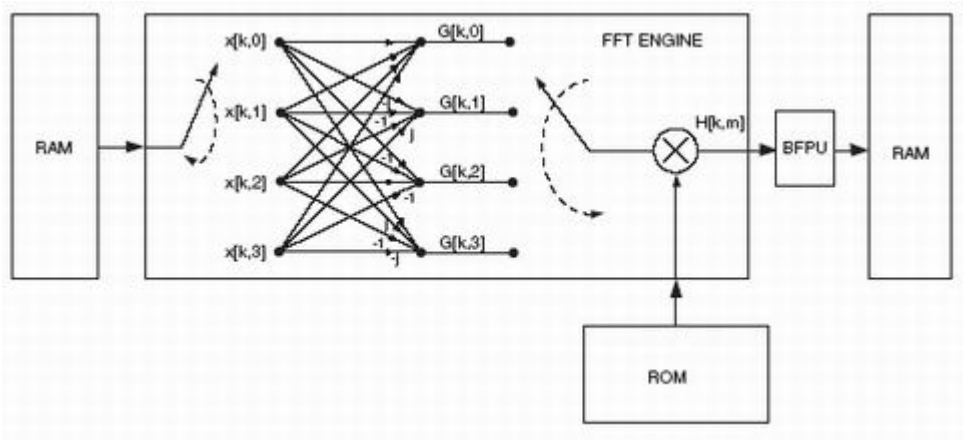


图4.8　Single Output[1]

其实我们可以不用太深究这两种结构到底差别在哪里，只是在性能和资源间进行折衷就好了。Quad Output 比 Single Output 性能好，FFT 引擎数目越多性能也越好，当然，性能越好所需的资源也越多。

4.3.6 复数乘法器实现

在最后一栏“ Implementation Option ”中，主要是对一些具体的实现细节进行设置。

复数乘法器的实现可以选择是用三个乘法器/五个加法器的方案实现还是四个乘法器/两个加法器实现。根据 FFT MegaCore v2.2.0 的勘误表[2]，在 Cyclone II 系列 FPGA 上如果采用四/二的结构，实际性能会和手册上相差很远，值得注意一下。

另外，也可以选择具体用什么资源实现乘法器，是用专门的 DSP Block 还是用纯 LE 实现。

4.3.7 RAM 选项

指主要选择什么样的 RAM，ALTERA 的不同型号 FPGA 中包括不同数目的 M-RAM，M4K 和 M512 等几种 RAM，主要是容量大小不同。

另外还可以指定是不是要使用 RAM 来部分实现用 LE 实现的逻辑功能等。

4.4 FFT MegaCore 的管脚功能和时序

了解了 FFT 的参数设置后，再介绍一下 FFT 的管脚功能和时序，使用这个 Core 就完全没有问题了。

除去 clk 和 reset 端口，FFT MegaCore 的管脚可分为数据管脚和控制管脚。数据管脚包括实部和虚部的输入和输出，以及指数的输出。实部和虚部的宽度由参数设置决定。指数用于对输出数据的调整，这在 4.5 小节会说到。下面重点说控制的管脚。

控制管脚使用的是 ALTERA 的 Atlantic 接口标准，请参考相关资料[3]。下文中只描述每个管脚的作用。

Master_sink_dav 管脚用于指示输入过程的开始。Master_sink_dav 一旦变为“1”就是通知 IP 核开始输入数据了。接下来，在数据输入的过程中，Master_sink_dav 必须一直为“1”。

Master_sink_sop 用于表示数据包的开始。比如 64 点的 FFT，输入的 64 个数据是串行的，master_sink_sop 必须在第一个数据的位置准确的置为“1”，不然会出现数据错位的情况。

Inv_I 表示是 FFT 正变换还是 IFFT 反变换，0 的时候是正变换，1 的时候是反变换。这是由于 FFT 和 IFFT 的运算过程相似而决定的。值得注意的是，这个端口的输入也是可以随时变化的。只要在每个 master_sink_sop 到来的时候做出改变就行。也就是说，FFT 核可以一个包进行 FFT 处理，下一个包进行 IFFT 处理，在实际中有时候非常有用。

输出端有三个管脚 Master_source_ena、master_source_sop 和 master_source_eop 指示着输出的数据情况，master_source_sop 和

master_source_eop 分别指示的输出数据包的开始和结尾，而 master_source_ena 为“ 1 ”表明输出的数据有效。

Master_sink_ena 是一个用于反馈的引脚，表明当前的 FFT 核能否接收数据。当 master_sink_ena 为“ 1 ”表示可以接收，为“ 0 ”表示不能接收。如果在这个引脚为“ 0 ”的时候继续发送数据会造成溢出错误。

Master_source_dav 是另一个重要的控制管脚，当它为“ 1 ”表示允许 FFT 核将处理过的数据输出，当它为“ 0 ”的时候则不输出数据。

举个例子，对于 Streaming 方式连续进行 IFFT 的数据处理：inv_I、master_sink_dav、master_source_dav 都可以一直设置输入为“ 1 ”，自然 master_sink_ena 的输出值也会一直为“ 1 ”，这时候只要正确地输入数据和控制 master_sink_sop 就可以连续地进行 FFT 变换了。

4.5 FFT 变换过程中的指数

在前文中已经叙述过，FFT MegaCore 使用的不是浮点数，但是它又不全是定点数，而是一种特殊的结构。浮点数是 FFT 变换中的每个数各自有各自的指数，而 FFT MegaCore 的结构则是所有的数共用一个指数，这个指数是所有数中最大的数的指数。

在 FFT 的每一级运算中，如果一旦越出了原数据的位数宽度，FFT 核就自动地以最大的数为准进行调整，给它加上一个额外的指数来保证它的数据宽度不变。举个例子，比如在十进制中，只能容纳两位，现在有四个数 110、90、65 和 7 等待调整。那么以最大的 110 为准，把它变成 11 仍保持两位，另外再引入一个系数（乘 10），这样，其它的三个数变为了 9、6 和 0，系数也都是“ 乘 10 ”。注意在这个过程中，相对小的数在缩小的时候被忽略了低位的数值，这也就是为什么 FFT MegaCore 的运算过程会产生误差了。

所以 FFT MegaCore 输入的时候只需要输入数据，输出的时候却多了一个指数部分。我们需要利用这个指数对输出进行调整后才能得到原数据。

因为是二进制系统，所以对于指数的调整其实就是对数据左移位或者右移位。ALTERA 的一个应用文档中对如何调整有着详细的说明[4]，其中的一幅图简单明了，我引用在下面。

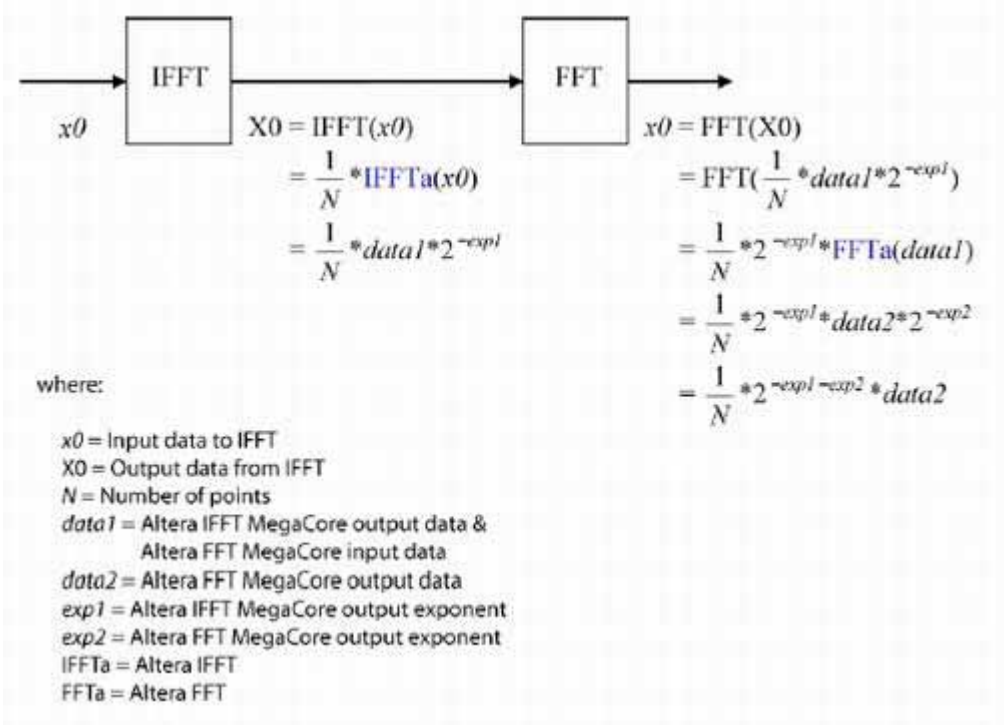


图 4.9　利用指数调整数据[4]

