****

**硕士学位论文**

**面向C-RAN的虚拟化硬件加速器的研究与实现**

姓 名：汤绍先

学 号：1433273

所在院系：电子与信息工程学院

学科门类：工学

学科专业：计算机科学与技术

指导教师：吴俊 教授

二〇一七年三月

****

A dissertation submitted to

Tongji University in conformity with the requirements for

the degree of Master of Engineering Science

**Research and Implementation of Virtualization Hardware Accelerator for**

**C - RAN**

Candidate: Shaoxian Tang

Student Number: 1433273

School/Department: Electronic and Information

Engineering

Discipline: Engineering

Major: Computer Science and Technology

Supervisor: Jun Wu

March, 2017

|  |
| --- |
| **面面向**  **C**  **-**  **R**  **A**  **N**  **的虚拟化硬件加速器的研究与实现**  **汤绍先**  **同济大学** |

**学位论文版权使用授权书**

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年 月 日

**同济大学学位论文原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年 月

**摘要**

C-RAN是新型的云接入网，使用集中式的基站资源池进行基带处理，能提高资源的利用率并提升网络性能。通用处理器实现的虚拟化平台，基带处理带宽有限，且目前相对功耗较高，限制了基站池的规模。因此使用硬件对基带处理模块进行加速，提高基站池的处理能力是研究的重点方向。

本文首先研究了LTE物理层基带处理流程，重点分析了耗时较长的Turbo译码和FFT/IFFT模块。研究了硬件加速器的种类，最后采用FPGA开发板实现这些模块的加速器，并将这些加速器应用于LTE的C-RAN网络中。本文选用Xen虚拟化技术搭建了C-RAN虚拟化平台，并设计了基于PCIe总线的硬件加速器高带宽数据传输接口，连接硬件加速器和LTE的C-RAN网络平台。

本文设计的Turbo译码硬件加速器，采用流水线模式，支持数据输入、译码和数据输出并行处理。且加速器工作在250MHz高频时钟下，是DMA数据传输时钟的4倍，可以最大程度减少运算时间，提高流水线的并发程度。通过LTE 5MHz带宽的测试，加速器将原来占LTE物理层处理流程近一半时间的译码操作时间减少了2/3以上，并且加速器支持更高带宽和多用户共享硬件加速器。

本文根据LTE物理层的处理流程，在已有的C-RAN平台中FPGA基带板上，设计并实现了FFT和IFFT硬件加速器，都支持流水线处理模式。IFFT加速器实现了CP的自动添加。FFT加速器实现了频移和去CP的操作，同时还实现了软件同步硬件执行和数据流切换操作。

将高时延的基带处理部分转移至硬件加速器中，减少了虚拟化平台处理的负荷，提高了LTE支持的带宽。设计的FFT和IFFT硬件加速器，支持多个相同加速器在同一个基带板上同时工作，具有较高的可移植性。通过仿真和板上测试表明，本文设计的Turbo译码硬件加速器和FFT/IFFT硬件加速器能够实现C-RAN中高带宽的云基站池处理任务，满足基站处理的延迟要求。

**关键词：**C-RAN，硬件加速器，Turbo，FFT，FPGA

**ABSTRACT**

C-RAN is a new type of cloud access network, it uses the centralized base station resource pool for baseband processing to improve resource utilization and the network performance. The virtualization platform, implemented by general processor, has limited baseband processing bandwidth and high power consumption, which limits the size of the base station pool. Therefore, the use of hardware to accelerate the baseband processing module to improve the processing capacity of the base station pool is the focus of the study.

This paper first studies the LTE physical layer baseband processing flow, focusing on the analysis of the long time-consuming turbo decoding and FFT/IFFT module. And the paper studies the types of hardware accelerators, the accelerators of these modules are implemented by FPGA development board, and these accelerators are applied to LTE C-RAN networks. This paper chooses the C-RAN virtualization platform by using Xen virtualization technology, and designs a high-bandwidth data transmission interface based on PCIe bus, which connects hardware accelerator and LTE C-RAN network platform.

This article designs the Turbo decoding hardware accelerator, using the pipeline pattern, supporting the data input, the decoding and the data output parallel processing. And the accelerator works at 250MHz high frequency clock, which is 4 times than the DMA data transfer clock. It can minimize the computing time and improve the degree of concurrent pipeline. With the LTE 5MHz bandwidth test, the accelerator reduced the decoding operation time by nearly two-thirds of the LTE physical layer processing, and the accelerator supports higher bandwidth and multi-user shared hardware accelerators.

Based on the processing flow of LTE physical layer, FFT and IFFT hardware accelerators are designed and implemented on FPGA baseband boards in existing C-RAN platforms, which support the pipeline processing mode. IFFT accelerator to achieve the automatic addition of CP. FFT accelerator implements the frequency shift and the CP remove operation, the software synchronization hardware execution and data flow switching operations.

The part of high latency baseband processing is transferred to the hardware accelerator, which reduces the load on the virtualization platform and improves the LTE bandwidth. The FFT and IFFT hardware accelerators supports multiple identical accelerators working on the same baseband board at the same time and has high portability. The simulation and on-board test show that the Turbo decoding hardware accelerator and FFT/IFFT hardware accelerator designed in this paper can realize the high-bandwidth cloud base station pooling task in C-RAN and meet the delay requirement of base station processing.

**Key Words:** C-RAN, hardware accelerator, Turbo, FFT, FPGA

**目录**

[第1章 绪论 1](#_Toc477307181)

[1.1 C-RAN背景与研究意义 1](#_Toc477307182)

[1.2 虚拟化概述 3](#_Toc477307183)

[1.3 硬件加速器概述 4](#_Toc477307184)

[1.4 FPGA开发流程 7](#_Toc477307185)

[1.5 论文主要内容、创新点与结构安排 9](#_Toc477307186)

[1.5.1 论文主要内容 9](#_Toc477307187)

[1.5.2 论文创新点 9](#_Toc477307188)

[1.5.3 论文结构安排 9](#_Toc477307189)

[第2章 C-RAN平台与硬件加速器接口设计 11](#_Toc477307190)

[2.1 LTE物理层概述 11](#_Toc477307191)

[2.1.1 LTE物理层简介 11](#_Toc477307192)

[2.1.2 LTE物理层处理流程 11](#_Toc477307193)

[2.3 Xen虚拟化平台架构 12](#_Toc477307194)

[2.3.1 Xen虚拟化技术 12](#_Toc477307195)

[2.3.2 分离设备驱动模型 13](#_Toc477307196)

[2.3.3 PCIe前后端虚拟驱动 14](#_Toc477307197)

[2.4 加速器硬件接口设计 14](#_Toc477307198)

[2.5 本章总结 16](#_Toc477307199)

[第3章 Turbo译码硬件加速器设计 17](#_Toc477307200)

[3.1 硬件结构设计 17](#_Toc477307201)

[3.2 Turbo译码算法 19](#_Toc477307202)

[3.3 Turbo译码硬核设计 19](#_Toc477307203)

[3.4 处理逻辑设计 20](#_Toc477307204)

[3.5 流水线设计 22](#_Toc477307205)

[3.6 调度处理 23](#_Toc477307206)

[3.7 功能仿真 23](#_Toc477307207)

[3.8 本章总结 25](#_Toc477307208)

[第4章 FFT/IFFT硬件加速器设计 27](#_Toc477307209)

[4.1 基站FFT处理 27](#_Toc477307210)

[4.2 FFT硬核设计 27](#_Toc477307211)

[4.3 IFFT硬件加速器设计 29](#_Toc477307212)

[4.3.1 硬件结构设计 29](#_Toc477307213)

[4.3.2 IFFT运算逻辑 31](#_Toc477307214)

[4.4 FFT硬件加速器设计 31](#_Toc477307215)

[4.4.1 数据流切换 31](#_Toc477307216)

[4.4.2 频移逻辑 33](#_Toc477307217)

[4.4.3 FFT运算逻辑 33](#_Toc477307218)

[4.5 功能仿真 34](#_Toc477307219)

[4.5.1 IFFT加速器功能仿真 34](#_Toc477307220)

[4.5.2 FFT加速器功能仿真 36](#_Toc477307221)

[4.6 本章总结 38](#_Toc477307222)

[第5章 加速器的验证与测试 41](#_Toc477307223)

[5.1 测试平台及测试指标 41](#_Toc477307224)

[5.2 测试方案设计 42](#_Toc477307225)

[5.3 板级测试与性能分析 42](#_Toc477307226)

[5.4 本章总结 46](#_Toc477307227)

[第6章 总结与展望 47](#_Toc477307228)

[6.1 总结 47](#_Toc477307229)

[6.2 展望 47](#_Toc477307230)

[**致谢** 49](#_Toc477307231)

[**参考文献** 51](#_Toc477307232)

[**附录** 55](#_Toc477307233)

[**个人简历、在读期间发表的学术论文与研究成果** 57](#_Toc477307234)

1. 绪论

1.1 C-RAN背景与研究意义

近年来，随着移动通信行业的高速发展，网络业务和智能终端设备的种类丰富多样。与此同时，终端设备对运营商的网络容量和带宽都提出了更高的要求。由于日益增长的基础设施资本支出和大量的能源消耗，移动运营商急需解决高额的网络建设成本和增长缓慢的用户平均收入（ARPU，Average Revenue Per User）之间的矛盾。

传统的无线接入网（RAN，Radio Access Network）具有许多特点：第一，基站的覆盖范围固定，只限于附近的区域，且不能处理临近区域的信号；第二，固定区域的系统容量有限，难以应对用户数量大规模的增加；第三，基站一般拥有多种不同网络制式的标准，如GSM、WCDMA、TD-SCDMA和LTE等，这些网络标准需要不同的平台和设备分开处理。因此，传统无线接入网的发展面临巨大挑战，移动运营商需要解决基站覆盖范围小和容量有限、不同标准网络的平台难以兼容和升级等问题。

面对移动设备的高速增长，数据的流量也呈现爆炸式增加，为了满足移动互联网等业务的需求，移动运营商需要不断增加基站的数量和空口的带宽，但是这些更多的传统的基站产生了巨额的资本和运营支出。同时，移动网络负载具有随时间迁徙的现象，在负载较高时段会导致处理能力的紧缺，但在负载较低时段却又造成大量处理能力的浪费，这种潮汐效应导致基站无法充分使用。

为解决这些挑战并追求未来可持续的增长，中国移动研究院提出了新型的无线接入网架构C-RAN(Cloud Radio Access Network)，并发布了C-RAN白皮书。白皮书中提出了未来的无线接入网的目标，主要包括：降低基站能源的开销，减少设备资本和运营的不必要投入；提高频谱效率，增加用户带宽；更开放的平台，可更好的兼容多种标准和更方便的升级；对终端提供更好更快的数据通信服务等。C-RAN是通过使用云计算的虚拟集中式的基带处理池、可协作的无线网络和基于开放平台的实时无线接入网，可以解决传统无线接入网所面临的问题，并实现未来的无线接入网的目标。

C-RAN架构在基站基带处理的资源管理方面引入了云计算技术。区别于传统的基站，云计算技术中的基站将资源集中起来，并通过高速光纤传输，在资源分配上可以达到充分利用并兼容多种标准。在进一步提升基站性能的同时，实现无线网络资源的共享和动态分配，降低能耗和成本。在C-RAN中，将基带处理单元集中起来，通过使用云计算技术，建立一个可扩展和开放的云计算平台，提高处理单元的资源的动态分配能力和资源的虚拟化。

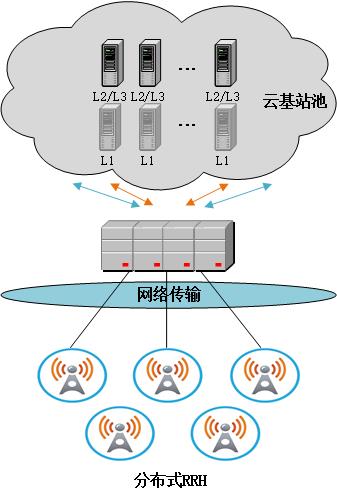


图1.1 C-RAN架构

C-RAN的架构如图1.1所示，在基站基带的处理部分使用云计算的虚拟化技术，将基带资源集中起来，并通过虚拟化技术进行分配。比如中国移动提出的C-RAN 网络构架，与图1.1的架构很相似。在基带处理中，将基带的处理单元与射频单元分开，处理单元集中部署到主要的地区，可以方便设备的维护和升级，提高运维的效率。射频单元根据需求，可以部署在需要的站点和地区，并通过高速的网络连接到基带处理单元，网络连接一般使用光纤等高速物理设备连接。这种部署的方式，可以充分利用虚拟化的技术优势，部署得到高动态可配置的云基站处理资源池。目前，在多个城市中已经拥有了基于C-RAN架构的基站资源集中化部署的实验，在不久的将来，C-RAN架构将在更多的城市广泛使用。

通用处理器平台具有良好的开放性、可扩展性和易于研发维护的优点，因此采用通用处理器完成 C-RAN基站功能是一个很好的选择。但是目前使用通用处理器作为处理的平台，能耗也相对较高，这限制了C-RAN 中基站池的规模。引入低功耗处理器架构和物理层硬件加速单元，实现更高密度的载波处理能力，并通过对这些资源的虚拟化，从而不影响通用处理器所具有的开放性和扩展性。

基站资源虚拟化是在集中基带RAN架构基础上的进一步架构创新，能够容纳多制式进行集中基带资源共享和调度，并能够具备更灵活的部署能力和容灾能力。对基站硬件加速器资源虚拟化的研究主要在包括两方面，一是用何种技术完成虚拟化的要求；另一个是作为加速器的硬件的选择。在本章的后面两节将对虚拟化技术和硬件加速器做简要的介绍。

本文的意义和应用价值在于引入虚拟物理层硬件加速单元，使云基站的架构具有更好的开放性和扩展性。实现基站处理单元硬件加速器资源的共享，实现资源动态可分配，并充分利用硬件资源，减少不必要的浪费。

1.2 虚拟化概述

在计算机中，虚拟化技术是一种资源抽象的技术。通过使用此技术，可以将计算机的硬件进行抽象，将物理资源抽象为虚拟的逻辑资源。通常可以将一个物理资源抽象为多个逻辑资源，当然也可以将多个物理资源抽象为一个逻辑资源，抽象之后的逻辑资源可以让应用使用。为了获得不同的使用要求，在虚拟化技术中，发展出了多种方案。通常可以根据虚拟化实现的方式，硬件抽象的层次和部分不同，分为不同的虚拟化方式，并具有不同的功能特点。由于计算机本身的层级特性，可以根据层次结构对每个层级进行抽象并虚拟化。所以按照抽象层次的不同，一般可以将虚拟化技术分为五种：指令级虚拟化、硬件级虚拟化、操作系统级虚拟化、应用级虚拟化和程序库级虚拟化。