注意 IFFT 因为本身没有除以 N 的运算过程，所以需要再外部进行除以 N 的运算操作。图中的 1/N 就是这么来的。

通过推导出的这个公式，就可以得出原数据了。当然，误差还是会有的，只能通过加大数据位宽来尽量减小误差。

4.6 FFT MegaCore 的 MATLAB 仿真

对于 FFT MegaCore 功能上的验证，其实除了利用 Quartus 的波形

仿真之外，还有其它的几种办法。最方便的是利用 MATLAB 来做验证。

在生产 IP Core 的时候，除了生成设计 FPGA 必要的文件，还会生成一个名称为“ name_model.m ”的 MATLAB 文件，name 是自己为 FFT 模块取的名字。

这个.m 文件是一个 MATLAB 函数，它内容的第一行是这样的：

% function[y, exp_out] = name_model(x,N,INVERSE)

其中 x 是输入的序列，N 是 FFT 点数，INVERSE 为“ 1 ”是 IFFT，“ 0 ”为 FFT。输出 y 和 exp_out 是两个和 X 一样维度的数组。输出序列是 y，exp_out 是输出的指数。

利用这个函数在MATLAB下面计算的结果和FFT实际电路中的运算结果是完全一致的，所以可以用它来实现功能验证。

下面给出了一段 MATLAB 程序作为参考：

程序 test.m

```
% 验证 FFT MegaCore 的功能


clc;
clear;


% 生成两个 64 点的 1 到 100 之间的实数随机序列 a,b
a = rand(1, 64);
a = a * 100;
a = int8(a); % 去掉小数部分
b = rand(1, 64);
b = b * 100;
```

```
b = int8(b); % 去掉小数部分

% 变成复数
x = double(complex(a,b));

% 进行 FFT+IFFT 变换
[y, exp1] = name_model(x, 64, 0);
[z, exp2] = name_model(y, 64, 1);

% 利用指数统一输出的位数
exp1 = exp1(1,1);
exp2 = exp2(1,1);
exp = 2.^(abs(exp1+exp2)-6);
z = z * exp;

% 最后输出结果
result = [x; z]
```

上述程序可以在 MATLAB 中运行，最后显示出输入和输出的结果。可以看出，输入的原数据和经过 FFT+IFFT 的数据之间确有误差，但是当宽度在 16bits 以上的时候，误差就很小了。

另外，有时候 FFT 和 IFFT 的两个部分并不在一起，比如 OFDM 系统的发射和接收机。这样，可以在一部分完成之后马上就用指数进行一下调整，然后再发送给接收端。当然，如果可能，也可以先做完 IFFT+FFT 两部分之后，再使用两个指数一起进行调整。

4.7 小节

本章对 ALTERA 的 FFT MegaCore 做了全面的使用介绍，看完本章之后应该就能完全地掌握其应用了。具有很好的参考意义。

另外，FFT 也是我毕设题目 OFDM 收发机的数据处理部分中的最为重要组成模块。我就用采用 FFT MegaCore 进行设计的。在此进行的介绍也是为了下一章全面介绍 OFDM 基带数据处理部分的铺垫。

参考文献

[1] User Guide, “FFT MegaCore Function User Guide”, ALTERA corp., 2005.

[2] Errata Sheet, “FFT MegaCore Function”, ALTERA corp., 2006.

[3] Functional Specification 13, “Atlantic Interface”, v3.0, ALTERA corp., 2002.

[4] Application Note 404, “FFT/IFFT Block Floating Point Scaling”, v1.0, ALTERA corp., 2005.

第五章　OFDM 硬件设计具体细节

我的毕业设计是关于 OFDM 通信系统的基带信号处理部分的 FPGA 实现，主要工作就是使用 VHDL 语言设计数字电路。我在电路的设计上投入了大量的精力，编写了由 18 个模块组成了电路，主要实现了信道编码、交织、星座映射、FFT 和插入循环前缀的功能，用于 OFDM 系统基带的数字处理部分。

在设计中，由于开始时没有注意到标准的重要性，我没有按照某个标准（比如 802.11a）中对 OFDM 物理层的要求来设计，而且将参数尽可能地做了简化，使整个 OFDM 系统只具有示意的性质。但是，这些简单的模块也仍然有一定的价值，或者设计思路上可以借鉴，或者可以稍加改动应用到更专业的 OFDM 系统中。

本章先介绍了我的一些设计理念，然后开始具体介绍电路的结构和各个模块的技术细节，最后还对仿真和验证的情况进行了介绍。

5.1 设计理念

本小节介绍我在设计 FPGA 中的一些理念。这些理念贯穿着我的整个设计过程。

5.1.1 基于 IP 的设计理念

现在，在 FPGA 的设计中，基于 IP 的设计理念已经深入人心了。所谓 IP（Intellectual Property）可以被理解为实现一定功能的电路模块。因为集成电路的规模越来越大，所以如果所有的设计都由一个人完成的话，效率既低，又容易出错。于是，就出现了一部分专门设计 IP 的工程师，他

们把能实现一定功能的电路经过精心设计构成一个完善的模块，再作为商品出售。这就是 IP。如果在一个大规模的系统中，先把电路按照功能分成几个模块，再使用相应的 IP 来实现的话，就能大大提高效率，加快设计周期，还提高了可靠性。

我在自己的 OFDM 系统中即采用了这样的设计思路，尽可能多地使用 IP 来实现。因此，在 Reed-Solomen 信道编码的部分和 FFT/IFFT 的部分，就采用了 ALTERA 的 MegaCore（IP）来实现。

使用这些 IP，确实提高了我的设计速度。举个例子来说，如果不采用 FFT 的 IP 核，我需要深入地研究 FFT 的原理和内部结构，再转化为 HDL 设计，再进行调试，测试。这需要相当多的时间和精力，也许整个毕设的时间全部来做 FFT 这一个模块都做不完。

但是这种设计思想还有一个最大的障碍，就是 IP 的价格还是太贵了。根据 ALTERA 在中国代理商给我的报价，ALTERA 的几个主要的 MegaCore 的价格分别是：FFT/IFFT，$7995；R-S encoder，$1995；R-S decoder，$7995；Serial Low-speed Viterbi Decoder，$9995；FIR compiler，$2995；NCO compiler，$2495。这个价格对于中国的小企业来说还是支付不起的，我觉得只有大公司或实验室才有能力购买这些 IP。我想，随着基于 IP 设计的思想的不断发展，会有更多的工程师投入 IP 设计的行业，而且 IP 的种类也会越来越多，应用更加广泛，竞争也会更加激烈。在这种趋势下，IP 的价格应该会变得更便宜。

5.1.2 不执著于节约硬件资源的思想

在 FPGA 的设计中，有时候一个电路模块设计完成后，尽管功能上已经完全能够达到要求，但是有些工程师还是会追求在此基础上做进一步的优化，力图使所用的 FPGA 资源达到最少。比如原来需要 9000 个 LE 的

电路经过优化变成了 8000 个 LE。

我觉得这种思路有它的意义所在，一个工程师之所以优秀，一方面就体现在他能在同等条件下做出占用资源更少的电路。但是，我认为不应该过分拘泥于此。特别是在时间和设计能力有限的前提下。

比如说，一个 FPGA 有 10000 个 LE。工程师 A 用了 9000 个 LE 做完了一个工程，这样只剩下 1000 个 LE 还能用做其它功能上的设计。而工程师 B 经过很长时间的优化精减到了 8500 个 LE，这样多省出了 500 个 LE。这看起来非常了不起了。但是现在的集成电路发展这么快，也许再过不久就推出了 20000 个 LE 容量的器件。原来精心省出的 500 个 LE 现在根本不算什么了。

还有，对于 ALTERA 的 FPGA，往往一个系列有四、五种型号。大小有可能分别是 1000、3000、5000、8000 和 10000 个 LE。如果是 9000 个 LE 的工程，那么一定会要使用最大容量的 FPGA，那么即使你将容量精减到了 8500 个 LE，还是需要使用最大的 FPGA 器件，看起来根本没有什么区别。

所以在我的设计中，只是力求使自己写的 HDL 逻辑清晰，功能明确，容易交流。比如某个模块用了 100 个 LE，就不会为能否再继续精减几十个 LE 而费心了。

5.1.3 规范的 HDL 书写风格

对于一个好的编程人员来说，代码一定要写得清晰规范，易于交流。对于 FPGA 的工程师来说，在写 HDL 的时候也是一样的。

我在写 HDL 代码的时候就全部按照一定规范来写代码，每个文件都写了文件头和大量的注释，下面给出了一张截图。

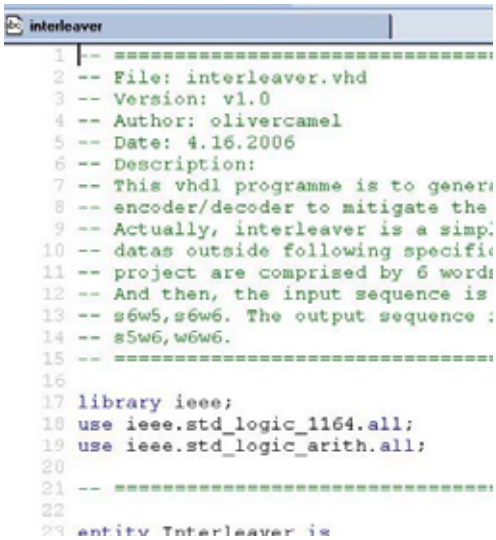


图 5.1　代码截图

5.2 整体系统描述

我设计的数字系统是一个 OFDM 通信系统中的基带数据处理部分，就是不包括变频，射频电路部分的设计，甚至没有加入同步的部分，而是只包括信道编码、交织、星座映射、FFT 和插入循环前缀的部分。这离一个完整的 OFDM 系统还差得很远，不过我觉得自己已经尽力了，也许以后有机会还可以在此基础上再做得更好些。下面叙述一下发射和接收端的具体结构。

5.2.1 发送端结构

发射端的结构框图如下图所示。

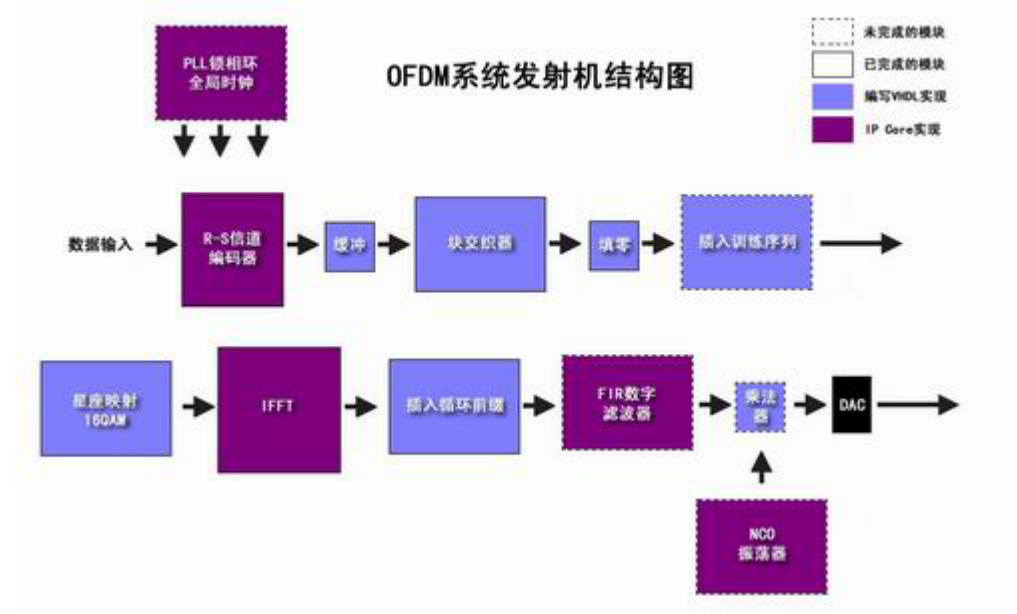


图 5.2　发送端结构示意图

在上图中，蓝色的模块代表用 VHDL 语言设计的模块，红色的模块表示采用了 IP 来实现的模块。外侧实线的模块代表已经完成的模块，虚线的模块代表没来得及做的部分。如果能这些能做完的话，整个系统会更加完善。

在没做完的部分中，NCO 和 FIR 也都可以使用 IP 来实现。

在信道编码部分（FEC）有多种编码技术，分组码主要是 R-S 编码，卷积码主要是 Viterbi 的解码方式。也有两种方式一起使用的，比如 DVB。在 802.11a 的标准中要求使用卷积码。另外也有使用 Turbo 编码的系统。上述的信道编码方式中，R-S 编解码和 Viterbi 解码这两个模块，ALTERA 都提供了 MegaCore。

另外在插入循环前缀以后，可以直接通过数字的方式调制，如结构图中所示。I、Q 支路分别乘以载波后再相加，最后能过 DAC 器件转化为模拟信号，最后通过射频电路发射出去。这一部分的工程量也比较大，所以我没有能够实现。

整个电路中有不少模块都可以采用 IP 来实现，实际中我也是这么设计的，这正是我在前面所阐述的基于 IP 的设计思想的体现。

这一部分的内容，ALTERA 公司有一个白皮书也专门对此进行了描述[1]。

5.2.2 接收端结构

接收端的结构与发射端类似，只不过顺序相反，而且加上了同步的部分。如下图所示。

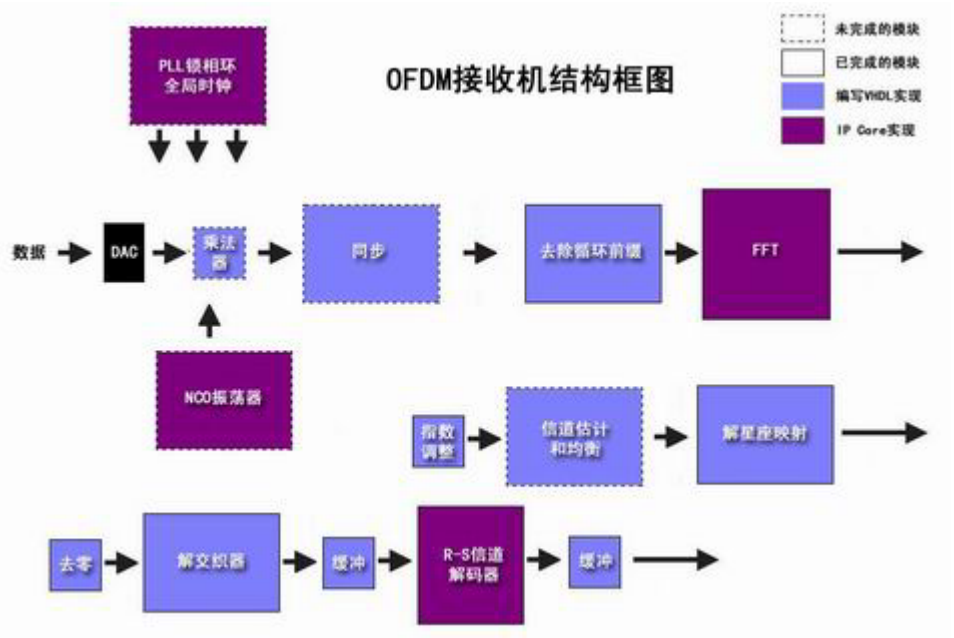


图 5.3　接收端结构示意图

同步模块的主要作用是时钟同步和帧定位。比较重要的还有信道估计和均衡部分。这些在硬件上实现起来都是比较复杂的，我也没有能够完成。

在 802.11a 的标准中 FEC 部分使用的是卷积码，所以解码部分可以使用 Viterbi MegaCore 来实现。而在我的设计中使用的是 R-S 编码和解码。

5.3 各模块功能描述

本小节开始具体描述我设计的每个模块电路的技术细节。在叙述之前，还要先对接口，数据流的连续与离散这两个方面的问题进行一下讨论。

5.3.1 各模块间统一的接口

ALTERA 的所有 IP 都使用统一的 Atlantic 接口，规范了系统中各个模块的通信[2]。受这种思路启发，我也在自己的系统中设计了统一的接口。我所采用的这种接口是在 Atlantic Interface 的基础上进行了简化，去掉了 master 和 slave 的区别。

使用统一接口一定会浪费一些资源，因为需要生成接口电路的逻辑。但是采用了统一接口之后，为系统的设计带来了极大的方便。这些好处主要包括两点：第一，在设计一个模块的时候不必考虑其它的模块，只需要让当前设计的模块符合接口的要求即可。第二，减小了系统调试出错的可能。当几个模块串接在一起调试时，如果没有一致的接口，模块间往往会互相干扰，导致出现一些单独调试一个模块时没有出现过的错误。

所以说，为设计接口多花些资源是完全有必要的。这也是上文叙述的设计思想，不要过分追求资源优化的体现。

具体来讲，每个模块的接口部分包括八个端口，与数据输入（sink）相关的有四个，与数据输出（source）相关的也有四个。其中比较重要的有两个，一是在输入端的 source_ena，一个是在输出端的 sink_ena。

Source_ena 表示输出使能，当为“ 1 ”时表示当前模块可以输出数据，为 “ 0 ” 时不能输出数据。

Sink_ena 是个输出信号，当它为“ 1 ”时表示当前模块可以接受数据，为 “ 0 ” 时表示不能再接受数据。这个信号通常作为一个反馈信号连接到上一模块的 source_ena 端，用来控制上个模块的输出。