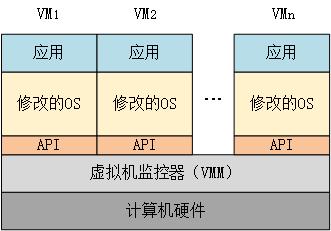
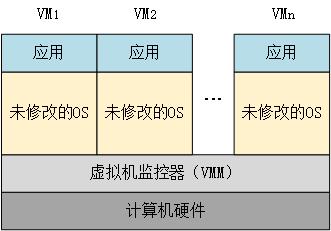


图1.2 全虚拟化方式 图1.3 半虚拟化方式

在操作系统级的虚拟化中，根据虚拟机的操作系统是否需要修改，可以分为全虚拟化(Full Virtualization)和半虚拟化(Para-Virtualization)。全虚拟化又称为硬件辅助虚拟化，是指在虚拟机和硬件中间加了一层管理虚拟机的程序，叫做虚拟机监控器（VMM，Virtual Machine Monitor），利用VMM来处理CPU指令，翻译后送给虚拟机。因此，VMM的资源消耗较大，会影响虚拟机的性能，造成一部分的浪费。但是其优点也很明显，就是可移植性很强，不需要对操作系统进行修改，只要底层的硬件支持虚拟化技术就可以使用。全虚拟化的方式如图1.2所示。一般全虚拟化的产品有：VMware、KVM、Xen、Virtual Box等。半虚拟化又称准虚拟化，是指在虚拟机的内核中需要根据需要进行更改，增加了专门的API接口代替VMM进行指令的处理。这样可以减少管理所需资源的消耗，提高虚拟化的性能。但是需要对操作系统进行更改，对于不能更改的操作系统则不能使用此虚拟化技术。半虚拟化与全虚拟化不同的特点可以从图1.3中看出，在操作系统中增加了API来处理指令。其中，半虚拟化典型的产品为Xen。

Xen 是一种开源的虚拟化技术，并且技术比较成熟，拥有出色的虚拟机的性能，且占用的资源比其他的虚拟化技术更少。使用Xen虚拟化技术，可以将一个物理机虚拟出多个相同的可用虚拟机，并且在Linux中添加了相关的支持，与KVM相似，可以很容易在Linux中进行相关的开发。在服务器的使用中，Xen作为成熟且高性能的虚拟化技术，可以充分利用服务器的资源，为使用人员节省一笔很可观的开支，并且有利于运维的管理，可以最大程度的提高硬件设备的利用率。

I/O 设备的虚拟化对于任何虚拟化技术来说都是必不可少的一部分，Xen 对于I/O 设备提供了两种不同的虚拟化方式，一种是半虚拟化，可以最大程度的提升系统的性能，减少Xen虚拟化方面的开销；另一种是完全虚拟化，用户可以在虚拟化的且未做修改的操作系统中使用，使用情况与未虚拟化的操作系统完全相同，用户不会感觉到有任何差别。Xen使用半虚拟化技术，可以最大程度的减少虚拟机管理方面的开销，一般情况下这方面的开销都是在3%以下，在虚拟机操作频繁的情况下，也不会超过10%。当使用全虚拟化技术时，用户可以轻松的将自己的产品放在虚拟化的未修改的操作系统中，实现快速部署。但是使用这种虚拟化的开销，一般情况下会达到20%左右，这对于非常追求性能和高效的产品来说，是不太能接受的。所以目前来说，在全虚拟化的性能与半虚拟化还有明显差距的时候，高性能要求的项目选择虚拟化技术实现的场景显而易见。Xen作为开源的虚拟化技术，以其高效的资源利用率，在对性能要求很高的场景中使用广泛，所以在C-RAN中可以选择Xen来开发虚拟化的硬件加速器。

1.3 硬件加速器概述

硬件加速器（Hardware Accelerator）就是使用硬件设备来替代软件，利用硬件比软件更加高效和快速的特点，进行算法和运算的实现，达到加速处理的目的。硬件一般具有高并发和高频等特性，硬件加速通常比软件算法的效率要高。在硬件加速器中使用的硬件包括可用于计算处理的芯片或处理器，通常所用的有ASIC、GPU、ASIP、FPGA等。

专用集成电路，ASIC（Application Specific Integrated Circuit）效率高、功耗低，可以设计出用户指定的集成电路，并且可以减少系统的外部连线，提高系统的可靠性。但是由于用户需要自行考虑芯片中的布局布线和工艺等问题，设计难度较高，且需要较大的投入，在后期的成熟产品中使用比较合适，不太适用于早期的产品开发阶段。当产品可以大批量生产时，使用ASIC可以显著的降低成本，获得更高的产品性价比和系统性能。

图形处理器，GPU（Graphics Processing Unit）是在计算机和移动设备等上进行图像处理工作的微型处理器。在显卡中，最核心的即为GPU，与计算机中的CPU类似，主要完成一些运算处理操作。不同的是，CPU可以主要负责程序代码的执行，而GPU执行的操作主要涉及复杂的浮点矩阵等运算和并行计算等。因此对于图像和高复杂度计算方面的应用，使用GPU可以显著的提高效率和性能。目前世界上一些超级计算机都使用带有很多GPU的核心，来增强计算的性能，这种在通用计算领域的叫做GPGPU（General Purpose GPU），同时通用并行计算CUDA（Compute Unified Device Architecture）平台的提出，可以使用C语言在平台上编程，方便了GPU的开发。但是GPU的功耗很高，大规模使用的能耗高于大部分其他硬件。

专用指令集处理器，ASIP(Application Specific Instruction Set Processor)针对特定应用进行定制化设计，能在功耗和灵活性间取得较好的折衷。相比ASIC会有更大的灵活性，设计的风险更低，比GPGPU又有更低的功耗，在嵌入式的系统中应用较多。数字信号处理器DSP（Digital Signal Processor）是一种专门用于数字信号处理的微处理器。因此ASIP和DSP具有类似的应用，都是特别定制使用的硬件。

FPGA（Field Programmable Gate Array），即现场可编程门阵列，是一种可重复编程，灵活度很高的编程器件。FPGA作为可编程的器件，是在ASIC这种集成电路发展后出现的半定制的电路，不仅解决了定制电路灵活性差的缺点，也大大提高了编程器件的元器件数目，使得开发更加便捷简单。作为FPGA的发明者和提供商，Xilinx发布了全新的28nm的产品，大大减少了FPGA的功耗，并提升了性能，使FPGA的应用场景更加丰富。FPGA的开发，一般使用硬件描述语言（HDL，Hardware Description Language）来完成电路的设计。一般选用Verilog或VHDL或者基于Verilog基础上的新的语言SystemVerilog（SV）进行开发，然后经过综合与布局布线，将生成的文件烧录至FPGA上进行测试，这种开发方法是现代IC设计验证的主流。

FPGA作为加速计算密集型应用较多，在通信领域用途广泛。Xilinx的FPGA系列，拥有众多各领域的IP模块可简化设计。国外有使用Xilinx的Virtex-6的FPGA，作为硬件加速器设计的解决方案：pvFPGA，该设计采用了Xen虚拟机监视器（VMM）来构建一个半虚拟化的环境。文献将FPGA用于稀疏矩阵向量乘法的加速，在性能和功耗上与使用GPU等进行了分析。在云计算中使用FPGA也有相关的研究，文献将FPGA的高吞吐、可编程和低功耗带入了云计算中，并通过Openstack搭建框架原型，使用基于KVM的虚拟技术，实现了具有良好性能的云计算应用。在涉及VPN网络的安全问题中，加密问题的加速处理也是可以使用基于FPGA实现。

传统的研究中，有许多是关于FPGA的通用计算，例如BORPH项目，是基于FPGA的可重构计算机操作系统。也有些文献研究了FPGA的虚拟化应用，但仅限于单个操作系统上的多任务处理，并不是面向C-RAN的领域的硬件加速器。

除了使用FPGA作为硬件加速器，还有很多使用GPU等作为图像处理等方面的加速器，文献采用了GPGPU来加速图像处理。文献对GPU和FPGA在加速器领域的应用进行了分析。在嵌入式系统中使用的ASIP，提供了面向某个特定领域制定硬件计算加速器的功能。

由于FPGA具有运行速度快、可实现大规模系统、运算能力强、方便进行二次开发的特点，被大量运用于无线通信系统中。因此，将虚拟化运用到FPGA硬件加速器中，可以很好地解决资源浪费、利用效率低等缺点，同时可以实现满足云基站要求的硬件加速器。所以在LTE的物理层中，可以使用FPGA来作为硬件加速器进行开发，在商用之后可以考虑使用固化的ASIC来降低功耗和成本，获得更高的资源利用效率。

在与加速器硬件的数据交互接口选择中，由于需要高带宽的数据传输，所以一般选用PCIe接口。PCIe是高速的串行点对点的双通道高带宽的差分传输总线，是重要的I/O接口标准。PCIe总线的频率和编码方式是影响每个通道峰值带宽的因素。主要的关系和传输带宽可如表1.1所示。

表1.1 PCIe总线性能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PCIe版本 | 编码方式 | 总线频率 | 原始传输率 | X1带宽 |
| 1.0 | 8b/10b | 1.25GHz | 2.5 GT/s | 250MB/s |
| 2.0 | 8b/10b | 2.5GHz | 5 GT/s | 500 MB/s |
| 3.0 | 128b/130b | 4GHz | 8GT/s | 984.6 MB/s |

在无线通信中引入虚拟化技术，实现C-RAN云基站的虚拟化硬件加速器，可以在很大程度上提高资源的利用率和数据处理的吞吐率。通过研究面向 C-RAN 的低功耗通用处理器架构，研制出支持虚拟化的硬件加速器功能单元，使多用户可以通过虚拟机同时访问硬件加速器，完成数据的及时处理。

1.4 FPGA开发流程

FPGA是可编程的器件， FPGA的设计和开发就是利用电子设计自动化（EDA，Electronic Design Automatic）开发软件，对FPGA的器件进行设计开发。开发一般采用自顶向下的设计方式，即采用独立于厂商的硬件描述语言对功能进行设计实现，并配以功能仿真验证功能的正确性；之后通过综合生成的具体芯片上的网表文件，布局布线完成实际板上的器件编程，最后在板上进行测试验证。主要的开发流程的步骤如图2.4所示。

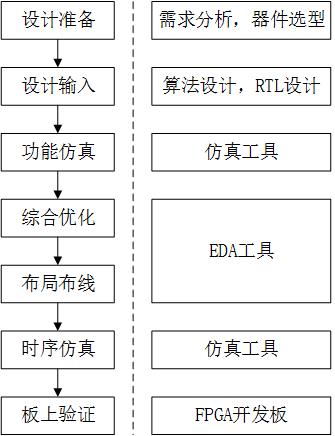


图2.4 FPGA开发流程

1. 设计准备

在进行设计之前，需要做好需求分析，了解所需设计的功能、实现的方案和使用的算法等情况。还要考虑开发板的具体性能，以选择合理的器件类型，使用最合理的实现方案，并使用自顶向下的设计方式进行前期的设计准备。

1. 设计输入

设计输入是开发过程中最主要的部分。设计输入不完整或者不正确则需要其他步骤来检测并完善，直至设计得到完整的功能需求。首先需要进行算法方面的设计，确定需要使用的算法和设计的逻辑结构。其次，RTL设计需要确定使用的硬件描述语言，如Verilog HDL和VHDL。在硬件编程设计的时候，需要区别于软件编程思想，时刻考虑到硬件的特性，注意硬件并行处理，相比软件存有较大差异。

1. 功能仿真

功能仿真是在硬件编程语言语法检查后，在编译前进行的逻辑功能验证，这对于初步的功能测试非常重要，可以快速了解设计输入的正确与否。如果发现错误则可以及时返回设计输入进行修正。功能仿真一般使用的仿真工具有：Synopsys VCS、Mentor ModelSim、Xilinx ISE等。

1. 综合优化

通过综合和优化这个过程，可以将设计输入的寄存器转换级电路（RTL，Register Transfer Level）代码，生成得到实际电路连接关系的网表类型文件。按照基本的设计原则，可以对设计的芯片编程进行优化，实现更高效率的电路实现。提高算法速度的思想主要有：面积与速度的平衡和互换，逻辑复制减少路径延迟，串并转换提高吞吐率和流水线结构等。

1. 布局布线

通过布局和布线，将网表文件按照约束，生成可以下载到板上进行测试的比特流数据文件。一般对于ASIC的布局和布线需要自行设计，给出布局布线的方案。但对于FPGA的开发，所使用开发平台的工具，一般都具有根据设计输入的代码进行布局布线的流程，所以在FPGA的开发中一般可以省去这部分的设计任务，直接使用软件得出比特流数据文件。

1. 时序仿真

时序仿真一般被称作后仿真，是在经过布局布线后，将器件和各模块的延迟数据经过分析，对电路行为做出预期估计。使用的仿真器可以和功能仿真相同，不同的是在仿真中加入了延迟考虑。

1. 板上验证

将已经设计好并且通过了仿真测试的编程数据文件，通过JTAG接口下载到FPGA板上，然后测试验证电路的实际运行情况。如果仿真与预期一致，但是板上验证有差别，则需要进行板上调试。一般可以使用逻辑分析仪或者FPGA芯片厂商提供的调试工具，如Xilinx ISE中的Chipscope、Altera QuartusII中的SignalTapII以及 SignalProb。例如使用Xilinx ISE的Chipscope来调试，需要编写cdc文件，添加需要观察的信号和触发条件的信号，并设置观察数据波形的周期深度，这和采样所选的时钟也有关系。然后将比特流文件下载到FPGA板上，通过设定触发器，在模块运行到所设置的触发条件后，可以获得所需数据和信号，以及触发后的一定周期内的数值变化情况。当然这种调试方式没有软件的灵活性大，但是基本可以满足板上调试的需求。

1.5 论文主要内容、创新点与结构安排

1.5.1 论文主要内容

本文主要对面向C-RAN的架构，搭建虚拟化平台，设计并实现了多个基于PCIe接口的硬件加速器，对LTE物理层的Turbo译码、FFT/IFFT等多个处理模块进行硬件加速。设计的硬件加速器均实现了流水线数据处理模式，并支持高时钟频率。

使用FPGA开发板和基带板作为硬件开发加速器，PCIe双工模式的DMA进行数据传输。使用Xen虚拟化技术实现硬件加速器的虚拟化，并建立虚拟机供LTE物理层软件执行其他流程。使用仿真工具和板级测试对使用的加速器进行功能验证、性能测试和结果分析。结果表明虚拟化的硬件加速器在C-RAN中的应用具有较明显的效果，可以提高云基站的处理性能和资源利用率。

1.5.2 论文创新点

本文的主要创新点可以总结如下：

1. 使用Xen虚拟化技术，搭建了高性能的虚拟化C-RAN平台，虚拟机的开销很少。在C-RAN平台中，通过使用硬件实现LTE物理层中高复杂度和高耗时的处理模块，并使用PCIe设备驱动调用硬件加速器，实现高吞吐量的数据传输，减少软件处理延迟，提高实时系统支持的带宽。
2. 设计的Turbo译码硬件加速器，使用了高频时钟，流水线模式，可以支持超过LTE物理层系统5MHz带宽的吞吐量。单块加速器可以支持多个译码用户同时使用，时间延迟较低，与单用户使用基本相同，平均时间为软件的1/3以下。
3. FFT和IFFT的硬件加速器在基带板上实现，不但可以去除软件这部分处理的时间，还能提高传输的效率，将经过硬件加速的数据直接送往射频收发板。当需要独立使用加速器时，还可以将FFT和IFFT的硬件模块移植在通用的FPGA开发板上。通过设计的硬件加速器接口，可以支持在一块FPGA上实现多个加速器，充分利用硬件资源，提高系统处理性能。