这样每个模块环环相扣，就形成了一个链条，代表了数据流的方向。如下图所示。

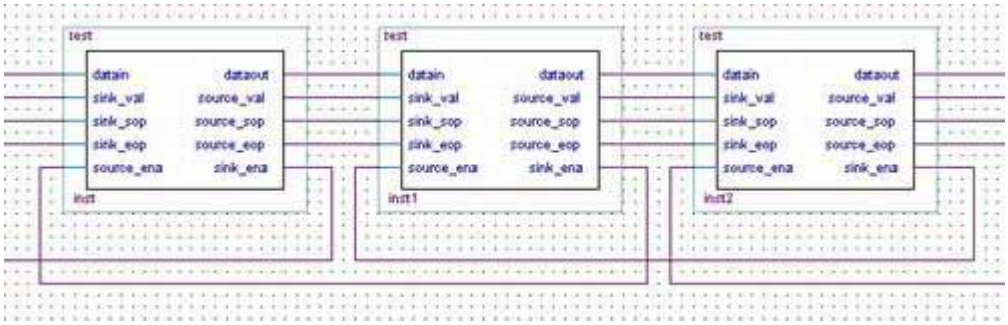


图5.4　接口信号示意图

如果最后一个模块在处理中不能再接收数据，它就通过 sink_ena 通知上一个模块不要再发送数据。中间的模块不能发送数据了，而又没有容量再接纳新的数据，就再通过 sink_ena 通知再上一个模块的 source_ena。这样就可以逐级反馈给整个系统。

另外还有六个端口，分别是输入端的 sink_sop，sink_eop，sink_val 和输出端的 source_sop，source_eop，source_val。其中，sop 和 eop 分别表示一个数据包的开始和结尾，val 则表示数据有效。

这八个端口构成了一个完善地基于数据包的接口模式。

5.3.2 如何设计一个连续数据流的系统

众所周知，有些 OFDM 系统要求能够处理连续的数据流，而有些基于 OFDM 的系统是针对数据包的系统设计的。通过前面小节对于接口的描述，很明显地，我所设计的系统中也是针对数据包的。

但是一个针对处理数据包而设计的系统，也可以应用于要求数据流连续的情况下。只需要在系统前后各加上一个足够大的 FIFO，再适当提高中间系统频率即可。

连续的数据流先存入输入端的 FIFO，而处理过的数据包则不断地堆积

到输出端的 FIFO 中，再连续地输出。这样从外部看起来，整个系统的输入输出都是连续的数据流，虽然在内部是以数据包的方式被处理的。为了能保证内部数据处理的过程不耽搁时间，往往内部时钟频率相对外部的数据流的时钟要高一些，这才能使数据“流动”起来。

有些连续处理数据的系统要求内部的每个小模块都能连续地处理数据，这就要求在内部使用不同的时钟。因为像插入循环前缀的模块，输入比如是 64 个数据，但是输出就要超过 64 个，比如是 80 个。如果它的时钟也跟前面的模块时钟一致，那么一定会造成还有 16 个数据没来得及输出，而下一个 64 的数据已经到来了。所以循环前缀模块的时钟速率一定要高于前面的模块。

在系统内部使用不同时钟往往会造成逻辑比较复杂，还不如上面第一种方法来得简单。所以我认为在可以满足需要的前提下，使用第一种方法更好。

5.3.3 R-S 编解码模块

本小节开始正式介绍每个模块的具体细节。

Reed-Solomon 编码，是通信系统中一种常用信道编码方式。信道编码的码型主要包括分组码和卷积码，R-S 码属于分组码中的循环码。

这种编码方式主要通过在数据包末尾添加用于纠错的冗余数据，在接收端，通过冗余数据检测并自动纠正在传输过程中可能出现的错误。对于 R-S 编码，假定数据长度为 d 个字节，附加的冗余数据为 t 个字节，那么实际的传输的数据长度为 L = d+t 个字节。t 个字节的冗余长度可以纠正 t/2 个字节的错误。例如，如果数据长度为 2 个字节，冗余数据长度为 4 个字节，实际传输的数据长度就是 6 个字节。那么在这 6 个字节中，只要出错的字节数不超过两个，就可以完全纠正过来，保证整个通信的过程正

确无误[3]。

这种信道编码的方式实质上是一种典型的牺牲效率而提高可靠性的方法。

在 OFDM 通信系统中，我采用的即为上述的编码方式。R-S 编解码器均由 ALTERA 公司的 MegaCore 实现。每个数据包 6 个 word，每个 word 由 4 个 bit 组成。6 个 word 中，有 2 个是数据，4 个是用于纠错的冗余数据。

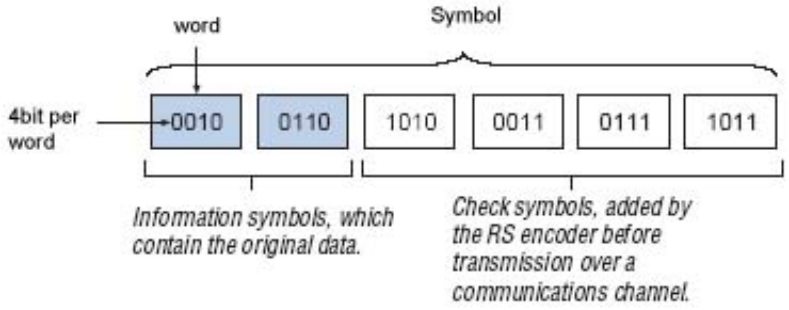


图 5.5　R-S 编码符号示意图[3]

这种两个数据加四个冗余的方式能保证只要错在两个以内，就都能纠正过来。但是效率显然比较低。我考虑因为是 OFDM 示意性的系统，另外可以方便地修改 R-S MegaCore 的参数，就没有过多在意这个问题。

5.3.4 R-S encoder 输出缓冲

R-S encoder 的输出端需要一个缓冲器。它收集够 36 个 word 后交给交织器。当交织器在工作的时候就继续缓存数据。

5.3.5 块交织器和解交织器

块交织器与 R-S 编码器组合使用，能够进一步降低数据传输中由突发差错引起的误码。块交织器的实质就是一块 RAM，通过一定的顺序对 RAM

进行读和写，可以将几个数据包打乱顺序发送出去。这样的话，如果信道中由于突发差错引起了一连串的错误，在接收端，通过解交织器，错误就被离散到不同的数据包中了，之后可以再通过编程编码的方式纠正错误。

我采用每个数据包(symbol)6 个 word，每个 word 由 4bit 构成。用于交织器的 RAM 设计为 36*4bit 容量。根据块交织器的原理，写入 RAM 的顺序为：s1w1,s1w2,s1w3,s1w4,s1w5,s1w6,s2w1,s2w2,……，s6w5，s6w6；输出的顺序为：s1w1，s2w1，s3w1，s4w1，s5w1，s6w1，s1w2，s2w2，……，s5w6，s6w6。通过这种读写的方式即可以实现交织作用。

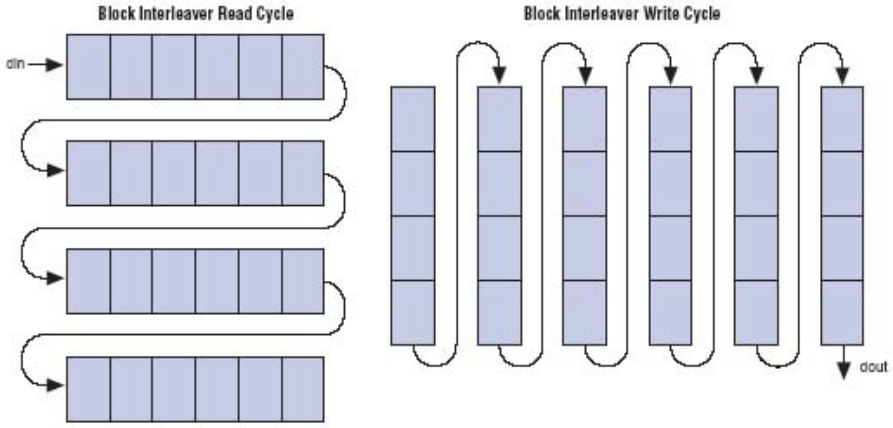


图 5.6　块交织器示意图[4]

解交织器的思路和交织器的类似。对于 6*6 方形结构的交织器，在接收端的解交织器和发送端的交织器完全是一样的，因此既简化了设计又可以实现资源的复用。

5.3.6 填充数据零

交织器输出的数据包为 36 个 words。为了进行 64 点的 IFFT，这个部分为每个数据包插入 28 个 0，扩大到 64 点。

此外，在 OFDM 中，需要插入导频数据，其实原理也填充数据零也是

一样的。

5.3.7 星座映射和解映射

星座映射是 OFDM 通信系统中比较重要一部分，它将传输过来的数据映射成具有一定规则的星座。它的主要作用有两个：一是将数据规则化，变成经过设计的星座；另一个是为数据引入虚部，使数据流变成复数的数据流，可以进行 FFT 的处理。

在 OFDM 中比较常用的映射方式有 QPSK，8PSK，16QAM，64QAM。在我的毕业设计中，采用了 16QAM 的映射方式。这种方式的好处在于它有 16 个星座位置，效率比 QPSK 和 8PSK 要高，相比 64QAM，实现起来又要简单一些。而且，我采用的每个 word 中 4 个 bit 的数据结构正好就有 16 种可能，可以与 16 个星座相对应。我使用 96，32，-33，-97 四个十进制数字的组合构成 16QAM 的星座。

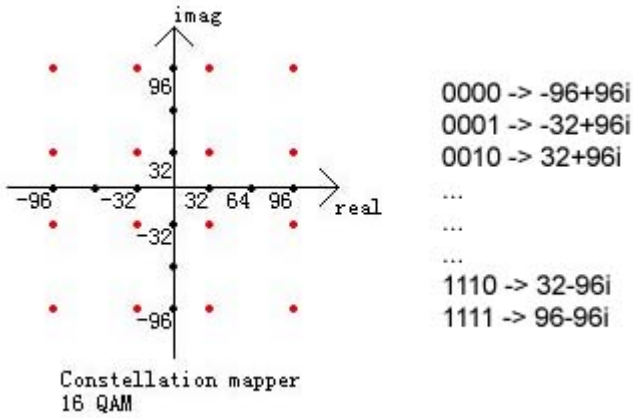


图5.7　16QAM 星座图

在解映射的部分采用的是硬判决，就是根据星座的位置画一个格子，落在格子里面的数就认为是这个星座。

星座映射的实质其实就是一个数据上的变换，将输入的每个字节的数据变为约定的星座。比如假设输入为“ 0001 ”，对应输出是-33+96i，实

际上就是将输入的 4bit 数据变成了两路 10bit 的数输出。之所以输出设计为 10bit，是为了照顾到后面 FFT 变换的精度。

总的来说，通过星座映射，将一路 4bit 的输入信号变成了两路 10bit 的输出信号。

5.3.8 IFFT 和 FFT

FFT 部分是整个 OFDM 发射和接收机中最核心的部分，通过它才能真正实现正交的功能。在第四章已经对 FFT 作了详细介绍，再此就不再多加叙述了。

5.3.9 插入和移除循环前缀

循环前缀为 OFDM 符号引入了时间上的保护间隔。经过 64 点的 FFT，输出的数据包为 64 个数据，在这个模块中再加上 16 个点的前缀。把 64 个数据中的最后 16 个复制到 64 个包的前面。

移除循环前缀位于接收机，作用是把这个 16 个前缀再次去掉。

因为 FFT 输出分为实部和虚部，所以各需要两个插入和移除循环前缀的模块。

5.3.10 其它

此外，还有其它一些小模块也在系统中发挥着作用。

FFT Scaler 模块用于调整 FFT+IFFT 变化后的指数。实质就要将数据乘以 8。

Zero Remover 模块位于解交织器和 R-S 解码器之间，作用是把发送端插入的 28 个零去除。

系统的最后一个模块是输出缓冲模块，实质是一块 FIFO，把收到的数

据积累起来再输出。

5.3.11 关于 VHDL 原代码

在文章中，我没有花太多时间去介绍 VHDL 原代码，因为似乎没有必要介绍每句 VHDL 代码的作用。所以在文章中，我更多的介绍了自己的设计思路，而不是代码本身。

在附录中，我也没有加入所有的原代码，只是加入了顶层文件和另外两个典型模块的代码。如果把全部的代码都放在附录中的话，将占用大量的篇幅。代码文件已经随论文一同交给指导老师了。

另外，全部的代码和相关资料我都放在了自己的主页上，需要的话可以去下载，http://www.olivercamel.com。如果有机会，我愿意和别人多交流，讨论和发现我的不足之处。我的邮箱是：olivercamel@gmail.com。

5.4 系统的验证

5.4.1 系统编译情况

在 VHDL 代码设计完成后，剩下的验证工作就相对简单了。因为 Quartus 软件的功能十分强大，剩下的工作就都交给它来完成即可。

下图显示模块的编译情况，整个系统共用了 9000 多个 LE 和 15000 多 bits 的 RAM。

Flow Status	Successful - Wed May 31 22:24:39 2006
Quartus II Version	5.1 Build 213 01/19/2006 SP 1 SJ Web Edition
Revision Name	ofdm
Top-level Entity Name	ofdm_down
Family	Cyclone
Device	EP1C12Q240C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	9,204 / 12,060 (76 %)
Total pins	13 / 173 (8 %)
Total virtual pins	0
Total memory bits	15,296 / 239,616 (6 %)
Total PLLs	0 / 2 (0 %)

图5.8　模块编译信息

5.4.2 软件仿真

利用 ALTERA 的时序仿真分析功能可以对模块的进行功能验证。如下图所示。经过仿真，确认模块的功能没有问题。

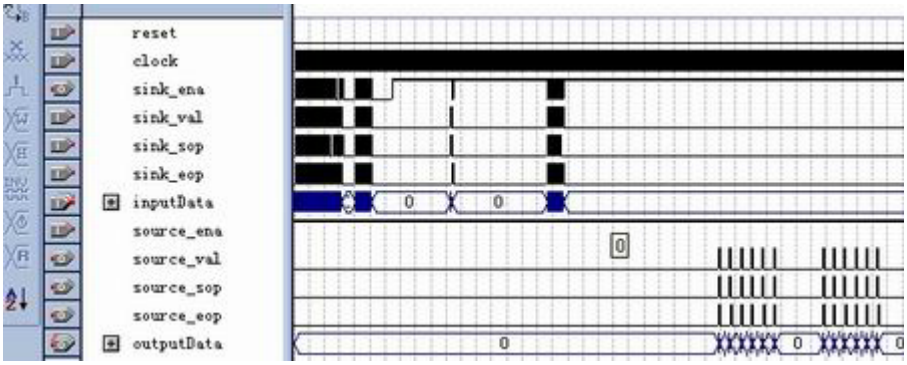


图5.9　模块仿真时序

5.4.3 FPGA 硬件验证

虽然一般来说，如果时序仿真没有问题的话，在硬件上运行的时候也不一定会有问题。但是，作为 FPGA 的设计最后一个步骤，还是一定要下载到芯片中进行验证。

在硬件的验证中，我使用的是一个非常普通的 FPGA 实验板，芯片是

EP1C12Q240C8，系统时钟为 16M。因为系统的输入和输出都是 4bits 的数据，所以我把输入接到四个开关上，输出接到了四个灯上。

经过硬件的验证，确实证明了我写的程序在逻辑上是正确无误的。

5.5 可以做得更好（结束语）

至此，所有关于毕设的内容就介绍完了。虽然我花费了一个学期的时间来尽力把毕设做好，但是由于时间和个人能力的原因，整个系统看起来还是显得非常的简单，只实现了一些最基本的功能。

其实还有许多的地方可以做得更好，比如对 OFDM 理论部分的学习，对接收端的同步和均衡部分的设计，Viterbi 解码的部分，A/D 和 D/A 的部分，FIR 滤波器的部分，甚至射频电路的设计等等。这些都成为了遗憾。也许以后有机会还可以在此基础上再做得更好些吧。

我一直认为毕业设计重在过程。确实是这样的。这个毕设的过程，其实也就是我不断学习的过程。在这个过程中我学到了许多新的知识，能力也提高了不少。这些收获给我带来的喜悦远远超过了完成毕设题目时给我带来的喜悦。我相信，以后再做这些设计的时候一定还可以做得更好。

参考文献：

[1] White Paper, “Implementing OFDM Using Altera Intellectual Property”, v1.0, ALTERA corp., 2001.

[2] Functional Specification 13, “Atlantic Interface”, v3.0, ALTERA corp., 2002.

[3] User Guide, “Reed-Solomon Complier User Guide”, v4.0.1, ALTERA corp., 2005.

[4] User Guide, “Symbol Interleaver/Deinterleaver MegaCore Function User Guide”, v1.3, ALTERA corp., 2002.