1.5.3 论文结构安排

本文共有六章，论文的结构安排如下：

第一章为绪论，简述了C-RAN的背景和发展情况，以及C-RAN技术相比传统架构的优势。并对虚拟化技术和硬件加速器的发展情况做了介绍。最后介绍了FPGA开发的流程。

第二章为C-RAN虚拟化平台和硬件加速器接口的设计。首先介绍了本文使用的虚拟化技术Xen，以及搭建的虚拟化平台。接着分析了硬件加速器的PCIe前后端驱动的设计。最后介绍了PCIe硬件加速器接口的设计。

第三章为Turbo译码硬件加速器的设计与实现。分析了Turbo码以及译码算法。并对加速器使用的Turbo译码硬核进行了介绍。接着对加速器的硬件设计进行分析，基于FPGA和PCIe驱动设计了流水线模式和调度方式。并使用ModelSim进行功能仿真测试，验证译码功能。

第四章为FFT和IFFT的硬件加速器的设计与实现。首先介绍了基站和终端的相关处理流程和FFT硬核模块的功能特点。接着对IFFT和FFT的硬件加速器设计分别进行了分析。最后进行功能仿真，分析FFT和IFFT硬件加速器的集成结果。

第五章为加速器的验证与测试。首先介绍了测试的平台和测试指标，并设计了测试的方案。使用Xilinx ISE进行综合并在FPGA板和基带板上连同C-RAN中的LTE平台，分别对Turbo译码加速器和FFT/IFFT加速器进行了测试，并与软件处理方式进行了分析与对比。并通过Xen的虚拟化平台测试了虚拟机中的使用性能。

第六章对本文进行总结，同时对C-RAN中加速器的未来进行展望。

第2章 C-RAN平台与硬件加速器接口设计

2.1 LTE物理层概述

LTE（Long Term Evolution），即长期演进，是由3GPP（The 3rd Generation Partnership Project，第三代合作伙伴计划）组织制定的通用移动通信系统的长期演进。LTE于2004年立项并启动，随后在全球范围内进行试验和部署，并得到了大规模的商用。LTE作为3G到4G的一个过渡，改进并增强了3G的空中接口技术。它采用了OFDM和MIMO等关键技术，显著增加了频谱效率和数据传输速率。

2.1.1 LTE物理层简介

LTE横向主要分为三层：物理层，数据链路层和网络高层；纵向包括两个平面：用户平面和控制平面。物理层主要给高层提供数据传输服务并负责传输信道的错误检测，完成基带信号的处理。数据链路层在用户和控制两个平面中都存在，在控制方面完成信令的传输和加密等工作；在用户方面完成用户数据的加密和传输等工作。网络层在控制平面中，主要用来完成用户终端和无线接入网间的信令的处理工作。物理层的主要特征有：多址技术、双工方式、多天线技术和业务资源共享等。

随着通信的发展，出现了多种差错控制的编码方式。香农定理中给出了信道信息传送速率的上限和信噪比以及带宽之间的关系。之后出现了一系列的编码方式，例如海明码为代表的线性分组码，卷积码和级联码等。而在后来的国际通信会议中提出的Turbo码，是一种全新的编码方案，在编码中使用了交织器和解交织器，充分利用了随机编码的条件，从而获得了超高的解码的性能，几乎可以接近香农定理极限。因此，Turbo码成为信道编码领域中的热点。目前在LTE中，信道编码主要采用的是Turbo码。

2.1.2 LTE物理层处理流程

LTE物理层中的网络链路主要可以分为上行和下行两部分。一般上行的链路中包括请求信息，下行的链路中主要为丰富的数据信息。下行物理信道主要有：物理层下行共享、广播、多播、控制格式指示、混合重传指示和控制等信道。上行物理信道主要有：物理层上行的共享、控制和随机接入等信道。

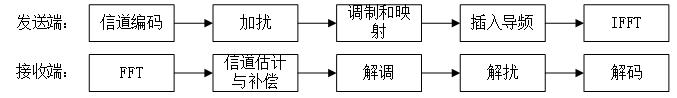


图2.1 LTE物理层一般处理流程

LTE的物理层处理流程可以如图2.1所示，在发送端，首先需要将待发射的数据进行信道编码，编码可以提高数据的可靠性，减少信道的干扰。编码如上节所说，在LTE中使用的是Turbo编码，具有很好的性能。之后对数据进行加密，加密使用加扰的方式进行。然后对星座进行调制和对资源块进行映射，通过这个步骤可以将发送的数据调制到所需的频率上。一般的星座调试方式包含BPSK、QPSK和QAM等。在物理广播信道中需要采用BPSK，用来保证数据传输时的准确性。之后需要插入导频，方便信道的估计。最后使用IFFT和插入循环前缀来将数据从频域转为时域。在接收端和发送端的过程基本相反，先要进行将数据从时域转换为频域的FFT过程，并去除循环前缀。之后再做信道估计与补偿，接着解调和解扰，最后是解码，完成数据的接收。

在LTE物理层处理的各个流程中，Turbo译码、FFT和IFFT等处理的时间较多。在Turbo译码阶段，由于需要多次迭代的过程以及采用的算法较为复杂等因素，使得译码复杂程度比一般的卷积码高得多，所以在软件上的处理速度一般较慢。通常可以使用较简化的算法以节省译码时间，但软件所需时间在整体LTE信号处理中还是占了大部分。在高带宽的LTE系统中，每帧数据的处理时间有限，所以需要硬件加速器对Turbo译码和FFT等模块进行加速处理，以提高LTE系统的整体性能和带宽。

2.3 Xen虚拟化平台架构

2.3.1 Xen虚拟化技术

虚拟化技术通过对资源进行抽象，将硬件或者系统的实际资源进行抽象，获得逻辑上可使用的抽象资源，这种抽象出来的可使用的资源一般被称为虚拟机（VM，Virtual Machine）。在实际的硬件资源中使用的系统一般被称为主机操作系统，在系统中管理虚拟机的程序，一般称为虚拟机监控器（VMM，Virtual Machine Monitor），利用VMM可以在主机操作系统中建立虚拟机。虚拟机中可以运行可修改或未修改的操作系统，这个系统一般被称为客户操作系统。虚拟化技术可以充分利用硬件资源，减少硬件设备的开销，提高资源的利用率。

Xen是开源的虚拟化项目，是较早出现的一种虚拟化技术。Xen就是一种虚拟机监控器，利用Xen对硬件资源的管理，可以在上层使用可修改的和未修改的两种操作系统。在Xen的早期阶段，它采用半虚拟化技术，需要在虚拟机中使用经过修改的操作系统。这种半虚拟化技术可以获得比全虚拟化更高的性能。一般情况下，Xen本身的开销不会超过10%。在后期，与之后流行的KVM等虚拟化技术一样，都支持全虚拟化技术。随着技术的成熟，这种技术的性能开销在降低，但是一般还是高于半虚拟化技术的开销。本文采用Xen来搭建C-RAN的虚拟化平台，可以实现LTE信号处理模块的硬件加速器的虚拟化，提高硬件资源的利用效率。

2.3.2 分离设备驱动模型

在Xen的虚拟化架构中，Xen设计的VMM在操作系统的底层，拥有比操作系统更高的权限，通过VMM来支持CPU内存等常用硬件资源的虚拟化调度。对于用户需要使用的I/O等设备，需要自行设计对应的虚拟设备调度方式。

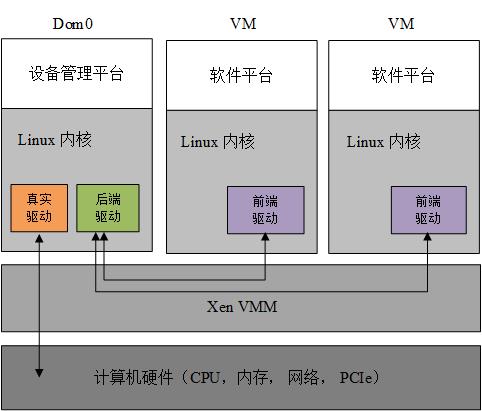


图2.2 Xen的设备驱动模型

在虚拟机中使用设备，需要虚拟的设备驱动。在Xen中，实现了虚拟机中驱动和实际驱动通信的机制。为了方便开发和调试，一般还可以将管理虚拟机中的驱动分离出来。虚拟机中的驱动一般称为前端驱动，管理前端驱动的虚拟机或主机中的驱动被称为后端驱动。驱动模型的结构可以如图2.2所示。在虚拟机VM中，前端驱动负责截获用户调用设备驱动的请求，并通过Xen的通信机制，实现向后端驱动转发。后端驱动可以和实际的设备驱动一同位于主机操作系统中（Domain0，Dom0）。后端驱动通过通信获得所有虚拟机中前端驱动的请求，并自行设计调度处理，排队使用实际的设备，最后将结果依次返回，完成真实设备的使用。

在前后端的通信机制中，Xen设计出了三种通信方式。其中，前端的请求通过I/O共享环来实现，当前端产生请求时，通过使用I/O共享环这块固定的内存，将请求放入其中，等待后端从中取出。在数据传输方面，这种分离的驱动模型为了提高传输效率，采用授权表方式。即前后端传输的数据使用页面映射的方式，将页面的授权索引通过共享环传输到另一端共享，省去内存拷贝的开销。事件通道用于各Dom间以及与Xen实现异步通知。授权表可以将内存页从一个Dom映射到另一个Dom内使用，通过授权表的页映射共享内存机制，进行无拷贝数据的高效传输。

2.3.3 PCIe前后端虚拟驱动

在C-RAN的虚拟化平台中，硬件加速器使用的是基于PCIe总线进行数据传输的FPGA开发板。由于在开发时需要频繁对硬件加速器进行改进和调试，使用可编程的FPGA可加速设计的开发。在加速器设计比较完善的后期，可以使用专用集成电路（ASIC）定制功能更完整和更低功耗的硬件加速器。

真实的PCIe设备驱动为使用DMA方式传输的，连接FPGA与服务器的PCIe数据采集卡驱动。该数据采集卡采用PCIe 2.0版本的X4接口与服务器做数据交互。实验中的服务器操作系统为Linux, 服务器内安装PCIe接口的转接卡，连接到FPGA开发板上。数据采集卡的PCIe数据传输功能由软硬件配合完成，软件工作在服务器端，硬件功能由采集卡上的Xilinx Virtex-6系列FPGA器件完成。

在物理主机中使用时，将PCIe的驱动模块加载到Linux内核中，并在FPGA中下载包含PCIe接口的硬件设计模块的比特流文件，之后在Linux系统中就可以调用驱动提供的读写等函数，和FPGA进行数据的交互。

根据Xen所提供的驱动模型，在虚拟化的平台中，需要设计出PCIe驱动的前后端虚拟驱动。前后端驱动首先根据Xen的通信机制来设计实现。对于前端驱动实现的内容，主要是将LTE物理层的调用加速器的请求，转发到后端驱动处理。后端驱动主要实现获取前端驱动的请求和对请求进行调度处理。每个虚拟机中都有一个前端驱动，在主机操作系统Dom0中只有一个后端驱动，后端驱动处理所有前端驱动的请求，因此后端会比前端较复杂。PCIe设备驱动实现的结构与图2.2的Xen设备驱动模型相同。

2.4 加速器硬件接口设计

PCIe设备驱动的接口为PCIe数据采集卡的硬件接口，由后端虚拟驱动调用真实驱动来控制。硬件接口包括DMA方式数据传输的TX、RX的FIFO接口和配置参数的寄存器接口。接口传输都对应有使能信号控制，时钟频率为62.5MHz。

FPGA器件中的DMA模块，负责在PCIe接口和采集卡上的用户逻辑之间做数据传递，同时也为用户逻辑提供PCIe BAR空间即参数寄存器的接口。

在使用中，数据的输入使用RX FIFO，接口的描述如表2.1所示。数据的输出使用TX FIFO，接口的描述如表2.2所示。

表2.1 DMA的TX FIFO接口

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 宽度(bits) | 描述 |
| tx\_dat\_fifo\_rclk | DMA->加速器 | 1 | FIFO的读时钟，62.5MHz |
| tx\_dat\_fifo\_rden | DMA->加速器 | 1 | FIFO读使能, 高有效 |
| tx\_dat\_fifo\_dout | 加速器->DMA | 64 | FIFO输出数据 |
| tx\_dat\_fifo\_empty | 加速器->DMA | 1 | FIFO 空标志，高有效 |

表2.2 DMA的RX FIFO接口

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 宽度(bits) | 描述 |
| rx\_dat\_fifo\_wclk | DMA->加速器 | 1 | FIFO的写时钟，62.5MHz |
| rx\_dat\_fifo\_wren | DMA->加速器 | 1 | FIFO写使能，高有效 |
| rx\_dat\_fifo\_din | DMA->加速器 | 64 | FIFO输入数据 |
| rx\_dat\_fifo\_full | 加速器->DMA | 1 | FIFO 满标志，高有效 |

硬件加速器在使用中还需要参数的传递，例如Turbo译码加速器中的码长和FFT加速器中软件同步的偏移量等。所以设计了传递参数的接口，具体如表2.3所示。此接口用来配置硬件中存储传参数的寄存器，不适合用作大量数据的传输。

当读参数时，gr1\_rd与gr1\_addr同时有效，加速器的逻辑最快在下一拍准备好读参数的寄存器值，准备好寄存器值的同时，将gr1\_rdy信号变高，gr1\_rdy信号最快只能在gr1\_rd有效的下一拍变高。

当写参数时，gr1\_wr ，gr1\_addr，gr1\_wdata同时有效。加速器逻辑保存数据后将gr1\_rdy信号变高。

表2.3 加速器参数接口

| 信号名称 | 方向 | 宽度(bits) | 描述 |
| --- | --- | --- | --- |
| clk\_dp | DMA->加速器 | 1 | 参数接口的时钟，62.5MHz |
| gr1\_rd | DMA->加速器 | 1 | 参数寄存器读使能 |
| gr1\_wr | DMA->加速器 | 1 | 参数寄存器写使能 |
| gr1\_rdy | 加速器->DMA | 1 | 参数寄存器ready信号 |
| gr1\_addr | DMA->加速器 | 6 | 读写操作的地址 |
| gr1\_wdata | DMA->加速器 | 32 | 写操作数据，与gr1\_wr同时有效 |
| gr1\_rdata | 加速器->DMA | 32 | 读操作数据，在读操作时与gr1\_rdy同时有效 |

2.5 本章总结

本章首先介绍了LTE物理层和物理层中处理流程的情况，并得出在物理层处理流程中，像Turbo译码和FFT等处理模块，运算的复杂度限制了系统的带宽，需要使用硬件进行加速处理。对基于Xen搭建的C-RAN虚拟化平台架构进行了介绍，分析了PCIe驱动的前后端设计和硬件接口的设计。最后分析了FPGA的开发流程，为后面的基于FPGA的硬件加速器的设计提供了设计流程。

第3章 Turbo译码硬件加速器设计

3.1 硬件结构设计

Turbo译码加速器的硬件设计部分主要包括：硬件整体结构层次和接口、译码层的逻辑处理、流水线模式设计和译码内部硬核调度策略以及软件调度处理方式。本节主要介绍硬件结构的设计，其他内容在后面分别详细设计。

Turbo译码硬件加速器的结构如图3.1所示。软件平台通过PCIe接口和硬件做数据交互。硬件端可以分为三层。顶层包含硬件的PCIe接口部分和DMA控制器。中间层为控制层，包含用户逻辑所使用高频时钟的生成模块和底层使用参数的控制逻辑。数据传输的时钟为62.5MHz，为了提高运算速度，使用时钟生成模块输出250MHz高频时钟。底层为译码层，包含具体Turbo译码的IP核模块、隔离时钟域并缓存数据的RX、TX FIFO和加速器的处理逻辑。



图3.1 Turbo译码器硬件结构

硬件加速器的顶层模块主要涉及PCIe接口和DMA控制器部分，主要的接口为设计的数据采集卡中使用的接口。控制层模块的接口信号如表3.1所示，信号主要包括时钟、参数接口和数据接口等。接口在第2章中进行了说明，数据传输信号主要包括RX和TX两个FIFO的接口信号，参数的接口支持使用参数BAR FIFO或者直接使用寄存器存储。

表3.1 控制层模块接口信号表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | I/O | 位宽 | 信号描述 |
| clk\_dp | Input | 1 | DMA接口时钟 |
| areset | Input | 1 | 模块初始化信号 |
| gr1\_rd | Input | 1 | 参数读使能 |
| gr1\_wr | Input | 1 | 参数写使能 |
| rx\_conf\_fifo\_din | Input | 13 | 参数输入数据 |
| tx\_conf\_fifo\_dout | Output | 13 | 参数输出数据 |
| rx\_dat\_fifo\_wclk | Input | 1 | RX FIFO写时钟 |
| rx\_dat\_fifo\_wren | Input | 1 | RX FIFO写使能 |
| rx\_dat\_fifo\_din | Input | 64 | RX FIFO输入数据 |
| rx\_dat\_fifo\_full | Output | 1 | RX FIFO满信号 |
| tx\_dat\_fifo\_rclk | Input | 1 | TX FIFO读时钟 |
| tx\_dat\_fifo\_rden | Input | 1 | TX FIFO读使能 |
| tx\_dat\_fifo\_dout | Output | 64 | TX FIFO输出数据 |
| tx\_dat\_fifo\_empty | Output | 1 | TX FIFO空信号 |

译码层模块的接口信号如表3.2所示。由于译码层和控制层的输入输出接口的功能都是进行数据和参数的交互，所以接口信号相似。控制层中的参数和数据通过名称相同的接口信号和译码层模块进行数据交互。接口主要的不同，在于控制层中利用输入的DMA时钟，生成了250MHz的高频时钟输入至译码层模块中。

表3.2 译码层模块接口信号表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | I/O | 位宽 | 信号描述 |
| clk\_dp | Input | 1 | DMA接口时钟 |
| clk\_250 | Input | 1 | 译码器核使用时钟 |
| areset | Input | 1 | 模块初始化信号 |
| gr1\_rd | Input | 1 | 参数读使能 |
| gr1\_wr | Input | 1 | 参数写使能 |
| rx\_conf\_fifo\_din | Input | 13 | 参数输入数据 |
| tx\_conf\_fifo\_dout | Output | 13 | 参数输出数据 |
| rx\_dat\_fifo\_wclk | Input | 1 | RX FIFO写时钟 |
| rx\_dat\_fifo\_wren | Input | 1 | RX FIFO写使能 |
| rx\_dat\_fifo\_din | Input | 64 | RX FIFO输入数据 |
| rx\_dat\_fifo\_full | Output | 1 | RX FIFO满信号 |
| tx\_dat\_fifo\_rclk | Input | 1 | TX FIFO读时钟 |
| tx\_dat\_fifo\_rden | Input | 1 | TX FIFO读使能 |
| tx\_dat\_fifo\_dout | Output | 64 | TX FIFO输出数据 |
| tx\_dat\_fifo\_empty | Output | 1 | TX FIFO空信号 |

3.2 Turbo译码算法

Turbo码是20世纪90年代出现的一种级联码，因译码过程类似涡轮工作被称为Turbo码。Turbo码拥有超高的解码性能，几乎可以接近香农定理极限，纠错能力也很突出，所以在通信编码领域应用非常广泛。

Turbo译码的算法可以分为两类，一种是基于后验概率，使用先验分布消除正反馈软输出的MAP算法及其简化算法：Log-MAP、Max Log-MAP。另一种是基于维特比（Viterbi）译码算法的软输出维特比译码（SOVA）。SOVA类算法虽然在译码时复杂度较低，但是相对性能不高，且不太稳定。MAP算法性能比较好，但是乘法运算量高，使译码速度受到影响，在实现和迭代运算处理中难度比较大。因此，基于MAP又产生了许多简化之后的算法，主要有Max Log-MAP和Log-MAP等，相较于MAP更易于运算和实现，很大程度上降低了算法的复杂度。

Log-MAP就是将MAP中的似然都采用对数似然来表示，使乘法变成加法运算，以提高译码速度。Max Log-MAP通过将似然运算中的加法转变为求解最大值的方式，从而减少了大部分加法运算，提高了算法效率，使算法稳定性能更好。

Turbo译码加速器中使用的是Xilinx 公司下3GPP LTE Turbo Decoder v2.0的IP核。该IP核使用的算法有Max Log-Map和Log-Map。相比Max Log-Map算法，Log-Map实现方式的误比特率性能会有0.3dB的提升，但是所需的硬件处理时间也会有增加。在信道条件一定的情况下，可以根据要求选择不同的算法进行硬件加速器的设计，以获取更高的运行效率。

3.3 Turbo译码硬核设计

本文采用的Turbo译码器的IP核为Xilinx 公司下3GPP LTE Turbo Decoder v2.0版本。这个译码器的IP核与编码器相对应，并且满足3GPP移动通信系统的规范。该IP核的主要特性如下：

1. 支持所有3GPP LTE中188个不同码长，长度从40至6144bits
2. 动态可配置的迭代次数，1-15次
3. 支持多个处理单元同时工作，数量为1、2、4或8个处理单元
4. 支持多种译码算法，包括Max Log-Map和Log-Map方式，支持1/3码率的输入
5. 支持并行的输入和输出

为了满足LTE规范所需的高吞吐量和时间延迟的要求，Turbo 译码器中具有多个处理单元（PUs，Processing Units）。 这些PU又可以成为解码单元（DUs，Decoder Units）。可以创建具有1、2、4和8个DU的译码器。在设计中增加额外的处理单元需要额外的FPGA资源，但相对的处理延迟会降低，且吞吐量的性能会提高。当采用多个处理单元的设计时，译码器的调度算法会尝试拆分可用的解码单元。较大的码块会被分成子块并且并行地被不同的解码单元执行运算，确保降低延迟，较小的块可以分配给单个解码单元进行处理。

3.4 处理逻辑设计

Turbo译码硬件加速器在Xen虚拟化平台中使用PCIe接入后，一次完整的用户请求使用此硬件加速器的主要流程如下：

1. 应用中，用户请求使用Turbo译码硬件加速器
2. PCIe前端虚拟驱动接收用户请求并转发给后端虚拟驱动
3. 后端驱动将请求加入到请求队列中，并从先入先出的队列中取出头请求，然后调用真实的数据采集卡驱动
4. 数据和译码的码长参数通过PCIe传输到FPGA硬件端
5. 硬件端将数据和参数通过控制层传入译码层中，使用FIFO缓存
6. 译码层使用高频时钟进行数据读出和送入译码器IP核中运算
7. 运算结果经高频时钟送出到FIFO中缓存
8. 后端驱动通过调用读函数，读出PCIe传输出来的缓存在FIFO中的有效数据
9. 后端驱动将数据转发给前端驱动
10. 前端驱动获得译码后的有效数据，返回给用户，进行后续操作，译码请求处理结束

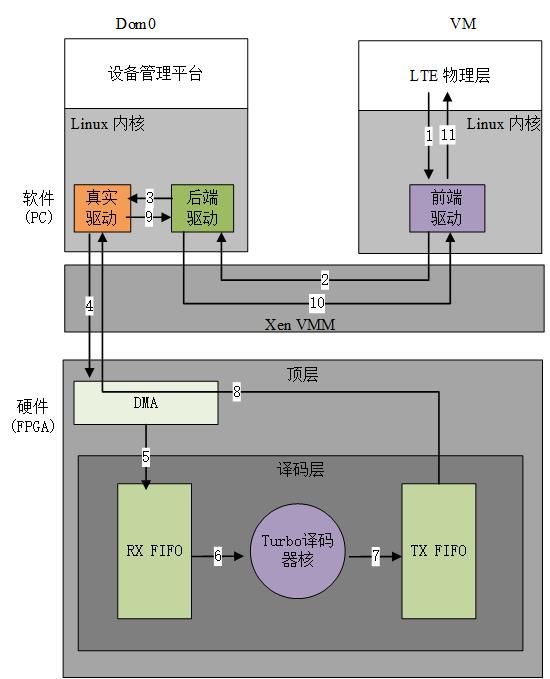


图3.2 Turbo译码器使用流程

图3.2为使用加速器的流程图，软件平台完成流程中的1-4步和8-10步，硬件的主要步骤为5-7步。在不同的时钟频域之间进行数据传输，可以使用数据缓存器FIFO（First In First Out）来实现。数据在硬件中的传输处理使用FIFO缓存数据，读取数据不需要使用地址，而是根据写入的顺序依次读出。在加速器的设计中，FIFO的一端为PCIe总线的接口。数据位宽为64bit，连接到DMA模块，DMA方式进行数据传输的TX、RX的FIFO接口时钟频率为62.5MHz。而在FIFO的另一端，连接Turbo译码器核和处理逻辑的时钟频率是250MHz的高频时钟。这个高频时钟clk\_250由控制层通过DMA的输入时钟clk\_dp生成，并传入decoder层。因为DMA的默认块大小为16KB，所以根据数据吞吐的需要，FIFO的深度一般可设为2048或更大。

参数的传输由于不同码长的数据块的存在，所以也使用FIFO进行缓存。在一个数据块进行数据传输时，对应的码长参数同时传到相应的参数FIFO中。在参数传递中，根据对应的参数FIFO的读使能和译码器IP核参数有效输入信号控制。

Turbo译码器的一次译码逻辑主要是如下过程：

1. 初始化。用户在需要时，调用PCIe驱动写寄存器，发送areset信号重置所有FIFO和译码器模块。重置的时间不是固定的时钟周期，但会在1000个时钟周期内完成，完成后信号sbin\_wready会变高。之后译码器处在可工作状态，可以接收数据和参数。
2. 参数输入。用户将数据包长大小写入寄存器，通过gr1\_wr写使能写入RX\_BAR\_FIFO。当此FIFO收到控制层传来的参数，参数FIFO的空信号和译码器IP核的准备信号sbin\_wready一起作为握手信号para\_in\_handshake，赋值给sbin\_wvalid，去读取参数FIFO中的一个码长参数。数据通过sbin\_wsize（来自RX\_BAR\_FIFO的dout）写入译码器IP核。
3. 数据输入。用户发送写数据请求，将数据写入驱动缓冲区，DMA将数据送至RX FIFO 。sdin\_wready指示准备好接收数据。当数据FIFO（RX FIFO）不为空，且译码器已准备好，产生data\_in\_handshake握手信号，送给sdin\_wvalid，数据通过sdin\_wsys0、sdin\_wp0、sdin\_wpint0接口，送入译码器IP核。数据可以两组并行输入，提高吞吐率。
4. 运算。当数据输入到要求的大小的长度后，sbin\_wflush信号有效，译码器此时开始运算。
5. 参数输出。运算完成后，mhd\_bout\_wsize输出数据大小，为13位，拼接成32位，送入TX\_BAR\_FIFO。
6. 数据输出。mhd\_dout\_wvalid指示数据输出，将有效数据写进TX FIFO，输出的数据位宽为32位，FIFO的输出为64位，与PCIe的64 bits位宽相匹配。由于DMA的块传输方式，不足的数据无法直接传输出来，所以增加逻辑将不足一块大小的数据补足至一整块大小。例如块大小为16KB，需要之后补足到16KB，后面可以使用最后一行的数据，增加FIFO不为空的时间，以降低逻辑的复杂程度。
7. 用户读。用户读出块大小和数据。DMA将数据搬送到服务器端PCIe驱动的缓冲区，默认大小为128MB。之后用户调用的读函数会从缓冲区中读出所需的译码后的数据。

3.5 流水线设计

Turbo硬件加速器主要处理流程可以分为三部分：编码后的数据输入、硬件译码运算、运算后的数据输出。为了提供对多用户的各种码长的数据包支持，设置了可配置的码长参数与数据包同步传入硬件译码逻辑中。

数据传输使用的PCIe接口支持双工模式，可以实现同时读写。Turbo译码的IP核模块也支持连续译码。因此，为了提高加速器的吞吐率，Turbo译码加速器数据的输入输出和运算可以分成三级流水并行执行，设计成流水线模式，如图所示。在第一个数据包译码结果输出的时候，同时可以进行第二个数据包的运算和第三个数据包的输入。



图3.6 流水线模式

全部编码的数据包长度从176B到24KB，数据包输入的平均时间为24us，译码数据输出时间为10us。

3.6 调度处理

硬件加速器的调度可以分为软件端的调度和硬件端的调度。软件端为用户的译码请求调度，硬件端为Turbo译码器的IP核的解码单元调度。为了实现加速器吞吐量的最大化，必须确保IP核内所有有效的解码单元都处于工作状态。所以硬件的IP核中调度程序算法尝试最小化译码的等待时间，同时最大化吞吐量。待译码块所需的确切译码时间，取决于当前在解码器内的块数量和大小。为了最大效率地使用这些解码单元，一般对需要进行译码的待译码块的大小进行降序排列，如果迭代次数不同，则迭代次数也需要先降序排列，保证迭代次数相同的待译码单元一块运算处理，这样可以使解码单元得到充分利用。同时译码器还需确保译码结果输出的顺序与待译码块输入的顺序一致。

Turbo译码硬件加速器的软件端调度处理，主要在PCIe的后端虚拟驱动中。普通情况下是根据收到的前端请求的先后顺序排列进请求队列中，之后按照此顺序依次调用硬件加速器。