附录一：部分 VHDL 原代码

块交织器模块

由于篇幅的限制，全部的原代码请到 http://www.olivercamel.com 下载。

```
-- ====================================================================
-- File: Interleaver.vhd
-- Version: v1.1
-- Author: olivercamel
-- Date: 4.16.2006
-- Description:
-- This vhdl programme is to generate a Interleaver which works together with R-S
-- encoder/decoder to mitigate the effects of noise in communications system.
-- Actually, interleaver is a simple RAM controller that writes datas into or reads
-- datas outside following specific orders. For example, symbols we used in this
-- project are comprised by 6 words. Throughput of the interleaver is 6 symbols.
-- And then, the input sequence is s1w1,s1w2,s1w3,s1w4,s1w5,s1w6,s2w1,s2w2,...,
-- s6w5,s6w6. The output sequence is s1w1,s2w1,s3w1,s4w1,s5w1,s6w1,s1w2,s2w2,...,
-- s5w6,w6w6. A 4bit * 64words RAM named ram_Interleaver is required in the codes.
-- Revision History:
-- v1.1, 4.27.2006, deassert sink_ena earlier to avoid a control bug.
-- ====================================================================

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

-- ====================================================================

entity Interleaver is
    port
    (
        -- clock input
        clk: in std_logic;
        -- asynchroism clear input
        aclr: in std_logic;
        -- 4bit width Data ports
        inputData: in std_logic_vector (3 downto 0);
        outputData: out std_logic_vector (3 downto 0);
        -- simple ALTERA Atlantic interface ports
```

```
            sink_val: in std_logic;
            sink_sop: in std_logic;
            sink_eop: in std_logic;
            sink_ena: out std_logic;
            source_val: out std_logic;
            source_sop: out std_logic;
            source_eop: out std_logic;
            source_ena: in std_logic
        );
end Interleaver;

-- ================================================================

architecture structure of Interleaver is

-- --------------------------------------------------------------------------------

-- component declaration
-- generate by ALTERA ip toolbench
-- name: ram_Interleaver
-- size: 4bit * 64words
component ram_Interleaver
        PORT
        (
            aclr: IN STD_LOGIC;
            clock: IN STD_LOGIC;
            data: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            rdaddress: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
            rden: IN STD_LOGIC;
            wraddress: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
            wren: IN STD_LOGIC;
            q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
        );
end component;

-- signals those connect to RAM
signal clk_ram: std_logic;
signal aclr_ram: std_logic;
signal inputData_ram: std_logic_vector (3 downto 0);
signal outputData_ram: std_logic_vector (3 downto 0);
signal readEnable_ram: std_logic;
```

```
signal writeEnable_ram: std_logic;
signal readAddress_ram: std_logic_vector (5 downto 0);
signal writeAddress_ram: std_logic_vector (5 downto 0);

-- signals those used to Atlantic interface outputs
signal interval_source_val: std_logic;
signal interval_source_sop: std_logic;
signal interval_source_eop: std_logic;
signal interval_sink_ena: std_logic;

-- flag signal indicates status: '1' input process, '0' output process`
signal inoutFlag: std_logic;

-- write address
signal wrAddNum: integer range 0 to 36;
-- read address
signal rdAddNum: integer range 0 to 36;
-- write enable
signal writeEnable: std_logic;
-- read enable
signal readEnable: std_logic;

-- delayd signals
-- sink_eop delay
signal sink_eop_d: std_logic;
-- readEnable delay
signal readEnable_d0: std_logic;
signal readEnable_d1: std_logic;
-- interval_source sop and eop delay or acceleration
signal interval_source_sop_d0: std_logic;
signal interval_source_sop_d1: std_logic;
signal interval_source_eop_a: std_logic;
signal interval_source_eop_d0: std_logic;
signal interval_source_eop_d1: std_logic;

-- ---------------------------------------------------------------------------

begin

-- ---------------------------------------------------------------------------
```

```
-- part1: ram connections
-- ram ports map
u1: ram_Interleaver port map
	(
		clock => clk_ram, aclr => aclr_ram,
		data => inputData_ram, q => outputData_ram,
		rdaddress => readAddress_ram, wraddress => writeAddress_ram,
		rden => readEnable_ram, wren => writeEnable_ram
	);

-- integer converts to std_logic_vector
readAddress_ram <= conv_std_logic_vector(rdAddNum,6);
writeAddress_ram <= conv_std_logic_vector(wrAddNum,6);

-- readEnable delay
process(clk,readEnable)
begin
	if falling_edge(clk) then
		readEnable_d0 <= readEnable;
		readEnable_d1 <= readEnable_d0;
	end if;
end process;

writeEnable_ram <= writeEnable;
readEnable_ram <= readEnable_d0;

clk_ram <= clk;
aclr_ram <= aclr;

inputData_ram <= inputData;

-- outputs data at clk's falling edge
process(aclr,clk,outputData_ram)
begin
	if aclr = '1' then
		outputData <= "0000";
	else
		if falling_edge(clk) then -- using falling edge
			outputData <= outputData_ram;
		end if;
	end if;
```

```
end process;
-- ----------------------------------------------------------------------------

-- part2: generate Flag signal inoutFlag

-- delay input signal sink_eop
process(clk,sink_eop)
begin
	if rising_edge(clk) then
		sink_eop_d <= sink_eop;
	end if;
end process;

-- control inoutFlag
process(clk,aclr,sink_eop_d,interval_source_eop_a)
begin
	if aclr = '1' then
		inoutFlag <= '1';
	else
		if rising_edge(clk) then
			if inoutFlag = '1' then
				if sink_eop_d = '1' then
					inoutFlag <= '0';
				end if;
			else
				if interval_source_eop_a = '1' then
					inoutFlag <= '1';
				end if;
			end if;
		end if;
	end if;
end process;

-- ----------------------------------------------------------------------------

-- part3: generate read/write Enable/Address

-- generate writeEnable
process(aclr,inoutFlag,sink_val)
begin
	if aclr = '1' then
```

```
            writeEnable <= '0';
        else
            if inoutFlag = '1' then
                writeEnable <= sink_val;
            else
                writeEnable <= '0';
            end if;
        end if;
end process;

-- generate readEnable
process(clk,aclr,inoutFlag,source_ena)
begin
        if aclr = '1' then
            readEnable <= '0';
        else
            if falling_edge(clk) then -- using falling edge
                if (inoutFlag = '0') and (source_ena = '1') then
                    readEnable <= '1';
                else
                    readEnable <= '0';
                end if;
            end if;
        end if;
end process;

-- write address
process(clk,aclr,writeEnable,sink_sop,sink_eop)
begin
        if aclr = '1' then
            wrAddNum <= 1;
        else
            if falling_edge(clk) then -- using falling edge
                if writeEnable = '1' then
                    if wrAddNum = 36 then
                        wrAddNum <= 1;
                    else
                        wrAddNum <= wrAddNum + 1;
                    end if;
                end if;
                if sink_sop = '1' then
```

```
                    wrAddNum <= 2;
                elsif sink_eop = '1' then
                    wrAddNum <= 1;
                end if;
            end if;
        end if;
end process;

-- read address
process(clk,aclr,source_ena,interval_sink_ena)
begin
        if aclr = '1' then
            rdAddNum <= 0;
        else
            if falling_edge(clk) then -- using falling edge
                if readEnable = '1' then
                    if rdAddNum = 0 then
                        rdAddNum <= 1;
                    elsif rdAddNum = 31 then
                        rdAddNum <= 2;
                    elsif rdAddNum = 32 then
                        rdAddNum <= 3;
                    elsif rdAddNum = 33 then
                        rdAddNum <= 4;
                    elsif rdAddNum = 34 then
                        rdAddNum <= 5;
                    elsif rdAddNum = 35 then
                        rdAddNum <= 6;
                    else
                        rdAddNum <= rdAddNum + 6;
                    end if;
                end if;
                if rdAddNum = 36 then
                    rdAddNum <= 0;
                end if;
            end if;
        end if;
end process;

-- ---------------------------------------------------------------------------
-- part4: Atlantic interface signals
```

```
-- source sop eop is controlled by rdAddNum
-- generate interval_source_eop_a at the same time
process(rdAddNum)
begin
    case rdAddNum is
        when 36 =>
            interval_source_eop <= '1';
            interval_source_sop <= '0';
            interval_source_eop_a <= '0';
        when 1 =>
            interval_source_sop <= '1';
            interval_source_eop <= '0';
            interval_source_eop_a <= '0';
        when 30 =>
            interval_source_sop <= '0';
            interval_source_eop <= '0';
            interval_source_eop_a <= '1';
        when others =>
            interval_source_sop <= '0';
            interval_source_eop <= '0';
            interval_source_eop_a <= '0';
    end case;
end process;

-- source_sop source_eop delay and output
process(clk,interval_source_sop)
begin
    if rising_edge(clk) then
        interval_source_sop_d0 <= interval_source_sop;
        interval_source_sop_d1 <= interval_source_sop_d0;
    end if;
end process;

process(clk,interval_source_eop)
begin
    if rising_edge(clk) then
        interval_source_eop_d0 <= interval_source_eop;
        interval_source_eop_d1 <= interval_source_eop_d0;
    end if;
end process;
```

```
process(clk,interval_source_sop_d1,interval_source_eop_d1)
begin
	if falling_edge(clk) then -- using falling edge
		source_sop <= interval_source_sop_d1;
		source_eop <= interval_source_eop_d1;
	end if;
end process;

-- generate sink_ena
process(clk,aclr,wrAddNum,rdAddNum)
begin
	if aclr = '1' then
		interval_sink_ena <= '1';
	else
		if rising_edge(clk) then
			if wrAddNum = 35 then -- edited in v1.1 deassert sink_ena earlier
				interval_sink_ena <= '0';
			elsif rdAddNum = 36 then
				interval_sink_ena <= '1';
			end if;
		end if;
	end if;
end process;

sink_ena <= interval_sink_ena;

-- generate source_val
process(clk,readEnable_d1)
begin
	if falling_edge(clk) then -- using falling edge
		interval_source_val <= readEnable_d1;
	end if;
end process;

source_val <= interval_source_val;

-- ---------------------------------------------------------------------------

end structure;

-- ============================================================
```

附录二：外文翻译

Iterative Timing Recovery

Methods for implementing timing recovery in cooperation with iterative error-control decoding.