3.7 功能仿真

使用Xilinx ISE进行综合后，得到Turbo译码加速器的资源使用情况。具体的资源消耗如表3.2所示。

表3.2 Turbo译码加速器资源消耗

|  |  |  |  |
| --- | --- | --- | --- |
| 资源名称 | 使用量 | 可用量 | 使用率 |
| Slice Register | 44141 | 301440 | 14% |
| Slice LUT | 28363 | 150720 | 18% |
| 36 Kb Block RAM | 76 | 416 | 18% |
| 18 Kb Block RAM | 56 | 832 | 6% |

利用功能仿真可以验证硬件加速器逻辑的正确性，本文通过使用Mentor ModelSim进行仿真测试。仿真测试的输入数据为LTE物理层软件运行时的待译码数据。译码层模块功能输入数据仿真图如图3.3所示，在信号sdin\_wready高有效时，硬核中即开始接收输入的数据。输出数据如图3.4所示，在运算完成后，通过mhd\_dout\_wdata信号可以输出译码后的数据，并送到TX FIFO中。

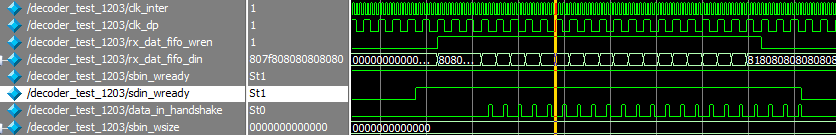


图3.3输入数据仿真图

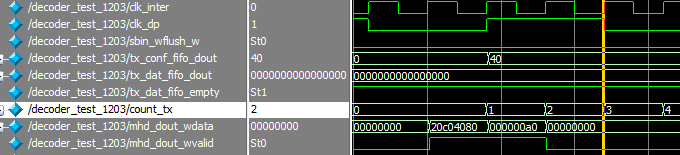


图3.4输出数据仿真图

根据仿真的输出，可以验证加速器的译码层模块功能正确。上层的PCIe和DMA的接口部分经过板级测试，功能都正确。

表3.3 Turbo译码仿真时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 码长（bits） | 输入时间（us） | 译码时间（us） | 输出时间（us） | 单块译码总时间（us） |
| 40 | 0.5 | 25.1 | 10 | 35.6 |
| 512 | 4.2 | 30.1 | 10 | 44.3 |
| 1024 | 8.3 | 31.8 | 10 | 50.1 |
| 2048 | 16.5 | 32.5 | 10 | 59 |
| 3008 | 24.2 | 38.4 | 10 | 72.6 |
| 4032 | 32.4 | 45.2 | 10 | 87.6 |
| 5184 | 41.6 | 52.1 | 10 | 103.7 |
| 6016 | 48.3 | 59.5 | 10 | 117.8 |

表3.3为硬件译码模块的仿真时间。从表中可以看出，Turbo译码硬件加速器模块的运算时间范围为25至60 us，这说明FPGA的开发板上的译码器性能非常好。使用软件译码，部分测试的译码时间可以在表3.4中看出。较长码块的时间会达到1000 us以上。使用硬件译码的时间，加上数据输入和输出时间，一次译码操作的总时间基本可以在100 us以下。对比普通软件的译码时间，效率是软件的3倍以上，是较长码块的十倍以上。在使用流水线模式下，硬件加速器的性能会比单块的仿真测试更好。

表3.4 软件Turbo译码时间

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 码长（bits） | 40 | 512 | 1024 | 2048 | 3008 | 4032 | 5184 | 6016 |
| 译码时间（us） | 114.9 | 299.3 | 481.2 | 686.5 | 1007.7 | 1168.8 | 1261.1 | 1340.9 |

因此，本文设计并实现的Turbo译码器硬件加速器功能满足要求，可以进行板级测试，并分析在LTE物理层的系统中使用情况。

3.8 本章总结

本章设计了Turbo码的硬件加速器。介绍了译码器使用的硬核的特性和算法，然后在此基础上设计了加速器处理逻辑。通过将整个译码过程分为输入、运算和输出三部分，设计了流水线模式提高加速器的吞吐率。接着介绍了使用硬件加速器的软件调度方式和在硬件中设计的解码单元调度算法。最后通过ModelSim进行功能仿真，与软件的译码进行对比，验证了Turbo译码硬件加速器逻辑功能的正确性，通过仿真数据分析了加速器的性能。

第4章 FFT/IFFT硬件加速器设计

4.1 基站FFT处理

在数字信号处理中，离散傅里叶变换（DFT，Discrete Fourier Transform）是常用的方法，被广泛应用于数字滤波、信号分析、视频编解码等诸多领域。DFT对周期性离散信号的变换，是分析离散信号的一种有力的数学工具。对于输入复数x0,…,xN-1，对应的离散傅里叶变换（DFT）为公式4.1所示。



（4.1）

按照这个定义，离散傅里叶变换的复杂度为O(N2)，因为复数xk总共有N个输出，每个输出还需要进行N项求和。但由于其运算复杂度较高，经常使用经过简化的快速变换的算法，即快速傅里叶变换（FFT, Fast Fourier Transform）。若使用FFT，变换的复杂度一般可以降至O(NlogN)，这样可以使运算时间减少一个数量级以上。使用FFT可以将信号在时域和频域上进行变换，可以容易的发现信号的特征，进行频谱等方面的分析。因此，FFT在信号处理等领域应用非常广泛，在LTE的基站处理中也是非常重要的一部分。

在基站处理中进行FFT运算，一般使用普通算法直接计算，或者使用FFTW（Fastest Fourier Transform in the West）。FFTW是用于计算任意大小的实数或复数的DFT和FFT的C子程序库。FFTW作为开源库，被称为是最快的FFT算法库。在已实现的LTE软件平台中，就使用FFTW来计算FFT。

对于LTE 5MHz带宽下的FFTW运算，虽然比普通的FFT算法已经缩短了近十倍。但在LTE物理层的整个处理流程中还是占了较多时间，缩短这部分的时间，可以提高整个系统的效率。在LTE中使用FPGA基带板进行数据收发的前后，加上硬件实现的FFT和IFFT加速器模块，可以尽可能缩短这部分时间。特别是在连续传输中，流水工作模式下，可以将数据传输的时间隐藏，在整个系统中占用的时间只包含硬件中FFT运算的时间。

4.2 FFT硬核设计

本文采用Xilinx 的LTE FFT v1.0的IP核。该IP核实现了所有3GPP LTE规范中要求的变换长度。包括支持15MHz带宽所需的1536点的FFT运算。该IP核的主要特性如下：

1. 支持所有3GPP LTE中要求的变换点大小：128、256、512、1024、1536和2048，并且支持前向和反向，可以动态配置。
2. 数据的样本精度为14-17 bits，一般选择16 bits，相位因子的位宽与其相同。
3. 缩放的规模动态可配置，根据需要在运行前输入相应的缩放参数。
4. 输出可选择自然顺序输出或者按位反转输出。
5. 在使用IFFT时，可以选择自动插入循环前缀（CP，Cyclic Prefix）。
6. 提供四种架构，以实现硬核的大小和变换时间之间选择的平衡。

FFT硬核的架构有四种，分别为： Streaming I/O、Radix-4 Burst I/O、Radix-2 Burst I/O和Radix-2 Lite Burst I/O。这四种架构所占用的资源和对应的吞吐量如图4.1所示。

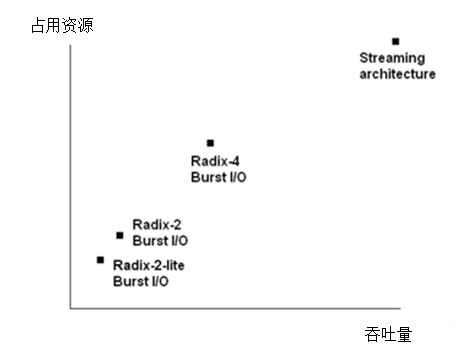


图4.1 FFT四种架构占用资源与吞吐量

FFT的IP核使用基二和基四分解法来计算离散傅里叶变换，对于Burst I/O 结构采用时域抽取法实现，对于Streaming I/O结构使用频域抽取法。流水线（Streaming I/O）结构允许数据的连续处理，在进行当前帧的数据变换时，可以加载下一帧的数据，同时输出上一帧的数据。流水线结构中采用了多个基二的蝶形处理单元，通过增加存储单元来存储输入和中间处理所需的数据，提高吞吐量。Radix-2 Lite 结构是基二结构的变体，通过采用时分复用的方式节省资源的使用，但是需要较长的转换时间，吞吐量相较于基二也有一些下降。

FFT的IP核接口如图4.2所示。通过接口可以看出，输入的数据为需要变换的实数点、变换点的大小和CP等控制信号。输出的为变换后的实数点，以及输入输出点的索引和指示信号等。其中实数包含实部和虚部两部分，每个部分的位宽为14至17位，因为软件使用short为16位，所以硬件中一般也选择使用16位的位宽。

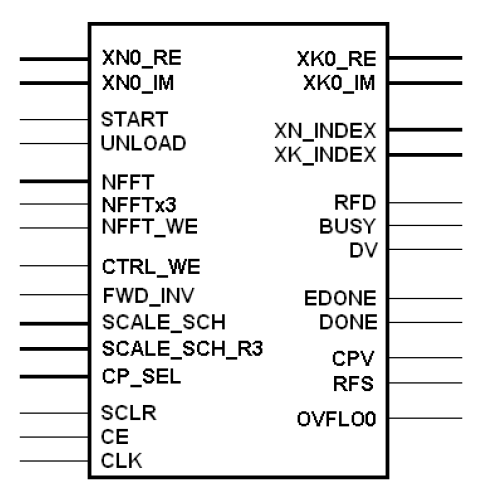


图4.2 FFT IP核接口图

FFT的IP核在运算时，需要对数据进行压缩处理，防止数据溢出。一般使用定点压缩的方式，这种方式可以在尽可能节省资源消耗的情况下，获得较好的数据精度。接口SCALE\_SCH可以配置每个阶段的压缩比例，例如512个点的FFT运算，压缩比例的参数共有10位，从最低位开始为第一阶的参数，在仿真中，通过调整参数，可以获得和软件相同的压缩比例。在测试中，通过引出ovflo引脚，观察仿真数据是否有溢出，若有溢出则需要提高压缩的比例后，再进行测试。

4.3 IFFT硬件加速器设计

4.3.1 硬件结构设计

IFFT是FFT的逆变换，在LTE基站的发送端使用，数据信号经过变换得到时域信号并加上循环前缀后，送至射频板中。在5MHz带宽的LTE系统中，每个时隙的FFT大小为512个点。在硬件中使用的实部和虚部的位宽为16 bits，因此每次IFFT的数据大小为512\*32 bits，即2KB。IFFT加速器将主机中IFFT模块后移至基带板中，主机与基带板的数据通道不变。因此，基带板中的顶层模块与Turbo译码硬件加速器类似。

IFFT的硬件整体结构如图4.3所示。硬件主要包含PCIe的顶层模块、射频部分数据收发与处理模块和IFFT的处理模块等。顶层的PCIe和DMA部分与Turbo译码硬件加速器部分基本相同，同样使用DMA的FIFO接口进行时钟域的隔离和数据的缓存。在Turbo译码硬件加速器的硬件结构设计中只包含Turbo译码和PCIe数据传输部分，并且数据的输入和输出对象相同，所以独立使用FPGA开发板，便于资源的管理和充分利用。而在IFFT加速器中，由于基站端接收到软件无线电平台的数据后，经过IFFT运算，通过AD9361发送模拟信号到终端。所以为了方便，直接将IFFT的硬件加速器放入与基处理的同一块硬件开发板中。

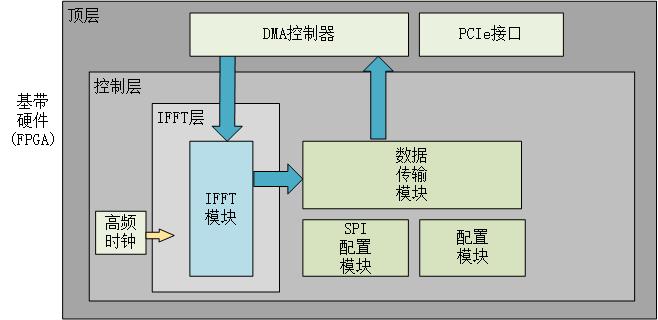


图4.3 IFFT基带板硬件结构图

IFFT模块接口的信号如表4.1所示。主要包括时钟和数据的接口信号。在仿真中，可以将内部信号引出观察，不需要放在模块的接口中，以简化接口。

表4.1 IFFT模块接口信号表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | I/O | 位宽 | 信号描述 |
| clk\_dp | Input | 1 | DMA接口时钟 |
| clk\_250 | Input | 1 | IFFT核使用时钟 |
| areset | Input | 1 | 模块初始化信号 |
| rx\_dat\_fifo\_wclk | Input | 1 | RX FIFO写时钟 |
| rx\_dat\_fifo\_wren | Input | 1 | RX FIFO写使能 |
| rx\_dat\_fifo\_din | Input | 64 | RX FIFO输入数据 |
| rx\_dat\_fifo\_full | Output | 1 | RX FIFO满信号 |
| tx\_dat\_fifo\_rclk | Input | 1 | TX FIFO读时钟 |
| tx\_dat\_fifo\_rden | Output | 1 | TX FIFO读使能 |
| tx\_dat\_fifo\_dout | Output | 64 | TX FIFO输出数据 |
| tx\_dat\_fifo\_empty | Output | 1 | TX FIFO空信号 |

4.3.2 IFFT运算逻辑

IFFT的IP核使用流水线模式接收FIFO中的数据，经过运算后，自动在每个块前加上对应的CP长度。CP的长度由参数CP\_SEL来控制。对于第一个符号，设置为二进制数01，其他符号设置为10。当其他符号输出六次后，一个时隙的符号输出结束，循环设置CP\_SEL。对于点数，通过参数NFFT来设置。若使用1536点数的IFFT时，需要额外设置参数NFFTx3，但是本文研究LTE平台未使用15MHz的带宽，所以不需要设置此参数。

IFFT的IP核运算需要连续输入一整块不间断数据，否则会导致输入时钟周期间断，输入数据错误。所以模块中设置了检测RX FIFO数据深度是否大于所需点数的逻辑，当FIFO中的数据足够时，才会触发运算的START信号，此时硬核会开始工作，进行IFFT运算。因此，如果想保持高吞吐以及流水线操作，需要数据持续不间断的传输到此模块中。当运算结束后，根据指示数据输出的DV信号，将数据写入TX FIFO中。读取FIFO中数据的模块为基带中的DATA模块，并最终发送出去。

IFFT加速器的输出需要添加循环前缀CP。根据不同的带宽和帧位置，添加的CP会有所不同，主要的循环前缀如表4.2 所示。当使用流水线模式时，由于需要在每帧前添加循环前缀，所以每个帧中间会增加循环前缀长度的时钟周期。这些输出的数据写入到TX FIFO中，不会影响数据接收端。

表4.2 不同带宽下的CP长度

|  |  |  |  |
| --- | --- | --- | --- |
| 带宽（MHz） | 点数 | 第一个CP长度 | 其他CP长度 |
| 1.25 | 128 | 10 | 9 |
| 2.5 | 256 | 20 | 18 |
| 5 | 512 | 40 | 36 |
| 10 | 1024 | 80 | 72 |
| 15 | 1536 | 120 | 108 |
| 20 | 2048 | 160 | 144 |

4.4 FFT硬件加速器设计

4.4.1 数据流切换

在LTE的基站FFT模块处理中，首先需要进行同步操作，获取帧的起始位置，之后对每帧中的每个块进行对应点数的FFT运算。在硬件实现中，同步过程依旧由软件来完成。同步完成后，需要软件将帧的相对偏移位置发送给硬件，硬件通过此偏移位置找出帧头并开始后面的频移和去CP后的FFT运算等操作。因此，在硬件中的数据流需要在同步前后进行切换。同步前，硬件发送到软件基站的数据为原始数据流，软件使用此数据进行同步操作。待软件找到同步点后，写寄存器通知硬件，待硬件在完成同步操作后，将经过FFT运算后的数据流写入待输出的TX FIFO中，并在数据流的开始处添加标志，以此通知基站完成数据流的切换。数据流切换过程如图4.5所示。

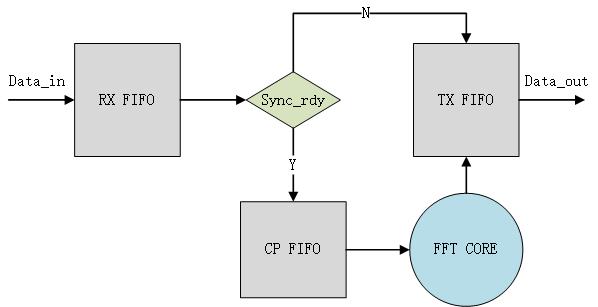


图4.5 FFT硬件数据流的切换

在设计中，根据基站同步信息进行硬件的帧同步是非常重要的一步。同步中，不仅需要获取软件端的同步偏移量，还需要计算相对软件的偏移量。因此，在硬件中使用了计数器，来计算硬件传到软件端的数据帧长。计数器使用对帧长取模的方式，统计出软件同步偏移量与硬件相差的点数。之后将相应的硬件偏移量移除，得到数据流的帧头位置。帧头找到后，即可开始后面的频移和FFT的运算逻辑部分。

硬件获取软件的同步偏移量并完成数据流切换的流程如下：

1. 硬件在未同步状态，加速器将RX FIFO中数据直接送至TX FIFO输出
2. 软件通过收到的数据，进行同步处理，得到帧起始位置相对收到数据帧的偏移量δ
3. 软件通过写寄存器，将偏移量通过硬件加速器参数接口写入FFT加速器模块中保存
4. 加速器收到控制写参数的使能信号gr1\_wr，通过逻辑将此参数写到寄存器中存储
5. 此时获取记录RX FIFO读出数据的个数的计数器值count\_rx，计数器对帧长76800取模循环累加。
6. RX FIFO需要继续读出76800-|count\_rx-δ|大小的点数据，通过控制读使能的信号来实现
7. 之后RX FIFO出来的数据为一帧数据的起始位置，将此时的数据读出后，不送往TX FIFO，改为进行频移处理，之后按照去CP的要求去掉每块的CP
8. 将处理完的数据送至CP FIFO中缓存，并设置加速器中已同步标志的信号sync\_rdy为真
9. FFT加速器通过同步信号sync\_rdy，将start信号变高，加速器开始工作，从CP FIFO中读取数据输入到FFT的核中
10. FFT运算后将数据输出到TX FIFO中，并在输出数据的开头加上标志，告诉软件数据流的切换。标志设计中，选择使用多行数据，将值设置为与软件端约定好的值
11. 软件端在做完同步后，进行循环读取一行数据，即一个点数的数据。若连续多行数据与硬件加速器约定的值相同，则开始将数据送至下一个软件处理流程中，最后进行解码获得最终数据

4.4.2 频移逻辑

在LTE上行中，由于基带数字的直流分量DC与模拟的DC错开半个子载波宽度（即7.5KHz），所以会有半个子载波的偏移。软件中使用查表获得每个点所需乘上的因子，并且使用指令集进行多点同时运算。在硬件中，使用了分布式内存来存储频移因子表，并使用了复数乘法器来进行频移因子的乘法运算。复数乘法器在使用时可以连续输出结果，不会成为传输带宽的瓶颈。将每帧的数据与对应点数的乘数因子输入复数乘法器，得到的结果作为下一步FFT运算的输入。

4.4.3 FFT运算逻辑

FFT模块在做完频移后，需要进行去CP操作。去除的CP长度与IFFT模块添加的相同。对于512个点的FFT，首个符号长度为40，其他的符号长度为36。经过频移后，通过控制CP FIFO的写使能信号cp\_fifo\_wr\_en，将去除CP之后的数据输入到CP FIFO中进行缓存。与IFFT相同，在FFT运算前需要保证FIFO中的数据大于运算的点数。当判断FIFO的深度大于点数后，再触发运算的START信号，此时硬核会开始工作，进行FFT运算。

FFT硬核使用的结构与IFFT的相同，也是流水线模式。FFT模块接口的信号如表4.3所示。信号主要包含时钟、数据接口信号和参数信号。

表4.3 FFT模块接口信号表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | I/O | 位宽 | 信号描述 |
| clk\_dp | Input | 1 | DMA接口时钟 |
| clk\_250 | Input | 1 | FFT核使用时钟 |
| areset | Input | 1 | 模块初始化信号 |
| fft\_conf\_din | Input | 17 | 模块同步输入参数 |
| gr1\_wr | Input | 1 | 参数输入控制信号 |
| fft\_conf\_dout | Output | 17 | 模块输出参数 |
| gr1\_rd | Input | 1 | 参数输出控制信号 |
| rx\_dat\_fifo\_wclk | Input | 1 | RX FIFO写时钟 |
| rx\_dat\_fifo\_wren | Input | 1 | RX FIFO写使能 |
| rx\_dat\_fifo\_din | Input | 64 | RX FIFO输入数据 |
| rx\_dat\_fifo\_full | Output | 1 | RX FIFO满信号 |
| tx\_dat\_fifo\_rclk | Input | 1 | TX FIFO读时钟 |
| tx\_dat\_fifo\_rden | Output | 1 | TX FIFO读使能 |
| tx\_dat\_fifo\_dout | Output | 64 | TX FIFO输出数据 |
| tx\_dat\_fifo\_empty | Output | 1 | TX FIFO空信号 |

4.5 功能仿真

在完成上述加速器逻辑功能的设计，并编写出相应的RTL代码后，需要通过功能仿真来验证逻辑的正确性，本文通过使用Mentor公司的ModelSim进行仿真测试。仿真测试中，将待做FFT/IFFT的数据写在文件中，编写相应的Testbench，读取上述文件中的数据，并给出对应的时钟和信号激励。将待运算的点数数据将送到设计的模块中运算，将得出的输出数据和软件运算的数据与Matlab的FFT/IFFT运算数据进行对比。

4.5.1 IFFT加速器功能仿真

首先通过使用Xilinx ISE进行综合，综合后的IFFT加速器资源消耗情况如表4.3所示。从表中可以看出，主要的存储资源使用是在FIFO的消耗上。为了提高加速器处理的带宽，将FIFO的深度设置的比较大，有利于加速器连续工作，对使用加速器的系统负载也有提高。对于FFT和IFFT的加速器，资源的使用不仅包括加速器，还包括了基带数据处理模块等。

表4.3 IFFT加速器资源消耗

|  |  |  |  |
| --- | --- | --- | --- |
| 资源名称 | 使用量 | 可用量 | 使用率 |
| Slice Register | 12784 | 301440 | 4% |
| Slice LUT | 9153 | 150720 | 6% |
| 36 Kb Block RAM | 168 | 416 | 40% |
| 18 Kb Block RAM | 13 | 832 | 1% |

IFFT的模块功能仿真的总体信号波形如图4.7所示，由于使用的是流水线模式结构，从图中有效数据输出的信号dv波形可以看出，每个FFT点块的输出间隔时间少于10us。模块功能仿真的部分输入数据如图4.8所示，从输入中可以看出，输入数据跟随的是rfd信号，rfd信号在start拉高后，信号开始拉高。输入的接口为xn0\_re和xn0\_im，都是由rx\_dat\_fifo\_dout输出赋值。部分输出数据如图4.9所示，输出的控制信号为dv，将输出的接口xk0\_re和xk0\_im赋值给tx\_dat\_fifo\_din，当需要读取数据时，只需要将TX FIFO的读使能拉高，若FIFO不为空则读出相应的数据。

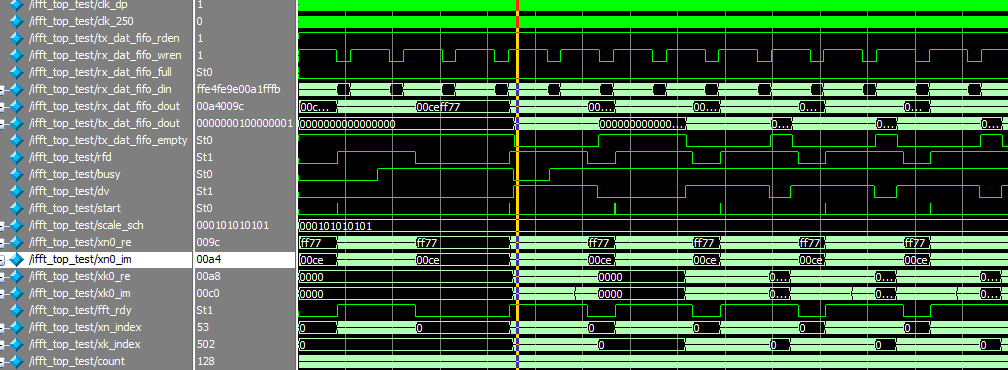


图4.7 IFFT模块功能仿真图

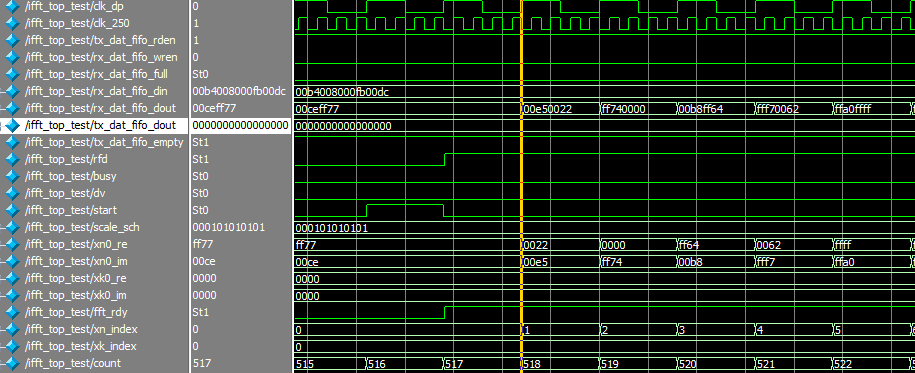


图4.8 IFFT模块输入数据仿真图

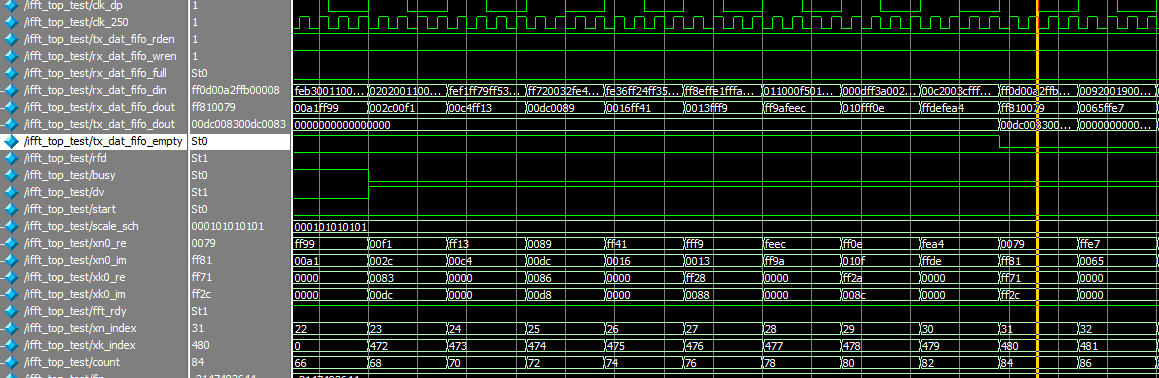


图4.9 IFFT模块输出数据仿真图

4.5.2 FFT加速器功能仿真

与IFFT加速器相同，首先进行综合，综合后的FFT资源消耗情况如表4.4所示。使用的资源情况基本与IFFT加速器相同。

表4.4 FFT加速器资源消耗

|  |  |  |  |
| --- | --- | --- | --- |
| 资源名称 | 使用量 | 可用量 | 使用率 |
| Slice Register | 13572 | 301440 | 4% |
| Slice LUT | 10948 | 150720 | 7% |
| 36 Kb Block RAM | 339 | 416 | 81% |
| 18 Kb Block RAM | 14 | 832 | 1% |

FFT的功能仿真，不仅要给输入数据激励信号，还需要给出同步的参数信号。图4.10为FFT模块的功能仿真波形图，在数据通过rx\_dat\_fifo\_din输入后，模拟软件端给出同步信号，在某个时间给出gr1\_wr写寄存器fft\_conf\_din的值。将软件同步的偏移位置发送到模块中。之后可以看到模块内的start信号。模块功能仿真的部分输入数据如图4.11所示，之后通过busy拉高运算，接着dv信号显示输出有效数据，部分输出数据如图4.12所示。在输入数据之前，经过了CP FIFO，输入的为处理了频移和去CP等操作后的数据。