John R. Barry,
Aleksandar Kavčić,
Steven W. McLaughlin,
Aravind Nayak, and
Wei Zeng

The last decade has seen the development of iteratively decodable error-control codes of unprecedented power, whose large coding gains enable reliable communication at very low signal-to-noise ratio (SNR). A by-product of this trend is that timing recovery must be performed at an SNR lower than ever before. Conventional timing recovery ignores the presence of error-control coding and is thus doomed to fail when the SNR is low enough. This article describes iterative timing recovery, a method for implementing timing recovery in cooperation with iterative error-control decoding so as to approximate a more complicated receiver that jointly solves the timing recovery and decoding problems.

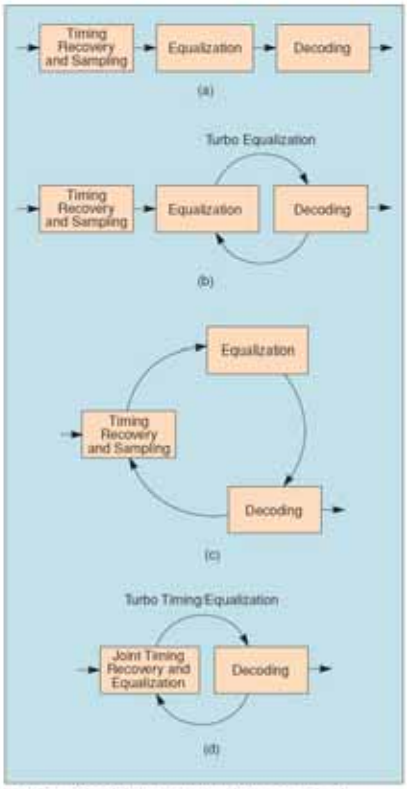
Timing Recovery Problem

At some point in a digital communications receiver, an analog waveform must be sampled. Sampling at the right times is critical to achieving good overall performance. The process of synchronizing the

1053-5888/04/$20.00©2004IEEE

sampler with the pulses of the received analog waveform is known as timing recovery.

The timing recovery problem might not be difficult in isolation, but a practical receiver must not only perform timing recovery but also detection of the transmitted message, which involves such tasks as equalization and error-control decoding. In doing so it must contend with not only uncertainty in the timing of the pulses but also with additive noise and intersymbol interference (ISI). In principle, one could formulate the problem of jointly determining the maximum-likelihood estimates of the timing offsets and the message bits. The solution to this problem would naturally perform the tasks of timing recovery, equalization, and decoding jointly. Unfortunately, the complexity would be prohibitive. Instead, a conventional receiver performs these tasks separately and sequentially, as illustrated in Figure 1(a). Specifically, a conventional timing recovery scheme ignores error-control coding, assuming instead that the transmitted symbols are mutually independent.



▲ 1. Conventional timing recovery with (a) conventional equalization and with (b) turbo equalization. Iterative timing recovery with (c) separate timing recovery and equalization and (d) joint timing recovery and equalization.

The conventional separation approach of Figure 1(a) works reasonably well at a high SNR but not at the low SNRs supported by capacity-approaching iteratively decodable codes. As an extreme example [1], a recent rate-1/31 code designed for a deep-space application will operate reliably when the SNR is less than −15 dB! At this low SNR, a timing recovery scheme that ignores the code will likely fail. In this article, we describe an iterative method for jointly performing the tasks of timing recovery and error-control decoding with complexity comparable to a conventional receiver.

Perhaps the first iterative timing recovery method is due to Georghiades and Snyder, who applied the expectation-maximization (EM) algorithm for the case of constant offset with an ideal, uncoded system [2]. Since then there have been many extensions to account for ISI, time-varying offsets, and error-control coding [1], [3]–[5]. There are also close ties between timing recovery and carrier recovery, and so the recent work on using iterative concepts to estimate carrier phase is relevant [1], [6]–[8].

In this article, we present some emerging iterative timing recovery schemes. In Figure 1(a), a classical (noniterative) receiver is shown, where the signal flow is unidirectional from the timing recovery unit to the decoding unit. The advent of turbo codes [9] has prompted the turbo equalization architecture of Figure 1(b), where the signal flow from the timing recovery unit is unidirectional while there is an iterative exchange between the equalizer and the decoder. The scenario of Figure 1(b) will not be further examined in this article since it is covered in detail in a different article of this issue [10]. Instead, we focus on two methods that extend turbo equalization to include timing recovery iterations. The first such method is schematically depicted in Figure 1(c), where a timing recovery step has been added to each iteration of a turbo equalizer. The second method [depicted in Figure 1(d)] merges the timing recovery unit and the soft-output equalizer into a single unit that performs joint timing recovery and soft-output equalization in a manner similar to the BCJR [11] or Baum-Welch algorithm [12].

Classical Timing Recovery

In this section we present a brief overview of the most widely used method for timing recovery; namely, a decision-directed, phase-locked loop.

A digital transmitter conveys information by modulating the amplitudes of a sequence of pulses. In theory, these pulses are transmitted at a fixed rate—the baud rate $1/T$—that is known to the receiver. In practice,

however, the baud rate may vary slowly with time, or the channel may introduce jitter because of Doppler effects or other nonidealities. Moreover, manufacturing imperfections may cause the transmitter and receiver clock frequencies to differ by a fraction of a percent. The reality is that the receiver never knows the precise arrival times of the pulses.

We consider the following simple model for the received waveform $r(t)$

$$r(t) = \sum_k a_k b\left(t - kT - \tau_k\right) + n(t), \qquad (1)$$

where $a_k \in \{\pm 1\}$ is the kth binary symbol (or amplitude), $b(t)$ is the received pulse shape, T is the signaling interval anticipated by the receiver, and $n(t)$ is additive Gaussian noise. The uncertainty in the timing is captured by the offset parameter τ_k, defined as the difference between the actual and expected arrival time of the kth pulse. For example, when the transmitter and receiver clock frequencies differ, τ_k increases linearly with time.

Our restriction to binary alphabets is sufficient to capture the gist of the timing recovery problem. Only minor modifications are needed to handle higher-order complex alphabets.

The optimal sampling times are $\{kT + \tau_k\}$. The job of a timing recovery scheme is to estimate the timing offsets before sampling. Nearly all existing timing recovery schemes are based on a *phase-locked loop* (PLL), which is easily described in terms of the following key definitions:

Let $\hat{\tau}_k$ denote the receiver's estimate of τ_k.
Let $\epsilon_k = \tau_k - \hat{\tau}_k$ denote the residual error in this estimate.
Let $\hat{\epsilon}_k$ denote the receiver's estimate of ϵ_k.

A second-order PLL estimates τ_k according to the recursion

$$\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha\hat{\epsilon}_k + \beta\sum_{i=0}^{k}\hat{\epsilon}_i, \qquad (2)$$

where α and β are the proportional and integral step sizes, respectively. In other words, the offset estimate is found by accumulating the output of a loop filter with transfer function $\alpha + \beta/(1 - z^{-1})$ and input $\hat{\epsilon}_k$. A block diagram of the PLL is shown in Figure 2.

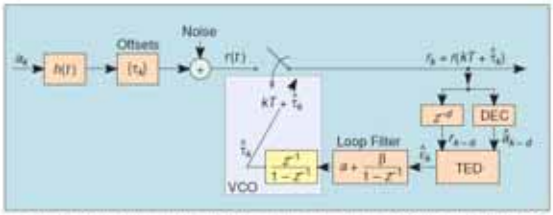
The PLL recursion (2) can be motivated heuristically as follows. Consider the case when β is zero, in which case (2) reduces to a first-order PLL. At time k, the receiver knows its current estimate $\hat{\tau}_k$, and it also has access to an estimate $\hat{\epsilon}_k$ of the residual error $\tau_k - \hat{\tau}_k$. If this estimate were somehow known to be precisely accurate, then the receiver could cancel the remaining residual timing error in only one step by choosing $\hat{\tau}_{k+1} = \hat{\tau}_k + \hat{\epsilon}_k$. In practice, however, $\hat{\epsilon}_k$ is a very noisy estimate of $\tau_k - \hat{\tau}_k$, and by attenuating $\hat{\epsilon}_k$ by a factor $\alpha < 1$, the PLL essentially low-pass filters the noise. The downside of this attenuation is that it will take more than one step to correct for the residual error, which implies that the PLL will be less agile when tracking a time-varying offset. In practice, the step size $\alpha \in (0, 1)$ is chosen to trade off the opposing goals of robustness to noise and agility. A nonzero value of β takes into account long-term trends in τ_k, giving a second-order PLL the ability to track a frequency offset with zero average steady-state error.