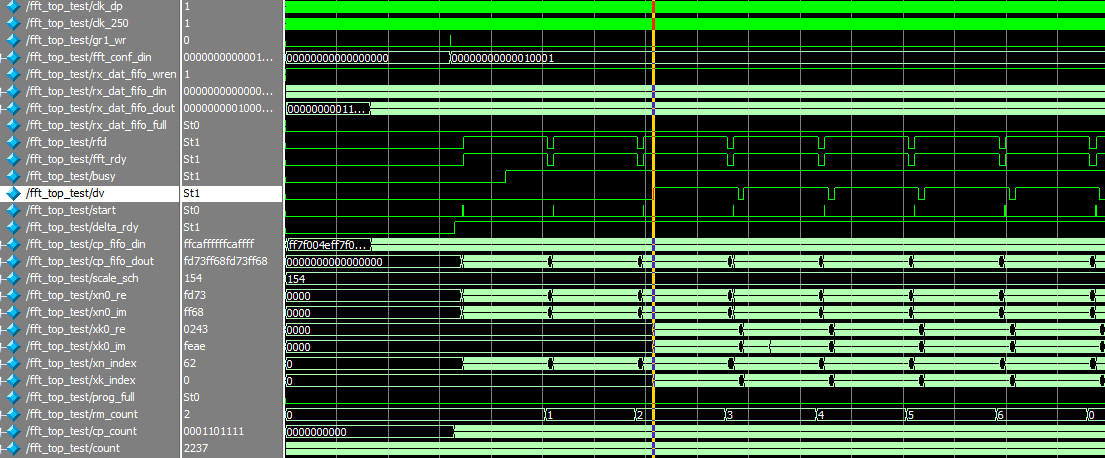


图4.10 FFT模块功能仿真图

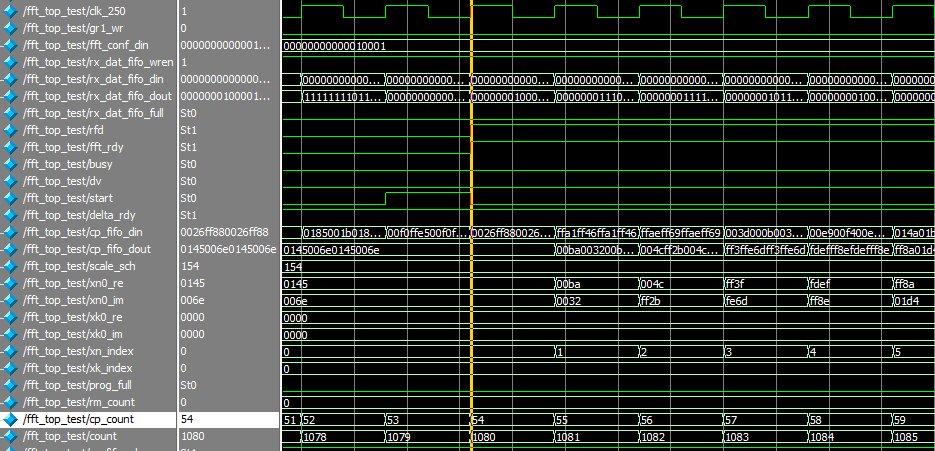


图4.11 FFT模块输入数据仿真图

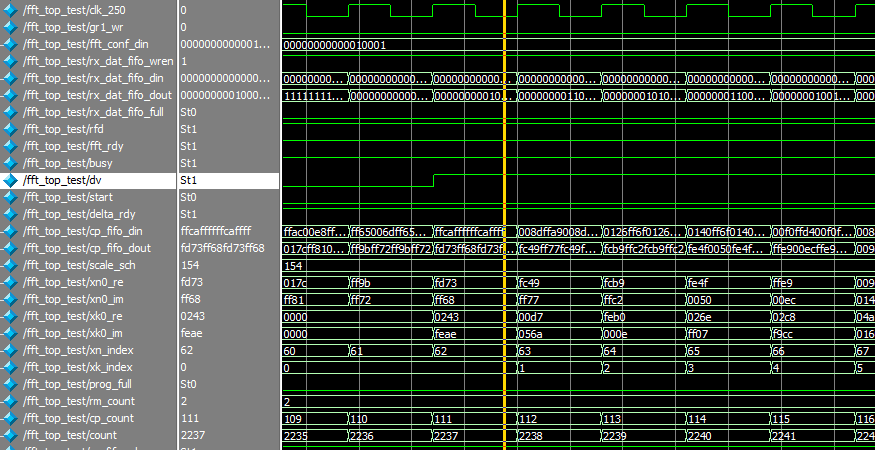


图4.12 FFT模块输出数据仿真图

根据仿真的输出，可以基本验证IFFT和FFT加速器的模块功能正确。因此，本文设计并实现的IFFT和FFT硬件加速器功能满足要求，可以进行板级测试。

4.6 本章总结

本章首先介绍了基站的FFT处理过程，以及使用的FFT硬核的特性和设计中的要求。在此基础上，设计了基站的发送端IFFT硬件加速器和接收端的FFT硬件加速器。对于IFFT加速器，运算中使用流水线模式，并且根据不同的符号加上不同长度的CP。在运算逻辑中，使用了FIFO深度控制信号，保证了IFFT的块运算的连续性。对于接收端的FFT硬件加速器，为了提高吞吐量，同样也使用了流水线模式，不同的是需要做同步的处理。同步的操作由软件完成，之后将偏移传送到硬件，由硬件计算内部的偏移量，并且完成频移和去CP操作，在硬核做完FFT运算后，转运算后的换数据流送给软件端。最后通过ModelSim进行功能仿真，与软件的译码进行对比，验证了IFFT和FFT硬件加速器的逻辑功能的正确性

第5章 加速器的验证与测试

5.1 测试平台及测试指标

测试的平台分为软件和硬件两部分。软件部分为基于EURECOM的OpenAir 4G移动网络和设备的原型开源解决方案，主要包含LTE 终端（UE）和基站（eNodeB）两部分，软件平台运行在Xen搭建的虚拟机中。

使用的Xen虚拟机监控器VMM的版本为4.4.3，在上面安装了对应控制设备驱动的虚拟机Dom0，还有用于运行UE和eNodeB物理层应用的虚拟机DomU。虚拟机DomU的数量可以设置多个，使多用户共享加速器，充分利用硬件加速器资源。DomU中加载PCIe设备前端驱动模块，用于截取用户使用加速器的请求，并发送至后端驱动。在Dom0中加载PCIe设备的后端驱动模块，其中包括了管理前端驱动送来的请求，并调用真实的驱动模块。

硬件部分主要包括使用的主机和连接主机的加速器。主机使用的是DELL工作站，主要硬件参数如表5.1所示。加速器部分由Xilinx的Virtex-6 ML605 FPGA开发板、PCIe接口设备转接卡、实验室自主设计的基带FPGA电路板等组成。基带的电路板中包含了FPGA的芯片，与单独使用Virtex-6的FPGA开发方式基本一致。基带板连接的射频板主要用于射频的调制、收发和控制等。FPGA开发板的芯片型号是XC6VLX240T，开发板包含了PCIe接口和丰富的硬件逻辑资源。

表5.1 虚拟化平台参数表

|  |  |
| --- | --- |
| 名称 | 描述 |
| DELL工作站 | Intel(R) Xeon(R) CPU E5-1650 v3 3.5GHz处理器，16G内存 |
| Xen VMM | Xen-4.4.3-3.e16 |
| Dom0操作系统 | Centos6.7 |
| DomU操作系统 | Ubuntu 14.04.1LTS |

测试的指标主要包含两部分：加速器处理的时间开销和硬件资源的消耗。资源消耗可以根据使用的Xilinx ISE综合后的报告得出，资源使用情况在加速器设计中已经给出。时间开销可以根据ModelSim仿真和板上测试得出，并与软件运行结果比较。

5.2 测试方案设计

针对Turbo译码硬件加速器，设计了两个测试方案。一是在物理主机中，直接使用脚本连续对不同码长的数据进行多次加速器调用，分析每个码长平均的时间消耗。此方案的具体步骤为：

1. 通过LTE物理层Turbo编码模块，生成不同长度的编码数据，存储在文本文件中。
2. 编写脚本，循环调用硬件加速器，并利用时间戳记录每次译码请求完成的时间。时间包括数据的写入、加速器的运算和最后的数据读出。
3. 通过Matlab分析数据的正确性，并统计出码长平均时间开销。

第二个方案为在虚拟机中调用加速器，使用基本相同的脚本，测试译码的性能，分析虚拟机占用的开销。

FFT和IFFT的硬件加速器放置在基带板上，在基带板中进行FFT和IFFT模块加速。测试方案中，在原有的5MHz带宽下，FFT加速器能够为LTE基站完成原有物理层流程，并得到正确的解码结果。IFFT加速器能够使终端得到正确的解码结果。

5.3 板级测试与性能分析

根据Turbo译码硬件加速器的第一个测试方案，在物理主机端使用脚本，读取编码后的不同码块长度的数据，调用Turbo译码硬件加速器进行译码，将每次译码的时间存在文本中，统计每个码块平均译码时间。根据方案二，在虚拟机中，同样使用脚本，获得码块的平均译码时间。通过比较两个方案的译码时间，可以得出虚拟机的开销大小。码块从40到6144 bits共188个不同码长大小，表5.2中统计了具有代表性的多个码块长度对应的物理主机译码的时间和虚拟机译码的时间，并且得出了虚拟机的开销百分比。在译码码块较短时，虚拟机相对开销较大；当数据量较大时，即码块长度较长，虚拟机的开销比例有较明显的减少。在Turbo译码部分，虚拟机的开销一般维持在十几微秒，在整个系统中，当数据吞吐量较大时，这部分开销不明显，不会影响虚拟机中基站软件的运行。

表5.2 Turbo译码板上测试时间

|  |  |  |  |
| --- | --- | --- | --- |
| 码长（bits） | 物理主机译码时间（us） | 虚拟机译码时间（us） | 虚拟机开销 |
| 40 | 68.3 | 77.2 | 11.5% |
| 512 | 69.2 | 77.6 | 10.8% |
| 1024 | 78.9 | 83.9 | 6.3% |
| 2048 | 87.1 | 95.5 | 9.6% |
| 3008 | 103.1 | 110.1 | 6.8% |
| 4032 | 117.4 | 127.1 | 8.3% |
| 5184 | 135 | 148.2 | 9.8% |
| 6016 | 150.8 | 160.4 | 6.4% |

通过使用ModelSim仿真，可以得到在不同码长的Turbo译码加速器中输入输出和硬核译码的时间情况，在第3章的功能仿真部分做了分析。从中可以看出，在码长较短情况下，主要的时间在于译码运算，数据的输入和输出占比很少。当译码的长度较长时，编码后的数据输入时间同样占了较大一部分。在表3.3中的输入和输出时间是数据和译码器的核交互的时间。根据仿真可以看出，PCIe传输和软件程序从驱动中获取数据的时间，大概为30至40微秒。当用户在主机端调用驱动使用加速器时，还需要包括PCIe驱动传输数据的时间延迟。因此，使用流水线可以有效的降低输入输出数据产生的延迟，提高译码加速器使用的效率。

当译码的请求较多和码块较长的情况下，可以获得较高的吞吐量，同时可以弱化虚拟机开销和PCIe传输的时间延迟。从表5.2和表3.3仿真时间的对比中也可以看出，Turbo译码硬件加速器的译码时间基本维持在70至160微秒，其中译码器的核运算时间范围为30至60微秒，这说明FPGA的开发板上的译码器性能非常好。其中一个重要的原因是译码器的核使用的是250MHz的高频时钟，相对于数据传输的62.5MHz时钟，可以节省近3倍的处理时间。

基站在主机中使用软件进行Turbo译码处理的时间如表5.3所示，根据Turbo译码硬件加速器的时间，可以分析出，在普通单块码长的软件译码时间使用上，至少是加速器的两倍，在码长较长时可以达到八倍以上。此时加速器还未完全充分利用，有许多空闲时间。当多个码长的译码请求集中到来时，加速器的效率可以达到软件的十倍以上。

表5.3 软件Turbo译码时间

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 码长（bits） | 40 | 512 | 1024 | 2048 | 3008 | 4032 | 5184 | 6016 |
| 译码时间（us） | 114.9 | 299.3 | 481.2 | 686.5 | 1007.7 | 1168.8 | 1261.1 | 1340.9 |

在FPGA基带板上进行FFT和IFFT的硬件加速器测试验证。图5.1为IFFT硬件加速器的输入数据，是一整帧的数据，点数总共为76800。图5.2为IFFT硬件加速器的部分输出数据和软件运算后的数据对比。硬件的输出数据通过使用Chipscope进行采样，获得IFFT处理后的数据。图5.3为使用Matlab分析的IFFT输出数据与软件部分幅值差，从图中可以看出，误差都小于2，这是由于使用的算法对数的凑整方式造成，不影响最后运算的正确性。

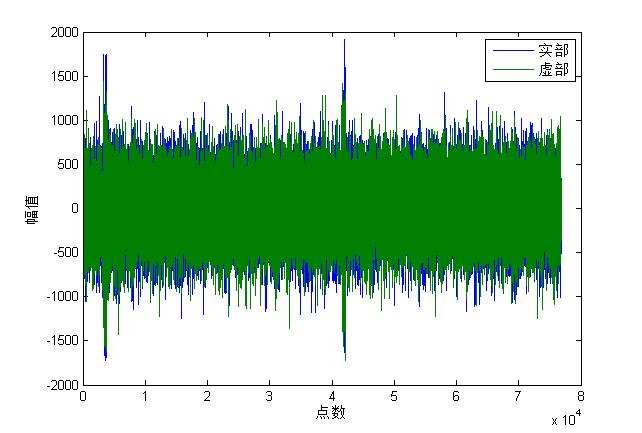


图5.1 IFFT输入数据

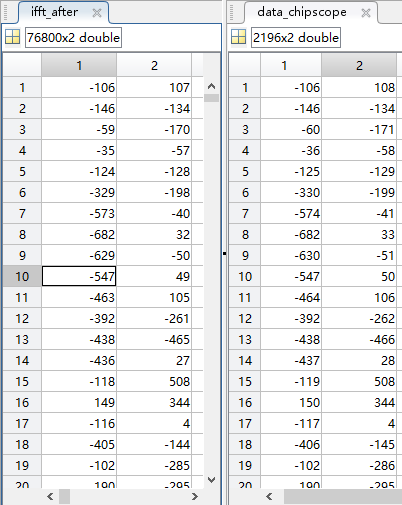


图5.2 IFFT输出数据与软件对比图

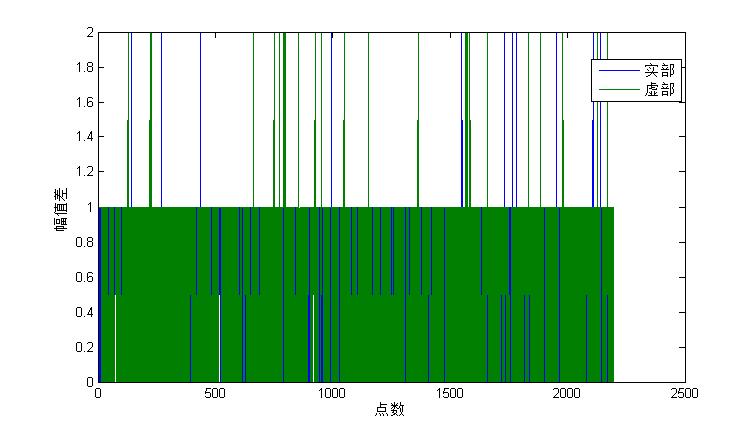


图5.3 IFFT输出数据与软件部分幅值差

对于FFT和IFFT加速器，数据的流向和PCIe驱动传输部分与不使用加速器时完全相同。所以增加的加速器模块的时间与模块仿真时间基本一致，通过仿真时间基本可以看出硬件实际使用的时间。IFFT加速器的仿真时间如表5.4所示。当使用流水线模式，在62.5MHz时钟下时，每块的处理时间可以从单块的27.3微秒缩短为11.1微秒，这个时间是指从start信号开始，一块数据从输入到这个块做完运算并输出的时间。这是由于流水线模式下，输入、运算和输出分别可以并发执行，并可以充分利用硬件中的逻辑资源，最大程度的节省时间，使加速器一直处于工作状态。当处理时钟为250MHz时，一个512点块的总时间为2.6us。相比62.5MHz时钟，时间可缩短近3倍。