The device that generates $\hat{\epsilon}_k$ is called a timing-error detector (TED), and its effectiveness is key to the overall performance. A popular choice is the Mueller and Müller (M&M) TED [13]

$$\hat{\epsilon}_k = r_k a_{k-1} - r_{k-1}a_k, \qquad (3)$$

where $r_k = r(kT + \hat{\tau}_k)$. This TED is most accurate when there is no ISI, so that $b(kT) = 0$ for nonzero integers k. Together, (2) and (3) define the conventional approach to timing recovery.

The M&M TED of (3) requires knowledge of the transmitted symbols $\{a_k\}$, which is generally available to the receiver only during an initial start-up or training phase. In practice, a receiver without training will implement timing recovery in decision-directed mode by replacing $\{a_k\}$ in (3) by tentative decisions. The decision device that generates these decisions will invariably introduce a processing delay, which we denote by d. Therefore, as shown in Figure 2, a decision-directed PLL is driven by $\{\hat{a}_{k-d}\}$ instead of $\{a_k\}$. The performance of a decision-directed PLL depends critically on two parameters: 1) reliability of the decisions and 2) processing delay d of the decision device. There is a fundamental tradeoff between these two parameters, since reliability can generally be increased at the expense of processing delay. The importance of



▲ 2. The traditional approach to timing recovery is based on a decision-directed PLL. DEC stands for decision device, whose processing delay is d. In practice, the shaded block is implemented by passing the loop filter output through a voltage controlled oscillator (VCO). Here we model the VCO as a simple integrator.

译文

在过去的十年间，可解码的循环差错控制编码（iteartively error-control code）有了空前的发展。通过编码，相当于为通信带来了很大的信号增益，使一个可靠的通信系统可以在信噪比非常低的条件下实现。这个发展趋势带来的一个副产品是定时恢复（timing recovery）必须要在更低的信噪比情况下进行。传统的定时恢复方法并没有把纠错编码的作用考虑进去，所以在信噪比非常低的时候注定会失效。本文介绍了循环定时恢复的方法（iterative timing recovery），这是一种可以和循环纠错编码协同工作的定时恢复方法。可以通过一个相对更复杂的接收机一次性实现定时恢复和解码的功能。

定时恢复中的问题

从数字通信的角度来说，模拟的波形在接收端一定要被采样。选择正确的采样时刻对整个系统的性能来说至关重要。通过收到的模拟波形来同步采样器的过程就叫做定时恢复。

定时恢复的问题单独来看也许并不复杂，但是实际中的接收机不仅要执行定时恢复，而且还要接收被发送的数据，完成包括均衡和差错控制解码的工作。在这些工作中，接收机不仅要面对定时脉冲相位不确定的问题，

还要解决由加性噪声和符号间干扰（ISI）所造成的影响。从理论上讲，同时决定定时偏置和信息比特的最大似然估计的问题可以被公式化。所以把定时恢复、均衡和解码三者放在一起处理自然是解决问题的最好方法。不幸的是，由于过于复杂，这种系统是不能实现的。因此，传统的接收机都是分别依次这些工作的。如图一所示。值得一提，传统的定时恢复都忽略了差错控制编码的作用，把收到的符号看作是相互独立的。

图一中这种传统的分立的方法在高信噪比的情况下性能相当好，但是却不适用于采用循环差错编码后的低信噪比的情况。举个极端的例子[1]，最近，一个为深度空间（deep-space）设计的编码速率为 1/31 的系统，竟然可以稳定工作在信噪比低于 15dB 的情况下。在这种情况下，如果定位恢复的设置忽略了差错编码作用的话，基本上是不能正确工作的。在本文中，我们将描述一种循环的方法来同时地完成定时恢复和差错控制解码的工作，而且复杂程度与传统接收机差不多。

第一个提出循环定时恢复的当属 Georghiades 和 Snyder，他们在偏置值是常数的理想、未编码的系统中使用了期望值最大化（EM）算法[2]。从那时候开始，在此基础上又不断地为 ISI、时变偏置和差错控制编码做了许多扩展[1], [3] - [5]。定时恢复和载波恢复有着密切的关系，所以最近的工作都是在利用循环的概念去估计载波的相位[1], [6] - [8]。

在这篇文章中，我们将展现一些新兴的循环定时恢复的方案。在图一

（a）中描述的是一种典型的（非循环）接收机，它的数据流从定时恢复单元到解码单元是单向的。Turbo 码的出现导致 Turbo 均衡结构的提出，如图一（b）。这种结构是指，从定时恢复单元输出的数据流仍然是单向的，但是在均衡器和解码器之间有一个循环交互的过程。图一（b）中的这种方案在文章中将不再做过多的讨论了，因为已经有许多不同的文章叙述过了这一问题[10]。我们将关注的是另外两种在 Turbo 均衡基础上包括了循环定时恢复的扩展方案。第一个方案的原理框图见图一（c）。定时恢复的过程被加入进来，Turbo 均衡器每循环一次都要执行一次定时恢复。第二种方案，图一（d），把定时恢复单元和软输出（soft-output）均衡器合并成了一个单元，同时完成定时恢复和软出来均衡的功能。这有点像 BCJR[11]或是 Baum-Welch 算法[12]。

经典定时恢复方法

在这部分，我们将概括地回顾一下被最广泛应用的经典的定时恢复方法，即采用硬判决和锁相环技术。

数字发射机通过调制脉冲序列的幅度来传递信息。在理论上，这些被发送的脉冲应该有一个固定的速率，如波特率 1/T，这应该是接收机预先知晓的。但是在实际中，波特率可能会随时间缓慢地发生变化，或者信道也会由于多普勒效应及其它非理想的情况而引入一些抖动。而且，制造工

艺上的缺陷也会造成发射机和接收机时钟频率的不一致，经常是一个小数的偏差。所以实际的情况是，接收机不可能精确地知道脉冲序列的到达时间。

我们来看下面这个简单的接收波形 r(t)的模型。

$$r(t) = \sum_k a_k h\left(t - kT - \tau_k\right) + n(t), \qquad (1)$$

其中 $a_k \in \{\pm 1\}$ 代表第 k 个二进制符号（或者幅度），$h(t)$是批接收到的波形形状，T 是接收机预先知道的信号周期，n(t)是加性高斯白噪声。假设用偏置参数 τ_k 来代表定时的不确定性，它定义为第 k 个脉冲的理论到来时刻和实际到来时刻的差。举例来说，如果发射机和接收机的时钟不一致，那么 τ_k 随时间是线性变化的。

上述模型中，我们把发送信号的幅度限制为二进制。因为这已经能够充分抓住定时恢复问题的要点了。如果采用多进制，只需要再稍加改动即可。

最佳的采样时刻是在{kT+ τ_k }，定时恢复的作用就是要在采样之前估计这个偏置。几乎正在所有的定时恢复方案都是基于锁相环的，它可以被简单地用下面几个关键定义来描述：

令 $\hat{\tau}_k$ 表示接收机对 τ_k 的估计；

令 ϵ_k = τ_k - $\hat{\tau}_k$，表示上这个估计中的残留偏差；

令 $\hat{\epsilon}_k$ 表示接收机对于 ϵ_k 的估计。

一个二阶的锁相环通过下面的递归公式来估计 τ_k ：

$$\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha\hat{\epsilon}_k + \beta\sum_{i=0}^{k}\hat{\epsilon}_i, \qquad (2)$$

其中，α 和 β 分别表示比例和积分级数的系数。从另一个角度来说，对偏置的估计，是在输入为 $\hat{\epsilon}_k$ 的情况下，通过对一个传递参数为 $\alpha + \beta/(\hat{1} - z^{-1})$ 的环路滤波器的输出不断累加的结果。这种结构的锁相环如图二所示。

通过锁相环的递归公式（2），我们可以获得如下启发。假定 β 等于零。这时公式（2）变成了一个一阶的锁相环。在 k 时刻，接收机估计出了当前的 $\hat{\tau}_k$，并通过 τ_k - $\hat{\tau}_k$ 得到了 $\hat{\epsilon}_k$。如果这是非常精确的话，那么接收机可以只通过令 $\hat{\tau}_{k+1}$ = $\hat{\tau}_k + \hat{\epsilon}_k$ 这一步即可消除剩余的残留定时误差。但是在实际中，$\hat{\epsilon}_k$ 对于 τ_k - $\hat{\tau}_k$ 的估计伴有噪声。所以为了通过 α<1 来衰减 $\hat{\epsilon}_k$，锁相环本质上就是低通过滤噪声。这种衰减的过程需要不只一次的过程来校正残留误差，这意味着锁相环捕捉一个随时间变化的偏差时，灵活性将受到影响。在实际中，系数 α 其实是在灵活性和系统抗噪声能力之间的一个折衷。另一个非零的系数 β 的设计，则是考虑到对 τ_k 的长期趋势的观察。它为二阶锁相环赋予了在稳态误差的均值为零时捕捉频率偏置的能力。