表5.4 IFFT加速器仿真时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 运算模式 | 输入时间（us） | 运算时间（us） | 输出时间（us） | 总时间（us） |
| 分块独立 | 8.2 | 14.3 | 8.8 | 27.3 |
| 流水线  62.5MHz | 8.2 | 连续工作 | 8.8 | 11.1 |
| 流水线  250MHz | 2.1 | 连续工作 | 2.2 | 2.6 |

LTE的IFFT处理中，对于5MHz的带宽，一个子帧点数为512\*14+512，共7680个点，其中后面的512为14个OFDM符号所加的循环前缀CP。在62.5MHz下，一个子帧使用硬件加速器所使用的时间为155.4us。在250MHz下，一个子帧使用硬件加速器所使用的时间为38.1us。在使用62.5MHz的时钟情况下，LTE的5MHz带宽的系统已经可以正常获得解码结果，在使用250MHz时钟的加速器时，通过Chipscope可以看出加速器不会一直处于busy工作状态。因此，设计的250MHz时钟可以支持更高带宽的LTE系统。

对于FFT加速器，和IFFT使用的是同一个IP核，设计的区别主要是，初始阶段需要从软件端拿到同步的偏移量，之后每个点块运算前需要做频移和去CP操作。同步一般只需要一次，所以时间虽然较长且主要来自于软件端，但在连续运算的时候可以不用考虑。对于频移和去CP的操作，利用流水线模式和控制写到CP FIFO的写使能信号，每个点块可以在40个左右的时钟周期内完成，其中频移操作也是连续出结果，只有4个周期的延迟，所以这部分时延很短。FFT加速器的流水线模式的块处理时间与IFFT基本相同，对于62.5MHz时钟，输入和输出时间都为8.8us，一个点块的总时间约为9us，一个子帧的总时间约为126us。使用250MHz时钟，一个子帧的总时间约为36.4us。FFT和IFFT的加速器，最后都能够成功得到正确的解码后的数据。

5.4 本章总结

本章主要完成硬件加速器的测试工作。首先介绍了测试的平台及测试指标。接着设计了各个加速器的测试方案，根据方案对加速器进行板级测试和性能分析。对于Turbo译码硬件加速器，使用了两个方案分别对物理主机和虚拟机中的性能进行了测试，并与仿真的结果一起进行了分析。对于FFT和IFFT的加速器，通过板级验证了加速器功能的正确性。

第6章 总结与展望

6.1 总结

本文根据集中式基站资源使用场景的要求，研究了C-RAN的虚拟化平台，并设计了支持虚拟化的Turbo译码硬件加速器和高速FFT/IFFT加速器，并使用FPGA开发板进行实现。主要完成了如下的工作：

1. 研究和分析了C-RAN中LTE物理层的处理流程，以及使用Xen虚拟化技术搭建的C-RAN虚拟化平台。并针对FPGA的PCIe接口驱动，设计了虚拟化的前后端驱动的软件和硬件接口等。并介绍了FPGA开发的流程，给出了开发的具体方法。
2. 对Turbo码的原理以及硬核的功能和特性进行了介绍。针对加速器的需要，设计了高时钟频率的硬核工作环境，并采用流水线模式，大大地提高了译码器的吞吐量。对包含硬核的各层进行了详细的设计，并获得Turbo译码硬件加速器的功能仿真和资源使用情况。
3. 根据LTE物理层的处理流程，在已有的FPGA基带板上添加了FFT和IFFT的硬件加速器。IFFT加速器实现了CP的自动添加，并且实现了流水线的模式。FFT硬件加速器实现了频移和去CP的操作，同时实现了软件同步硬件执行和数据流切换的模式。
4. 通过测试平台对Turbo译码硬件加速器和FFT/IFFT硬件加速器进行了测试，通过测试指标，分析并得出了所设计的硬件加速器的性能。设计的加速器都支持250MHz高时钟工作频率，满足基站处理的延迟要求，支持高带宽的云基站池处理任务。

6.2 展望

C-RAN的发展，对于未来的无线接入网具有很重要的意义。在C-RAN中设计使用硬件加速器，可以在很大程度上提高基站资源的利用率和降低能源消耗，非常符合C-RAN的目标。本文在对硬件加速器的设计中，学习到了FPGA开发的相关知识，以及LTE物理层的相关处理。但还存在以下需要进一步研究和改进的地方：

1. 在搭建的C-RAN平台中，延迟是非常敏感的一个因素。系统的首要目标就是低延迟，因此在设计中，需要考虑虚拟化的开销和虚拟机的性能。本文使用的是Xen虚拟化方式，虚拟机等的开销较低，是较合适的方式之一。但在目前的虚拟化领域中，新出现了一些新技术，例如Docker容器等，在应用上这类轻量型的虚拟化会更加快速和便捷。在未来的研究中，应该多关注虚拟化方式带来的影响。
2. 本文使用的硬件为FPGA Virtex-6，属于性能较高的开发板。相较于其他的加速器硬件设备，FPGA的编程能力十分强大，但是由于价格比较昂贵，比ASIC功耗高等缺点，在后期加速器比较成熟的阶段，可以使用像ASIC这样的硬件作为加速器，便于大规模产品化和高效利用。
3. 在数据传输中，使用了PCIe 2.0X4的接口。目前的PCIe接口已经拥有了4.0版本，带宽是2.0版本的4倍，并且5.0版本也在研究阶段，带宽更是接近2.0版本的8倍。所以，在未来的集中式基站池的发展中，数据的传输可以达到非常可观的带宽。在与加速器硬件配合使用中，可以大大提高数据的吞吐量。
4. 本文Turbo译码器核心的选择，是使用了Xilinx公司的IP核。该IP核的优点很明显，作为成熟的产品，拥有可靠的功能和超强的性能，但是价格也十分昂贵。目前已有很多的自己设计的Turbo译码器的实现，特别是基于FPGA开发板的实现。在实际的生产中，可以根据选择的硬件特性和云平台上的要求，自行开发更符合要求的Turbo译码器，实现资源的充分利用。

**致谢**

时光飞逝，研究生的学习阶段已进入尾声。本论文是对我的研究生三年的学习研究的总结，在这里衷心感谢所有老师、同学、朋友和亲人对我的指导和关心。

首先，非常感谢我的导师吴俊教授。从本人即将进入同济大学读研究生时候，导师的关怀和对工作的热忱都深深地影响着我。在通信与智能计算的实验室中，我度过了美好的三年研究生生活。实验室提供了丰富的硬件设施，为我日常的研究和实验提供了良好的环境。在工作中，导师也与我并肩工作学习，他循循善诱的教导和敏锐的思考总是能令我耳目一新。本文中的实验和论文写作的选题都凝结了导师的心血和汗水。吴俊教授精益求精的治学态度、严谨的工作作风和朴实的生活态度将使我受益终身。

同时，我还要感谢C-RAN项目组的其他成员对我的帮助和照顾。感谢朱慧与我一起研究C-RAN项目的设计，感谢郭栋学长在自己繁忙的工作中参加我们的项目和会议，给出很多指导意见，感谢石丰略同学在前期对项目的帮助，你的成果是我研究的基石。感谢刘典学长和丁壮在项目中遇到问题时的及时帮助，感谢张志峰老师和任浩琪老师对我开发硬件的悉心指导。同时还要感谢实验室其他所有的同学们以及实验室的本科学弟们的付出。

最后，我要感谢我的父母和亲人们，无论在什么情况下，他们都给我无微不至的关怀，支持和爱护，有了他们对我的关爱才能让我顺利毕业。

2017年3月

**参考文献**

1. Xiaoyun W, Yuhong H, Chunfeng C, et al. C-RAN: Evolution toward green radio access network[J]. China Communications, 2010, 7(3): 107-112.
2. Chen K, Duan R. C-RAN: The Road Towards Green RAN, White Paper Version 2.5, China Mobile Research Institute, Oct. 2011[J].
3. Wu J, Zhang Z, Hong Y, et al. Cloud radio access network (C-RAN): a primer[J]. IEEE Network, 2015, 29(1): 35-41.
4. 任磊, 张霖, 张雅彬, 等. 云制造资源虚拟化研究 [J][J]. 计算机集成制造系统, 2011, 17(3): 511-518.
5. Abels T, Dhawan P, Chandrasekaran B. An overview of xen virtualization[J]. Dell Power Solutions, 2005, 8: 109-111.
6. Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends[J]. Computer, 2005, 38(5): 39-47.
7. Bugnion E, Devine S, Rosenblum M, et al. Bringing virtualization to the x86 architecture with the original vmware workstation[J]. ACM Transactions on Computer Systems (TOCS), 2012, 30(4): 12.
8. Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]. Proceedings of the Linux symposium. 2007, 1: 225-230.
9. Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[C]. ACM SIGOPS operating systems review. ACM, 2003, 37(5): 164-177.
10. Watson J. Virtualbox: bits and bytes masquerading as machines[J]. Linux Journal, 2008, 2008(166): 1.
11. Owens J D, Houston M, Luebke D, et al. GPU computing[J]. Proceedings of the IEEE, 2008, 96(5): 879-899.
12. Han T D, Abdelrahman T S. hiCUDA: High-level GPGPU programming[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(1): 78-90.
13. Keutzer K, Malik S, Newton A R. From ASIC to ASIP: The next design discontinuity[C]. Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on. IEEE, 2002: 84-90.
14. Strauss W. Digital signal processing[J]. IEEE Signal Processing Magazine, 2000, 17(2): 52-56.
15. Wu X, Gopalan P, Lara G. Xilinx next generation 28 nm FPGA technology overview[J]. White Paper: 28nm Technology, July, 2010, 23: 5.
16. Thomas D, Moorby P. The Verilog® Hardware Description Language[M]. Springer Science & Business Media, 2008.
17. Navabi Z. VHDL: Analysis and modeling of digital systems[M]. McGraw-Hill, Inc., 1997.
18. Spear C. SystemVerilog for verification: a guide to learning the testbench language features[M]. Springer Science & Business Media, 2008.
19. Wang W, Bolic M, Parri J. pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment[C]. Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on. IEEE, 2013: 1-9.
20. Fowers J, Ovtcharov K, Strauss K, et al. A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication[C]. Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on. IEEE, 2014: 36-43.
21. Chen, F., Shan, Y., Zhang, Y., Wang, Y., Franke, H., Chang, X., & Wang, K. (2014, May). Enabling FPGAs in the cloud. In Proceedings of the 11th ACM Conference on Computing Frontiers (p. 3). ACM.
22. Cheung O Y H, Leong P H W. Implementation of an FPGA based accelerator for virtual private networks[C]//Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on. IEEE, 2002: 34-41.
23. So H K H, Brodersen R. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH[J]. ACM Transactions on Embedded Computing Systems (TECS), 2008, 7(2): 14.
24. E. El-Araby, I. Gonzalez, T. El-Ghazawi. Virtualizing and sharing reconfigurable resources in High-Performance Reconfigurable Computing systems. Second International Workshop on High-Performance Reconfigurable Computing Technology and Applications, pp. 1-8, 2008.
25. I. Gonzalez, S. Lopez-Buedo, G. Sutter, D. Sanchez-Roman, F. J. Gomez-Arribas, J. Aracil. Virtualization of reconfigurable coprocessors in HPRC systems with multicore architecture. Journal of Systems Architecture, vol. 58, no. 6, pp. 247-256, March 2012.
26. C. H. Huang, P. A. Hsiung, J. S. Shen. Model-based platform-specific co-design methodology for dynamically partially reconfigurable systems with hardware virtualization and preemption. Journal of Systems Architecture, vol. 56, no. 11, pp. 545-560, August 2010.
27. E. Lübbers. Multithreaded Programming and Execution Models for Reconfigurable Hardware. PhD thesis, Computer Science Department, University of Paderborn, 2010.
28. M. Sabeghi and K. Bertels. Toward a runtime system for reconfigurable computers: A virtualization approach. Design, Automation & Test in Europe Conference & Exhibition, pp. 1576-1579, April 2009.
29. G. Giunta, R. Montella, G. Agrillo and G. Coviello. A GPGPU Transparent Virtualization Component for High Performance Computing Clouds. In Proc. of Euro-Par, pp. 379-391, Heidelberg, 2010.
30. S. Che, J. Li, J. Lach, and K. Skadron. Accelerating compute intensive applications with GPUs and FPGAs. In Proc. of the 6th IEEE Symposium on Application Specific Processors, pp. 101-107, June 2008.
31. Wu M, Sun Y, Cavallaro J R. Implementation of a 3GPP LTE turbo decoder accelerator on GPU[C]. Signal Processing Systems (SIPS), 2010 IEEE Workshop on. IEEE, 2010: 192-197.
32. 3GPP TR 36.211 V11.1.0. Multiplexing and channel coding [s]. Technical report, 2013
33. 3GPP TR 36.211 V11.1.0. Physical channels and modulations[s]. Technical report, 2013.
34. Calazans N, Moreno E, Hessel F, et al. From VHDL register transfer level to SystemC transaction level modeling: a comparative case study[C]. Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on. IEEE, 2003: 355-360.
35. Robertson P, Villebrun E, Hoeher P. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain[C]. Communications, 1995. ICC'95 Seattle,'Gateway to Globalization', 1995 IEEE International Conference on. IEEE, 1995, 2: 1009-1013.
36. Hagenauer J, Hoeher P. A Viterbi algorithm with soft-decision outputs and its applications[C]. Global Telecommunications Conference and Exhibition Communications Technology for the 1990s and Beyond'(GLOBECOM), 1989. IEEE. IEEE, 1989: 1680-1686.
37. LogiCORE IP 3GPP LTE Turbo Decoder v2.0 [DB/OL] (2011-05). https://china.xilinx.com/member/lte\_turbo\_dec\_eval/tcc\_decoder\_3gpplte\_ds675.pdf
38. Frigo M, Johnson S G. FFTW: An adaptive software architecture for the FFT[C]. Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on. IEEE, 1998, 3: 1381-1384.
39. LogiCORE IP LTE Fast Fourier Transform v1.0 Product Brief [DB/OL] (2011-03). https://www.xilinx.com/support/documentation/ip\_documentation/lte\_fft\_xmp125.pdf

**附录**

**个人简历、在读期间发表的学术论文与研究成果**

**个人简历：**

汤绍先，男，1992年9月生。

2014年6月毕业于上海大学 计算机科学与技术专业 获学士学位。

2014年9月入同济大学读硕士研究生 计算机科学与技术专业。

**已发表论文：**

1. FPGA-based Turbo Decoder Hardware Accelerator in Cloud Radio Access Network(C-RAN). China Communication and Network, chinacom.

**发明专利：**

1. 吴俊，汤绍先，朱慧，郭栋，石丰略，苏立峰. 一种基于Xen的FPGA加速器虚拟化平台及应用. 2015